Model-View-Controller-Service-Paradigma

Dieser Artikel erfüllt die GlossarWiki-Qualitätsanforderungen **nur teilweise**:

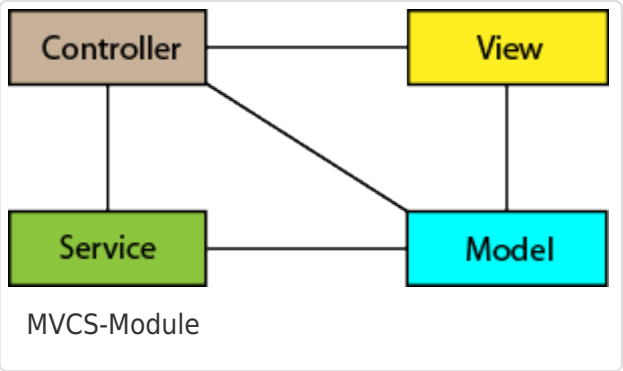
Korrektheit: 4 (großteils überprüft)	**Umfang**: 3 (einige wichtige Fakten fehlen)	**Quellenangaben**: 4 (fast vollständig vorhanden)	**Quellenarten**: 4 (sehr gut)	**Konformität**: 4 (sehr gut)

Diese Bewertungen beziehen sich auf alle im nachfolgenden Menü genannten Artikel gleichermaßen.

MVC-Paradigma:	Model (Data) | View | Controller	MVC-Pattern:	Singletons | Dependency Injection | Observers
MVCS-Paradigma:	Service	MVCS-Pattern:	Singletons | Dependency Injection | Observers
LDVCS-Paradigma:	Logic	VCLSD-Pattern:	Singletons | Dependency Injection | Observers

Inhaltsverzeichnis

1 Definition



MVCS-Module

Als Model-View-Controller-Service-Paradigma (engl. Model-view-controller-service paradigm) oder -Architektur (engl. architecture), kurz MVCS-Paradigma, MVCS-Architektur oder auch nur MVCS, bezeichnet man ein Architekturmuster, bei der eine Anwendungs-Komponente in vier eigenständige Module unterteilt wird: Model (Modell), View (Darstellung, Präsentation), Controller (Steuerung) und Service (Service, Zugriff auf externe Daten).

1.1 Model (Modell)

Ein MVCS-Modell (engl. MVCS model) einer MVCS-Anwendung dient zur Speicherung bestimmter Daten, d. h. zur Speicherung von Teilen des aktuellen Zustands der Anwendung.

Ein MVCS-Modell kann weitere Aufgaben übernehmen:

anderen Modulen Zugriff auf die Zustandsdaten gewähren
andere Module über Änderungen informieren (meist mittels des Observer-Patterns)
Umsetzung der Komponentenlogik

1.2 View (Darstellung, Präsentation)

MVCS-Views (engl. MVCS views) sind die grafischen, akustischen, haptischen und olfaktorischen Schittstellen einer MVCS-Anwendung. Sie „visualisieren“ Daten, die in Modellen der Anwendung enthalten sind.

Eine MVCS-View kann weitere Aufgaben übernehmen:

bei Daten-Änderungen in den zugehörigen Modellen der Anwendung die View-Repräsentation dieser Daten automatisch anpassen
Benutzeraktionen, die über grafische Eingabeelemente – wie Textfelder oder Buttons – erfolgen, an einen Controller weiterleiten
Visualisierung von Status- und Fehlermeldungen von Controllern

1.3 Controller (Steuerung)

MVCS-Controller (engl. MVCS controllers) dienen zur Steuerung einer MVCS-Anwendung. Dazu nimmt ein Controller Eingaben aus verschiedensten Quellen entgegen (z. B. Sensor-Daten oder Daten, die ein Benutzer über eine beliebige Benutzer-Schnittstelle wie eine Tastatur oder eine Maus eingibt) und leitet diese bereinigt und normalisiert an Modelle und/oder Servicemodule weiter.

Ein MVCS-Controller kann weitere Aufgaben übernehmen:

Verarbeitung von Systemsignalen, wie z. B. einer Systemuhr (z. B. „Spielzeit ist abgelaufen“)
Umsetzung der Komponentenlogik

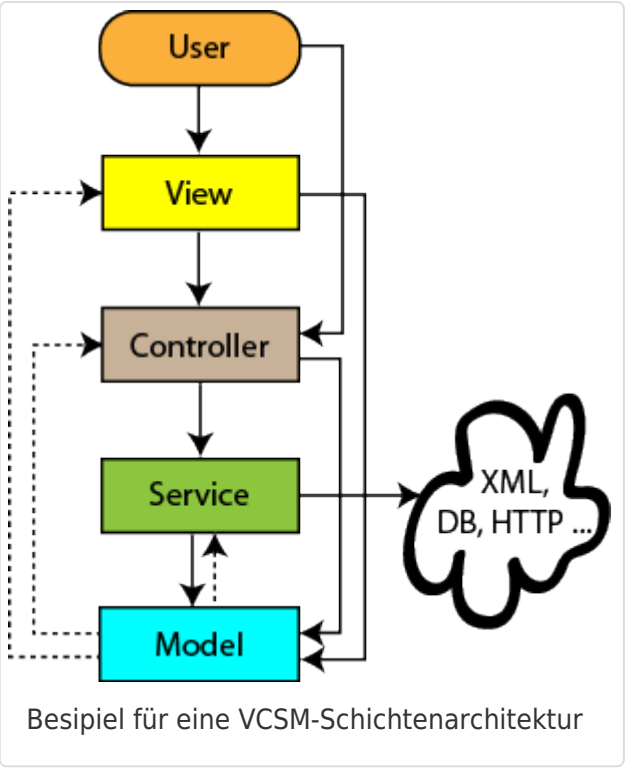
1.4 Service

Ein Service (engl. service) dient einer MVCS-Anwendung zur Kommunikation mit der Außenwelt, d. h. mit Dienste-Anbietern wie Web-Servern, Datenbanksystemen oder auch Dateisystemen. Die Kommunikation kann in beide Richtungen erfolgen: Services können sowohl Daten aus Modellen auslesen und in externe Repositories schreiben, als auch Daten aus externen Repositories lesen und in Modelle einfügen.

1.5 Data Access Object

> TO BE DONE

2 MVCS-Paradigma: Varianten (Definitionen nach Kowarschick (MMProg))



Besipiel für eine VCSM-Schichtenarchitektur

2.1 VCSM-Paradigma

Ein MVCS-Paradigma wird **VCSM-Paradigma** genannt, wenn die vier zugehörigen Module folgende Schichtenarchitektur bilden: *View*, *Controller*, *Service*, *Model*.

Die vier Module einer MVCS-Komponente sind gemäß dem Schichtenparadigma in Ebenen angeordnet. Die höheren Ebenen können auf tiefergelegene Ebenen zugreifen, aber nicht umgekehrt.

Die unserste Ebene enthält das Modell. Das zugehörige Modul weiß nichts von den über ihr liegenden Module und kann mit diesen nur mit indirekt – z. B. durch Antworten auf Nachrichten oder mit Hilfe des Observer-Patterns – kommunizieren.

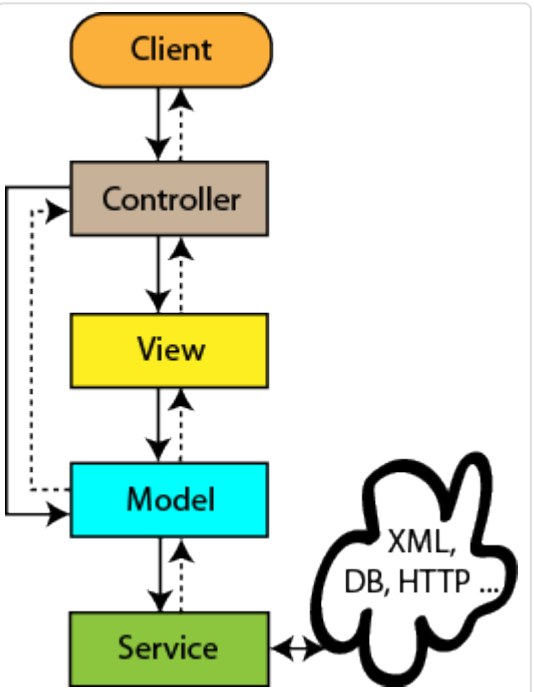
Ein Servicemodul kennt nur die darunterliegende Modelle und kann nur auf diese zugreifen (sowie i. Allg. auf externe Datenquellen, sofern es nicht nur Filterfunktionen wahrnimmt).

Ein Controllermodul kann entweder direkt auf ein Modellmodul zugreifen oder indirekt mit Hilfe eines Servicemoduls (zum Beispiel um Daten aus einer externen Quelle zu laden, oder um zu überprüfen, ob die entsprechenden Zugriffsrechte für eine Datenmanipulation überhaut gegeben sind).

Man beachte, dass in diesem Paradigma die Logik der Anwendung in den Controllern und nicht in den Modellen realisiert werden sollte, da die Modelle keinen direkten Zugriff auf andere Module haben.

Eine View kommuniziert i. Allg. direkt nur mit Controllermodulen, um diesen Benutzeraktionen, die über die View erfolgen, mitzuteilen. Ein direkter Zugriff auf ein Datenmodul ist nur dann notwendig, wenn die Nachrichten, die von den Datenmodulen verschickt werden, nicht alle relevanten Informationen enthalten. Zugriffe auf Servicemodule sind nicht vorgesehen.

Der Benutzer stellt das „oberste Modul“ dar. Er kommuniziert nur mit View- und Controllermodulen.



Besipiel für eine CVMS-Schichtenarchitektur

2.2 CVMS-Paradigma

Ein MVCS-Paradigma wird **CVMS-Paradigma** genannt, wenn die drei zugehörigen Module folgende Schichtenarchitektur bilden: *Controller*, *View*, *Model* und *Service*.

Die Controller-Module können direkt (z. B. via Unicast-Nachrichten) auf die View- und Modell-Module zugreifen, die View-Module können direkt auf Modell-Module zugreifen und die Modell-Module können direkt mit Service-Modulen kommunizieren, um Daten mit externen Repositories auszutauschen. Alle anderen „Zugriffe“ erfolgen nur indirekt (i. Allg. als Antworten auf Unicast-Nachrichten, evtl. auch

mittels Callback-Routinen).

Man beachte, dass in diesem Paradigma die Logik der Anwendung in den Controllern und nicht in den Modellen realisiert werden sollte, da die Modelle keinen direkten Zugriff auf andere Module haben.

2.3 MVCS-Multicast-Paradigma

Ein MVCS-Paradigma wird MVCS-Multicast-Paradigma genannt, wenn alle Module nur indirekt, d. h. mit Hilfe von Multicast-Nachrichten kommunizieren.

2.4 MVCS-Pattern (Definitionen nach Kowarschick (MMProg))

Wenn für ein MVCS-Paradigma – im Sinne eines Entwurfmusters – eine konkrete Klassen- und Objekt-Struktur für die vier Module Model, View, Controller und Service vorgegeben ist, spricht man von einem MVCS-Pattern.

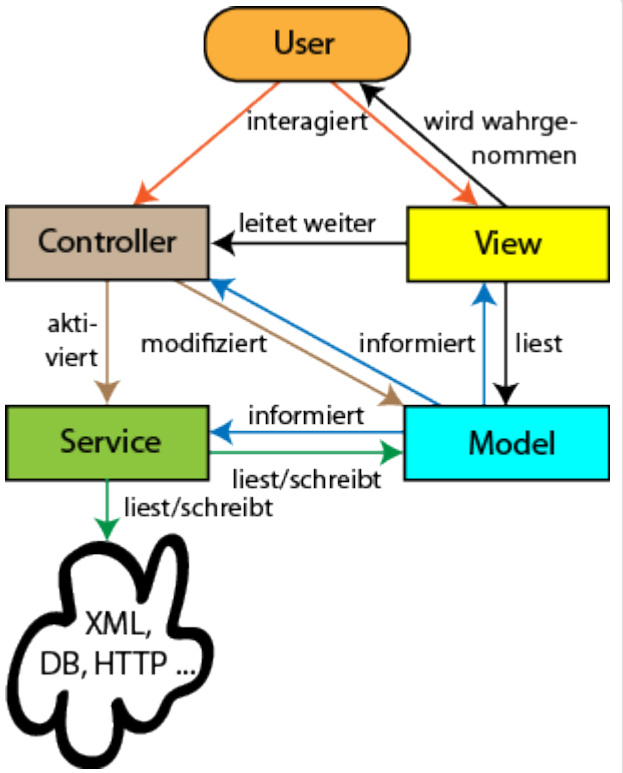
2.4.1 VCSM-Pattern

Ein MVCS-Pattern, das ein VCSM-Paradigma realisiert, wird auch **VCSM-Pattern** genannt.

2.4.2 MVCS-Multicast-Pattern

Ein MVCS-Pattern, das nur auf Multicast-Nachrichten basiert, wird auch **MVCS-Multicast-Pattern** genannt.

3 Der MVCS-Prozess



Der MVCS-Prozess

Das MVCS-Paradigma erweitert das MVC-Paradigma um die so genannte Servicemodule. Die Servicemodule übernehmen die Aufgabe, mit externen Anwendungen/Datenpools zu kommunizieren. Hierbei handelt es sich um eine Aufgabe, von der nicht klar ist, ob sie im MVC-Paradigma eher von den Modellmodulen oder den Steuerungsmodulen übernommen werden sollte. Externe Datenbestände gehören eigentlich zu Modell. Die Steuerung des Transports dieser Daten wäre allerdings eher eine Aufgabe der Steuermodule. Joe Berkovitz schlägt daher vor, für diese Aufgabe eigenständige Module zu etablieren. Die Aufgabe der Präsentationsmodule (Views) ändert sich nicht.[1]

Über ein Steuermodul kann der Benutzer das Modell manipulieren, das heißt, dessen Zustand (und damit auch die zugehörigen Views) ändern. Er kann insbesondere veranlassen, dass die Daten des Modells – mit Hilfe eines Servicemoduls – an eine externe Anwendung weitergeleitet oder in einer externen Datenbank, XML-Datei o.Ä. dauerhaft gespeichert werden. Auch das Lesen von Daten aus einer externen Datenquelle kann auf diese Weise initiiert werden.

Es ist sogar möglich, das Modell mit Hilfe einer Servicekomponenten automatisch mit einem externen Datenpool zu synchronisieren. Hierbei kommt keine Steuerkomponente zum Einsatz.

3.1 Beispiel Warenkorb

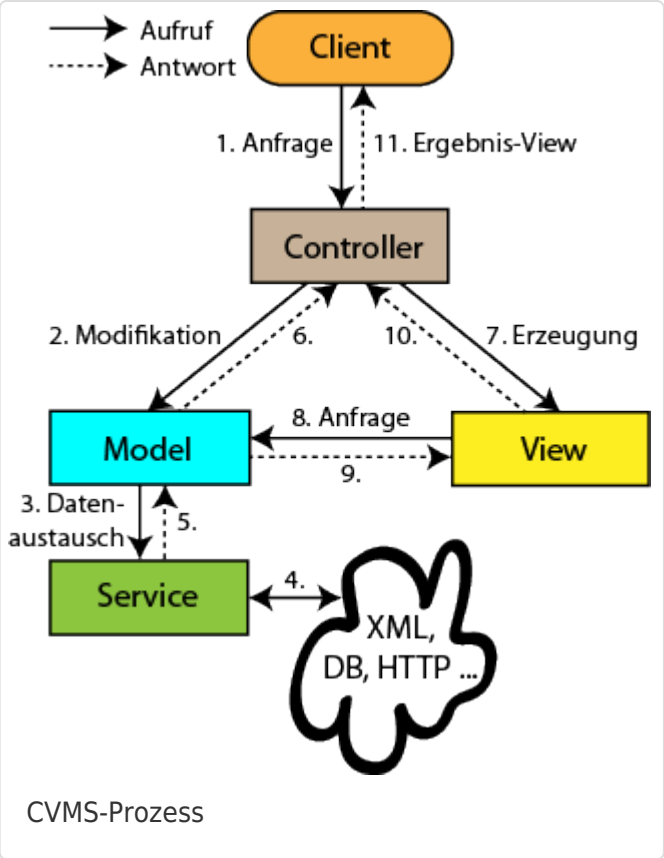
In einer Warenkorb-Anwendung werden im Modell Daten über den Warenkorb und den Besteller (sobald dieser eingeloggt ist) gespeichert: eine aktuelle Auswahl des Warenkataloges, den Inhalt des Warenkorbs, der Gesamtpreis der Waren im Warenkorb, der Adresse des Besterller etc. Der Warenkatalog selbst wird hingegen nicht im Modell, sondern in einer externen Datenbank abgelegt.

Die View visualisiert Elemente der aktuellen Auswahl des Warenkataloges (siehe Modell) mit Hilfe von Texten, Bildern und Videos. Der Benutzer kann die Auswahl der visualierten Elemente durch Filter oder Navigation abändern, er kann Elemente des Kataloges in seinen Warenkorb legen und dort wieder entfernen etc. Für all diese Aktionen werden dem Benutzer in der View spezielle Eingabefelder (Checkboxes, Drop-Down-Menüs, Textfelder, Links etc.) präsentiert. Jedes mal, wenn der Benutzer eine dieser Felder mit Hilfe der Maus oder der Tastatur bedient, leitet das Darstellungsmodul eine entsprechende Nachricht an das zuständige Steuermodul weiter.

Das Steuermodul analysiert jeweils die gewünschte Aktion des Benutzers und führt die entsprechenden Änderungen am Modell durch. Zum Beispiel kann sie veranlassen, die aktuelle Auswahl des Warenkataloges den Wünschen des Benutzers gemäß zu ändern, indem sie das entsprechende Servicemodul veranlässt, die neuen Daten aus der Datenbank in das Modell zu übertragen.

Wenn der Benutzer zu guter Letzt eine Bestellung tätigt, veranlassst das Steuermodul, dass die Bestellung aus dem Modell mit Hilfe eines Services an eine geeignete Anwendung weitergeleitet wird.

4 CVMS-Prozess, Model 2



CVMS-Prozess

Den meisten „Web Application Frameworks“ liegt heutzutage ein MVC-Paradigma zugrunde (vgl. MVC-Paradigma/Umsetzungen). Häufig wird das MVC-Paradigma dabei als CVMS-Schichtenarchitektur realisiert. Beispielsweise wurde in der Spezifikation von JSP 0.92[2][3] das so genannte Model 2 eingeführt, welches eine CVMS-Architektur beschreibt.

Bei einem CVMS-Prozess läuft die Kommunikation i. Allg. synchron ab (siehe Abbildung „CVMS-Prozess“):

Ein Client (wie z. B. ein Web-Browser) übermittelt an einen Controller eine Anfrage (1.) – z. B. in Form einer URL plus weiteren Daten – und wartet dann auf eine Antwort (11.): eine View, die er dem Benutzer präsentieren kann. (Dies gilt sogar, wenn Ajax zu Einsatz kommt. Auch hier wartet der Client, bis die Anwort eingetroffen ist. Die als Ergebnis vom Server gelieferte Teilview wird dann vom Client verwendet, um die aktuell dem Benutzer präsentierte clientseitige View zu aktualisieren.)

Um eine Antwort generieren zu können, leitet der Controller die erhaltenen Daten an ein Modell weiter (2.). Das Modell enthält dabei nicht nur typische Session-Daten – wie Session-ID, Benutzer-Daten, Warenkorb-Inhalt etc. –, sondern auch Informationen, die dem Benutzer präsentiert werden sollen – wie z. B. bestimmte Waren, für die sich der Benutzer aktuell interessiert. Das Modell kann mit Hilfe von Service-Modulen Daten aus externen Datenbeständen (Datenbanken, Web-Services, XML-Repositories etc.) nachladen oder auch Informationen dorthin übertragen (3., 4. und 5.).

Sobald das Modell die Daten vollständig verarbeitet hat, meldet es den Erfolg oder Misserfolg dieser Verarbeitung an den Controller (6.). Daraufhin stößt der Controler die Erzeugung (das Rendering) einer View an (7.). Welche View erzeugt wird, hängt dabei von den Daten (insbesondere der URL) ab, die dem Controller vom Client übergeben wurden (1.), sowie von der Antwort des Modells (6.). Wenn das Modell beispielsweise meldet, dass die Berechtigung zum Zugriff auf die gewünschten Daten fehlt, wird der Controller eine Login-View generieren, anderenfalls wird er eine View erzeugen, die die gewünschten Informationen präsentiert.

Der View-Renderer kann beliebig oft auf bestimmte Modelle zugreifen (8.), um Daten zu erfragen (9.),

die in die View integriert werden. Sobald der Rendervorgang abgeschlossen ist, erhält der Controller die erzeugte View (10.). Der Controller leitet diese als Antwort an den Client zurück (11.).

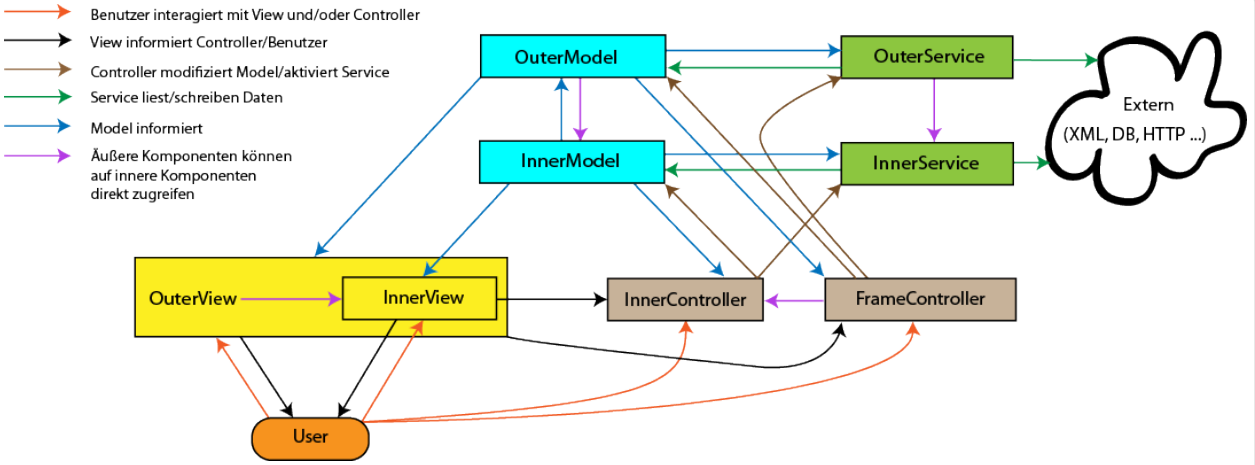
4.1 Beispiel „JSP“

siehe CVM-Prozess: Beispiel „JSP“

5 Kommunikation zwischen zwei MVCS-Komponenten

Genauso wie mehrere MVC-Komponenten miteinander kommunizieren können, können auch mehrere MVCS-Komponenten miteinander kommunizieren.

Viele Anwendungen bestehen aus mehreren MVCS-Komponenten: Eine oder mehrere Kernanwendungen (Domain-Komponenten) sowie einer Rahmenanwendung (Frame-Komponente), die den Zugang zu und zwischen den Kernanwendungen steuert. In diesem Fall gibt es mehrere relativ unabhängige MVCS-Prozesse. Die äußeren Komponenten (wie z. B. die Rahmenkomponente) können dabei mit den inneren Komponenten (wie z. B. die Kernkomponenten) kommunizieren. Der umgekehrte Weg sollte vermieden werden, damit Kernkomponenten problemlos in andere Umgebungen integriert werden können.



MVCS-Prozess 2 (mit Trennung in äußere und innere MVCS-Komponenten)

Eine typische Aufgabe eines Rahmenservices ist es, XML-Dateien zum Initialisieren des Systems einzulesen. Die Kernservice greifen dagegen häufig auf Datenbanken oder Web-Anwendungen zu, die bestimmte Daten des Kernmodells dauerhaft speichern (vgl. obiges Beispiel).

5.1 Beispiel „Spiel mit Highscore-Verwaltung“

Wenn man ein „Spiel mit Highscore-Verwaltung“ realisieren möchte, sollte man zwei unanhängige Kernanwendungen schreiben. Das Spiel selbst und eine spielunabhängige Highscore-Verwaltung. Die Rahmenanwendung verknüpft diese beiden Anwendungen.

Die Rahmenanwendung liest beispielsweise bei Spielende die erreichten Punkte aus dem Spielmodell aus und leitet diese an das Highscore-Modell weiter. Eine weitere Möglichkeit wäre, dass die

Rahmenanwendung über die Highscore-Anwendung ermittelt, ob der Spieler schon mindestens fünf Level erfolgreich gespielt hat. Nur in diesem Fall gewährt sie den Zugang zum Trainingsmodus des Spiels.

6 Quellen

1. **Berkovitz (2006)**: Joe Berkovitz; An architectural blueprint for Flex applications; https://web.archive.org/web/20070105004310/http://www.adobe.com:80/devnet/flex/articles/blueprint.html; 2006; Quellengüte: 2 (Web)
2. JavaServer™ Pages™ – Specification 0.92
3. Servlets and JSP Pages Best Practices
4. **Kowarschick (MMProg)**: Wolfgang Kowarschick; Vorlesung „Multimedia-Programmierung“; Hochschule: Hochschule Augsburg; Adresse: Augsburg; Web-Link; 2018; Quellengüte: 3 (Vorlesung)

7 Siehe auch

Model-View-Controller-Paradigma
Logic-Data-View-Controller-Service-Paradigma

Kategorien:
Objektorientierte Programmierung
Glossar
MVC
Kapitel:Multimedia-Programmierung