Kurz & Gut

Model-View-Controller

von Leif Singer

Einführung

Model-View-Controller (MVC) ist ein Design Pattern das zum Ziel hat, die Präsentation eines Programmes von seiner Logik und den Daten zu trennen. So soll vor allem die Wartbarkeit verbessert werden. Ausserdem werden Projekte strukturierter und flexibler, da die Logik oder auch die Ansicht einfacher austauschbar sind.

MVC wurde etwa 1978 im Xerox PARC (Palo Alto Research Center) im Zusammenhang mit Smalltalk-80 entwickelt und wird Trygve Reenskaug (Norwegen) zugerechnet. Heute gilt MVC als einer der bisher wichtigsten Entwicklungsschritte in der GUI Programmierung.

MVC klassisch

MVC organisert ein Programm in drei Aspekte: das Model, d.h. die Daten und Programmlogik, den View bzw. die Präsentationsschicht, sowie den Controller, der zwischen Model und View vermittelt.

Model

Das Model repräsentiert die eigentliche Kernfunktionalität sowie den internen Zustand der Anwendung und besteht zumeist aus mehreren Klassen. So finden sich hier Dinge wie Datencontainer, der Datenbankzugriff sowie alle von der GUI unabhängigen Berechnungen.

View

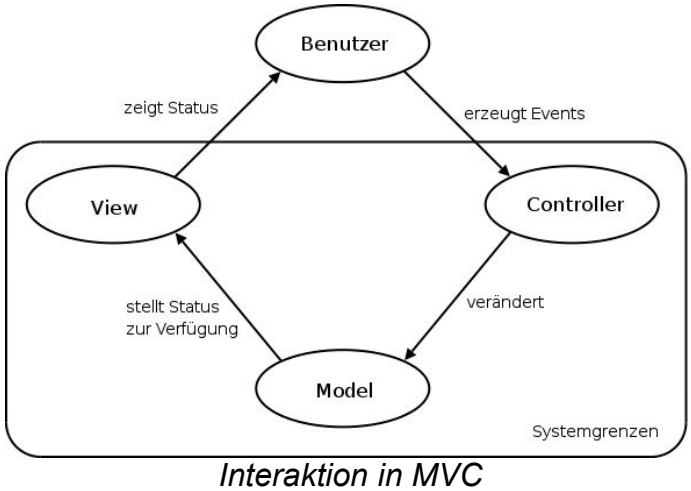
Der View generiert die Benutzeroberfläche der Anwendung. In Java sind dies zum Beispiel die Swing Klassen bzw. diejenigen Klassen in der Anwendung, die die Swing Oberfläche generieren. Bei Webanwendungen sind es diejenigen Programmteile / Klassen, die das HTML und CSS generieren. Der View greift hierbei auf die vom Model verwalteten Daten zurück.

Controller

Der Controller nimmt die im View generierten Anforderungen des Benutzers entgegen und übersetzt sie in Nachrichten an das Model. Hierfür muß der Controller natürlich die Events des Views interpretieren können, so daß es in der Regel pro View auch einen Controller gibt. Eine Ausnahme bilden hier Webanwendungen nach *Model 2* (s.u.). In Java AWT-/Swing-Anwendungen ist der Controller bspw. oft als *ActionListener* direkt im View implementiert.

Um die MVC-Struktur zu realisieren wird das Observer Pattern benutzt. Das Model ist ein Observable während die Views als Observer fungieren. Sobald sich das Model, d.h. der interne Zustand der Anwendung ändert, werden die Views hierüber informiert und haben so die Möglichkeit sich neu zu zeichnen, um den neuen Zustand zu reflektieren.

Es ist ebenso möglich, mehrere Views für ein Model zur Verfügung zu stellen – hierfür wird in der Regel auch je ein weiterer Controller benötigt. Dies ist zum Beispiel nötig, um die Views analog zum Composite Pattern verschachtelt aufzubauen und damit die Wiederverwendung von einmal fertiggestellten Komponenten zu vereinfach. Swing ist hierfür ein sehr gutes Beispiel.



Interaktion in MVC

Variationen

Heute hat sich MVC insbesondere auch in Webanwendungen durchgesetzt. Gerade in diesem Bereich, in dem die Trennung von Struktur, Inhalt und Präsentation zur Zeit eines der meistdiskutierten Themen ist, ist MVC vor

allem für mittlere und große Projekte unverzichtbar geworden.

Speziell bei Webanwendungen wird zwischen den MVC Variationen *Model 1* und *Model 2* unterschieden.

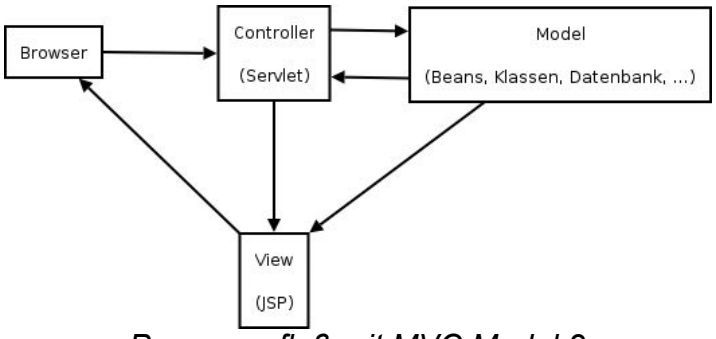
MVC Model 1 – "seiten-zentrisch"

In diesem Modell besteht eine Webanwendung aus einer Menge unabhängiger Seiten, die jeweils aus einem eigenen Controller und View bestehen, jedoch durchaus die gleichen Models (oft z. Bsp. Beans) benutzen können. Die Seiten referenzieren sich untereinander einfach über den Dateinamen bzw. den URL. Die Kommunikation zwischen den Seiten verläuft klassisch über direkte Übergabe von Parametern im URL oder über Formulare.

Diese Variante wird nur für kleine, schnelle Projekte angeraten, da bspw. Änderungen der API oder des Dateinamens einer Seite ggf. auf allen anderen Seiten manuell berücksichtigt werden müssen.

MVC Model 2 – "controller-zentrisch"

MVC Model 2 wird meist mit Hilfe von Frameworks wie bspw. *Apache Struts* oder Suns *Java Server Faces* realisiert, da es für die ganze Anwendung nur einen einzigen Controller (vgl. Patterns *FrontController* und *ApplicationController*) gibt und eine einheitliche API zum Aufruf von Methoden (hier *Actions*), dem Referenzieren von Seiten und dem Austausch von Parametern zwischen den einzelnen Seiten verwendet wird.



Programmfluß mit MVC Model 2

Hier geht eine Anfrage des Benutzers immer an den Controller. Dieser ruft ggf. zunächst Funktionen aus dem Model auf, die bei jedem Request durchgeführt werden müssen – bspw. Benutzerauthetifizierung oder Eingabevalidierung – und leitet dann die Anfrage in angepasster Form an den zuständigen View weiter. Dieser führt für auf dem Model Aktionen für die aktuelle Seite aus und gibt dann eine entsprechende Antwortseite an den Benutzer zurück.

Diese Variante wird aufgrund ihrer erhöhten Wartbarkeit für alle mittleren und größeren Webanwendungen empfohlen.

Fazit

Trotz seines fortgeschrittenen Alters hat MVC keineswegs an Aktualität verloren und ist übliche Praxis in der Software-Entwicklung. Zusätzlich zu den ursprünglich anvisierten Problemen hilft es selbst bei sehr aktuellen Problemstellungen wie Collective Code Ownership und der Berücksichtigung immer unterschiedlicherer Endgeräte.

Die Verwendung der einen oder anderen MVC Variation ist heute durch den hohen Gewinn an Übersicht und Wartbarkeit in so gut wie jedem Projekt empfehlenswert. Nur in sehr kleinen Anwendungen, Laborprototypen etc. kann es sinnvoll sein, auf MVC zu verzichten.

Literatur:

- http://de.wikipedia.org/wiki/Xerox
- http://www.parc.xerox.com/about/history/
- http://struts.apache.org/
- http://www.ibm.com/developerworks/java/library/j-struts/
- http://java.sun.com/j2ee/javaserverfaces/
- "Core J2EE Patterns" von Sun, ISBN 0131422464
- "Design Patterns" von Erich Gamma u.a., ISBN 0201633612
- http://en.wikipedia.org/wiki/MVC
- http://c2.com/cgi/wiki?ModelViewController
- http://www.javaworld.com/javaworld/jw-04-1998/jw-04-howto.html
- http://java.sun.com/blueprints/patterns/MVC-detailed.html