JAVA EE
ARCHITEKTUR & DESIGN

CAS Enterprise Application Development Java EE

Simon Martinelli, 02.2015
simon@martinelli.ch | about.me/simas_ch

INHALT

1 ARCHITEKTUR

DER WERT DER ARCHITEKTUR

- Architektur ist das Fundament einer Applikation

Erfolgreiche Architektur

Die Cheops Pyramide, ca. 2680 v. Chr

Weniger erfolgreiche Architektur

Der Schiefe Turm von Pisa, 1173

DEFINITION

- Unter dem Begriff Softwarearchitektur versteht man eine strukturierte oder hierarchische Anordnung der Systemkomponenten sowie die Beschreibung ihrer Beziehungen.

  Balzert H, Lehrbuch der Softwaretechnik,
  Spektrum Akademischer Verlag, 2. Auflage, 2001

SOFTWARE ARCHITEKTUR

- Eine Software Architektur ist der Bauplan für ein Software System
- Sie ist massgeblich für die Zusicherung der geforderten Systemeigenschaften
  - Wenn ein Software System nicht tragfähig ist, stürzt es ein: ist zu langsam, zu fehlerhaft, kann die Anfragen der Nutzer nicht befriedigend bedienen
  - Viele Systeme werden während der Entwicklung eingestellt bzw. können nicht eingesetzt werden, da die Architektur nicht trägt
  - Systeme ohne geeignete Architektur sind nicht wartbar!

SYSTEMANFORDERUNGEN

- Funktionale Anforderungen
  - Use Cases, Szenarios etc.
- Nichtfunktionale Anforderungen
  - Zuverlässigkeit
  - Security
  - Einhaltung von Standards
  - Verfügbarkeit (Beispiele: 7*24, 5*10, 99,99% pro Jahr)
  - Ausfallsicherheit (Beispiel: Fail Over, Stand By)
  - Performance (Beispiel: Antwortzeit <= 0,5 Sec)
  - Skalierbarkeit (Beispiel: Erweiterung von 10 auf 100 parallele Nutzer)

NICHT FUNKTIONALE ANFORDERUNGEN ISO/IEC 9126-1

- Functionality
  - Suitability
  - Accuracy
  - Interoperability
  - Security
- Reliability
  - Maturity
  - Fault tolerance
  - Recoverability
- Usability
  - Understandability
  - Learnability
  - Operability
  - Attractiveness
- Efficiency
  - Time behaviour
  - resource utilization
- Maintainability
  - Analyzability
  - Changeability
  - Stability
  - Testability
- Portability
  - Adaptability
  - Installability
  - Replacability

AUFGABEN DES SOFTWARE ARCHITEKTEN

KOMPONENTEN

- Komponenten-Gedanke von David Parnas von 1972
  - Eine Komponente ist eine abgeschlossene Einheit einer Software, bestehend aus einer Folge von Verarbeitungsschritten und Datenstrukturen
  - Komponenten bieten eine Kapselung (Encapsulation) durch die Trennung von Schnittstelle und Implementierung
  - Eine Komponente kann selbst weitere Komponenten aufrufen - so ist eine Hierarchie von Programmaufrufen möglich
  - Komponenten sind aus mehreren Gründen von Bedeutung:
    - Programmlogik wird wiederverwendbar, ohne dass Code redundant erstellt und gepflegt werden muss
    - Grosse, komplexe Programme können durch den Einsatz von Komponenten gegliedert und strukturiert werden
    - Funktionalitäten können nach dem Baukastenprinzip eingebunden
    - Mehrere Entwicklergruppen können unabhängig voneinander einzelne Komponenten bearbeiten und testen

TRENNUNG VON INTERFACE UND IMPLEMENTIERUNG

- Interface
  - Jede Komponente besitzt ein Interface, welches eindeutig die Leistung der Komponente spezifiziert, unabhängig davon wie diese Leistung realisiert wird
  - Ein Interface stellt die Summe der Funktionen oder Services dar, welche eine Komponente seinen Benutzern zur Verfügung stellt
- Implementierung
  - Der Code, welcher tatsächlich ausgeführt wird
  - Diese Implementierung ist austauschbar, wobei, um die Konsistenz von Komponenten zu erhalten, das Interface konstant bleibt

SEPARATION OF CONCERNS

- Separation of concerns aus "On the role of scientific thought” von Edsger W. Dijkstra, 1974
- Trennung der Verantwortlichkeiten
- Häufig verwendet um funktionale von nichtfunktionalen Anforderungen zu trennen
- Der Businesscode soll nicht “verschmutzt” werden
- Kann durch Aspekt Orientierte Programmierung (AOP) erreicht werden

WARUM ÜBERHAUPT VERTEILUNG?

- Generell gilt: Zentrale, nicht verteilte Anwendungen sind im Allgemeinen
  - sicherer
  - performanter
  - einfacher zu implementieren
- Wesentlicher Grund für Verteilung: gemeinsame Nutzung von Ressourcen
  - Hardware → Kostenersparnis
    - Drucker, Plotter, Scanner
  - Daten und Informationen → Informationsaustausch
    - Fileserver, Datenbank, World Wide Web, Suchmaschinen
  - Funktionalität → Fehlervermeidung und Wiederverwendung
    - Zentralisierung der Funktionalität

DIE DREI DIMENSIONEN VERTEILTER SYSTEME

- Verteilung und Kommunikation
  - Komfort der Programmierung
  - Middleware
  - Remote Procedure Call (RPC)
- Nebenläufigkeit
  - Gleichzeitig voneinander unabhängig
  - Synchronisation
  - Parallelität
  - Synchron
  - Asynchron (Callback, Polling)
- Persistenz
  - Dauerhafte Speicherung

ARCHITEKTURTYPEN

- Schichten
- Event-Driven
- Microkernel
- Microservice
- Space Based

Quelle: Software Architecture Pattern, Mark Richards,
http://www.oreilly.com/programming/free/software-architecture-patterns.csp

SCHICHTEN ARCHITEKTUREN

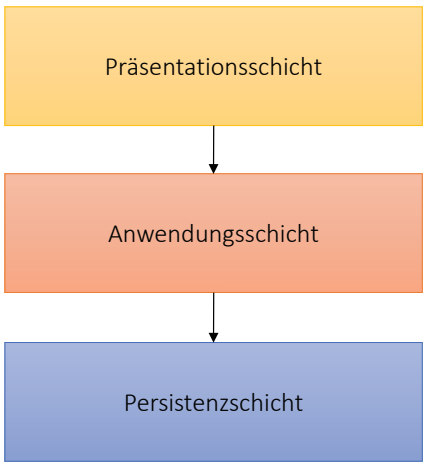
- Wir unterscheiden
  - Physikalische Schichten
  - Logische Schichten

PHYSIKALISCHE SCHICHTEN

- Eine physikalische Schicht kennzeichnet einen unabhängigen Prozessraum innerhalb einer verteilten Anwendung.
- Die physikalische Schicht dient als Grundlage der n-Tier Architekturen.
- n-Tier Architekturen sind eine Erweiterung / Verfeinerung des Client-Server Architekturmodells.
- Eine n-Tier Architektur legt fest, wie viele unterschiedliche Client- und Server-Prozessräume es innerhalb einer verteilten Anwendung gibt.
- Das n gibt an wie viele Prozessräume beteiligt sind.
- Ein Prozessraum kann, muss jedoch nicht (!), einem physikalischen Rechner entsprechen.
- Die Verwendung von n-Tier Architekturen macht vor allem für Informationssysteme Sinn:
  - Verteilung der Software zur Reduktion der Komplexität
  - Interaktive Mensch-Maschine Schnittstelle
  - Datenzentriertes Vorgehen

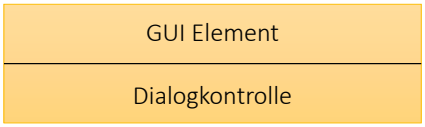
LOGISCHE SCHICHTEN

- Vorteile von verteilten Systemen
  - Zentrale Datenhaltung
  - Skalierbarkeit
  - Fehlertoleranz
- Klassische logische Aufteilung in 3 Schichten
  - Präsentationsschicht
  - Anwendungsschicht
  - Persistenzschicht



PRÄSENTATIONSSCHICHT

- Primäre Aufgabe
  - Benutzeroberfläche um Daten darzustellen und auf Benutzereingaben zu reagieren
- Design Pattern Model View Controller (MVC)
  - GUI Element (View)
  - Dialogkontrolle (Controller)
  - Daten (Model)
- Zur Erreichung einer losen Kopplung weiss die Präsentationsschicht möglichst wenig von der Geschäftslogik. Darüber hinaus soll sie – im Sinne einer Schichtenarchitektur – keinerlei Kenntnisse über die Persistenzschicht besitzen



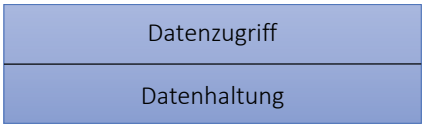
ANWENDUNGSSCHICHT

- Primäre Aufgabe
  - Fachliche Objekt und Logik um Geschäftsprozesse abzubilden
- Weiter Aufteilung in
  - Services
    - Stellt Methoden und Dienste zur Verfügung um die Geschäftsprozesse zu realisieren
    - Erfüllt die Anforderungen der Präsentationsschicht
    - Meist Zustandslos
  - Geschäftsobjekte
    - Business Object Model
    - In der Regel persistent

PERSISTENZSCHICHT

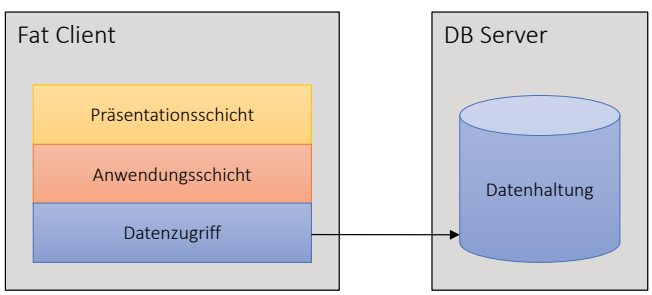
- Primäre Aufgabe
  - Dienste die für die dauerhafte Verwaltung der fachlichen Daten in einem Datenbanksystem sorgen

- Weiter Aufteilung in
  - Datenhaltung
    - Datenbanksystem (meist relational)
  - Datenzugriff
    - O/R Mapping (z.B. JPA)
    - JDBC
    - SQL



2-TIER

- Das Modell unterstützt 2 Tiers: Client- und Server-Tier
- Zuordnung von Aufgaben zu Tiers:
  - Präsentation – Client-Tier
  - Anwendungslogik – Client-Tier und/oder Server-Tier
  - Datenhaltung – Server-Tier
- Die Verteilung der Anwendungslogik auf Client- und Server-Tier kann variieren.

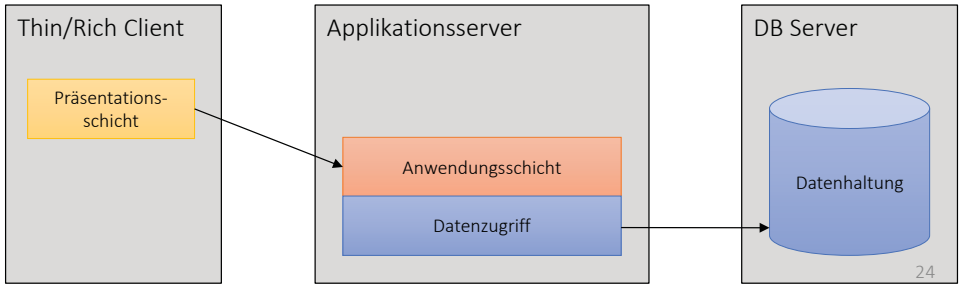


2-TIER

- Erstes und ältestes Verteilungsmodell
- Wird häufig im Zusammenhang mit Stored Procedures und 4GL eingesetzt:
  - 4GL (Fourth Generation Languages) unterstützen direkt Sprachelemente zum Zugriff auf eine Datenbank
  - Stored Procedures sind vordefinierte SQL Statements, die in der Datenbank abgelegt und zur Laufzeit aufgerufen werden.
- Vorteile:
  - Einfach und schnell umzusetzen
  - Performant
- Probleme:
  - schwer wartbar
  - schwer skalierbar
  - Software-Update Problem

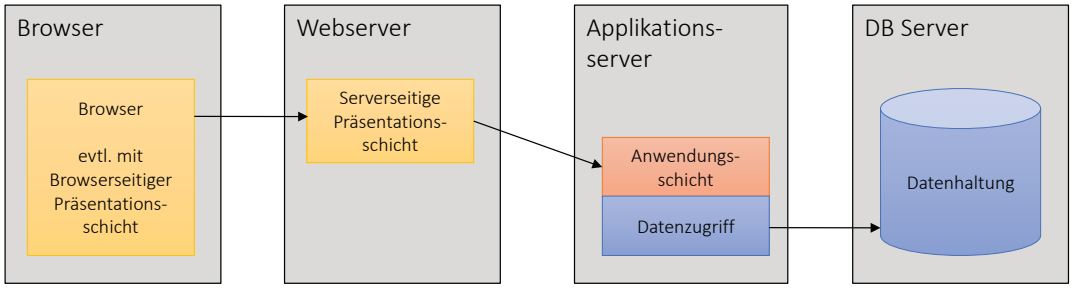
3-TIER

- Leichtgewichtiger Client (Thin- oder Rich-Client)
- Komplette Businesslogik auf dem Applikationsserver
- Vorteile
  - Besser skalier- und überwachbar
  - Einfachere Softwareverteilung
  - Auch auf leistungsschwachen Geräten (PDA, Handys etc.) ausführbar

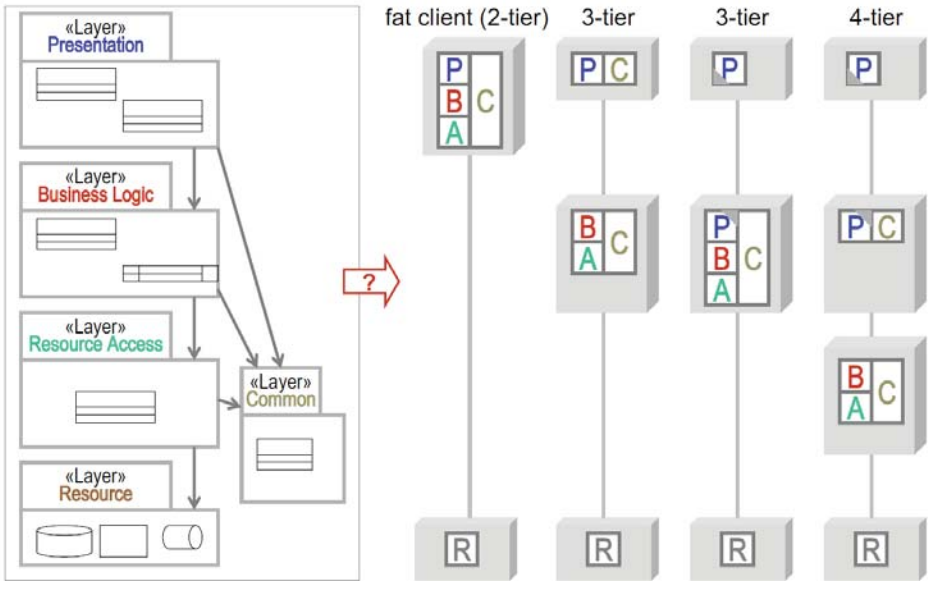


N-TIER

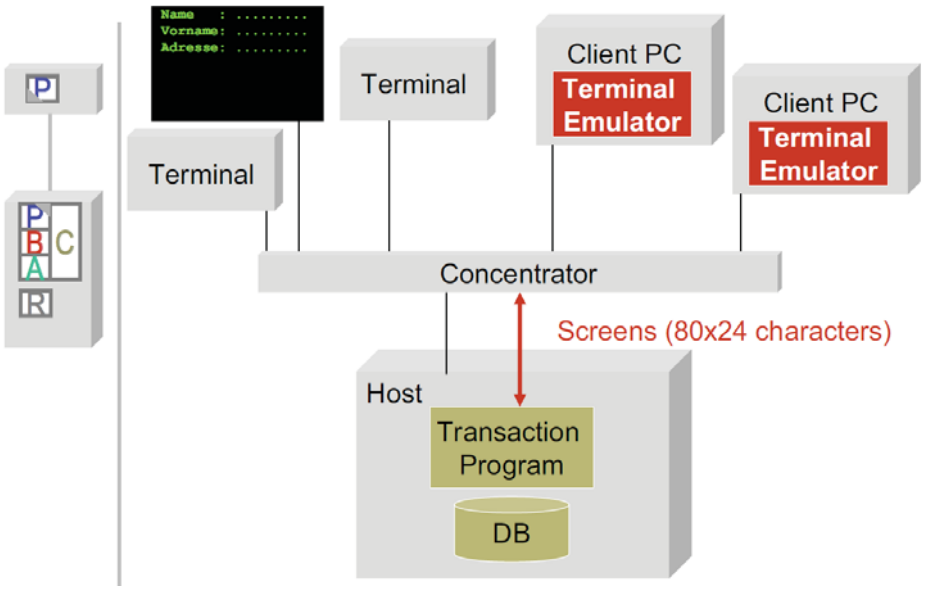
- Clients sehr leichtgewichtig
- Webserver (meistens im Applikationsserver integriert) erzeugt dynamische Inhalte (JSP, JSF, ASP.NET u.ä.)
- Client Hardware auf ein Minimum reduzierbar
- Kein Softwareverteilung nötig



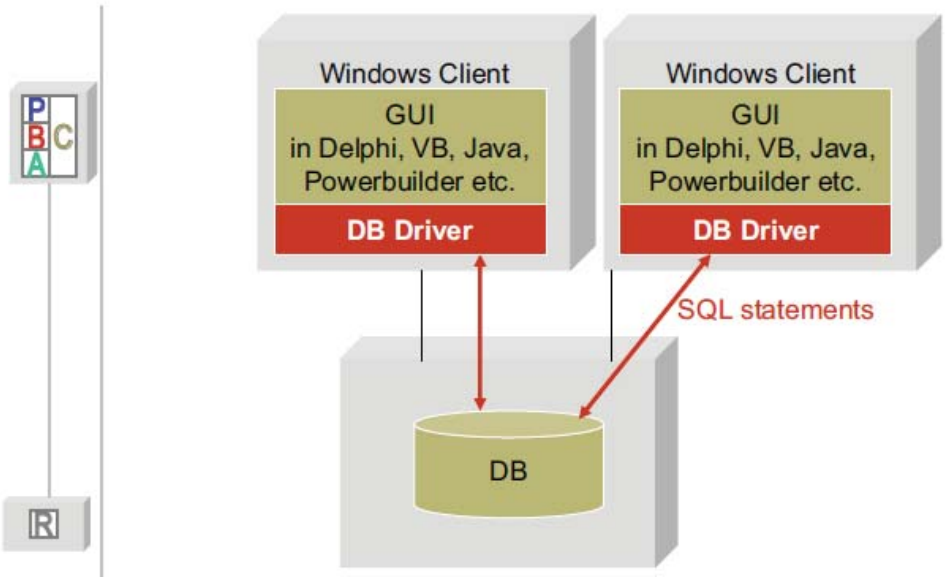
SCHICHTEN UND VERTEILUNG



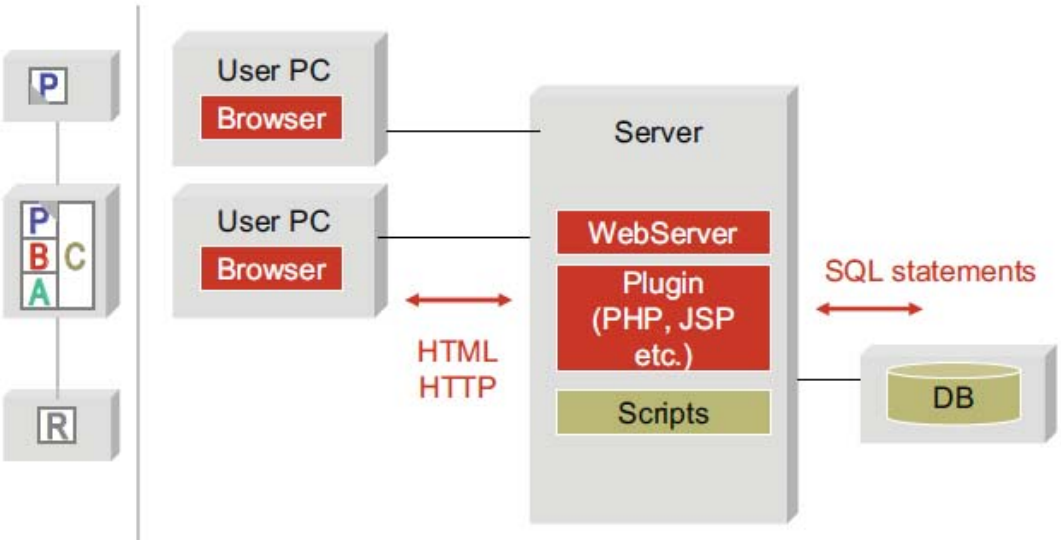
HOST



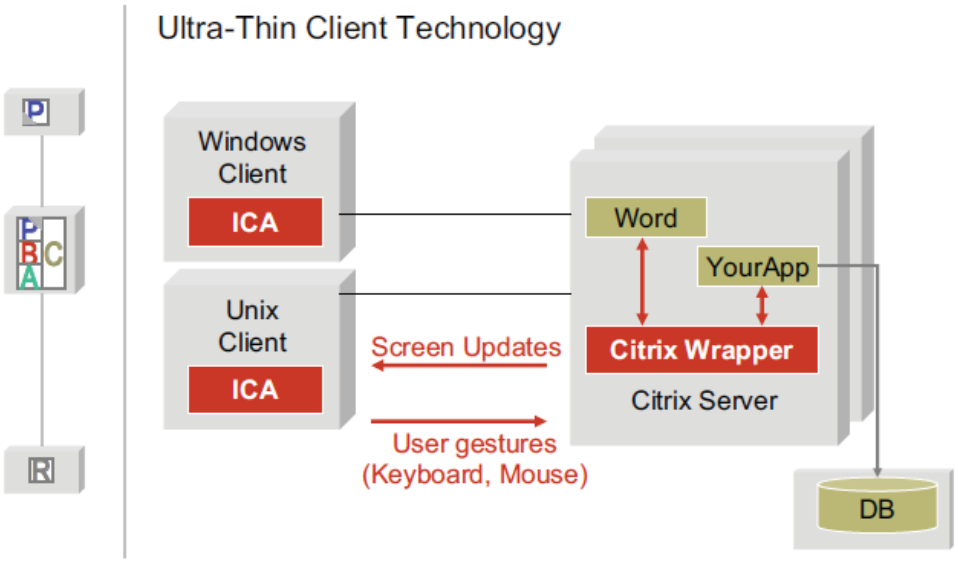
FAT CLIENT



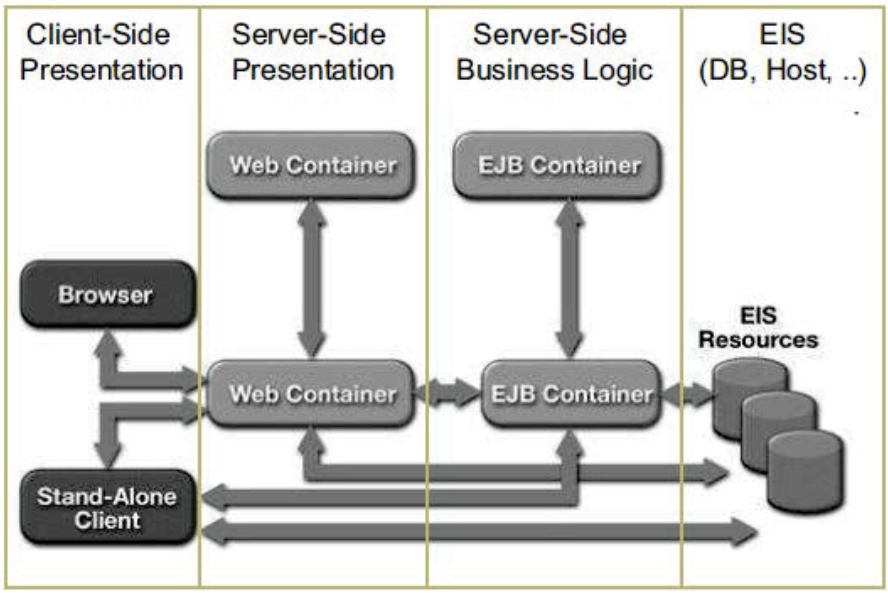
WEB



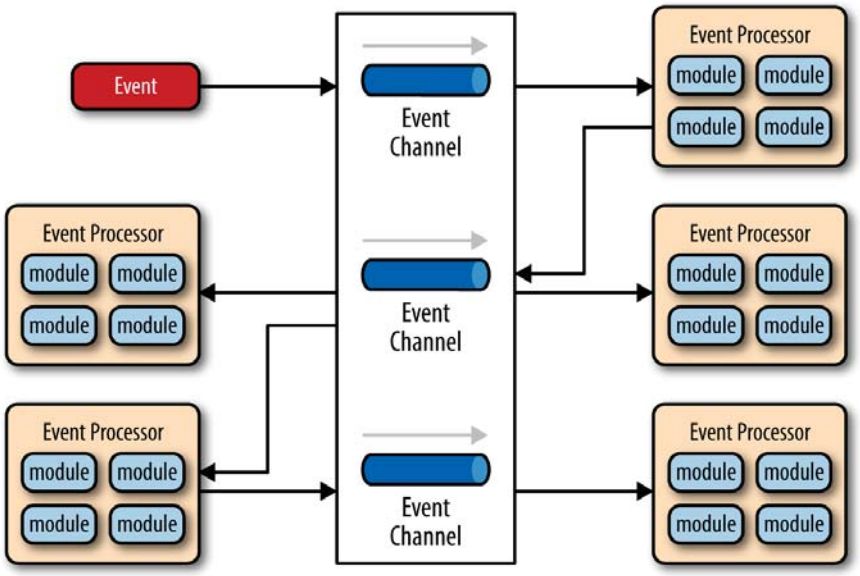
CITRIX METAFRAME



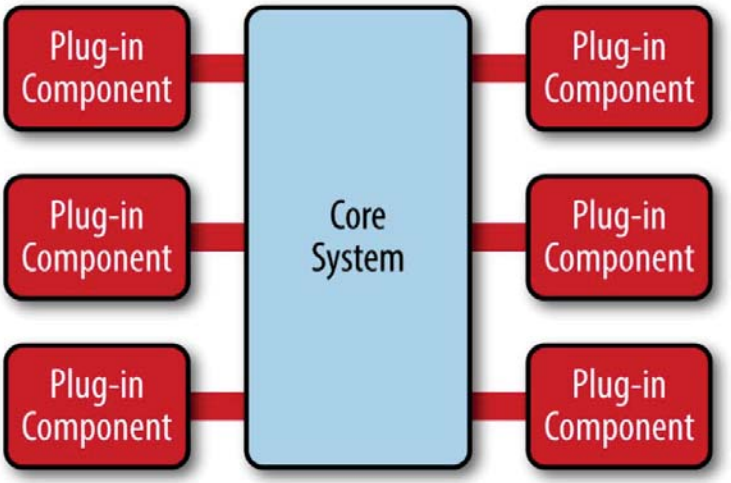
JAVA EE



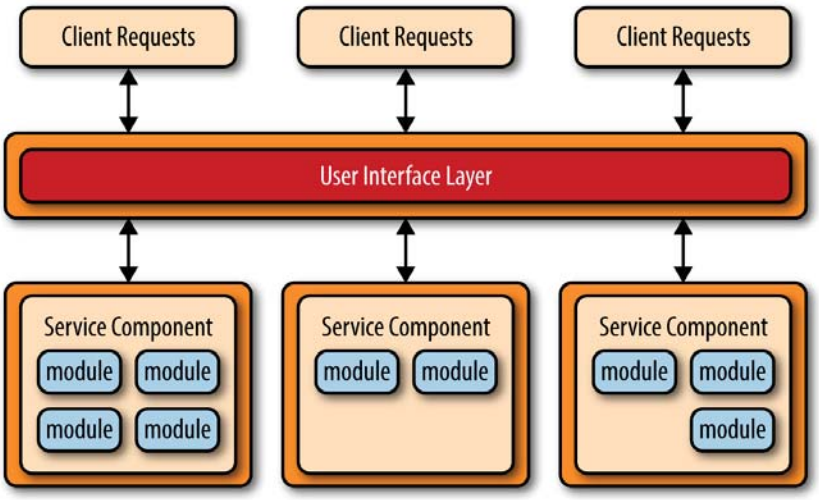
EVENT-DRIVEN ARCHITEKTUR



MIKROKERNEL ARCHITEKTUR

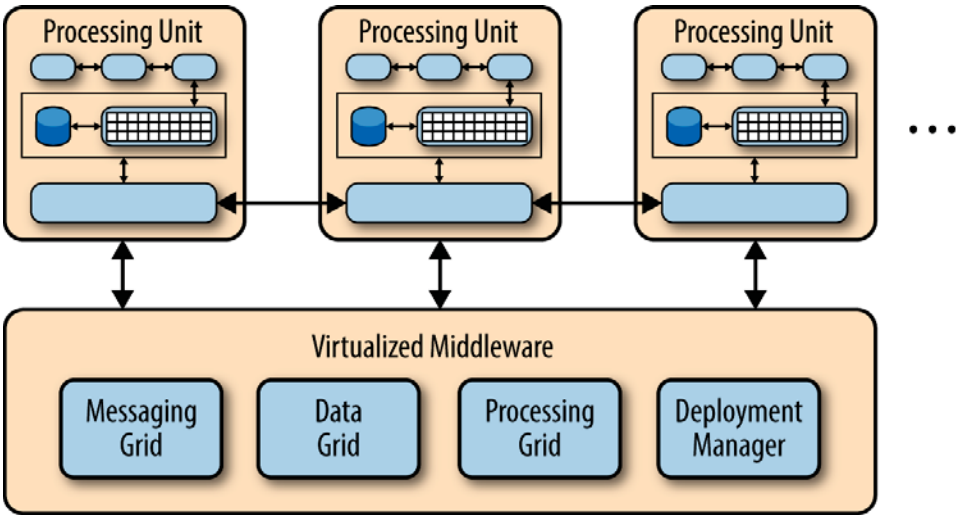


MICROSERVICE ARCHITEKTUR



http://martinfowler.com/articles/microservices.html

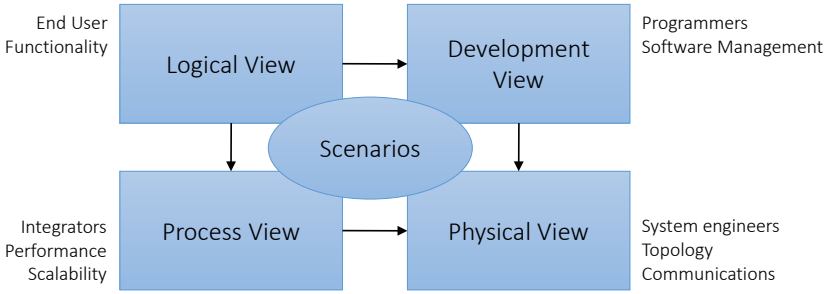
SPACE-BASED ARCHITEKTUR



1.1 ARCHITEKTUR-DOKUMENTATION

4+1 View Model

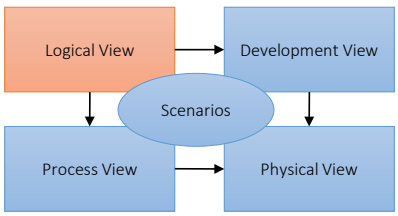
- Architectural Blueprints: The “4+1” View Model of Software Architecture von Philippe Kruchten (Rational Software Corp.).
- Basiert auf mehreren konkurrierenden Sichten (Views).
- Die Sichten werden gebraucht um allen Stakeholdern (Benutzern, Entwicklern, Projektleitern etc.) gerecht zu werden



LOGICAL VIEW

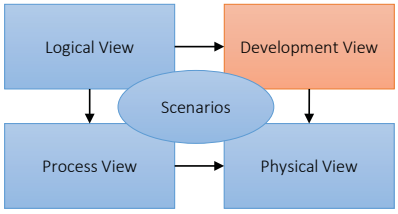
- Die logische Sicht behandelt die Funktionalität die das System dem Benutzer zur Verfügung stellt

- Verwendete UML Diagramme
  - Komponentendiagramm
  - Klassendiagramm
  - Kommunikationsdiagramm
  - Sequenzdiagramm



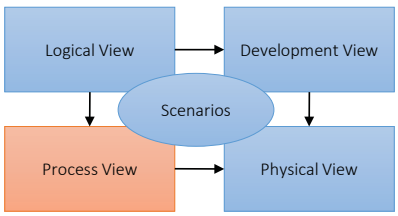
DEVELOPMENT VIEW

- Die Entwicklungssicht illustriert ein System aus Entwicklersicht
- Diese Sicht ist auch als Implementationssicht bekannt
- Verwendete UML Diagramme
  - Klassendiagramm
  - Paketdiagramm



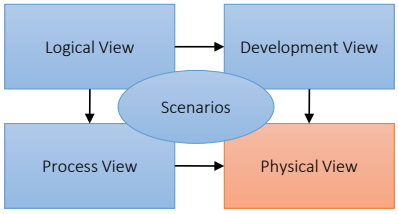
PROCESS VIEW

- Das Prozessmodell behandelt die dynamischen Aspekte eines Systems
- Es erklärt die Systemprozess und wie diese miteinander kommunizieren
- Der Fokus liegt auf den Laufzeitverhalten des Systems
- Die Prozesssicht adressiert Nebenläufigkeit, Verteilung, Integration, Performance, Skalierbarkeit usw.
- Verwendete UML Diagramme
  - Sequenzdiagramm
  - Aktivitätsdiagramm



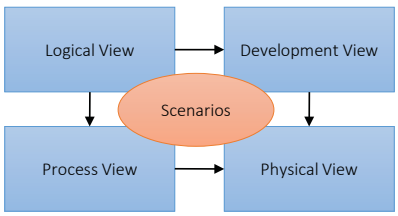
PHYSICAL VIEW

- Die physikalische Sicht beleuchtet das System aus der Sicht des System Engineers oder des Deployers
- Es beschreibt die Topologie der physikalischen Sicht wie auch die Kommunikation zwischen den Komponenten
- Diese physikalischen Sicht ist auch als Deployment- oder Verteilungsschicht bekannt
- Verwendete UML Diagramme
  - Deploymentdiagramm



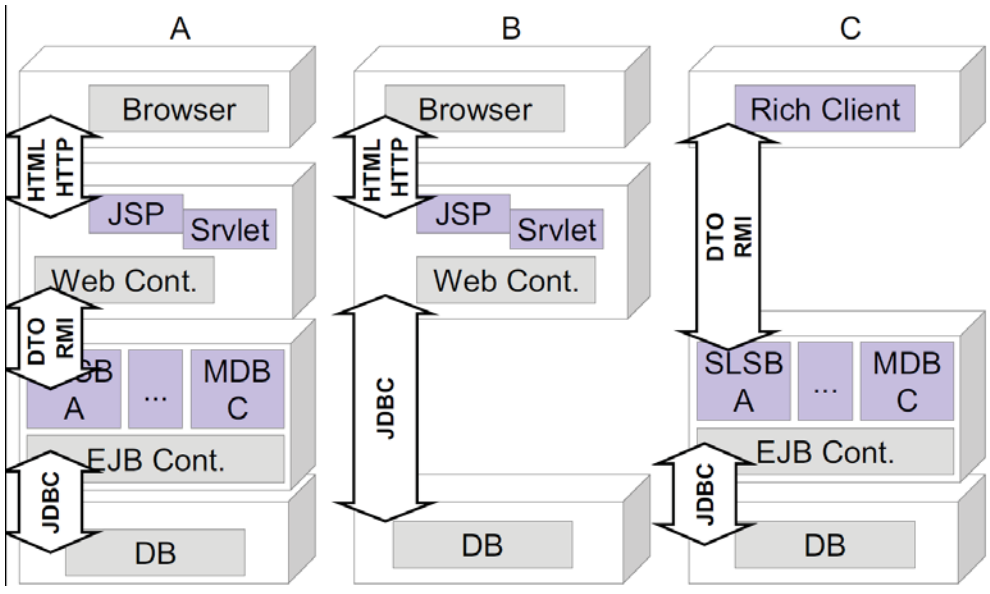
SCENARIOS

- Die Scenarios oder Use Cases dienen dazu Architekturelemente zu identifizieren, zu illustrieren und zu validieren
- Sie dienen ebenfalls als Ausgangspunkt für einen Prototyp
- Die Use Cases ergeben die fünfte Sicht und überlagern die anderen Sichten
- Verwendete UML Diagramme
  - Use Case Diagramm

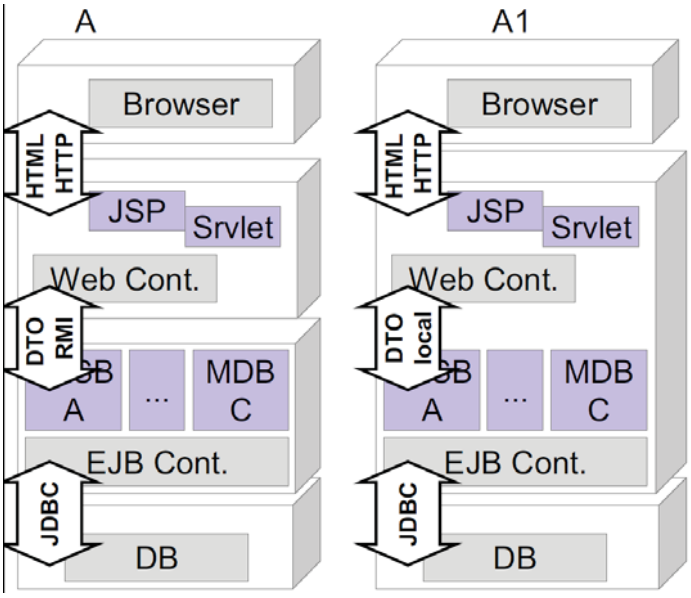


2 DESIGN

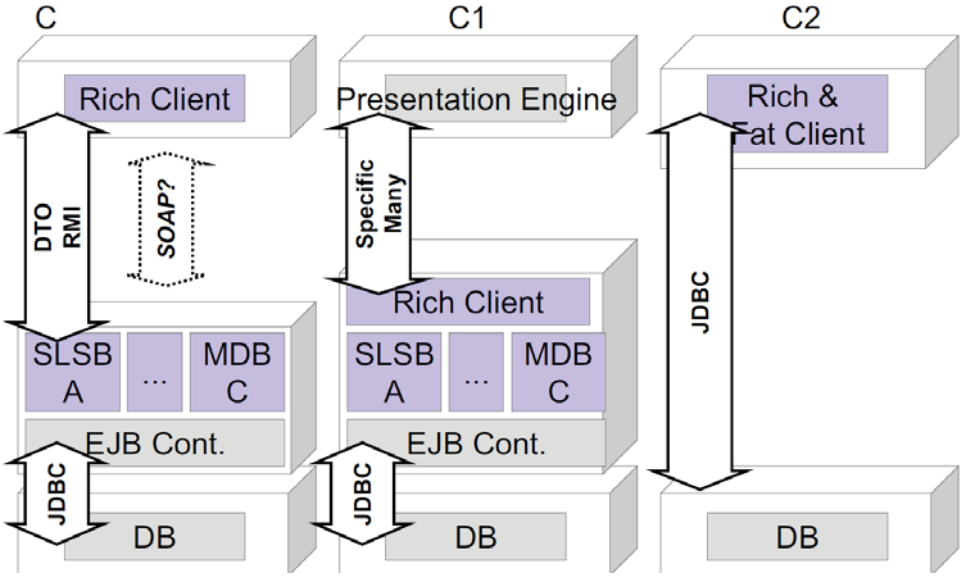
ALTERNATIVEN



VARIANTEN ZU A



VARIANTEN ZU C



ÜBUNG VARIANTENVERGLEICH

Give each variant a mark (1 worst, 6 best) and explain each mark with in few words

	A	A1	B	C	C1	C2
Development effort						
Performance - response time						
Performance - scalability						
Security						
Ease of Deployment						
Richness GUI						
Operating effort						
....						

3 PATTERNS

WAS SIND PATTERN?

- Pattern beschreiben allgemeine Lösungen zu häufig wiederkehrenden Problemen
- Ermögliche Wiederverwendung auf konzeptueller Ebene
- Stammen aus dem Bereich der Gebäudearchitektur (Christopher Alexander, 1970er). Beispiel: six foot balcony
- Werden heute auf verschiedene Bereiche (Prozesse, Softwareentwicklung) angewendet
- Wurden in Form von Design- und Architektur-Pattern für die objektorientierte Software Entwicklung adaptiert

GESCHICHTE DER PATTERN

- 1987: Ward Cunningham und Kent Beck schreiben die ersten fünf Software-Pattern und stellen sie auf der OOPSLA 87 vor
- 1990: Erich Gamma stösst auf Pattern im Rahmen seiner Dissertation
- 1991: Erich Gamma und Richard Helm treffen sich auf Bruce Andersons Workshop „Toward an Architecture Handbook“ auf der OOPSLA 91
- 1992: Ralph Johnson und John Vlissides stossen hinzu und bilden mit Erich Gamma und Richard Helm die „Viererbande“ (GoF, Gang of Four)
- 1994: Die erste Ausgabe von „Design Patterns“ erscheint und wird schnell zum Bestseller
- 1996: „Pattern-Oriented Software Architecture“ von Buschmann et. al. erscheint.
- 1999: Patterns werden zum „Hype“
  Neben Design- und Architektur-Pattern finden sich unter anderem Analyse-Pattern, Prozess-Pattern und viele andere

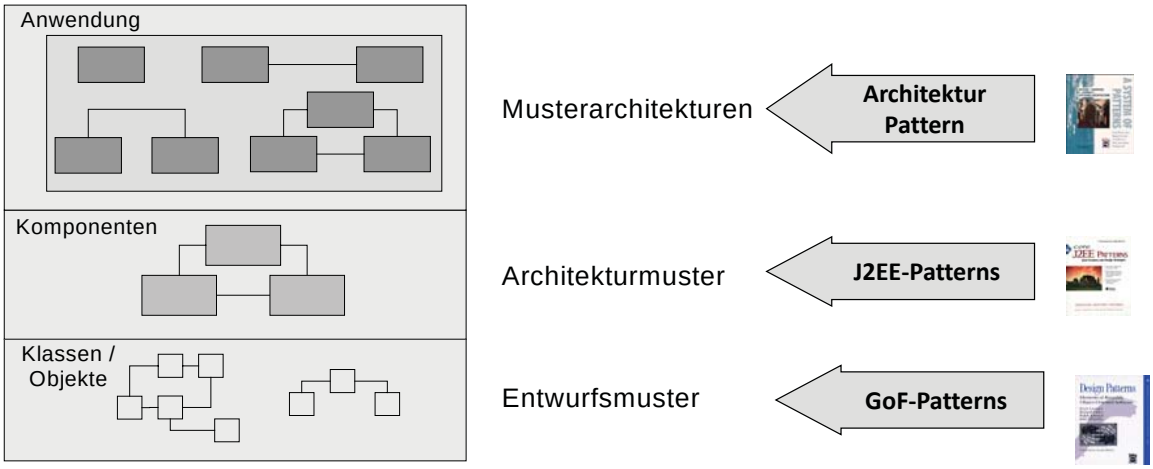
CHARAKTERISTIKEN VON PATTERN

- Werden durch Beobachtung identifiziert, nicht neu entwickelt
- Werden in einem vorgegebenen strukturierten Format aufgeschrieben
- Existieren auf verschiedenen Abstraktionsebenen
- Werden in der Regel in Kombination eingesetzt werden
- Sie stammen immer aus realen Projekten, werden abstrahiert und aufbereitet

DESIGN PATTERN VERSUS ARCHITEKTUR PATTERN

- Design Pattern (GoF) beschreiben Zusammenhänge zwischen Klassen
  Beispiele:
  - Singleton Pattern: Objekt liegt als einzige Instanz vor
  - Adapter Pattern: Adapter kapselt Zugriff auf Objekt bzw. Nachbarsystem
  - Factory Pattern: Objekte als Instanzfabriken
- Architektur Pattern (Buschmann et al.) betrachten vollständige(re) Systemarchitekturen
  Beispiele:
  - Layers: Verwendung von Schichten zur Strukturierung von Systemen
  - Model-View-Controller: zur Trennung von Präsentationslogik, Daten und Business Logik
  - Broker

EBENEN ZUM EINSATZ VON PATTERN



GOF PATTERN

- Drei Musterkategorien:
  - Erzeugungsmuster verstecken den Prozess der Instanziierung von Objekten vor der Anwendung. Beispiel: Abstract Factory
  - Strukturmuster beschreiben die Komposition von Klassen oder Objekten zu grösseren zusammenhängenden Gebilden. Beschrieben werden die beteiligten Klassen oder Objekte sowie die strukturellen Abhängigkeiten. Beispiel: Proxy
  - Verhaltensmuster beschreiben die Art und Weise der Zusammenarbeit zwischen Klassen und Objekten. Beispiel: Observer
- Unterscheidung zwischen klassen- und objektbasierten Mustern
  - Klassenbasierte Muster arbeiten mit Vererbung und Schnittstellenimplementierung
  - Objektbasierte Muster konzentrieren sich auf die Komposition von Objekten und ihre Beziehungen

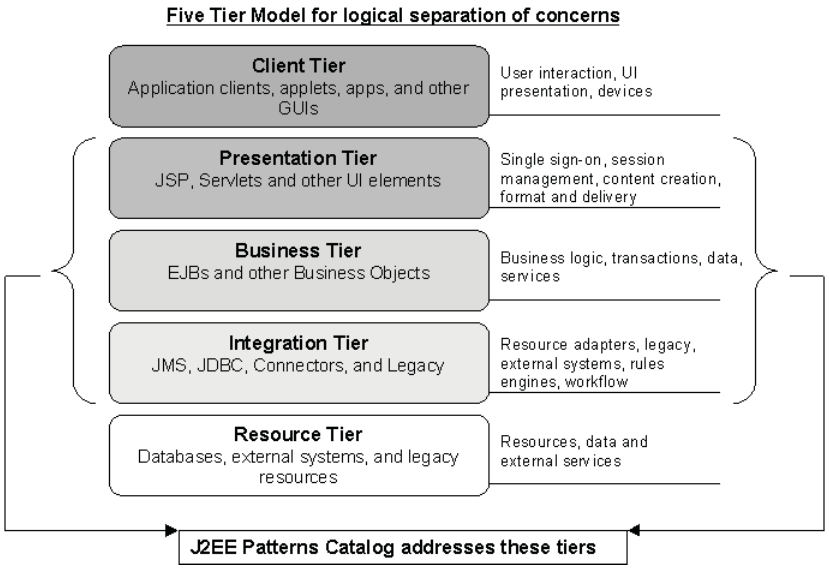
BESCHREIBUNG VON PATTERN

- Problem
  - Kurze Darstellung des Problems, das durch das Pattern gelöst werden kann
  - Ausführliche Beschreibung der Problematik
- Kräfte
  - Beschreibt Voraussetzungen, bzw. Fragestellungen wann das Pattern eingesetzt werden sollte
  - Beschreibt den Kontext in dem der Einsatz des Pattern Sinn macht
- Lösung
  - Liefert eine Zusammenfassung, sowie eine ausführliche Beschreibung der Lösung
  - Beschreibt die Elemente, ihre Rollen, ihre Beziehungen, die Verantwortlichkeiten und die Interaktionen
  - Liefert ggf. eine Beispielimplementierung
- Konsequenzen
  - Beschreibt positive und negative Konsequenzen bei Einsatz des Pattern
    - Was wird erreicht durch den Einsatz des Pattern?
    - Was kann nicht erreicht, gelöst werden?

J2EE PATTERN

- Sind von der Granularität her zwischen Design- und Architektur Pattern anzusiedeln
- Wurden von Sun auf Basis der ersten Erfahrungen mit dem J2EE Standard identifiziert und ausgearbeitet
- Werden nach den Tiers klassifiziert
  - Presentation-Tier Pattern
  - Business-Tier Pattern
  - Integration-Tier Pattern
- Decken insbesondere das Design von Web Applikationen auf der Basis von Servlets /JSP sowie EJB ab
- Zur Zeit gibt es 21 J2EE Pattern
- Viele wurden durch Java EE obsolet

DER TIER-STACK VON SUN



PRESENTATION-TIER

- Design auf der Basis von Servlets, Java Server Pages (JSP) und Klassen.
- Design-relevante Themen
  - Annehmen und Überwachen von Client Zugriffen.
  - Aufbereiten und Speichern der eingegebenen Daten.
  - Lokalisierung von Business Komponenten und Dialogen.
  - Verwaltung komplexer Dialoge.
  - Trennung von reiner Präsentation und Darstellungslogik.

PRESENTATION-TIER PATTERN

- Intercepting Filter:
  - Problem: You want to intercept and manipulate a request and a response before and after the request is processed.
  - Solution: Use an Intercepting Filter as a pluggable filter to pre- and post-process requests and responses.
- Front Controller:
  - Problem: You want to centralize access points for presentation-tier request handling.
  - Solution: Use a Front Controller as the initial point of contact for handling all related requests.

PRESENTATION-TIER PATTERN

- Context Object:
  - Problem: You want to avoid using protocol-specific system information outside of its relevant context.
  - Solution: Use a Context Object to encapsulate state in a protocol-independent way to be shared throughout your application.
- Application Controller:
  - Problem: You want to centralize and modularize action and view management.
  - Solution: Use an Application Controller to centralize retrieval and invocation of request processing components, such as commands and views.

PRESENTATION-TIER PATTERN

- View Helper
  - Problem: You want to separate a view from its processing logic.
  - Solution: Use views to encapsulate formatting code and Helpers to encapsulate view-processing logic.
- Composite View:
  - Problem: You want to build a view from modular, atomic component parts that are combined to create a composite whole, while managing the content and the layout independently.
  - Solution: Use Composite Views that are composed of multiple atomic subviews. Each subview can be included dynamically in the whole, and the layout of the page can be managed independently of the content.

PRESENTATION-TIER PATTERN

- Service to Worker:
  - Problem: You want to perform core request handling and invoke business logic before control is passed to the View.
  - Solution: Use Service to Worker to centralize control and request handling to retrieve a presentation model before turning control over to the view. The view generates a dynamic response based on the presentation model.
- Dispatcher View:
  - Problem: You want a view to handle a request and generate a response, while managing limited amounts of business processing.
  - Solution: Use Dispatcher View with views as the initial access point for a request. Business processing, if necessary in limited form, is managed by the views.

PRESENTATION-TIER: DESIGN-RELEVANTE THEMEN

- Annehmen und Überwachen der Clientzugriffe.
  - Intercepting Filter, Front Controller
- Aufbereiten und Speicherung der eingegebenen Daten.
  - Context Object
- Lokalisierung von Business Komponenten und Dialogen.
  - Application Controller, Dispatcher View
- Verwaltung komplexer Dialoge.
  - Composite View
- Trennung von reiner Präsentation und Darstellungslogik.
  - View Helper

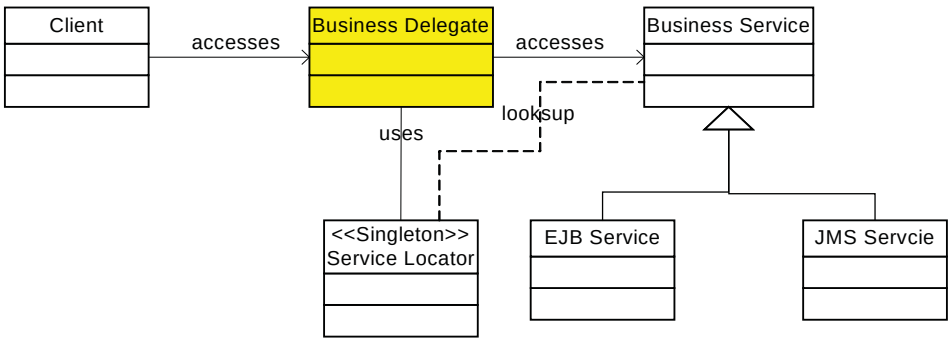
BUSINESS-TIER

- Design auf der Basis von EJB und Klassen
- Design-relevante Themen:
  - Transparenter Zugriff vom Client auf den Server
  - Auffinden von Services am Server
  - Serviceorientierte Schnittstelle, Verbergen der Datenschicht,
  - Verwaltung der Datenschicht
  - Transport von Daten, Datenverwaltung, Minimierung von Zugriffen

BUSINESS-TIER PATTERN

- Business Delegate:
  - Problem: You want to hide clients from the complexity of remote communication with business service components
  - Solution: Use a Business Delegate to encapsulate access to a business service. The Business Delegate hides the implementation details of the business service, such as lookup and access mechanisms
- Ziele:
  - Agiert als client-seitiger Proxy des Services
  - Verbirgt technische Details der unterliegenden Kommunikation vor dem Client
    - Lokalisierung des Naming Service
    - Lokalisierung des Services
    - Fehlerbehandlung (Retry und Recovery)
    - Caching von Referenzen auf Services.

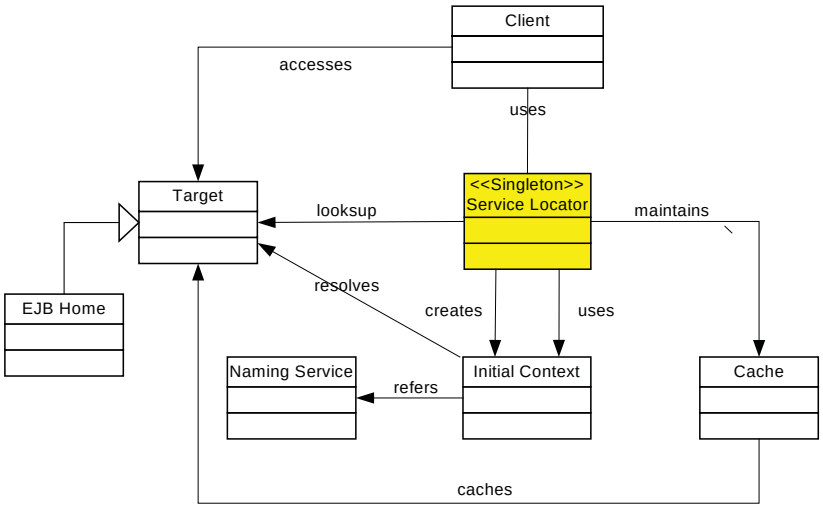
BUSINESS DELEGATE - KLASSENDIAGRAMM



BUSINESS-TIER PATTERN

- Service Locator:
  - Problem: You want to transparently locate business components and services in a uniform manner
  - Solution: Use a Service Locator to implement and encapsulate service and component lookup. A Service Locator hides the implementation details of the lookup mechanism and encapsulates related dependencies
- Wird als Singleton realisiert.
- Ziele
  - Kapselt insbesondere den Zugriff auf den Naming Service über JNDI
  - Unterstützt das Auffinden von verschiedenen Services
    - EJB Service Locator
    - JMS Service Locator
    - JDBC Service Locator
  - Kann immer eingesetzt werden, wenn Zugriff auf den Naming Service erforderlich ist

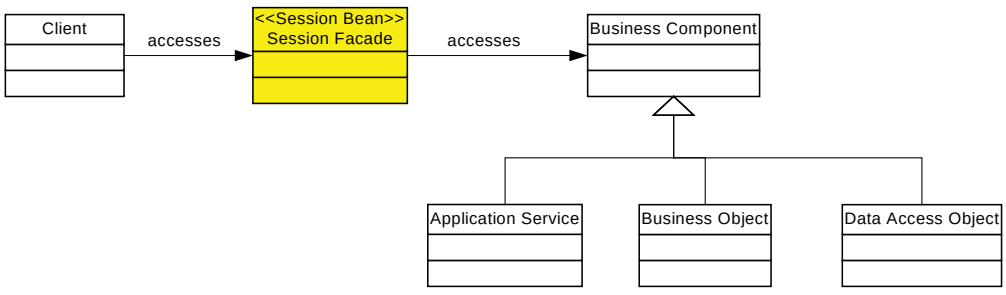
SERVICE LOCATOR - KLASSENDIAGRAMM



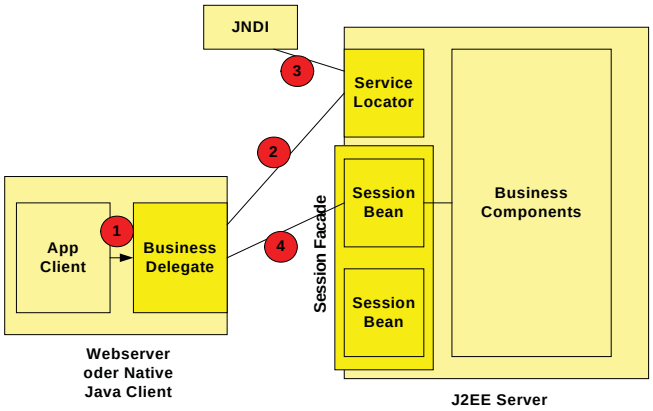
BUSINESS-TIER PATTERN

- Session Facade:
  - Problem: You want to expose business components and services to remote clients
  - Solution: Use a Session Facade to encapsulate business-tier components and expose a coarse-grained service to remote clients. Clients access a Session Facade instead of accessing business components directly
- Eine Session Facade wird als (Stateful oder Stateless) Session Bean realisiert.
- Ziele:
  - Bietet dem Client eine dienstorientierte Schnittstelle (orientiert sich an den Anwendungsfällen)
  - Kapselt den Zugriff auf die Datenschicht

SESSION FACADE - KLASSENDIAGRAMM



DIE ZUGRIFFSPATTERN IM ZUSAMMENHANG

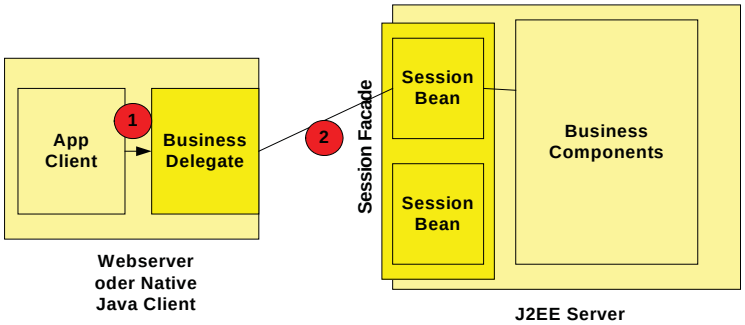


Fall 1:

Der Client kennt seinen Service nicht

1. Der Client instantiiert die Business Delegate Klasse und sendet seinen Request an den Delegate.
2. Der Business Delegate fordert beim Service Locator eine Referenz auf den geforderten Business Service an.
3. Der Service Locator sucht die Referenz aus dem JNDI Baum und übergibt sie dem Business Delegate.
4. Über der Referenz (Home Interface) kann der Service lokalisiert und der Request an die entsprechende Session Facade weitergeleitet werden.

DIE ZUGRIFFSPATTERN IM ZUSAMMENHANG



Fall 2:

Der Client kennt seinen Service.

1. Der Client instantiiert den Business Delegate mit einem Handle auf den gewünschten Service.
2. Der Business Delegate lokalisiert den Service und leitet alle weiteren Requests an die Session Facade mit dem Service weiter.

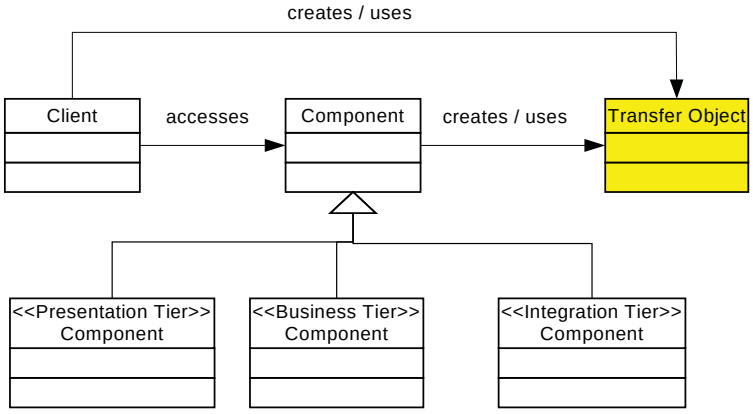
BUSINESS-TIER PATTERN

- Business Object:
  - Problem: You have a conceptual domain model with business logic and relationships
  - Solution: Use Business Objects to separate business data and logic using an object model
- Business Objects kapseln persistente Daten in der Datenschicht
- Sie werden unterschieden nach Vaterobjekte bzw. abhängigen Objekten
- Geschäftsprozesse werden davon getrennt realisiert, beispielsweise in Session Beans
- Ziele:
  - Wiederverwendbarkeit der Daten in der Datenschicht für verschiedene Geschäftsprozesse

BUSINESS-TIER PATTERN

- Transfer Object:
  - Problem: You want to transfer multiple data elements over a tier
  - Solution: Use a Transfer Object to carry multiple data across a tier
- Daten werden in Datencontainern zusammengefasst und über Tier-Grenzen transportiert
- Das Konzept sollte generell an Tier-Grenzen eingesetzt werden
- Ziele:
  - Verringerung der Remote Zugriffe
  - Verringerung der übertragenen Datenmenge

TRANSFER OBJECT - KLASSENDIAGRAMM



TO UND JPA

- Mit Constructor Expressions existiert eine einfache Möglichkeit direkt aus den Resultaten Transfer Objects zu erzeugen.

```
public class EmpListItem {
  public EmpListItem(String employeeName, String deptName){...}
}

List result = em.createQuery(
  "SELECT NEW jpa.util.EmpListItem(e.name, e.department.name) "
+ "FROM Project p JOIN p.employees e "
+ "where p.name = "ZLD").getResultList();

for (EmpListItem item : result) {
  log.info(item.employeeName + "," + item.deptName);
}
```

BUSINESS-TIER: DESIGN-RELEVANTE THEMEN

- Transparenter Zugriff vom Client auf den Server
  - Business Delegate
- Auffinden von Services am Server
  - Service Locator
- Serviceorientierte Schnittstelle, Verbergen der Datenschicht,
  - Session Facade
- Verwaltung der Datenschicht
  - Business Object
- Transport von Daten, Datenverwaltung, Minimierung von Zugriffen
  - Transfer Object

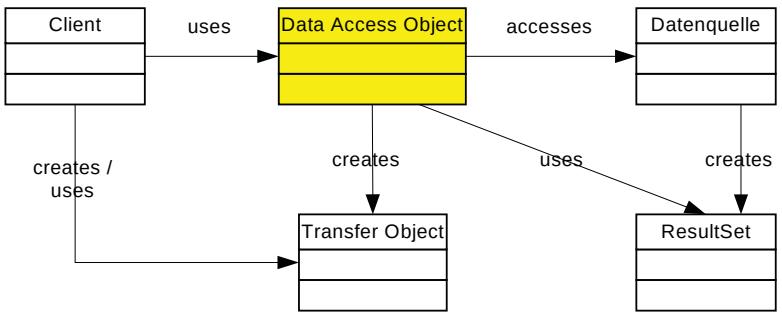
INTEGRATION-TIER

- Design-relevante Themen
  - Datenbankzugriffe
  - Asynchrone Kommunikation
  - Zusammenfassung von Business Services zu Web Services

INTEGRATION-TIER PATTERN

- Data Access Object (DAO):
  - Problem: You want to encapsulate data access and manipulation in a separate layer
  - Solution: Use a Data Access Object to encapsulate all access to the persistent store. The Data Access Object manages the connection with the data source to obtain and store data
- DAOs stellen eine leichtgewichtige Alternative zur Verwendung von Entity Beans dar
- DAOs werden als einfache, zustandslose Klassen realisiert, die den Zugriff auf eine Datenquelle kapseln
- Ziele:
  - Trennung von Business Logik und technischer Zugriffslogik.
  - Leichtgewichtige Persistenz (insbesondere Lesezugriffe).
  - Kapselung von Zugriff auf Fremdsysteme

Data Access Object - Klassendiagramm



DAO UND JPA

- Häufig überflüssig dank EntityManager
- Wenn schon DAO dann generisch

```
public interface GenericDAO {
  <T extends BaseEntity> T create(T t);
  <T extends BaseEntity> T find(Class<T> type,
                                Serializable id);
  <T extends BaseEntity> T update(T t);
  void delete(Object t);
  List findByNamedQuery(String queryName);
  List findByNamedQuery(String queryName,
                        Map<String, Object> parameters);
}
```


INTEGRATION-TIER PATTERN

- Service Activator:
  - Problem: You want to invoce services asynchronously
  - Solution: Use a Service Activator to receive asychronously requests and invoke one or more business services

- Ein Service Activator wird umgesetzt als Message Driven Bean
- Ziele:
  - Empfängt asynchrone Requests und vermittelt sie zur Bearbeitung weiter an einen Service (Session Bean)

INTEGRATION-TIER PATTERN

- Web Service Broker:
  - Problem: You want to provide access to one ore more services using XML and web protocols.
  - Solution: Use a Web Service Broker to expose and broker one or more services using XML and web protocols.

- Ein Web Service Broker ist als Web Service realisiert.
- Ziele:
  - Er fasst mehrere feingranulare Business Services zusammen zu einem gröberen Services, der an der Webschnittstelle zur Verfügung gestellt wird.

INTEGRATION-TIER: DESIGN-RELEVANTE THEMEN

- Datenbankzugriffe
  - Data Access Object
- Asynchrone Kommunikation
  - Service Activator
- Zusammenfassung von Business Services zu Web Services
  - Web Service Broker

4 PERFORMANCE

DURCHSATZ, ANTWORTZEIT, SKALIERBARKEIT, LAST

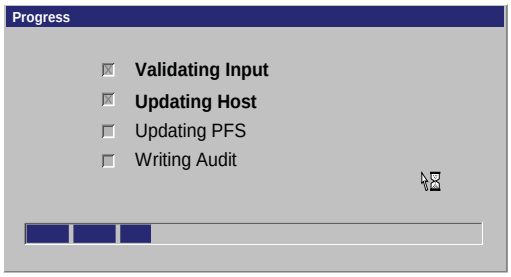
- Antwortzeit (a.k.a. latency)
  - Benutzersicht: Wie lange muss ich auf das Resultat warten?
- Durchsatz
  - Betreiber-/Ownersicht: Wieviel Anfragen kann das System in einer definierten Zeitperiode verarbeiten?
- Skalierbarkeit
  - Betreiber-/Ownersicht: Wie verhält sich das System bei höherer Last? Kann ich die Hardware verbessern um grössere Last zu verarbeiten?
- Last
  - Betreiber-/Ownersicht: Was ist die Frequenz der Benützung der Applikatoin?

BEISPIEL ANTWORTZEIT UND DURCHSATZ

- Antwortzeit
  - Anforderung: After clicking on the “Find”-Button, I get the search-results in 5 seconds
  - Response-Time = 5 Sekunden
- Durchsatz
  - With 1000 concurrent user performing a search every 110 seconds, the response-time is 10 seconds.
  - Think time = 110 s
  - Cycle time = Think Time + Response Time = 110s + 10s = 120s
  - Throughput = 500 requests/min = 8.3 requests/s

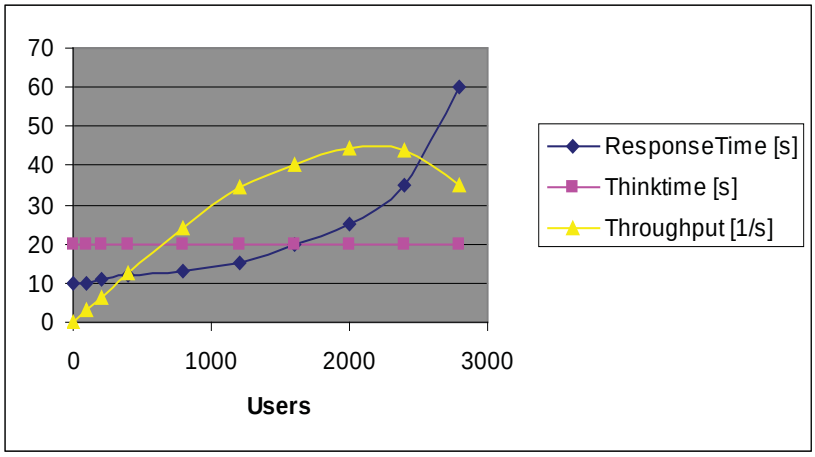
DIE PSYCHOLOGIE DER ANTWORTZEIT

- 3 Sekunden sind das Maximum für häufig durchgeführte Aktionen
- Eine «durchgängige» Antwortzeit ist anzustreben
- Benutzer bemerken nur lange Antwortzeiten
- Fortschrittsanzeigen sind wichtig
- Manchmal ist es auch möglich Teilresultate anzuzeigen



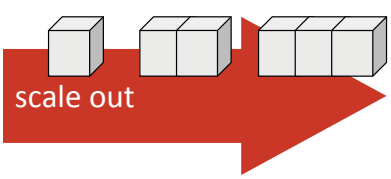
ANTWORTZEITEN UND DURCHSATZ UNTER LAST

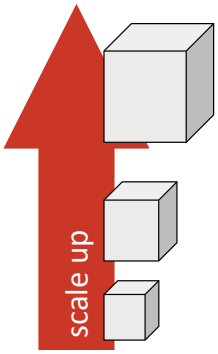
- There are n users, everyone thinks 20s, then starts a request, the response-time (for an unloaded system) is 10s



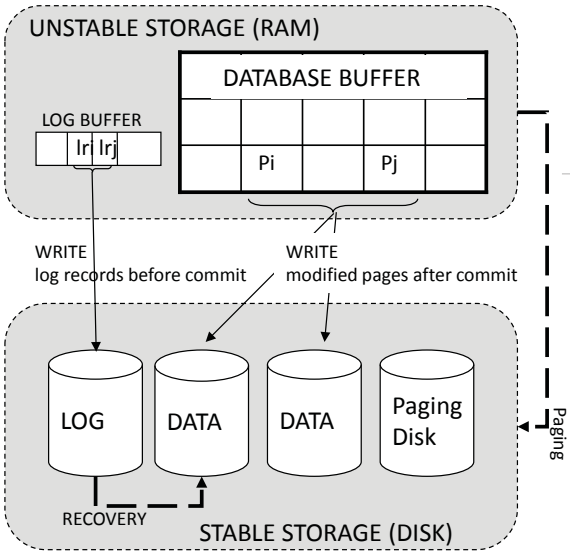
SKALIERBARKEIT SCALE-UP VS. SCALE-OUT

- Skalierbarkeit
  - Skalierbare Systeme können durch hinzufügen von schnellerer oder mehr Hardware wachsen ohne dass die Applikation verändert werden muss
- Idealerweise die Skalierbarkeit ist linear
- Es gibt zwei Arten zu wachsen
  - Scale Up
    - Prozesser, Memory oder Disk hinzufügen
  - Scale Out
    - Nodes hinzufügen um die Arbeit zu verteilen
    - Cluster oder Grid





DATENBANK PERFORMANCE



Die Anzahl Client-Prozesse können ein Problem sein

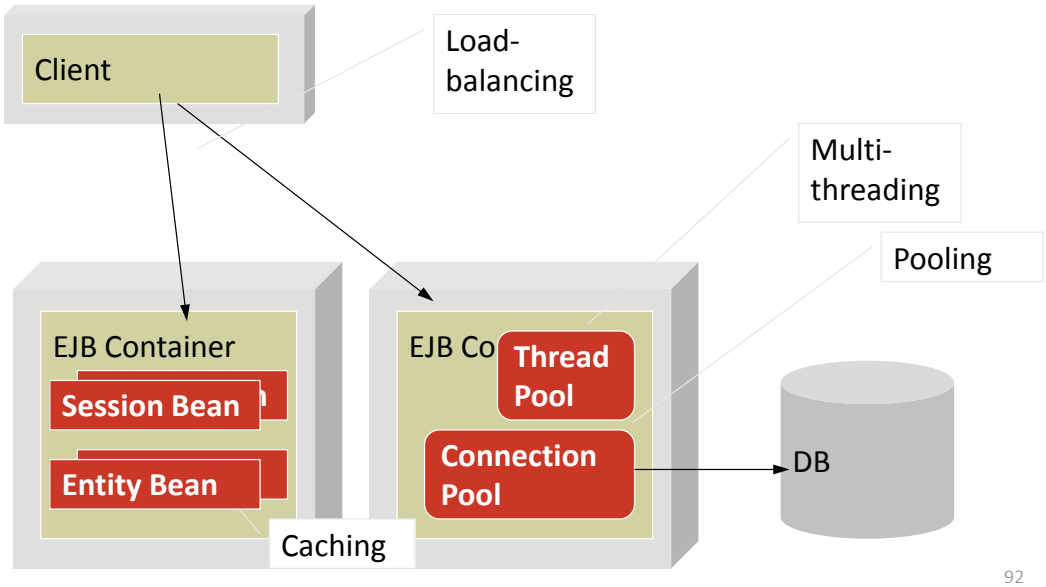
- Middleware mit Connection Pooling
- Multithreading

Der Datenbank Buffer ist ein Cache um Diskzugriffe zu minimieren

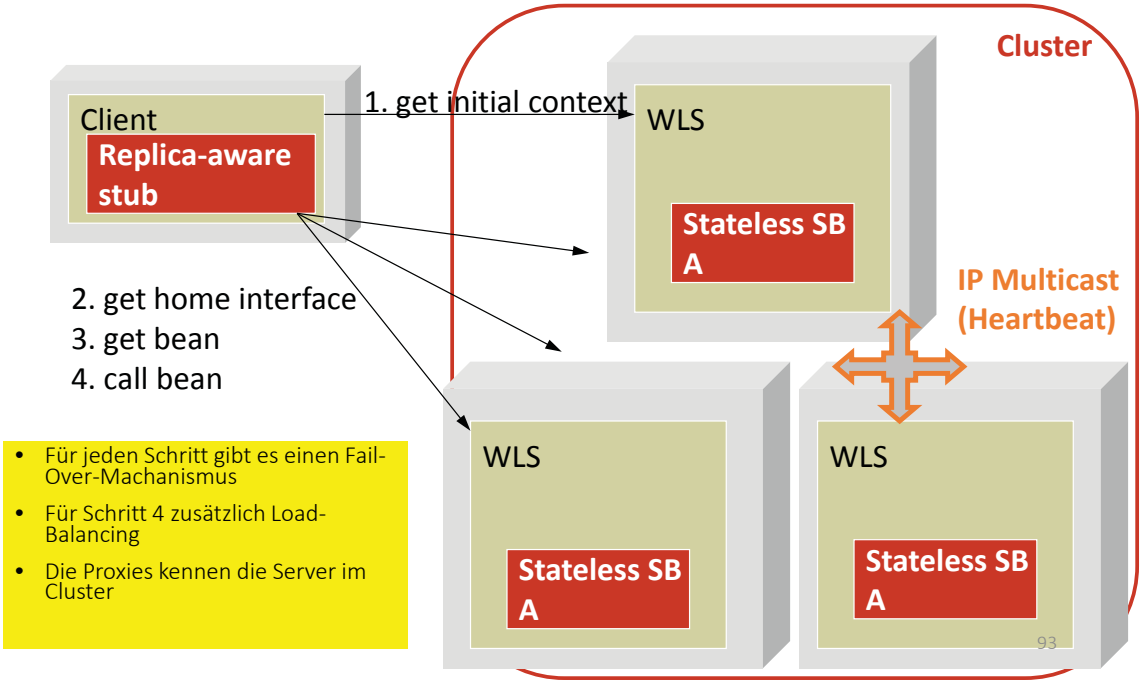
- Daten im Buffer werden von der Disk gelesen
- Updates werden zuerst ins Log geschrieben und dann batchmässig auf die Disk

- Separate Disks für die Logs verwenden
- Buffersize tunen

WAS TUT DER JAVA EE APP SERVER FÜR DIE PERFORMANCE?



LOAD-BALANCING UND FAIL-OVER



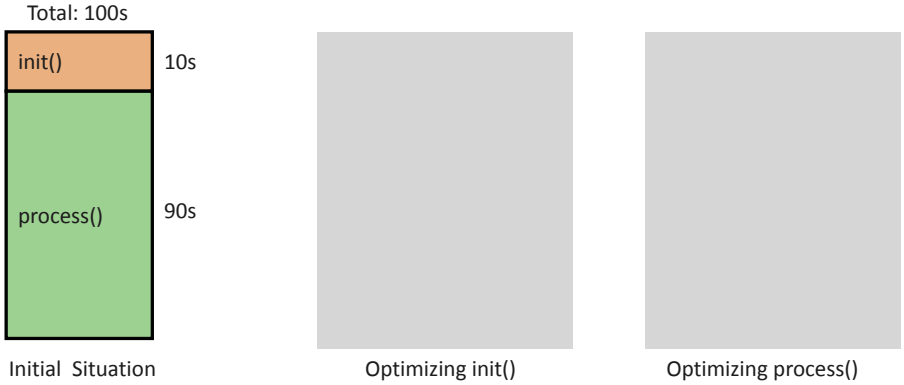
LOAD-BALANCING UND FAIL-OVER

- Load-Balancing Algorithmen
  - Round-Robin
  - Random
  - Weight-based
  - Die obgenannten mit Affinität (Sticky Sesssion)
    - Existiernde Verbindungen werden bevorzugt, damit nicht jeder Client eine offene Verbindung zu jedem Server hat
- Stateful Session Beans
  - Eine Bean-Instanz kann auf allen Nodes gehostet werden
  - Alle Zugriffe gehen auf den primären Node (Sticky Sesssion)
  - Die anderen Nodes werden über die Zustandsänderungen informiert

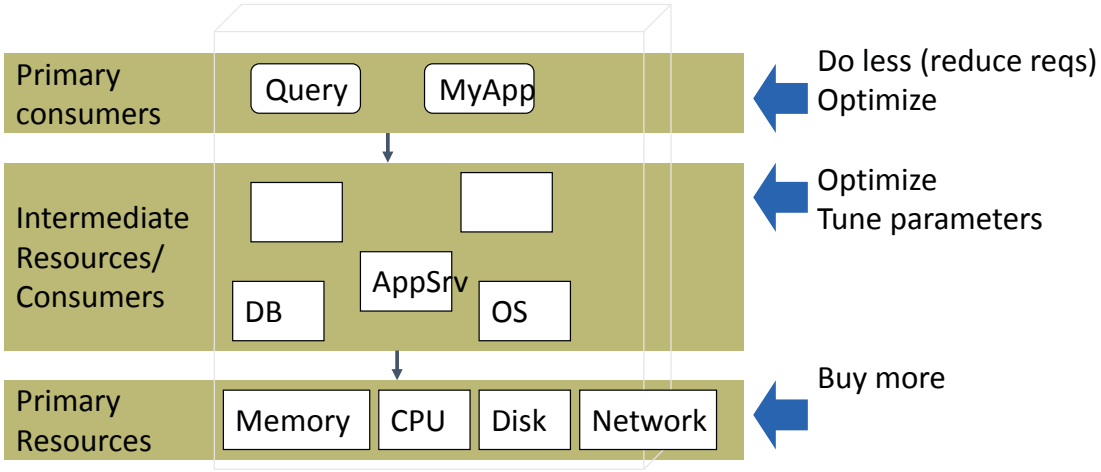
4.1 PERFORMACE-PROBLEME ANALYSIEREN

WO IST OPTIMIERUNG SINNVOLL?

- Problem
  - A program run takes 100 seconds, which is too long
  - There are 2 procedures: init() and process()
  - With the same engineering effort I can make run init() 5 times as fast as before or process() 2 as fast
- Question: which procedure should I optimize ?



CONSUMER-PRODUCER CHAIN



4.2 PERFORMANCE ENGINEERING STRATEGIEN

UEBERBLICK

- Schneller Infrastruktur benützen (mehr RAM, schneller CPU und Netzwerk)
- Konfigurationsparameter tunen (OS, DB, JVM)
- Last reduzieren
- Last verteilen
  - Clustering
  - Paralleliseriung
  - Arbeit verschieben (Defer work)
- Batching
- Caching
- Sharing / Pooling

EINFACH REGELN

- Think globally, fix locally
  - Ist es relevant? Wo fange ich an?
- Partitionierung vermeidet Bottle-Necks
  - Arbeit auf später verschieben, Last aufteilen
- Start-up costs are high, running costs are low
  - Disk Zugriff, Netzwerkkommunikation
- Be prepared for trade-offs
  - DB Index beschleunigt Selcts aber verlangsamt Inserts
  - Caching verursacht Probleme bei Verteilung
- Do less

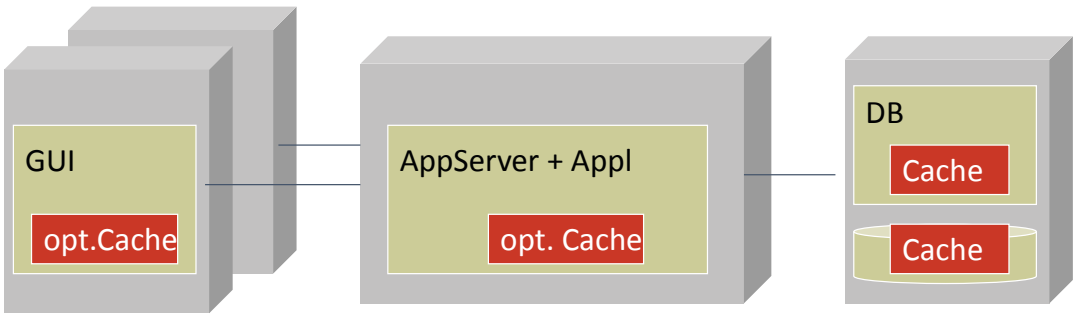
LAST REDUZIEREN

- Last = Frequenz der Ausführung * Kosten pro Ausführung
  - Anforderungen vereinfachen
  - Fokus auf teure Operationen (I/O, network calls, random disk access)
  - Fokus auf die am häufigsten ausgeführten Operationen
- Beispiele
  - Messages komprimieren welche über das Netzwerk gehen
    - Übertragung geht schneller
    - Trade-off: CPU vs. Network
  - Stored Procedures verweden um Netzwerkverkehr zu minimieren
  - Lokale Verarbeitung bevorzugen

CACHING

- Caching ist ein Trade-off:
  - Höherer Speicherverbrauch für
    - Weniger Netzwerkverkehr
    - Weniger Diskzugriffe
    - Weniger Prozessorlast
- Es gibt mehrere Ebenen wo Caching angewendet werden kann
- Achtung
  - Caching im RAM vs. Swapping
    - Cachegrösse muss überwacht werden
  - Datenaktualität
    - Cache aktualisieren

CACHING EBENEN



SHARING / POOLING

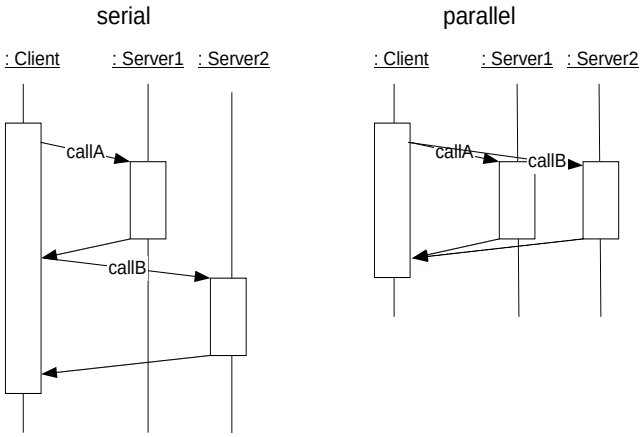
- Globale Datenstrukturen, verteile Caches
- Thread-Pools
- EIS / DB Connection-Pools
- Object Pools
  - avoid creating expensive objects (e.g. in Swing GUIs)

BATCHING

- Startup und/oder fixe Kosten „amortisieren“
  - Netzwerklatenz
  - Starten der JVM
- Keine feingranularen Aufrufe (Data Transfer Object)
- Effizientere Algorithmen oder Locking Strategien
  - Exklusiver Zugriff
    - Table-Lock
    - Isolationslevel reduzieren

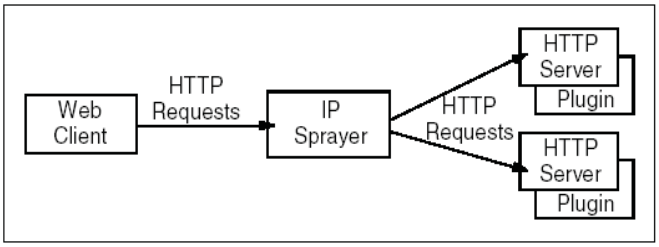
PARALLELISIERUNG

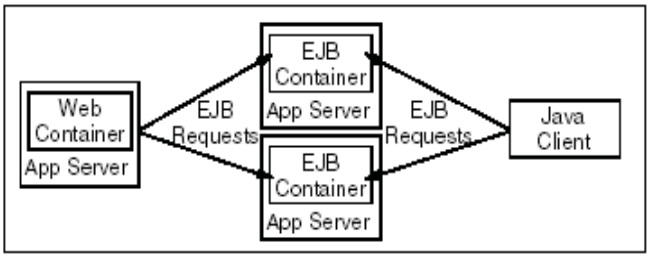
- Ist das auch eine Option, wenn Sie nur einen Server haben?
- Einfluss der Parallelisierung auf Antwortzeit und/oder Durchsatz?
- Was ist mit Multithreading in einem Client oder einem Batch-Prozess?
- Parallelisierung im Applikationsserver. Wie?



REPLIKATION UND LOAD-BALANCING

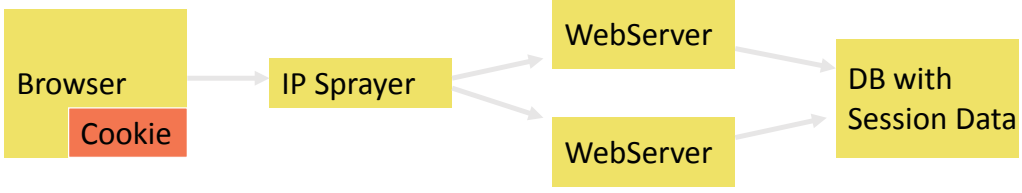
- Webserver
  - DNS Round-Robin
  - IP Sprayers
- App Server
  - Clustering
- Load-Balancing
  - Round-Robin
  - Random
  - Load-based





MANAGING HTTP SESSION STATE

- HTTP is a stateless protocol: each request is à priori independent from other request of the same user
- If state needs to be maintained between requests (e.g. shopping basket) then the session must be identified with a key (cookie, URL-rewriting)
- This key is used to access session data in RAM or on disk. Session data must be shared to enable replication (sharing via DB or in-memory synch by AppServer).

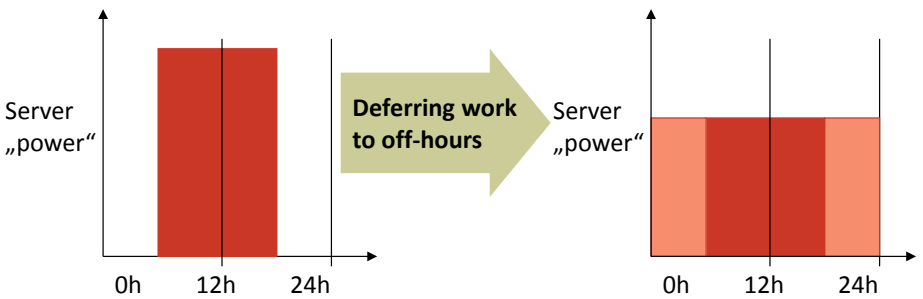


MANAGING EJB SESSION STATE

- Stateless Session Beans
  - Kein Zustand
- Stateful Session Beans
  - Der Client muss immer auf dieselbe Instanz zugreifen
- Message Driven Beans
  - Kein Zustand

DEFERRING WORK

- Nicht dringende Arbeit auf später verschieben
  - Verbesserung der Antwortzeiten
  - Weniger beanspruchte Zeiten können verwendet werden
- Beispiel
  - Audit-Einträge in Queue schreiben
  - Analyse und Speicherung in der DB auf später verschieben



4.3 PERFORMANCE TUNING

TUNING SCHRITTE

1. Performance-Ziele definieren
2. Fehler korrigieren, die das Ergebnis beeinflussen können (z.B. Memory Leaks)
3. Tests planen und Testumgebung aufbauen
4. Lasttest bauen
5. Performance mit einem einzigen Client messen. Code Profiling und Ausführungszeit messen
6. Code verbessern aufgrund der Erkenntnisse von 5
7. Performance und Durchsatz unter hoher Last messen
8. Code verbessern aufgrund der Erkenntnisse von 7
9. 5 bis 8 wiederholen

WAS BRAUCHE ICH ZUM TUNEN?

1. Ein realistische Testsystem
2. Die Anforderungen an Last und Performance
3. Realistische Datenmenge
4. Testautomatisierung und Lastgeneratoren
5. Messinstrumente

TYPISCHE TUNING PARAMETER

- Applikationsserver
  - Memory: Heap Size, Garbage Collector
  - Thread-, Connection- und EJB-Pool
  - Transaktions-Timeouts
  - Log Level
- Datenbank
  - Isolations Level, Lock Modi, Cache Grösse
  - Execution Plan, Statistiken
- Administratoren-Handbuch beachten!

4.4 ZUSAMMENFASSUNG

A CLEAN AND A FAST SYSTEM

- “Premature optimization is the root of all evil…“ [D. Knuth]

- Why?
  - optimization might not be necessary (wasted effort)
  - optimization is (often) bad for reliability and testability
  - optimization is (often) bad for understandability and maintainability
- What helps
  - getting a feeling for the current level of performance, using performance-oriented unit-tests and profiling your code once in a while

ZUSAMMENFASSUNG

- Konzepte: Antwortzeit, Durchsatz, Skalierbarkeit und Last
- Antwortzeit verschlechtert sich nicht-linear bei mehr Last
- Messungen sind nötig bevor Änderung gemacht werden dürfen
- Schlimmste Probleme zuerst anpacken
- An Performance denken, aber nicht zu früh optimieren

5 VERFÜGBARKEIT

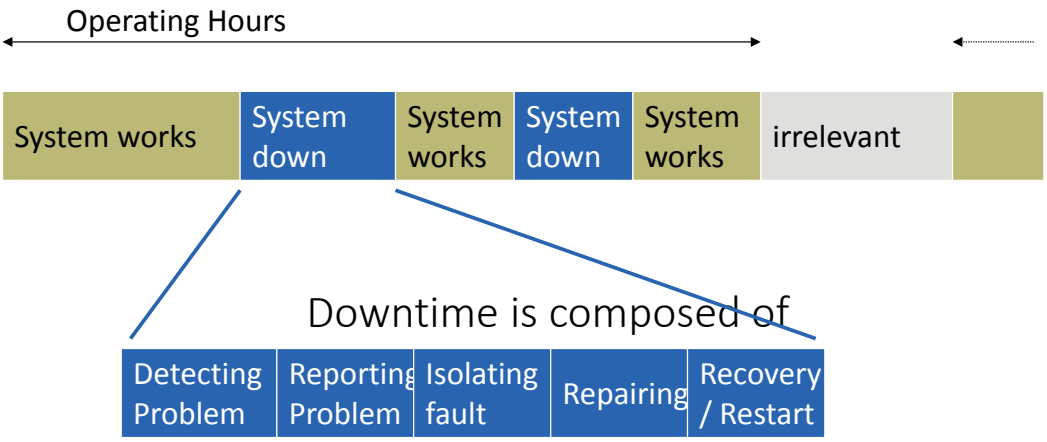
VERFÜGBARKEITSEBENEN

- Prozessverfügbarkeit
- Datenverfügbarkeit
- Kommunikationsverfügbarkeit

DEFINITION VERFÜGBARKEIT

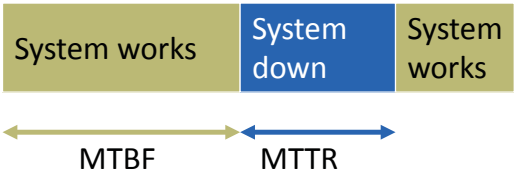
- Prozentsatz der geforderten Betriebszeit während denen das System normal funktioniert
  - Beispiele: 99%, 99.99%
- Betriebszeit
  - Zeitplan während dem das System verfügbar sein muss
  - Beispiele: Montag bis Freitag von 8h00 bis 18h00 oder 24h x 365d
- Ausfallzeit
  - Zeit während der das System nicht verfügbar ist
  - Es wird zwischen geplanten und ungeplanten Ausfällen unterschieden

VERFÜGBAREKIT



VERFÜGBARKEIT

- Statistische Angaben
  - MTBF Mean Time Between Failure
  - MTTR Mean Time To Repair



- Verfügbarkeit kann damit berechnet werden
  - MTBF / (MTBF + MTTR)

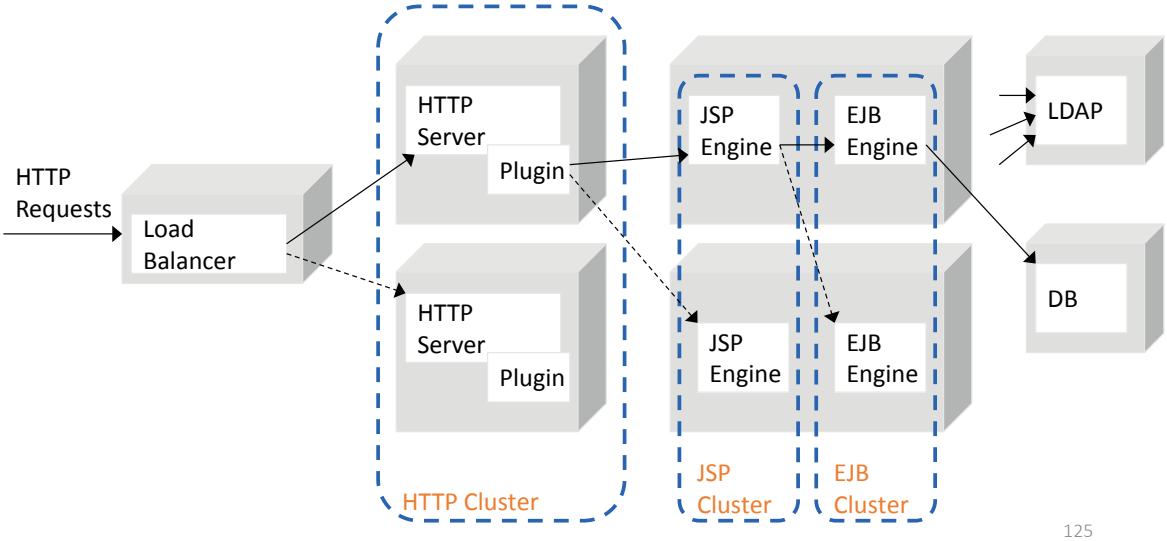
4.1 VERFÜGBARKEITS-STRATEGIEN

REDUNDANZ

- Die Frage
  - Basierend auf den Verfügbarkeitsanforderungen, was ist das billigste System um diese Verfügbarkeit zu erreichen?
- Der Schlüssel zu höherer Verfügbarkeit ist Redundanz
- Beachte:
  - Nur 10% aller Ausfälle ist Hardware-begründet
  - Ausfälle können nicht nur durch Hardware-Redundanz reduziert werden
  - Redundanz machen das System komplexer (15% aller Ausfälle werden durch Systemadministratoren verursacht)

REDUNDANZ

- Welche Single-Points-of-Failure bleiben?



VERFÜGBARKEITSEBENEN

Basic System	**No special measures. Regular backup. In case of failure system is restored from backup. Substantial amounts of user work may be lost.**
Redundant Data	**Disk-Redundancy (RAID) and/or mirroring. Restarting is faster (still needs DB recovery). Only in-flight user transactions may be lost.**
Component Failover	**Components of the system are deployed redundantly**
System failover	**Active/active or active/passive**
Disaster Recovery	**Maintaining Systems in different Sites, while keeping the data +- up-to-date (backups or mirroring)**

CLUSTERING

- Ein Cluster ist eine Gruppe von verteilten Komponenten die zusammen eine Einheit bilden

- Clustering wird gebraucht bei
  - Computer (IBM Sysplex, VaxCluster, Sun, Microsoft)
  - Datenbanken (e.g. Oracle Real Application Cluster)
  - Java EE Applikationsserver

- Die Cluster-Sematik kann variieren. D.h. der Cluster kann mehr oder weniger gut ein einzelnes System darstellen

SINGLE POINTS OF FAILURES

- Redundanz verbessert die Verfügbarkeit, daher muss den nicht redundanten Komponenten besonder Beachtung geschenkt werden

Failure Point	Possible solutions
Disk	**Disk mirroring, RAID-5**
Network Service (DNS etc.)	**Multiple DNS services**
Power Outage	**UPS**
NIC	**Multiple NICs in a Host**
Hub	**Multiple interconnected network paths**
OS / SW crash	**Clustering, switching to healthy node**
Firewall	**Firewall Cluster or high-availability Firewall**
Room disaster	**Put system in different room / building / city**
Operator error	**Train people, simplify system mgt**
Upgrades	**Rolling upgrades**

PROZESSREDUNDANZ

- Cluster-fähige Middleware
  - JEE Application Servers
  - Transaction Monitors (Tuxedo)
  - Message Queuing Systems
- Web-Applikationen mit Load-Balancer
- Do-it-yourself

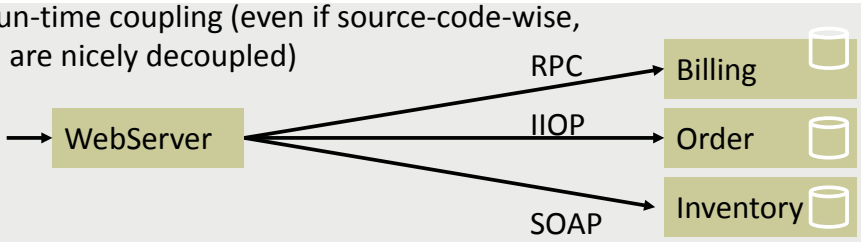
DATENREDUNDANZ

- Redundanz auf Diskebene
  - RAID 1, 5, 10
- Redundanz im Diskzugriff
  - Multihost Disks
  - SAN (Storage Area Network)
  - NAS (Network Attached Storage)
- Redundanz der Datenbank
  - Replication, Clustering

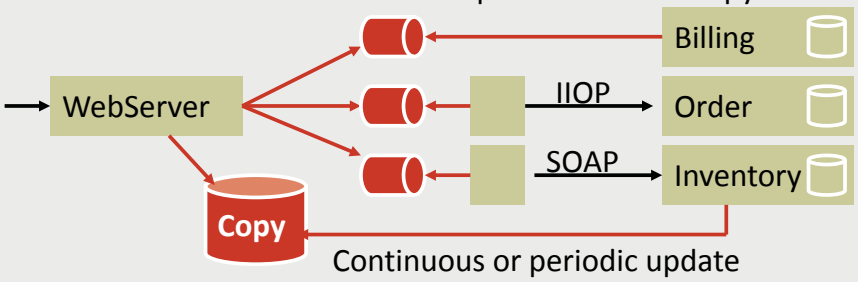
Redundancy may hide problems. Active monitoring must ensure that e.g. a defect disk in a RAID is detected

DE-COUPLING

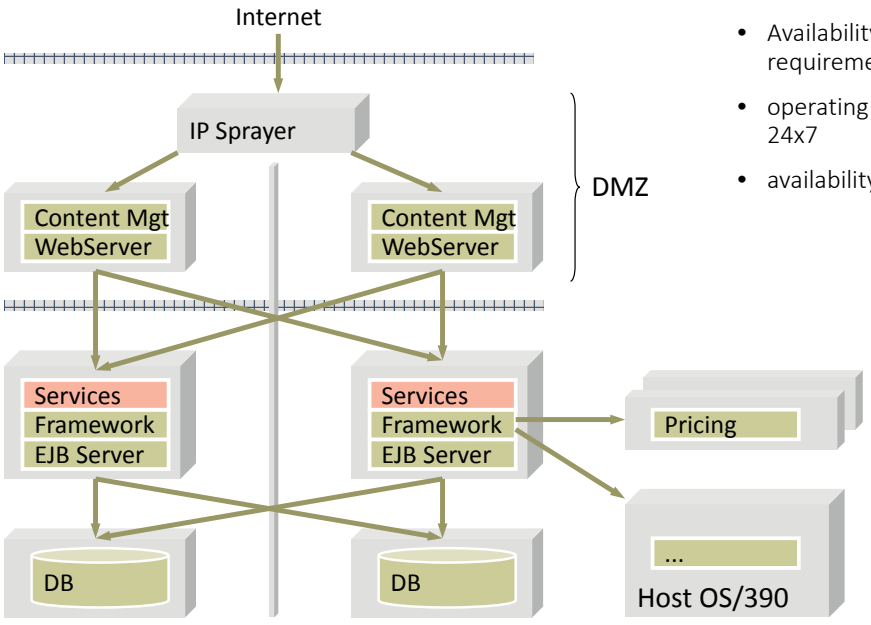
Tight run-time coupling (even if source-code-wise, system are nicely decoupled)



Communication via (highly reliable) intermediary decouples system. For basic user information some data is replicated in local copy.



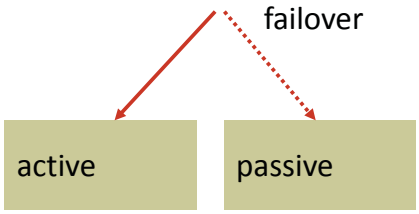
EXAMPLE: PROJECT XYZ

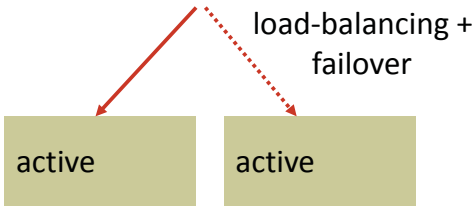


- Availability requirements
- operating hours: 24x7
- availability: >99%

VERFÜGBARKEIT UND PERFORMACE

- Generelle Prinzipien
  - Replikation
    - Redundanz für höhere Verfügbarkeit
    - Mehr Ressourcen für mehr Performance
  - Zugriffe verteilen
    - Fail-Over für höhere Verfügbarkeit
    - Load-Balancing für mehr Performance





ZUSAMMENFASSUNG

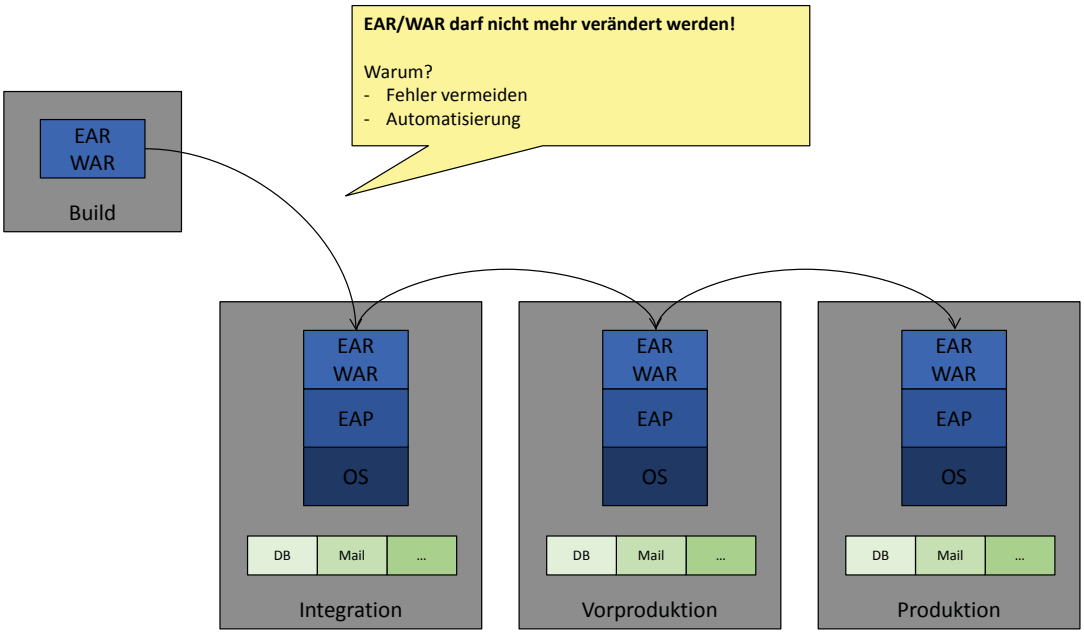
- Verfügbarkeit ist ein zentraler Aspekt moderner Systeme
- Verfügbarkeitsstrategien
  - Redundanz
  - Decoupling
  - MTTR reduzieren
  - Qualität (MTBF erhöhen)

6 KONFIGURATION

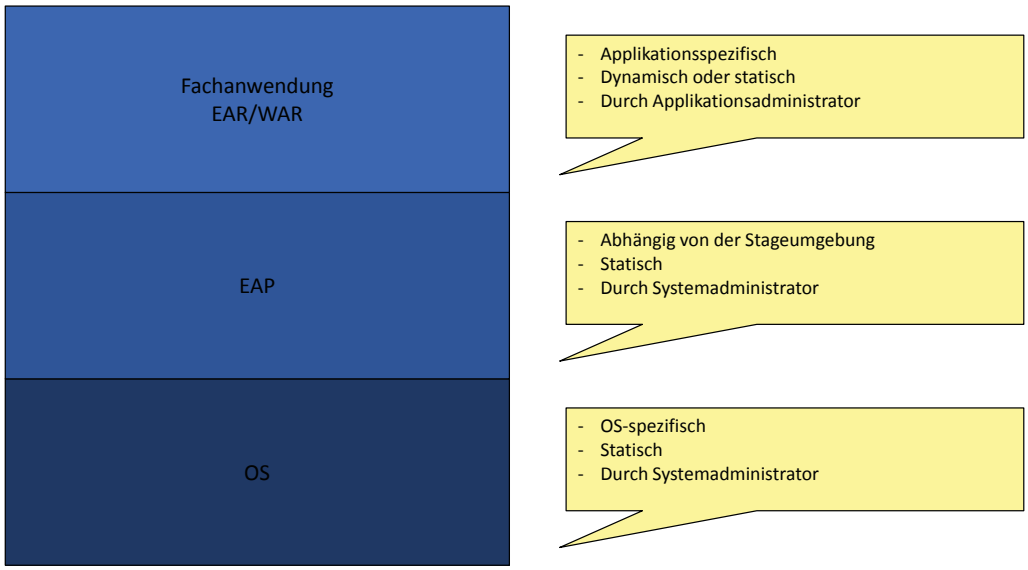
KONFIGURATIONSMANAGEMENT

- Grundsatz: «Das Deployment-Artefakt (EAR, WAR, JAR) darf bei der Installation nicht verändert werden»
  - Gilt auch bei Staging auf Entwicklung, Test, Integration und Produktion
- Fazit: Die Applikation muss konfiguriert werden können

STAGING



KONFIGURATIONSEBENEN



KONFIGURATIONSDATEN

- Format
  - JNDI
  - System Properties
  - XML
  - .properties Files
  - Datenbank
  - usw.
- Zugriff
  - Möglichst einheitlich
  - Dependency Injection nutzen

KONFIGURATIONSBEISPIELE

Scope	Wo	Art	Was	Beispiel
Applikation	EAR/WAR	Statisch	Applikationskonfiguration	Applikationsversion
Applikation	DB	Dynamisch	Applikationsspezifische Informationen	Mailempfänger
Plattform	EAP	Statisch	Schnittstellen zu Umsystemen	Webservice-URL
Plattform	OS	Statisch	Konfiguration des Servers	Hostname

Wo	Wie	Beispiel	Zugriff
EAR/WAR	.properties File	`applicationVersion=5.0.1`	`prop.getProperty("applicationVersion");`
DB	Tabelle	`emailKtAG | ag@agate.ch`	JDBC/JPA
EAP	System Properties	`<property name="WebServiceUrl" value="https://xy/">`	`System.getProperty("WebServiceUrl");`
OS	Environment Variable	`export HOSTNAME=evd-z3910`	`System.getEnv("HOSTNAME");`

IMPLEMENTATIONEN

- Apache DeltaSpike (CDI Erweiterungen)
  http://deltaspike.apache.org/documentation/configuration.html

- Beispielprojekt mit EJB Singleton
  https://github.com/simasch/config/

BEISPIEL DELTASPIKE

```
@ApplicationScoped
public class SomeRandomService
{
    @Inject
    @ConfigProperty(name = "endpoint.poll.interval")
    private Integer pollInterval;

    @Inject
    @ConfigProperty(name = "endpoint.poll.servername")
    private String pollUrl;

    ...
 }
```

7 MONITORING

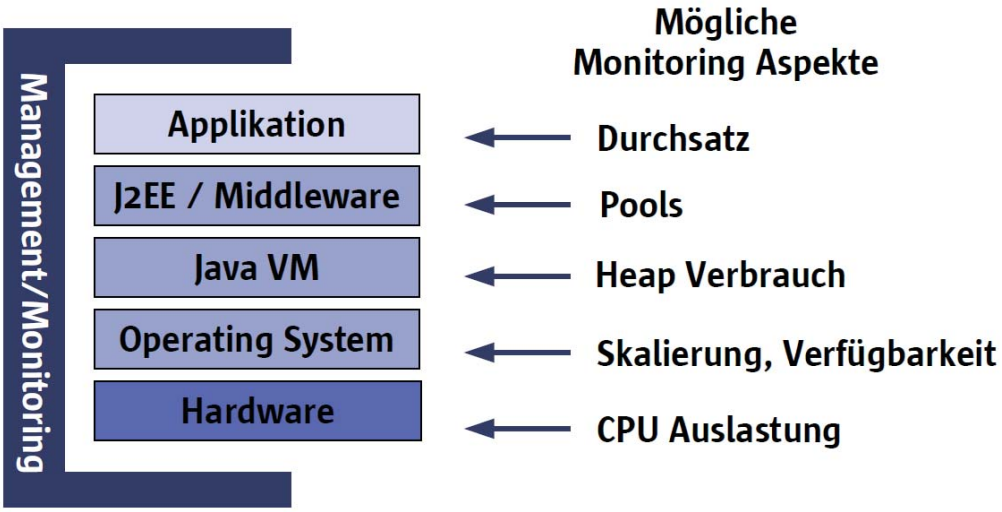
WAS IST MONITORING?

- Überwachen eines Systems bzw. einzelner Ressourcen im laufenden Betrieb z.B. für
  - Status Monitoring
  - Performance Analyse
  - Kapazitätsplanung
  - Diagnose
  - Debugging
  - Ausgangspunkt für Performance Tuning
  - Betriebsführung/Operating der Applikation

PROBLEME DES APPLIKATIONS MONITORINGS

- Ein Gesamtsystem besteht aus einer Vielzahl von Einzelkomponenten
  - Hardware, Storage, OS, Middleware, Applikation
  - Einheitliches Management über alle Layer ist kaum zu erreichen
- Unterschiedliche Standards für unterschiedliche Aspekte des Managements
  - SNMP, WBEM/CIM,...
- Management Produkte mit jeweils eigenen Protokollen und Schwerpunkten
  - IBM Tivoli, HP OpenView, Sun Solstice EM,...
- Monitoring und Management wird nicht von Entwicklern ausgeübt!

TYPISCHE JAVA EE SYSTEM ARCHITEKTUR



MONITORING AUF SYSTEM EBENE

- Operating System
  - Virtual Memory Benutzung (Swapping/Paging,...)
  - I/O Statistiken (KB RW/Subsystem)
  - CPU Daten (vol/invol. Context Switches, Usr/Sys load, Lastverteilung auf CPUen in SMP Systemen)
  - Netzwerkinterfaces (Retransmissions, KB/Sec, Listen Drops,...)
- Wie monitored man das OS?
  - Solaris/Linux/Unix:
  - Virtual Memory mit vmstat
  - I/O mit iostat
  - CPU Daten mit mpstat
  - Netzwerk mit netstat
  - Weitere Tools: prstat, top, dimstat, truss, sar, SE-Tools,kstat,...

MONITORING DER JAVA VIRTUAL MACHINE

- Java Virtual Machine
  - Garbage Collection Verhalten (Min/Max Heap, Young Generation vs. Old Generation, Perm Space, Tenuring Distribution)
  - JIT/Compiler Aktivitäten
  - Anzahl Threads, Thread Contention
- Wie monitored man eine (Sun HotSpot) JVM?
  - Java VM Flags:
    - Garbage Collection: -verbose:gc, -XX:+PrintHeapAtGC, -Xloggc:<filename>,
    - Compile Vorgänge: -XX:+PrintCompilation
  - Serviceability Agent
  - Profiling Agent
  - VisualVM

MONITORING EINES APPLICATION SERVERS

- Relevante Parameter eines Application Server
  - Thread Pools (Auslastung, Sizing, Shrinking)
  - Connection Pools (Größe, Statement Cache,...)
  - Object Request Broker (Threads, CBV/CBR)
  - EJB Container (Pools/Caches, Passivierungen von SF SB's)
  - ...
- Wie monitored man einen Application Server?
  - Keine Standards, jeder Server bringt eigene Tools mit sich
  - Meist Web basiert, ergänzt durch CLI Tools
  - Manchmal SNMP Adapter, die jedoch meist nicht alle Parameter monitoren lassen

PROBLEME BEIM MONITORING

- Derzeit meist kein ganzheitliches Management des Systems möglich
  - Meist wird nur der Application Server und evtl. die JVM überwacht
  - Applikationen/Komponenten als Black Box
    - Häufig bieten die Admin Tools nur Deploy/Undeploy Features
- Wie kann man die eigene Applikation überwachen?
  - Häufig muss derzeit auf 3rd Party Applikationen zurückgegriffen werden
  - Entwickler greifen oft zu selbst entwickelten Management Lösungen
    - Code Instrumentierung mittels System.out.println() oder Tracing Features
    - Meist unflexibel, schwer wartbar und oft statisch

7.1 JMX

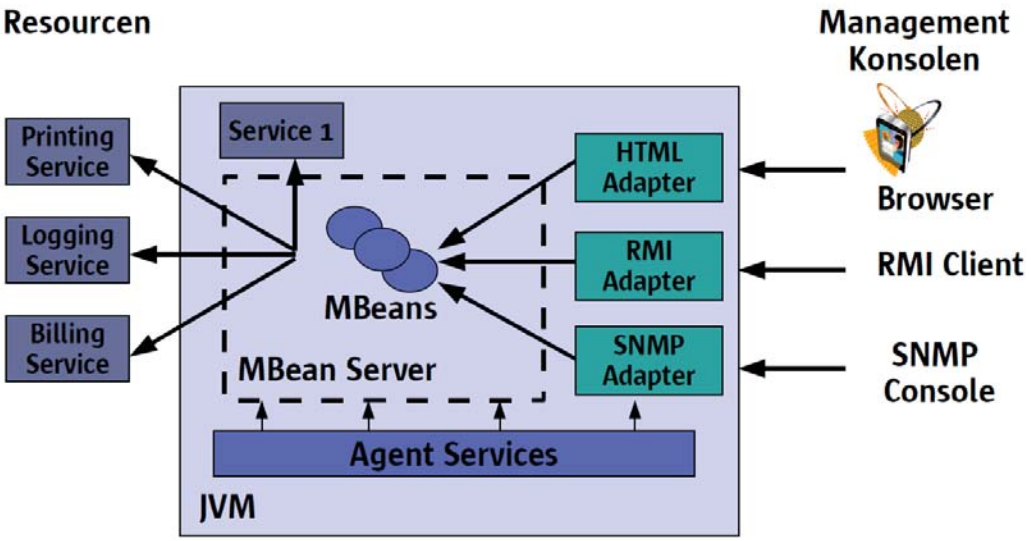
JAVA MANAGEMENT EXTENSIONS

- JMX ist Bestandteil von Java SE
  - Enstanden aus dem Produkt Sun Java Dynamic Mangement Kit
- JMX besteht aus einer Architektur, einer API und Services für das Management von Java Applikationen
  - Ermöglicht das Überwachen von Java Applikationen mit Standard Management Konsolen
  - Wird kontinuierlich im Java Community Process weiterentwickelt
  - Bewährte Technologie
  - Findet sukzessive Einzug in neue Java Plattformen
  - Kann auch in J2EE Servern benutzt werden

JMX TERMINOLOGIE

- Instrumentierung
  - Realisierung von managebaren Ressourcen als JMX MBeans
- MBean
  - Designpattern für managebare Java Beans
- MBeans Server
  - In Process Registry für MBeans
- JMX Agent
  - Service Container für MBeans und Server
- Connectors/Adapter
  - Stellen Protokolladapter für Remote Zugriff dar

JMX ARCHITEKTUR



TYPISCHE EINSATZGEBIETE FÜR JMX

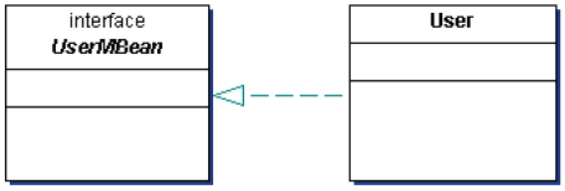
- Auslesen/Ändern von Applikationsparametern zur Laufzeit
- Zur Verfügung stellen von Statistiken
  - Performance
  - Ressourcenbenutzung
  - Logs/Probleme
- Zustellung von Nachrichten
  - Fehler
  - Zustandsänderungen
- Automatisches Monitoring kritischer Systemkomponenten

INSTRUMENTATION LEVEL

- Alles was als Java Objekt darstellbar ist kann mit JMX instrumentiert werden (HW, SW, Services,...)
- Die Instrumentierung erfolgt über MBeans, von denen JMX verschiedene Varianten vorsieht
  - Standard MBeans
  - Dynamic MBeans
  - Model MBeans
  - Open Mbeans

STANDARD MBEANS

- Einfachster Weg für JMX Instrumentierung
  - Management Interface wird vom User explizit spezifiziert
- Agent benutzt Introspection für das Erkennen der Management Interfaces
- Verpflichtend seit JMX 1.0
- Vererbung wird unterstützt
- Nachteil: Statisches Interface



KONVENTIONEN

- JMX spezifiziert Namenskonventionen für Standard MBeans
  - Die managebare Ressource muss ein Interface mit gettern/settern für R/W Attribute bzw. gettern für RO Attribute spezifizieren
  - Interface und Implementierung müssen im gleichen Package sein
  - Das Interface trägt den Namen <Ressource>MBean
  - Die Klasse <Ressource> muss das Interface implementieren
  - Exponiert werden Attribute, Methoden, Konstruktoren

MBean

```
public interface BatchControllerMBean {

  /** Anzeigen des Status des BatchControllers */
  String getStatus();

  /** Stoppt den BatchController */
  void stop();

  /** Startet den BatchController */
  void start();
}
```


MBean REGISTRIERUNG

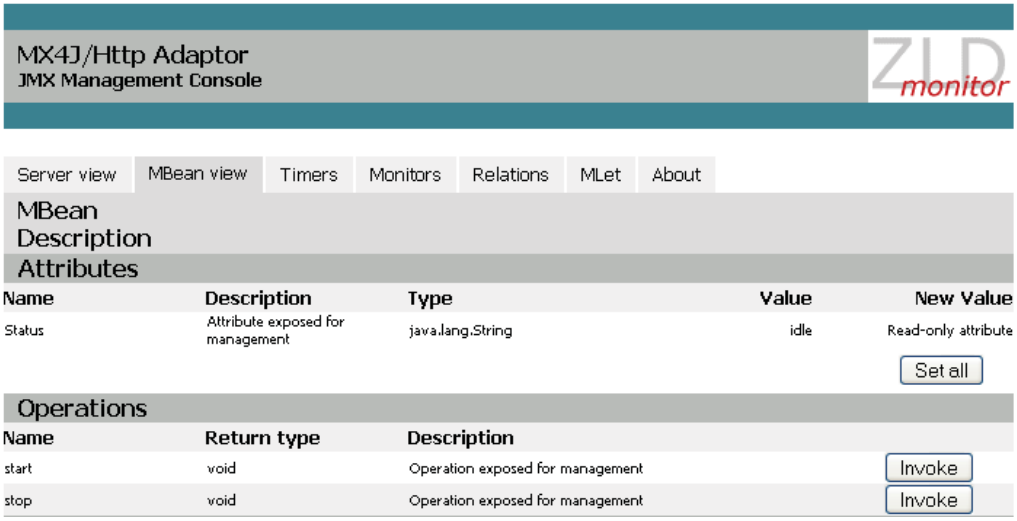
```
//Create MBeansServer
MBeanServer mbs = MBeanServerFactory.createMBeanServer();

//Create and Register HTML Adapter
ObjectName name = new ObjectName("mx4j:id=mx4j");
HttpAdaptor adaptor =
          new mx4j.tools.adaptor.http.HttpAdaptor(8112, "0.0.0.0");
mbs.registerMBean(adaptor, name);

// Register BatchControllerMBean
BatchController batchController = new BatchController();
ObjectName on = new ObjectName("ZLD Controller:id=zld");
mbs.registerMBean(batchController, on);

//Start the WebServer
mbs.invoke(name, "start", null, null);
```

BEISPIEL GUI



JMX UND JAVA EE?

- JMX wurde für Java SE entwickelt bevor es Java EE gab!
- Es gibt derzeit keine Standard Integration von JMX in Java EE Umgebungen
- Viele Java EE bieten jedoch bereits seit J2EE 1.3 eine JMX Implementierung
  - Beispiele: JBoss, Oracle, IBM
  - Häufig wird JMX als Backbone der Application Server Implementierung verwendet
    - Web/EJB-Container, Threadpools etc. sind als MBeans instrumentiert um Performancestatistiken zu liefen oder um die Konfiguration vorzunehmen
  - JMX kann jedoch auch für eigene Java EE Komponenten benutzt werden

TYPISCHE JAVA EE MONITORING AUFGABEN

- Performance Daten
  - Ermitteln der Aufrufzahlen für Servlets/JSP
  - Antwortzeiten (min,max,avg)
- Transaktionen
  - Anzahl von Commits/Rollbacks pro EJB
  - Durchschnittliche Anzahl von Beans die gelockt sind
- Connection Pools
  - Anzahl von Connections im Pool
  - Avg. Wartezeit auf Pool-Connections
- Überwachung der eigenen Applikation
  - Backend Ressourcen
  - Fachliche Properties

JMX IN JAVA EE SERVERN

- Die JMX Instrumentierung der gängigen J2EE Server lässt sich für zahlreiche eigene Monitoring Aufgaben verwenden
  - Es können eigene MBeans für die Applikation in die MBeans Server registriert werden
    - z.B. Um dynamisch Traces zu triggern
  - Monitoring/Timer MBeans können zur Überwachung des Servers oder von eigenen kritischen Ressourcen eingesetzt werden
    - z.B. E-Mail Notifikation bei Überschreitung von Schwellwerten
  - Fortlaufendes Monitoring für Performance Degration
  - Monitoring eigener Connection Pools oder Resource Adapter
  - Erhöhen von gepoolten Ressourcen
  - Umschalten von Log Levels zur Laufzeit

VORTEILE VON JMX

- Verwendung existierender Technologien
  - Adapter für RMI, IIOP, HTTP
- Notifikationsmechanismen
  - Events, Alerts und Listener
- Management muss nicht selbst entwickelt werden
  - Entwickler müssen keine Adapter bzw. Protokolle implementieren
  - Aufwand für MBean Instrumentierung ist überschaubar
  - JMX Adapter/Konnektore erlauben flexible Integration unterschiedlicher Management Konsolen