

Architekturstile und -muster

Ein Katalog von Konzepten

Burkhardt Renz

Fachbereich MNI
Technische Hochschule Mittelhessen

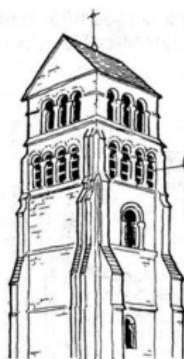
Wintersemester 2017/18

Übersicht

- Architekturstile
 - Was sind Architekturstile?
 - Arten von Komponenten und Konnektoren
 - Übersicht wichtiger Stile
- Beschreibung wichtiger Stile
- Ableitung einer Architektur
- Beispielhafte Architekturen

Was sind Architekturstile?

Clochers



Clocher roman
toit en bâtière
1 Abat-son



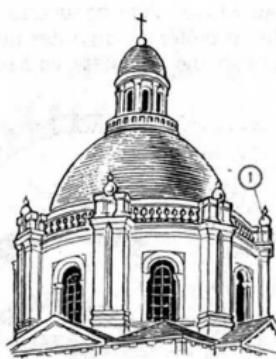
Flèche romane
polygone sur
tour carrée



Flèche gothique
aiguë et
ajourée



Clocher
Renaissance
1 Lanternon



Dôme classique
avec coupole à lanterne
1 Pot à feu

Was sind Architekturstile?

Bestandteile der Architektur

- Komponenten
- Konnektoren
- Topologie

Architekturstil

- Abstraktion von den Spezifika einer konkreten Architektur
- Leitlinie für Art der Komponenten und Konnektoren – und ihr Zusammenspiel
- Architektonische „Constraints“ bezüglich des Zusammenwirkens von Komponenten und Konnektoren
- Paradigma, Rahmen für Entwurfsentscheidungen

Auswirkungen von Architekturstilen

- prinzipielle Festlegung der Aufbaustruktur
- *zugleich*: Beschränkung der Art der Elemente und ihrer Verbindung
- *damit*: Festlegung der spezifischen Eignung und Qualitäten eines Systems

Übersicht

- Architekturstile
 - Was sind Architekturstile?
 - Arten von Komponenten und Konnektoren
 - Übersicht wichtiger Stile
- Beschreibung wichtiger Stile
- Ableitung einer Architektur
- Beispielhafte Architekturen

Arten von Komponenten

Berechner führt eine Art Berechnung aus; nur lokaler Zustand während der Berechnung; kein Gedächtnis; z.B. Funktion, Filter

Manager verwaltet Zustand, auch über mehrere Aufrufe hinweg; bietet Methoden, um den Zustand zu beeinflussen; z.B. Server, zustandsorientierte Objekte

Controller steuert die (zeitliche) Abfolge von Aktionen, reagiert entsprechend auf Ereignisse, z.B. Scheduler, Eventhandler

Speicher enthält persistente Daten in strukturierter Form; Zugriff verschiedener anderer Komponenten möglich; z.B. Datenbanken, Dateien

Kanal Ort, an dem Daten beobachtbar sind; Bestandteile des Datenflusses, z.B. Pipe, Funktionsparameter

Arten von Konnektoren

Funktionsaufruf Gemeinsamer Adressraum; Kontrolle geht an die aufgerufene Komponente und kehrt nach „Erlledigung“ zurück.

Datenfluss Prozesse interagieren durch einen Datenstrom z.B. Pipe; Komponenten sind unabhängig

Impliziter Aufruf Komponente wird durch ein Event aktiviert; sie weiß nicht, wer den Dienst benötigt

Nachrichtenaustausch Unabhängige Prozesse interagieren durch Protokoll; kann synchron sein z.B. Request/Response oder asynchron z.B. Message Queue

Gemeinsame Daten Komponenten arbeiten auf denselben Daten; Steuerung durch Datenzustand; Synchronisation; Beispiel Repository

Übersicht

- Architekturstile
 - Was sind Architekturstile?
 - Arten von Komponenten und Konnektoren
 - Übersicht wichtiger Stile
- Beschreibung wichtiger Stile
- Ableitung einer Architektur
- Beispielhafte Architekturen

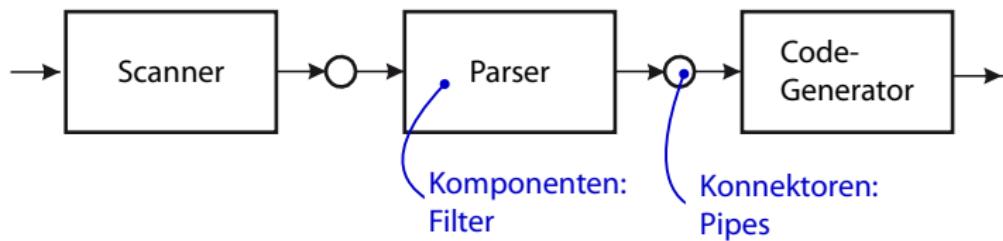
Übersicht wichtiger Architekturstile

- Datenflusssysteme
 - Batch sequenziell
 - Pipes und Filter
 - Prozess-Kontrollschleifen
- Kontrollfluss
 - Funktionale Zerlegung
 - Objektorientierte Organisation
 - Schichten
- (Unabhängige) Komponenten
 - Kommunizierende Prozesse
 - Ereignisgesteuerte Systeme
 - Plugin-Architektur
- Datenzentrierte Systeme
 - Repositories
 - Blackboard
- Virtuelle Maschinen
 - Interpreter
 - Regelbasierte Systeme

Übersicht

- Architekturstile
- Beschreibung wichtiger Stile
 - „Klassische“ Stile
- Ableitung einer Architektur
- Beispielhafte Architekturen

Pipes und Filter



Pipes und Filter – Charakteristik

Komponenten Filter transformieren Eingabeströme in Ausgabeströme

Konnektoren Pipes bewegen und puffern Daten zwischen den Filtern

Struktur Datenfluss zwischen unabhängigen Komponenten; Schnittstelle durch Input- und Output-Formate definiert

Systemmodell Kontinuierlicher Datenfluss zwischen Komponenten, die inkrementell die Datenströme transformieren

Pipes und Filter – Diskussion

Varianten

- Lineare Folge von Filtern: Pipeline
- Spezielle Pipes: Formatprüfung
- Push-Methode z.B. Unix-Pipes
- Pull-Methode z.B. yacc ruft lex
- Filter teilen gemeinsame Daten z.B. Symboltabelle eines Compilers

Pipes und Filter – Vor- und Nachteile

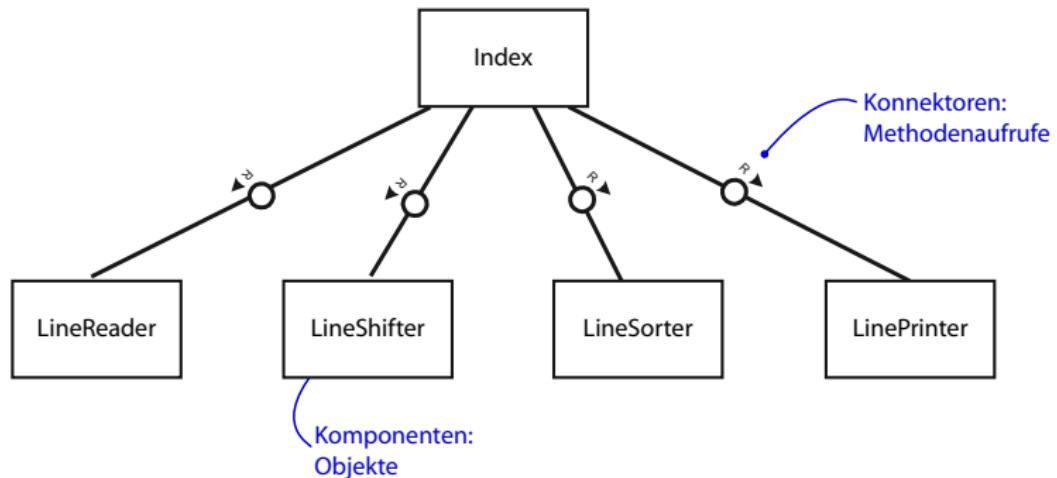
Vorteile

- Einfach zu kombinieren und wiederzuverwenden
- Einfach auszutauschen
- Effizient durch Parallelität
- Leicht zu analysieren und zu testen

Nachteile

- Fehlerbehandlung
- Zugriff auf gemeinsame Daten?
- Kopplung durch Datenformate, Kosten für Datentransfer
- Skalierbarkeit
- kaum geeignet für interaktive Anwendungen

Objektorientierte Organisation



Objektorientierte Organisation – Charakteristik

- Komponenten** Objekte, die Daten kapseln;
Objekte, die als Manager und Controller agieren
- Konnektoren** Methodenaufrufe
- Struktur** Kollaborierende Objekte
- Systemmodell** Komponenten verwalten „ihre“ Daten,
bündeln Verantwortlichkeit für Daten und
Funktionalität

Objektorientierte Organisation – Diskussion

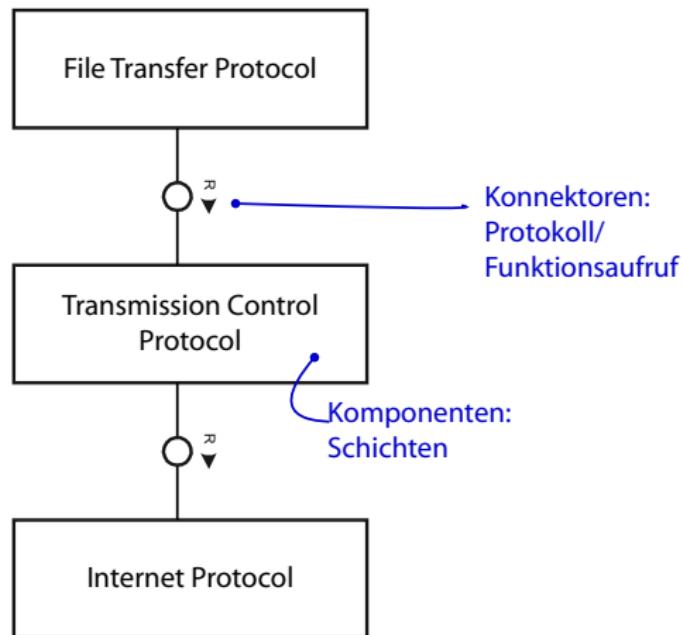
Vorteile

- Trennung Implementierung – Verwendung erleichtert Austauschbarkeit
- Wiederverwendbarkeit, etwa Klassenbibliotheken
- Wiederverwendbarkeit in neuem Kontext durch dynamischen Polymorphismus („Alter Code ruft neuen Code“)

Nachteile

- Verwender müssen Objektidentität kennen
- Zerfaserung von Funktionalität auf viele Objekte
- Bruch der Kapselung bei Implementierungsvererbung

Hierarchische Schichten



Hierarchische Schichten – Charakteristik

Komponenten In Schichten organisierte Komponenten, die verschiedenen Typs sein können – virtuelle Maschinen

Konnektoren Funktionsaufrufe zu darunterliegenden Schichten – mögliche Benutzung

Struktur Hierarchie

Systemmodell Jede Schicht stellt eine Abstraktion bereit, wodurch der Verwender einer Schicht getrennt ist von Details der Implementierung.

Der Verwender nutzt den Dienst einer Schicht (einer virtuellen Maschine).

Hierarchische Schichten – Diskussion

Sprechweisen

- Schichten als Separierung von Verantwortlichkeit:
insbesondere bei interaktiven Systemen z.B. Trennung
Präsentation – Anwendungslogik – Persistenzmechanismus
- Schichten als Konzept der Modellierung: insbesondere in
UML-basierte Konzepte z.B. Trennung
`«boundary» – «controller» – «entity»`
- Schichten als physische Verteilung:
2-tier, 3-tier und n-tier Architekturen
- Hierarchische Schichten (im engeren Sinne):
etwa Hardware Abstraction Layers u.ä. – Strukturierung des
Codes und der Aufrufhierarchie

Hierarchische Schichten – Diskussion

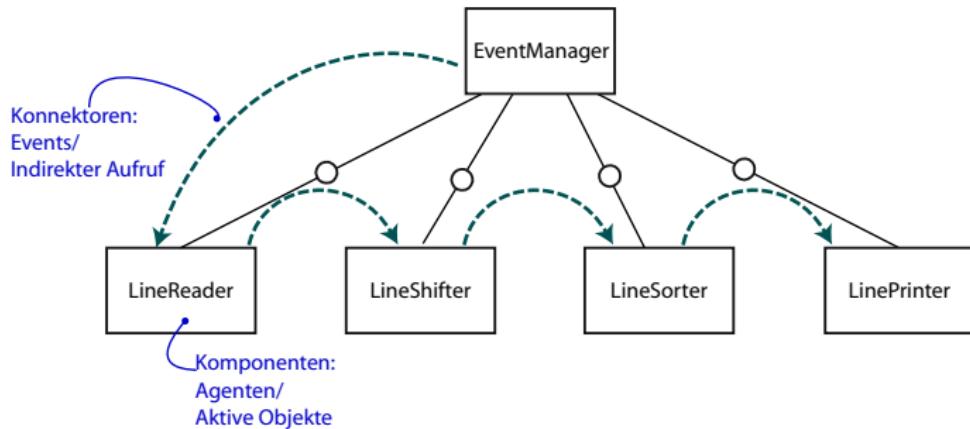
Vorteile

- Abstraktion
- Lokalisierung von Entscheidungen in Schicht – leichte Änderbarkeit
- Austauschbarkeit von Schichten

Nachteile

- Geeignete Abstraktionen schwierig zu finden
- Viele Indirektionen – Folge: *layer bridging*

Ereignisgesteuerte Systeme



Ereignisgesteuerte Systeme – Charakteristik

Komponenten Prozesse, Threads, aktive Objekte, die Events erzeugen und auf Events reagieren.

Konnektoren Die Komponenten registrieren sich für die Benachrichtigung, wenn bestimmte Events auftreten. Das Event führt so zum impliziten Aufruf der Komponente

Struktur Unabhängige, nicht zentral gesteuerte Komponenten
- benötigt ein Framework für den Eventmechanismus

Systemmodell Die Komponenten sind nicht durch direkte Referenzen gekoppelt, sondern reagieren auf Events.
Häufig sind die Komponenten nebenläufig.
Reihenfolge der Aufrufe nicht deterministisch.

Ereignisgesteuerte Systeme – Diskussion

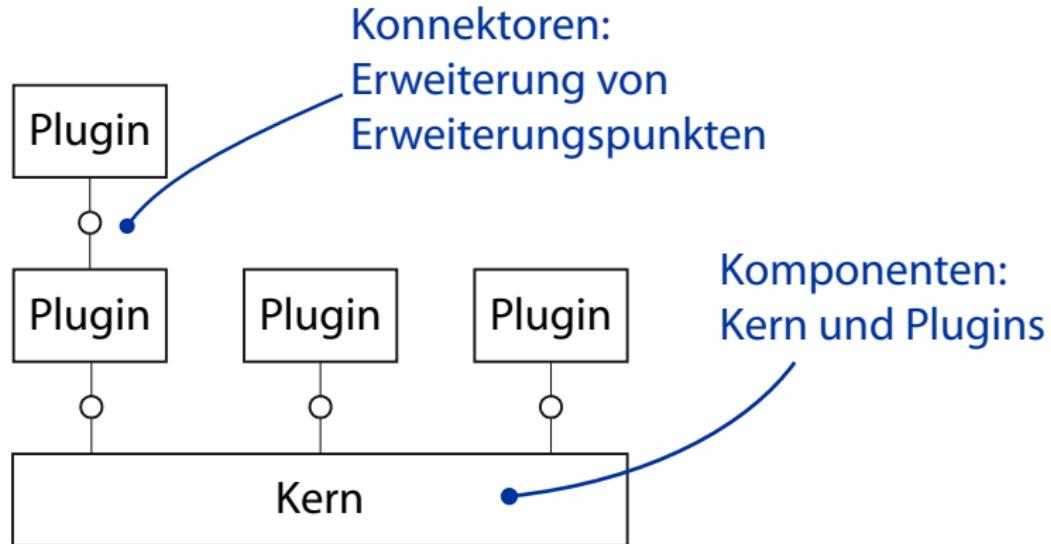
Vorteile

- Austauschbarkeit der Komponenten zur Laufzeit
- Evolution des Systems durch Veränderung und Erweiterung einzelner Komponenten, ohne andere zu tangieren

Nachteile

- Kontrollfluss nur schwer zu verfolgen
- Komponenten müssen Infrastruktur und Protokolle gemeinsam haben
- Fehlerbehandlung in Bezug auf die Interaktion der Komponenten
- Zugriff der Komponenten auf gemeinsam verwendete Daten?

Plugin-Architektur



Plugin-Architektur – Charakteristik

Komponenten der Kern,
die Plugins

Konnektoren Erweiterung von Erweiterungspunkten des Kerns oder
von anderen Plugins

Struktur Baumartige Struktur von sich erweiternden Plugins

Systemmodell Vertrag zwischen Angebot zur Erweiterung und
Nutzung des Angebots.

Plugin-Architektur – Diskussion

Beispiele

- OSGi Java-Bundles und Services
- Eclipse und Eclipse Rich Client Platform

Arten

- Spezieller Vertrag zwischen Wirt und Plugin, Wirt sucht Plugin – z.B. .NET Assemblies
- Plugin-Registry wie bei Eclipse
- Zeitpunkt der Plugin-Aktivierung: Startzeitpunkt, zur Laufzeit

Plugin-Architektur – Diskussion

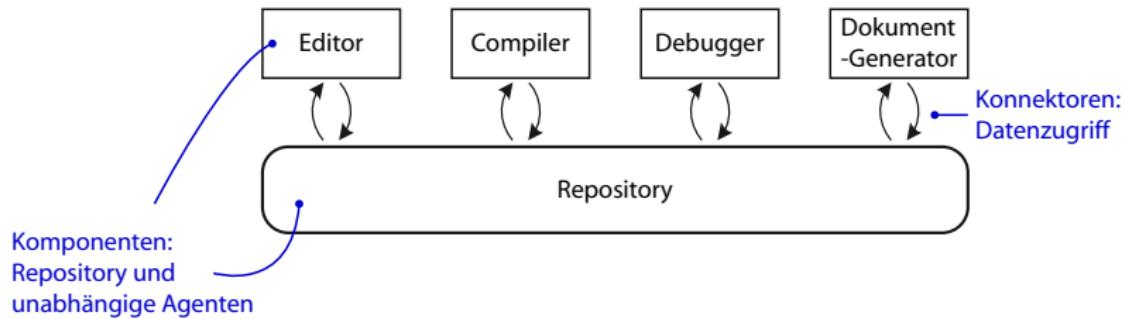
Vorteile

- Erweiterung – ohne Modifikation des vorhandenen Systems
- Verbesserte Wartbarkeit durch Austausch von Plugins
- Wiederverwendung von Plugins (auch aus anderen Anwendungen)

Nachteile

- Starke Kopplung zwischen Komponenten kann leicht entstehen
- Fehlerhaftes Plugin kann Stabilität des Systems gefährden
- Subtile Abhängigkeiten zwischen Plugins können entstehen

Repository



Repository – Charakteristik

Komponenten Eine Speicherkomponente und viele Agenten, die auf diese Daten zugreifen

Konnektoren Der Zugriff auf die Speicherkomponente verbindet die verschiedenen Agenten

Struktur Die Steuerung ergibt sich durch die Entscheidungen der Agenten, die sie auf Grund der Datenlage in der Speicherkomponente treffen

Systemmodell Zentralisierte Daten sind die Basis der autonomen Entscheidung von Agenten, die diese Daten lesen und verändern. Datenintegration als wichtigstes Ziel

Repository – Diskussion

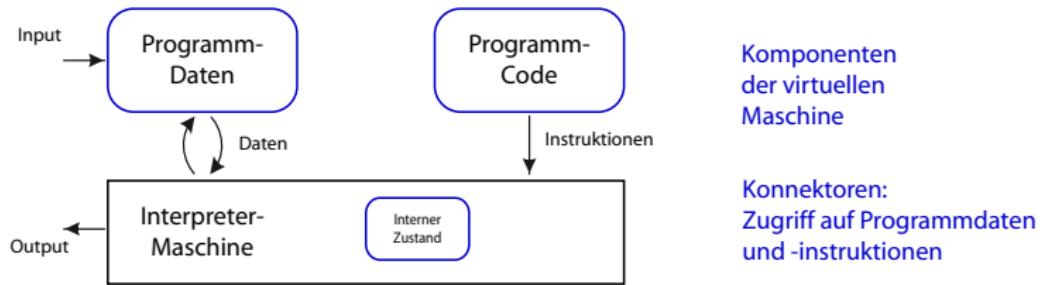
Vorteile

- Austauschbarkeit von Agenten
- Kontrolle der Daten durch zentrale Datenhaltung

Nachteile

- Repository kann zum Engpass werden
- Fehlerbehandlung durch Unabhängigkeit der Agenten erschwert
- Kontrollfluss evtl. schwer zu überblicken

Interpreter



Interpreter – Charakteristik

Komponenten Komponenten sind:

die Maschine,
der Programmcode (beinhaltet den Kontrollfluss) und
der Speicher der Maschine (enthält die Daten)

Konnektoren Interner Ablauf des Interpreters

Struktur Metalevel: hohe Dynamik durch Steuerung des Verhaltens zur Laufzeit

Systemmodell Steuerung des Interpreters durch eine *Sprache*

Interpreter – Diskussion

Beispiele

- Java Virtual Machine
- SQL-Interpreter in Datenbanksystemen
- Dynamische interaktive Oberflächen erzeugt aus Beschreibungen
- ABAP in SAP R/3 - interpretierte Sprache mit spezialisierten Konstrukten im Zugriff auf Repository

Arten

- Datensteuerung – einfach Formen von Interpretern durch Konfiguration zur Laufzeit
- Domänenspezifische Sprachen
- Alternative: Codegenerierung

Interpreter – Diskussion

Vorteile

- Sprache - gut verstanden, gut wartbar, gut erweiterbar
- Sprache - mächtig
- Testbarkeit wegen Einsatz einer Sprache
- Domänenspezifika können direkt formuliert werden

Nachteile

- Leistungsfähigkeit – kompensierbar durch Objektserialisierung, Zwischencode, JIT Compiler
- Sicherheitsprobleme durch Interpreter häufig: SQL Injection, Windows Scripting
- aufwändig, einmal konstruierte Sprache nur schwer später zu revidieren

Literatur



Mary Shaw, David Garlan

Software Architecture: Perspectives on an Emerging Discipline
Upper Saddle River, NJ: Prentice-Hall, 1996.



Paris Avgeriou, Uwe Zdun

Architectural Patterns Revisited – A Pattern Language
EuroPLoP 2005.

Übersicht

- Architekturstile
- Beschreibung wichtiger Stile
- Ableitung einer Architektur
 - ... am Beispiel der Architektur des Webs
- Beispielhafte Architekturen

Die Architektur des Webs

Komponenten Server mit Ressourcen (z.B. Apache), Gateways, Proxies, Klienten wie Browser, Search Bots

Konnektoren APIs zum Zugriff auf http-Server wie libwww, Serverseitige APIs wie Apache API, Servlet API ...

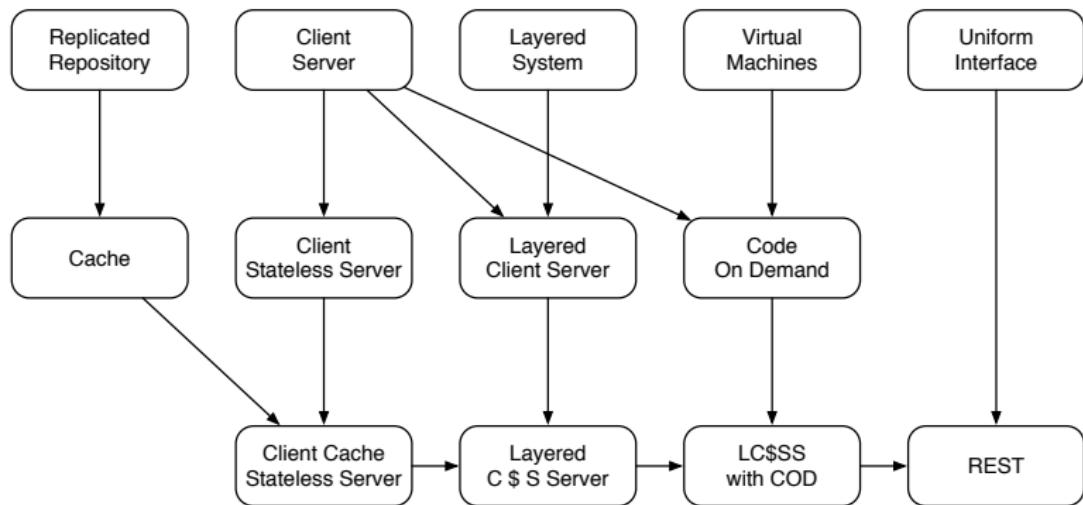
Daten Ressourcen, Identifizierer (URL), Repräsentation der Ressource (z.B. als HTML, XML, JPG etc), Metadaten, Steuerdaten.

Topologie Viele Klienten, viele Server, Intermediäre/Proxies auf beiden Seiten.

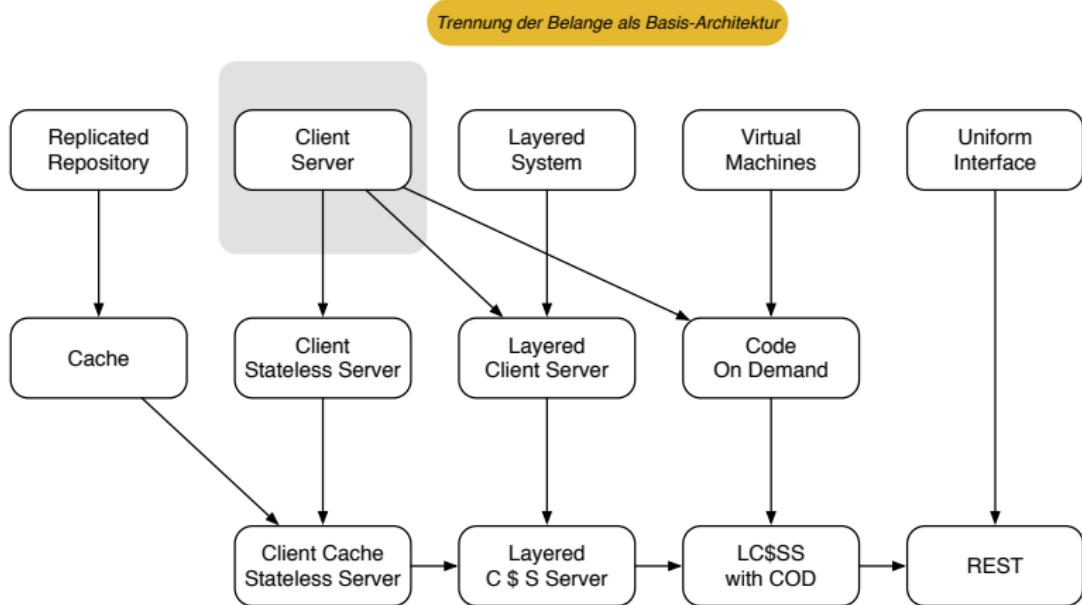
REST: Representational State Transfer

- Die Schlüsselabstraktion ist die *Ressource* – eine identifizierbare Informationseinheit.
- Die Repräsentation einer Ressource ist eine *Bytefolge* zusammen mit *Metadaten* zu ihrer Beschreibung; sie hat einen bestimmten Typ: etwa HTML, JPG o.ä..
- Alle Interaktionen sind kontextfrei.
- Komponenten verwenden eine kleine Menge wohldefinierter Methoden: get, put, post, delete ...
- Operationen sollen möglichst idempotent sein.
- Intermediäre und Caches sind erwünscht.

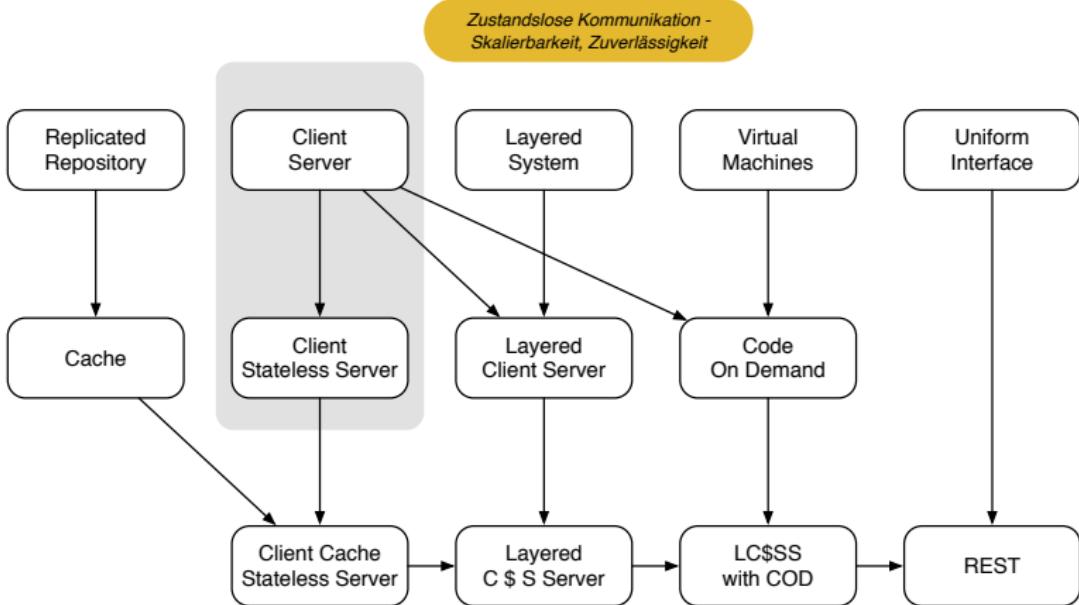
Ableitung von REST



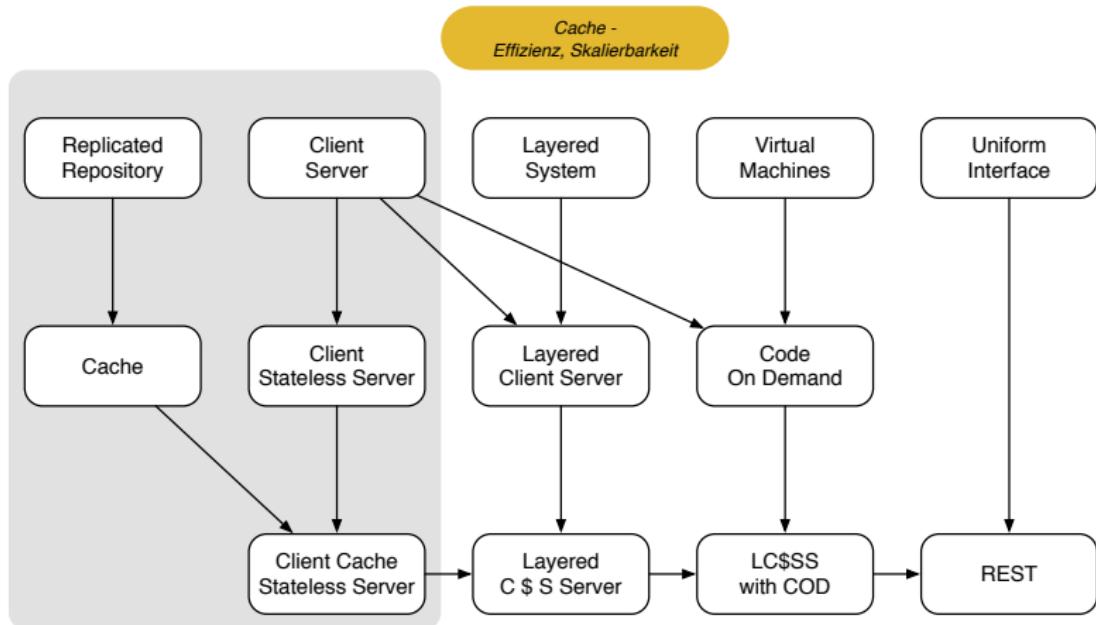
Ableitung von REST



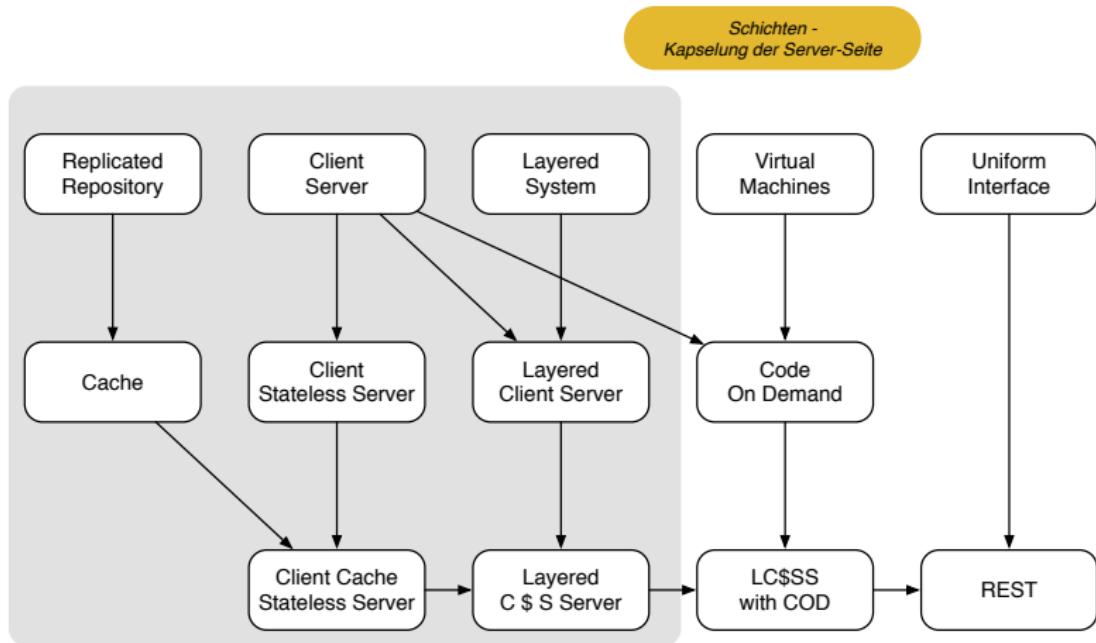
Ableitung von REST



Ableitung von REST

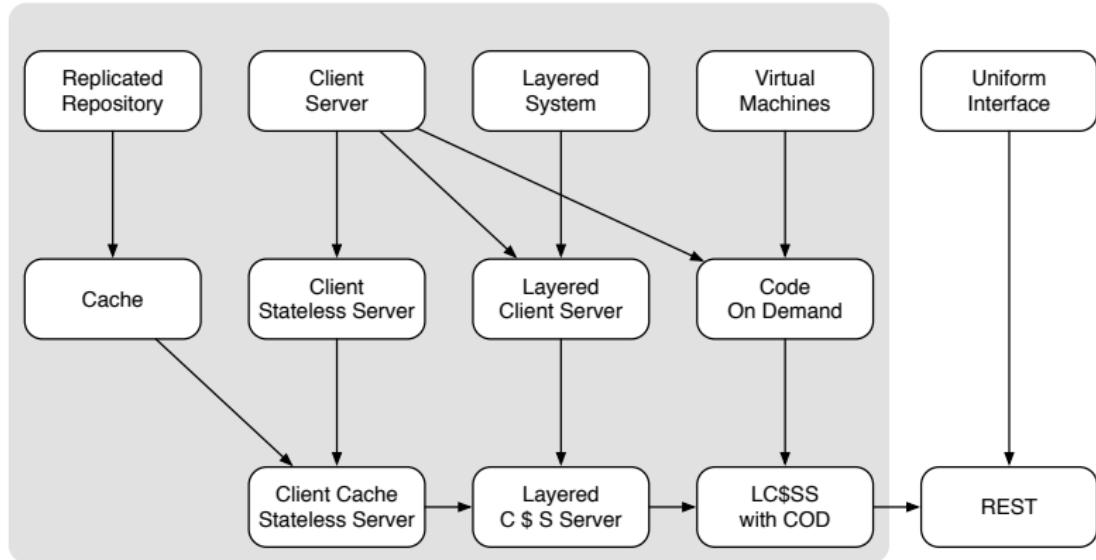


Ableitung von REST



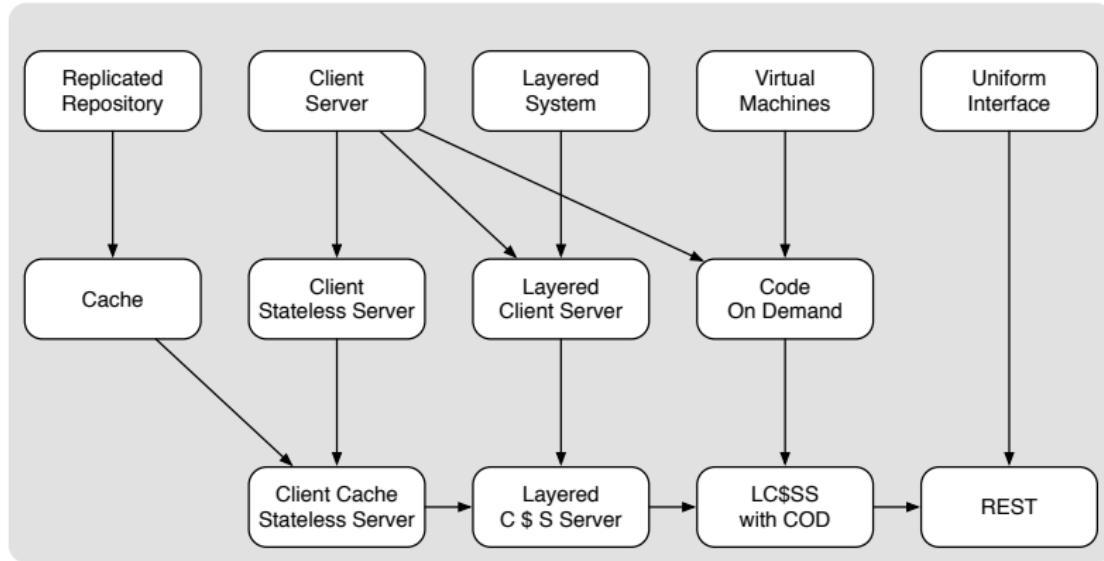
Ableitung von REST

Code on Demand -
Dynamik, Erweiterbarkeit



Ableitung von REST

Generische Schnittstelle -
Universalität



Literatur



Roy Thomas Fielding

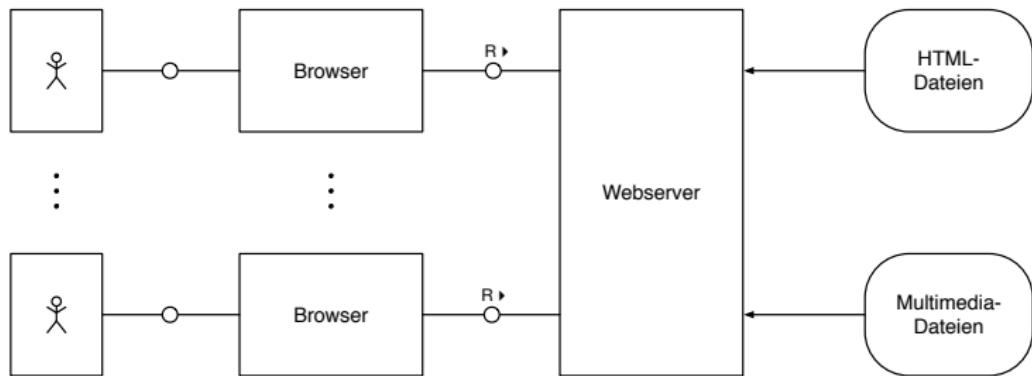
Architectural Styles and the Design of Network-based Software Architectures

PhD Thesis Irvine, CA; 2000. <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

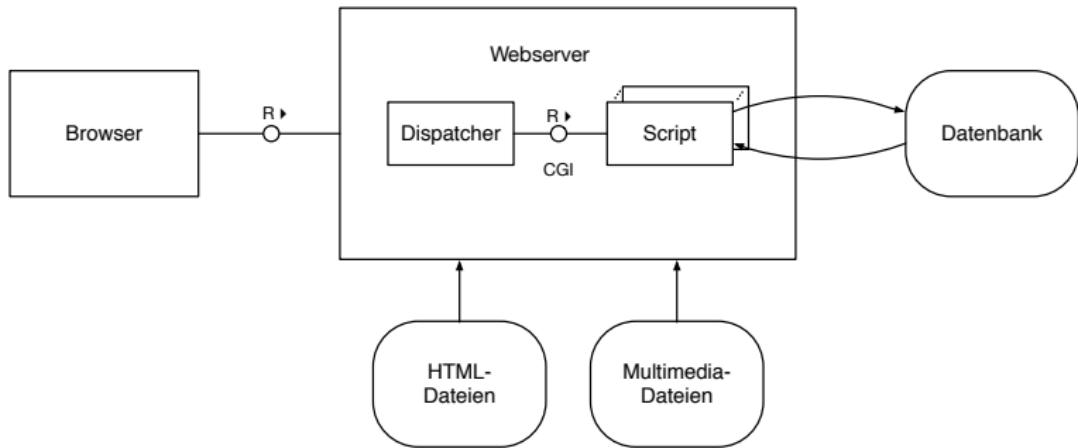
Übersicht

- Architekturstile
- Beschreibung wichtiger Stile
- Ableitung einer Architektur
- Beispielhafte Architekturen
 - Entwicklung der Architektur von Web-Anwendungen
 - Exkurs zu MVC (Model – View – Controller)
 - Container – Interceptoren: Applikationsserver
 - Metalevel-Architektur und domänen spezifische Sprachen
 - MapReduce

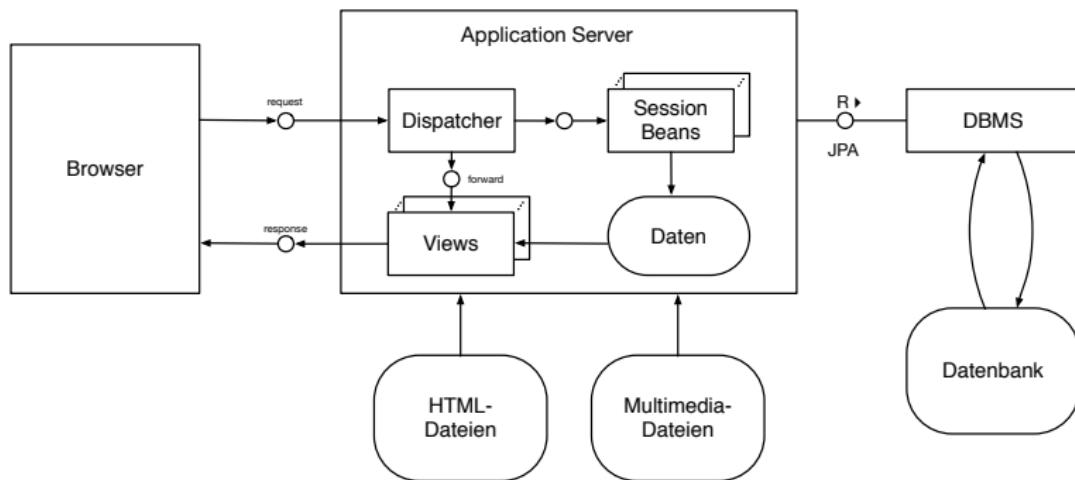
Statische, verlinkte Webseiten



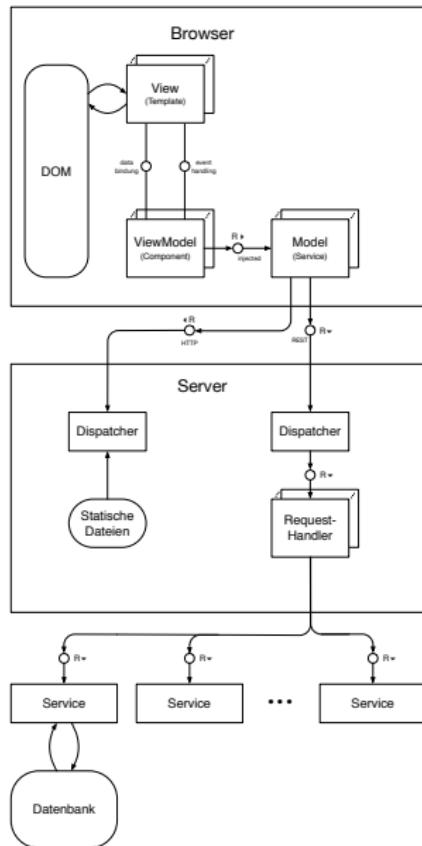
Verlinkte Webseiten mit Scripting (CGI)



Geschäftsanwendungen im Web



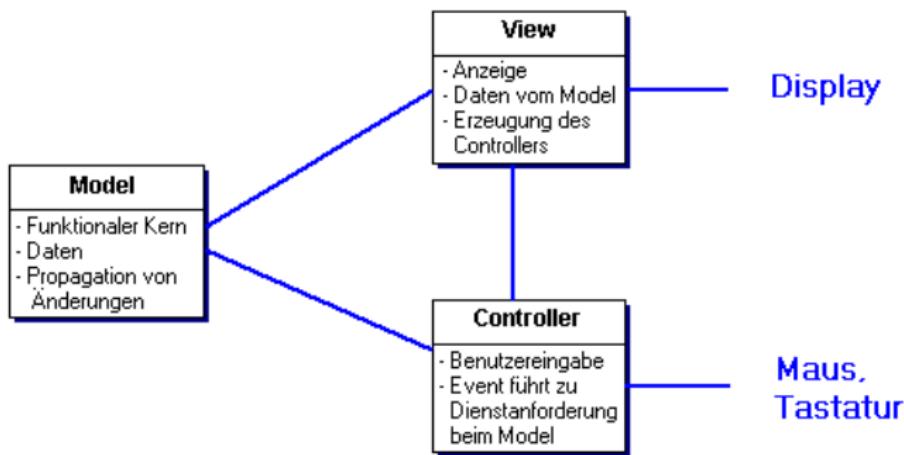
Single-Page-Anwendungen im Web



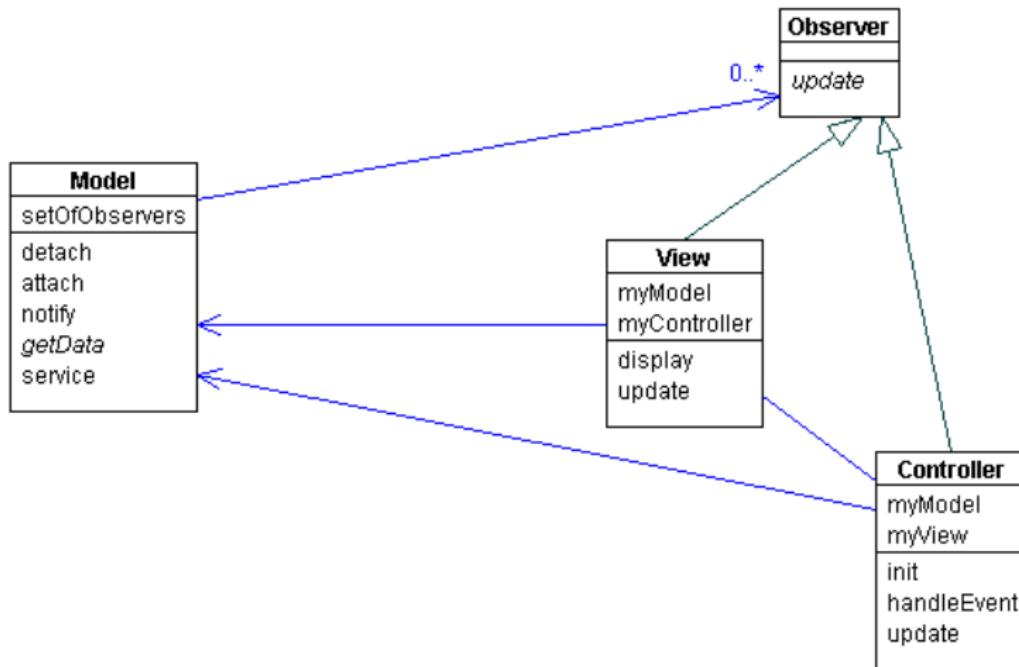
Übersicht

- Architekturstile
- Beschreibung wichtiger Stile
- Ableitung einer Architektur
- Beispielhafte Architekturen
 - Entwicklung der Architektur von Web-Anwendungen
 - **Exkurs zu MVC (Model – View – Controller)**
 - Container – Interceptoren: Applikationsserver
 - Metalevel-Architektur und domänen spezifische Sprachen
 - MapReduce

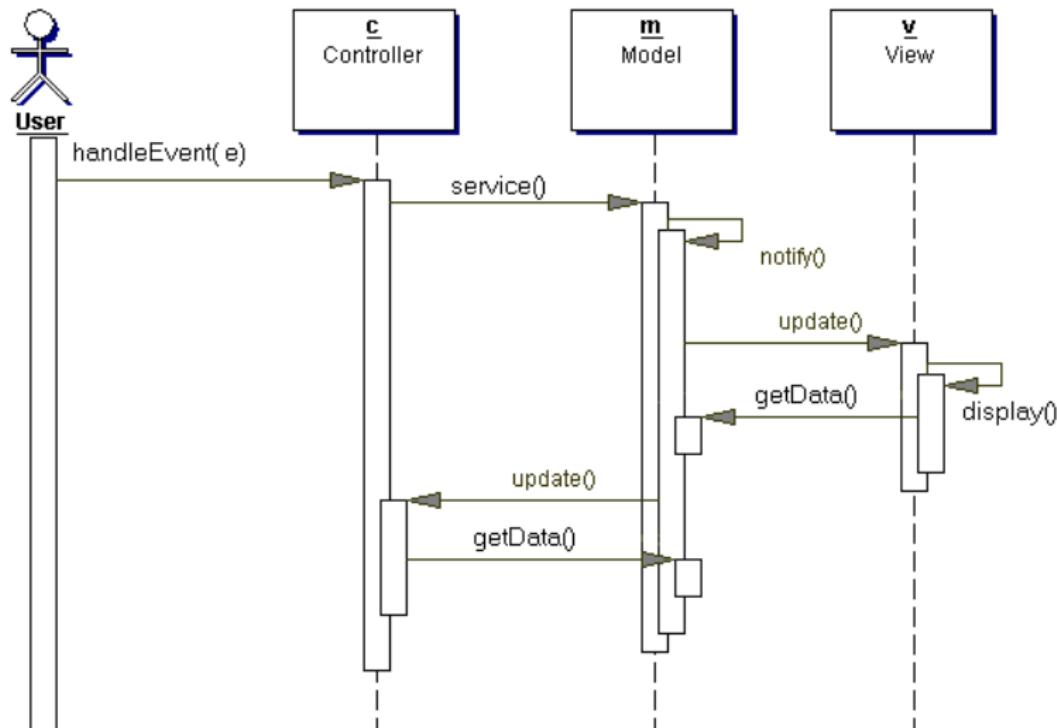
„Klassisches“ MVC – Prinzip



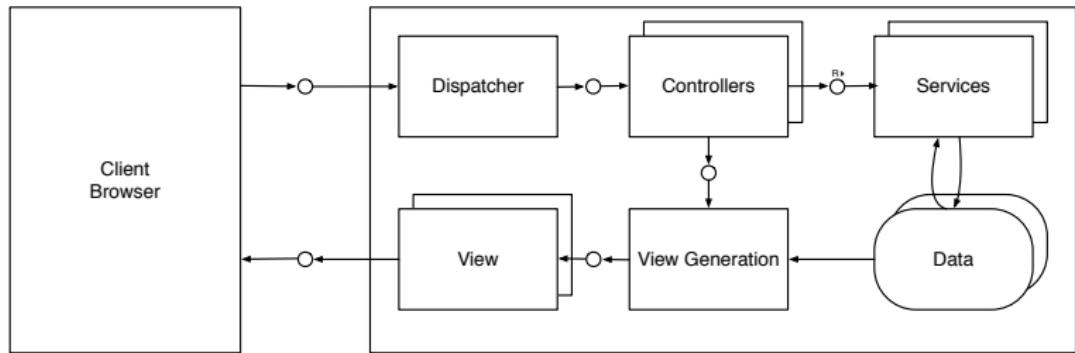
„Klassisches“ MVC – Codestruktur



„Klassisches“ MVC – Interaktion



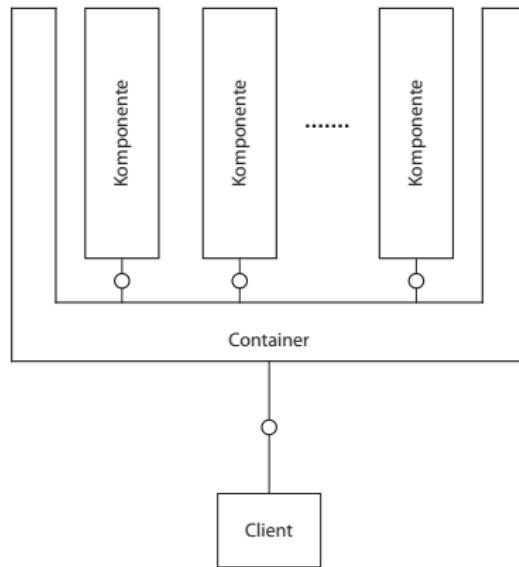
MVC im Web – Konzept



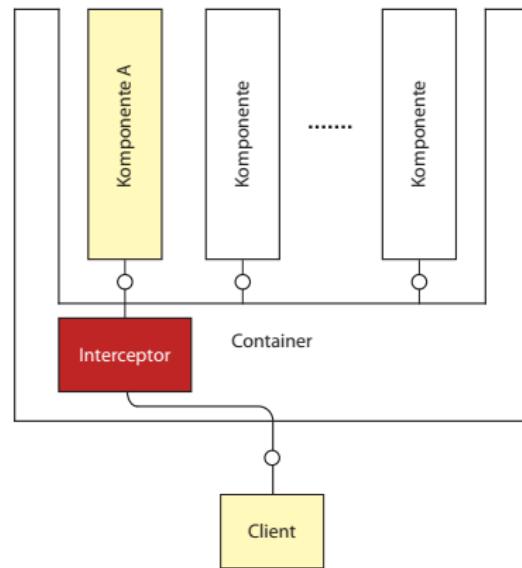
Übersicht

- Architekturstile
- Beschreibung wichtiger Stile
- Ableitung einer Architektur
- Beispielhafte Architekturen
 - Entwicklung der Architektur von Web-Anwendungen
 - Exkurs zu MVC (Model – View – Controller)
 - **Container – Interceptoren: Applikationsserver**
 - Metalevel-Architektur und domänen spezifische Sprachen
 - MapReduce

Container – Konzept



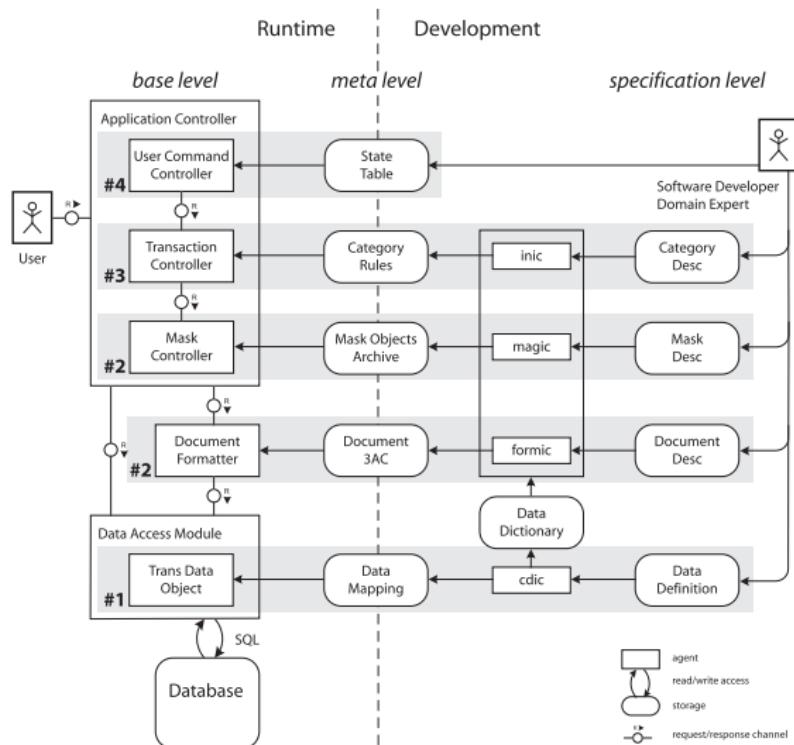
Interceptor – Konzept



Übersicht

- Architekturstile
- Beschreibung wichtiger Stile
- Ableitung einer Architektur
- Beispielhafte Architekturen
 - Entwicklung der Architektur von Web-Anwendungen
 - Exkurs zu MVC (Model – View – Controller)
 - Container – Interceptoren: Applikationsserver
 - Metalevel-Architektur und domänen spezifische Sprachen
 - MapReduce

Metalevel-Architektur – Beispiel



Übersicht

- Architekturstile
- Beschreibung wichtiger Stile
- Ableitung einer Architektur
- **Beispielhafte Architekturen**
 - Entwicklung der Architektur von Web-Anwendungen
 - Exkurs zu MVC (Model – View – Controller)
 - Container – Interceptoren: Applikationsserver
 - Metalevel-Architektur und domänen spezifische Sprachen
 - **MapReduce**

map und reduce in funktionalen Sprachen

```
(map fn coll)
```

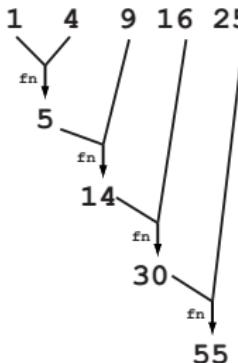
```
(map square [1 2 3 4 5])
```

1	2	3	4	5	
fn↓	fn↓	fn↓	fn↓	fn↓	unäre Map-Funktion
1	4	9	16	25	

map und reduce in funktionalen Sprachen

(**reduce** fn coll)

(**reduce** + [1 4 9 16 25])



binäre Reduce-Funktion

map und reduce in funktionalen Sprachen

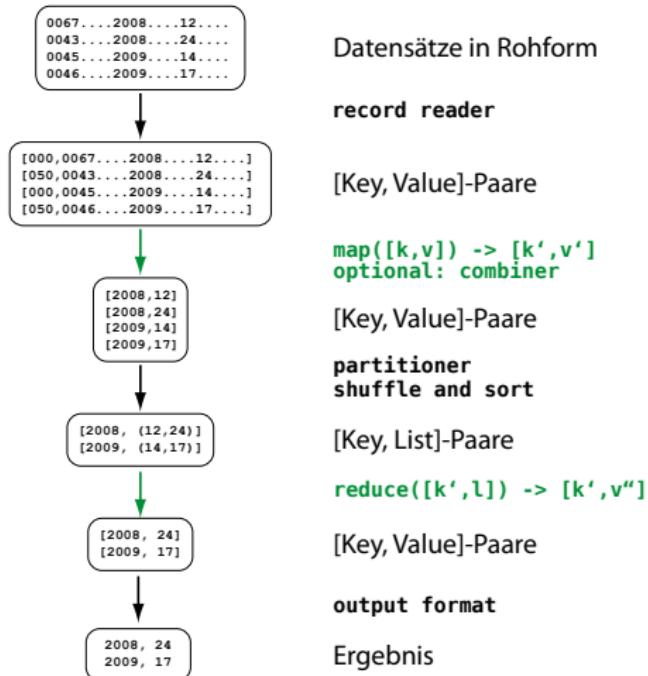
(mr map-fn reduce-fn coll)

unäre Map-Funktion

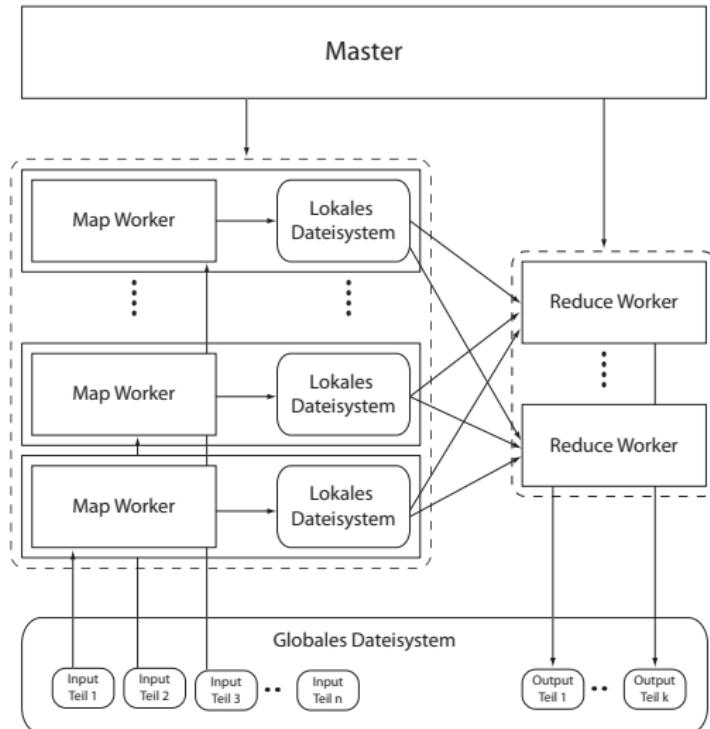
binäre Reduce-Funktion

(mr square + [1 2 3 4 5])

Grundidee von MapReduce



Architektur von MapReduce



Literatur

-  Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal
Pattern-orientierte Software-Architektur: Ein Pattern-System
Bonn: Addison-Wesley, 1998.
-  Douglas C. Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann
Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects
J. Wiley & Sons, 2000.
-  Martin Fowler
Patterns of Enterprise Application Architecture
Addison-Wesley, 2003.

Literatur

-  **Jeffrey Dean, Sanjay Ghemawat**
MapReduce: Simplified Data Processing on Large Clusters
OSDI'04: Sixth Symposium on Operating System Design and Implementation, 2004.
-  **Tom White**
Hadoop: The Definitive Guide
O'Reilly, 2012.
-  **Donald Miner, Adam Shook**
MapReduce Design Patterns
O'Reilly, 2013.