Grundlagen Software Engineering

Entwurf

Entwurf
Inhalt

- Anwendungs-Kategorien
- Plattformen
- Entwurf der Fachkonzeptschicht
- Entwurf der GUI-Schicht und Anbindung an die Fachkonzept-Klassen
- Entwurf der Anbindung an die Datenhaltung

OOD-Architekturentwurf

- Zur Historie
  - Dr. Bertrand Meyer
    * 1950 in Paris
    Präsident der Firma ISE Inc.
    (Interactive Software Engineering)
    in Santa Barbara, USA
  - OOD-Pionier
  - Erfinder der Programmiersprache Eiffel (1985/86)
  - Buch *Object-Oriented Software Construction* 1988

OOD-Architekturentwurf

- Ziel der Definitionsphase
  - Produkt-Definition erstellen, die das Fachkonzept der gewünschten Anwendung in Struktur und Semantik aus Anwendersicht vollständig, konsistent und eindeutig beschreibt
  - Für ein OOA-Modell bedeutet dies
    - Es beschreibt die fachliche Lösung, nicht die technische Lösung
    - Es enthält keine Optimierungen
    - Es enthält keine Objektverwaltung
    - Alle Assoziationen, Aggregationen und Kompositionen sind in der Regel bidirektional

OOD-Architekturentwurf

- Ziel der Entwurfsphase
  - Produkt-Entwurf erstellen, der die Produkt-Anforderungen realisiert und die Anwendung architektonisch in die Anwendungs- und Plattformumgebung einbettet
  - Durchführung des Architekturentwurfs
    - Softwarearchitektur wird festgelegt
    - Hängt wesentlich von der Art der Anwendung und der verwendeten Plattform ab
  - Kein Strukturbruch
    - Da sowohl in OOA als auch in OOD die objektorientierten Konzepte verwendet werden, gibt es keinen »Strukturbruch« von OOA nach OOD

OOD-Architekturentwurf

- Start mit OOA
  - OOA-Modell wird erweitert, modifiziert, optimiert und an die umgebende Architektur angepasst
  - Es ergibt sich ein OOD-Modell
- Wiederverwendung
  - Vorteil der OO-Softwareentwicklung
    - Ihre Konzepte – insbesondere das Vererbungskonzept – unterstützen die Wiederverwendung vorhandener Klassen und Pakete
  - Beim gesamten Entwicklungsprozess
    - Suchen nach wiederverwendbaren Teilen und das Ablegen wiederverwendbarer Teile wesentlich

OOD-Architekturentwurf

- Wahl der Programmiersprache
  - Ein fertiges OOD-Modell muss in der Implementierungsphase realisiert werden
  - Die verwendete Sprache hat daher u. U. massive Rückwirkungen auf den Architekturentwurf
  - Aus heutiger Sicht sollte immer eine OO-Sprache wie Java oder C++ gewählt werden
- Komponentenmodelle
  - Die Wahl der Sprache beeinflusst auch die möglichen Komponenten-modelle
- Interdependenzen
  - Vielfältige Abhängigkeiten bei den Entscheidungen, so dass eine sorgfältige Analyse erforderlich ist

OOD-Architekturentwurf

- Mehrere Programmiersprachen
  - Bei komplexen Unternehmenslösungen ist oft auch der Einsatz mehrerer Sprachen sowie die Einbindung von Altsystemen mit deren Sprachen erforderlich
  - Insbesondere bei Web-Architekturen werden zusätzlich noch Skript-Sprachen verwendet
  - Am durchgängigsten kann heute sicher Java eingesetzt werden

Anwendungs-Kategorien

- Architekturentwurf
  - Wird wesentlich von der Anwendungs-Kategorie determiniert, in die die zu entwickelnde Anwendung fällt
  - Anwendungen lassen sich einer oder mehreren der folgenden Kategorien zuordnen
    - *Desktop*-Anwendung
    - Klassische *Client/Server*-Anwendung
    - Web-Anwendung
    - Angepasste Standard-Software
    - Mischform
    - Steuerung

Anwendungs-Kategorien
Desktop-Anwendungen

- Erlauben eine relativ einfache Systemarchitektur
  - Sie werden auf einem einzelnen Computersystem installiert
  - Es arbeitet meist nur ein Benutzer mit ihnen
- Drei-Schichten-Architektur erforderlich
- Abnehmende Bedeutung
  - Der anonyme Massenmarkt ist die traditionelle Domäne von *Desktop*-Anwendungen
  - Er wird zum großen Teil von wenigen großen Software-Häusern dominiert
  - Hinzu kommt, dass mit Software für Privat-Personen immer weniger Geld zu verdienen ist

Anwendungs-Kategorien
Klassische *Client/Server*-Anwendungen

- Je eine eigenständige Anwendung auf dem *Client* und auf dem *Server*, die über ein Netzwerk miteinander kommunizieren
- Nutzen des jeweiligen Betriebssystems
  - *Client*-Seite dominiert derzeit *Windows*
  - Im *Server*-Bereich *Windows* und UNIX-Derivate
- TCP/IP-Protokoll
- Oft ist es sinnvoll, eine (komponentenbasierte) Verteilungs-Plattform einzusetzen
  - CORBA (Common Object Request Broker Architecture, OMG)
  - EJB (Enterprise Java Beans, SUN)
  - Früher COM+ ; jetzt .NET (Microsoft)

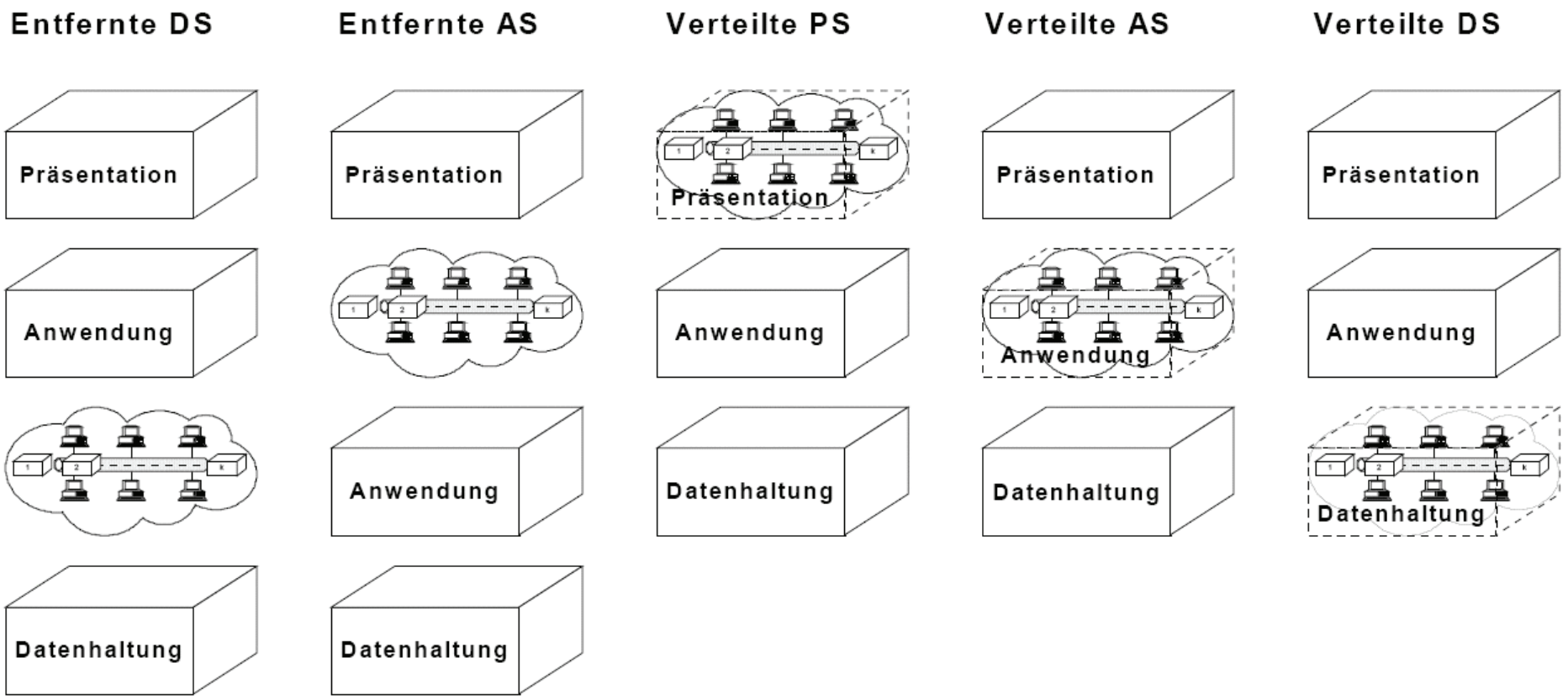
Verteilung und physische Schichtung

- Die logischen Schichten können unterschiedlich auf mehrere Rechner verteilt werden. Optionen sind
  - Horizontaler Schnitt zwischen zwei Schichten
  - Horizontaler Schnitt innerhalb einer Schicht
  - Vertikaler Schnitt innerhalb einer Schicht
- Gründe für und gegen eine Verteilung
  - \+ Ortsgebundene Funktionalität
    - \+ Präsentation
    - \+ Rechtliche Aspekte
  - \+ Performanzgewinn durch Parallelisierung
  - \+ Erhöhte Zuverlässigkeit
    - \+ Spiegelung, Clustering
    - \+ Weiterarbeit nach Teilausfällen
  - \+ Integration der Funktionalität vorhandener Systeme

Verteilung und physische Schichtung

- Performanzverlust durch Kommunikation
  - Erhöhung der Antwortzeit
  - Netzwerk als Flaschenhals
- Verringerte Zuverlässigkeit
  - Einsatz komplexer Middleware
  - Beherrschung verteilter Abläufe
- Test aufwändig
- Installation und Betrieb aufwändig

Horizontaler Schnitt zwischen *Client* und *Server*



Anwendungs-Kategorien
Web-Anwendungen

- Immer beliebter
  - *Web-Server* ohnehin meist vorhanden
  - Internet soll zur Verbesserung der eigenen Geschäftsprozesse genutzt werden, z. B. für die Kommunikation mit Händlern und Kunden
  - *Web-Browser* ist Mitarbeitern & Kunden bekannt
  - Hemmschwelle zur Nutzung ist dadurch geringer
  - Bei kleinem Funktionsumfang
    - *Server*seitige Web-Konzepte (z. B. *Servlets*, JSP, ASP) und ein Datenbanksystem ergeben gute, skalierbare und leicht wartbare Anwendungen
  - Steigender Funktionsumfang
    - Einsatz einer Verteilungsplattform
  - *Servlet: machen serverseitigen Code für webbasierte Clients nutzbar; Sun stellt ein entsprechendes API zur Verfügung*
  - *JSP: Java Server Pages, SUN; ASP: Active Server Pages (Microsoft)*

Web-Anwedungen
Server-seitige Ansätze

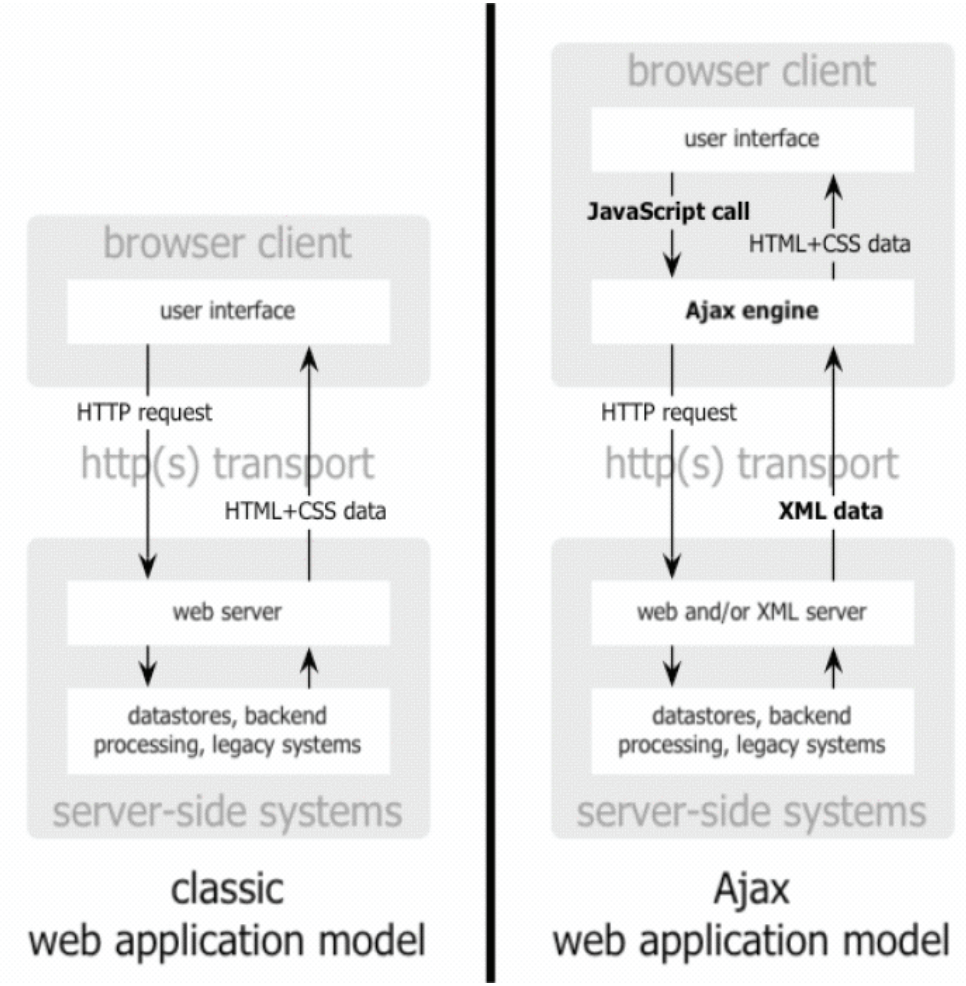
- Generierung statischer HTML-Seiten
- Common Gateway Interface und CGI-Skripte
- Neuere Web Server APIs (z.B. Servlets)
- Server Side Includes (SSI)
- Server-seitige Skripte
  - In HTML-Seiten werden zusätzliche *HTML-generierende Quellen* integriert
  - Typische Vertreter
    - PHP - Personal Home Page
    - ASP - Active Server Pages (Microsoft)
    - Server Side JavaScript (Netscape)
    - JSP - Java Server Pages (SUN)

Web-Anwendungen
Client-seitige Ansätze

- Aus Gründen der Lastverteilung können Aufgaben am *Client* ausgeführt werden
  - Prüfung von Benutzereingaben
  - Erstellung von Graphiken
  - Manipulation von WWW-Dokumenten
  - Zusätzliche Möglichkeit für die Gestaltung der Benutzerschnittstelle
- Vertreter
  - Browser Plug-ins (z.B. *Macromedia Flash-Player*)
  - Skriptsprachen (JavaScript,...)
  - Java Applets
  - *Microsoft ActiveX*

Web-Anwendungen
Neue Ansätze - AJAX

- *Asynchronous JavaScript and XML* (AJAX)
- Browser als *Thin-Client*
  - JavaScript als Verarbeitungssprache
  - XHTML + CSS + DOM als GUI-Framework
  - Datenrepräsentation und Manipulation mit XML + XSLT
  - Datenaustausch über Format XMLHttpRequest
- Vorteile
  - Asynchron, Benutzer kann also weiterarbeiten während er bei klassischen Web-Anwendungen warten muss



Anwendungs-Kategorien
Angepasste Standard-Software

- *Microsoft Office*
  - Auch eine Entwicklungsplattform für neue Anwendungen
- Baukastenprinzip
  - Fertige Einzelteile werden zu einer fertigen Anwendung zusammengesetzt
  - Der Architekturentwurf wird in diesem Fall sehr stark durch die Struktur von *Office* geprägt
- Nur Plattformen *Windows* und *Apple Macintosh*
- Kostenreduktion
- Verteilungsplattform
- Gefahr groß, ohne Analyse und Entwurf einfach »drauf los zu *hacken*«

Anwendungs-Kategorien
Mischformen

- Für jeden Teilbereich kann der jeweils beste Lösungsansatz verwendet werden
  - Erfassung von Massendaten
    - Klassische *Client*-Anwendung, da die Bedienungsgeschwindigkeit oder eine vollständige Tastaturbedienung eine wichtige Rolle spielen
  - Kundenzugriff über das Internet auf firmeninterne Daten
    - Web-Anwendung
  - Um Adressen aus der unternehmensweiten Kundendatenbank automatisch in einen Musterbrief einzubetten
    - Keine eigene Textverarbeitung, sondern man passt z. B. *Word* entsprechend an

Plattformen

- *Microsoft* vs. »Rest der Welt«
- *Microsoft*
  - Einziger Anbieter, der für alle Kategorien Produkte anbietet und auch ein Datenbanksystem im Programm hat
  - Grundsätzlich können alle Produkte »aus einer Hand« gekauft werden
  - \+ Alle Produkte (inklusive des Betriebssystems) sind relativ gut aufeinander abgestimmt
  - \+ Inkompatibilitäten treten seltener auf
  - \+ Außerdem bietet *Microsoft* eine gute technische Unterstützung für alle Produkte an und ist schon aus eigenem Interesse an einer leichten Integration seiner Produkte interessiert

Plattformen

- *Microsoft* geht in einigen Bereichen eigene Wege
- Java spielt bei *Microsoft* nur eine untergeordnete Rolle
  - EJBs, *JavaBeans* oder *Servlets* werden durch hauseigene Konzepte ersetzt
- Außerdem ist man von einem einzelnen Hersteller und dessen Produkt- und Preispolitik abhängig
- In eine reine *Microsoft*-Lösung später Produkte anderer Hersteller zu integrieren, kann sehr schwierig werden

Plattformen

- Verschiedene Hersteller
  - Alternative: Mischung aus Produkten und Konzepten verschiedener Hersteller
  - Java hat sich als Sprache inzwischen fest etabliert
  - Standards auf Java: EJBs und *Servlets*
- Datenaustausch
  - Immer mehr Standards, die auf XML-basierten Sprachen beruhen
    - z. B. X-EDI *(EDIFACT: Electronic Data Interchange for Administration, Commerce and Transport)* für B2B-Anwendungen
  - XML (E**x**tension **M**arkup **L**anguage): Von *Microsoft* und von den meisten anderen Herstellern als Meta-Sprache favorisiert
- Offene Standards
  - Trend immer mehr zu offenen Standards

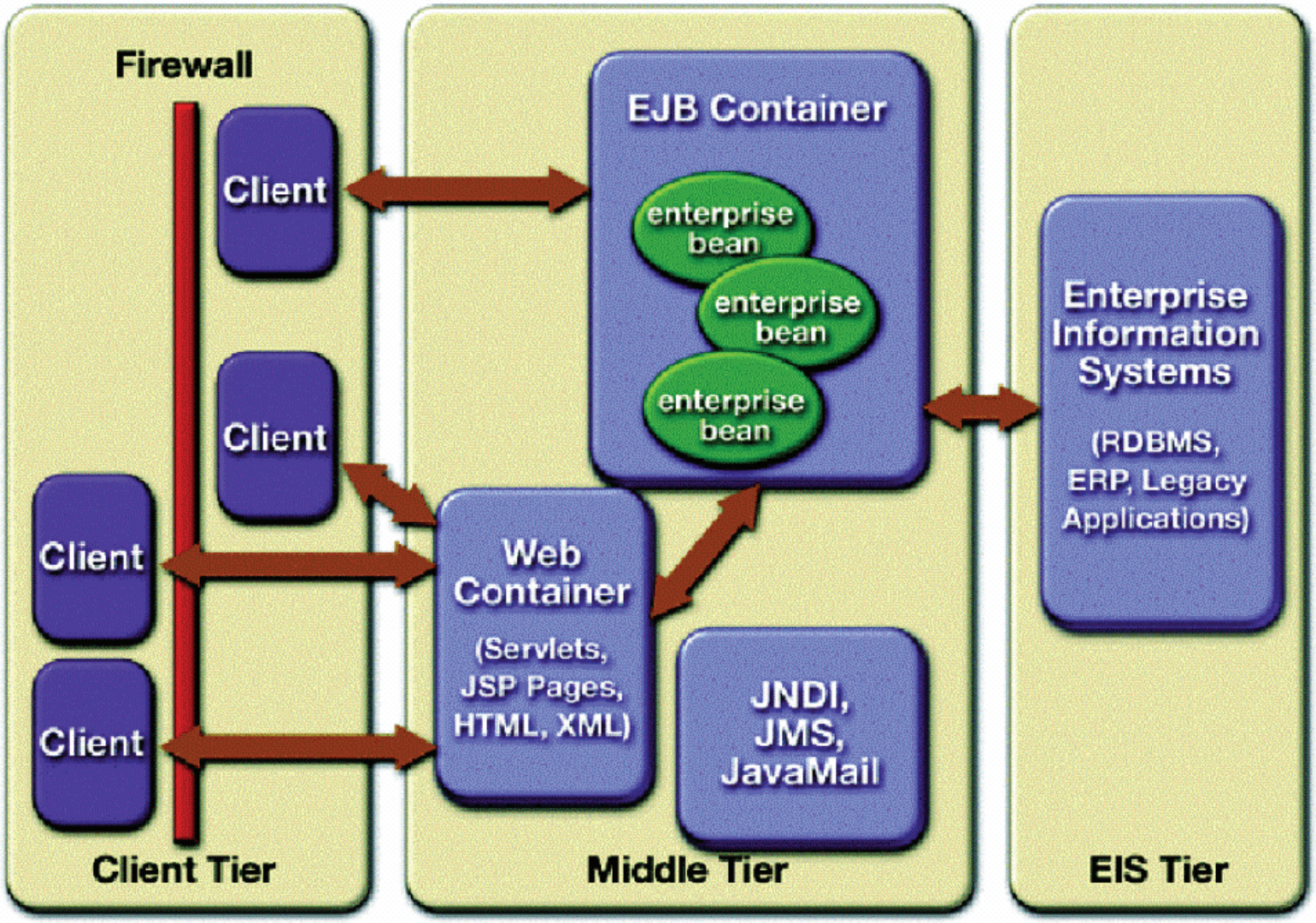
Verbreitete Plattformen (*middleware)*

- Datenbankkommunikationsprodukte (ODBC, JDBC, …)
- Transaktionsmonitore
  - CICS (IBM Mainframes)
- Nachrichtenorientierte
  - CORBA Event Service, CORBA Notification Service
  - Java Messaging Service (JMS)
  - Microsoft Message Queue
  - IBM MQSeries
- Verteiltes Objektmanagement
  - Microsoft .Net
  - OMGs CORBA
  - J2EE Application Server (EJB-Container,…)

Plattformen
.NET

- .NET ist die aktuelle Programmier-Plattform von Microsoft
- .NET fasst Betriebssystem-Funktionen zusammen und bietet diese zentral an
- .NET ersetzt ältere Technologien (z.B. COM, API-Aufrufe)
- Microsoft entwickelt .NET als Realisierung des Common Language Infrastructure-Standards (CLI)
- Diese besteht neben einer Laufzeitumgebung aus einem Framework von Klassenbibliotheken und aus angeschlossenen Diensten, die gemeinsam eine Basis für die Software-Entwicklung bieten
- .NET setzt **NICHT** auf Java, sondern auf C#

Plattformen
J2EE

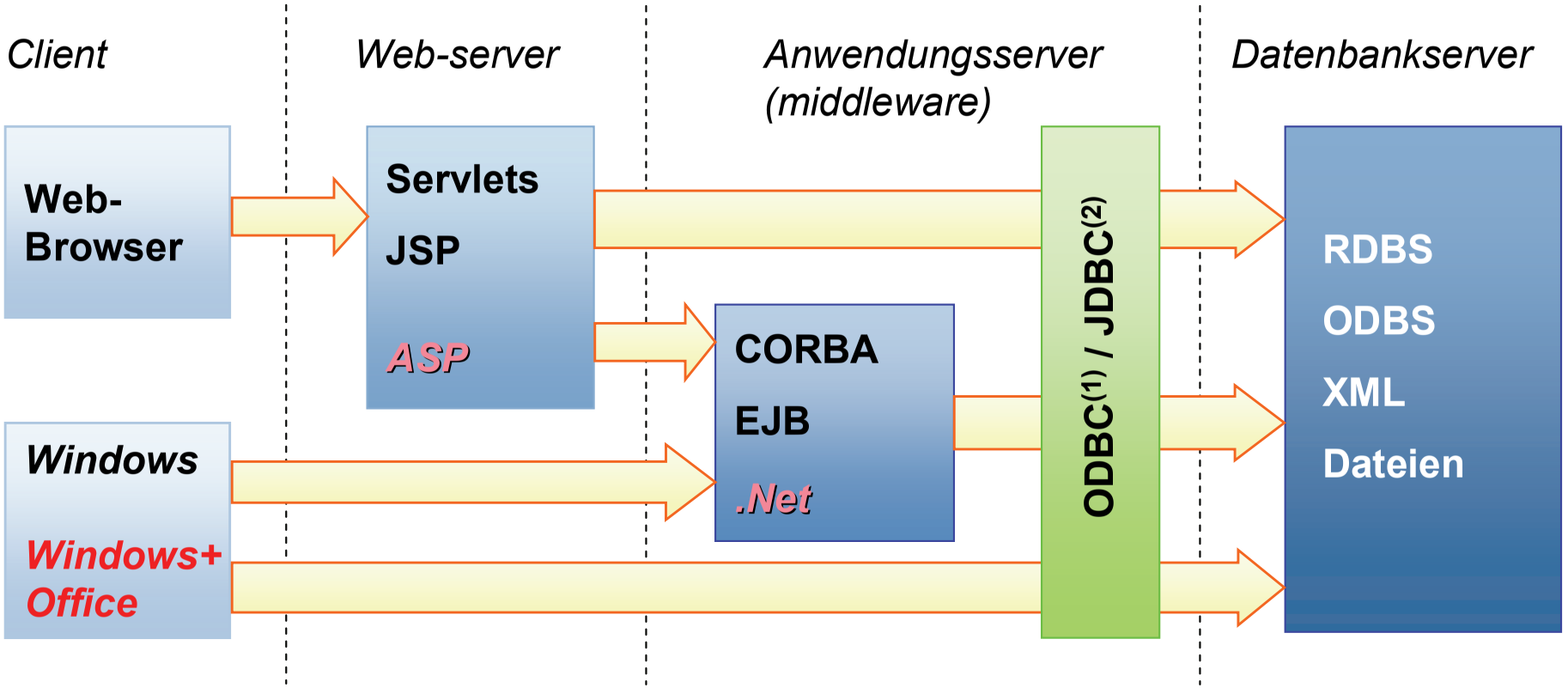


Plattformen
J2EE

- EJB-Container sind fertige Produkte, die von verschiedenen Herstellern angeboten werden und sich gemäß Sun‘s EJB-Spezifikation verhalten
- EJB-basierte Serverprodukte verwenden oft CORBA (IIOP) , um mit nicht-Java-Objekten zu kommunizieren
- Internet Inter-ORB Protokoll (IIOP) ist Teil der EJB-Spezifikation
  - IIOP ist GIOP (General Inter-ORB Protokoll) auf der Basis von TCP/IP
  - Kommunikation zwischen Object Request Brokers (CORBA) basiert auf GIOP
- CORBA und EJB sind komplementäre Techniken
- Verbreitete EJB-Container
  - Bea Weblogic
  - IBM Websphere
  - JBoss (open source)

Plattformen

- Konzepte für Mehrschichtige Architekturen im Überblick



(1) Open Database Connectivity (Microsoft, Rel. DB)
(2) Java Database Connectivity (Java, Rel. DB)

Model Driven Architecture MDA

- Modellierung verteilter Anwendungssysteme
  - Fokus auf Funktion und Verhalten des Systems
  - Trennung von Implementierungsdetails und Funktionalität
  - Funktionalität wird nur einmal implementiert
- Modell
  - Eine Darstellung von Funktionen, Struktur und Verhalten eines Systems
- Plattform
  - Software-Infrastruktur, die mittels spezifischer Techniken (Unix, Windows, CORBA) auf bestimmter Hardware implementiert ist
  - Der Begriff *Plattform* beschreibt technische Details, die für die Beschreibung der grundlegenden Funktionalität einer Softwarekomponente irrelevant ist
- MDA besteht aus
  - Einem Plattform-unabhängigen UML Modell (PIM)
  - Ein oder mehrere Plattform-spezifische Modelle (PSM) in UML

Entwurf der Fachkonzeptschicht

- Ausgangspunkt OOA-Modell: 1. Version der Fachkonzeptschicht, die unter den Aspekten des Entwurfs verfeinert und überarbeitet wird

1. Modifikation der Klassenstruktur
   - Objektverwaltung durch Container-Klassen
   - Analyseklassen 1:1 in die Fachkonzeptschicht übernehmen
     - Falls Komplexität einer Klasse zu hoch Teilaufgaben an detailliertere Klassen delegieren
     - *Performance* sicherstellen
       - Klassen mit starker Interaktion – d. h. mit einer hohen Kopplung – zusammenfassen
     - Hinzufügen weiterer Klassen
       - Z. B. um Zwischenergebnisse zu modellieren, d. h. mehrere abgeleitete Attribute in einer neuen Klasse zu »bündeln«
     - Assoziative Klassen in »normale« Klassen auflösen

Entwurf der Fachkonzeptschicht

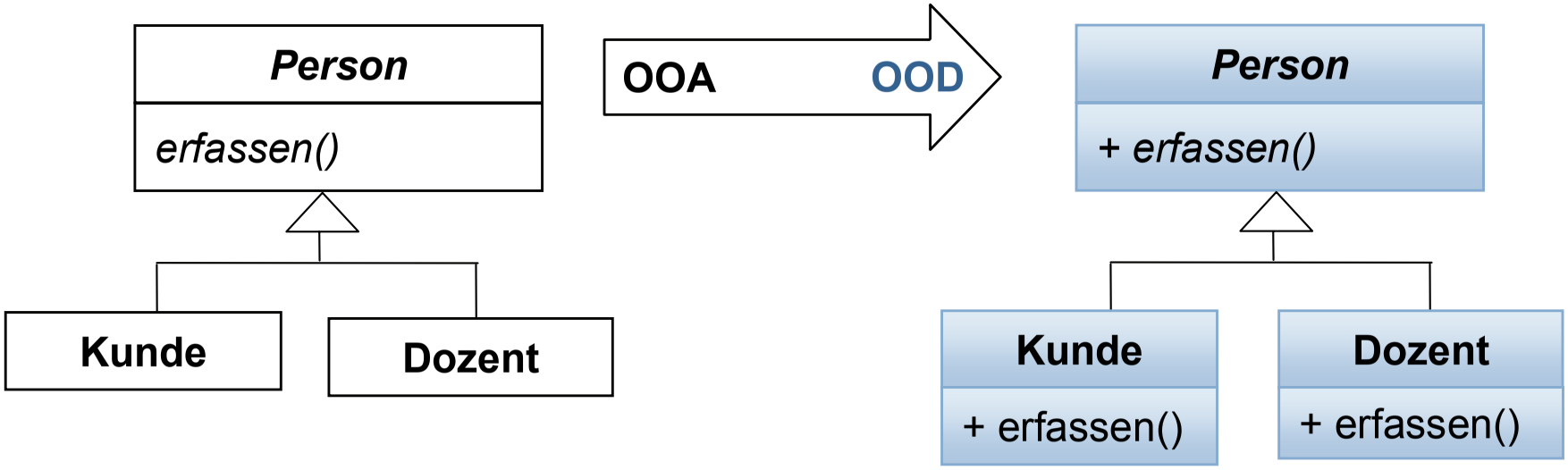
2. Verfeinern der Attribute
   - Für »abgeleitet« Attribute prüfen, ob die Werte zu speichern sind oder ob sie jeweils aktuell berechnet werden sollen
     - Handelt es sich um viele Attribute, so ist es sinnvoll, dafür eine neue Klasse in das Modell einzufügen
3. Verfeinern der Operationen
   - Spezifizierte Operationen sind aus Entwurfssicht detaillierter zu beschreiben
   - Komplexe Operationen sind in Teiloperationen zu gliedern
   - Besitzt die Klasse einen Lebenszyklus, so ist eine auszuführende Operation von dem jeweiligen Objektzustand abhängig
     - Dann muss der Algorithmus entsprechende Abfragen enthalten oder es ist das Zustandsmuster anzuwenden

Entwurf der Fachkonzeptschicht

4. Verfeinern von Assoziationen
   - Prüfen, ob eine Navigationsrichtung ausreicht
     - Richtung im Klassendiagramm durch einen Pfeil kennzeichnen
   - Assoziationen unter dem Gesichtspunkt des optimalen Zugriffs auf Objekte modellieren
   - Für jede Operation prüfen, welche Assoziationen sie »durchlaufen« muss, um an die benötigten Informationen zu gelangen
   - Beispiel
     - Assoziation zwischen Klassen A & B wird nur von A nach B als Zeiger in A implementiert
       - Bei einem Zugriff in der Gegenrichtung müssen alle Objekte von A betrachtet und gefiltert werden

Entwurf der Fachkonzeptschicht

5. Verfeinern der Vererbung
   - Beispiel
     - Im OOA-Modell
       - `erfassen()` gilt für alle Objekte ihrer Unterklassen
     - Im OOD-Modell
       - Erfassen bei beiden Unterklassen unterschiedlich
       - Jede Unterklasse enthält diese Operation



Entwurf der Fachkonzeptschicht

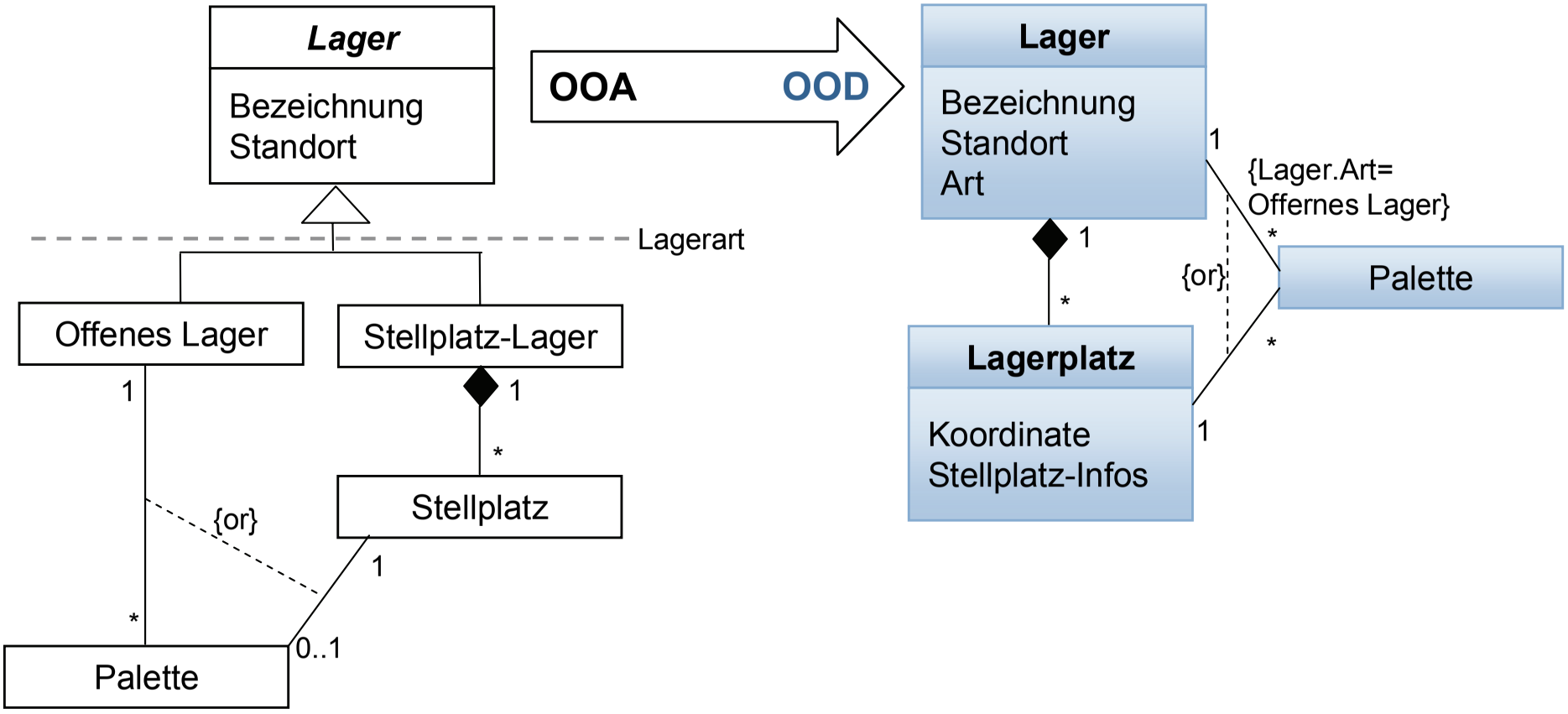
- Abstrakte Klassen
  - Werden stets künstlich in das Modell eingefügt
  - Möglichst viele abstrakte Klassen, weil dadurch das Hinzufügen neuer Klassen erleichtert wird
  - Beispiel
    - OOA-Modell mit Klassen `Rechteck`, `Ellipse` und `Linie` (mit entsprechenden Unterklassen), die gleichartige Operationen besitzen (z. B. `verschieben()`, `duplizieren()`, `vergrößern()`)
    - Gemeinsamkeiten durch eine abstrakte Oberklasse `Grafikobjekt` beschrieben
- Keine »Spaghetti«-Vererbung
  - Der Klassenname besitzt keine Aussagefähigkeit oder steht in keiner Beziehung zu den Attributen und/oder Operationen der Klasse

Entwurf der Fachkonzeptschicht

- Maximaler Polymorphismus
  1. Alle Operationen von Unterklassen so hoch wie möglich in der Vererbungshierarchie unterbringen
  2. Operationsnamen so wählen, dass ein einziger Name für konzeptionell gleiche Operationen verwendet wird, z. B. `drucken()` oder `erfassen()`
  3. Alle Operationen sind in der Schnittstelle so allgemein wie möglich zu halten
     - Dazu ist zu überlegen, welche Änderungen evtl. an dem System vorgenommen werden können

Entwurf der Fachkonzeptschicht

- Komprimieren
  - Vererbungsstruktur wieder zu einer Klasse zusammenfügen



Entwurf der GUI-Schicht und Anbindung

- Abgrenzung Fachkonzept – GUI
  - Faustregel: Alle Objekte, die unabhängig von dem verwendeten GUI dargestellt werden, zählen zum Fachkonzept
- GUI-Schicht
  - Die Architektur der Benutzungsoberfläche wird durch das verwendete GUI-System geprägt
  - Eine GUI-Bibliothek besteht meistens aus einem oder mehreren größeren Bäumen
    - Für jedes Interaktionselement gibt es eine Blattklasse
    - Für die Fenster gibt es eine Oberklasse, von der dann die individuellen Fenster abgeleitet werden.

Entwurfskriterien
Generelle Entwurfskriterien

- Architekturen von Softwaresystemen werden – neben ggf. existierenden speziellen Zielsetzungen – anhand der Kriterien *Bindung* und *Kopplung* bewertet.
- Bindung und Kopplung sind methodenunabhängige Kriterien, d.h. sie gelten z.B. für nicht objektorientierte Architekturen genauso wie für objektorientierte Architekturen.

Bindung und Kopplung

- Die **Bindung** (cohesion) innerhalb einer Systemkomponente und die Kopplung der Systemkomponenten untereinander bestimmen die Struktur eines Software-Systems. Bindung (cohesion) ist ein qualitatives Maß für die Kompaktheit einer Systemkomponente. Es werden dazu die Beziehungen zwischen den Elementen einer Systemkomponente betrachtet
- **Kopplung** (coupling) ist ein qualitatives Maß für die Schnittstellen zwischen den Systemkomponenten. Es werden der Kopplungsmechanismus, die Schnittstellenbreite und die Art der Kommunikation betrachtet

Bindung und Kopplung

- Die **Bindung** (cohesion) innerhalb einer Systemkomponente und die Kopplung der Systemkomponenten untereinander bestimmen die Struktur eines Software-Systems. Bindung (cohesion) ist ein qualitatives Maß für die Kompaktheit einer Systemkomponente. Es werden dazu die Beziehungen zwischen den Elementen einer Systemkomponente betrachtet
- Die Beziehungen zwischen den Elementen einer Systemkomponente sollen möglichst ausgeprägt sein. Die Systemkomponenten einer guten Architektur besitzen eine **starke** Bindung.
- Die Bindung (B) ist ein Maß, das sich auf eine einzelne Systemkomponente (M) bezieht, d.h. **B = f(M).**

Entwurfskriterien
Bindung und Kopplung

Bindung und Kopplung

- **Kopplung** (engl. coupling) ist ein Maß für die Stärke der Beziehung zwischen zwei betrachteten Systemkomponenten.
- Es werden der sogenannte Kopplungsmechanismus, die Schnittstellenbreite und die Art der Kommunikation betrachtet.
- Die Kopplung soll möglichst **schwach** sein.
- Die Kopplung (K) bezieht sich auf zwei betrachtete Systemkomponenten (M1, M2), d.h. **K = f(M1, M2).**

Entwurfskriterien
Bindung und Kopplung

Bindung und Kopplung

- Das Ziel des Entwurfs ist es eine Architektur so zu konzipieren,
  - dass die Bindung der Systemkomponenten möglichst hoch ist und
  - die Kopplung zwischen den Systemkomponenten möglichst gering ist.
- Kopplung und Bindung sind zwei Entwurfskriterien, die einander unterstützen. Eine Stärkung der Bindung führt zwangsläufig zu einer Schwächung der Kopplung. Es ist daher im Grunde nicht erforderlich einen Kompromiss zu erreichen.
- Verletzungen der Bindungs- und Kopplungskriterien können jedoch fachliche Gründe besitzen (z.B. bessere Verständlichkeit der Struktur).
- Eine starke Kopplung und schwache Bindung begünstigt z.B. die Wartbarkeit und Wiederverwendbarkeit.
- Die Überprüfung von Bindung und Kopplung sollte in jedem Fall Bestandteil der Qualitätssicherung einer Architektur sein.

Entwurfskriterien
Bindung

Bindung

- Eine gute Bindung liegt vor, wenn nur solche Elemente zu einer Einheit zusammengefasst werden, die auch zusammen gehören
- Die folgenden Bindungen werden unterschieden (in aufsteigender Stärke)
  1. zufällige Bindung
  2. logische Bindung
  3. zeitliche Bindung
  4. prozedurale Bindung
  5. kommunikative Bindung
  6. sequentielle Bindung
  7. funktionale Bindung

Entwurfskriterien
Bindung

Bindungsarten

- **Eine sequentielle Bindung** liegt vor, wenn mehrere Teilfunktionen, die hintereinander ausgeführt werden, zu einer Systemkomponente zusammengefasst werden. Sequentiell gebundene Systemkomponenten können offensichtlich nicht wiederverwendet werden, wenn eine andere Ausführungssequenz erforderlich ist.
- **Eine kommunikative Bindung** liegt vor, wenn Teilfunktionen aufgrund von Kommunikationsbeziehungen zu einer Systemkomponente zusammengefasst werden.
- Bei einer **temporalen Bindung** ist der gemeinsame Ausführungszeitpunkt entscheidend für die Bildung der Systemkomponente (z.B. Realisierung einer Komponente, die alle Initialisierungen durchführt).
- Einer **zufälligen Bindung** liegt kein erkennbares Prinzip zugrunde.

Entwurfskriterien
Bindung

Bindung

- Ziel: Erreichen einer funktionalen Bindung
  - Alle Elemente sind an der Verwirklichung einer einzigen, abgeschlossenen Funktion beteiligt
  - Komplexe Funktionen werden realisiert, indem importierte Funktionen verwendet werden, die selbst funktional gebunden sind
- Kennzeichen einer funktionalen Bindung
  - Alle Elemente tragen dazu bei, ein einzelnes spezifisches Ziel zu erreichen
  - Es gibt keine überflüssigen Elemente
  - Die Aufgabe kann mit genau einem Verb und genau einem Objekt beschrieben werden
  - Austausch gegen anderes Element, welches denselben Zweck erfüllt, leicht möglich
  - Hohe Kontextunabhängigkeit, d.h. einfache Beziehungen zur Umwelt

Bindung

- Vorteile einer funktionalen Bindung
  - Hohe Kontextunabhängigkeit (die Bindungen befinden sich innerhalb der Prozedur, nicht zwischen Prozeduren)
  - Geringe Fehleranfälligkeit bei Änderungen
  - Hoher Grad der Wiederverwendbarkeit
  - Leichte Erweiterbarkeit und Wartbarkeit, da sich Änderungen auf isolierte, kleine Teile beschränken
- Konzept der Bindung verallgemeinert die Regeln für „guten Code“ zu Regeln für „guten Software-Entwurf“
- Die Bindungsart einer Prozedur lässt sich nicht automatisch ermitteln, sondern nur durch manuelle Prüfmethoden

Funktionale Bindung

- Bei einer funktionalen Bindung tragen alle Elemente dazu bei, ein einzelnes spezifisches Ziel zu erreichen. Diese Funktionalität kann daher mit genau einem Verb und genau einem Substantiv beschrieben werden:
  - „Drucke Bericht“,
  - nicht „Drucke und versende Bericht“,
  - ebenfalls nicht „Drucke Bericht und Deckblatt“.
- Sicherheitskritische Funktionen können geeignet gekapselt und vom Rest des Systems getrennt werden => aufwändige Techniken für den Umgang mit sicherheitskritischer Funktionalität müssen nur auf jene Komponenten angewendet werden, die dieses zwingend erfordern. Unkritischere Komponenten können abgetrennt und mit konventionelleren Techniken behandelt werden.

Bindung

- Bindung von Datenabstraktionen/Klassen
  - Beschreibt das Zusammenwirken verschiedener Funktionen, welche derselben Datenabstraktion oder Klasse zuzuordnen sind.
    Voraussetzung: Alle Methoden sind funktional gebunden
  - Gute Bindung einer Klasse (model cohesion) liegt vor, wenn
    - sie ein einzelnes semantisch bedeutungsvolles Konzept repräsentiert
    - die Klasse keine verborgenen Klassen enthält und
    - keine Operationen enthält, die an andere Klassen delegiert werden können
  - Wird in der Literatur auch als Kohärenz bezeichnet.
  - Für Klassen ist weiter die Bindung innerhalb von Vererbungsstrukturen wesentlich

Entwurfskriterien
Bindung

Informale Bindung

- [Myers 78] fordert für abstrakte Datenobjekte informale Bindung.
  - Diese liegt vor, wenn mehrere, in sich abgeschlossene, funktional gebundene Zugriffsoperatoren, die zu einer Datenabstraktion gehören, auf einer einzigen Datenstruktur operieren
  - Idee: hinter der gemeinsamen Funktionalität liegt auch ein gemeinsames Datenmodell
- Merkmale
  - Unterstützt das Geheimnisprinzip, d.h. die Datenstruktur gehört nur zu einer Datenabstraktion
  - Änderungen der Datenstruktur tangieren nur eine Datenabstraktion
  - Problem der Vermischung von Zugriffsoperationen, da alle auf derselben Datenstruktur operieren

Entwurfskriterien
Bindung

Informale Bindung

- Kann auch als funktionale Bindung betrachtet werden.
- Dies ist dann gegeben, wenn der Zweck der Datenabstraktion die Bereitstellung einer einzelnen, definierten Dienstleistung ist – z.B. „Lese Antriebsparameter“.
- Es kann funktionale Bindungen auf allen Abstraktionsebenen eines Systems geben.
- Funktional gebundene Systemkomponenten auf höheren Abstraktionsebenen sollten funktional gebundene Komponenten enthalten.
- Zusätzlich können Daten hinzutreten, die zur Erledigung der Aufgabe nötig sind.

Entwurfskriterien
Bindung

- Beispiel: Die Komponente „Lese Antriebsparameter“ enthält die funktional gebundenen Komponenten „Lese Temperatur“, „Lese Öldruck“, „Lese Ladestrom“ und die Daten „Antriebsparameter“:
  - Einerseits verhindert es nicht die funktionale Bindung, dass „Lese Antriebsparameter“ mehrere Funktionen und zugehörige Daten enthält, falls „Lese Antriebsparameter“ ansonsten eine abgeschlossene Funktion erbringt.
  - Andererseits ergibt nicht jede Zusammenfassung von funktional gebundenen Komponenten ein funktional gebundenes übergeordnetes System:
    - So wird „Lese Antriebsparameter und stelle diese dar“ nicht funktional gebunden sein, obwohl die beteiligten Komponenten durchaus funktional gebunden sein können.
    - Eine geeignete Lösung könnte die Auftrennung in die zwei funktionalen Bindungen „Lese Antriebsparameter“ und „Stelle Antriebsparameter dar“ sein.

Entwurfskriterien
Bindung

Bindung in Vererbungsstrukturen

- Die ganze Vererbungshierarchie muss untersucht werden
- Starke Vererbungsbindung liegt vor, wenn die Hierarchie eine Generalisierungs-/Spezialisierungshierarchie im Sinne der konzeptuellen Modellierung ist
- Schwache Vererbungsbindung liegt vor, wenn die Hierarchie nur zum "code sharing" verwendet wird
- Das Ziel jeder neu definierten Unterklasse muss sein, ein einzelnes semantisches Konzept auszudrücken

Entwurfskriterien
Kopplung

- Kopplung ist ein Maß für die Stärke der Interaktion zweier betrachteter Komponenten.
- Die Forderung zu einer starken Bindung führt zwangsläufig zu einer schwachen Kopplung.
- Eine schwache Kopplung gewährleistet eine hohe Unabhängigkeit der Systemkomponenten voneinander. Dies begünstigt die Wartbarkeit, den Austausch von Komponenten aber auch alle Qualitätssicherungsaktivitäten, z.B. den Test.
- Die schwächste und daher beste Kopplung ist die so genannte Datenelementkopplung. Diese liegt dann vor, wenn die Schnittstelle zweier Komponenten nur elementare Daten (d.h. keine Datenstrukturen) und keine Steuerinformation in Vorwärtsrichtung enthält.
- Daten werden selbst verarbeitet, während Steuerinformationen in Vorwärtsrichtung die Verarbeitung steuern, d.h. nicht selbst verarbeitet werden:
  - Einer Komponente „Drucke oder zeige Öldruck an“ muss mitgeteilt werden, ob der Öldruck gedruckt oder angezeigt werden soll. Dieser Schnittstellenparameter ist Steuerinformation in Vorwärtsrichtung, weil er die Verarbeitung steuert, d.h. in der Komponente die auszuführenden Teile bestimmt. Dieser Steuerparameter ist erforderlich, weil die Komponente nicht funktional gebunden ist. Durch Auftrennung der Komponente in die zwei Komponenten „Drucke Öldruck“ und „Zeige Öldruck an“ entstehen funktionale Bindungen. Jede Komponente kann für sich aufgerufen werden. Der Steuerparameter ist nicht erforderlich. Steuerinformationen in Rückwärtsrichtung ist akzeptabel insofern es sich um Status-Rückmeldungen handelt. So kann die Komponente „Drucke Öldruck“ z.B. eine Statusrückmeldung (ok, Papier, Toner, ausgeschaltet) zurückliefern.

Entwurfskriterien
Kopplung

- Eine **Datenstrukturkopplung** liegt vor, wenn strukturierte Daten übergeben werden. Dies bezieht sich nicht auf vordefiniert strukturierte Datentypen – z.B. „string“. Diese sind erlaubt und werden als elementare Daten betrachtet. Vielmehr führen selbst definierte strukturierte Daten zu einer Datenstrukturkopplung. Es ist erforderlich, dass die Definition der Datenstruktur den interagierenden Komponenten bekannt ist. Falls Änderungen dieser Definition erforderlich sind, so müssen diese konsistent in allen betroffenen Komponenten durchgeführt werden. Die interagierenden Komponenten verlieren daher an Unabhängigkeit. Fehler werden begünstigt; die Wartung und Weiterentwicklung wird erschwert.
- Eine **Hybridkopplung** liegt vor, wenn zusätzlich zu den zu verarbeitenden Daten Steuerinformationen in Vorwärtsrichtung übergeben werden (siehe Beispiel „Drucke oder zeige Öldruck an“) . Hybridkopplungen treten oft als Folge einer schlechten Bindung auf, weil mehrere Funktionen in einer Komponente zusammengefassst wurden, zwischen denen mit Hilfe von Steuerinformation ausgewählt werden muss.

Entwurfskriterien
Kopplung

- Eine **Externe Kopplung** liegt vor, wenn Komponenten über globale Daten miteinander kommunizieren. Jede Änderung der globalen Daten muss konsistent in allen Komponenten berücksichtigt werden. Derartige Strukturen begünstigen Fehler und führen im Extremfall zu kaum noch wartbaren Systemen.
- Eine **Inhaltskopplung** liegt vor, wenn zur korrekten Funktion einer Komponente Details der Realisierung einer anderen Komponente notwendig sind. Durch Modifikationen von Komponenten können Fehlverhalten an weit entfernter Stelle verursacht werden. Auswirkungen von Modifikationen können bei dieser Art der Kopplung systemweit auftreten.

Entwurfskriterien
Bindung und Kopplung

Qualitätssicherung

- Übergeordnete Ziele: Erreichung einer starken Bindung und einer schwachen Kopplung => Überprüfung, ob die Komponenten auf jeder Abstraktionsebene funktional gebunden sind und ob ausschließlich Datenelementkopplungen vorliegen.
- Falls das nicht erfüllt ist, so ist die Notwendigkeit der vorliegenden Abweichungen zu diskutieren. Ein typisches Beispiel ist die Schaffung einer Initialisierungskomponente, die alle Systemfunktionen in definierte Startzustände versetzt. Diese zeitliche Bindung mag der funktionalen Bindung an dieser Stelle aufgrund des Wunsches, alle Initialisierungen an einer Stelle durchzuführen, vorgezogen werden.

Entwurfskriterien
Bindung und Kopplung

Qualitätssicherung

- Die folgenden Aspekte sollten bei der Beurteilung des Entwurfs beachtet werden:
  - Sind die Komponenten funktional gebunden, d.h. kann das Verhalten jeder Komponente durch ein Verb und ein Substantiv beschrieben werden? Gegebenenfalls ist durch Auftrennung und neues Zusammenfassen eine funktionale Bindung erreichbar. Gibt es wichtige Gründe für nicht funktional gebundene Komponenten?
  - Liegen ausschließlich Datenelementkopplungen vor? Hybridkopplungen können oft durch eine Auftrennung der dienstanbietenden Komponente beseitigt werden. Externe Kopplungen können durch Realisierung einer Datenabstraktion beseitigt werden. Inhaltskopplungen erfordern i.allg. umfassende Änderungen.

Entwurf: Anbindung an die Datenhaltung

- Anbindung an die Datenhaltung hängt im Wesentlichen davon ab, ob die Datenhaltung durch
  - eine objektorientierte Datenbank
  - eine relationale Datenbank
  - ein Dateisystem

  erfolgt

Entwurf: Anbindung an die Datenhaltung
Anbindung an OO-Datenbank

- Relativ einfach
- Beispiel: Java-Sprachanbindung des ODBS Poet
  - Alle Klassen, deren Objekte durch das ODBS persistent gemacht werden sollen, müssen in der Konfigurationsdatei `ptjavac.opt` aufgeführt werden
  - Die Klasse *Artikel* wird wie folgt persistent

```
//ptjavac.opt
[classes\Artikel]
persistent = true
```