Architektur II
Patterns (Entwurfsmuster)

Vorlesung: Software-Engineering für große Informationssysteme

TU-Wien, Sommersemester 2002

Wolfgang Keller

Überblick

- Was sind Patterns (Entwurfsmuster)
  - Herkunft, Begriffe, Beispiele, Arten
- Ein größeres Beispiel
  - Zugriffsschicht relationale Datenbanken
- Beispiele aus Bereich Versicherungssoftware
- Wo und wie verwenden wir Patterns
- Zusammenfassung: FAQs zu Patterns
- Web-Adressen und Literatur

Was sind Patterns (Entwurfsmuster)?

Herkunft, Begriffe, Beispiele, Arten

Etwas Geschichte ...

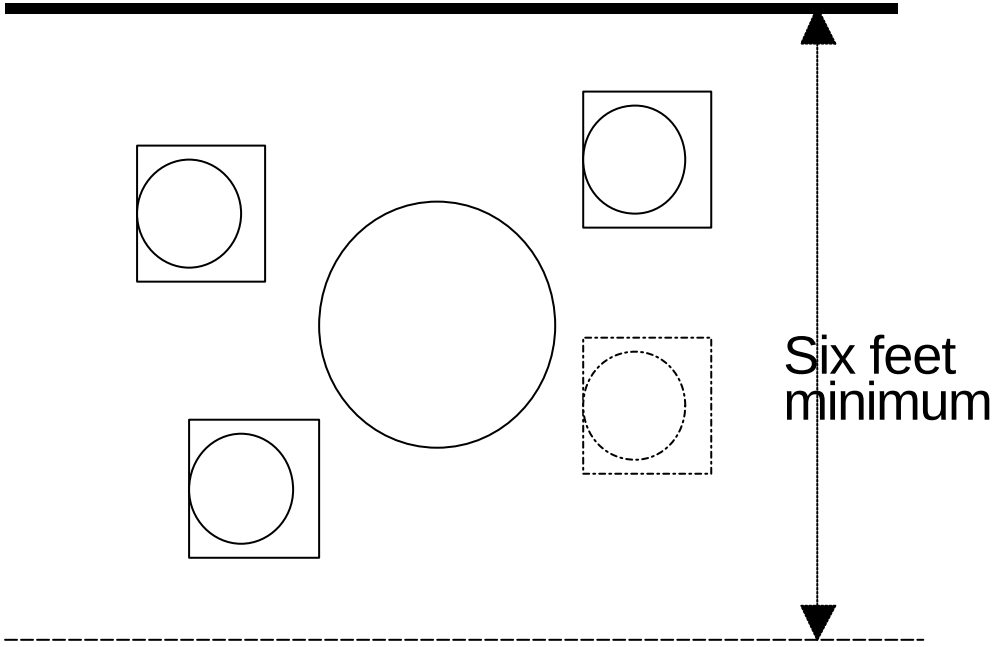
- **Christopher Alexander**
  - „Bau“-Architekt mit dem Anspruch, ästhetisches und gutes Design erklärbar und erlernbar zu machen.
  - **1977**: A Pattern Language: Towns, Buildings, Construction, Oxford University Press,
  - **1979:** Timeless Way of Building, Oxford University Press
- **OOPSLA'87, Kent Beck, Ward Cunningham**
  - "Using Pattern Languages for Object-Oriented Programs".
- **1995: Design Patterns** von Gamma, Helm, Johnson, Vlissides

Gängige Definition

- Ein Entwurfsmuster (Pattern) ist ein Problem mit seiner Lösung in einem Kontext.

- In allen möglichen Ingenieurdisziplinen ist es nicht unüblich, Entwurfshandbücher zu benutzen, die Lösungen zu bestimmten Problemen bereithalten.

Erstes Beispiel
Six Foot Balcony



Balconies and porches which are less than six feet deep are hardly ever used

Balconies and porches made very small to save money; but when they are too small, they might as well not be there.
Al balcony is first used properly when there is enough room for a small table where they can set down glasses, cups, and the newspaper. ...

Pattern #167, Alexander, Ishikawa, Silverstein,
A Pattern Language, Oxford University Press 1977

Problem / Lösung / Kontext

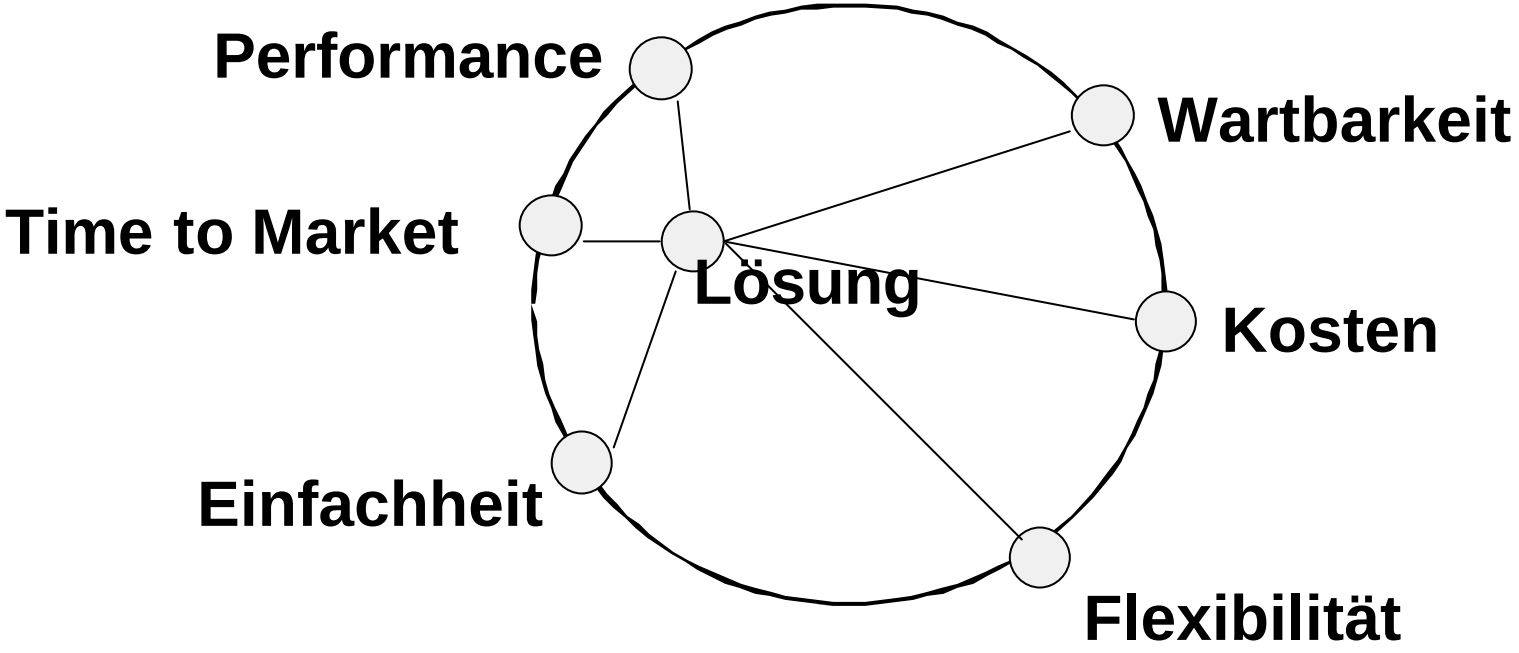
- Kontext
  - Bau von Mehrparteienhäusern. Große Balkone kosten viel Geld, bringen wenig Miete und werden daher oft klein gebaut und nicht benutzt. Alle wünschen sich einen hellen Balkon.
- Problem
  - Wie baut man eine Balkon so, daß er auch benutzt wird?
- Lösung
  - Man baue ihn mit einer Tiefe von mindestens 2 Metern, so daß man darauf bequem einen Tisch und Stühle für mehrere Personen unterbringen kann.

Kontext enthält Zielkonflikte, die der Designer abwägen muß

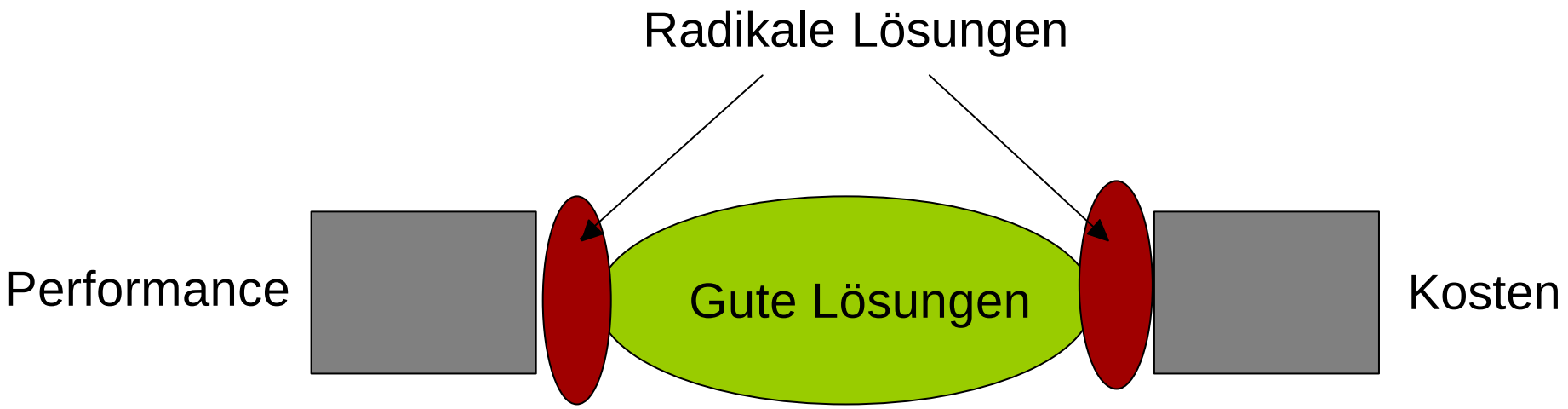
- Kosten **gegen** Benutzbarkeit
- Lichteinfall für den Nachbarn unter mir **gegen** Benutzbarkeit

- Bei Software sind Zielkonflikte zwischen nichtfunktionalen Eigenschaften typisch:
  - Einfachheit **gegen** Wartbarkeit
  - Performance **gegen** Kosten
  - Lese-Performance **gegen** Schreib-Performance
  - Batch-Performance **gegen** Online Performance
  - ... und viele mehr

Gut geschriebene Patterns machen neben der Nennung einer Lösung für ein Problem die Zielkonflikte explizit und zeigen die Konsequenzen, die eine Lösung hat auf ...



Gute Lösungen liegen selten genau auf einem Extrem eines Zielkonfliktes ...



Was ist ein Pattern und was nicht …

- Patterns sind bewährte Lösungen, die beschrieben werden, weil sie in der Praxis unabhängig voneinander an vielen Stellen zu finden sind
- Ein eigenes Design, das man einmal gemacht hat und das man in der Form von Patterns beschreibt, ist noch lange kein Pattern (es fehlen die sog. **Known Uses**)
  - Eine Lösung zu einem häufig auftretenden Problem ist noch lange kein gut geschriebenes Patterns, wenn die Zielkonflikte und die Konsequenzen der Lösung auf die Zielkonflikte nicht erläutert sind (es fehlen die sog. **Forces** und **Consequences**)

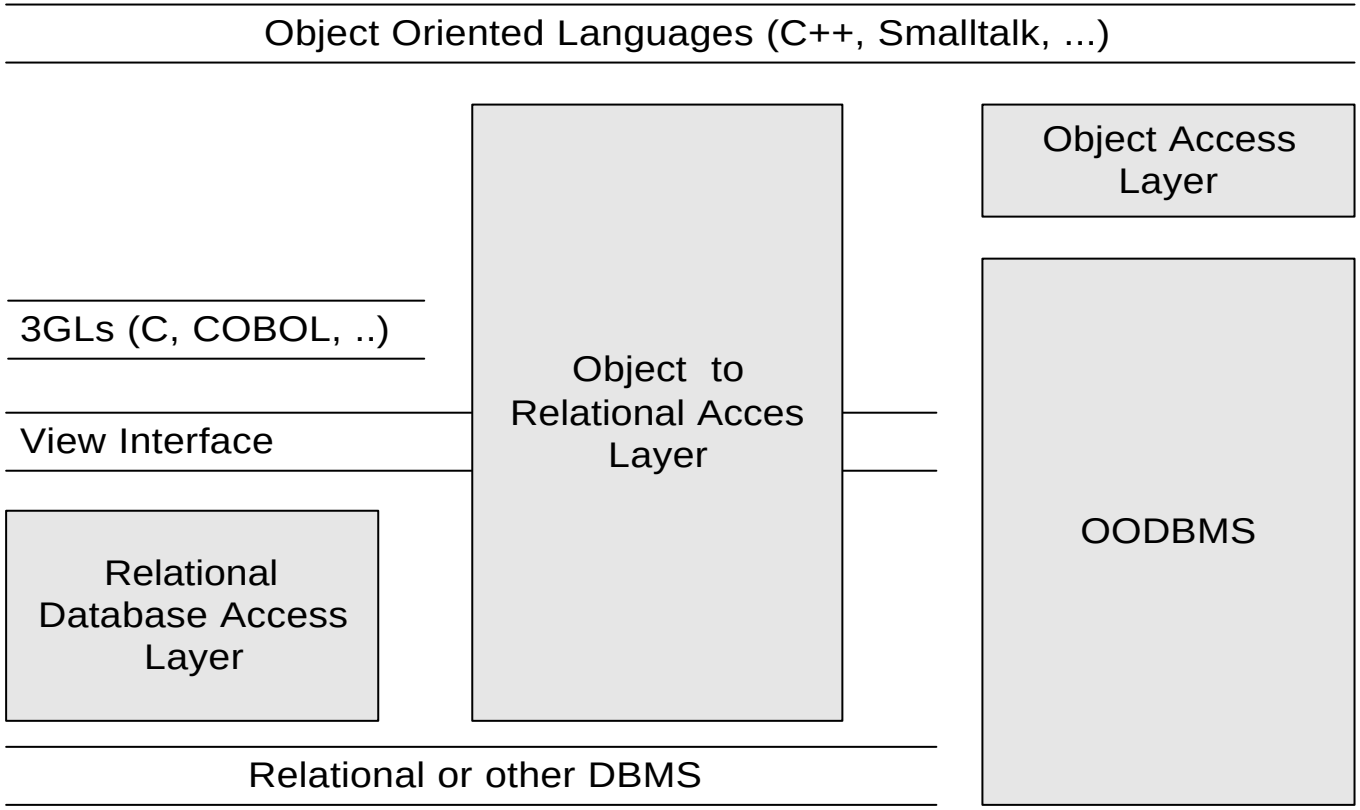
Wesentliche Arten von Patterns

- Analyse Patterns
  - Beschreiben wiederkehrende Lösungen in fachlichen Objekt- und Datenmodellen. Beispiel: Rolle.
- Architektur-Patterns
  - Beschreiben Muster auf der Ebene von Software-Architekturen: Beispiele: Schichten, Pipes&Filters, Repository, Blackboard, ...
- Design-Patterns
  - Beschreiben Interaktionen von Klassen auf der Ebene von technischem Design: Beispiele: PAC, MVC, Observer, Factory, ...
- Idioms
  - Beschreiben Muster für guten Code in einer speziellen Programmiersprache: Beispiel Letter/Envelope, Counted Reference, ..

Ein größeres Beispiel

Eine Menge von Patterns für Zugriffsschichten auf relationale Datenbanken

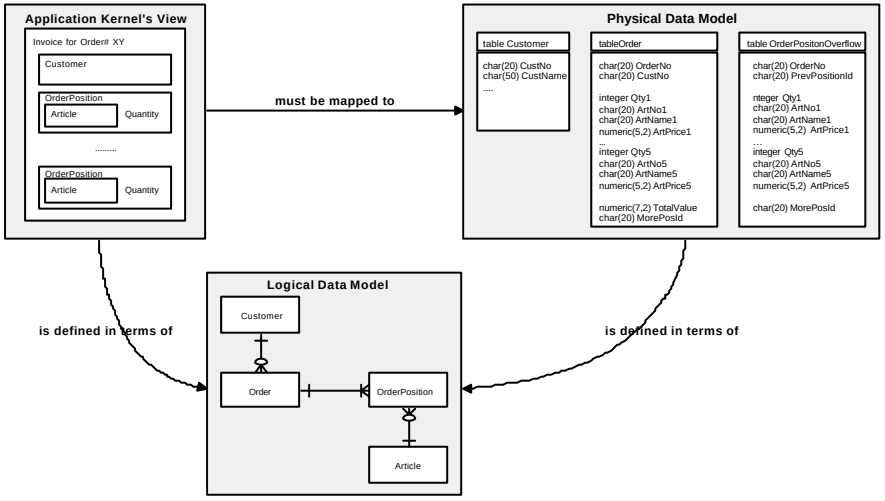
Database Access Layers
Three Different Types



Relational Database Access Layers
A Pattern Language

- Problem
- Framework Overview
- Roadmap of the Pattern Language
- The Hierarchical Views Pattern
- The Physical Views Pattern
- The Query Broker Pattern

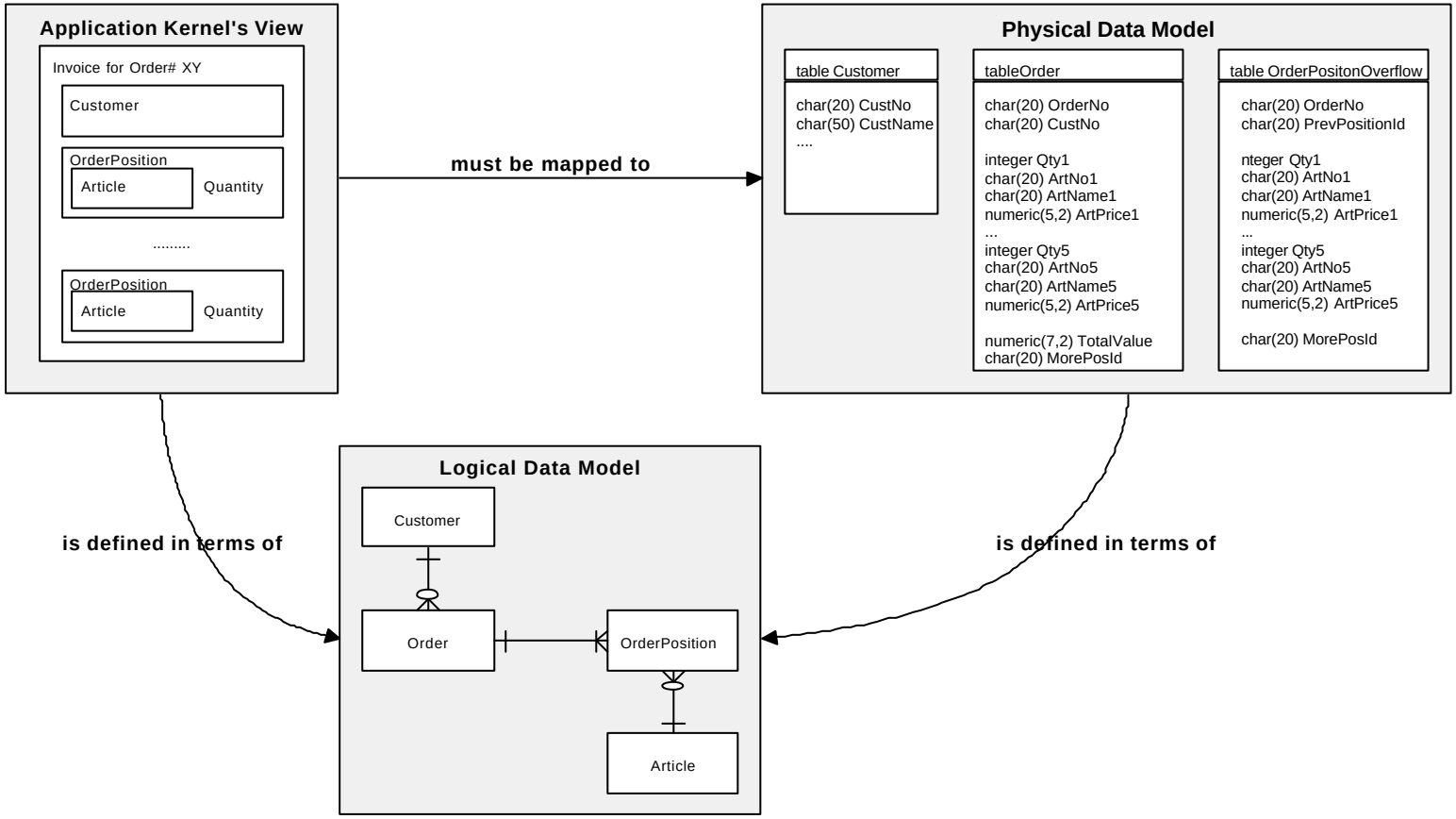
Problem



How do you access a relational databases in a 3-Layer architecture?

- Persistence for “relational” applications
- Physical data models tend to be unstable
- Application kernel is stable

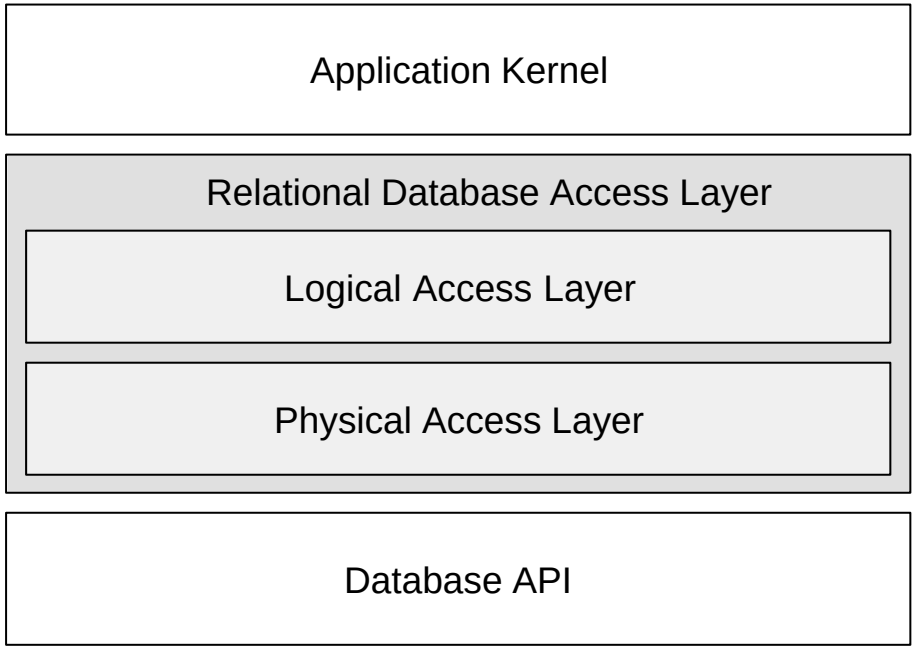
Problem


Application Kernel's View
Invoice for Order# XY
Customer
OrderPosition
Article
Quantity
.....
OrderPosition
Article
Quantity
must be mapped to
Physical Data Model
table Customer
char(20) CustNo
char(50) CustName
....
tableOrder
char(20) OrderNo
char(20) CustNo
integer Qty1
char(20) ArtNo1
char(20) ArtName1
numeric(5,2) ArtPrice1
...
integer Qty5
char(20) ArtNo5
char(20) ArtName5
numeric(5,2) ArtPrice5
numeric(7,2) TotalValue
char(20) MorePosId
table OrderPositonOverflow
char(20) OrderNo
char(20) PrevPositionId
nteger Qty1
char(20) ArtNo1
char(20) ArtName1
numeric(5,2) ArtPrice1
...
integer Qty5
char(20) ArtNo5
char(20) ArtName5
numeric(5,2) ArtPrice5
char(20) MorePosId
is defined in terms of
Logical Data Model
Customer
Order
OrderPosition
Article
is defined in terms of

Forces

- Separation of concerns vs. cost
  - Separating database code increases design cost
- Ease-of-use vs. power
  - A powerful interface is complex to use
- Performance
- Flexibility vs. complexity
- Legacy systems vs. optimal design
  - Legacy systems dictate desing - and not always the best one

Framework Structure Solution

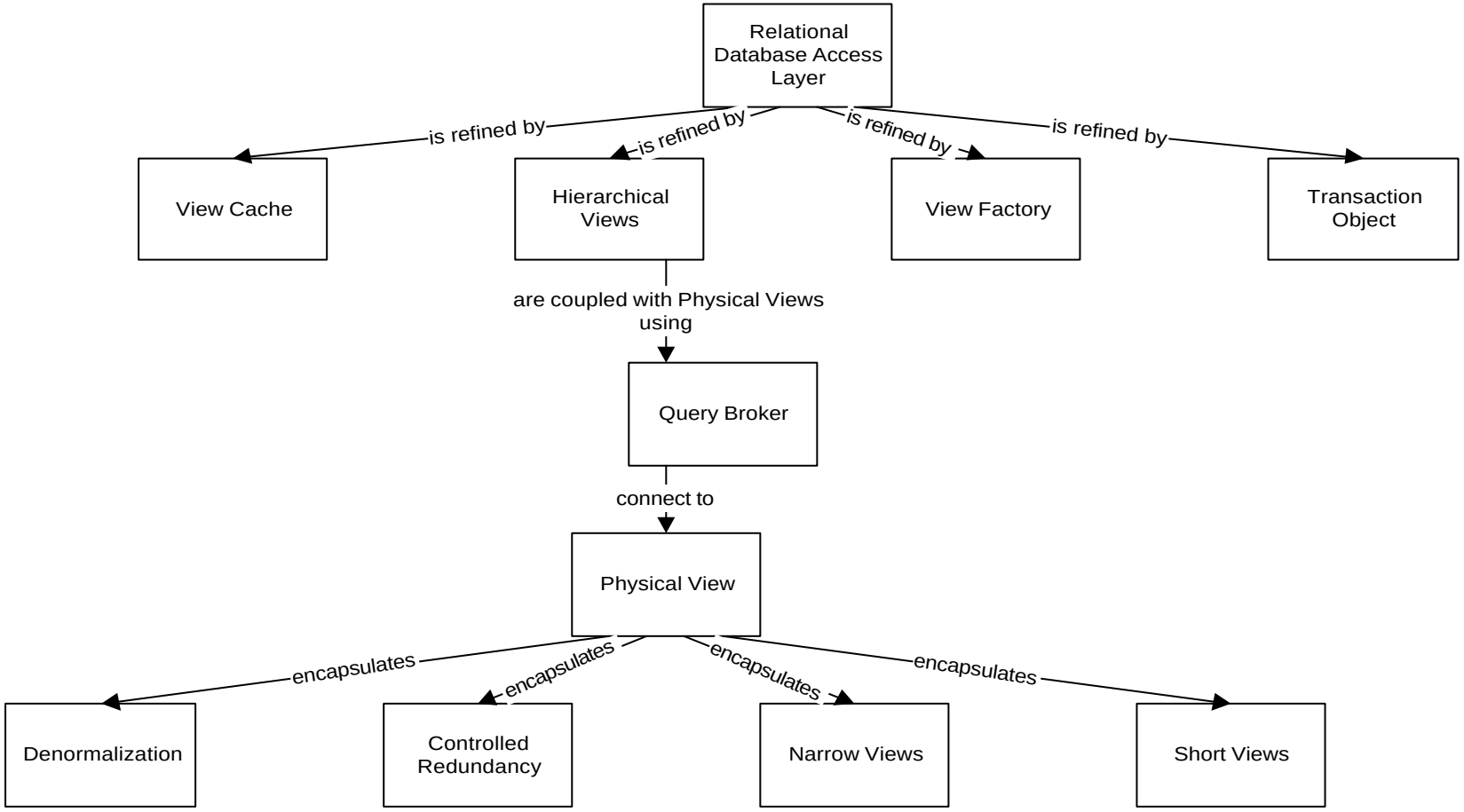


- use a layered architecture
- decouple the layers

Consequences

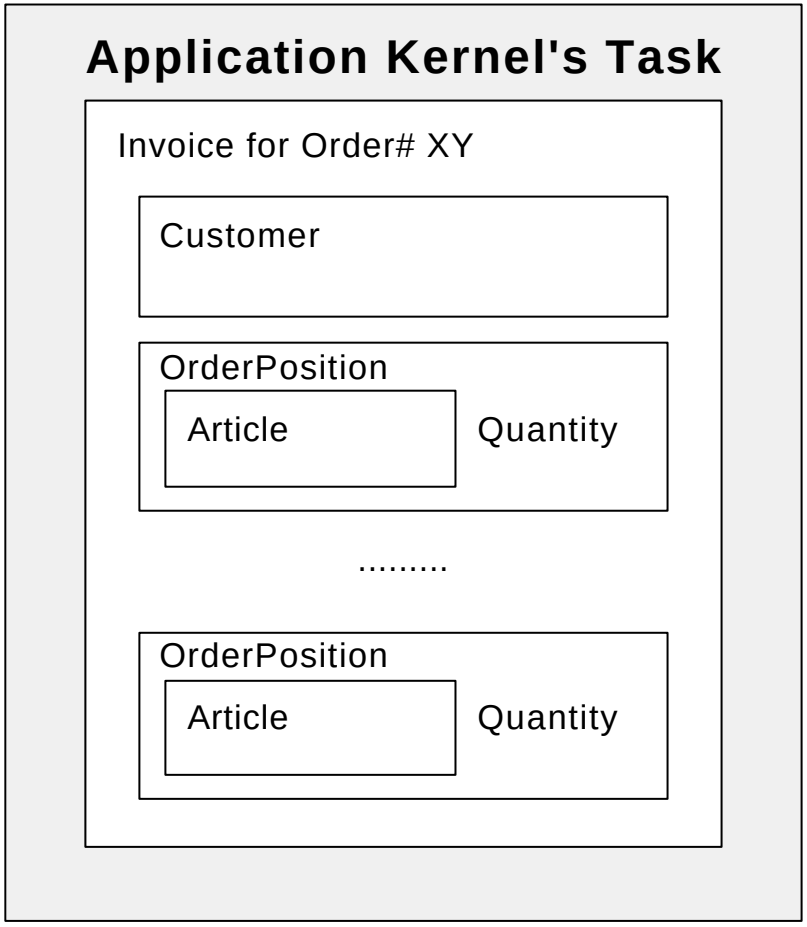
+ Separation of concerns
+ Easy-to-use interface
+ Easy tuning and database maintenance
+ May be used as reengineering strategy
+ Use of non-relational databases possible
+ Interface covers all requirements
– Little performance penalty
– Cost
– Reengineering of triggers and stored procedures necessary
– Many customized classes

Roadmap of the Pattern Language



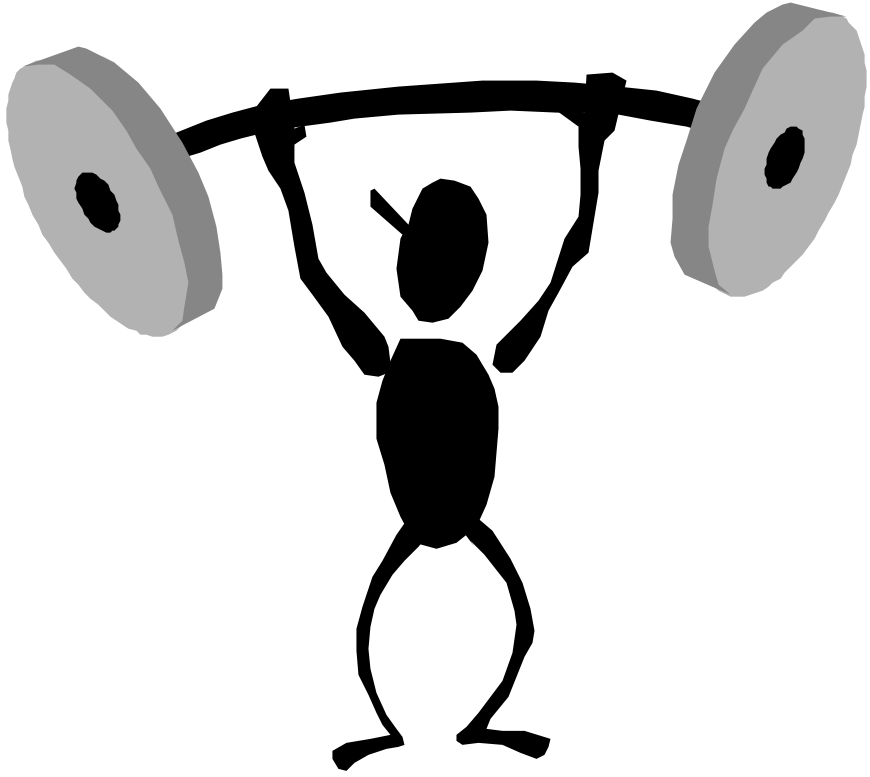
Hierarchical Views Context and Problem

- Context
  - Use of Relational Database Access Layer
- Problem
  - What interface does the access layer present to the application kernel?

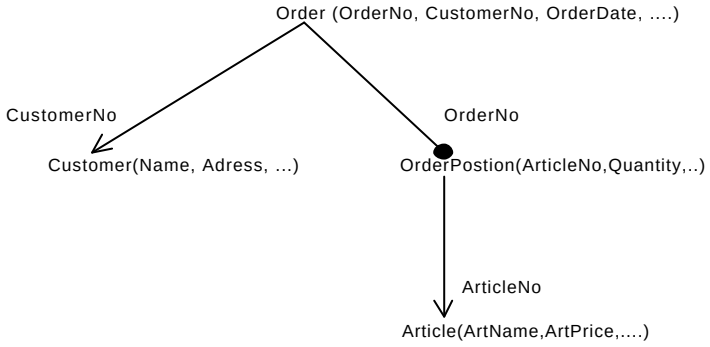


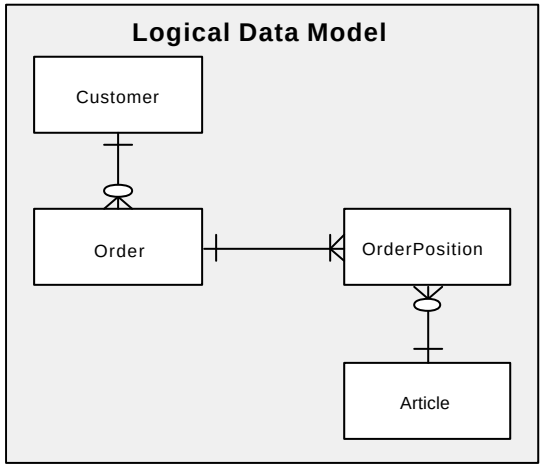
Hierarchical Views Forces

- Forces
  - Complexity vs ease-of-use and cost
  - Flexibility vs. speed of development
  - Performance
  - Mass problems



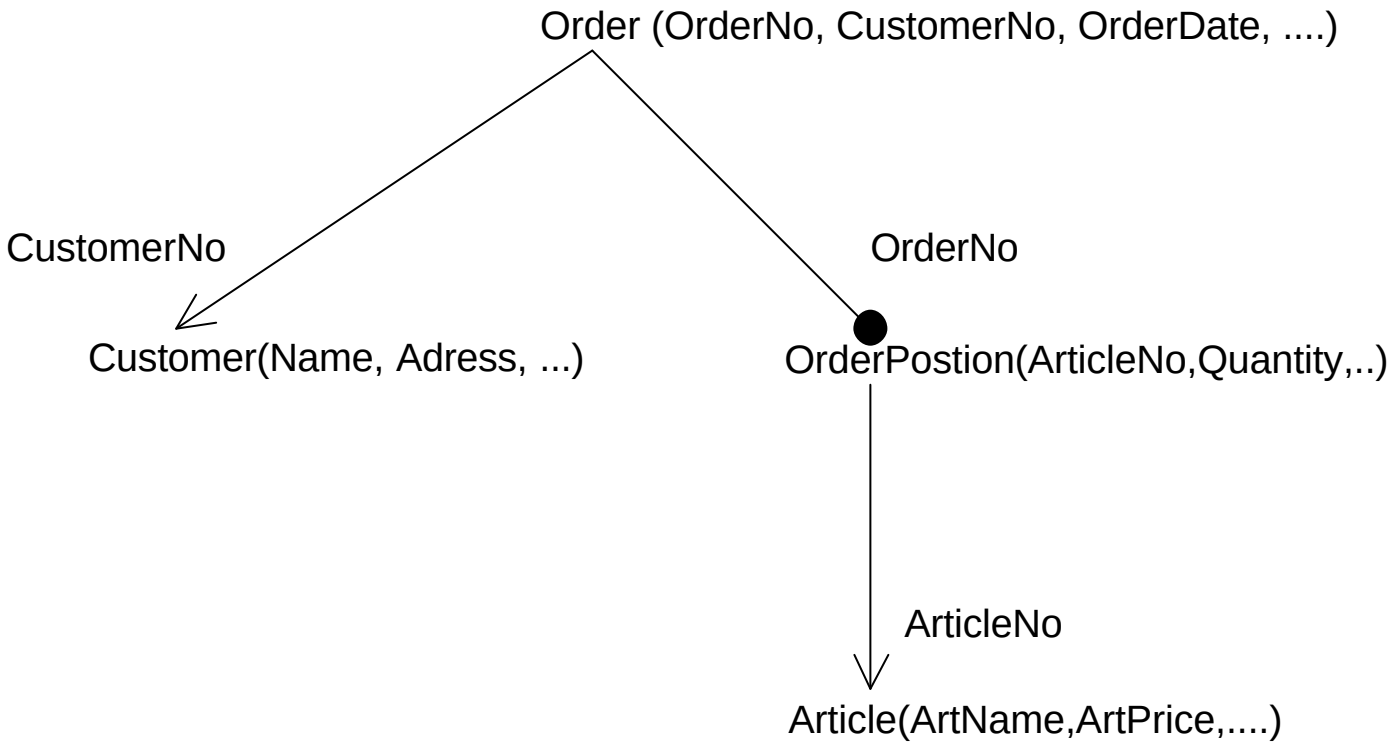
Hierarchical Views Solution



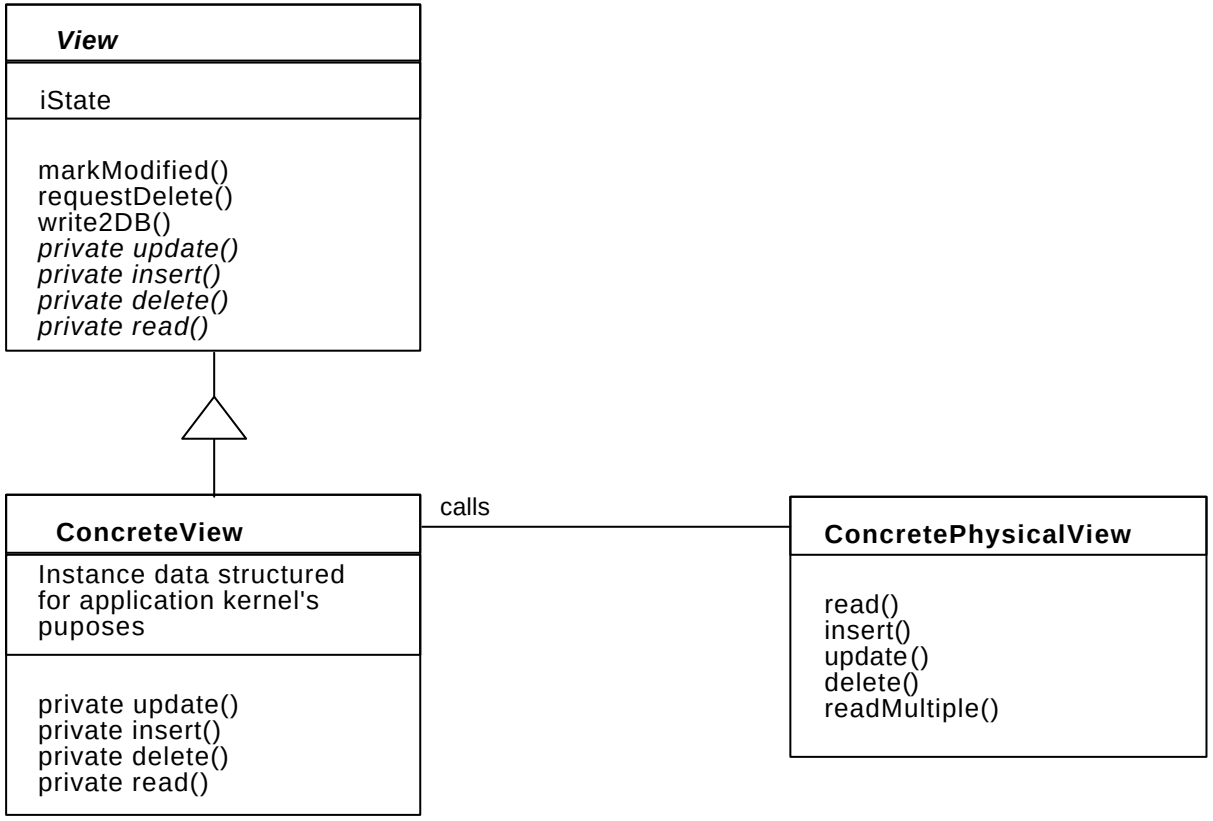


- Express the interface in terms of the data model
- Build an acyclic navigation graph for every use case (DAG)
- Label nodes with a projection of an entity
- Label edges for foreign key relations and arity

Hierarchical Views Solution



Hierarchical Views Structure



Hierarchical Views Interface Implementation Example


```
struct Customer {
    CustomerKeyType       iCustNumber;
    CustomerNameType      iCustName;
    CustomerAdressType    iCustAdress;
};
struct Article {
    ArticleNumberType     iArticleNumber;
    ArticleNameType       iArticleName;
    Money                 iArticlePrice;
};
struct OrderPosition {
    Article               iArticle;
    QuantityType          iQuantity;
};
```

Hierarchical Views Interface Implementation Example


```
class OrderInvoiceView : public View {
public:
    OrderInvoiceView ( OrderKeyType anOrder );
    ~OrderInvoiceView ( void );
    // instance variables
    OrderKeyType                                iOrder;
    Customer                                    iCustomer;
    Clist<OrderPosition,OrderPosition &>        iPositions;
    Money                                       iSumOfInvoice;
private:
    virtual void update ( void );
    virtual void insert ( void );
    virtual void remove ( void );
    virtual void read ( void );
};
```

Hierarchical Views Consequences

+ Minimal interface following use case structure
+ Easy navigation through the access tree
+ DAG notation may be used to automatically generate the interface
– No support for inheritance and polymorphism
– Depends on use cases
– Large number of classes

Physical Views
Context and Problem

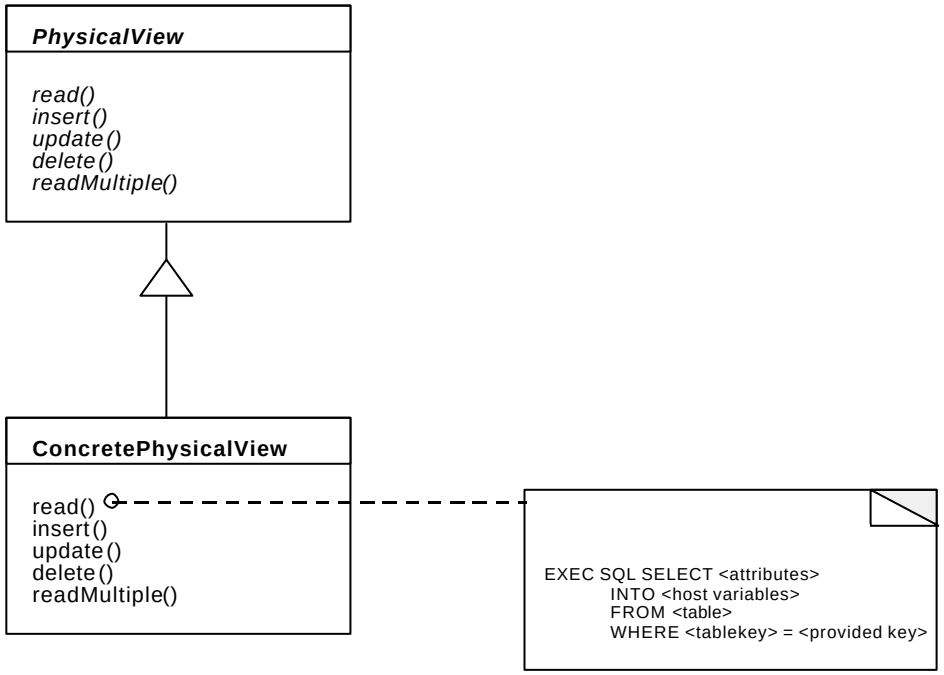
- Context:
  - Relational Database Access Layer
  - Hierarchical Views contain no database access
- Problem:
  - How do you provide an easy-to-use interface to your physical database?

Physical Views
Forces

- Simplicity vs. Performance
  - Simple 3NF data models result in an inefficient database design
  - Efficient database designs result in complex tables structures
- Flexibility
- Mass problems and cost

Physical View Solution

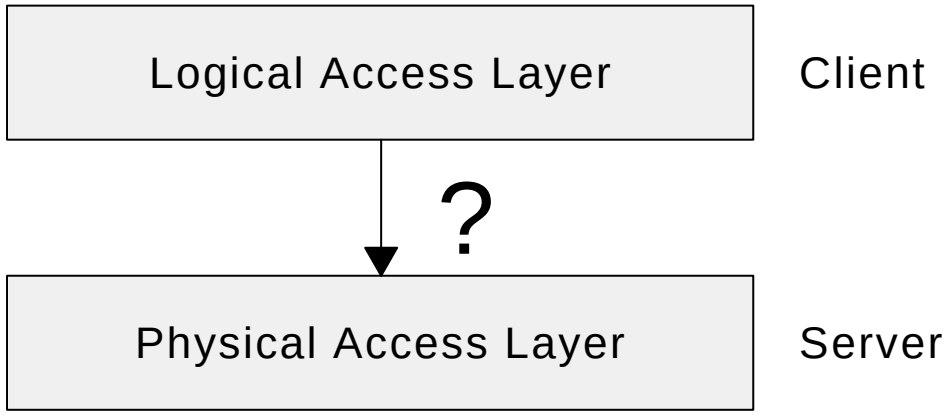
- Encapsulate every table and every view with a class
- Let this class handle overflow tables and other tricks too



Physical Views Consequences

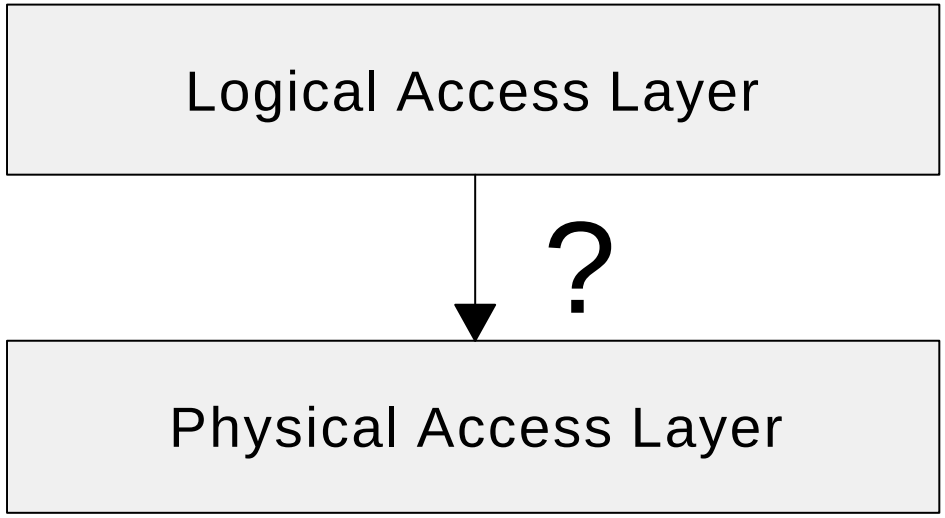
+ Simple interface
+ Database design is flexible
+ Technical code is encapsulated
– Additional level of indirection
– Many similar classes

Query Broker
Context and Problem



- Context:
  - Relational Database Access Layer
  - Hierarchical and Physical Views
- Problem:
  - How to connect the Hierachical Views and the Physical Views?

Query Broker Forces

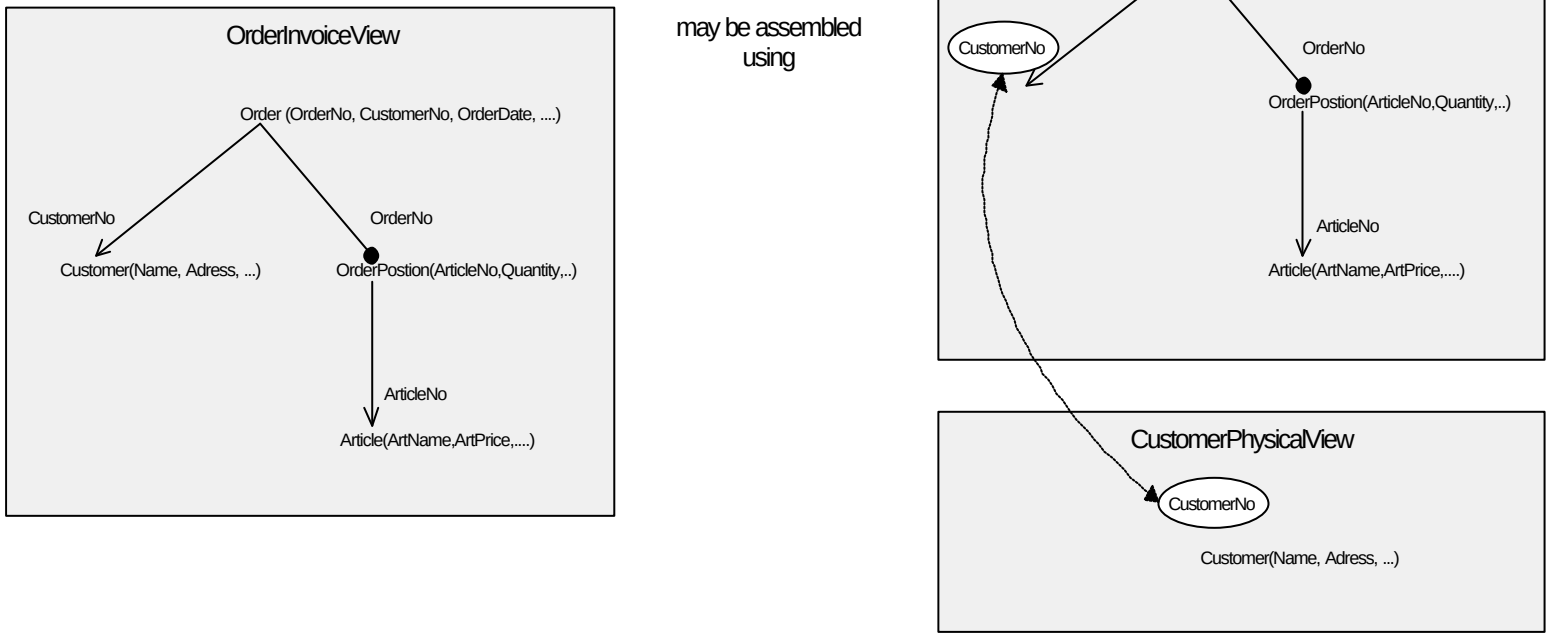


- Cost vs. flexibility
- Reusability across several applications
- Complexity

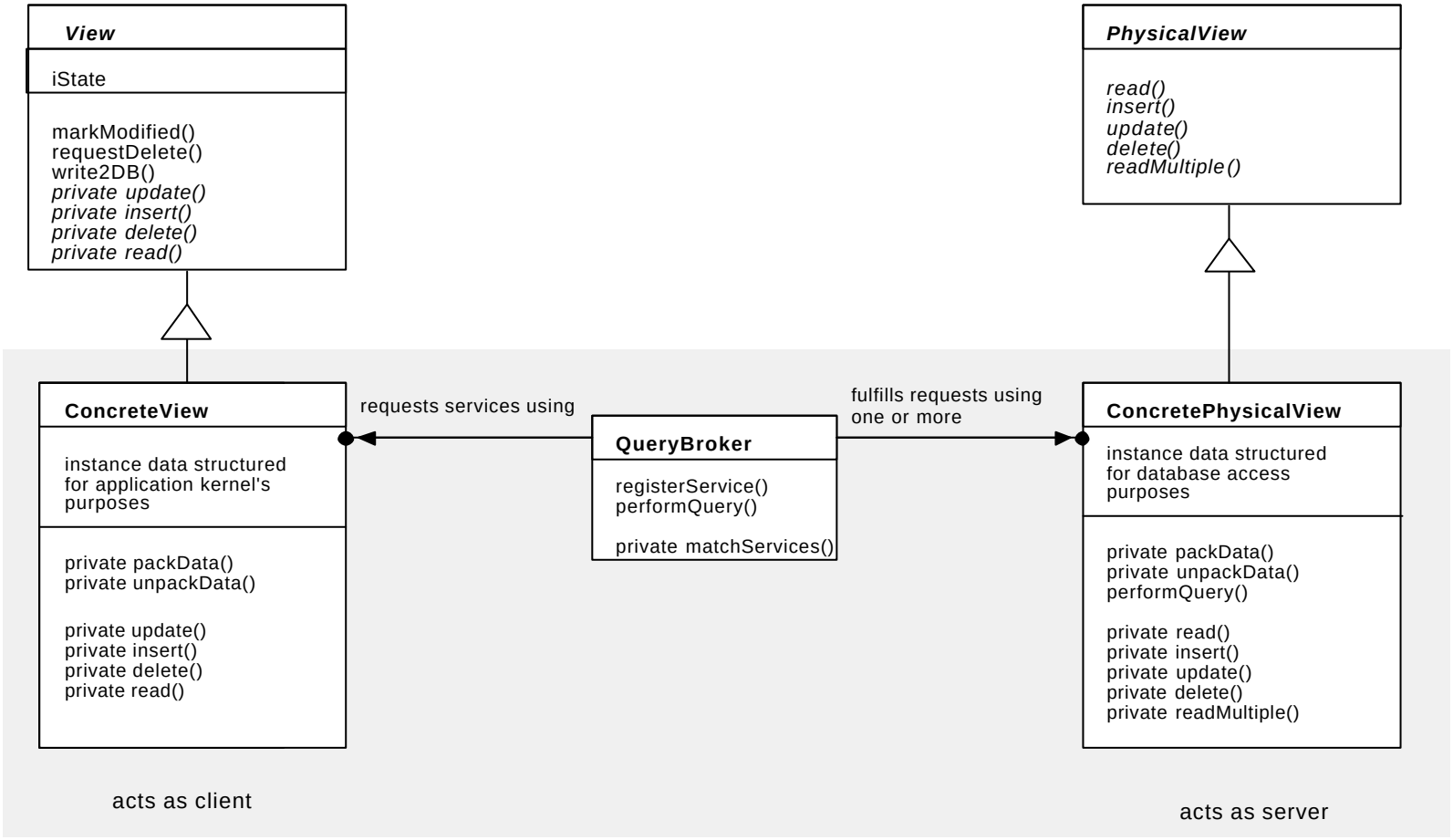
Query Broker Solution

- Use a broker and call it *Query Broker*
- The ConcreteViews form the client side of the broker, the PhysicalViews the server side.
- Describe services as DAGs in terms of the logical data model
- Use a tree matcher to find best queries
- Let the Query Broker assemble the results

The Query Broker Pattern Solution



Query Broker Structure



The Query Broker Pattern Consequences

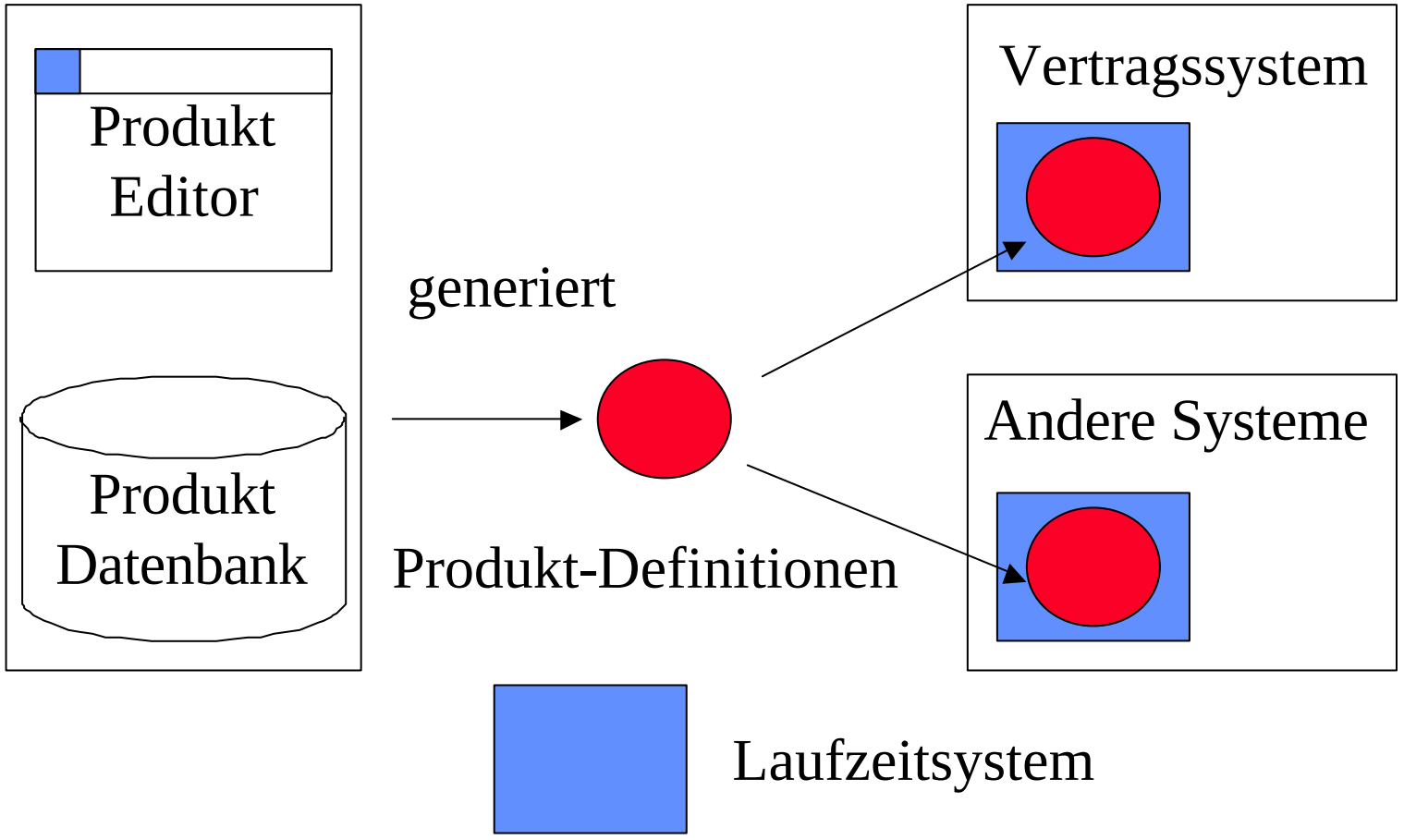
+ Flexibility
+ Decoupling
+ Decoupling
– Complexity
– Cost
– Minor run time penalty

Beispiele aus dem Bereich Versicherungssoftware

Beispiel aus dem Versicherungsbereich Architekturmuster Produktserver

- Problem
  - Neue Versicherungsprodukte sollen innerhalb von Tagen im DV System verfügbar sein. Wie baut man ein System, das dies möglich macht?
  - Faktoren, die das Design beeinflussen (exemplarisch ...)
  - Plattformunabhängigkeit **gegen** Kosten: Produktwissen muß plattformunabhängig zur Verfügung stehen und in vielen Teilsystemen.
  - Kapselung des Produktwissens **gegen** optimales Design von User Interfaces: Gut designte Benutzungsschnittstellen enthalten ebenfalls Produktwissen - eine vollständige Auslagerung in einen Produktserver führt zu User Interfaces, die nicht optimal zu benutzen sind.
  - ... Viele viele weitere ...

Architekturmuster Produktserver Lösung

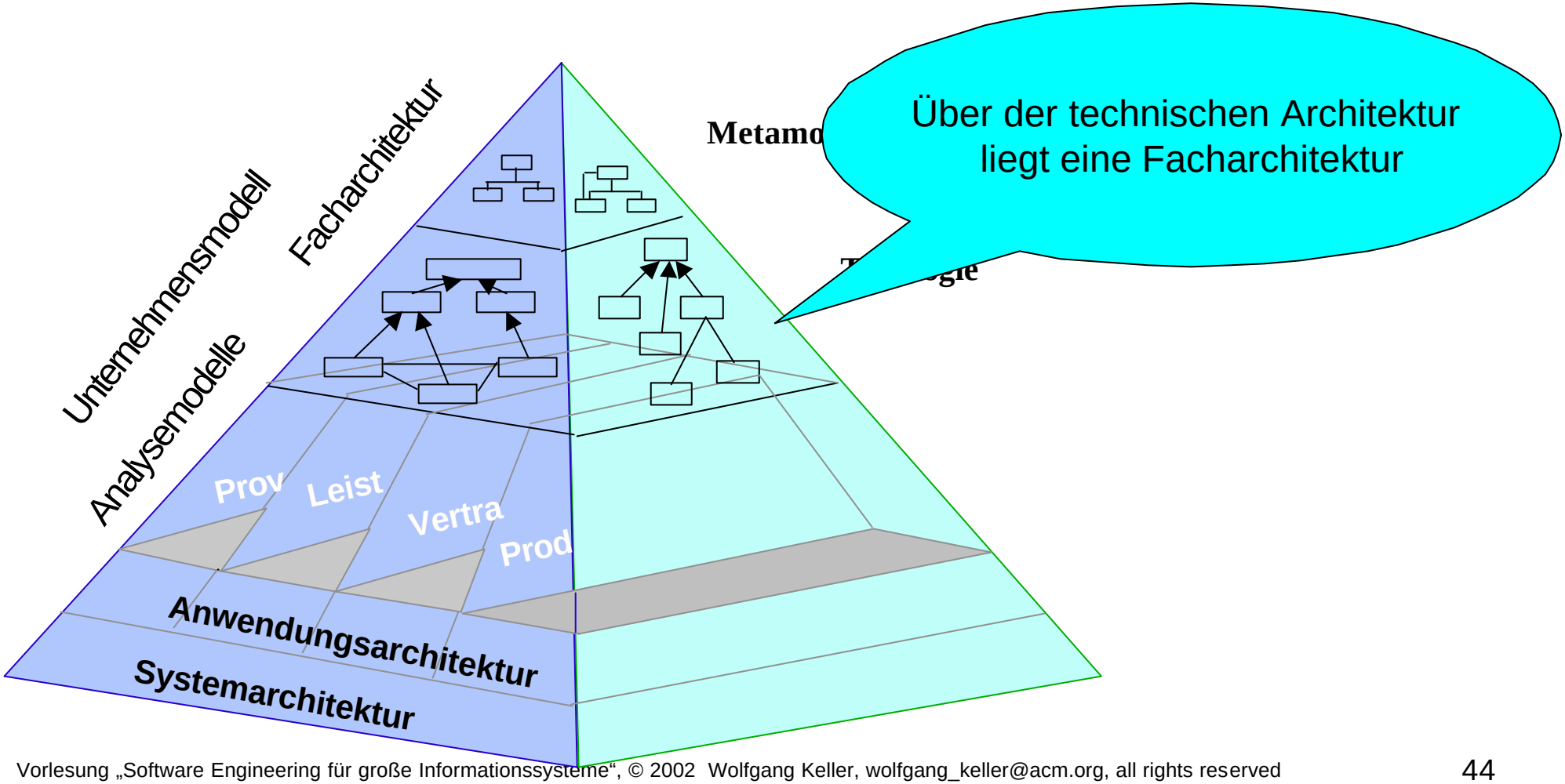


Architekturmuster Produktserver

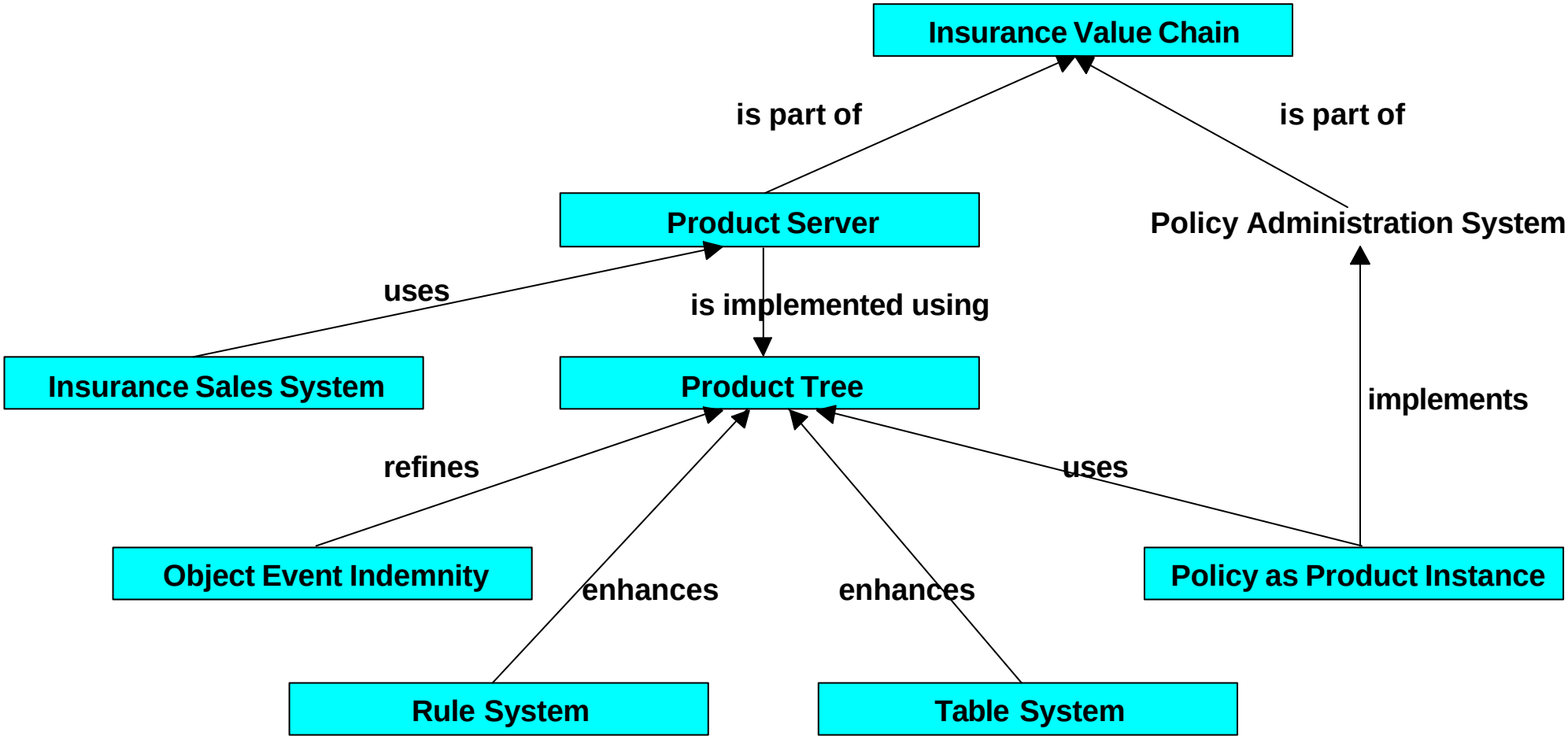
- Auswirkungen (exemplarisch ...)
  - Produktwissen wird plattformunanbhängig zur Verfügung gestellt
  - Benutzungsschnittstelle selten voll aus dem Produktwissen zu generieren - also hier noch Programmierung bei der Einführung neuer Produkte
  - ... Viele weitere ...

- Zum Nachlesen im Web unter ..
  - Some Patterns for Insurance Systems, Wolfgang Keller, PLoP 1998, http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/

Wo und wie ... Analyse- und Datenmodell-Patterns



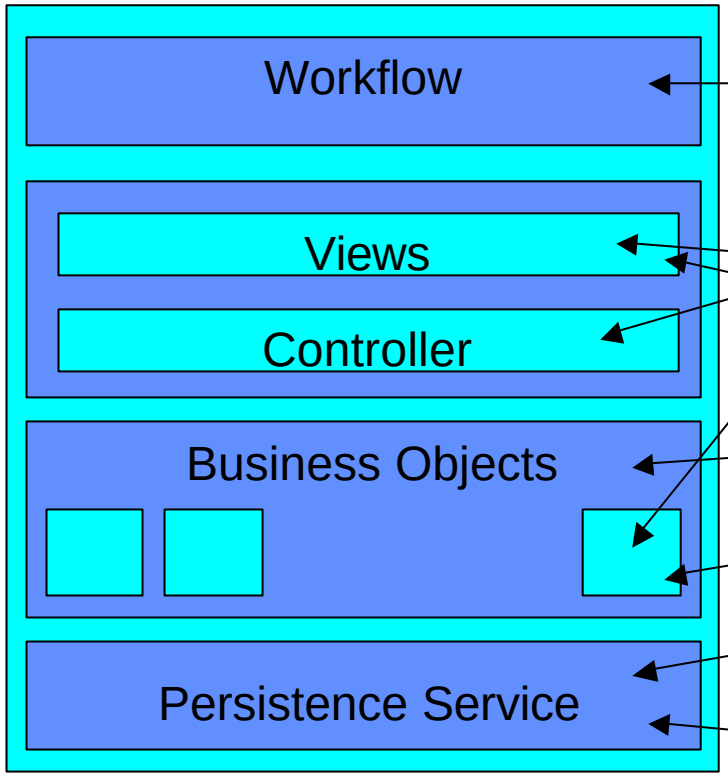
Beispiel: Landkarte bisher selbst beschriebener fachlicher Muster ..



Wo und wie haben wir Patterns zum Beispiel in Phoenix verwendet

Wo und wie ...
Design- und Architektur-Patterns

Entwurf und Erklärung von Architekturen



Schichtenarchitektur

MVC, PAC

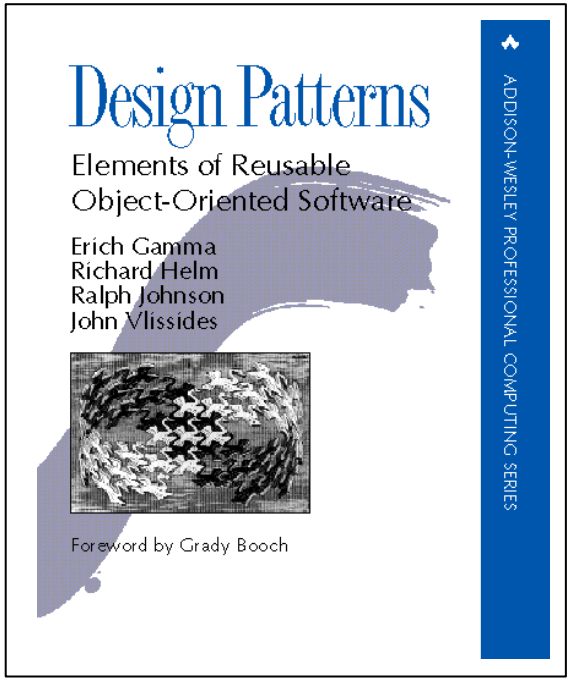
Fassade

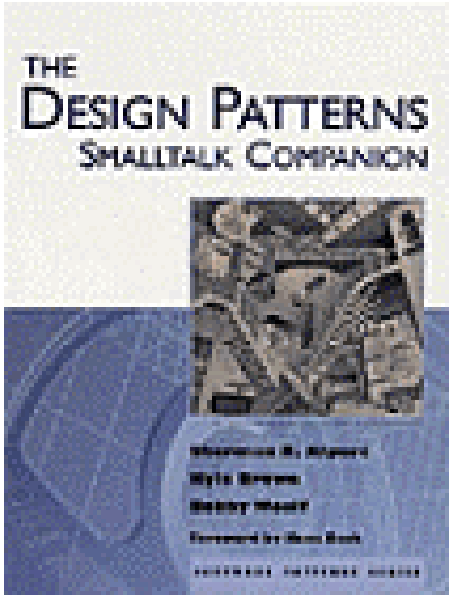
Observer

Singleton

Proxy

Gängige Bücher: Design- und Architektur-Patterns

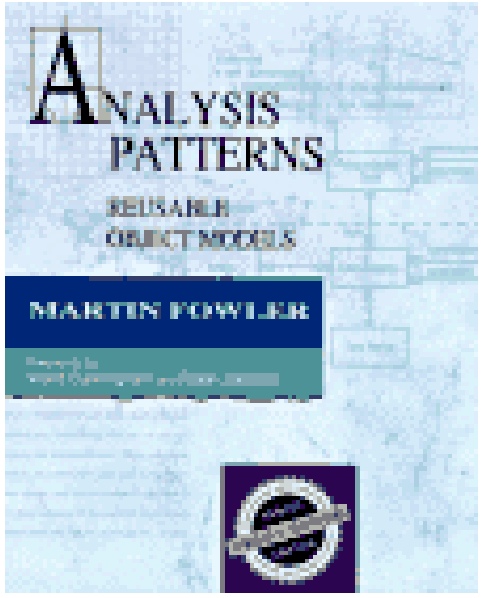




Wo und wie ... Analyse- und Datenmodell-Patterns

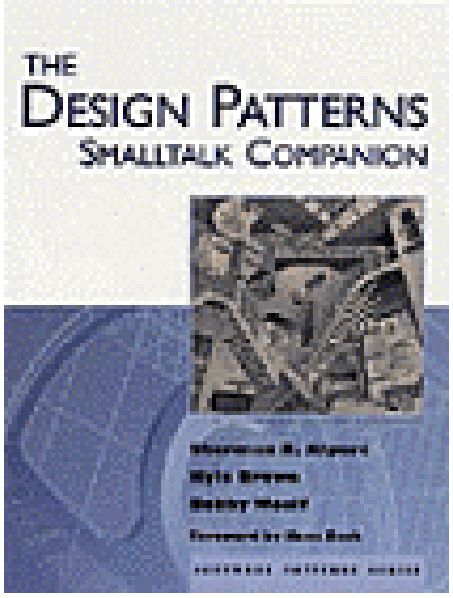
- Facharchitektur beschreibt fachliche Referenz-Objekt-Modelle
- Auch hier gibt es inzwischen viele Muster, die Sie sicher auch verwenden ...
  - Partner-Rollen (Partnersysteme)
  - Gespiegelte Stückliste (produktflexible Vertragssysteme)
  - Historien (Vertragssysteme)
  - Buchungen (in vielen Feldern)

Gängige Bücher: Analyse- und Datenmodell-Patterns



Wo und wie ... Programmier-Idioms, Design-Patterns

- Im Smalltalk-Bereich gibt es mehrere gute Bücher, die das Erstellen eigener Programmierrichtlinien we setzen ..



Wo und wie ...
Eigene fachliche Muster

- Verwenden wir, um den versicherungsfachlichen Teil der Facharchitektur zu beschreiben
- Zum Beispiel
  - Wertkette der Versicherung
  - Produktserver
  - Außendienst-Shell
  - Produktbaum
  - Objekt / Ereignis / Leistung
  - Gespiegelte Stückliste ...
  - ... Weitere

Fragen und Antworten zu Patterns

FAQ: Was sind Patterns nicht?

- Patterns sind nicht die Lösung aller Design Probleme
  kein Silver Bullet
- Patterns machen aus einem Anfänger keinen Spitzendesigner
- Patterns sind keine fertige Spezifikation
- Patterns ersetzen kein DV-Konzept und auch kein sauberes Design. Aber sie unterstützen beides

FAQ: Wo können Patterns helfen?

- Patterns schaffen eine **Sprache für Design**
- Das Problem und die Kräfte werden explizit gemacht - nicht nur die Lösung.
- Man bekommt fertige Skizzen für gutes Design
- Man kann sein eigenes Design besser dokumentieren
- Man wird schneller ein guter Designer

FAQ: Soll ich alle meine DV-Fachkräfte auf Pattern-Schulungen schicken?

- Man sollte besser von DV-Fachkräften erwarten, daß sie sich in Ihrem Beruf selbst auf dem laufenden Stand halten - und Patterns gehören dazu.
- Initialzündungen und kurze Veranstaltung sind gut - wochenlange Schulungen sollte man vermeiden
- Besser auf die Eigeninitiative setzen und gute Bücher leicht zugänglich machen. Patterns bei Design-Reviews etc. einfach verwenden und immer wieder erwähnen.
- Bei Externen voraussetzen.

Web-Adressen und Literatur

Anhang: Web-Adressen

- Eine Einführung findet man unter: Patterns and Software: Essential Concepts and Terminology
  - http://www.enteract.com/~bradapp/docs/patterns-intro.html
- Die Patterns Homepage enthält einen Index auf alles, was mit Patterns zu tun hat und eine komplette Buchliste ...
  - http://hillside.net/patterns/patterns.html
- Kultstatus hat das Wiki-Wiki-Web
  - http://c2.com/wiki
- Patterns für betriebliche Informationssysteme hat das ARCUS - Projekt beschrieben
  - http://www.objectarchitects.de/arcus/

Anhang: Gängige Bücher

- Architektur-Patterns
  - Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal: Pattern Oriented Software Architecture - A System of Patterns, Wiley 1996.
  - Mary Shaw, David Garlan: Software Architecture, Perspectives of an Emerging Discipline, Prentice Hall 1998 (nicht in Pattern Form - aber gut).
- Design-Patterns
  - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns, Elements of Reusable Object-oriented Software, Addison-Wesley 1995
  - Sherman Alpert, Kyle Brown, Bobby Woolf: The Design Patterns Smalltalk Companion, Addison Wesley 1998.

Anhang:
Gängige Bücher

- Idioms / Smalltalk
  - Kent Beck, Smalltalk Best Practice Patterns, Prentice-Hall 1996.
  - Suzanne Skublics, David Thomas, Edward Klimas: Smalltalk with Style, Prentice-Hall 1995.
- Idioms / C++
  - Jim Coplien, Advanced C++ Programming Styles and Idiomd, Addison Wesley, 1991.
  - viele weitere ....
- Analyse-Patterns
  - Martin Fowler: Analysis Patterns; Addison Wesley Longman, 1997.
  - David C. Hay, Data Model Patterns, Dorset House Publishing, 1995.