

## E Java & Komponentenmodelle & Jini

### E.1 Überblick

- Software-Komponenten
- Komponentenmodelle
- Java & Komponentenmodelle: JavaBeans
  - Architektur
  - Properties
  - Ereignisse (Events)
  - Introspection
- Jini
  - ein verteiltes Komponentenmodell
- Anwendbarkeit von Software-Komponenten

### E.2 Literatur

- Edw01. W. Keith Edwards. *Core Jini*. Prentice Hall, Upper Saddle River, NJ, 2nd Edition 2001.
- Edw00. W. Keith Edwards. *Core Jini*. Prentice Hall, München, 2000 (deutsche Übersetzung der ersten Auflage, umfasst ebenfalls bereits Jini 1.1).
- Szy98. Clemens Szyperski. *Component Software — Beyond Object-Oriented Programming*. Addison-Wesley, Harlow, England, 1998.
- Grif98. Frank Griffel. *Componentware — Konzepte und Techniken eines Softwareparadigmas*. dpunkt-Verlag, Heidelberg, 1998.
- JavaBeans. <http://www.sun.com/beans/>
- Jini. <http://www.sun.com/jini/specs/>

## E.3 Komponentenmodelle

### 1 Software Komponenten

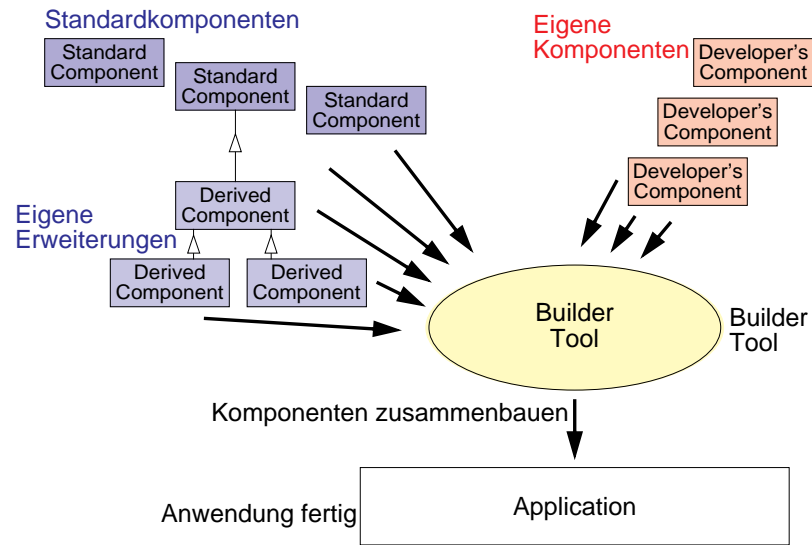
- Definition (Szyperski)  
Eine Softwarekomponente ist ein wiederverwendbares Stück Software
  - hat eine gut spezifizierte Schnittstelle
  - kann in nicht-vorhergeplanten Kontexten eingesetzt werden
  - ist eine eigenständig vermarktbar Einheit
- Beispiele
  - Elemente einer Fenster-Oberfläche
  - Software-Modul, das ein Gerät repräsentiert (Messgerät, Drucker, ...)
  - Rechtschreibprüfungs-Modul für Textverarbeitungssysteme

? also einfach Klassen im objektorientierten Sinn —  
oder was ist da neu dran

## 2 Softwarekomponenten (2)

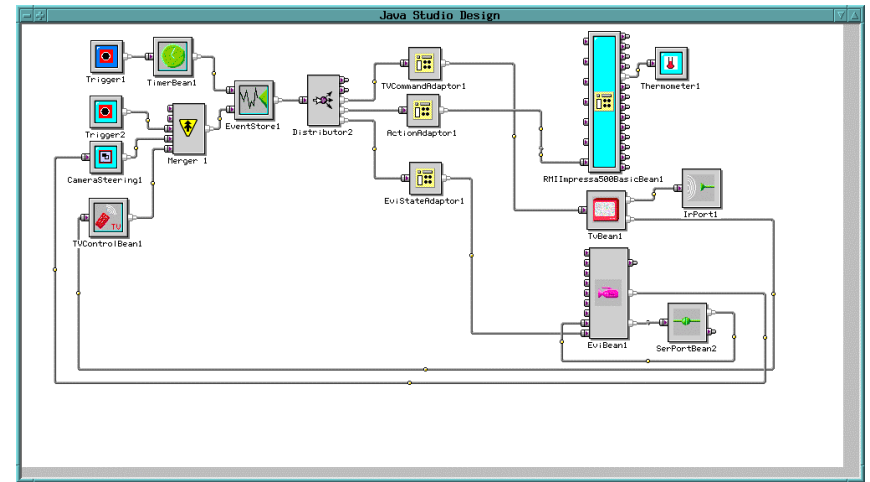
- Standard-Konventionen für die Schnittstellen von Software-Komponenten
  - ↑ **Komponenten-Modell**
  - ↳ Einfache Verwendbarkeit + Wiederverwendbarkeit
- Software-Komponenten sind selbst-beschreibend
  - Automatische Analyse von Schnittstelle und Eigenschaften möglich
    - Introspection / Reflection - Mechanismen
    - Interface-Repository
    - + Namens-Konventionen
- Zusammenstellung zu komplexen Anwendungen mit grafischen Werkzeugen (Builder-Tools)
  - Software-Baukasten

### 3 Philosophie



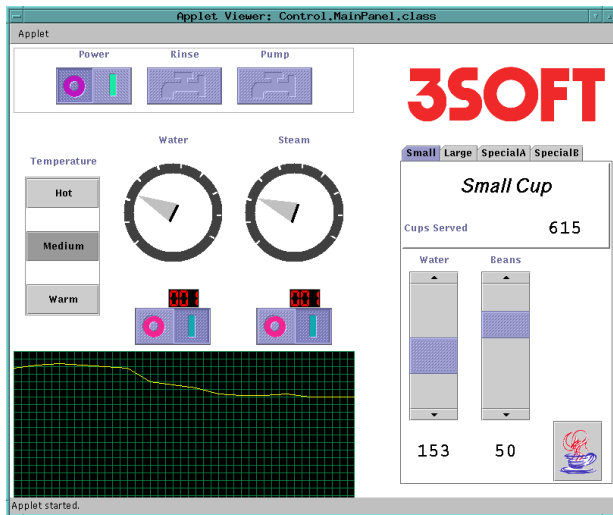
### 4 Beispiel für ein Builder Tool

#### ■ JavaStudio



## 5 Beispiel für eine Anwendung

### ■ Bedienoberfläche unserer Kaffeemaschine



## 6 Komponentenmodell

- Richtlinien für Software-Komponenten
  - Namenskonventionen
  - Repositories, Introspection-Mechanismus
  - Schnittstellen-Semantik
  - Programmiermethodik
- Existierende Komponentenarchitekturen
  - ◆ JavaBeans
  - ◆ CORBA Component Model
  - ◆ ActiveX
    - nicht portabel, proprietär
  - ◆ OpenDOC
    - ziemlich tot
  - ◆ Hersteller-spezifische Speziallösungen
    - GUI-Builder Klassenbibliotheken

## E.4 Java & Softwarekomponenten

### 1 Java — Ziele

- Lösung der verbreiteten Probleme bei der Entwicklung und Verteilung von Software
  - ◆ verschiedene Betriebssysteme (Unix, Windows, MacOS, ...)
  - ◆ verschiedene Hardware-Architekturen
- Java: Sprache und Ausführungsumgebung für *sichere, schnelle und sehr robuste Anwendungen auf verschiedenen Plattformen in heterogenen verteilten Netzen*

## 2 Java — wesentliche Eigenschaften für Komponenten

- objektorientiert
- Polymorphismus bei Methodenaufrufen
- sehr flexibel (dynamisches Laden und Binden)
- Reflection/Introspection Mechanismus

### 3 Java-Komponentenmodelle

- JavaBeans
- Jini

## E.5 JavaBeans

### 1 Definition

- JavaBeans ist eine Schnittstellen-Spezifikation für wiederverwendbarer Software-Komponenten in Java
  - definiert die Java-Komponenten
  - und wie sie zusammenarbeiten
- Eine Bean ist eine beliebige Java-Klasse, die den JavaBeans-Konventionen folgt
- Die offizielle Definition:
 

*A Java Bean is a reusable software component that can be visually manipulated in builder tools.*

## 2 Architektur

- Eigenschaften (Properties)
  - initiale Einstellung von Eigenschaften (Zustand) einer Bean
- Methoden  
Ereignisse (Events)
  - die Verknüpfungspunkte für Beans
- Adapter
  - zur Schnittstellenanpassung, wenn Beans nicht einfach zusammenpassen
- Introspection
  - statt eines Interface-Repositories:  
einfach in die Bean reinschauen was sie kann

### 3 Beispiele

- Grafische Beans einer Benutzeroberfläche
  - Knöpfe, Regler, Textfenster
  - HTML rendering Bean
  - OpenGL canvas
- Unsichtbare Beans
  - Datenbank-Schnittstelle
  - Timer
    - löst Ereignisse in bestimmten Abständen aus
    - kann komplexe Zeit-Logik kapseln
- Softwarekomponenten, die Geräte/Geräteteile repräsentieren
  - Schalter
  - Sensoren, Aktuatoren
  - Video-Rekorder
  - Kaffeemaschine

### 4 Properties

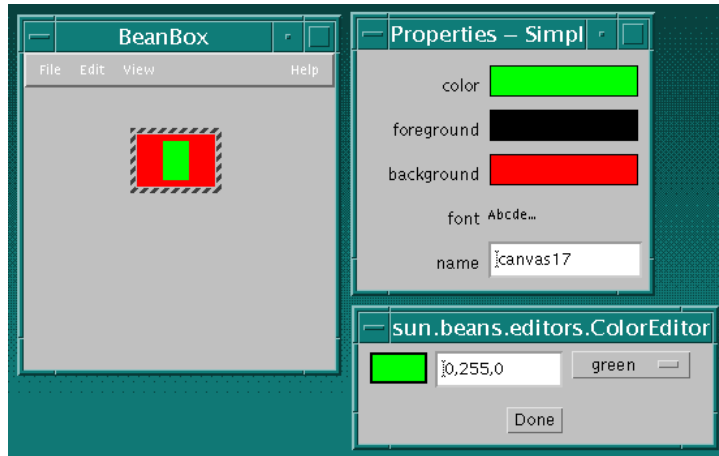
- Beschreiben Eigenschaften von Komponenten
- Jede Property hat
  - einen **Namen** — symbolische, aussagekräftige Beschreibung der Eigenschaft (Color, Font, ...)
    - `private Color color;` (Instanzvariable der Bean-Klasse)
  - einen **Typ** — Java-Klasse, kapselt den Wert
  - Constraints — optionale Einschränkungen (z. B. read-only)
- Namens-Konventionen für Methoden zum Zugriff (Methoden der Bean-Klasse)
  - get-Methode zum Lesen
 

```
public Color getColor(){ return color; }
```
  - set-Methode für Modifikationen
 

```
public void setColor(Color newColor){
    color = newColor;
    repaint();
}
```

## 4 Properties (2)

- Beispiel: Property Color der Bean "SimpleBean"
  - SimpleBean wird in BeanBox geladen, Feld color anklicken öffnet ColorEditor



## 4 Properties (3)

- Simple properties
  - repräsentieren einen einzelnen Wert, Zugriff mit set/get-Methoden
- Indexed properties
  - repräsentieren ein Feld, set/get-Methoden haben einen index-Parameter
- Bound properties
  - informieren andere Objekte über Zustandsänderungen (PropertyChange event)
- Constrained properties
  - nicht erlaubte Zustandsänderungen können zurückgewiesen werden



## 5 Events

### ■ Umsetzung des Source-Listener Design-Patterns

- Source-Objekt informiert Listener über Zustandsänderungen
  - EventSource
    - hat Methoden für Listener, um sich zu registrieren/abzumelden

```

public void addTimerListener(TimerListener l)
public void removeTimerListener(TimerListener l)

```

gleicher Name

add/remove      TimerListener muss spezielles  
EventListener-interface implementieren

- EventObject
  - kapselt Informationen über Ereignis
- EventListener
  - Klasse die EventListener-Interface implementiert (Subtyp von `java.util.EventListener`)

## 5 Events (2)

### ■ PropertyChangeSupport

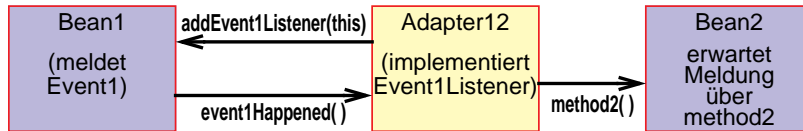
- automatisches Versenden von notification-Events, immer wenn sich der Wert einer Property ändert

### ■ VetoableChangeSupport

- ermöglicht das Zurückweisen von Property-Werten, die außerhalb des gültigen Bereichs liegen

## 6 Adapter

- Anpassung von Methoden einer Bean an Ereignisse einer anderen



- Adapter12 implementiert passendes Event-Listener Interface
- Adapter12 registriert sich für Event1
- Event1 findet statt, Bean1 ruft Methode die im event-Interface festgelegte Methode (`event1Happened()`) bei allen registrierten Event-Listener-Objekten auf
  - `event1Happened()` im Adapter12 ruft `method2()` an Bean2 auf
- Adapter-Objekt kann Daten evtl. manipulieren (z. B. umrechnen)
- Einfache Adapter können automatisch erzeugt werden

## 7 Introspection

- Automatische Analyse von Eigenschaften einer Bean
- Java1.1 Reflection API
  - Analyse von Java-Klassen zur Laufzeit
  - Instanzvariablen: Name & Typ
  - Methoden: Name, Parameter, Ergebnis-Typ
- JavaBeans Namens-Konventionen
  - get/set Methoden → Properties
  - add/remove Methoden → Events
  - andere Methoden → normale Methoden
- Alternative: Information in einer BeanInfo-Klasse
  - ◆ ähnlich wie ein Interface-Repository
  - Entwickler kann Properties und Events explizit spezifizieren

## 8 JavaBeans — Summary

- Beans = zusammensteckbare Software-Bausteine
  - Zustand = Properties
  - Eingänge = Methoden
  - Ausgänge = Events
  - wenn Stecker-Buchse nicht passen: Adapter
- ➔ z. B. ideal zur Software-Repräsentation von Geräten
- Probleme / Defizite
  - Zusammenstecken einfacher Komponenten zu komplexen Komponenten
    - ↳ hierarchische Beans
  - Ankoppeln neuer Software-Komponenten an laufende Anwendungen
  - Software-Komponenten im verteilten System

## E.6 Jini

### 1 Überblick

- Standard für die Kommunikation zwischen "intelligenten" Geräten (Sun 1998)
- Komponentenarchitektur für verteilte Systeme
  - Komponenten (Services) registrieren sich bei einem Lookup-Service
  - Klienten erhalten von dort Referenzen (RMI-Stub oder *Smart Proxy*)
  - Referenzen sind zeitlich begrenzt gültig (*Leasing*)
- Ziel: Geräte können sich flexibel an Netzwerk an- und abmelden und finden automatisch ihre Kommunikationspartner
  - Komponenten versuchen selbst die Verbindungen zu schalten
  - Konfiguration durch Builder-Tool nicht vorgesehen

## 2 wesentliche Konzepte

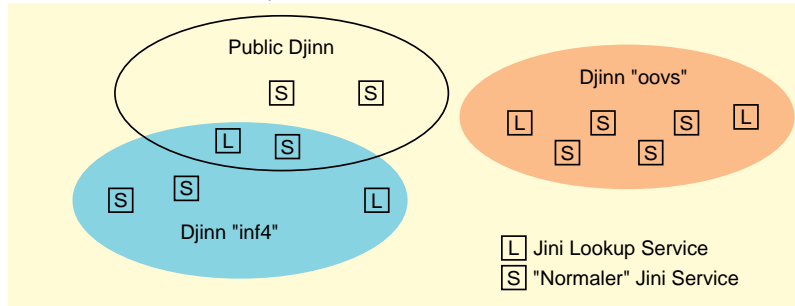
- Service
- Community
- Federation
- Discovery
- Lookup Service  
(the naming service)
- Leasing
- Events
- Security
- Java RMI
- Transactions

## 3 Jini Services

- Mitglieder einer Jini-Community (*djinn*) haben gemeinsam Zugriff zu Diensten (Services)
- Service = etwas, das genutzt werden kann von:
  - einer *Person*, einem *Programm* oder einem anderen *Service*
  - Berechnung
  - Speicher
  - Kommunikationskanal
  - Software Filter
  - Gerät (Hardware)
  - ein anderer Benutzer
- Services werden bei einem Lookup Service registriert
  - Clienten können sich dort eine Referenz besorgen  
(Stub einer RMI-Remote-Referenz oder *Smart Proxy*)
- Services werden zur Durchführung einer bestimmten Aufgabe zusammengestellt

## 4 Djinn

- Gruppe von Jini Services, auch Community oder Federation genannt
- Alle Services in einem Djinn sehen sich



- Flacher Namensraum der Djinn

## 5 Discovery

- Auffinden von Jini Federations
  - Suche von verfügbaren Lookup-Diensten
  - Beitritt zu einer Federation: Teilnahmeprotokoll
- Verschiedene Discovery-Protokolle
  - Multicast Request Protocol  
Finden von Lookup-Diensten
  - Multicast Announcement Protocol  
Ankündigung eines Lookup-Dienstes
  - Unicast Discovery Protocol  
Anmeldung bei bekanntem Lookup-Dienst

## 6 Leasing

- Teilweiser Ausfall in einem verteilten System führt zu speziellen Problemen
  - Referenzen werden ungültig
  - Betriebsmittel können nicht freigegeben werden
- Services in Jini werden für eine bestimmte Zeit "gemietet"
  - Lease = Zugriffsgarantie zu einem Service für eine bestimmte Zeitspanne
  - kann erneuert werden, wenn sie länger benötigt wird
  - Erneuerung kann vom Service-Provider verweigert werden
  - Lease läuft ab wenn sie der Client nicht erneuert (oder wegen eines Fehlers nicht erneuern kann)
  - nach Ablauf der Lease können die zugehörigen Ressourcen freigegeben werden

## 7 Ereignisse (Events)

- JavaBeans unterstützt nur lokale Events
  - Objekt, das den Event auslöst und alle empfangenden Objekte müssen innerhalb einer JVM laufen
- Jini unterstützt verteilte Events
  - Registrierung erfolgt über RMI-Aufruf und der Übergabe eines Stubs
- Jini-Events werden nur solange zugestellt wie die zugehörige Lease gültig ist

## 8 Sicherheit

- Sicherheitsmodell auf der Basis von *principal information* und *access control lists*

## E.7 Realistische Anwendungen für Komponenten-Software?

- Anwendungen, die zwischen Programmierung und Einsatz umfangreiche Konfiguration erfordern
  - Eingabe/Abfrage-Oberflächen (z. B. SAP-Systeme)
  - Steuerungssysteme  
flexible Test-Systeme
    - ➔ Aufbau aus Software-Komponenten, die die Geräte repräsentieren (z. B. Gebäudeinstrumentierung)
- + Konfiguration erfordert wesentlich weniger Spezialwissen
- + Anwendungsarchitektur offener
  - ➔ kundenspezifische Erweiterungen einfacher realisierbar
- ➔ Probleme bei hochdynamischen Anwendungssystemen (Objekte verschwinden, neue kommen hinzu)

## E.8 Software-Komponenten für realistische Anwendungen?

- Klassenbibliotheken mit Komponenten für grafische Oberflächen
- Geräte, die ihre Software-Repräsentation "mitbringen"
- Komponenten aus dem Office-Automation-Bereich (Textverarbeitung, Datenbank-Anbindung, Kalkulation, ...)
- ...
- ? **wo ist das Problem**
  - kein Standard-Komponenten-Modell
  - Komponenten-Modelle haben zu starke Einschränkungen
  - Builder-Tools nicht mächtig genug + nicht adaptierbar
- ? **Lösung**
  - offene Komponenten-Modelle
  - erweiterbare Werkzeuge (Builder-Tools), die Interoperabilität herstellen

## E.9 OSGi — Open Services Gateway Initiative

### 1 Überblick

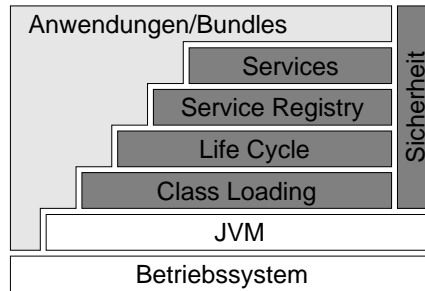
- OSGi Spezifikationen:  
standardisierte, komponenten-orientierte Ausführungsumgebung für vernetzte Dienste
- Basis: Java
- Ziel-Anwendungsbereiche
  - Mobiltelefone
  - Automobil (Infotainment-Bereich)
  - Telematik
  - Eingebettete Systeme
  - Haustechnik
  - PC- und Mainframe-Anwendungen
- Literatur: <http://www.osgi.org/>

## 2 Einsatzbeispiele

- NG Smart Phones
  - einheitliche Plattform für Mobiltelefon-Anwendungen
  - Anwendungen zentral administrierbar
- Shell HomeGenie
  - Softwareplattform zur Integration von Kameras, Sensoren, Thermostaten, Stromschaltern, etc. im Hausautomatisierungsbereich
- Eclipse
  - Java-Entwicklungsumgebung
  - ab Version 3.0: Installation und Update von Plug-ins zur Laufzeit
- BMW iDrive (5er und 7er, neuer 3er)
  - Basis: Betriebssystem VxWorks und JVM Jeode von Windriver
  - OSGi-Implementierung von Siemens VDO
  - Integration von Navigation, Mobiltelefon (Bluetooth), Internet-Anbindung und anderen Infotainment-Anwendungen



### 3 Architektur



- mehrere unabhängige Anwendungen in einer JVM
- Anwendung = Bundle
  - JAR-Datei mit zusätzlichen Header-Informationen (Manifest)
  - Schnittstellenkonventionen -> Komponentenmodell
  - Installation, Update, Start, Stop und Deinstallation über Netzwerk

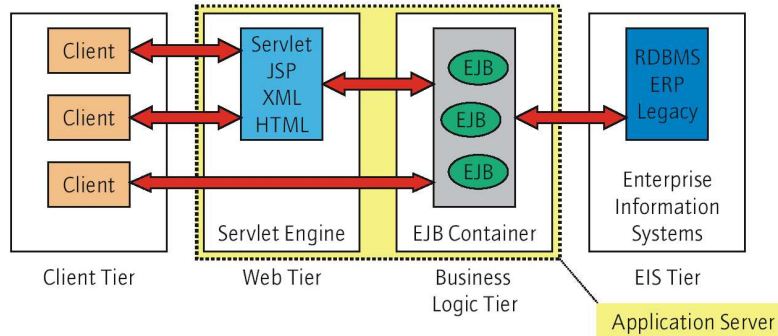
### 4 Services

- Class Sharing
  - Bundles können Pakete exportieren und von anderen Bundles importieren
  - vermeidet Redundanz von Code
  - ermöglicht Bereitstellung von Diensten in einer OSGi-Umgebung
  - Regeln zum Umgang mit unterschiedlichen Versionen
- Life Cycle Management
  - Dienste zur Installation von Bundles
- Service Registry
  - Nameservice
  - registrierte Objekte werden *Services* genannt: Schnittstelle und Menge von Eigenschaften (*Properties*)
  - automatische Abmeldung bei Bundle-Deinstallation + Information anderer Bundles

## E.10 Web-Services

### 1 EJB-Architektur (Details später in Kapitel EJB)

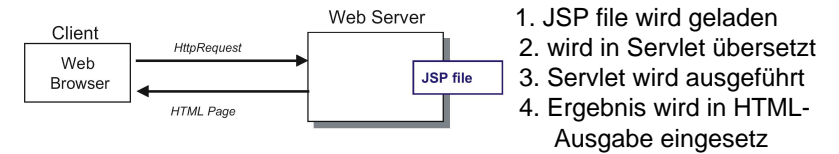
- Server-seitiges Komponentenmodell für *Multi-Tier* Anwendungen



- nicht mit JavaBeans verwechseln!

## 2 Java Server Pages

- HTML-Seiten mit integriertem Java-Code
- Schnittstelle zwischen Web-Seiten und verteilten objektorientierten Anwendungen



- Können eingesetzt werden
  - als Ersatz für traditionelle CGI-Skripten
  - als HTML-Schnittstelle zu objektorientierten Anwendungen (RMI, CORBA)

### 3 JSP Beispiel

```
<html>
<body>
<h2>Welcome to the Weather Site</h2> <hr>
Today's date is <%= new java.util.Date() %>
</body>
</html>
```

### 4 JSP Ziele

- Wiederverwendbarer Code wird in Komponenten oder JSP-Beans zur Verfügung gestellt
- Trennung von Web-Seiten-Design und Komponenten-Entwicklung

### 5 Servlets

- Grundproblem: Dynamik in Web-Seiten

- Lösungen:

- Client-seitig: JavaScript, Applets
- Server-seitig: CGI, ASP, LiveWire, PHP ...

- ➔ Servlets:

- “A servlet is a Java technology-based **web component**, managed by a container, that generates dynamic content. Like other Java-based components, servlets are platform-independent Java classes that are compiled to platform-neutral bytecode that can be loaded dynamically into and run by a Java-enabled web server. Containers, sometimes called servlet engines, are web server extensions that provide servlet functionality. Servlets interact with web clients via a **request/response** paradigm implemented by the servlet container.”

- Standard-API
- Plattformunabhängig Java
- Threads

- Ausführungsumgebung für Servlets: Servlet-Engine / Container

- Netzwerk-Service
  - (empfangen und zustellen von) Requests
  - (versenden von) Responds
  - Verwendung von HTTP (evtl. zusätzlich HTTPS, etc.)
- Steuerung des Lebenszyklus von Servlets / JSPs
  - laden, initialisieren, aufrufen, löschen
- Deployment
  - einrichten ("installieren") der Applikation