

---

# Design Pattern

Universität Kassel  
FB 16 Elektrotechnik / Informatik

WS 2005 / 06

Dr.-Ing. Dipl.-Wirt. Ing.  
Diethelm Bienhaus  
Berufsakademie Nordhessen

# Aus der Vorlesungsankündigung

---

Software-Entwurf ist eine anspruchsvolle Tätigkeit und erfordert Erfahrungen. Qualitativ-hochwertige und wiederverwendbare Software zu erstellen ist schwer. Die Idee von "Design Pattern" (dt. Entwurfsmuster) ist es, Erfahrungen von Experten zu sammeln und so darzustellen, dass diese leicht auf neue Aufgaben übertragen werden können.

Die Idee stammt ursprünglich aus der Architektur und geht vor allem auf den Architekten Christopher Alexander zurück. Die Entwurfsmuster von Alexander präsentieren Lösungen für den Entwurf von Häusern und Städten.

Entwurfsmuster im Software-Engineering zeigen bewährte Lösungen für die Konstruktion von Software. Inhalt dieser Vorlesung sind Grundlagen und eine Übersicht der verschiedenen Entwurfsmuster-Ansätze.

Eine Reihe von Entwurfsmustern für die Softwarekonstruktion werden vorgestellt und es wird dargelegt, wie die jeweiligen Muster einzeln und vor allem als "Mustersprache" helfen, Software flexibler und vor allem wiederverwendbar zu erstellen.

# Übersicht

---

- Teil I: Einleitung / Motivation / Begriffsbestimmungen
- Teil II: Klassifikation und Struktur
- Teil III: Design Pattern in der OOP
- Teil IV: Design Pattern für Verteilte Systeme
- Teil V: Pattern in Java
- Teil VI: Design Pattern in HCI / Pattern Language Markup Language

---

# Klassifikation und Struktur

# Klassifikation von Design Pattern

---

- Kataloge benötigen Ordnungs- / Klassifikationsschema
  - leichteres Auffinden von bestehenden Pattern
  - Hinzufügen von neuen Pattern
- Klassifikationsmerkmale und *Ausprägungen* nach GoF
  - Aufgabe
    - *Erzeugungsmuster*
    - *Strukturmuster*
    - *Verhaltensmuster*
  - Gültigkeitsbereich
    - *klassenbasierte Muster*
    - *objektbasierte Muster*

# Merkmale „Aufgabe“

---

- **Erzeugungsmuster**
  - Lösungen für den Prozess der Objekt-Erzeugung enthalten.
  - Ziel: Entkopplung von konkreter Realisation der Generierung. Eigentliches Programm arbeitet auf abstrakteren Ebene und delegiert Erzeugung der Objekte an Erzeugungsstruktur.
- **Strukturmuster**
  - Lösungen für das Zusammenfassen von Klassen bzw. Objekte zu größeren Strukturen
  - Meist Objektkomposition zur Laufzeit
- **Verhaltensmuster**
  - Lösungen zur Organisation und Strukturierung der am Kontrollfluss beteiligten Objekte.
  - Meist Algorithmen und Delegation von Zuständigkeiten

# Merkmale „Gültigkeitsbereich“

---

- **klassenbasiertes Muster**
  - Festlegung von Strukturen zur Übersetzungszeit
  - statische Strukturen
  - Klassenvererbung
- **objektbasierte Muster**
  - Strukturen entstehen zur Laufzeit
  - dynamische Strukturen
  - Objektkomposition

# Übersicht des GoF-Katalogs

Entwurfsmuster		Aufgabe		
		Erzeugungsmuster	Strukturmuster	Verhaltensmuster
Gültigkeitsbereich	klassenbasiert	Fabrikmethode	Adapter	Interpreter Schablonenmethode
	objektbasiert	Abstrakte Fabrik Erbauer Prototyp Singleton	Adapter Brücke Dekorierer Fassade Fliegengewicht Kompositum Proxy	Befehl Beobachter Besucher Iterator Momento Strategie Vermittler Zustand Zuständigkeitskette

# ... und in Englisch mit Seitenangeben

Design Pattern		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method (107)	Adapter (139)	Interpreter (243) Template Method (325)
	Object	Abstract Factory (87) Builder (97) Prototype (117) Singleton (127) Adapter (139) Bridge (151) Composite (163) Decorator (175) Facade (185) Proxy (207)	Adapter (139) Bridge (151) Composite (163) Decorator (175) Facade (185) Proxy (207)	Chain of Responsibility (223) Command (233) Iterator (257) Mediator (273) Memento (283) Flyweight (195) Observer (293) State (305) Strategy (315) Visitor (331)

# Struktur: Grundelemente

---

- **Pattern Name**
  - Treffender, bedeutungsvoller Name dient der Kommunikation
- **Problem**
  - Problem und Kontext, in dem das Muster verwendbar ist
  - Welche Randbedingungen müssen erfüllt sein, damit das Muster anwendbar ist
- **Solution**
  - Beschreibung der Elemente (Klassen, Objekte)
  - Verantwortlichkeiten
  - Struktureller Zusammenhang (Klassendiagramm)
  - Dynamisches Zusammenwirken (Sequenz- / Kollaborationsd.)
  - Keine konkrete Implementierung, sondern abstrakte Beschreibung
- **Consequences**
  - Vor- / Nachteile, Verweise auf andere Muster

# GoF Pattern Template (1/3)

---

- **Pattern Name and Classification**
  - Good , concise name for the pattern and the pattern's type
- **Intent**
  - Short statement about what the pattern does
- **Also Known As**
  - Other names for the pattern
- **Motivation**
  - A scenario that illustrates where the pattern would be useful
- **Applicability**
  - Situations where the pattern can be used

# GoF Pattern Template (2/3)

---

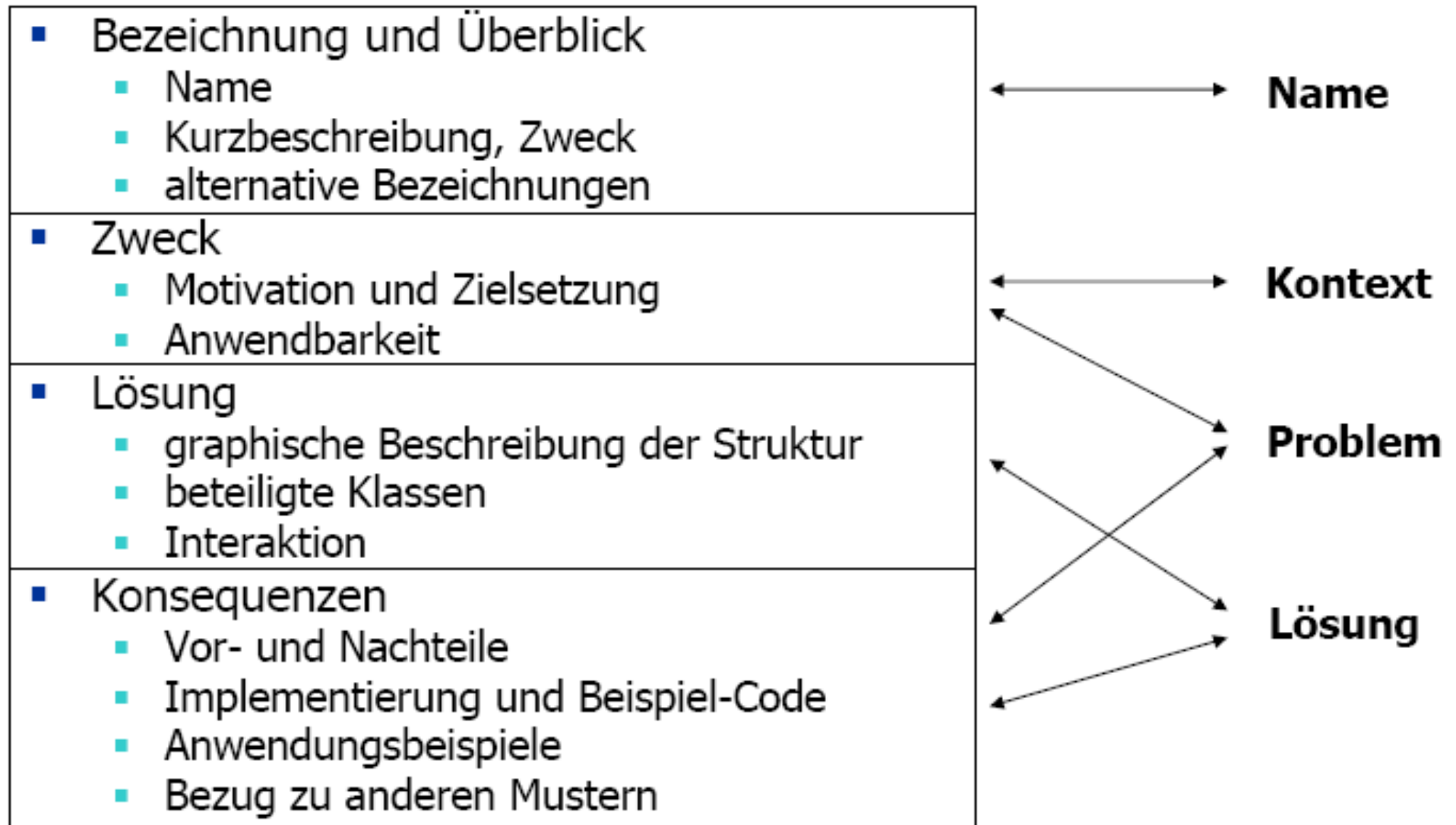
- **Structure**
  - A graphical representation of the pattern
- **Participants**
  - The classes and objects participating in the pattern
- **Collaborations**
  - How do the participants interact to carry out their responsibilities?
- **Consequences**
  - What are the pros and cons of using the pattern?
- **Implementation**
  - Hints and techniques for implementing the pattern

# GoF Pattern Template (3/3)

---

- **Sample Code**
  - Code fragments for a sample implementation
- **Known Uses**
  - Examples of the pattern in real systems
- **Related Patterns**
  - Other patterns that are closely related to the pattern

# GoF Template im Überblick



# GoF Notation

---

- Object Modeling Technique (OMT)
- Einer der Vorgänger der UML

# Beispiel eines GoF Musters

## „Composite“

---

Name: Composite

Zweck:

- Repräsentiere baumartige Hierarchien mit Hilfe von Objekten. Auf einfache und zusammengesetzte Objekte kann in der gleichen Weise zugegriffen werden.

Motivation:

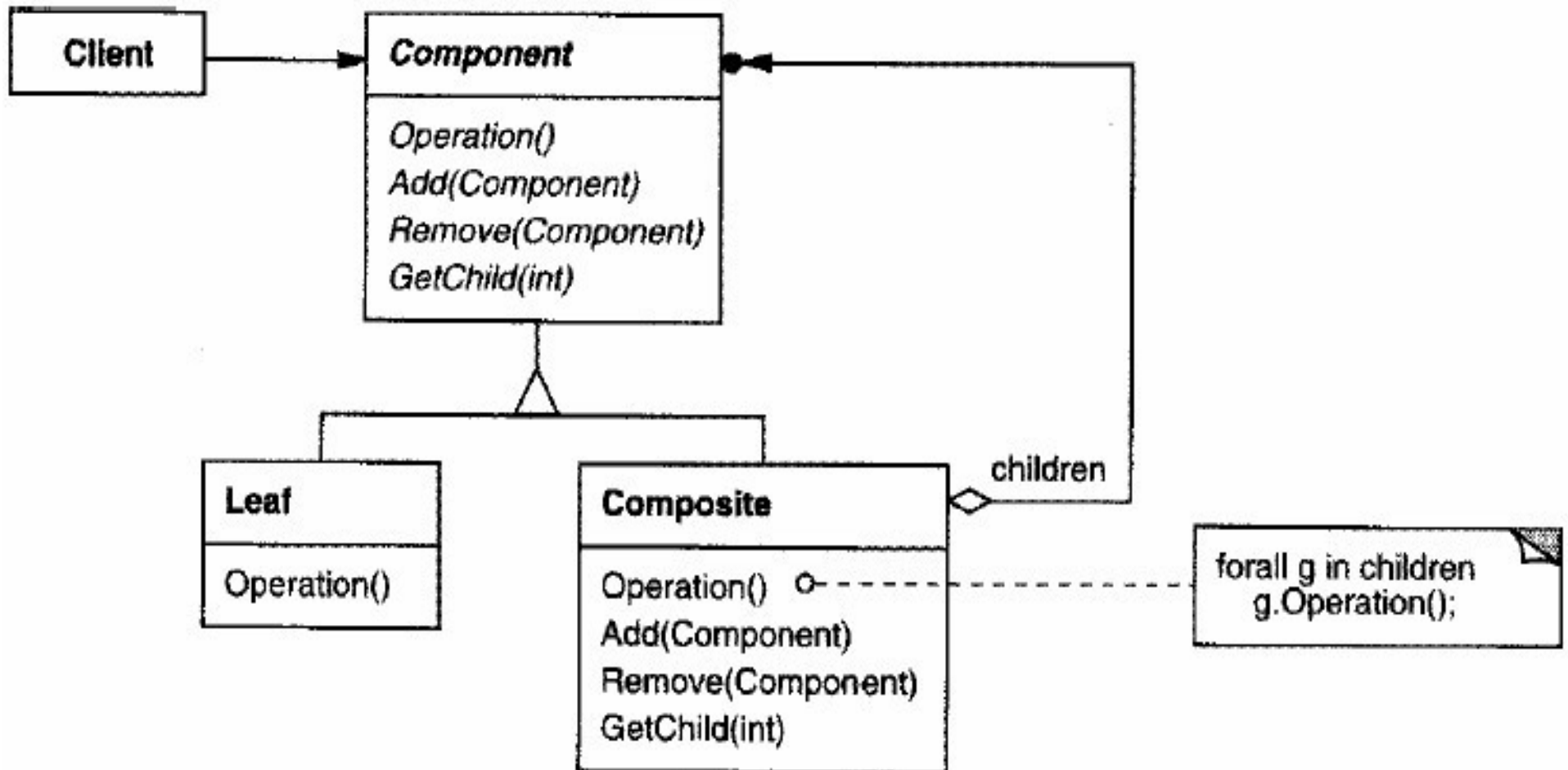
- Graphik-Editor

Anwendbarkeit

- Darstellung von Hierarchien (Whole-Part-Structures)
- Für Clients soll kein Unterschied in der Schnittstelle von zusammengesetzten und einfachen Objekten sein

# Composite

- Struktur



# Composite

---

## Teilnehmer

- **Component (Komponente)**
  - definiert Schnittstelle für zusammenges. und einfache Objekte
  - implementiert ggf. Default-Verhalten
  - definiert die Schnittstelle zum Zugriff und Verwalten von Unterkomponenten (Kindern)
- **Leaf (Blatt)**
  - stellt elementare Objekte dar; ein Blatt hat keine Kinder
  - definiert Verhalten elementarer Objekte
- **Composite (Zusammengesetztes Element)**
  - definiert Verhalten zusammengesetzter Objekte (mit Kindern)
  - speichert Unterkomponenten, implementiert „getChild“
- **Client (Benutzer)**
  - verwaltet Objekte mittels der Component-Schnittstelle.

# Composite

---

## Interaktion

- Vaterknoten können Aufträge an ihre Kinder weiterleiten und davor/danach noch eigene Operationen durchführen

## Konsequenzen

- + Einheitliche Schnittstelle für Clients erlaubt es, Clients einfach zu halten
- + Neue Component-Klassen können leicht über Vererbung hinzugefügt werden, ohne dass die Clients geändert werden müssen
- – Über die uniforme Schnittstelle zum Einfügen beliebiger Component-Objekte können beliebige Baumstrukturen erzeugt werden; Einschränkungen über Runtime-Checks
- Andere bekannte Einsatzgebiete: Ausdrücke, Kommandofolgen

# Composite

---

Implementierung

Beispiel-Code

Anwendungsbeispiele

- ET++ Vobjects
- InterViews Glyphs
- RTL Smalltalk Compiler Expressions

Bezug zu anderen Mustern

- Chain of Responsibility
- Decorator
- Flyweight
- Iterator
- Visitor

# Anmerkungen zum GoF-Ordnungssystem

---

- Fokus nur auf Objektorientierung
- Schwerpunkt Design
- Unterscheidung klassen- vs. objektbasiert ist nicht besonders trennscharf
- Spezifika von unterschiedlichen Anwendungsdomänen werden nicht herausgearbeitet
- Ordnungssystem wurde für 23 Muster geschaffen: Erweiterbarkeit?

# Weitere Klassifikationsansätze

---

- Klassifikation im „Siemens Pattern Buch“ [BMR+96]
- Muster-Katalog nach Verwendungszweck

# Klassifikation nach Buschmann et al. 1995, 1996

---

## • Abstraktionsgrad

- höchste Abstraktionsstufe: *Architectural Patterns*
  - Lösungen für fundamentale strukturelle SW-Organisation
  - Subsysteme als „Teilnehmer“ / Elemente
  - Spezifikation der Funktionen und Beziehungen der Subsysteme
- mittlere Abstraktionsstufe: *Design Patterns*
  - Verfeinerung von Subsystemen oder Komponenten
  - Lösungen für wiederkehrende Designprobleme
- niedrigste Abstraktionsstufe: *Idiome*
  - Lösungen für Programmiersprachen-spezifische Probleme
  - Beispiele: Cope's Idiome für C++ [Coplien1992,Coplien2000]
- Problembereich
  - z.B.: Communication, Access Control, Distributed Systems

# Kategorien nach Zweck

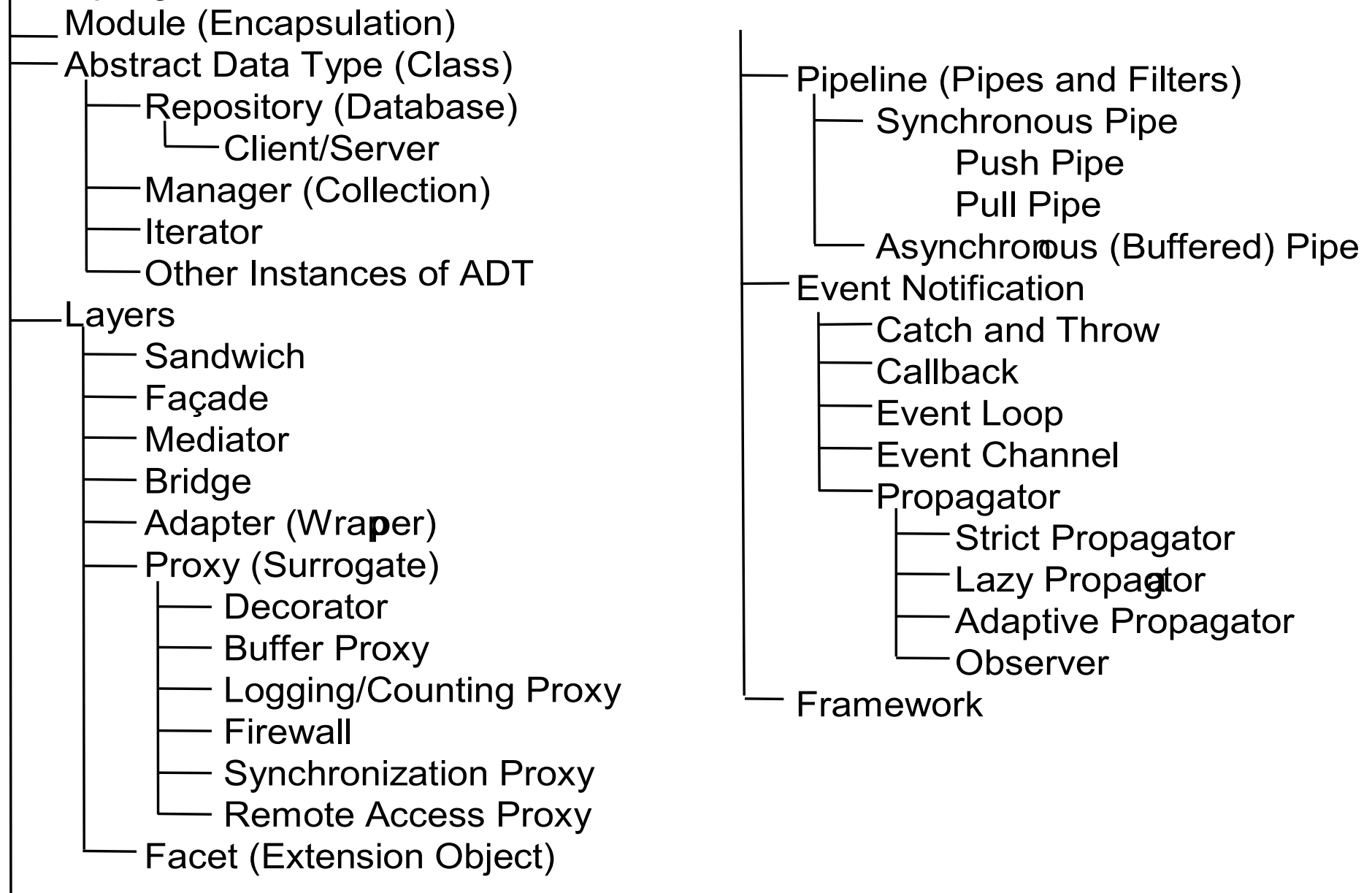
## Tichy 1998

---

- Decoupling
- Control
- Concurrency
- Distribution

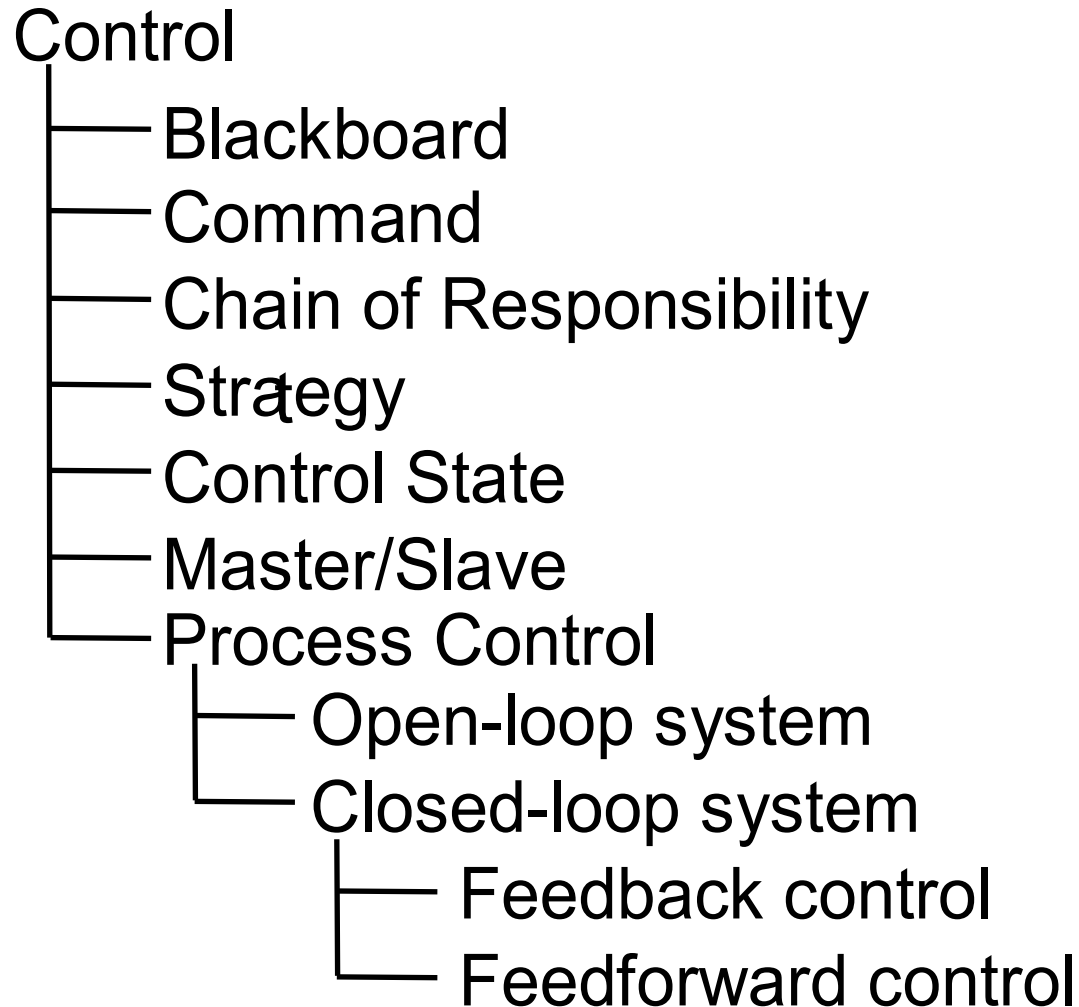
# Zweck: *Decoupling*

## Decoupling



# Zweck: *Control*

---



# Zweck: *Concurrency*

## Concurrency

- Semaphore
- Critical Region
- Conditional Critical Region
- Monitor
- Double-Checked Locking
- Thread Management
- Event Loop
- Send/Receive
- Reactor
- Rendezvous
- Listener
- Daemon
- Readers/Writers
- Bounded Buffer
- Active Object (Future)
- Asynchronous Completion Token
- Half-Sync/Half-Async
- Transaction and Rollback

# Zweck: *Distribution*

---

## Distribution

- Protocol Stack
- Remote Procedure Call
- Router
- Acceptor and Connector
- Gateway
- Forwarder/Receiver
- Client/Server
- Client/Dispatcher/Server
- Broker
- Distributed State  
(Recoverable Distributor)

# Pattern Almanach von Linda Rising

---

- Ansatz definiert 65 Klassen („Categories“)
- Jedes Muster wird möglichst einer Klasse zugeordnet
- Ordnungssystem (Auszug)
  - Accounting
  - Design Process
  - Air Defense
  - Event-Driven Systems
  - Analysis
  - GUI Development
  - Architectural
  - Health Care
  - Banking
  - Integration
  - C++ Idioms
  - Reactive Systems
  - Client-Server
  - Refactoring
  - Concurrent Systems
  - ...

# Anmerkungen zum Almanach

---

- Nachschlagewerk für „Experten“
- Bedeutung der einzelnen Klassen nicht/wenig erläutert
- unausgewogene Klassenbelegung 1 bis 181 Klassen
- keine Hierarchisierung
  - z.B. nach Domäne, Verwendungszweck
- Klassen unterstellen weitgehend eine eindeutige Einordnung eines Musters
  - Dies ist kaum möglich ...
- Trotzdem: große Leistung!
- Online-Version von Charles Weir:  
<http://www.smallmemory.com/almanac/>

# Beispiele aus dem Online Pattern Almanach

---

## **Pattern: Model-View-Controller**

**Pages:** 125-143

Divide an interactive application into three components. The model contains the core functionality and data. Views display information to the user. Controllers handle user input. Views and controllers together comprise the user interface. A change-propagation mechanism ensures consistency between the user interface and the model.

**Category:** Architectural

- In [Presentation-Abstraction-Control](#), the abstraction component corresponds to the model in Model-View-Controller (MVC), and the view and controller are combined into a presentation component. Communication between abstraction and presentation components is decoupled by the control component. The interaction between presentation and abstraction is not limited to calling an update procedure, as it is in MVC.
- [Session Control & Observation](#) is a specialization of MVC that controls and maintains the state of a networked multimedia session and notifies those interested in that state.

Anmerkung: Seitenangaben beziehen sich auf die jeweilige Druckversion

# Beispiele aus dem Online Pattern Almanach

---

## **Pattern: Publisher-Subscriber**

**Pages:** 339-343

Keep the state of cooperating components synchronized by enabling one-way propagation of changes. One publisher notifies any number of subscribers about changes to its state.

**Category:** Behavioral

- The publisher is the subject in [Observer \[Gamma+95\]](#). The subscribers are observers.

## **Pattern: Observer**

**Pages:** 293-303

Define a one-to-many dependency between objects so that when one object changes state, the others are notified and updated automatically.

**Category:** Behavioral

- Colleagues can use this pattern to communicate with a [Mediator](#)
- [Implementation Patterns for the Observer Pattern](#), [Observation](#), [Observer Update Message](#), [Publisher-Subscriber](#)

# Überblick von Pattern Katalogen

Ansatz	Anzahl Muster	Klassifikationsprinzip	Anzahl Klassen
[Quibeldey-Cirkel 1999]	-	Facettenklassifikation mit 4 Facetten	576
[Buschmann+ 1994]	23	Facettenklassifikation mit 3 Facetten	48
[GoF 1995]	23	Facettenklassifikation mit 2 Facetten	6
[PLoP 1-4]	je ca. 30**	Klassifikation ohne Überlagerung	8-10
[Gamma 1992]	35*	Hierarchische Klassifikation	7
[POSA 1]	40	Facettenklassifikation mit 2 Facetten	42
[Tichy 1998]	42***	Hierarchische Klassifikation	111***
[POSA 2 2000]	49	Facettenklassifikation mit 2 Facetten	21
[Rising 2000]	1140****	Klassifikation mit Überlagerung	65

Legende:

nach [FL01]

\* einige Muster wenig ausgearbeitet, eher allg. Entwurfsrichtlinien

\*\* Angegeben sind Artikel über Muster, wobei in einem Artikel z. T. mehr als ein Muster beschrieben wird

\*\*\* Muster / Klassen nicht klar getrennt, 42 Muster durch Lit. nachgewiesen, insg. sind 111 Muster/Klassen angegeben

\*\*\*\* Angabe geschätzt

# Generelle Anforderungen an Ordnungssysteme

---

- Formal-wissenschaftliche Kriterien:
  - Vollständigkeit
  - Präzision
  - Konsistenz
- Weitere Kriterien:
  - Erweiterbarkeit
  - Benutzbarkeit
  - Wirtschaftlichkeit
- Problematik der Pattern
  - Heterogenität der Domänen, Stile, Zielsetzungen ...
  - in Diskussion mit Coplien sagte er „Patterns are Literature“

---

# 1. Übung

# Observer und MVC

---

---

# Literatur

---

- AIS+95 Christopher Alexander and Sara Ishikawa and Murray Silverstein and Max Fiksdahl-King and Shlomo Angel, Eine Muster-Sprache. Städte, Gebäude, Konstruktion, 1995
- Bie00 Diethelm Bienhaus, Entwurfsmusterorientierter Ansatz zur einfacheren Realisierung verteilter Systeme, 2000
- BMR+96 Frank Buschmann and Regine Meunier and Hans Rohnert and Peter Sommerlad and Michael Stal, Pattern-orientierte Softwarearchitektur. Ein Pattern-System, 1998
- Bor01 Jan Borchers, A Pattern Approach to Interaction Design, 2001
- Cop91 James O. Coplien, Advanced C++ Programming Styles and Idioms, 1991
- GHJ+95 Erich Gamma and Richard Helm and Ralph Johnson and John Vlissides, Design Patterns. Elements of Reusable Object-Oriented Software, 1995
- GHJ+96 Erich Gamma and Richard Helm and Ralph Johnson and John Vlissides, Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software, 1996
- SSRB02 Douglas Schmidt and Michael Stal and Hans Rohnert and Frank Buschmann, Pattern-orientierte Softwarearchitektur. Muster für nebenläufige und vernetzte Objekte, 2002
- FL01 Fettke, P.; Loos, P.: Zur Klassifikation von Patterns. In: Organisatoren der Net.ObjectDays (Hrsg.):  
Net.ObjectDays 2001 - Tagungsband, 10. - 13. September 2001, Messekongresszentrum Erfurt. Erfurt 2001, ISBN 3-00-008419-3, S. 251-252.