11. Design Patterns as Role Models

Prof. Dr. U. Aßmann

Chair for Software Engineering

Faculty of Informatics

Dresden University of Technology

Version 13-1.1 12/2/13

1) Design Patterns as Role Models

2) Composition of Design Patterns with Role Models

3) Effects of Role Modeling in Frameworks

4) Optimization of Design Patterns

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

▲ Dirk Riehle. Bureaucracy. In Robert Martin, Dirk Riehle, and Frank Buschmann, editors, Pattern Languages of Program Design 3, pages 163-185. Addison Wesley, 1998.
  ▪ http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.2034

Other Literature

- ▲ Walter Zimmer. Relationships Between Design Patterns. Pattern Languages of Program Design 1 (PLOP), Addison-Wesley 1994

- ▲ 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 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

11.1 Design Patterns as Role Diagrams

... more info...

Design Patterns have Role Models
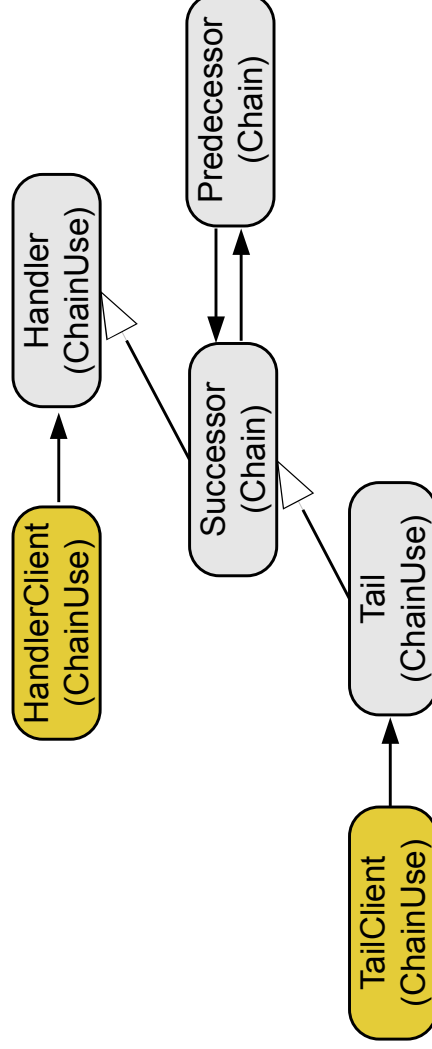
▲ Observer role model

Observer
(FigureObserver)

0..*

0..*

Subject
(FigureObserver)

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)
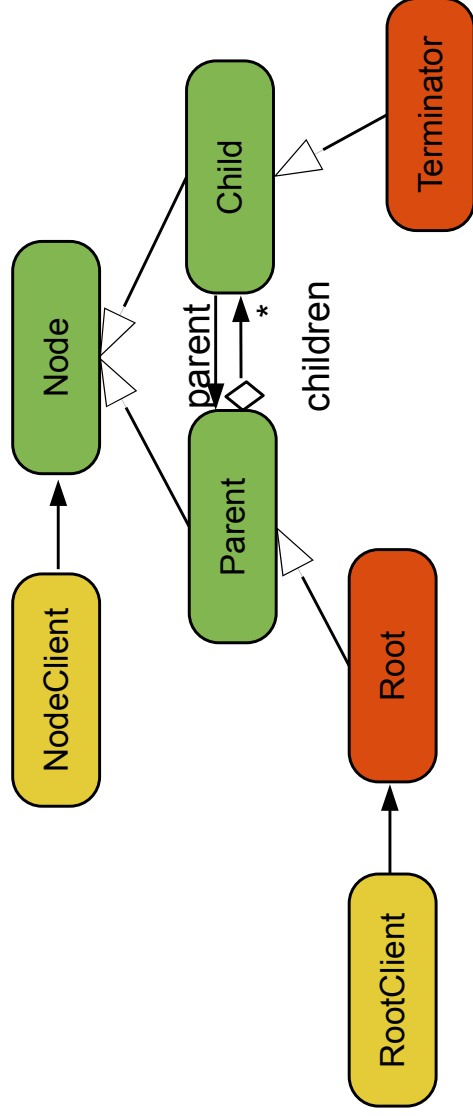
– ChainUse: (Handler, HandlerClient, Tail, TailClient)

Handler
(ChainUse)

Predecessor
(Chain)

HandlerClient
(ChainUse)

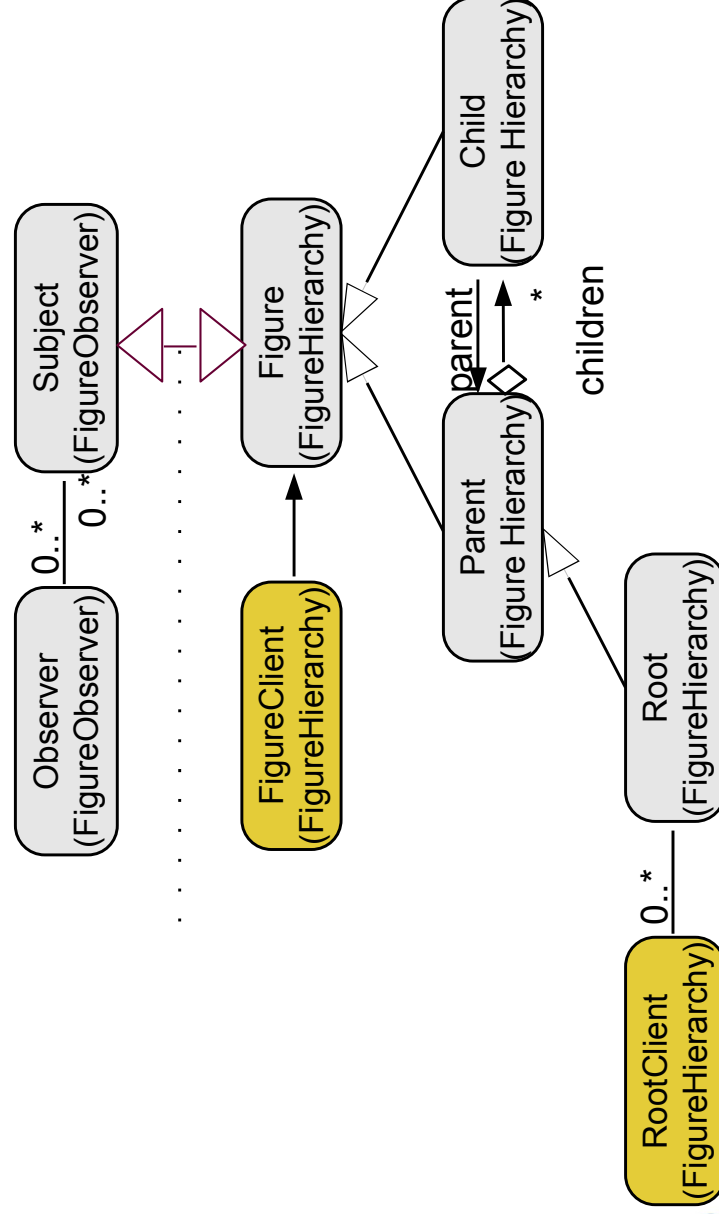Successor
(Chain)

Tail
(ChainUse)

TailClient
(ChainUse)

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
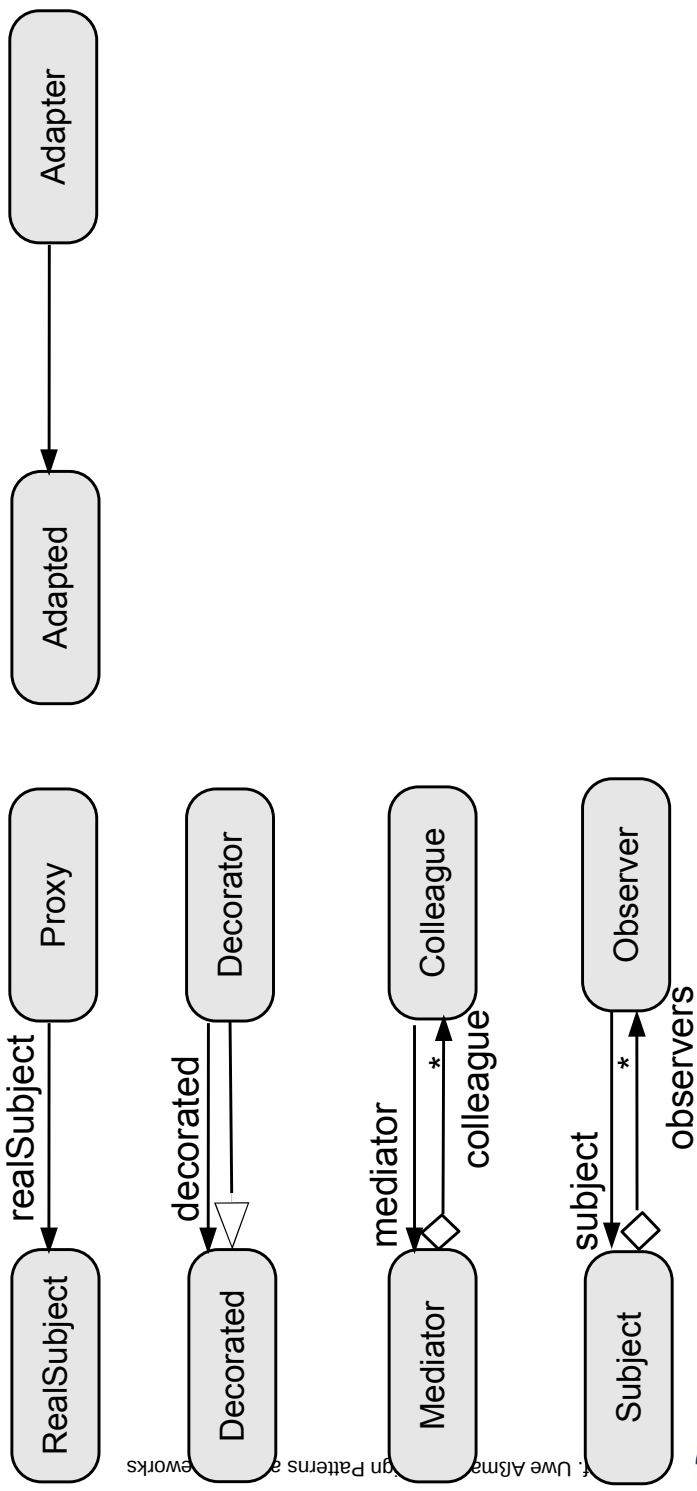

Composing (Overlaying) Role Models

▲ Overlaying the FigureHierarchy with the FigureObserver role model by role biimplication

Core Role Diagrams of Several Patterns

▲ Many of them are quite similar

Adapter → Adapted

Proxy — realSubject → RealSubject

Decorator — decorated → Decorated

Mediator — mediator / colleague * → Colleague

Subject — subject / observers * → Observer

What does Role-Type Merging Mean?

▲ Merging of attribute set
▲ Merging of method set

11.2 Composite Design Patterns with Role Model Composition

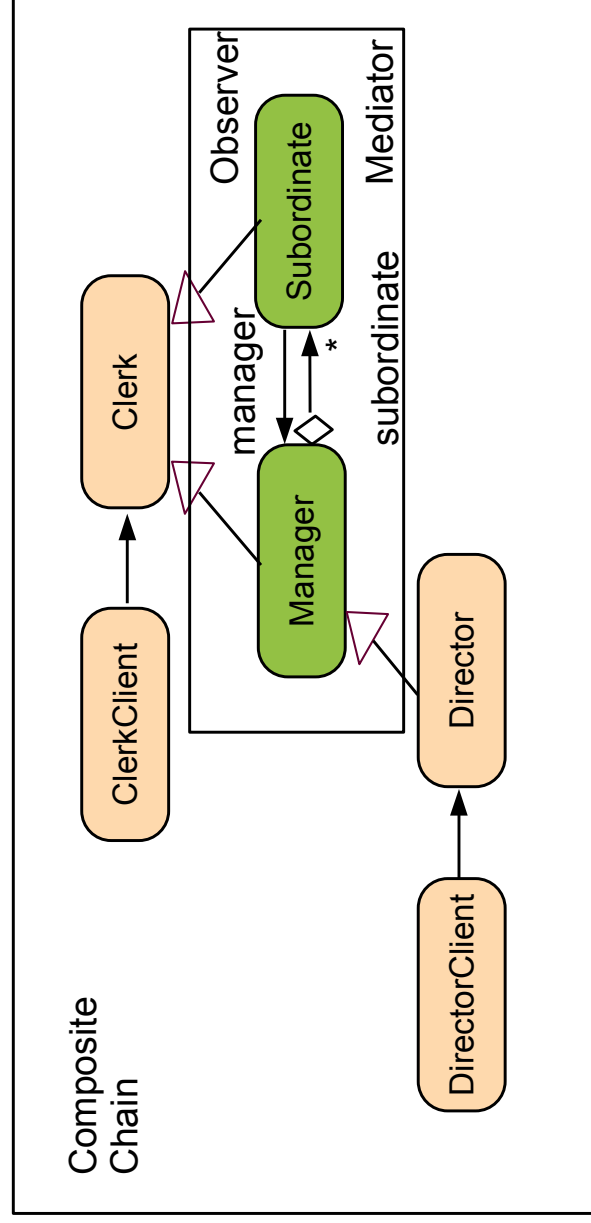.. how to create bigger design patterns as composed role models..

Design Patterns and Frameworks, © Prof. Uwe Aßmann

11.2.1 Example: Bureaucracy

▲ A pattern to model organizations that have a tree-like structure (as opposed to matrix organizations)

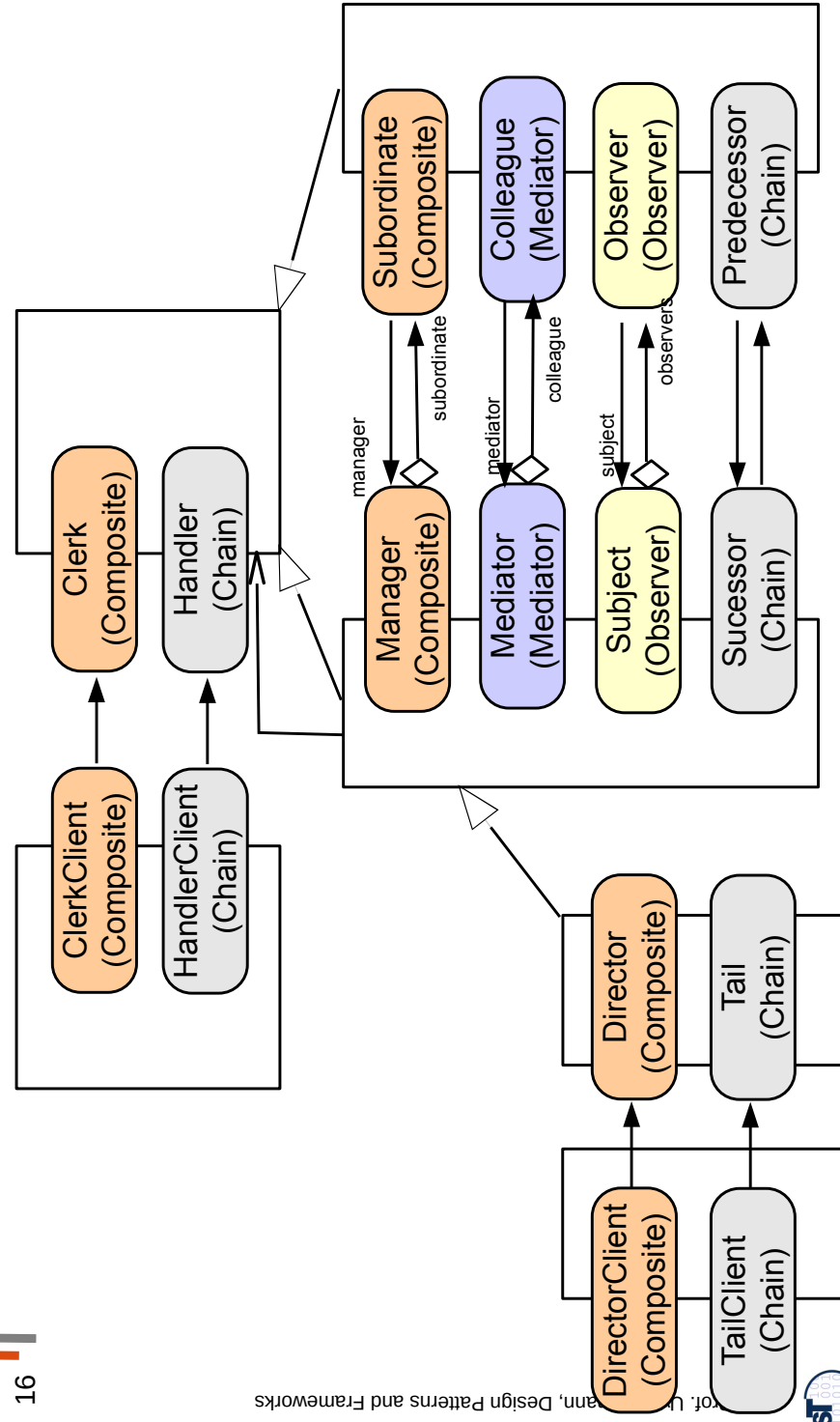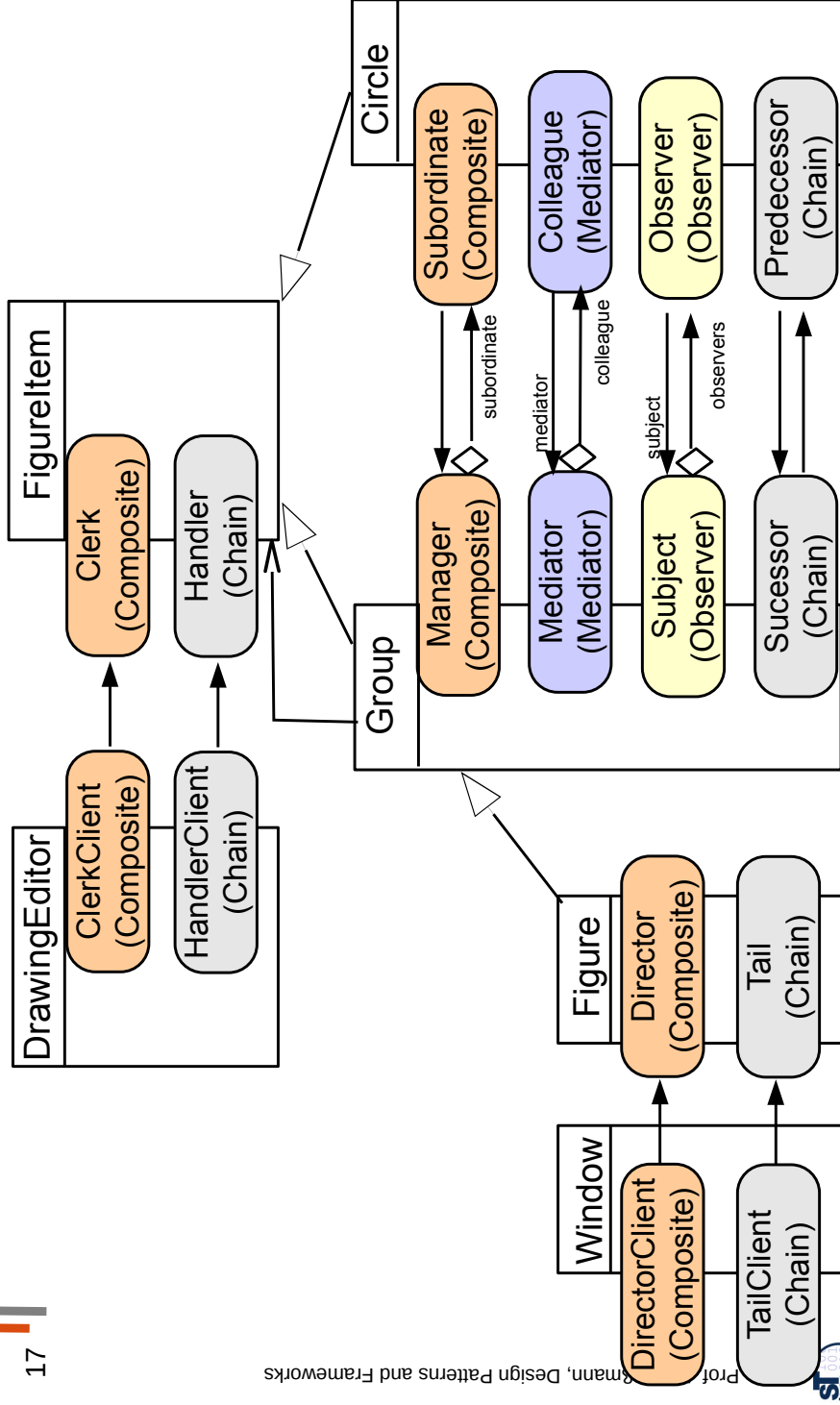– composed of the role models of Composite, Mediator, Chain, Observer

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

Prof. Uwe Aßmann, Design Patterns and Frameworks

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

Prof. Uwe Aßmann, Design Patterns and Frameworks

11.2.2 Model-View-Controller (MVC)

Design Patterns and Frameworks, © Prof. Uwe Aßmann

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

Prof. Uwe Aßmann, Design Patterns and Frameworks

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)

11.2.3 Composition of Simple Variability Patterns

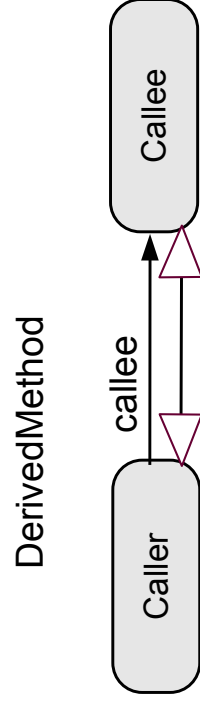Design Patterns and Frameworks, © Prof. Uwe Aßmann

Warning

▲ The following is an attempt to build up the basic GOF patterns from simple role models

▲ The compositions of patterns depend on the concrete form of their role models

▲ It explains why Strategy is different from Bridge and TemplateClass, etc.
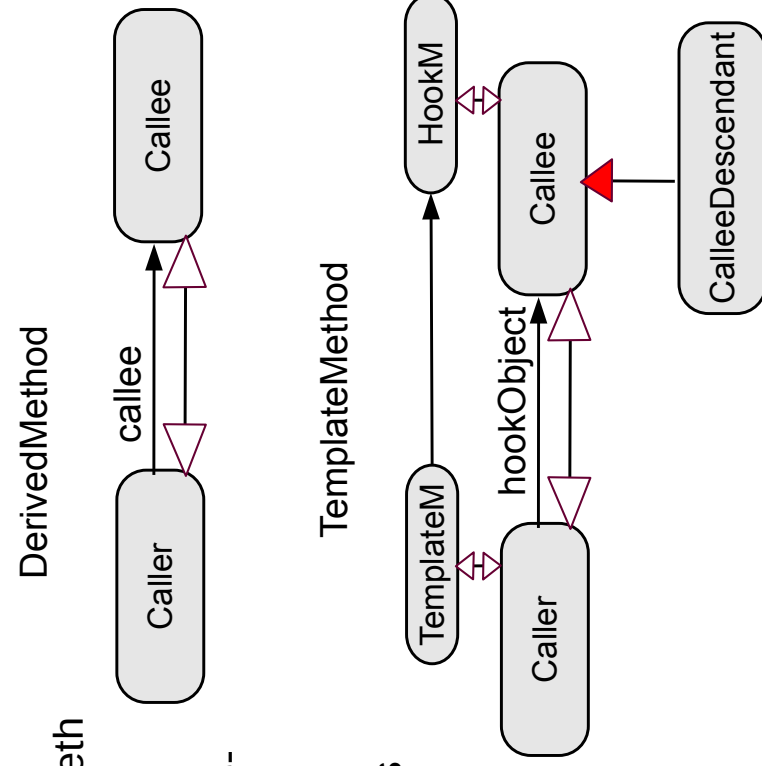
Derived Method

▲ 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

DerivedMethod

Derived Method and TemplateMethod

▲ TemplateMethod is a DerivedMethod that has

 – an additional TemplateMethod/HookMeth od role model

 – Inheritance hierarchy on right side (implied by role-class inheritance constraint)

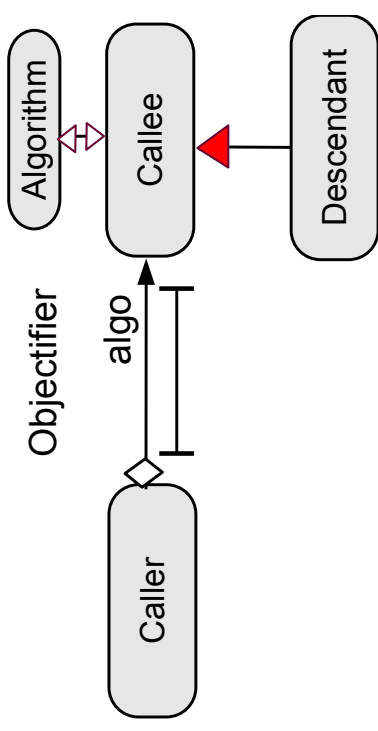 – The template role implies no hierarchy on left side

DerivedMethod

TemplateMethod

Objectifier and Strategy

Objectifier
- algo
- Algorithm
- Callee
- Descendant
- Caller

Strategy
- algo
- Algorithm
- Callee
- Descendant
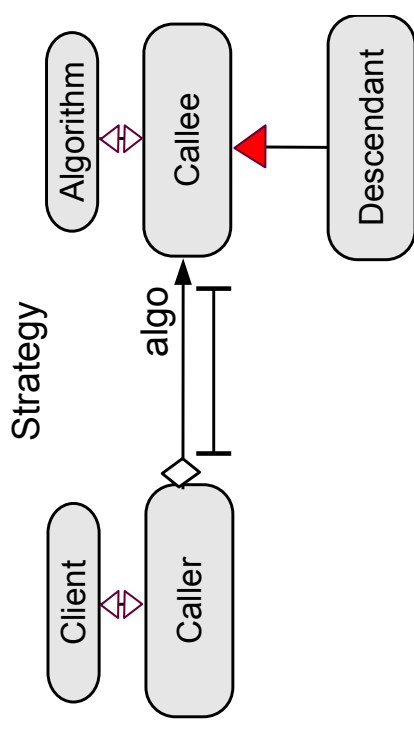- Client
- Caller

- ▸ 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
- HookM
- Called
- CalleeDescendant
- TemplateM
- Caller
- hookObject
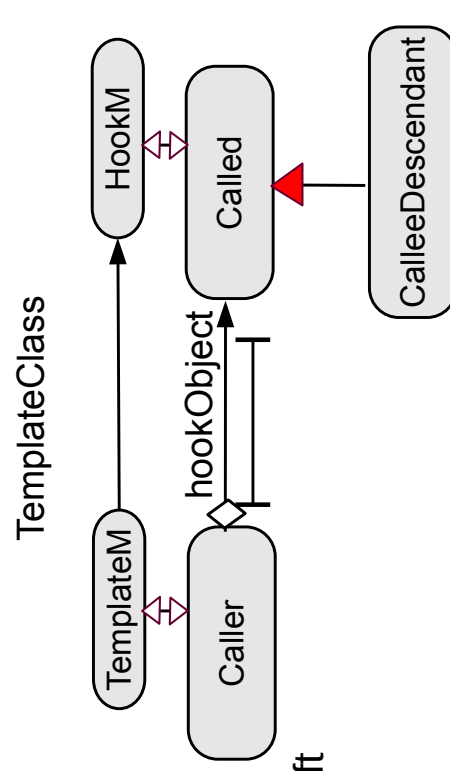
- ▸ TemplateClass is an Objectifier with
  - – An additional TemplateMethod/HookMethod role model
  - – TemplateMethod role implies no hierarchy on left side
  - – HookMethod role implies inheritance hierarchy on right side
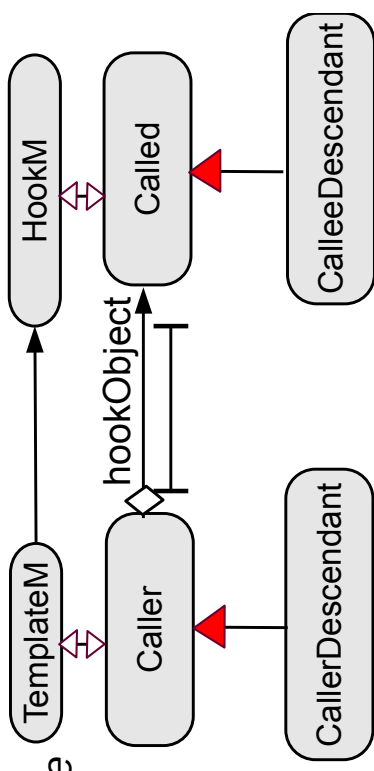 - – *No client or algorithm role, otherwise like Strategy*

DimensionalHierarchies

▲ DimensionalClassHierarchies is a TemplateClass

– *Without template-hook constraint, but still TemplateMethod/Template Hook constraint*
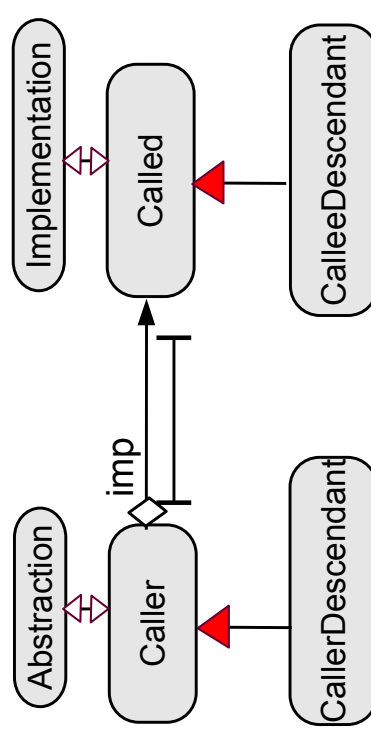
– With left hierarchy constraint

TemplateM ⇨ HookM

Caller

Called

hookObject

CallerDescendant

CalleeDescendant

Bridge

▲ Bridge is a DimensionalHierarchies with

– An additional abstraction/implementation role model

– *No template/hook role*

Abstraction ⇨ Caller

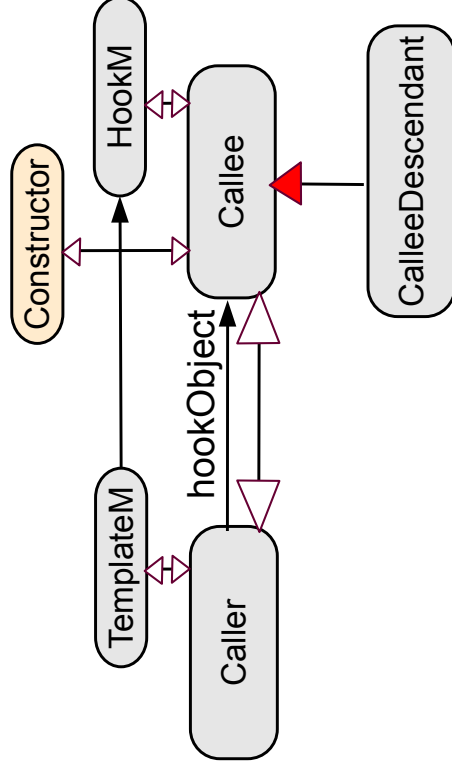Implementation ⇨ Called

imp

CallerDescendant
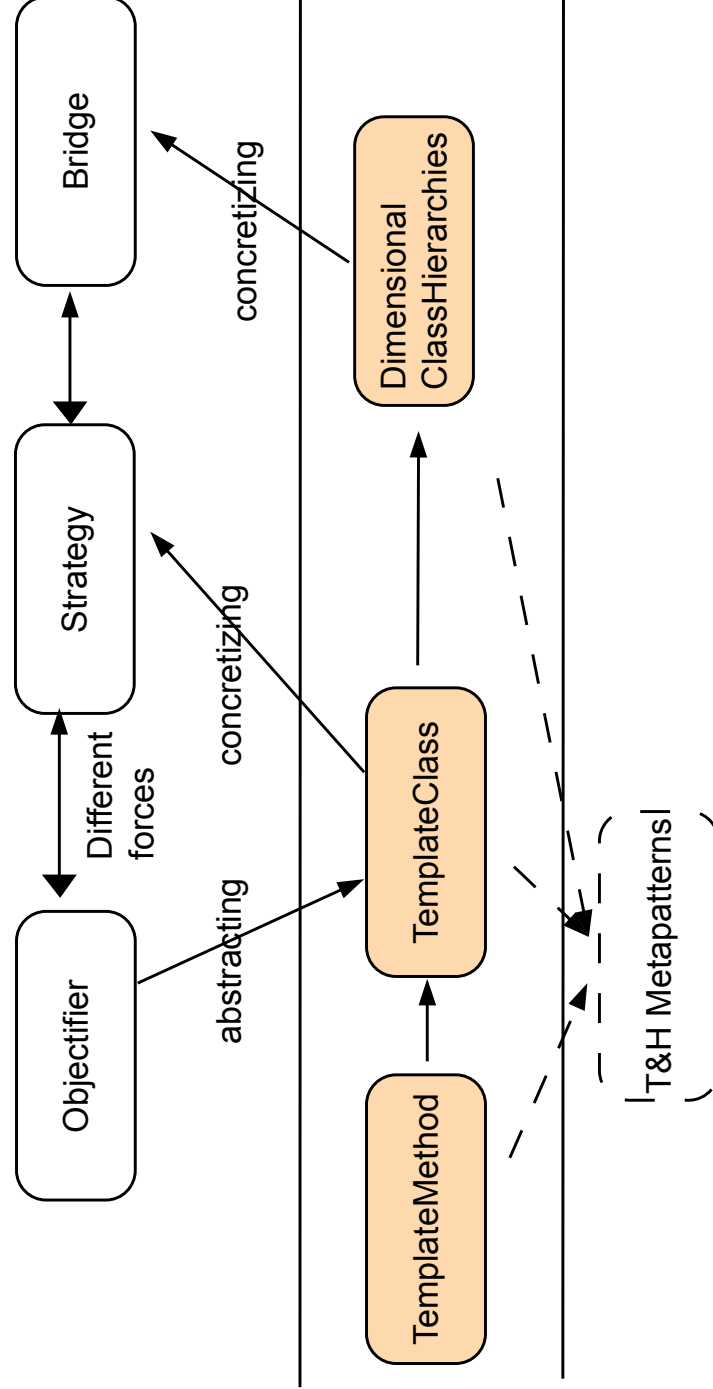
CalleeDescendant

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)

11.2.4 Composition of Simple Extensibility Patterns
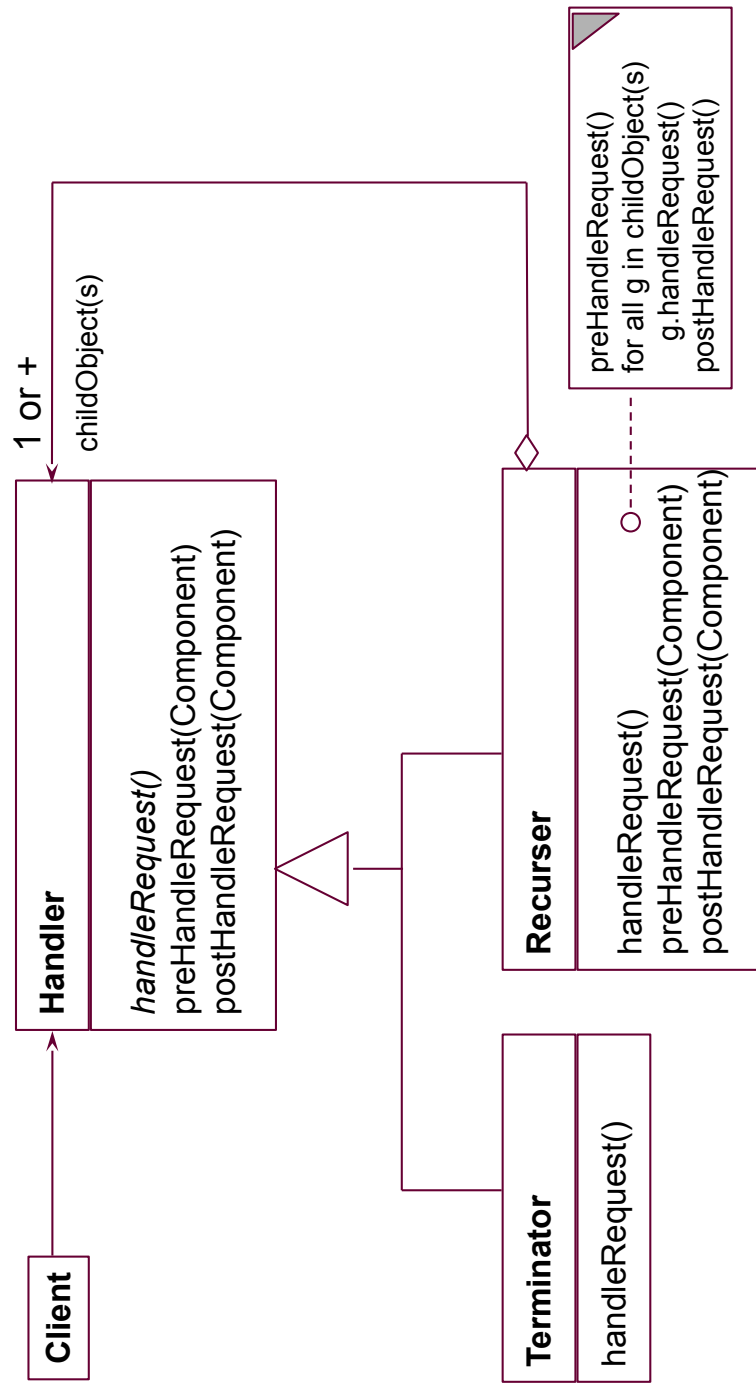
Design Patterns and Frameworks, © Prof. Uwe Aßmann

Object Recursion

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

Client

Handler

handleRequest()
preHandleRequest(Component)
postHandleRequest(Component)

1 or +   childObject(s)

Recurser

handleRequest()
preHandleRequest(Component)
postHandleRequest(Component)

preHandleRequest()
for all g in childObject(s)
    g.handleRequest()
postHandleRequest()

Terminator

handleRequest()

ObjectRecursion

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

RootClient

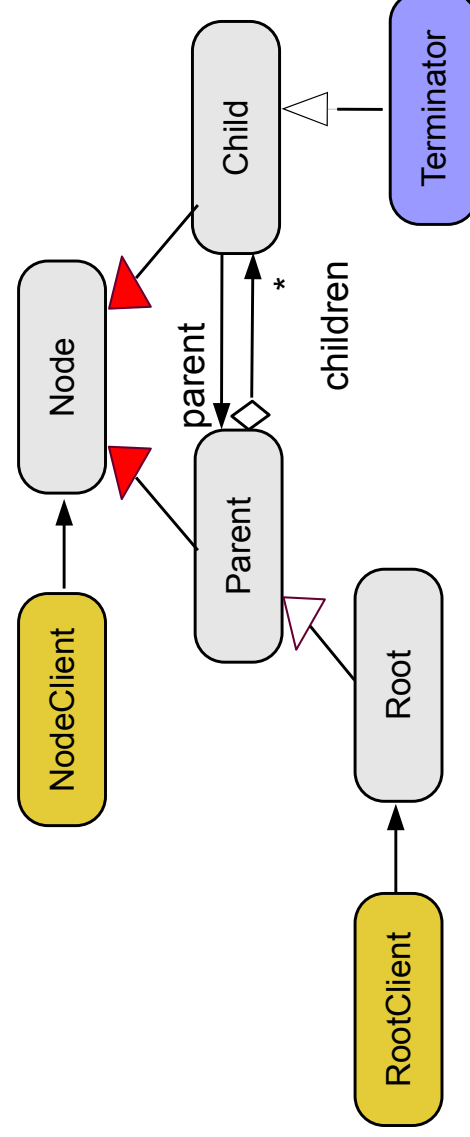NodeClient

Handler

Recurser

Root

Child

Terminator

parent

children { 1 or * }

Composite

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

RootClient

NodeClient

Node

Parent

Root

Child

Terminator

parent

children *

Decorator

- ▲ 1-ObjectRecursion
- ▲ other role pragmatics, similar pattern

NodeClient

Node

Decorated

Mimiced

Decorator

hidden

1

RootOfList

RootClient

Chain of Responsibility

- ▲ No real ObjectRecursion

Handler
(ChainUse)

Predecessor
(Chain)

HandlerClient
(ChainUse)

Sucessor
(Chain)

Tail
(ChainUse)

TailClient
(ChainUse)

Remember:
Relations Extensibility Patterns

Specific Patterns

Visitor

n-Brigde

Observer

Bridge

Proxy

Chain

Decorator

Composite

ObjectRecursion

- - - - - - abstracting

Dimensional ClassHierarchies

Connection T&H Pattern

Still something to discover...

Framework Patterns

- - - - - - abstracting

Recursive T&H Pattern

11.2.5 Consequences of the Riehle/Gross Law

Zimmer's Classification and the Riehle-Gross Law

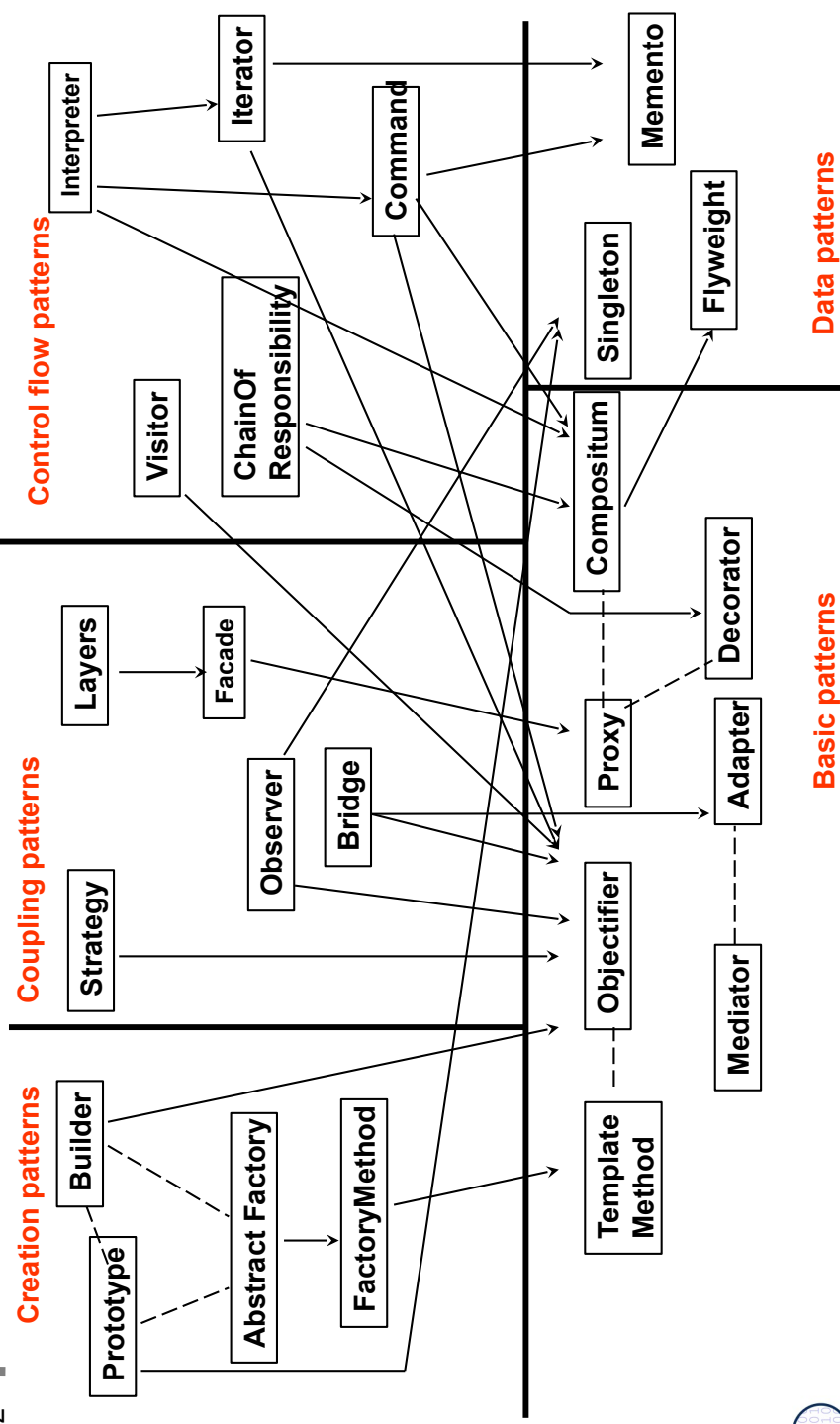▲ Zimmer's hierarchy [Zimmer, PLOP 1] lists use-relationships between design patterns

– But actually, he means composition of role models of design patterns

– but Zimmer could not express it conceptually

Relations between Patterns [Zimmer, PLOP 1]

Creation patterns

Builder
Prototype
Abstract Factory
FactoryMethod
Template Method

Coupling patterns

Strategy
Layers
Observer
Facade
Bridge
Objectifier

Control flow patterns

Interpreter
Iterator
Visitor
Command
ChainOf Responsibility

Data patterns

Memento
Singleton
Flyweight

Basic patterns

Compositum
Proxy
Decorator
Adapter
Mediator

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

(Strategy) =!= (TemplateClass)

Consequence for Pattern Mining

▲ When you identify a pattern in the product of your company, use **pattern decomposition** and **composition**

– 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

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

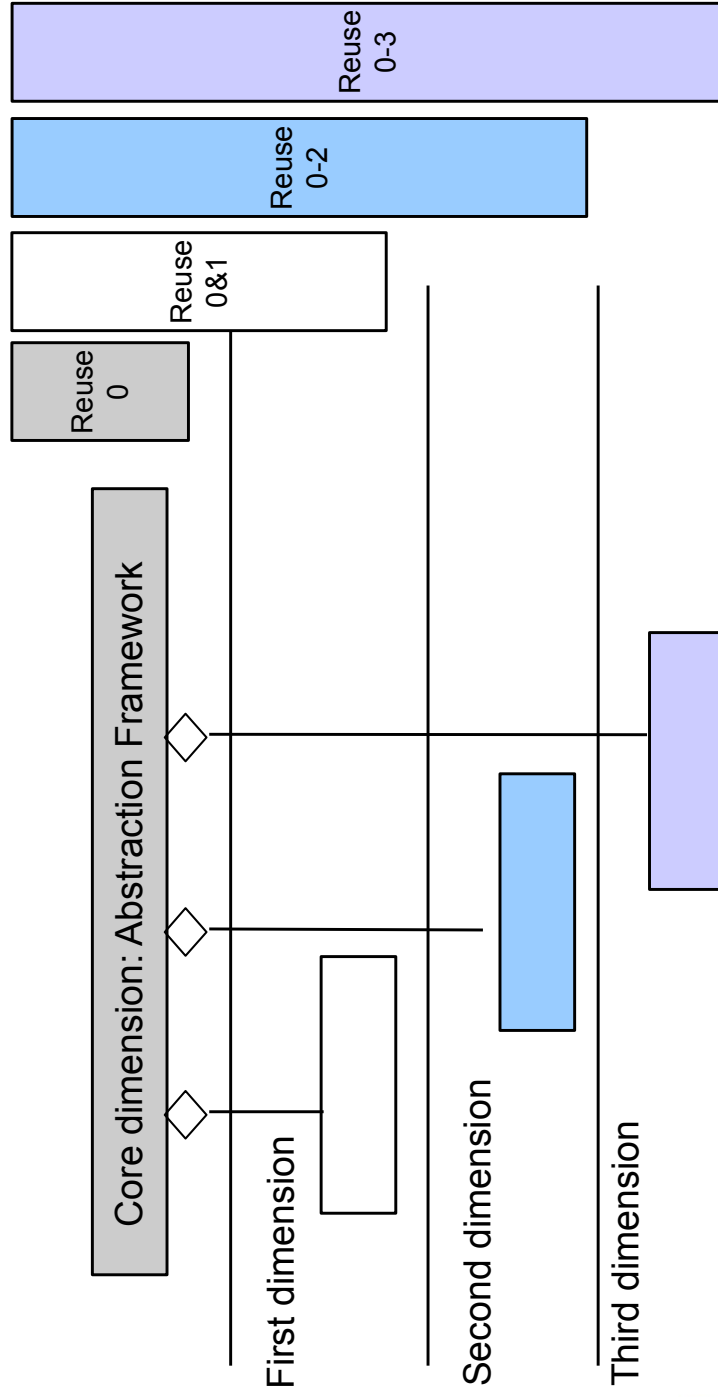Design Patterns and Frameworks, © Prof. Uwe Aßmann

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

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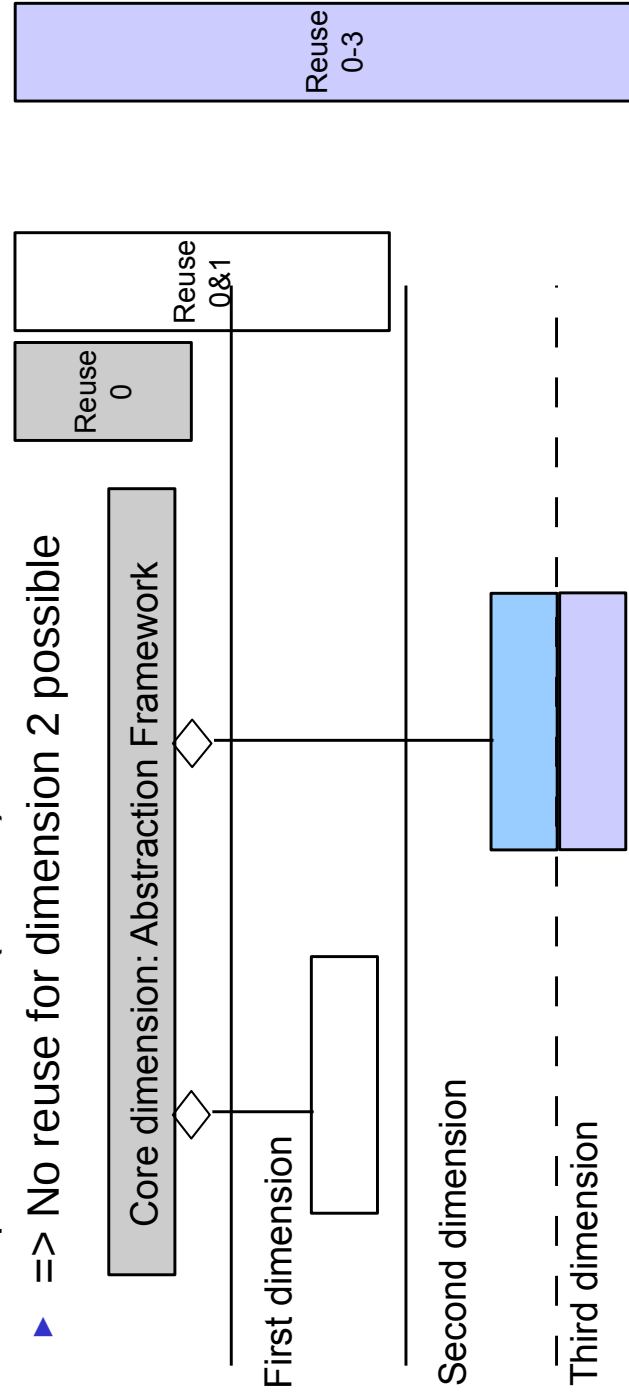- Similar for Chain-Bridges and layered frameworks

Reuse 0-3

Reuse 0-2

Reuse 0&1

Reuse 0

Core dimension: Abstraction Framework

First dimension

Second dimension

Third dimension

Merging dimensions of Facet/Layered Frameworks

- If the dimensions are seen as role models, it can be chosen to merge them, i.e., the role models
- Here: merge second and third dimension into one physical implementation (mixins)
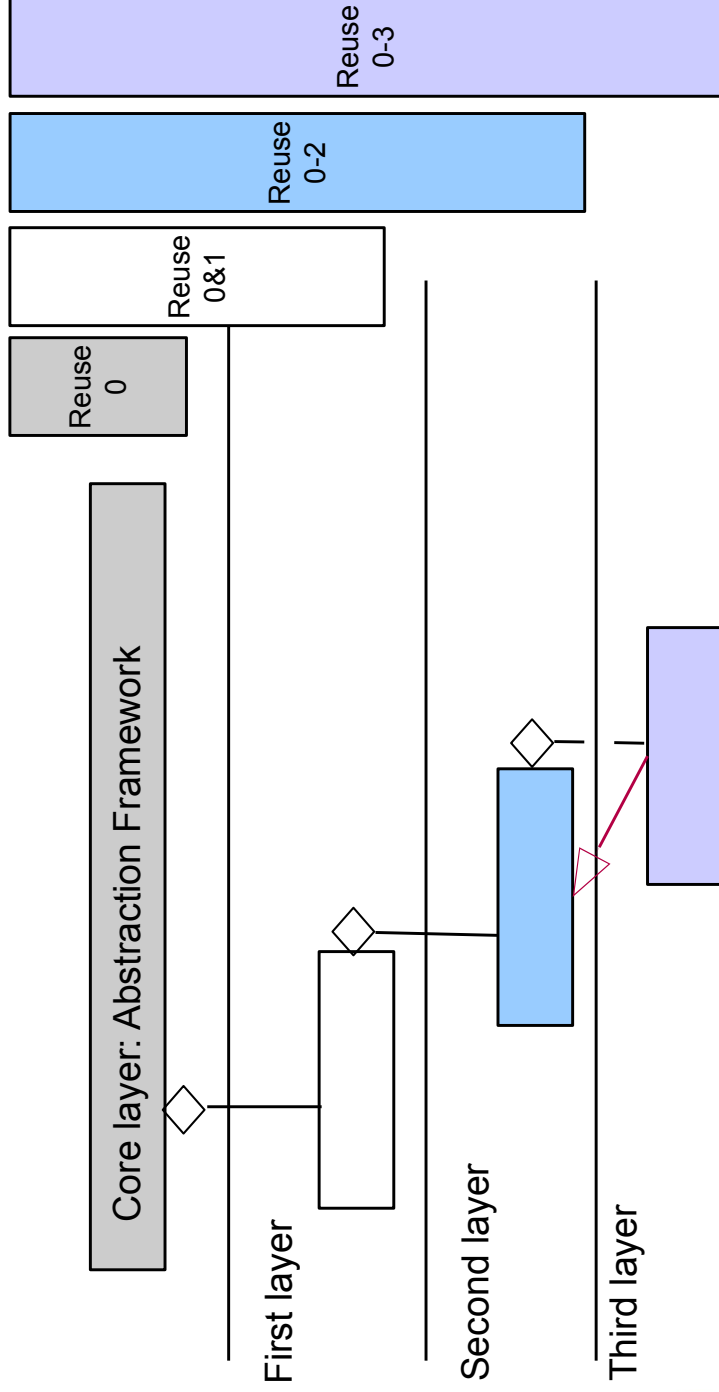- => No reuse for dimension 2 possible

Reuse 0-3

Reuse 0&1

Reuse 0

Core dimension: Abstraction Framework

First dimension

Second dimension

Third dimension

Role Models and Layered Frameworks

▲ Similar for Chain-Bridges and layered frameworks

Core layer: Abstraction Framework

Reuse 0

Reuse 0&1

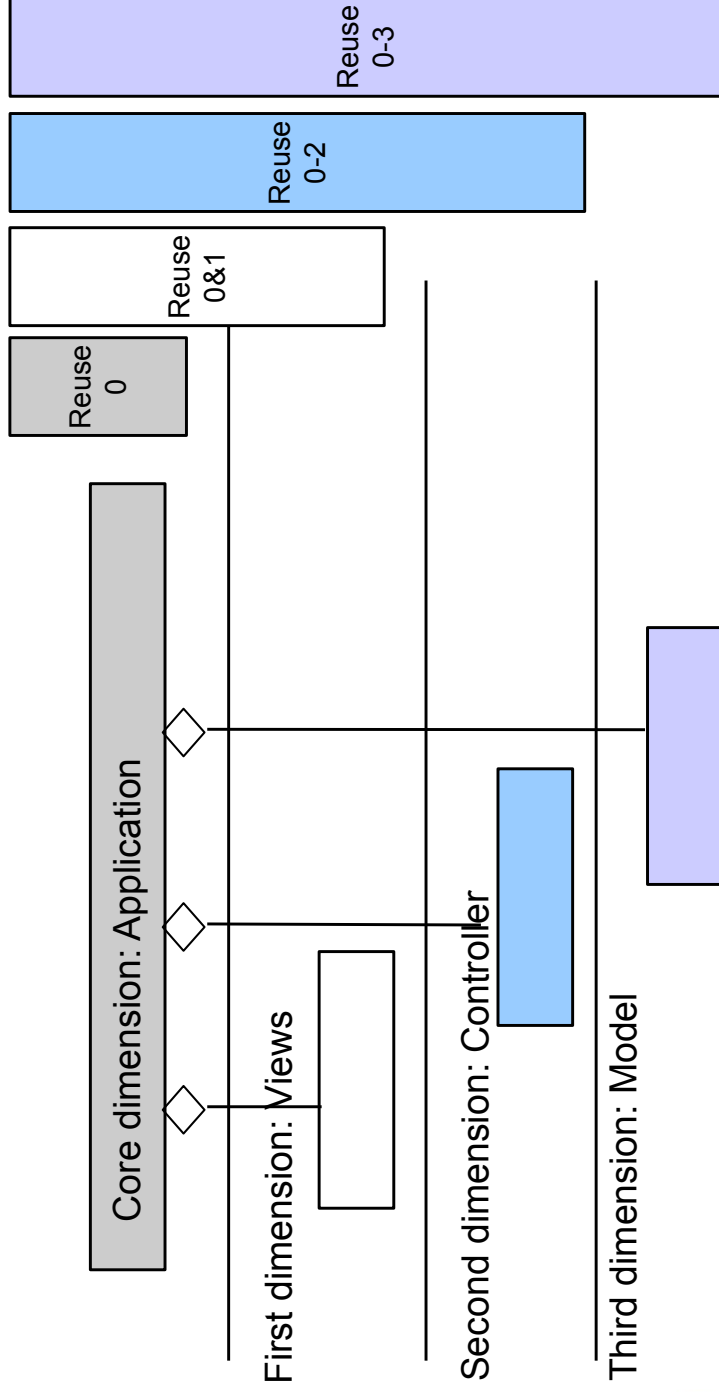Reuse 0-2

Reuse 0-3

First layer

Second layer

Third layer

Merging Dimensions/Layers of Dimensional/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)
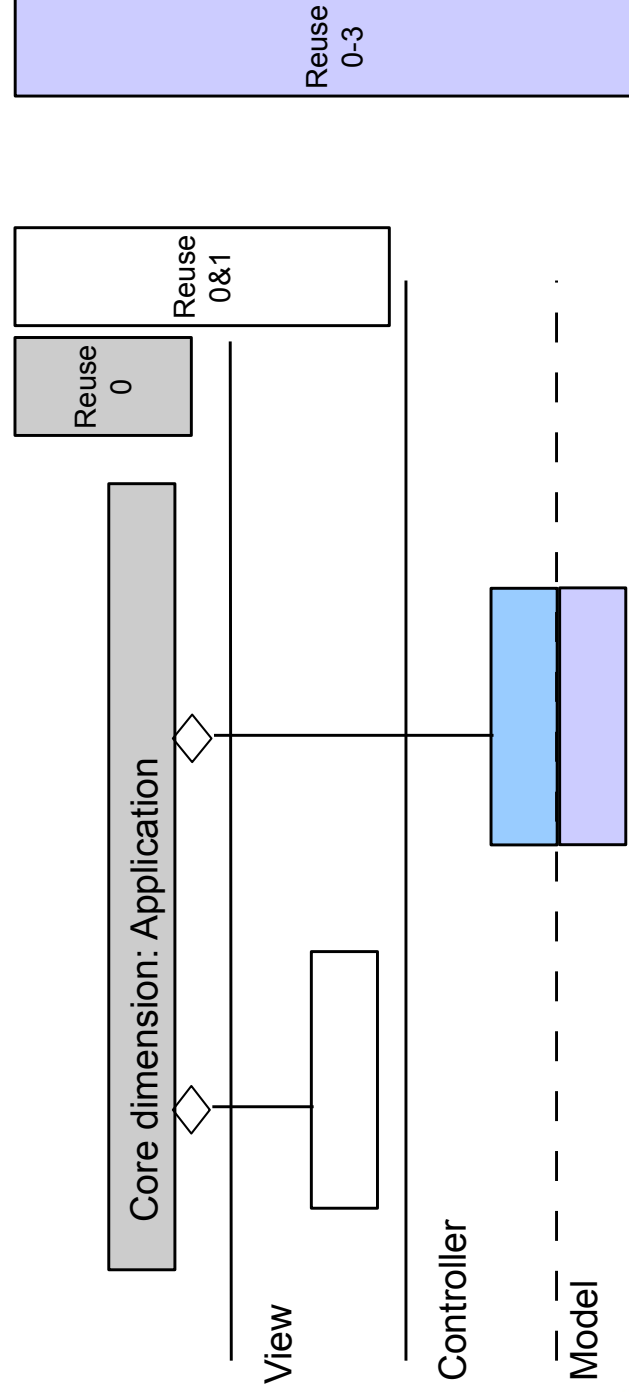- Less runtime flexibility (less dynamic variation)

MVC as Multi-Bridge Framework

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

Reuse 0-3

Reuse 0-2

Reuse 0&1

Reuse 0

Core dimension: Application

First dimension: Views

Second dimension: Controller

Third dimension: Model

MVC as Optimized Multi-Bridge Framework

▲ Model and Controller layer can be merged

▲ Less variability, but also less runtime objects

Reuse 0-3

Reuse 0&1

Reuse 0

Core dimension: Application

View

Controller

Model

11.4 Optimization of Design Patterns with Role Models
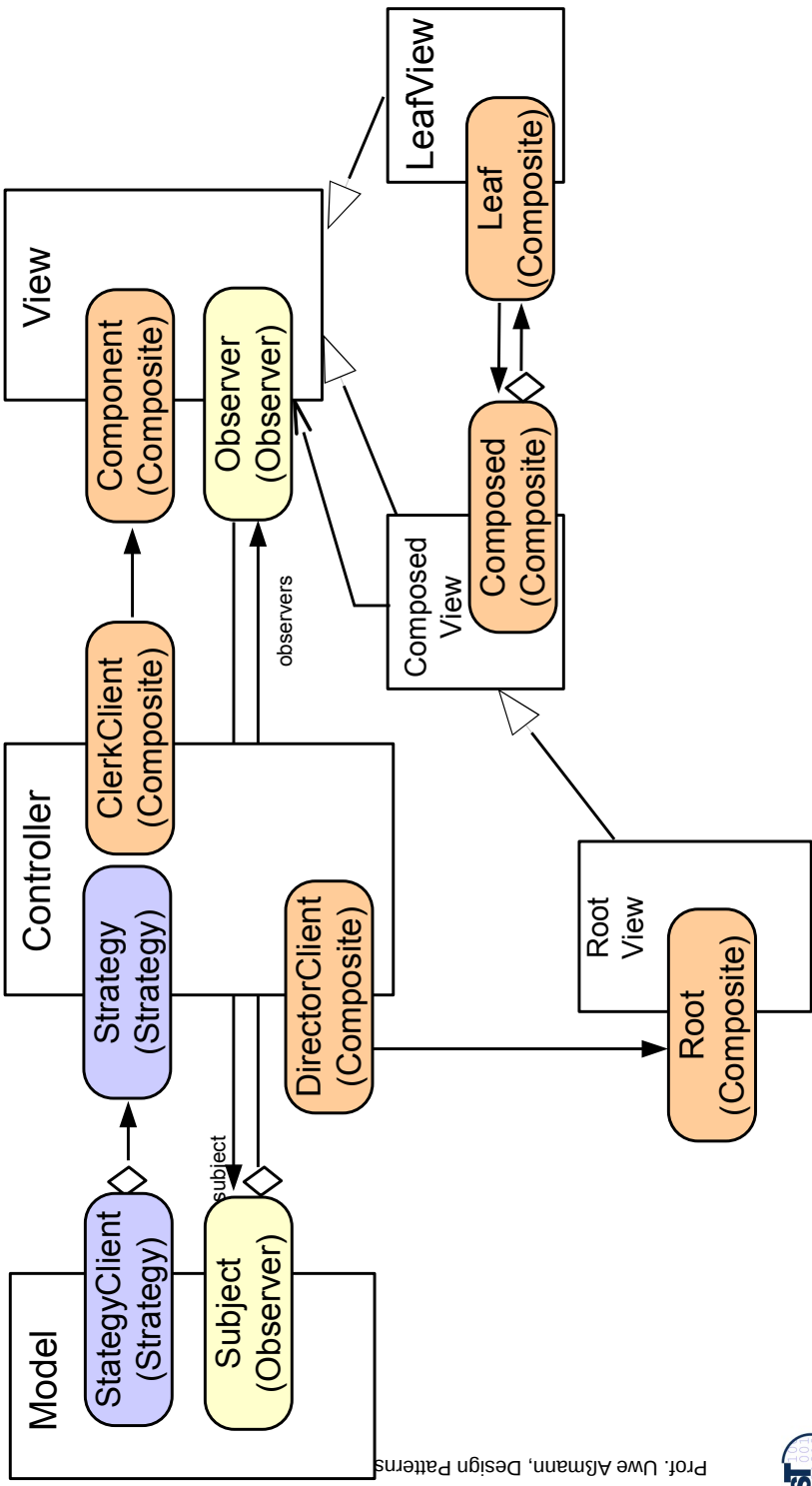

Law of Optimization for Design Patterns

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:

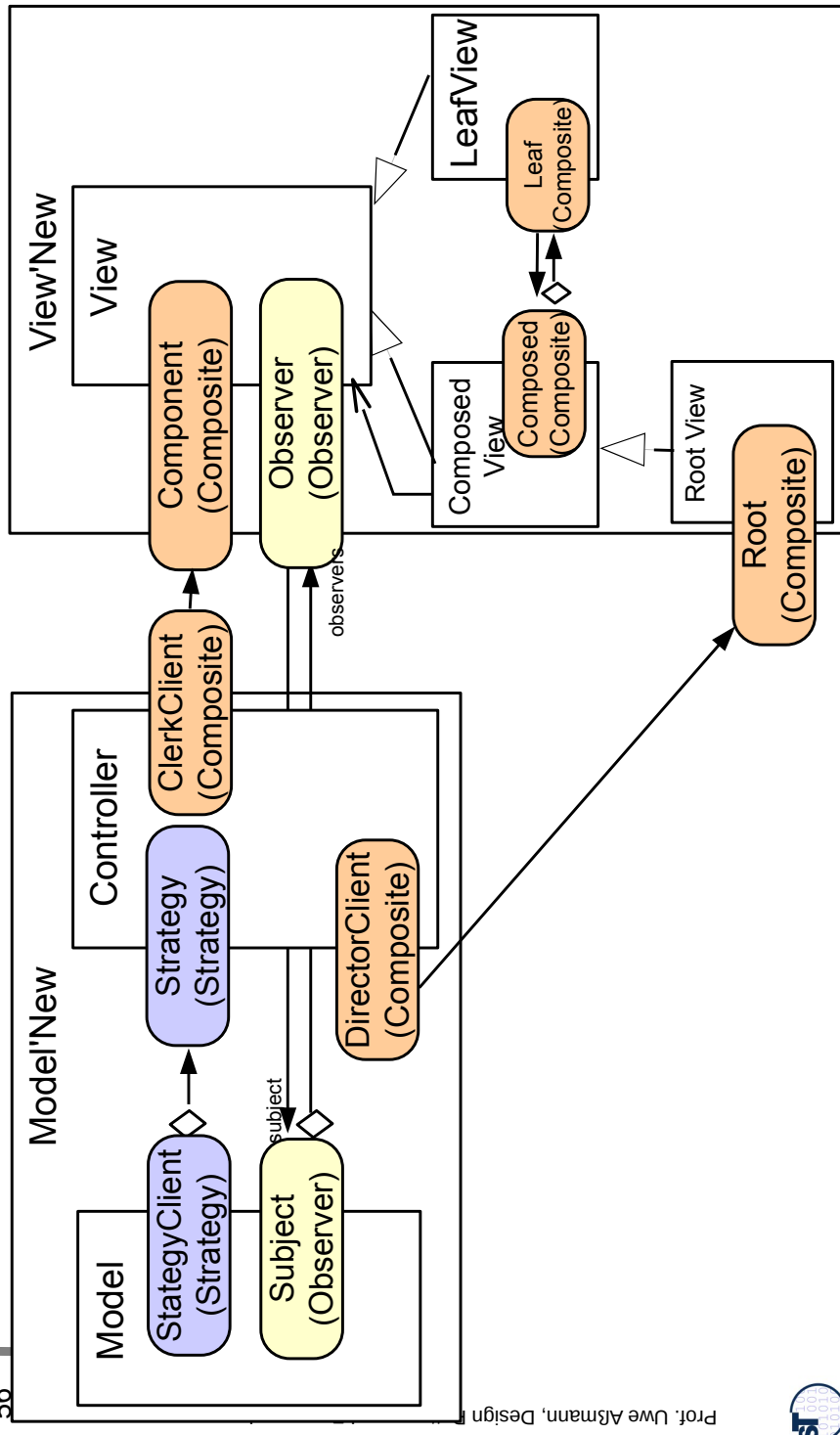– Less variability

– Less runtime objects

– Less delegations

Original Role-Class Model of MVC

Optimized Role-Class Model of MVC
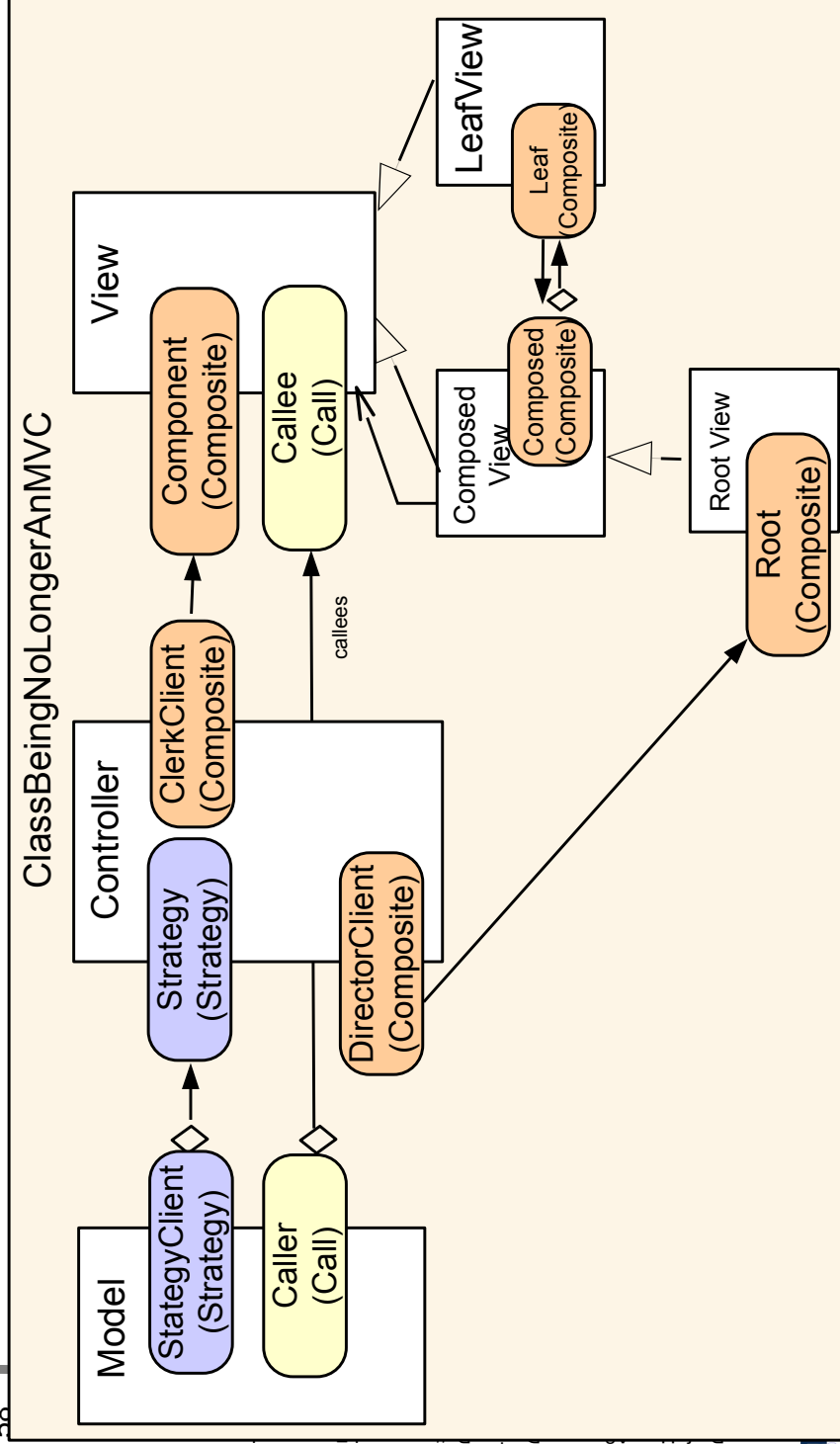
Optimized Role-Class Model of MVC

- ▲ 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 (Monolithic)

ClassBeingNoLongerAnMVC

Model — StategyClient (Strategy), Caller (Call)

Controller — Strategy (Strategy), ClerkClient (Composite), DirectorClient (Composite)

View — Component (Composite), Callee (Call), Composed View, Composed (Composite), Root View, Root (Composite)

LeafView — Leaf (Composite)

callees

- ▲ 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