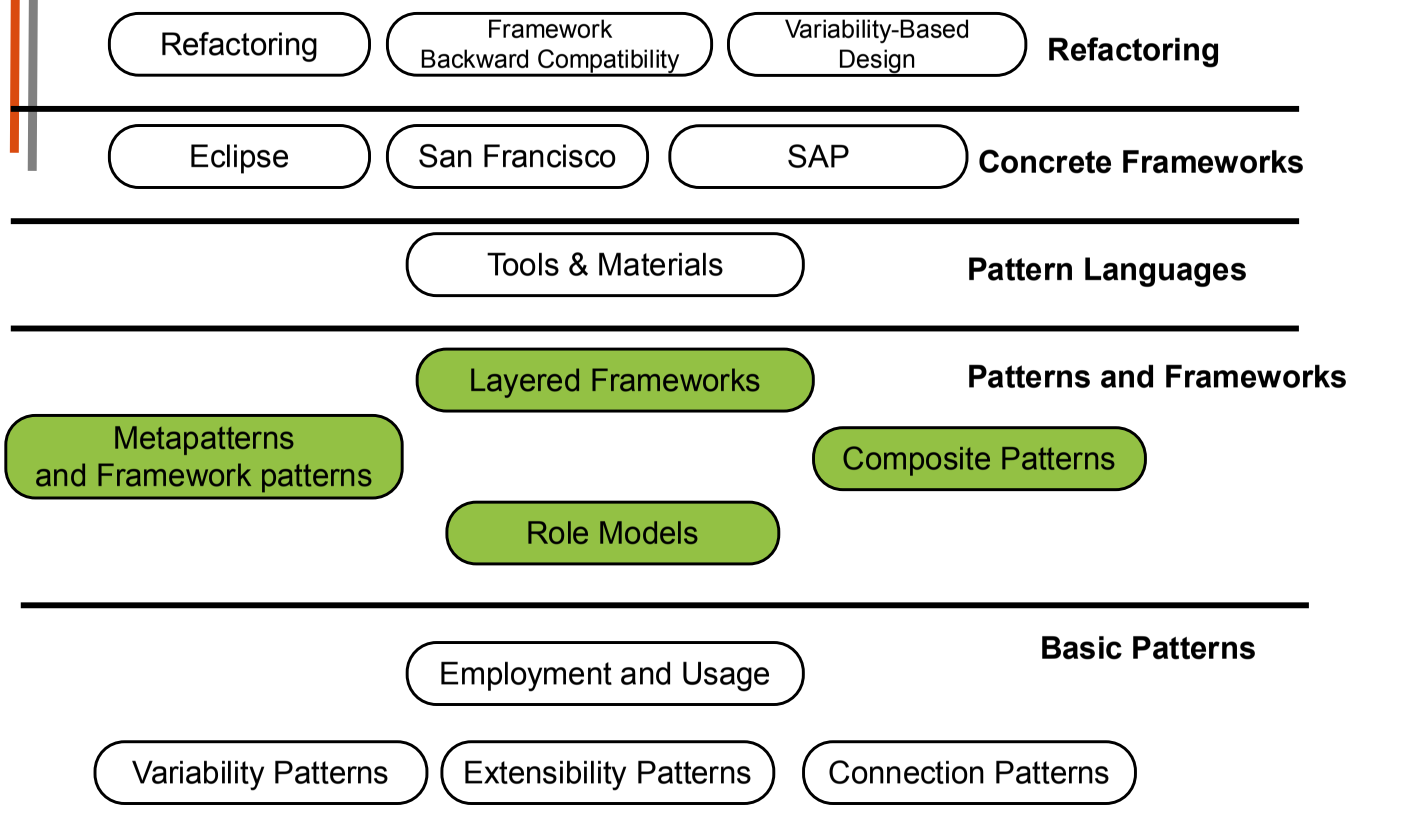
Part II
Design Patterns and Frameworks

Prof. Dr. U. Aßmann
Chair for Software Engineering
Faculty of Informatics
Dresden University of Technology
11-0.1, 11/12/11

10) Role-based Design
11) Framework Variability
12) Framework Extensibility

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

1) Role-based Design
2) Role-Model Composition
3) Role Mapping in the MDA
4) Implementing Abilities
5) Design Patterns as Role Models
6) Composition of Design Patterns with Role Models
7) More on Roles
8) Effects of Role Modeling in Frameworks

Literature (To Be Read)

- D. Riehle, T. Gross. Role Model Based Framework Design and Integration. Proc. 1998 Conf. On Object-oriented Programing 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, MDAFA 2003 and MDAFA 2004, Lecture Notes in Computer Science, Volume 3599, 2005, DOI: 10.1007/11538097
  - http://www.springerlink.com/content/f8u0vmbbt2mf/#section=590861

Other Literature

- 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.

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

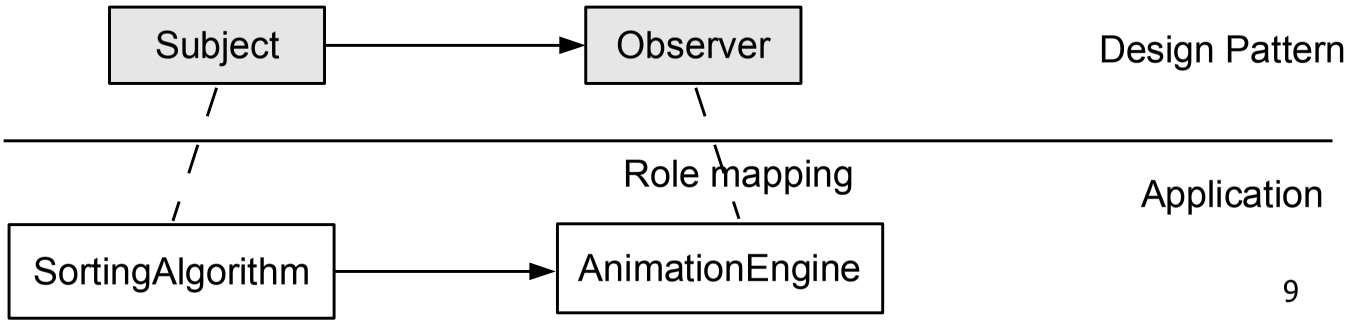
Goal

- Understand the difference between roles and objects, role types and classes
- Understand 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
- Understand how to optimize layered frameworks and design patterns

10.1 Role-based Design With Role Models

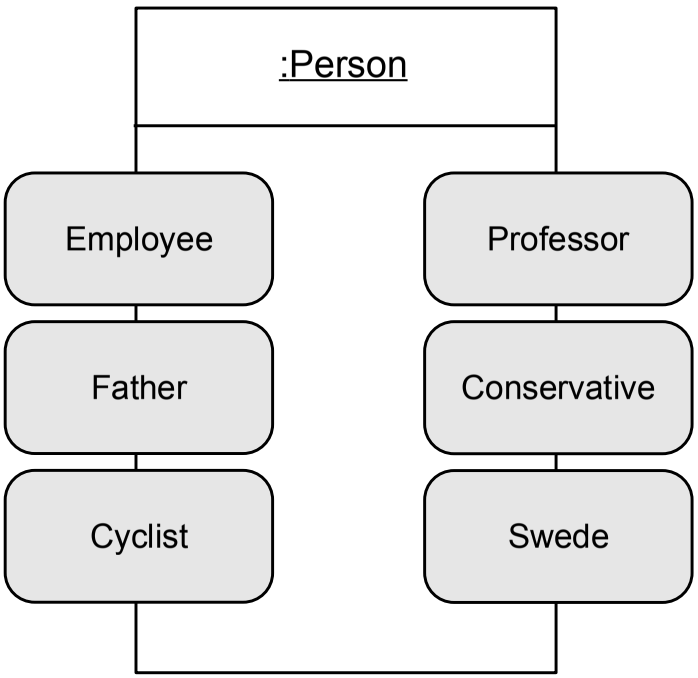
Purpose of Teaching Role-based Design

- 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

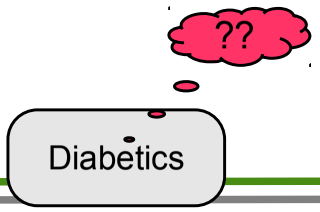


What are Roles?

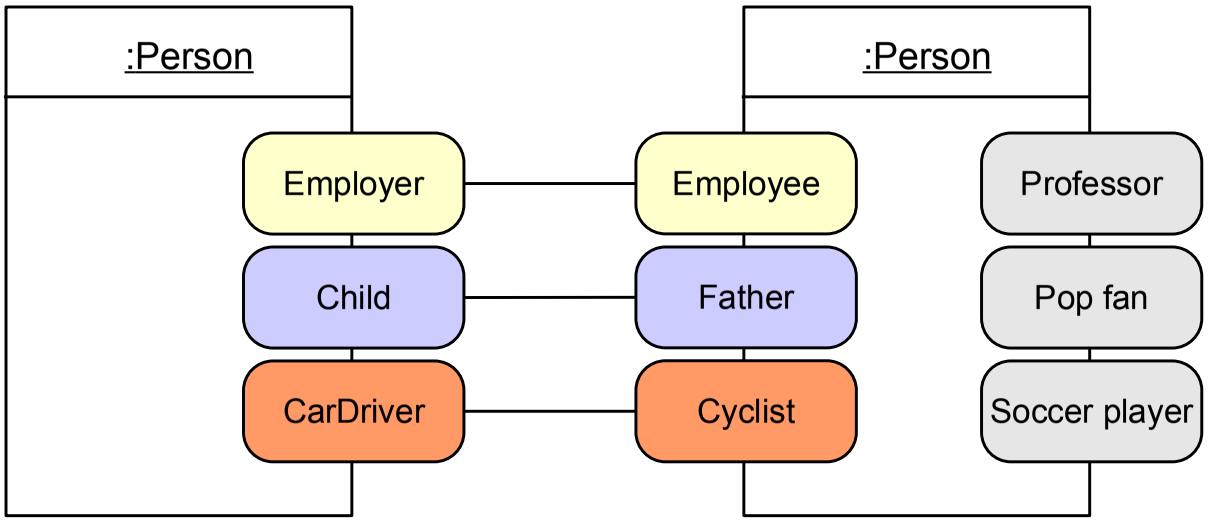
- A *role* is a *dynamic view* onto an object
  - The view can change dynamically
  - A role of an object belongs to a area of concern
- Roles are *played* by the objects (the object is the *player* of the role)
  - Playing a role means entering a state
  - Active roles correspond to states of an object



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)

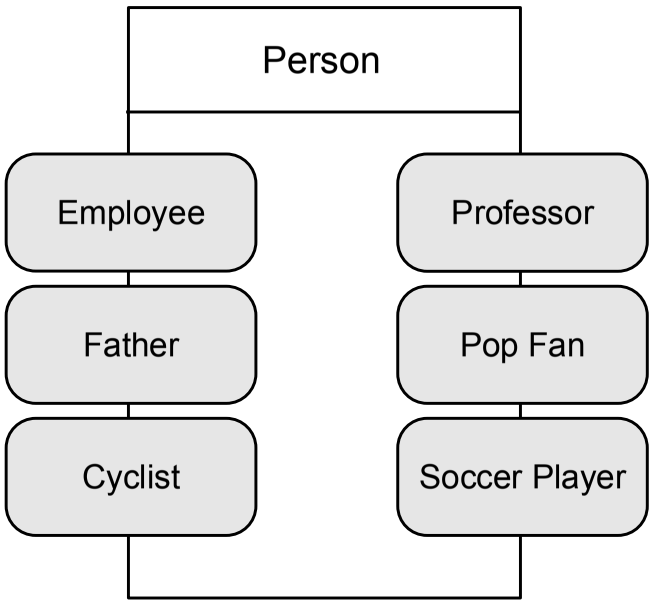


What are Role Types?

- 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 an 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)

- 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

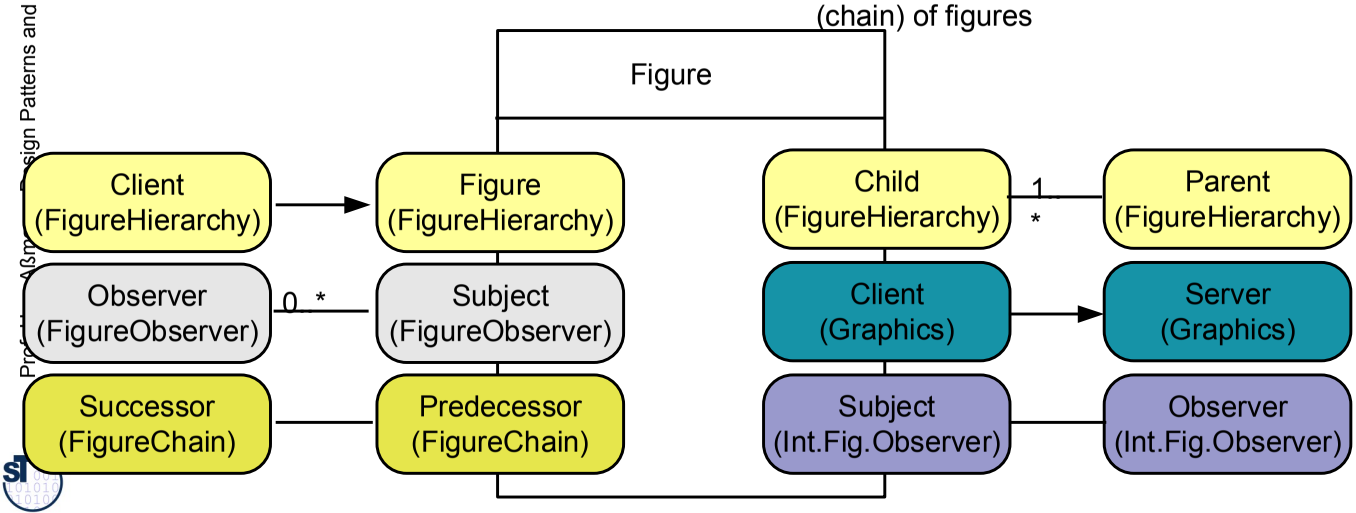


A Class-Ability Model For Figures in a Figure Editor

- 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

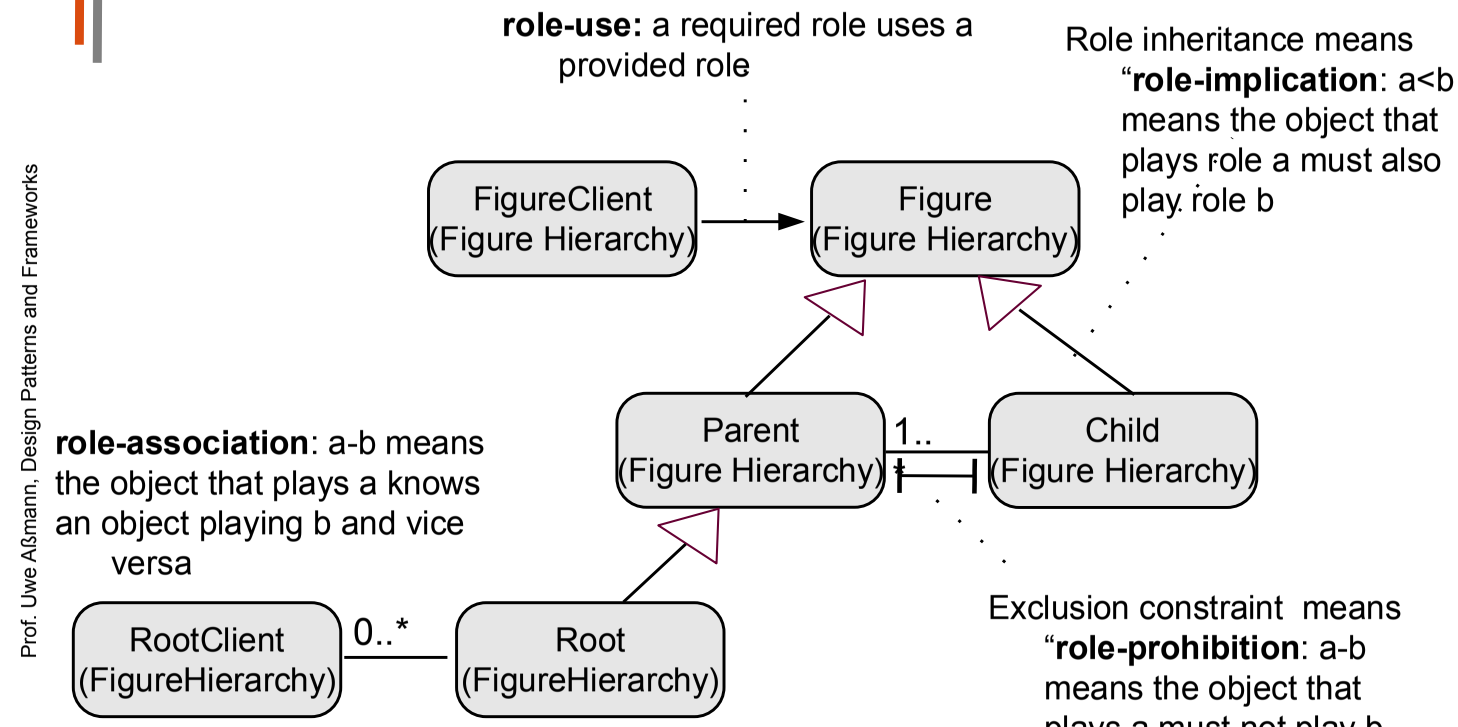
Explanation of some role types:

- 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

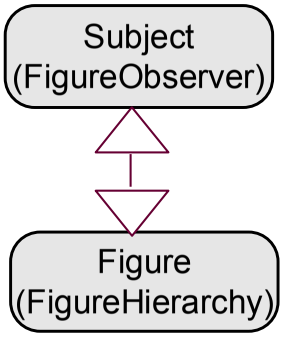
- Arrows denote constraints between roles (role constraints)

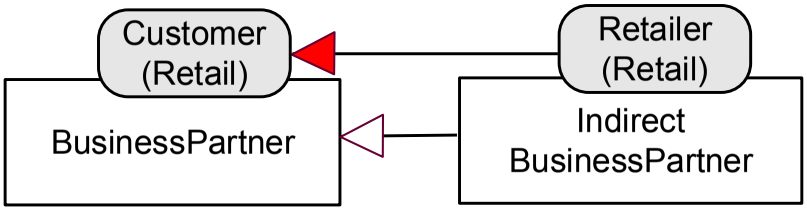


More Constraints

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

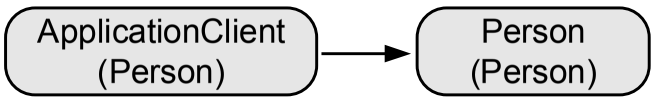
- Ask the central question:
  - Which role does my object play in this context?
  - Which responsibility does my object have in this context?
  - Which state is my object in in this context?
- If you develop with CRC cards, the questions lead to a grouping of the responsibilities (i.e., roles) on the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

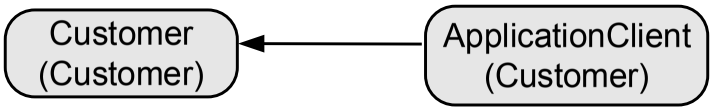
Role-Based Design with Role Models

- 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 Design
  - Roles split a class into smaller pieces
  - Roles emphasize *collaborations* in design, i.e., 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?

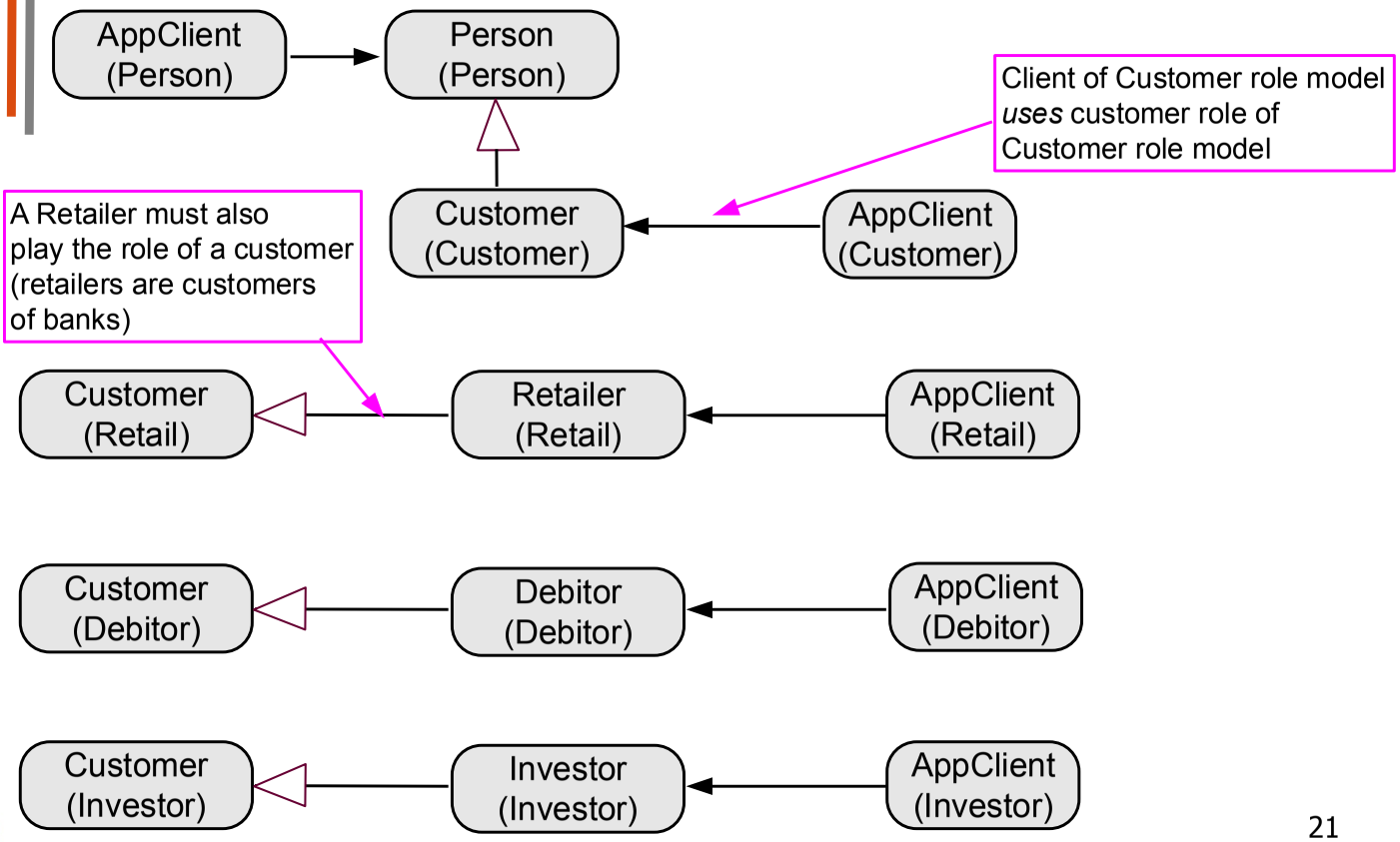
10.2 Composition of Role Models

Role Models of Persons in Business Applications



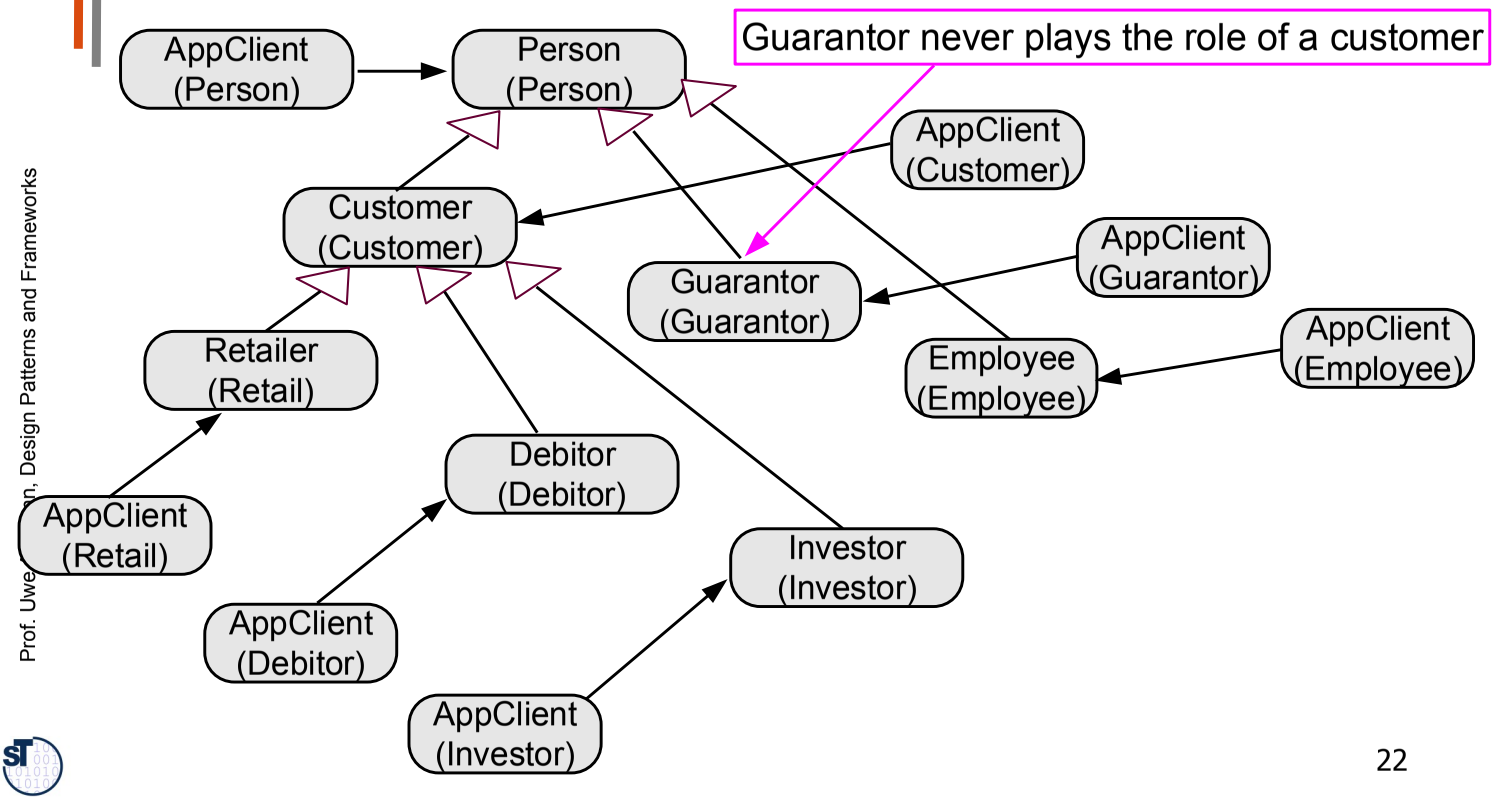


Role Models of Persons in Business Applications



Merging Role Models of Persons in Business Applications

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

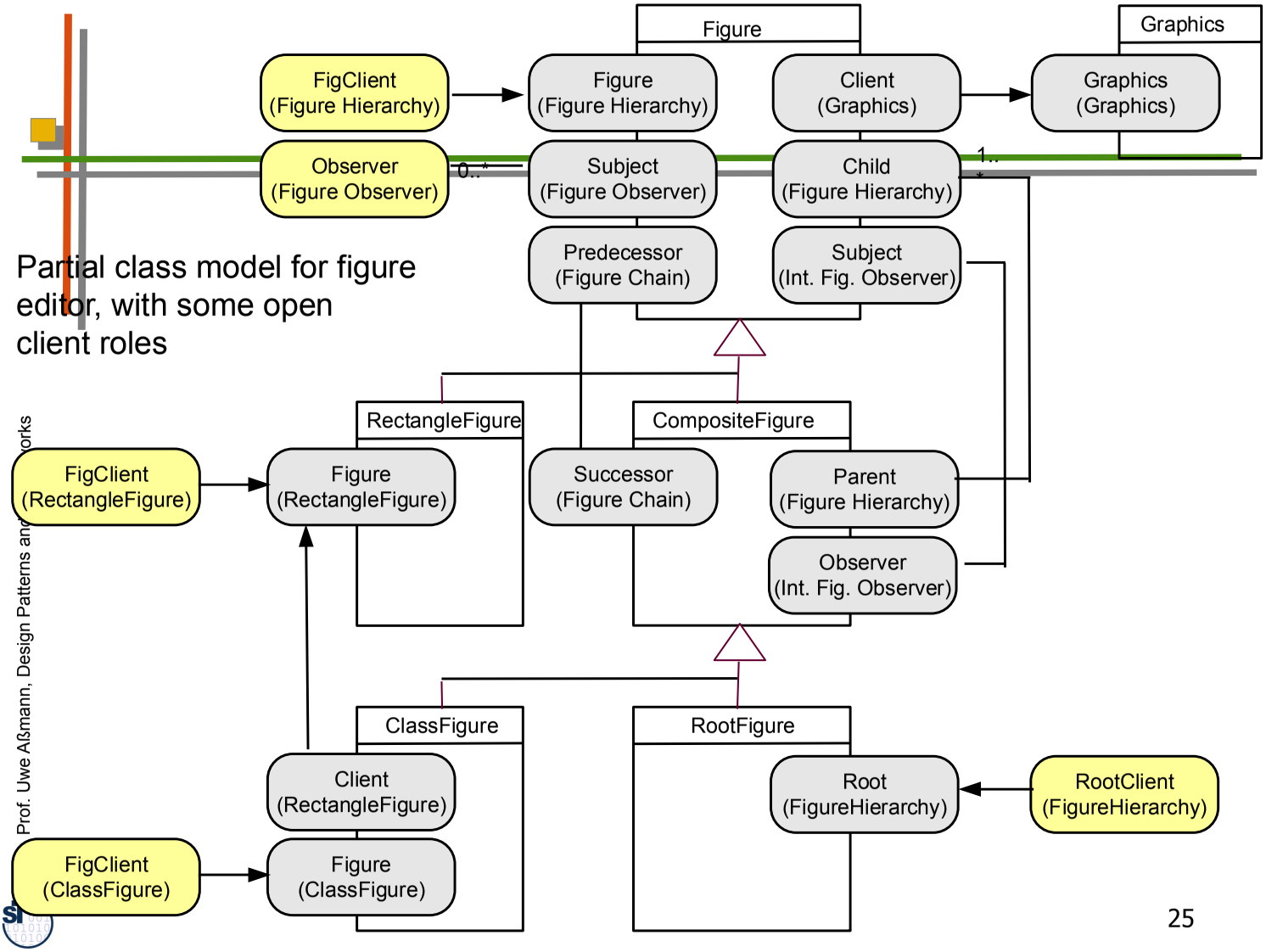


Merging Role Models into Class Diagrams

How role models are merged to class models

Composing Role Models To Partial Class Diagrams

- Classes combine roles
  - 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 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

- FigureHierarchy: composite figures (with root figure and other types, such as rectangluar 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

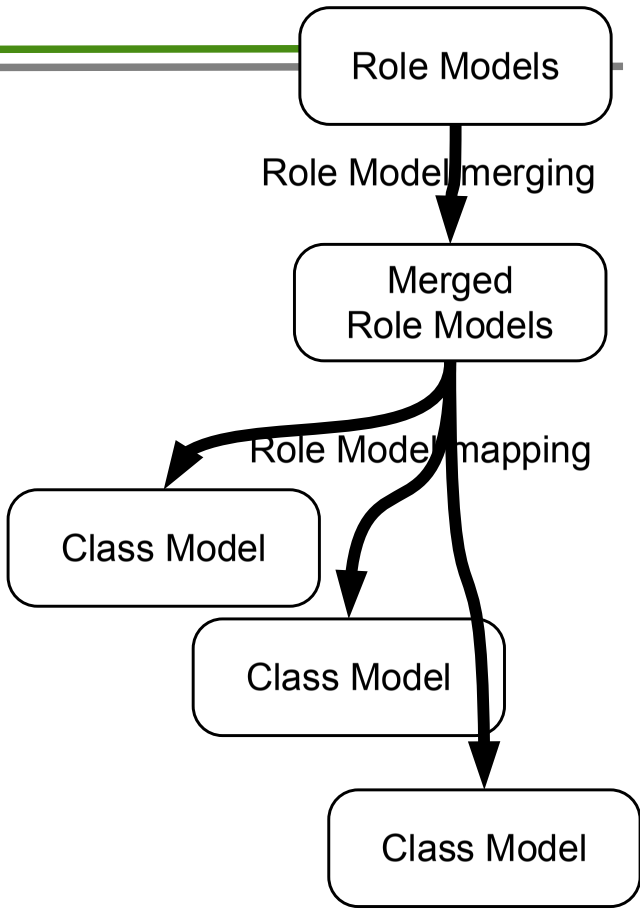
10.3 Role Mapping in the MDA

Merging role models to class models can be seen as a step of MDA

[Zhao]

Steps In Role-Based Design

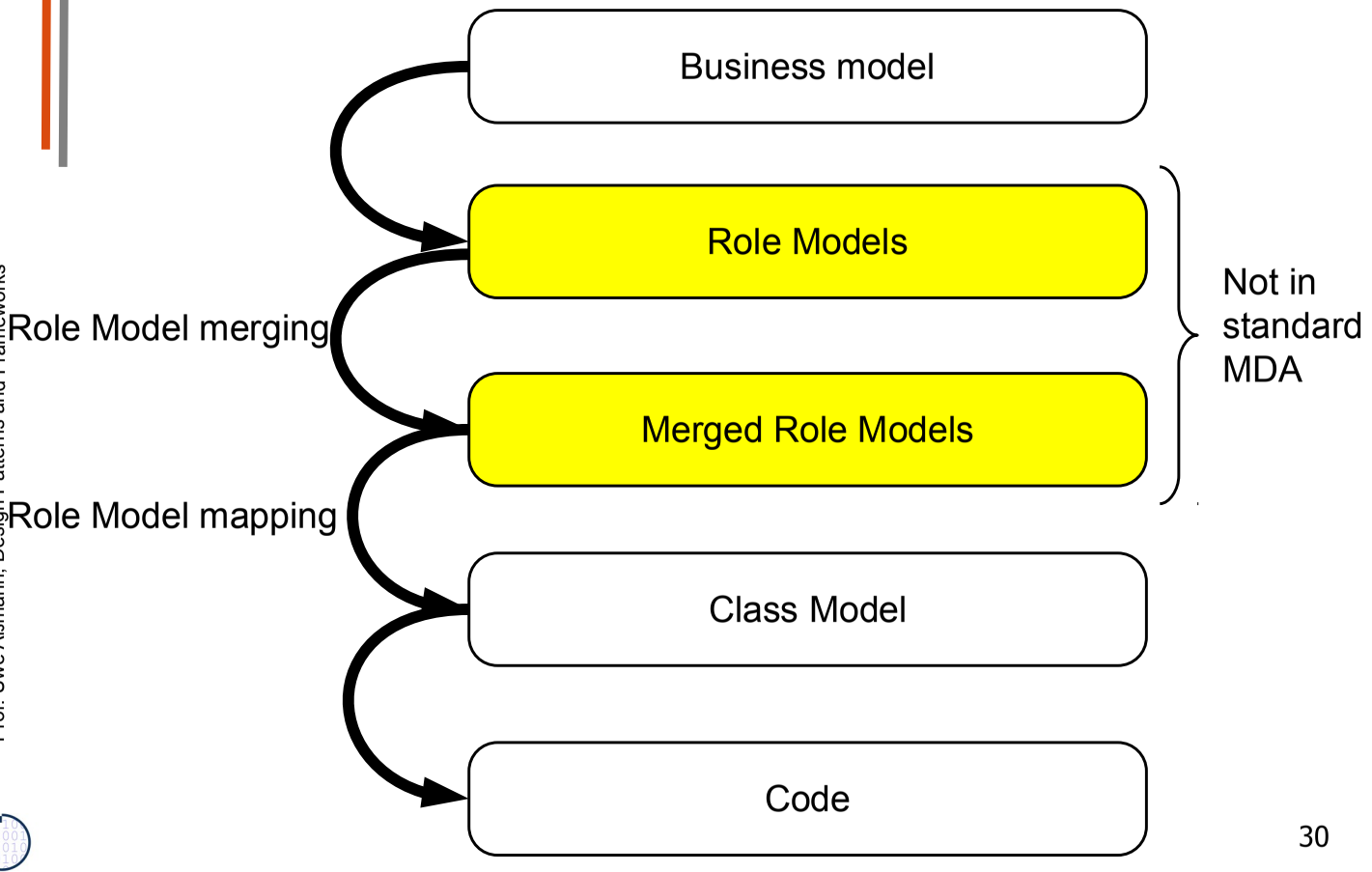
- First, do role models
  - Roles are all kept distinct
  - Find out about role constraints that constraint which objects execute which roles
- Secondly, compose (merge) them
  - And set up new constraints between roles of different models
- Thirdly, map role models to class diagram
  - By merging the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



The Role Mapping Process and Model-Driven Architecture

- 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*, 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



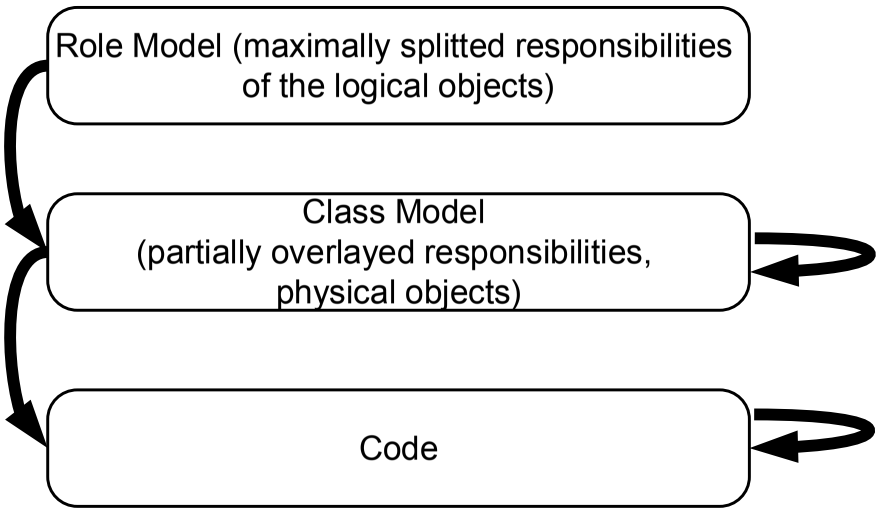
The Influence of the Role Constraints on Role Model Mapping

- *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 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)

Computing Physical Objects

- 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



10.4 Implementing Abilites By Hand

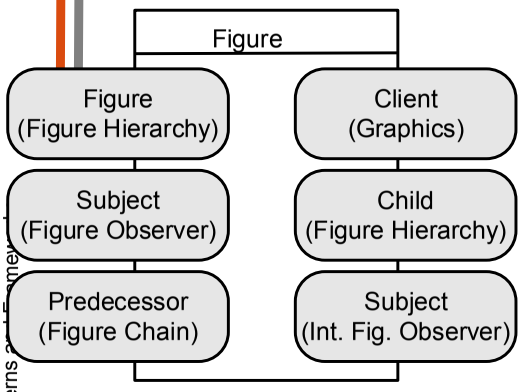
Implementation of Abilities

Abilities can be merged into classes in several ways:

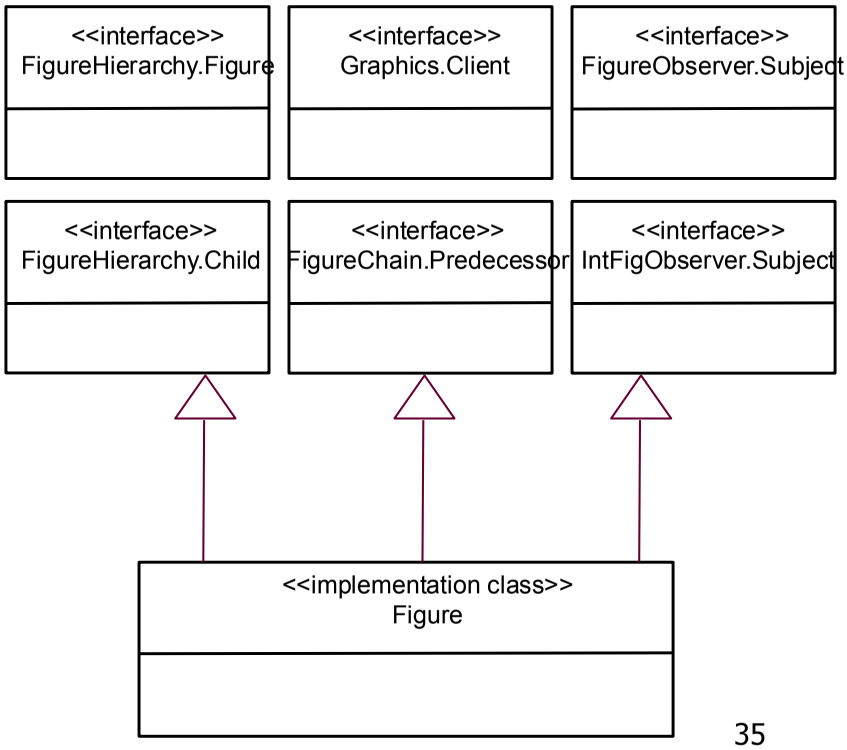
- With interfaces
  - Then, code for the interfaces must be written by hand
- With multiple inheritance
  - Then, there are two layers of classes: role classes and standard classes
- With mixin classes
  - Some language allow for composing “mixin” classes into classes
    - CLOS, Scala
    - “include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

With Interfaces

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

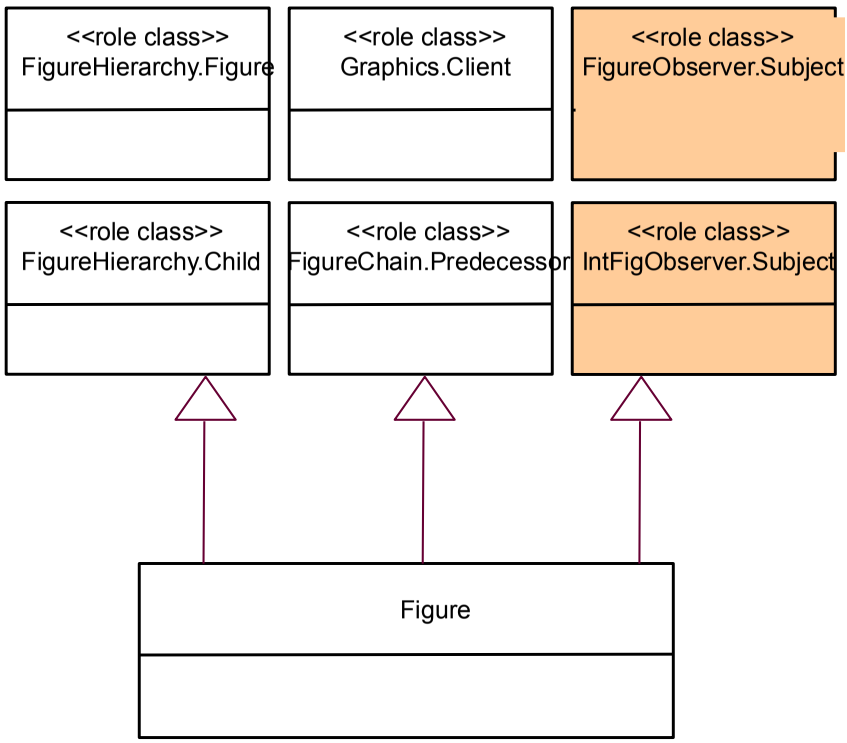



```
public class Figure implements
    FigureHierarchy.Figure,
    FigureHieachy.Child,
    Graphics.Client,
    IntFigObserver.Subject,
    FigureObserver.Subject,
    FigureChain.Predecessor
{   ... implementations of
    role-interfaces ...
}
```



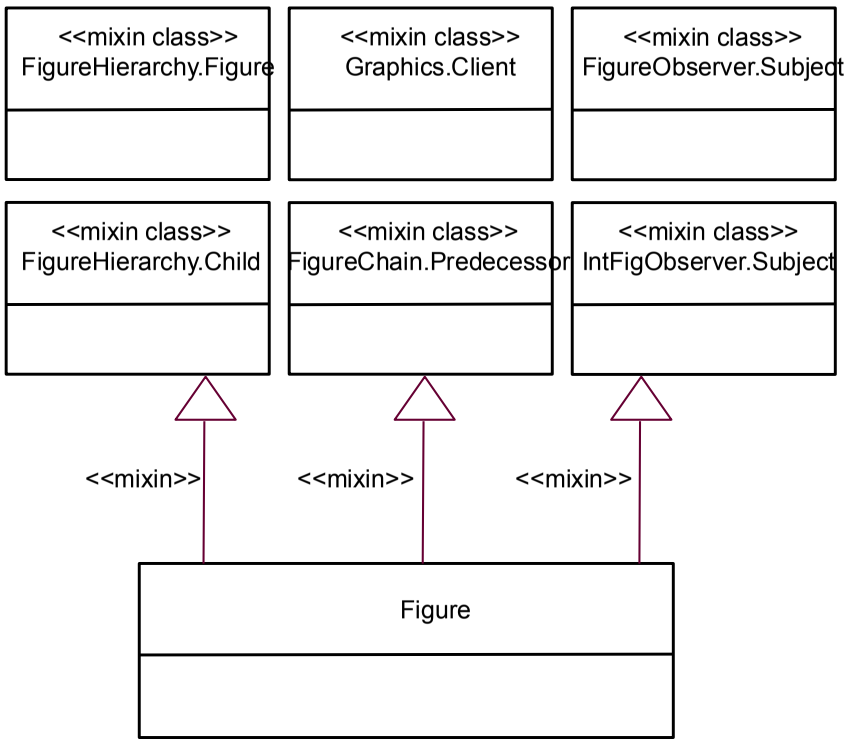
Embedding With Multiple Inheritance

- 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



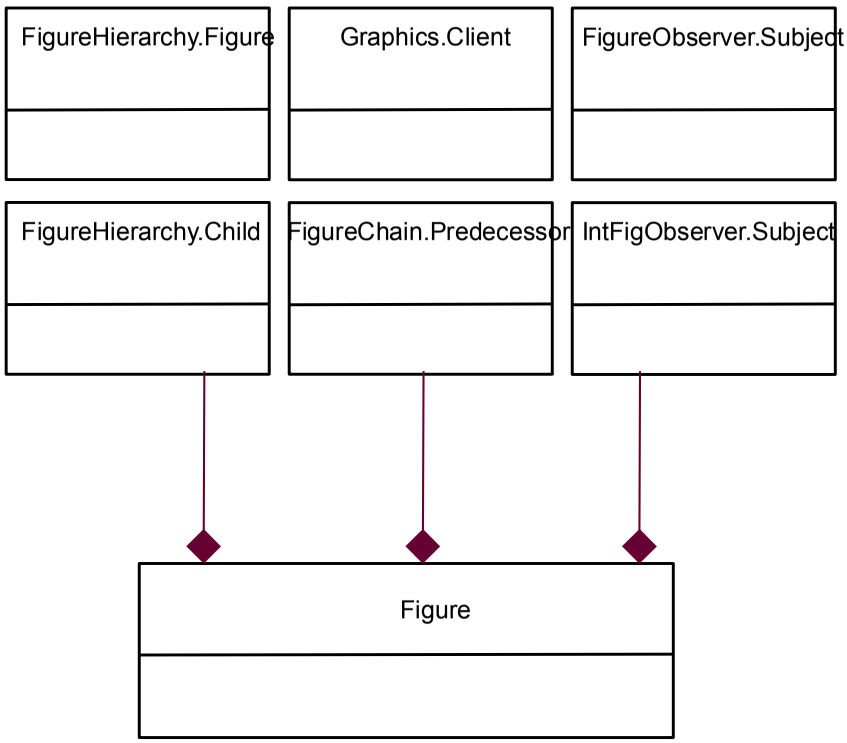
Embedding With Mixin Classes

- 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



Implementation With Multi-Bridges and Role Objects

- A role object represents only one role
- A role class only one role type
- There is a core object that aggregates all role objects
- Also with “Role Object” pattern (later)

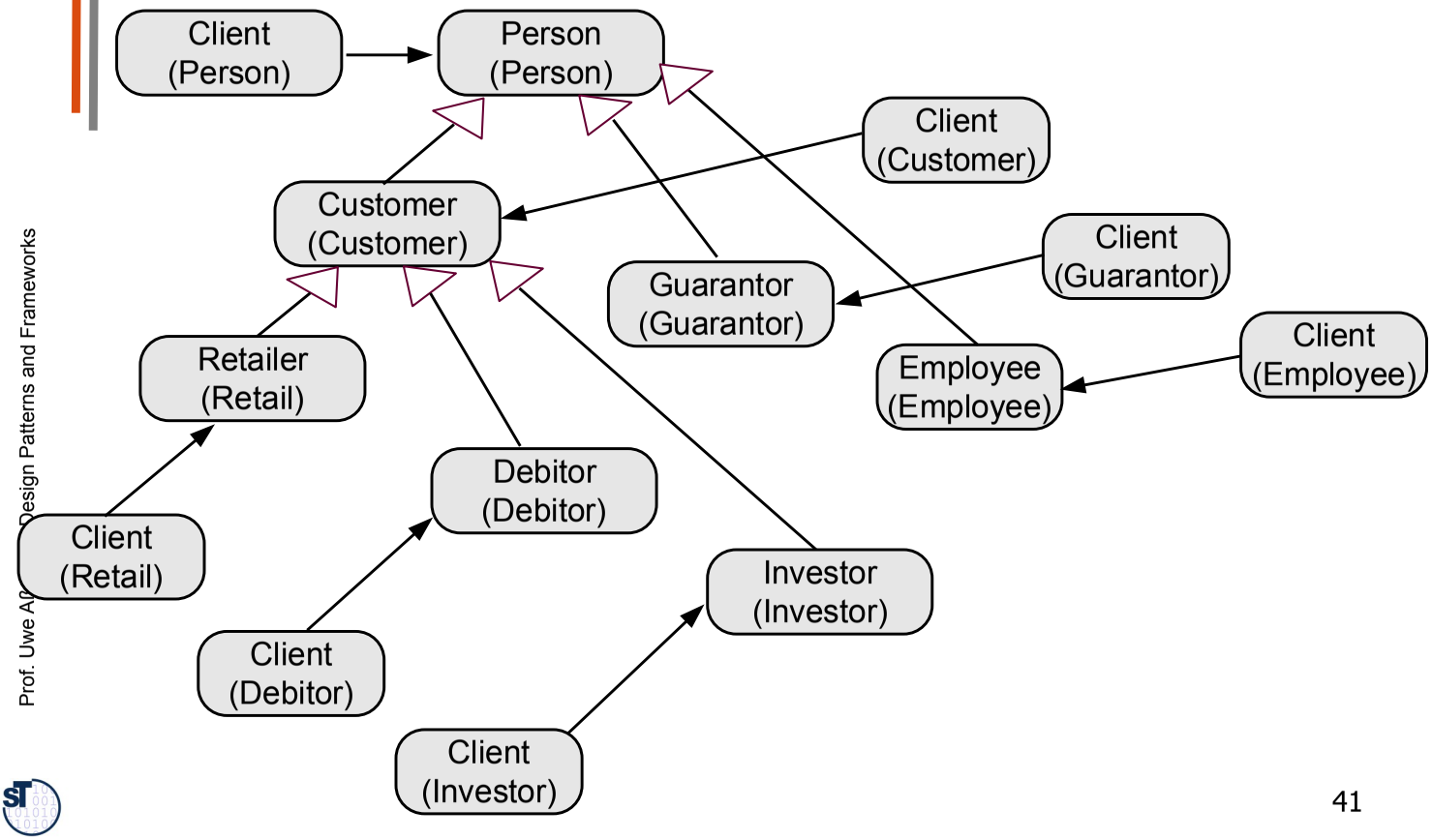


The Difference of Roles and Facets

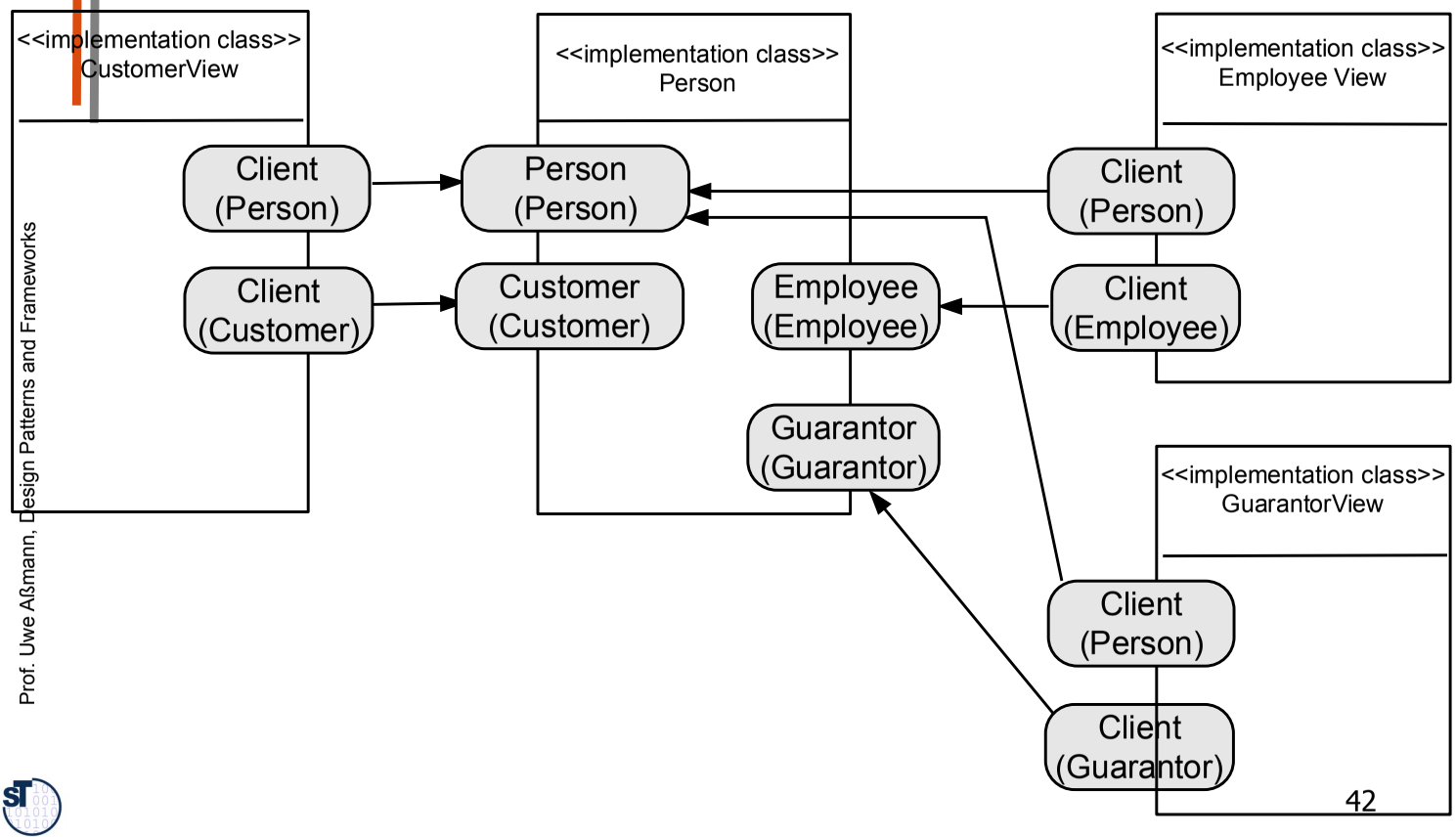
- A faceted class is a class with n *dimensions*
  - If the facet has a collaboration partner, it turns out to be a role
    - Each facet is a role type
    - Role types are independent of each other
    - However, the role type is *static,* not dynamic: facets are lasting

Example of Persons in Business Applications

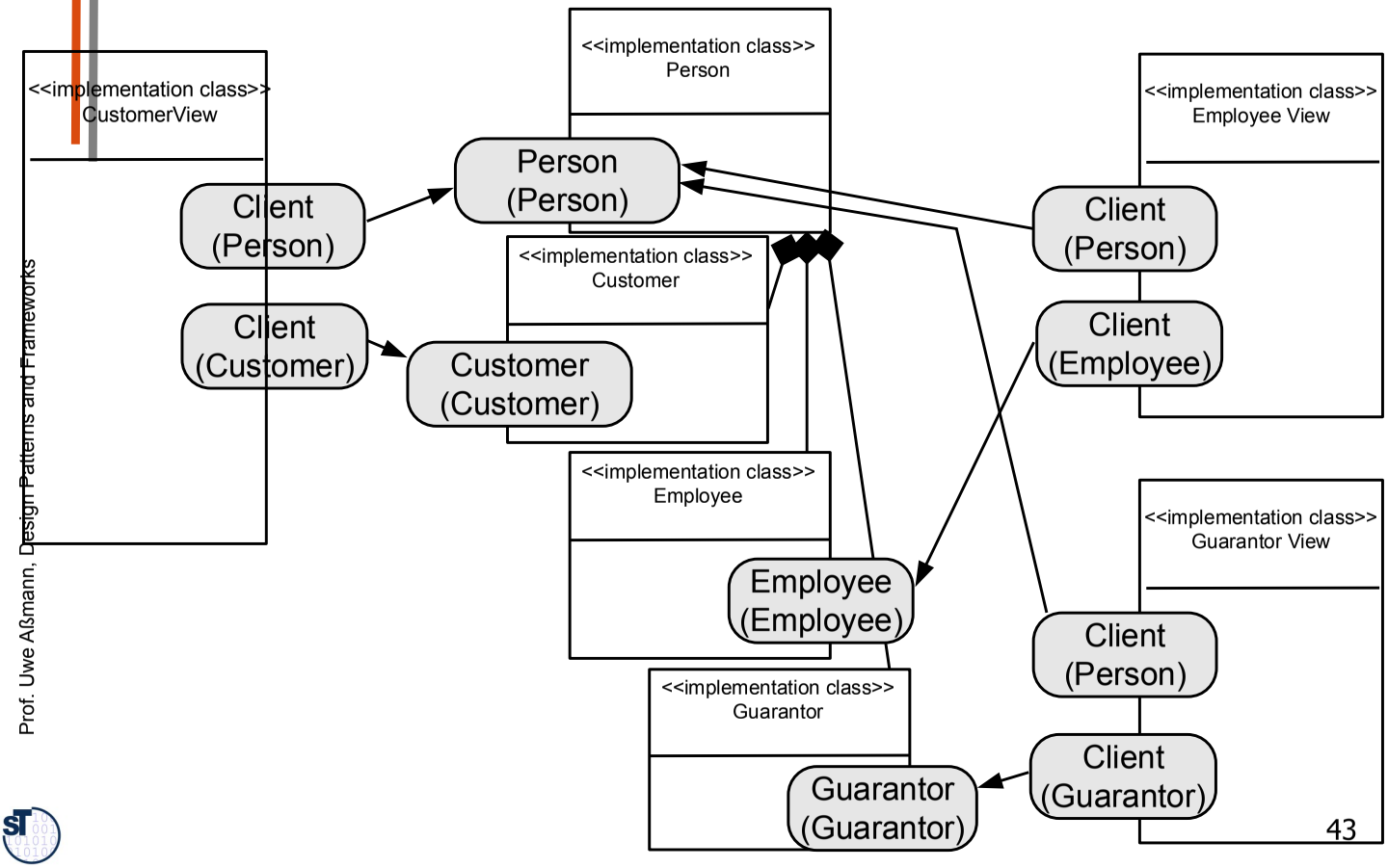
Role Models of Persons



Implementation With Interfaces (or Mixins)



Implementation of Person With Multi-Bridge (Role Objects)

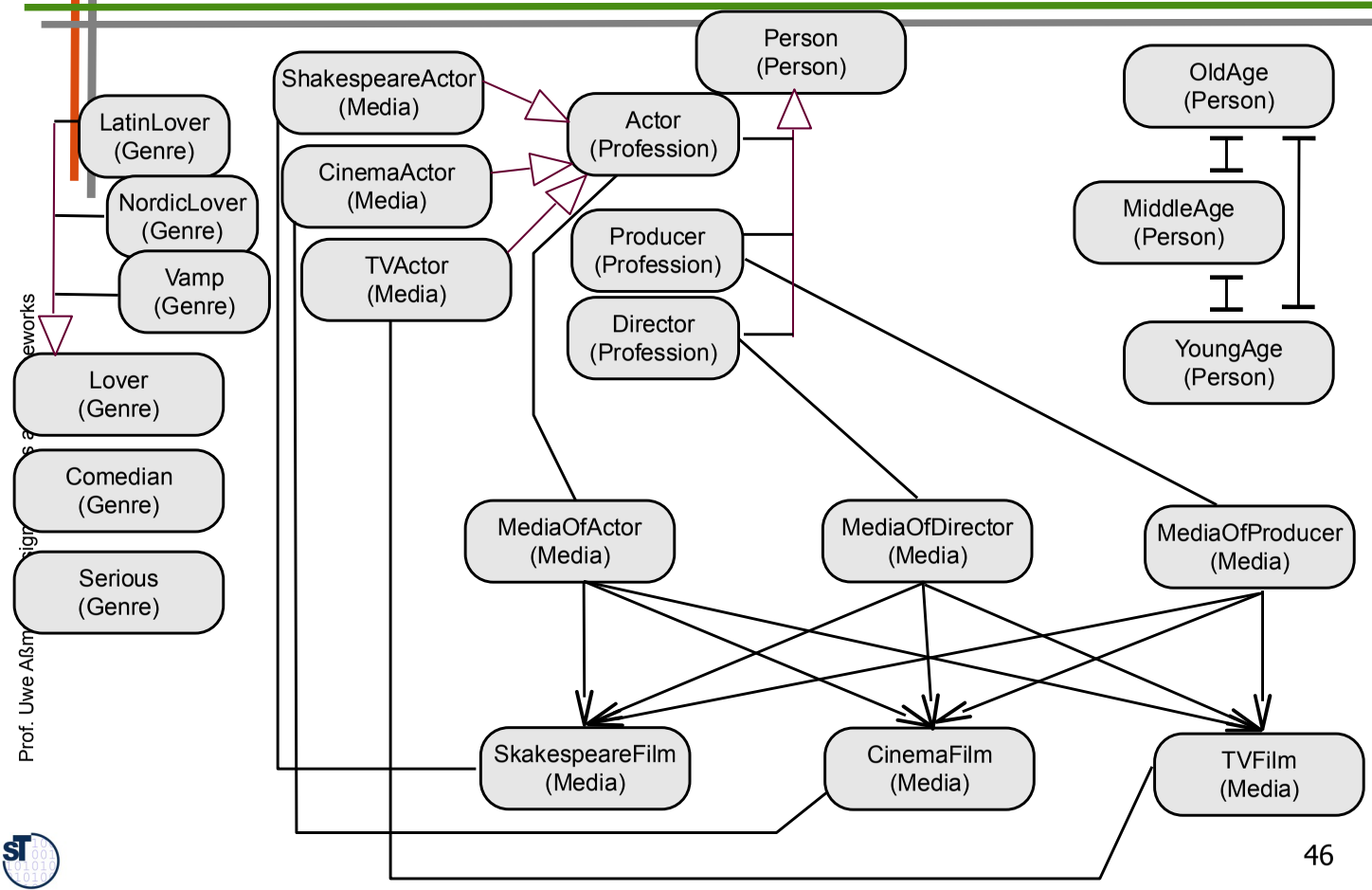


Example: Actors, Films, and Directors

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

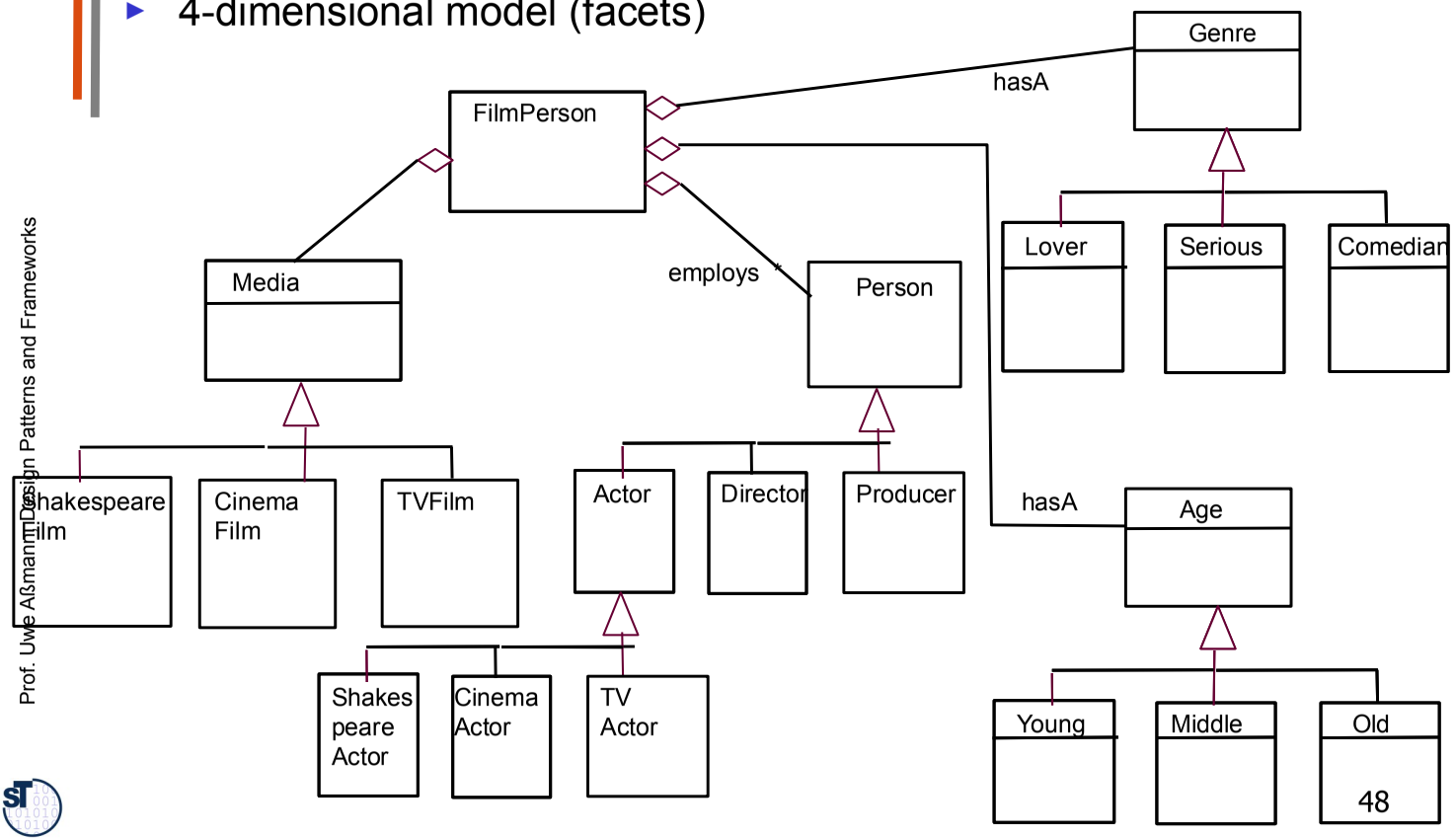


There are Many Ways to Implement This Role Model

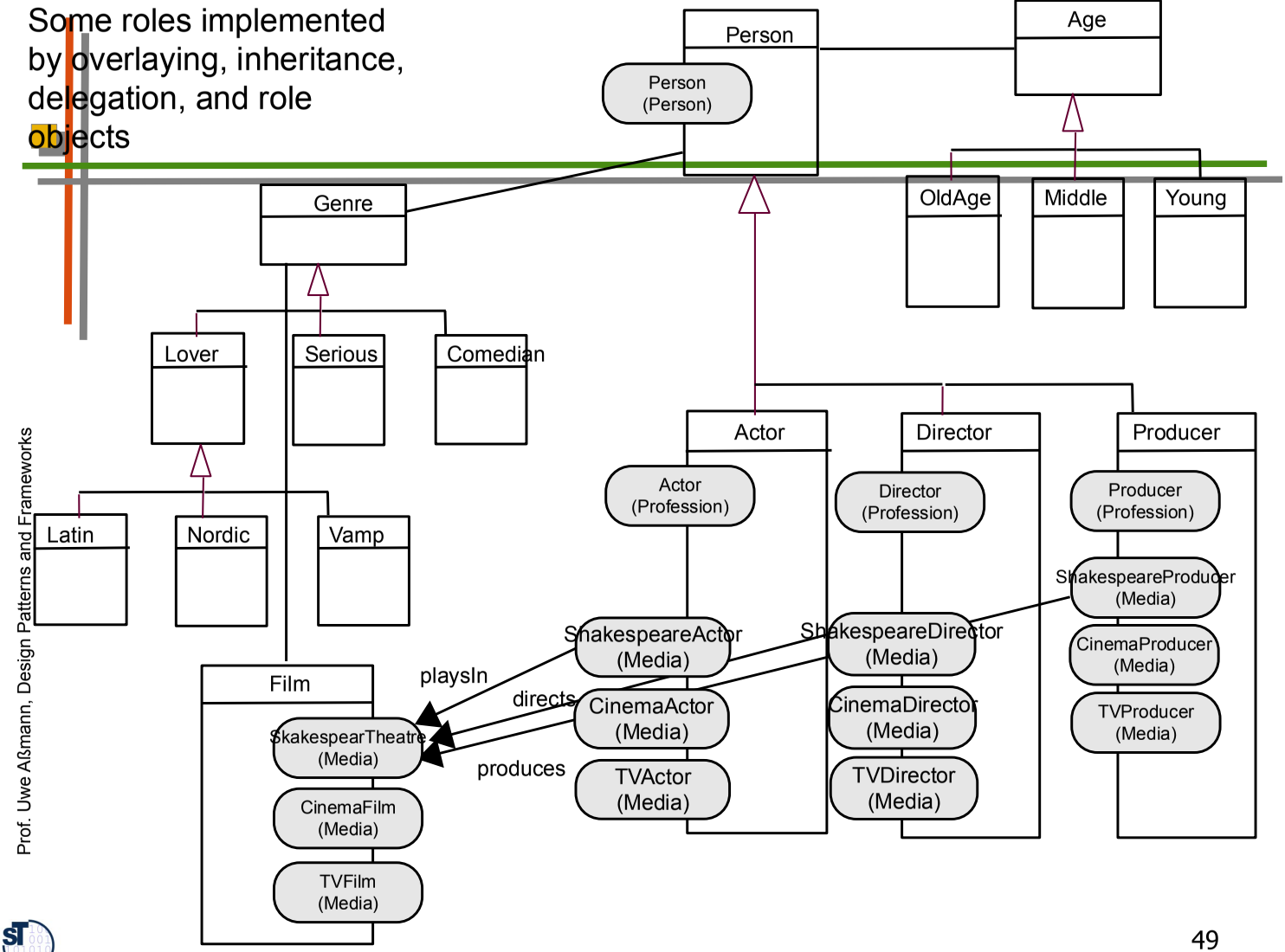
- With a facet based model, modelling some role models as class hierachies of a Dimensional Hierarchies model

Very Simple Class Model for Actors and Films

- 4-dimensional model (facets)



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

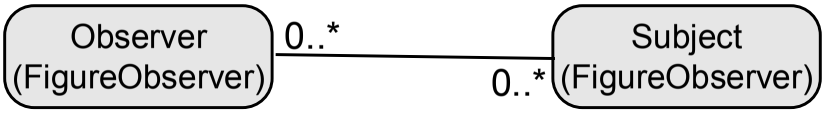


10.5 Design Patterns as Role Diagrams

... more info...

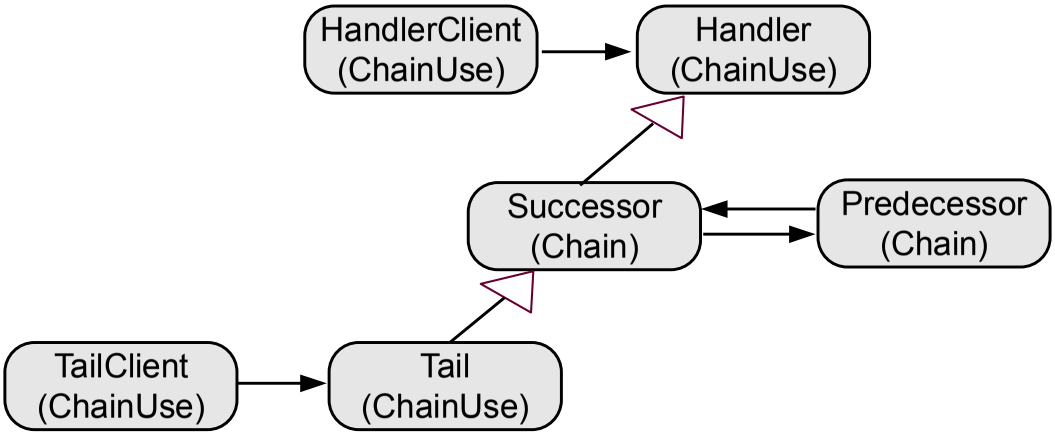
Design Patterns have Role Models

- Observer role model



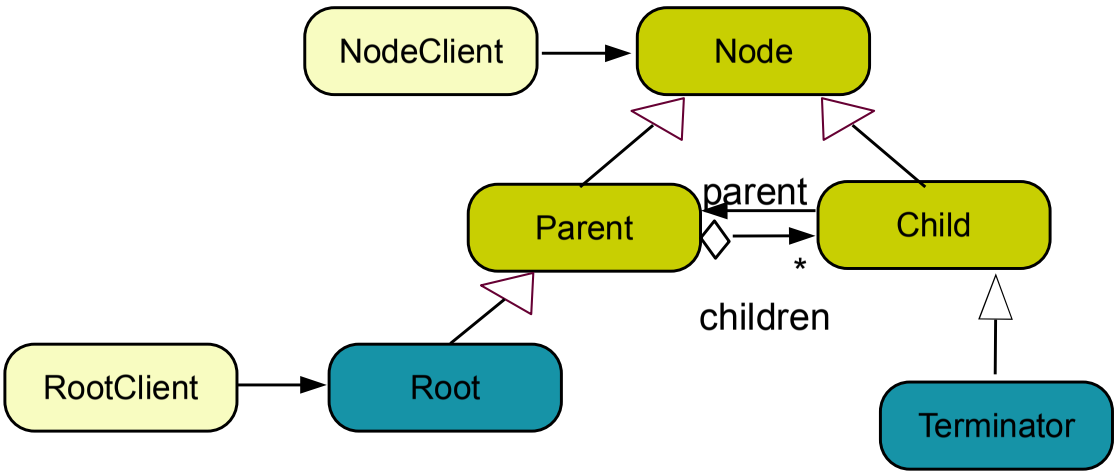
Structure Diagrams of DP are Role Diagrams

- The “participant” section of a GOF pattern is a *role model*
- Roles of Chain of Responsibility:
  - Chain: (successor, predecessor)
  - ChainUse: (Handler, HandlerClient, Tail, TailClient)



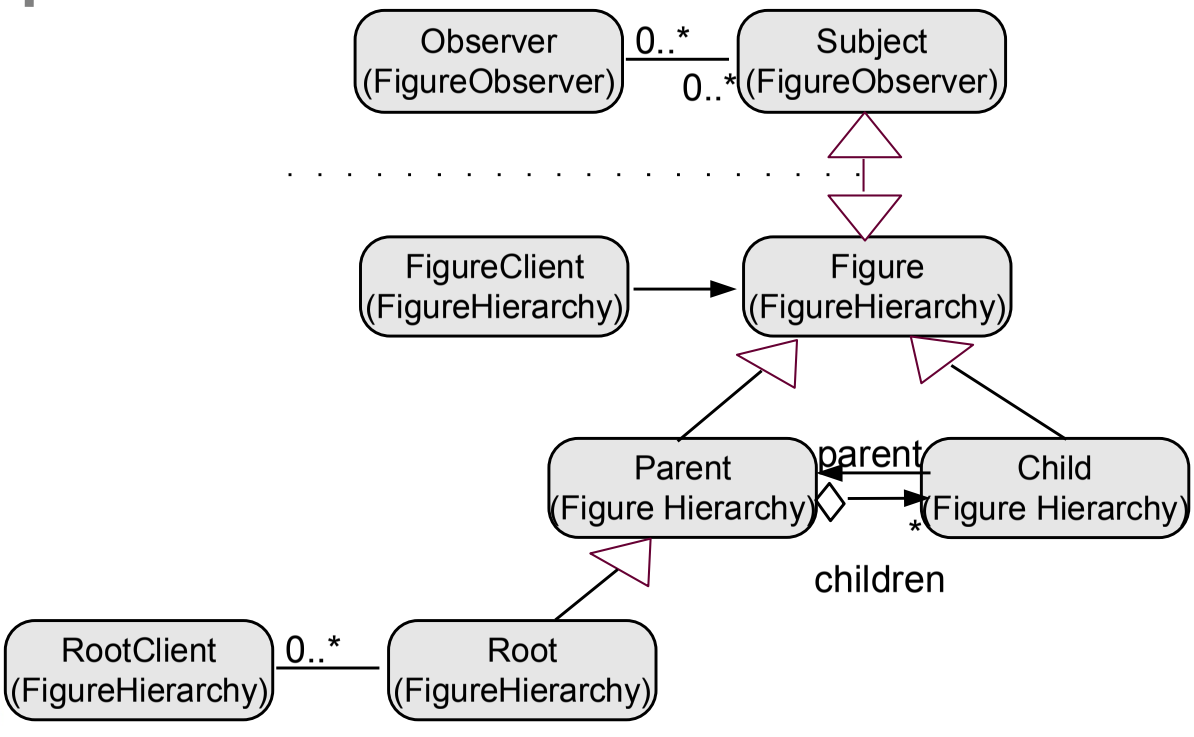
Role Diagram of Composite

- Root role is not in the standard pattern description
- Attention: role models are not standardized – it depends on the designer what she wants to model! (many variants of a role model for a design pattern may exist). Here: Root, Terminator, clients optional



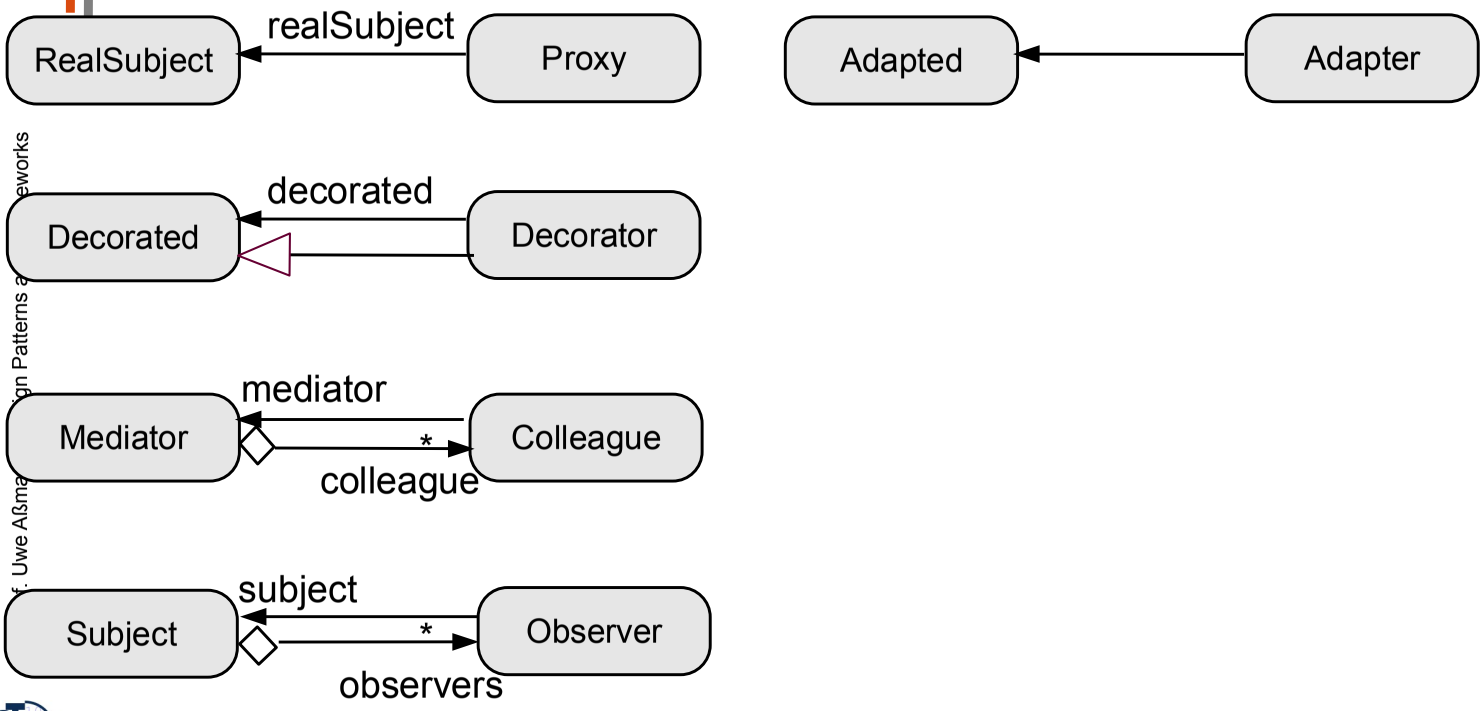
Composing (Overlaying) Role Models

- Overlaying the FigureHierarchy with the FigureObserver role model



Core Role Diagrams of Several Patterns

- Many of them are quite similar

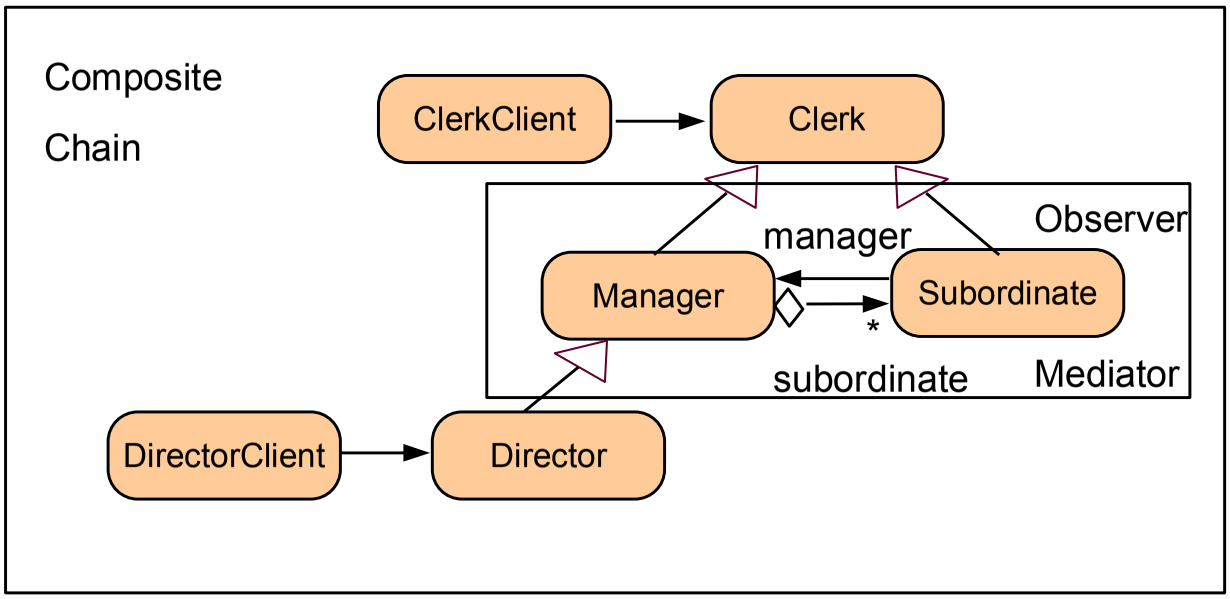


10.6 Composite Design Patterns with Role Model Composition

.. how to create bigger design patterns as composed role models..

Example: Bureaucracy

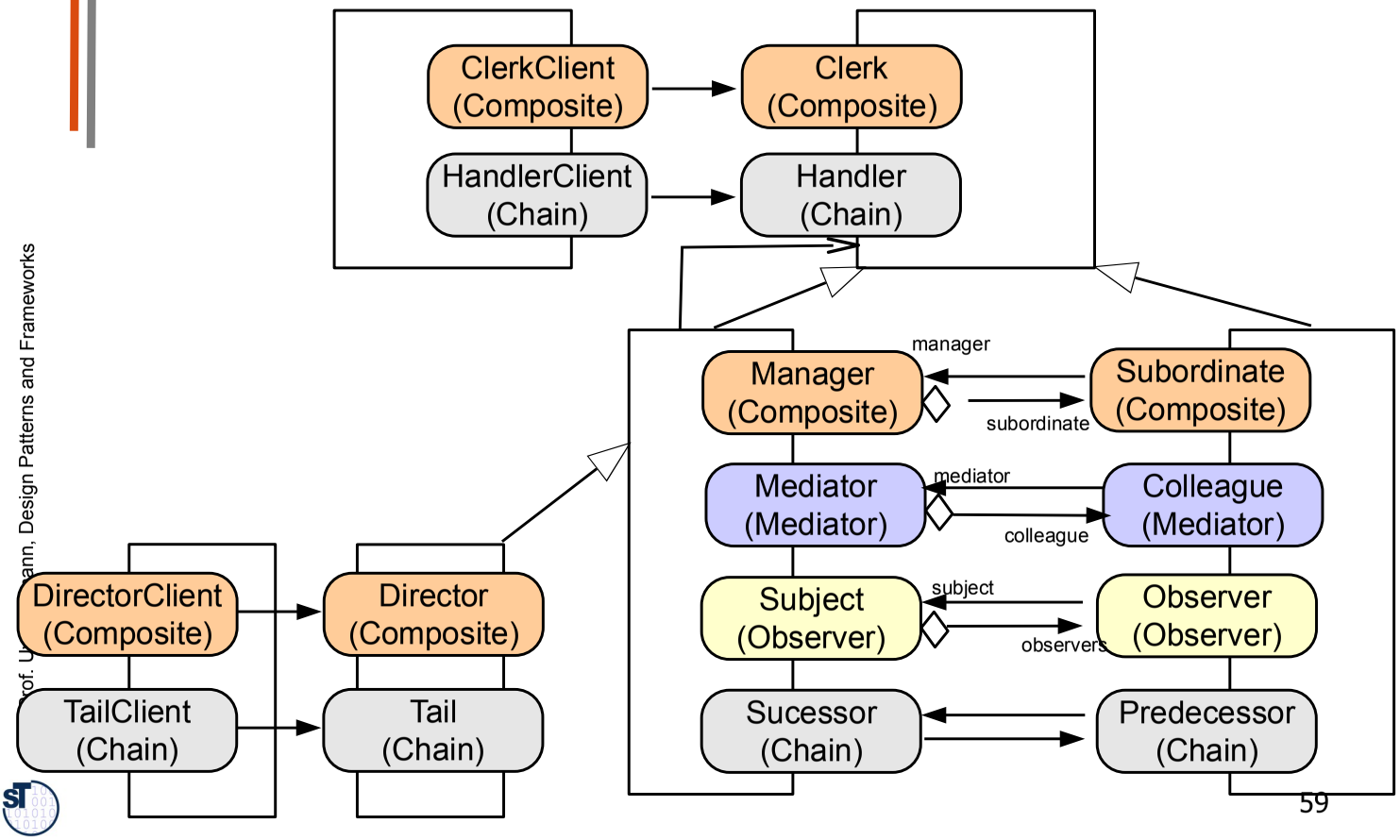
- A pattern to model organizations that have a tree-like structure (as opposed to matrix organizations)
- Is composed of the role models of Composite, Mediator, Chain, Observer



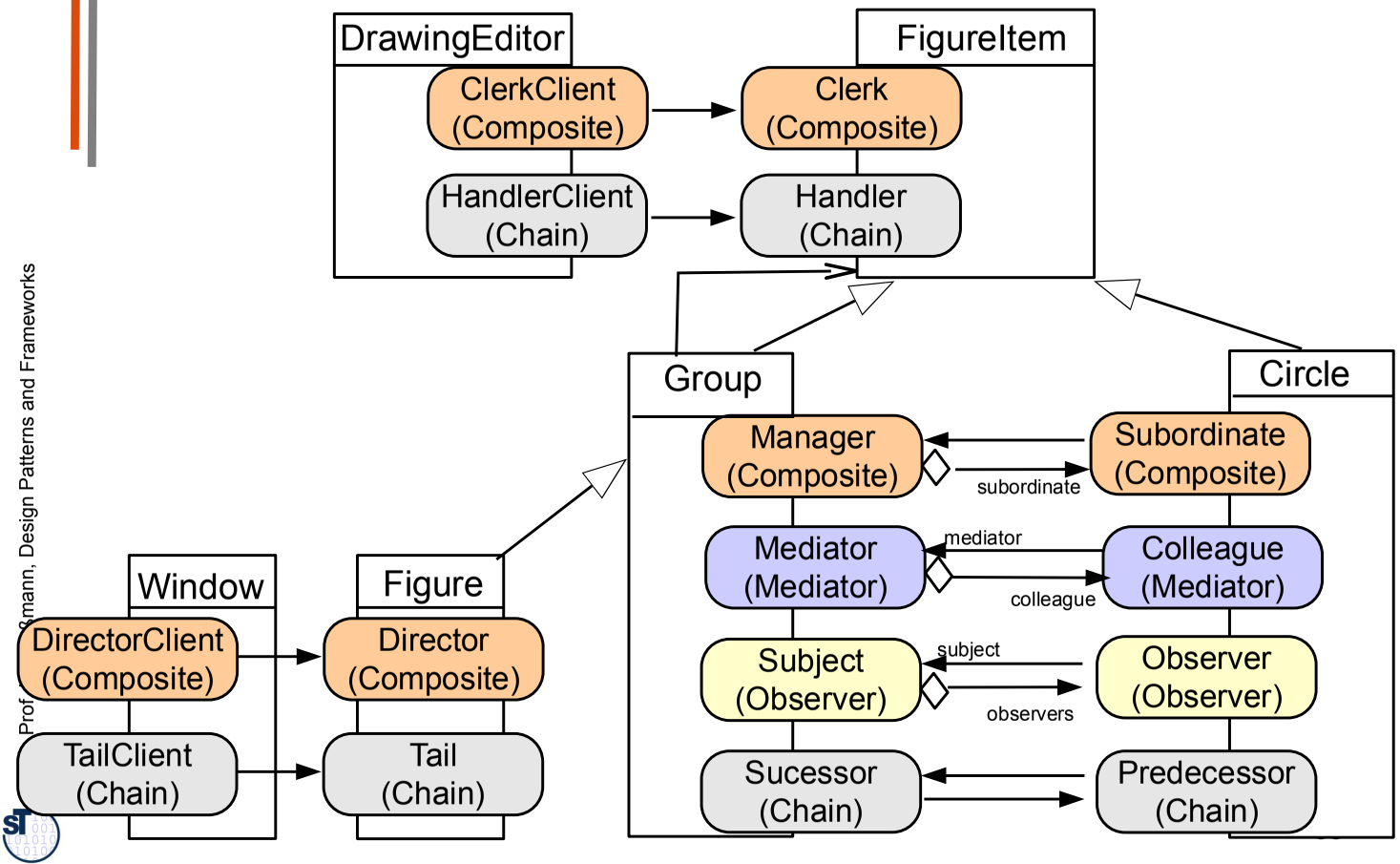
Example: Bureaucracy

- The *Composite* defines the organizational hierarchy of managers
- The *Mediator* is used to let talk children talk to their siblings (colleague roles) via a parent (mediator role)
- The *Chain* handles requests of clients
  - Every node may handle requests
  - If a node cannot handle a request, it is passed up in the hierarchy (on the path to the root)
- The *Observer* is used to listen to actions of a parent node
  - If a parent node (subject) changes something, its child (observer) listens and distributes the information accordingly

Class-Ability Model of Bureaucracy



Bureaucracy
Class-Ability Model of Figures



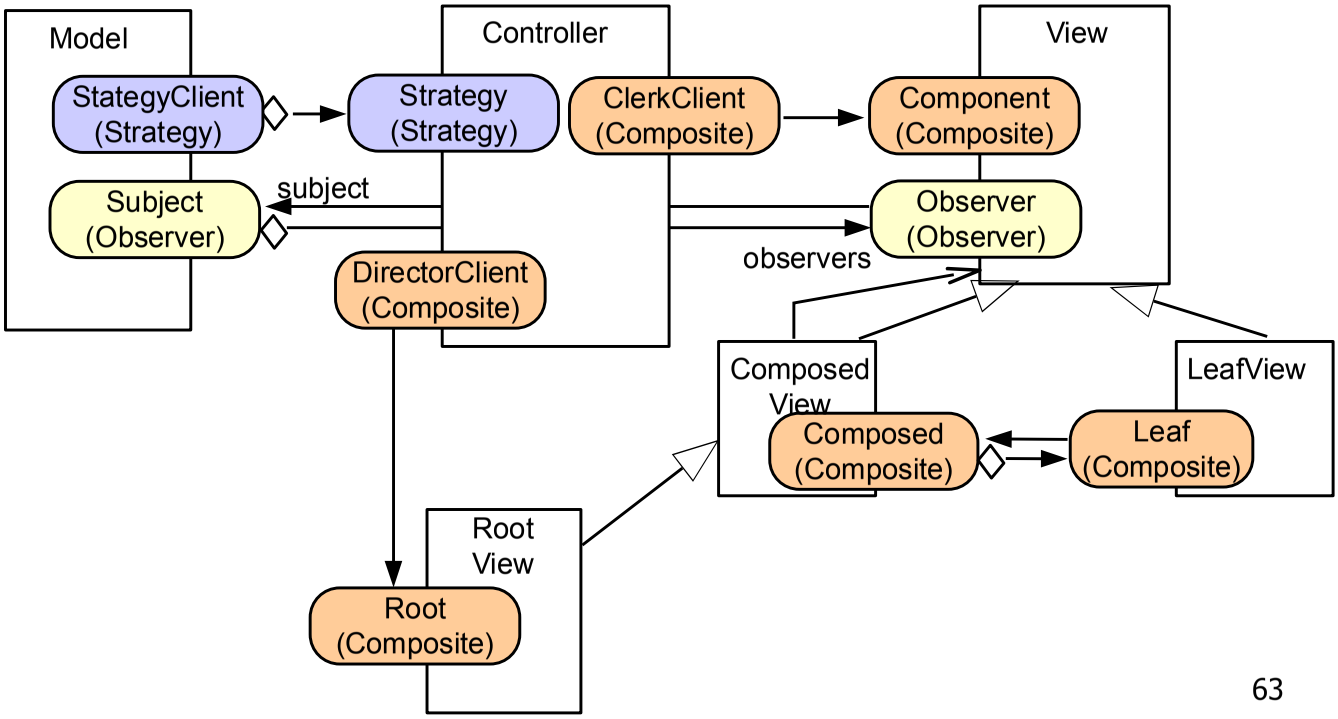
Application of Bureaucracy

- For all hierarchies
  - Figures in graphic and interactive applications
  - Widgets in GUIs
  - Documents in office systems
  - Piece lists in production management and CAD systems
  - Hierarchical tools in TAM (see later)
  - Modelling organizations in domain models: companies, governments, clubs

Model-View-Controller (MVC)

Class-Ability Model of MVC

- From Tyngre Reenskaug and Adele Goldberg
- MVC role model can be composed from the role models of Observer, Strategy, Composite



This Closes a Big Loop

- Remember, Reenskaug developed MVC 1978 with Goldberg, while working on Smalltalk-78 port for Norway
- Starting from his MVC pattern, Reenskaug has invented role-based design
- 1998, Riehle/Gross transferred role-based models to design patterns
- Today, MVC can be explained as composed role models of other design patterns

Riehle-Gross Law On Composite Design Patterns

The role model of a composite design patterns is composed of the role models of their component design patterns

- Concequences
  - Complex patterns can be easily split into simpler ones (decomposition)
  - Variants of patterns can more easily be related to each other (variability of patterns)
    - e.g., ClassAdapter and ObjectAdapter
  - Template&Hook conceptual pattern can be explained as role model (see next chapter)

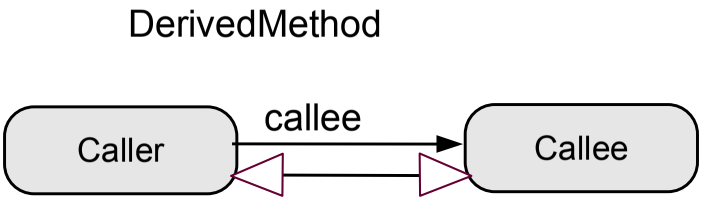
10.6.2 Composition of Simple Variability Patterns

Warning

- The following is an attempt to build up the basic GOF patterns from simple role models
  - It is probably not stable
- It explains why Strategy is different from Bridge and TemplateClass, etc.

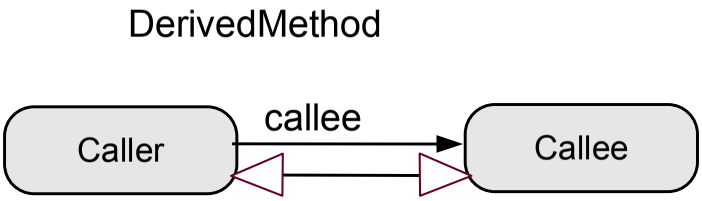
Derived Method

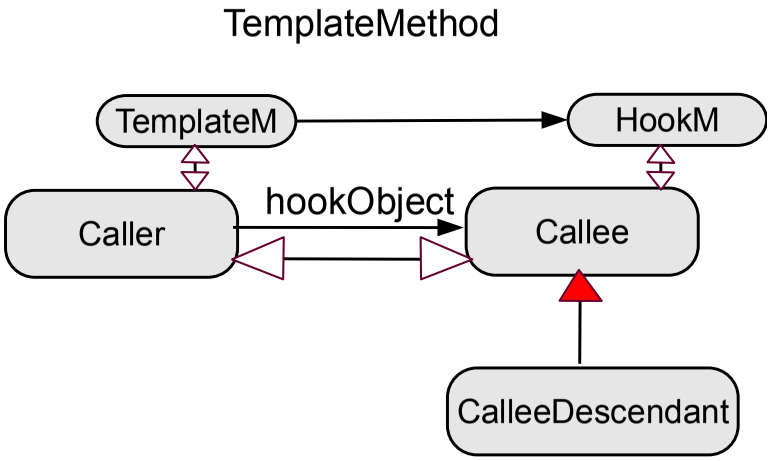
- In a class,
  - A *kernel method* implements the feature directly on the attributes of the class, calling no other method
  - A *derived method* is implemented by calling only kernel methods



Derived Method and TemplateMethod

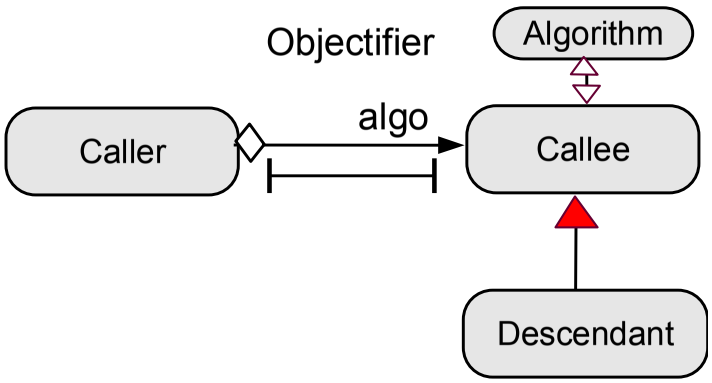
- TemplateMethod is a DerivedMethod that has
  - an additional TemplateMethod/HookMethod role model
  - Inheritance hierarchy on right side (implied by role-class inheritance constraint)
  - The template role implies no hierarchy on left side

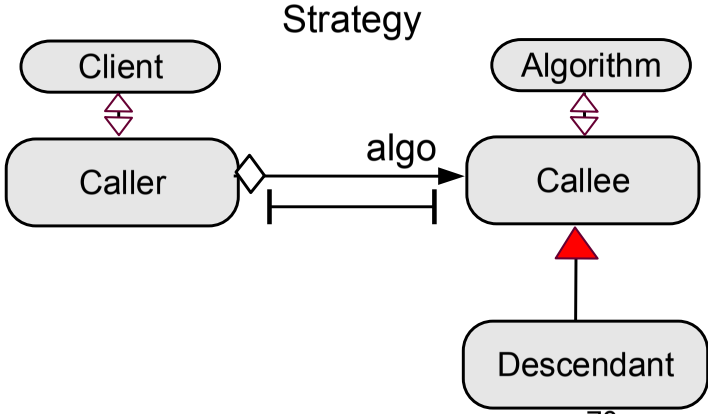




Objectifier and Strategy

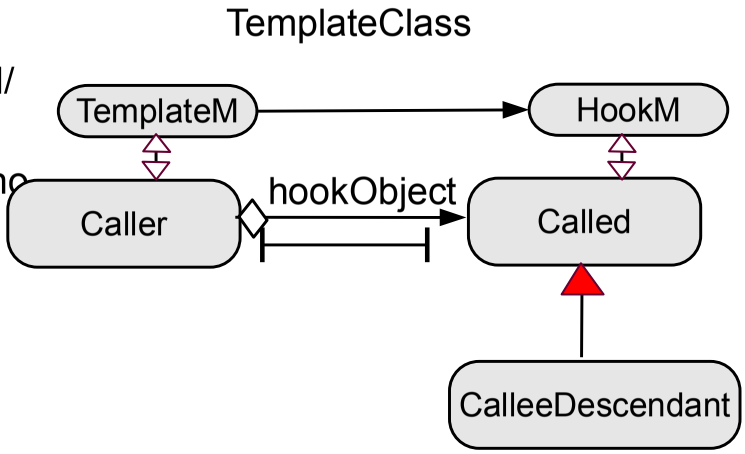
- Objectifier has
  - An additional exclusion constraint on Caller and Callee
  - An aggregation
  - An algorithm role
  - A subclassing constraint (right hierarchy)
  - No template role
- Strategy is an Objectifier with
  - Client role
  - Algorithm role
  - Hierarchy on right side
  - No template role





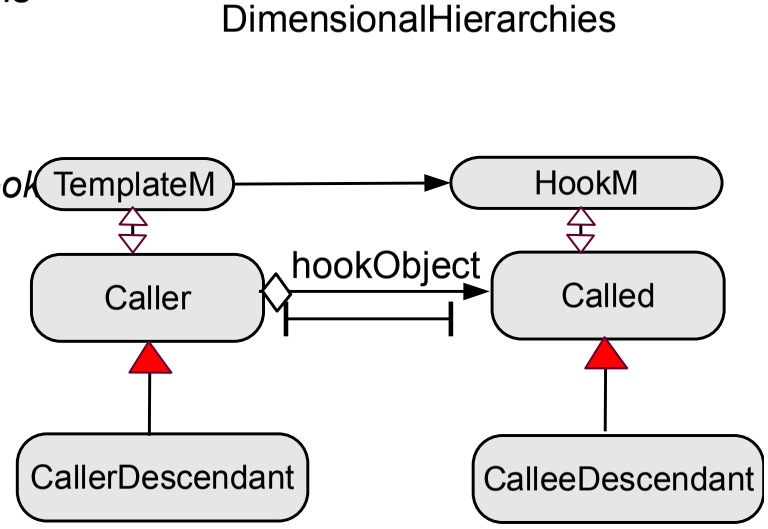
TemplateClass

- TemplateClass is an Objectifier with
  - An additional TemplateMethod/ HookMethod role model
  - TemplateMethod role implies no hierarchy on left side
  - HookMethod role implies inheritance hierarchy on right side
  - *No client or algorithm role,* otherwise like Strategy



DimensionalClassHierarchies

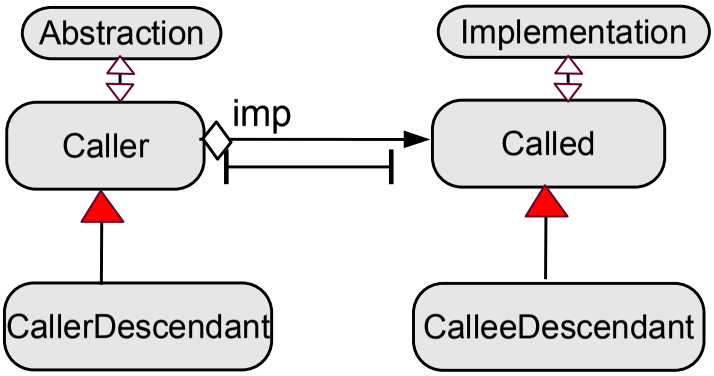
- DimensionalClassHierarchies is a TemplateClass
  - *Without template-hook constraint, but still TemplateMethod/TemplateHook constraint*
  - With left hierarchy constraint



Bridge

- Bridge is a DimensionalHierarchies with
  - An additional abstraction/implementation role model
  - *No template/hook role*

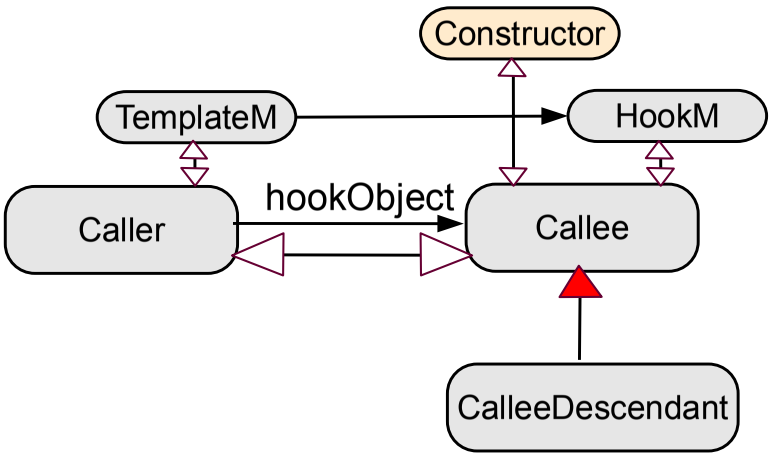
Bridge



Creational Patterns

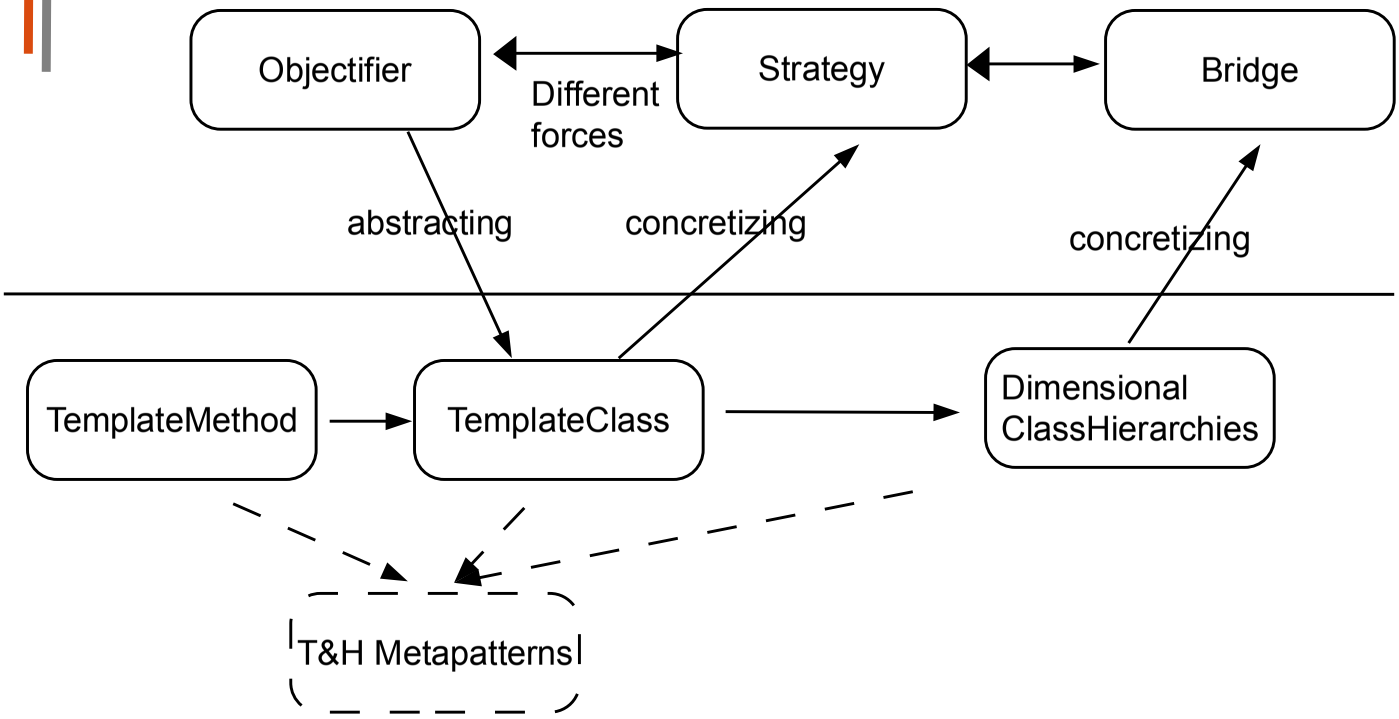
- Add more roles with semantics about creation
- E.g., FactoryMethod is a TemplateMethod with a creational role model

FactoryMethod



Remember: Relation TemplateMethod, TemplateClass, Strategy, Observer

More specific patterns (with more intent, more pragmatics, specific role denotations)

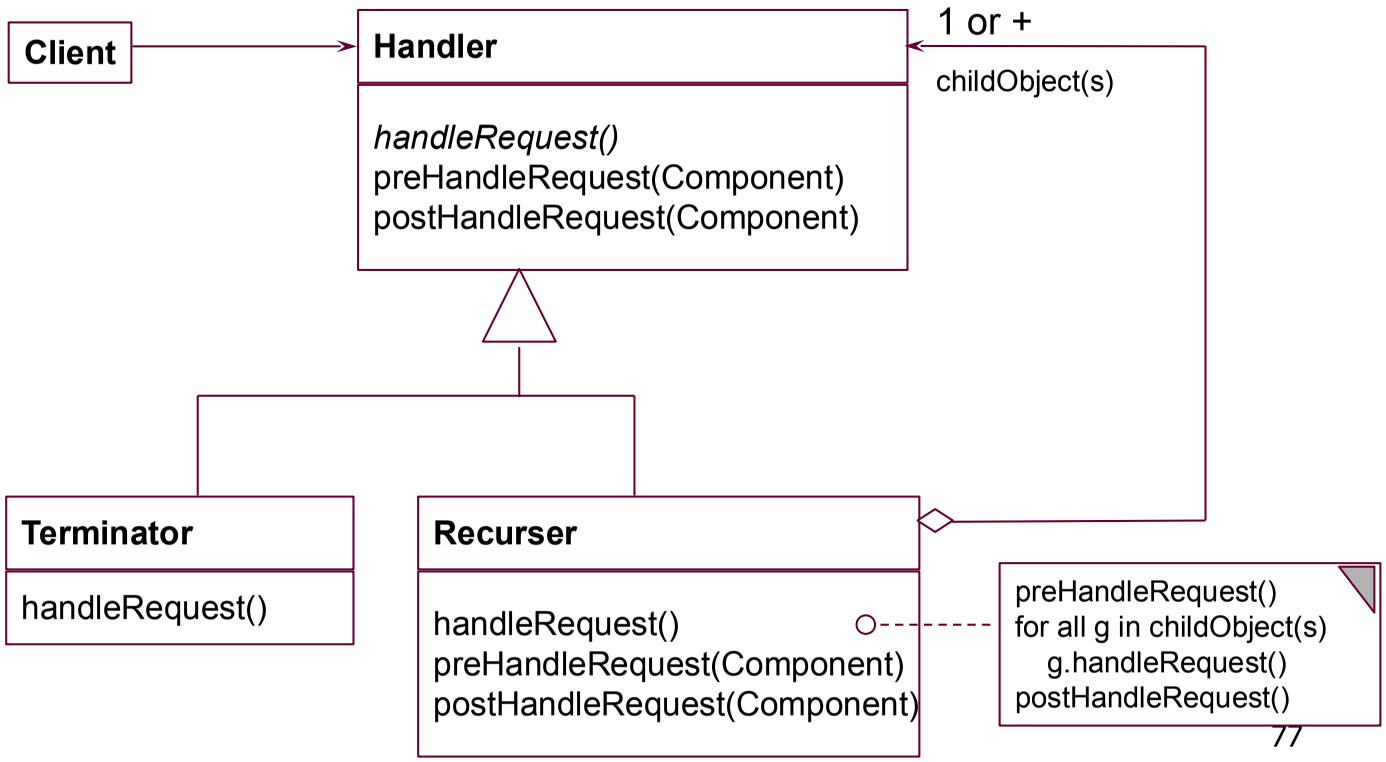


Framework Patterns (with TemplateM/HookM role model)

10.6.3 Composition of Simple Extensibility Patterns

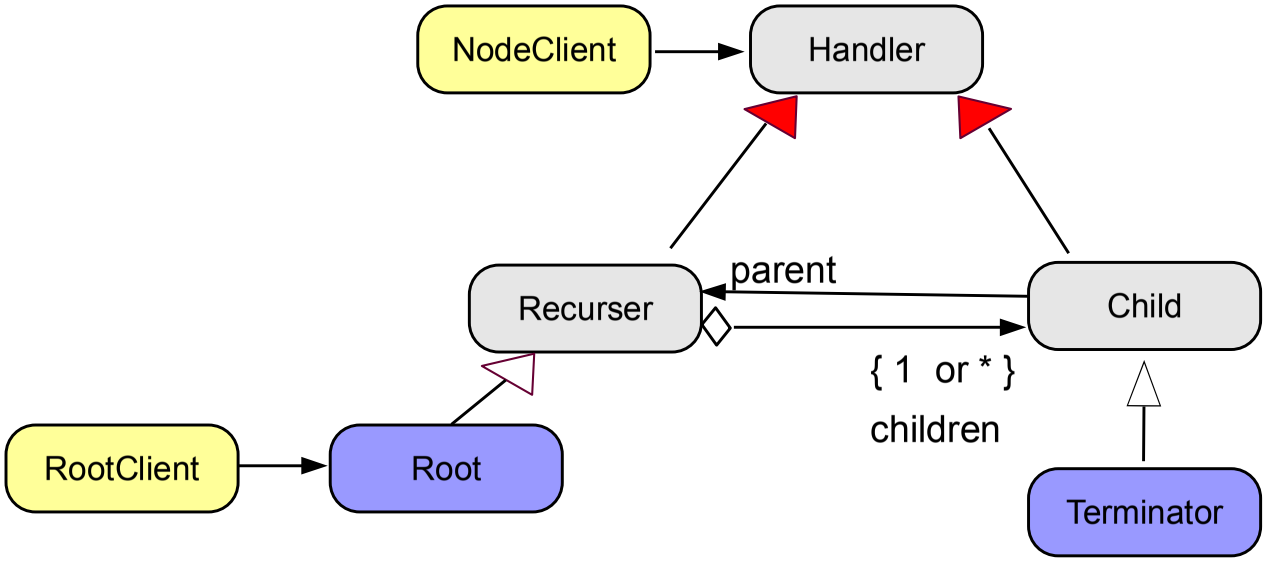
Object Recursion

- The aggregation can be 1:1 or 1:n (1-Recursion, n-Recursion)



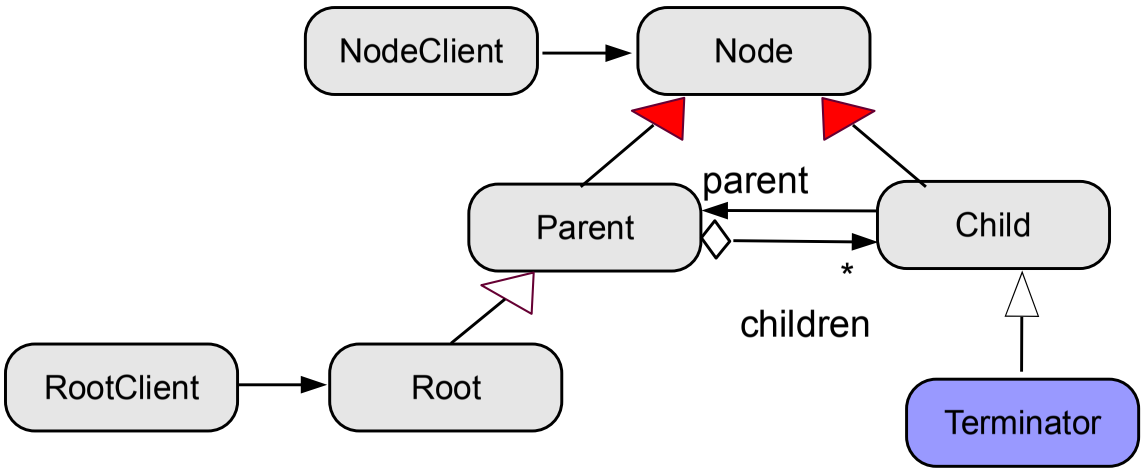
ObjectRecursion

- Essential roles are Handler, Recurser, Child
- Root, Terminator can, but need not be modeled
- Clients are optional, parent is optional



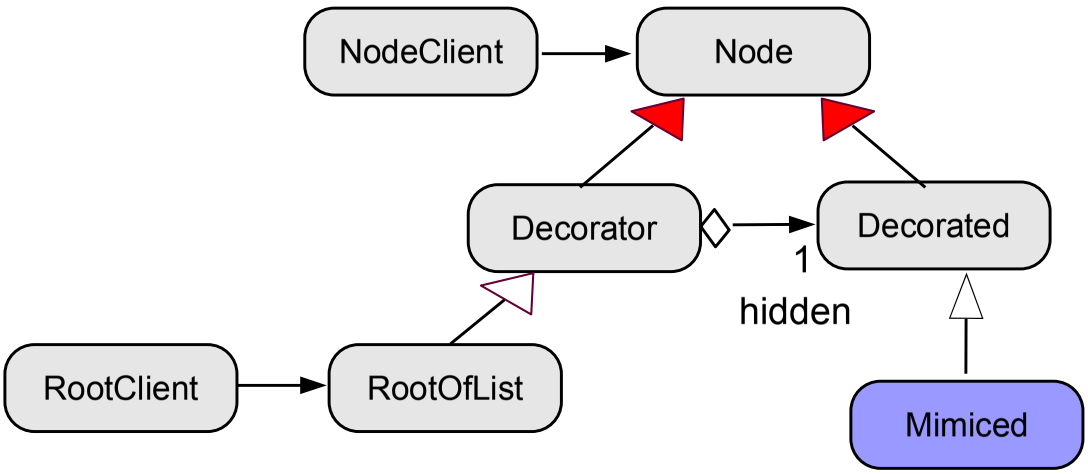
Composite

- n-ObjectRecursion
- Other role pragmatics, similar pattern
- Perhaps with additional parent relation



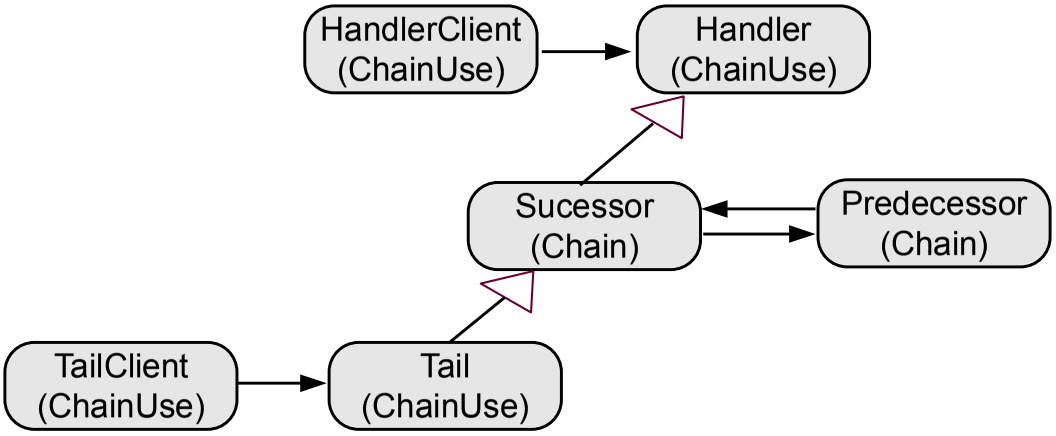
Decorator

- 1-ObjectRecursion
- other role pragmatics, similar pattern

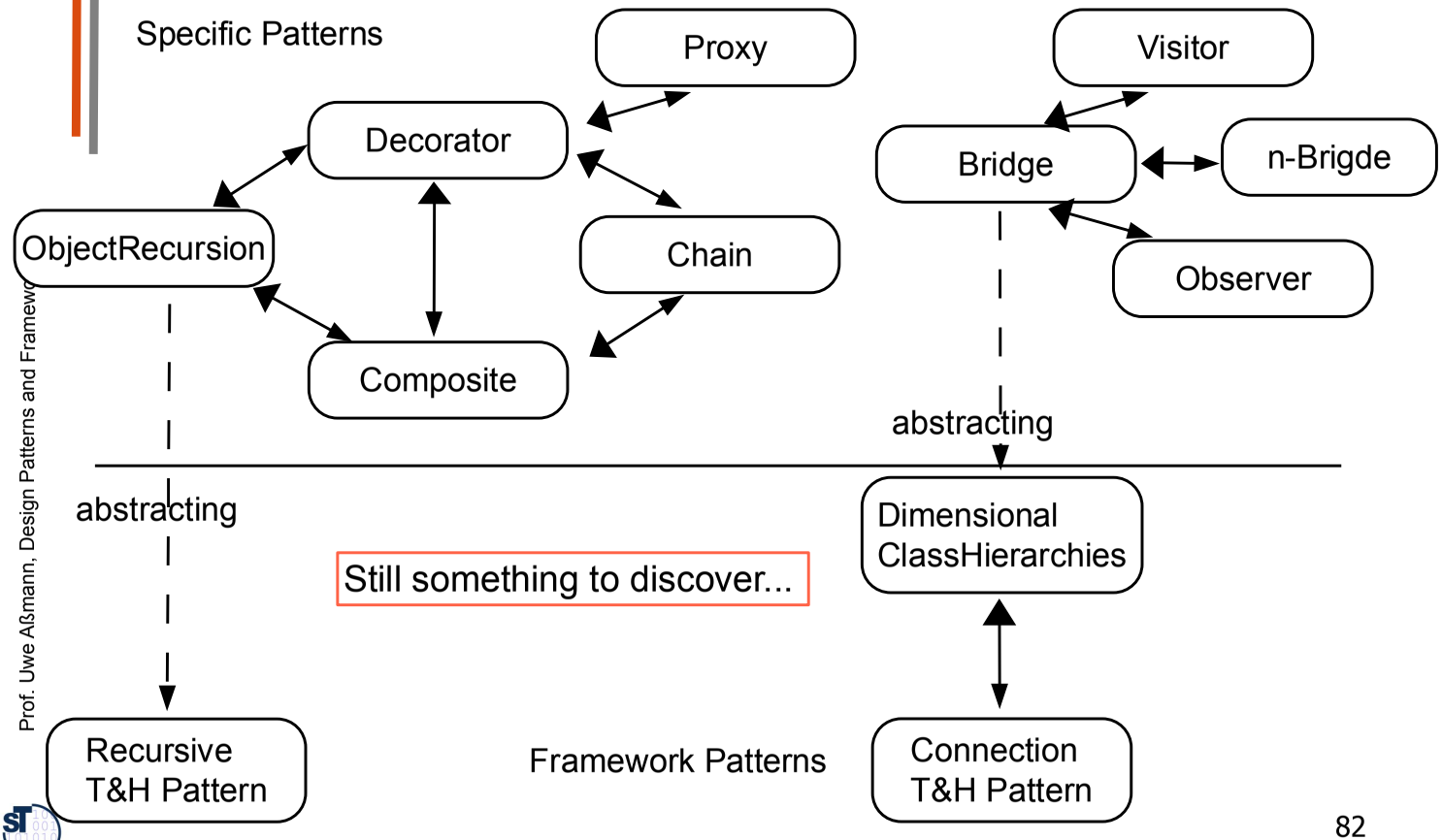


Chain of Responsibility

- No real ObjectRecursion



Remember: Relations Extensibility Patterns

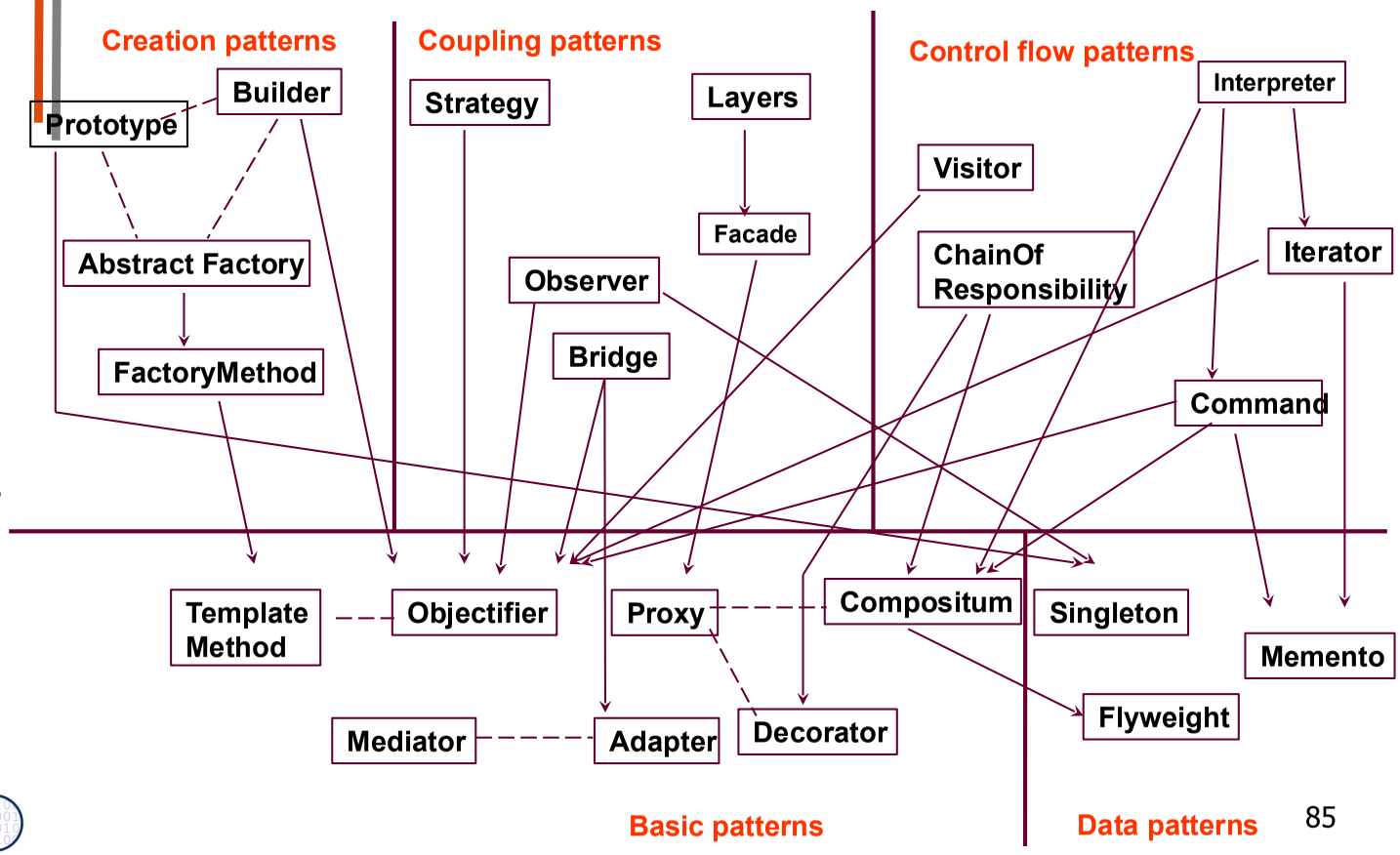


10.6.4 Consequences of the Riehle/Gross Law

Zimmer's Classification and the Riehle-Gross Law

- Zimmer's hierarchy notes use relationships between design patterns
  - But actually, he means composition of role models of design patterns
  - but Zimmer could not express it conceptually

Relations between Patterns [Zimmer, PLOP 1]



Consequence for Pattern-Based Design

- With different role models, the fine semantic differences between several patterns can be expressed syntactically
  - A role model can capture *intent (pragmatics)* of a pattern
  - While patterns can have the same structure, the intent may be different
  - It is possible to distinguish a Strategy, TemplateClass, a Bridge or DimensionalClassHierarchy
- This makes designs more explicit, precise, and formal

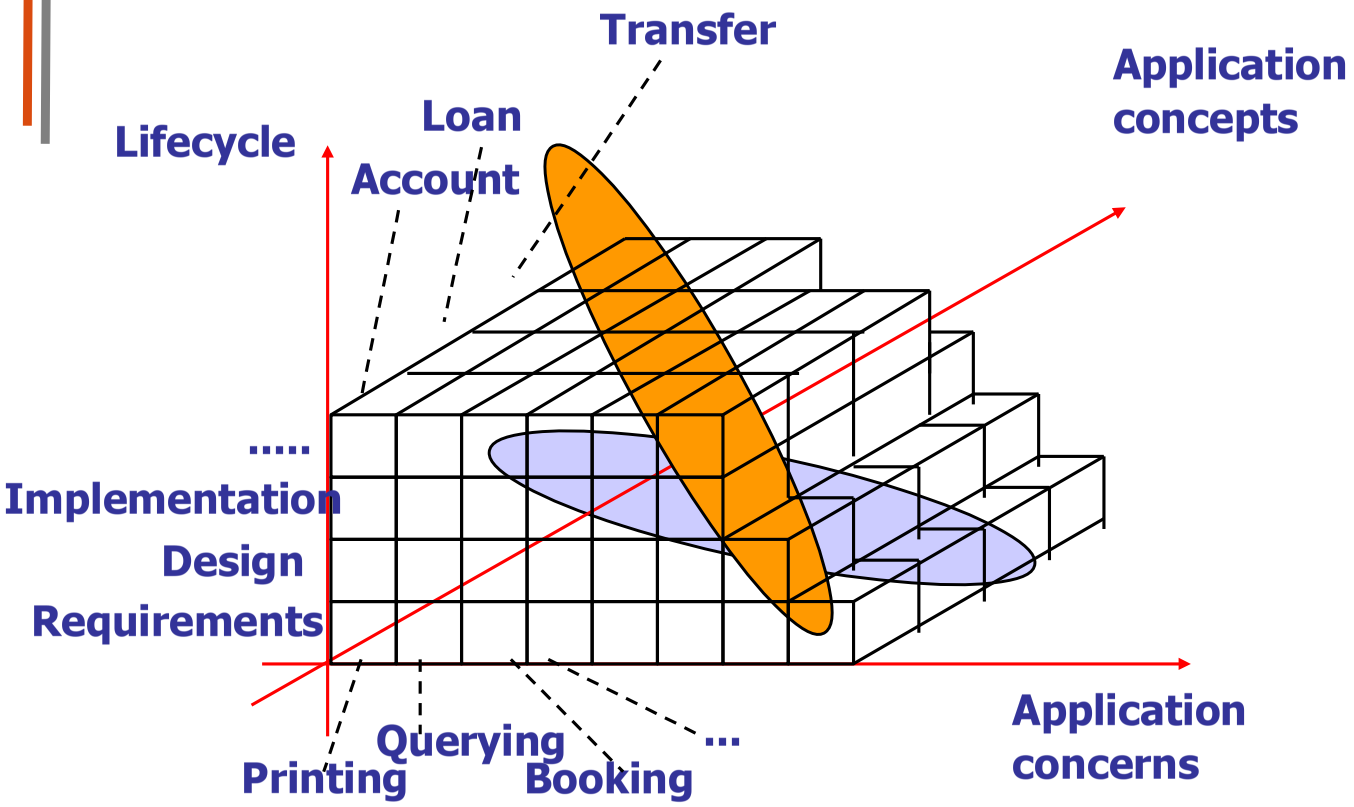
Consequence for Pattern Mining

- When you identify a pattern in the product of your company,
  - Try to define a role model
  - Split the role model into those that you know already
  - I.e., decompose the complex pattern in well-known ones
- Advantage:
  - You know how to implement the well-known patterns
  - You can check whether an implementation of the composite, new pattern is correct
  - If all component patterns are implemented correctly, i.e., conform to their role models.
  - Be Aware: These Role Models Are Not Stable
    - Role models provide freedom; so there may be several ones for one pattern

10.7 More on Roles

10.7.1 Relation of Role Modelling to Other Software Engineering Technologies

Hyperslices are Named Slices Through the Concern Matrix



Hyperslice Composition and Role Mapping

- 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

- 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 last over the entire lifetime of the object

10.7.2 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
- Semantically rigid types are *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

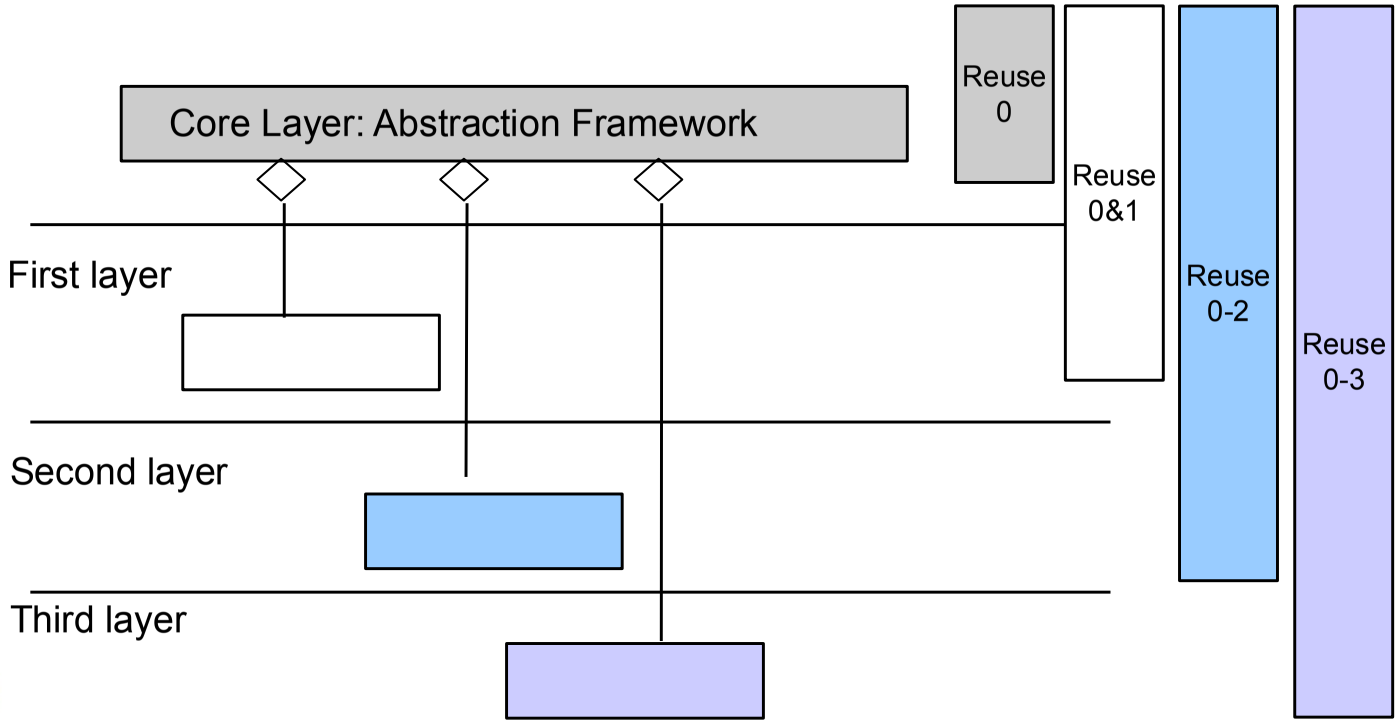
10.8 Effects of Role-Based Design Patterns on Frameworks and Applications

Effect of Role Models

- Role modelling allows for *scaling of delegation*
  - By default, all roles are overlaid by their class
  - But some can stay separate
  - Layered frameworks split all roles off to role objects

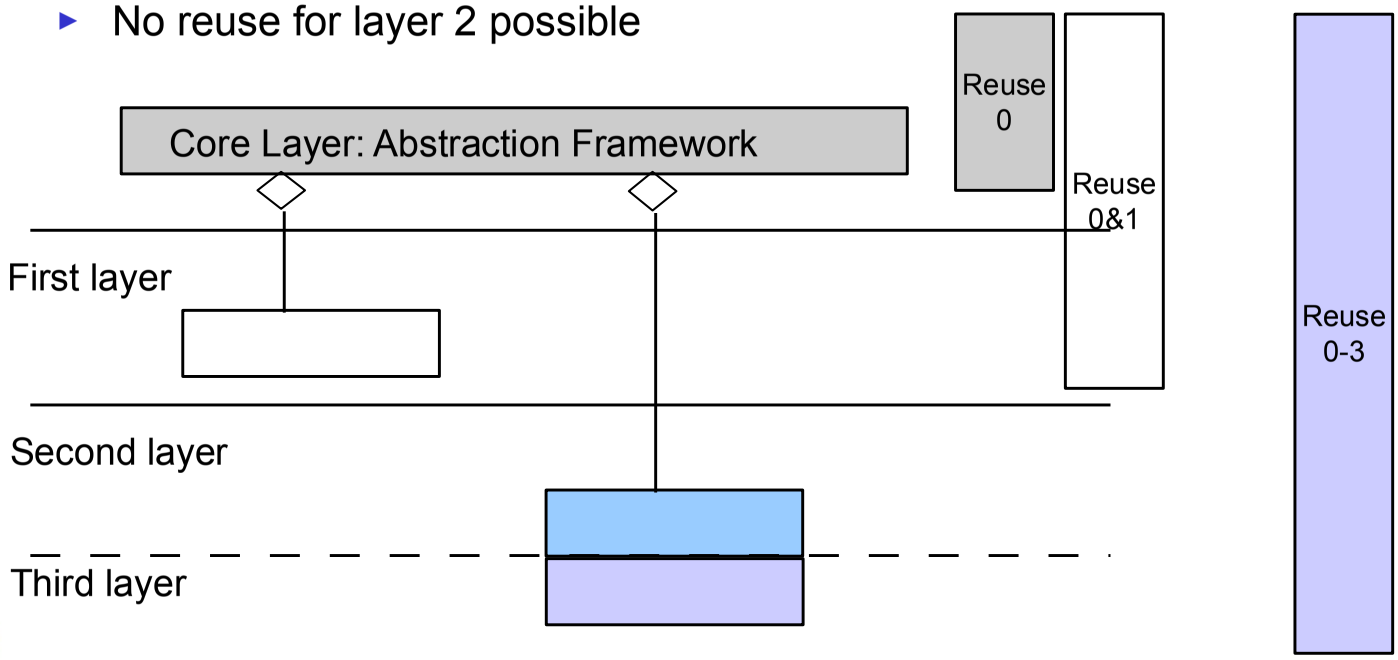
Role Models and Facet/Layered Frameworks

- An n-Bridge framework maintains roles (role models) in every facet (because a facet model is based on a class-role model)
- Similar for chain-Bridges and layered frameworks



Merging Layers of Facet/Layered Frameworks

- If the layers are seen as role models, it can be chosen to merge the layers, i.e., the role models
- Here: merge second and third layer into one physical implementation layer
- No reuse for layer 2 possible

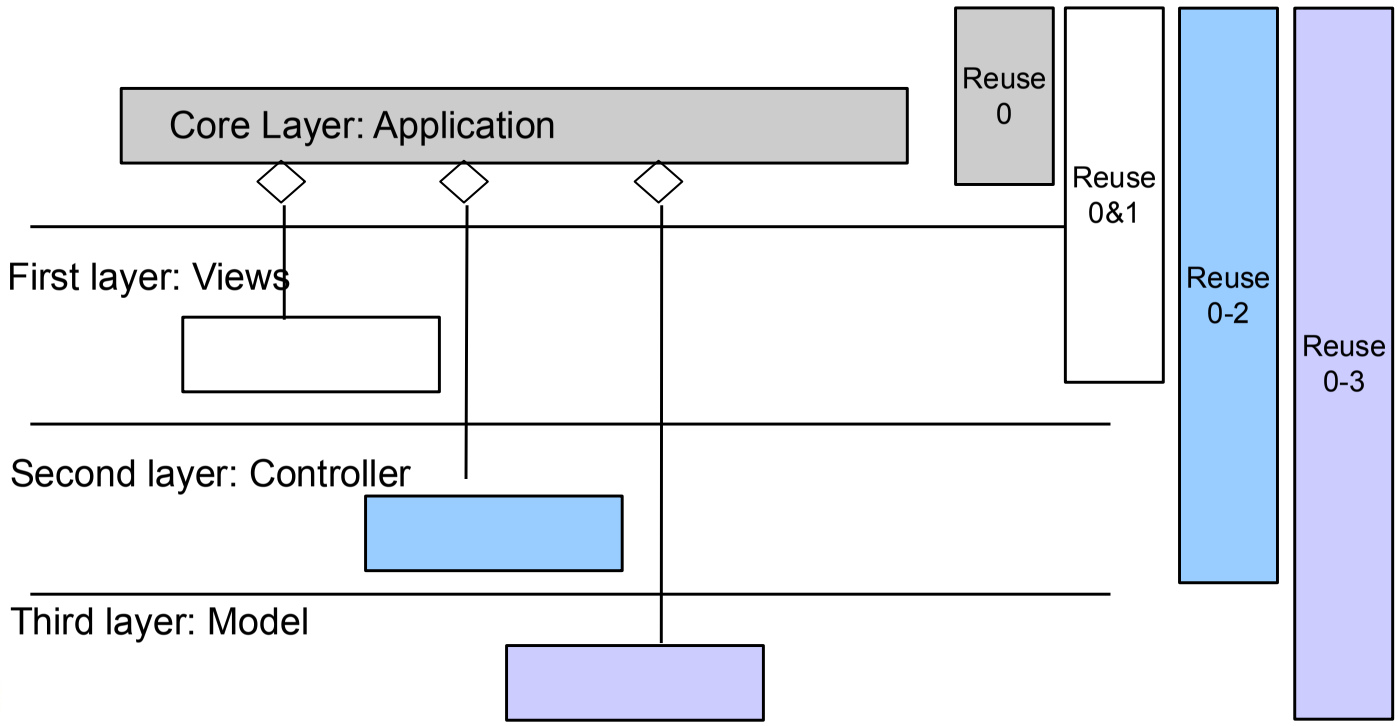


Merging Layers of Layered Frameworks

- When two layers are merged, the variability of a framework sinks
- But its applications are more efficient:
  - Less delegations (less bridges)
  - Less allocations (less physical objects)
  - Less runtime flexibility (less dynamic variation)

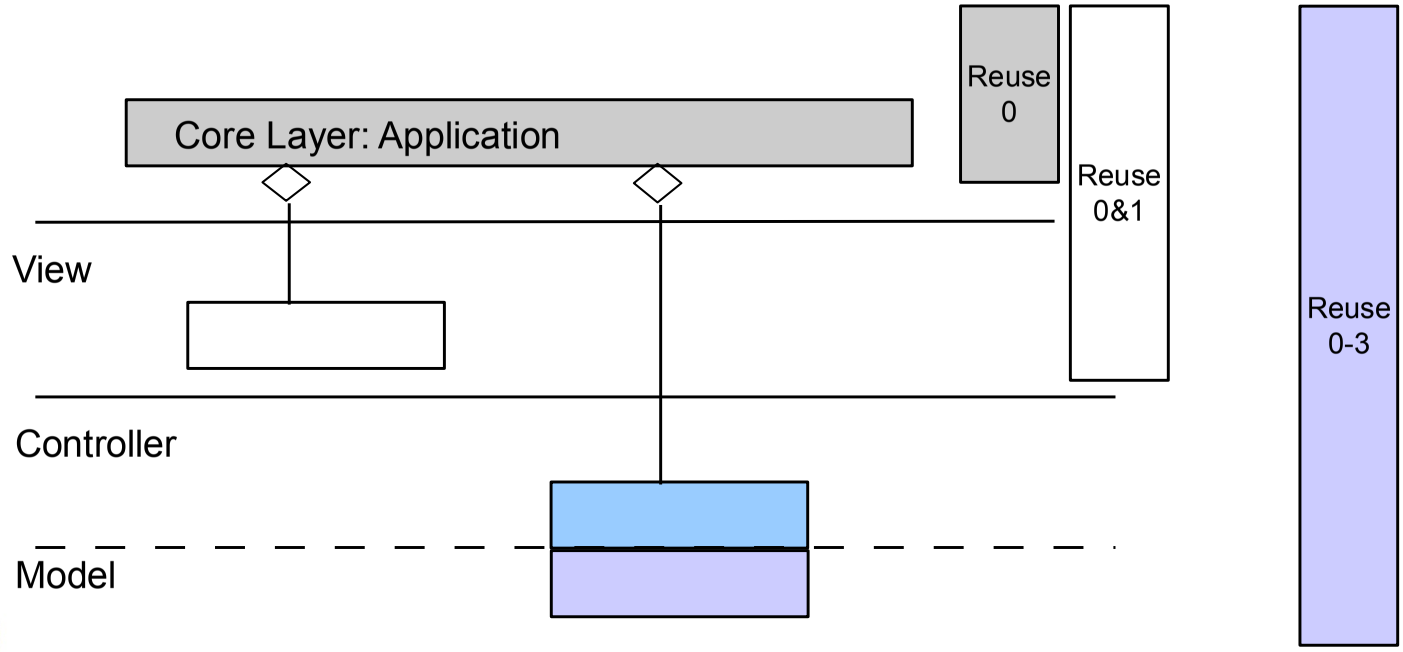
MVC as Multi-Bridge Framework

- The roles of MVC can be ordered in a n-Bridge framework



MVC as Optimized Multi-Bridge Framework

- Model and Controller layer can be merged
- Less variability, but also less runtime objects



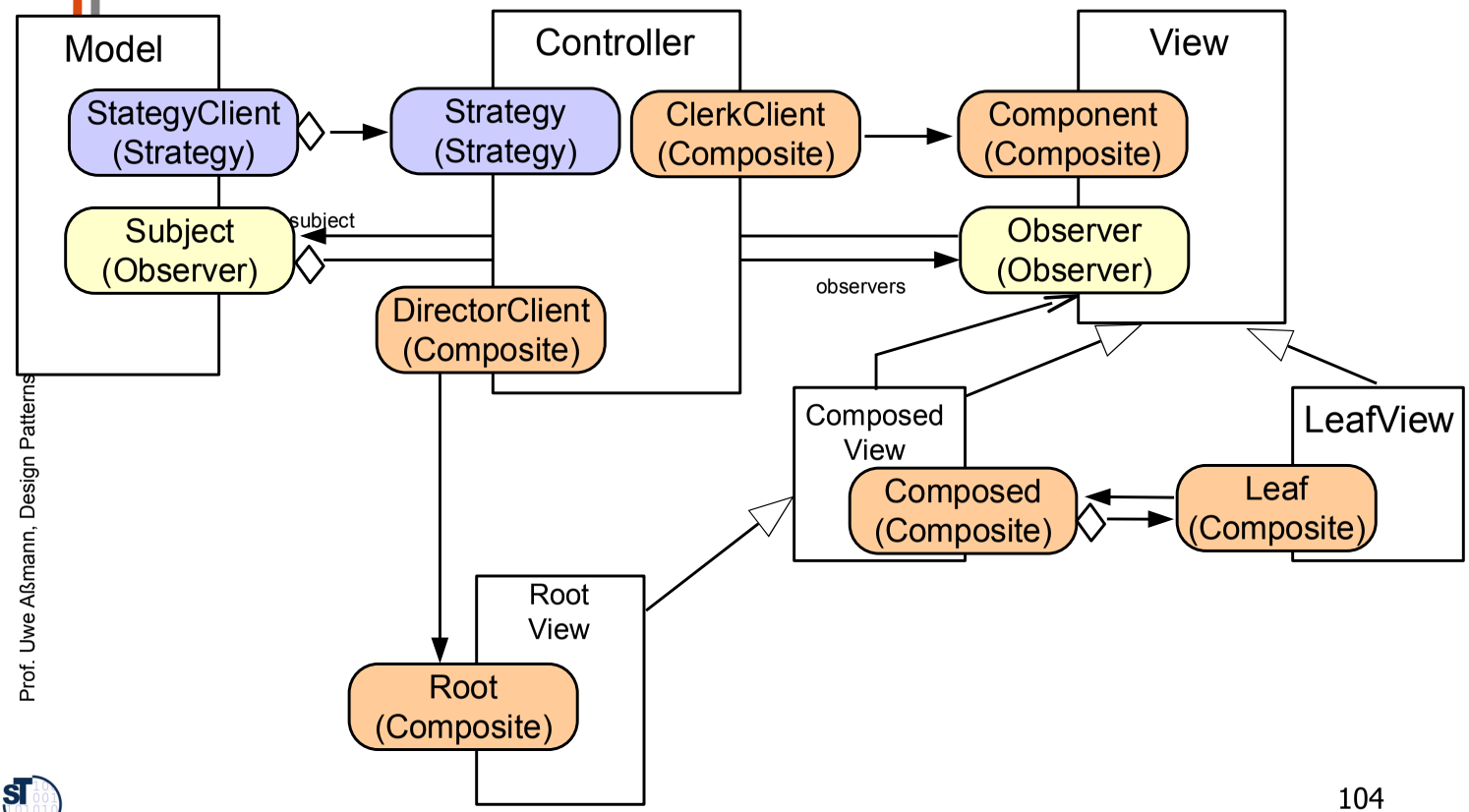
10.8.2 Optimization of Design Patterns with Role Models

Law of Optimization for Design Patterns

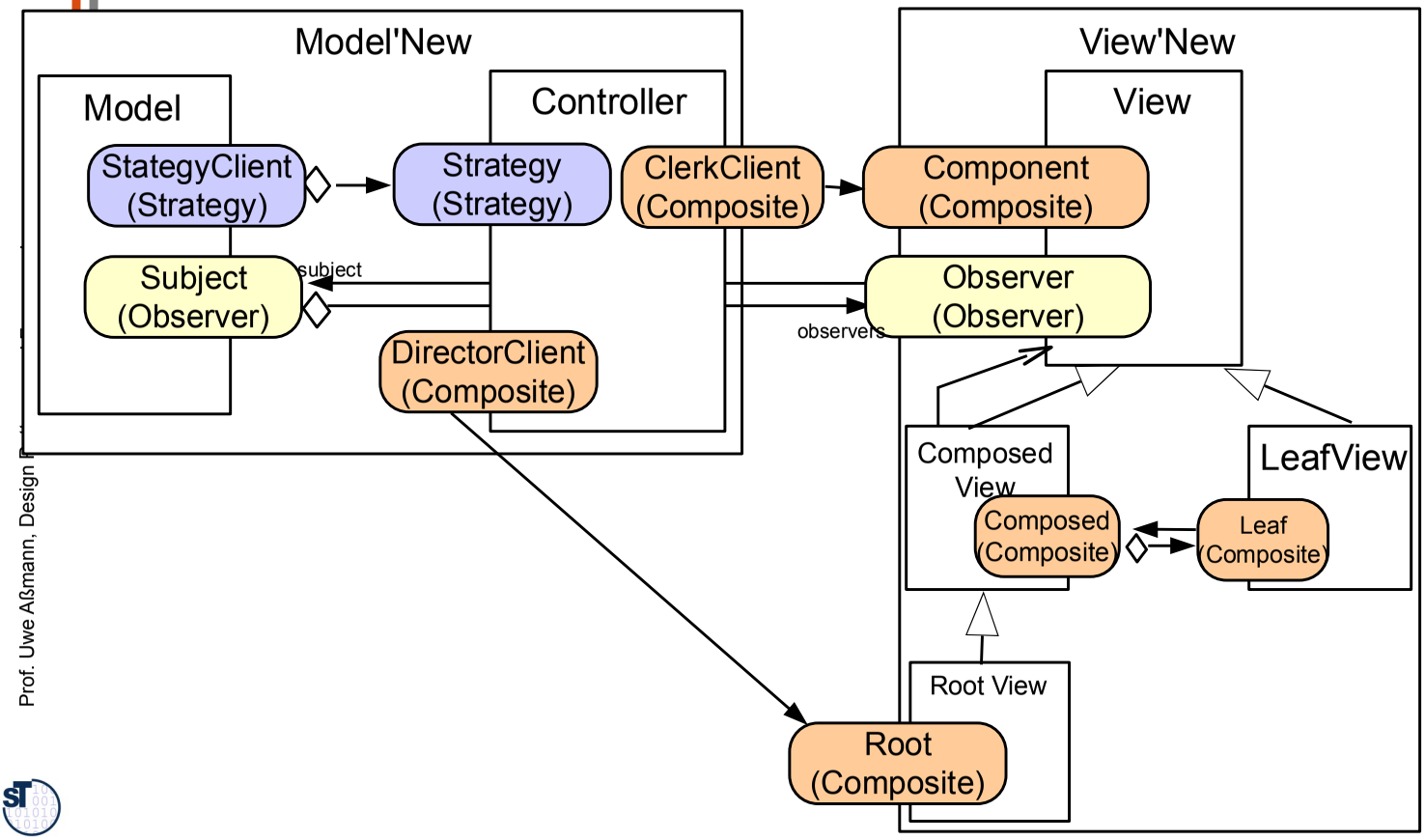
Whenever you need a variant of a design pattern that is more efficient, investigate its role model and try to merge the classes of the roles

- Effect:
  - Less variability
  - Less runtime objects
  - Less delegations

Original Role-Class Model of MVC



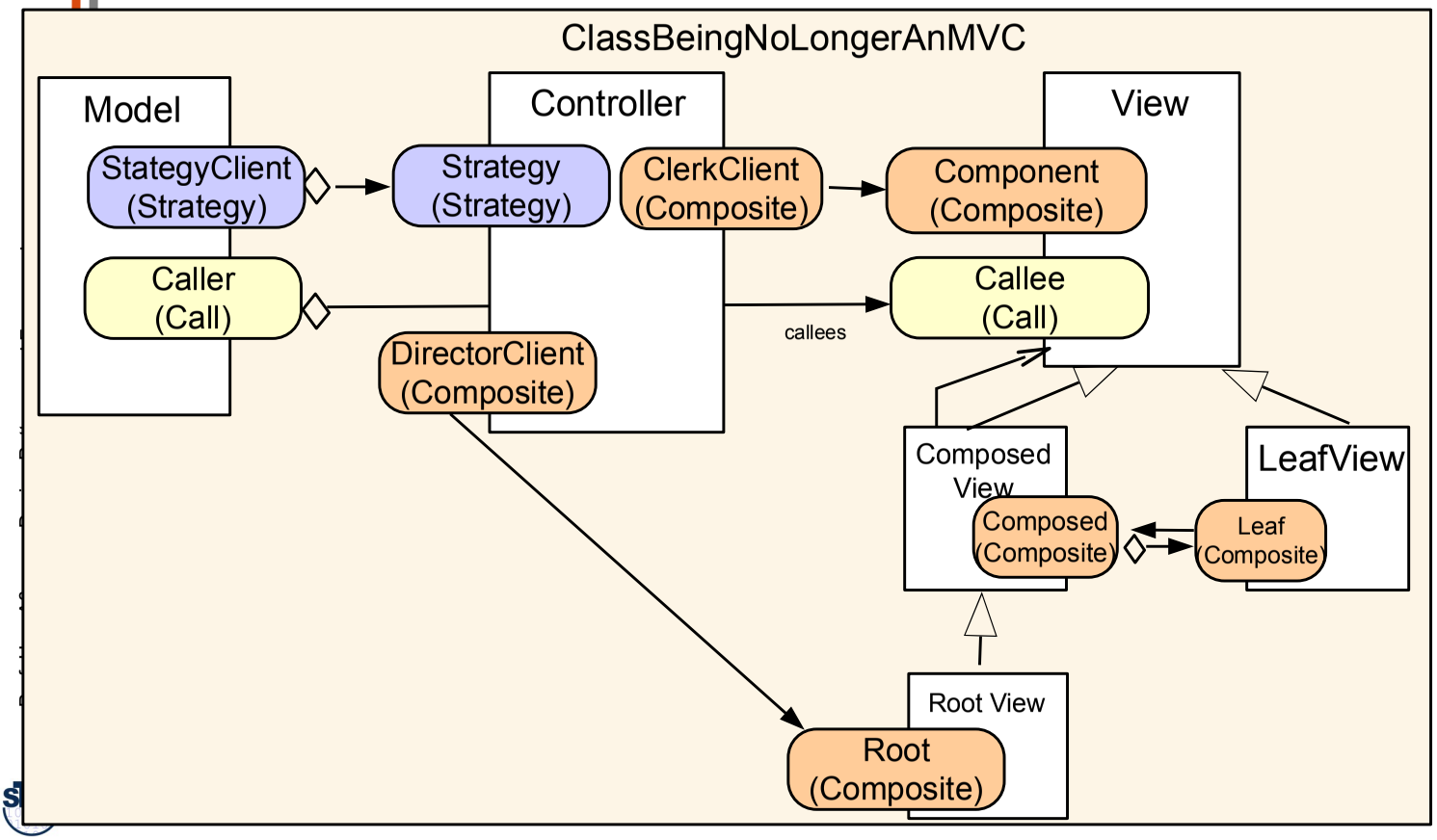
Optimized Role-Class Model of MVC



Optimized Role-Class Model of MVC

- The optimized model merges all roles into two classes
  - No strategy variation
  - No composite views
- Only 2 instead of 3+n objects at runtime
  - Faster construction
  - Essence of the pattern, the Observer, is still maintained
- However, restricted variability

Super-Optimized Role-Class Model of MVC



- In this design, the ClassBeingNoLongerAnMVC merges all roles
  - It should be a superclass of all contained classes
- The Observer pattern is exchanged to a standard call
- No variability anymore
- But only one runtime object!

The End: Summary

- 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
- 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)
- Composite design patterns are based on role model composition
- Layered frameworks and design patterns can be optimized by role merging