MVC

Das Model-View-Controller-Konzept

SE3 Team
Silvio Kunaschk

Juni 2009

Für die wahlobligatorische Vorlesung **Software Engineering 3** von Frau Professor Dr. Hauptmann, muß am Ende des Semesters in einer Gruppe von Studenten eine Belegarbeit zu einem gewählten Thema angefertigt werden. Im SS09 wählte unsere Gruppe das Thema **Model-View-Controller-Konzept**.

Die Zielstellung ist ein Software-System zu entwickeln zur Simulation des MVC-Konzeptes. Unsere Gruppe hat sich dazu entschieden, das Konzept mit Hilfe des *Model/View Programming Framework* von *Qt* zu erläutern.

Dieses Dokument beschreibt den allgemeinen Ansatz des Model-View-Controller-Konzeptes und erörtert dazu, anhand der Gesichtspunkte eines Frameworks, die Umsetzung des Model-View-Controller Paradigma im Model/View Programming Framework.

Die Gruppe besteht aus:

- Uwe Hausbrandt
- Tobias König
- Silvio Kunaschk

Inhaltsverzeichnis

1 Allgemeine Betrachtungen

In der Entwicklung von Software-Systemen haben sich (letztendlich) die Systeme durchgesetzt, welche mit Hilfe von graphischen Bedienoberflächen es dem Benutzer erlauben, das System bzw. eine Anwendung, interaktiv und nutzerfreundlich zu bedienen.

Dem Benutzer wird dadurch ein einfacher Zugriff auf das Software-System ermöglicht und dabei geholfen, eine Anwendung zu verstehen und schneller und komfortabler damit zu arbeiten.

An der Akzeptanz und Verbreitung von Computer-Systemen in nahezu allen Bereichen der Industrie und Wirtschaft, und in letzter Zeit auch immer mehr im privaten gesellschaftlichen Bereich, und der (sehr wahrscheinlichen) Annahme, dass die meisten Anwender nur über rudimentäre Informatikkenntnisse verfügen, aber trotzdem in der Lage sind mit Hilfe der graphischen Benutzerschnittstelle die verschiedensten Aufgaben mit Software-Systemen einfach und effizient zu lösen, kann man erkennen, welchen Stellenwert dieser Bestandteil von interaktiven Software-Systemen besitzt.

Möchte man, dass sich sein Anwendungssystem bzw. Applikation durchsetzt, ist es heutzutage unabdingbar eine ausgefeilte Benutzerschnittstelle mit zu implementieren, oder aber das System so zu entwickeln, dass dies von jeweiligen Fachleuten übernommen werden kann.

Spezifiziert man die Architektur eines interaktiven Software-Systems, muß man den funktionalen Teil von der Bedienschnittstelle unabhängig halten. So können Änderungen an den Teilen unabhängig voneinander durchgeführt werden, z.B. um die Benutzerschnittstelle an andere Bedürfnisse anzupassen, ohne Auswirkungen auf den Funktionalen Teil des Systems.

Für diese grundlegende strukurelle Organisation interaktiver Software-Systeme wendet man das Architekturmuster Model-View-Controller an.

2 Das Model-View-Controller Muster

2.1 Das MVC Paradigma

Das Model-View-Controller Muster teilt eine interaktive Anwendung in drei Komponenten auf.

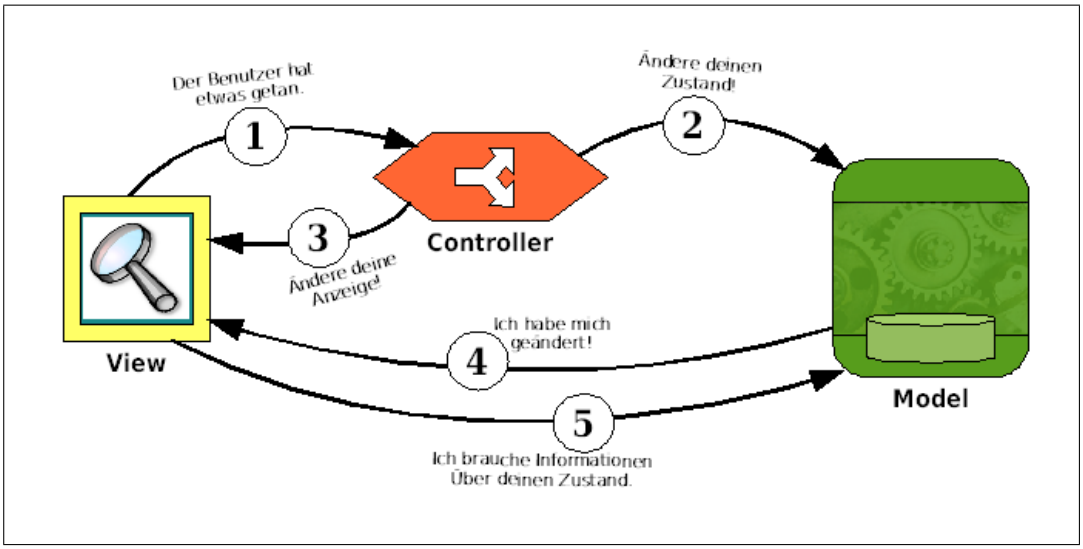

Abbildung 2.1: Die MVC Komponenten in Aktion

Model Das Model stellt das Anwendungsobjekt dar. Es enthält die gesamte Daten, Zustands- und Anwendungslogik. Es weiß nichts über Views und Controller, es bietet allerdings eine Schnittstelle an, über die sein Zustand beeinflußt und abgerufen werden kann. Außerdem kann es Benachrichtigungen über seine Zustandsänderungen an seine Beobachter senden.

View Die View ist die Bildschirmrepräsentation des Anwendungsobjektes. Sie erhält den Zustand und die Daten (normalerweise) direkt vom Model.

Controller Der Controller bestimmt die Möglichkeiten, mit denen die Benutzungsschnittstelle auf Benutzereingaben reagieren kann. Er nimmt die Eingaben des Benutzers entgegen und stellt fest, was diese für das Model bedeuten. Er verarbeitet die Bedieneingaben.

Die View- und die Controllerkomponente beschreiben zusammen die Bedienschnittstelle. Ein Benachrichtigungsmechanismus sichert die Konsistenz zwischen der Bedienschnittstelle und dem Model.

Das MVC-Paradigma entkoppelt die Benutzungsschnittstellen der View vom Model. Dies erhöht die Flexibilität und die Wiederverwendbarkeit.

2.2 Das MVC etwas genauer betrachtet

Schaut man beim Model-View-Controller etwas genauer hin, erkennt man, dass das MVC ein Satz von Mustern ist, die in einem Entwurf zusammenarbeiten.
Es besteht eigentlich aus 3 Entwurfsmustern die hier aber nur kurz erläutert werden sollen.

2.2.1 Observer-Muster

Das wichtigste Muster beim MVC-Entwurf, auch für dessen Verständis, ist das Observer-Muster, weshalb es hier jetzt genauer betrachtet wird.

Def. Definiere eine 1-zu-n-Abhängigkeit, zwischen Objekten, so dass die Änderung des Zustands eines Objektes dazu führt, dass alle abhängigen Objekte benachrichtigt und automatisch aktualisiert werden.

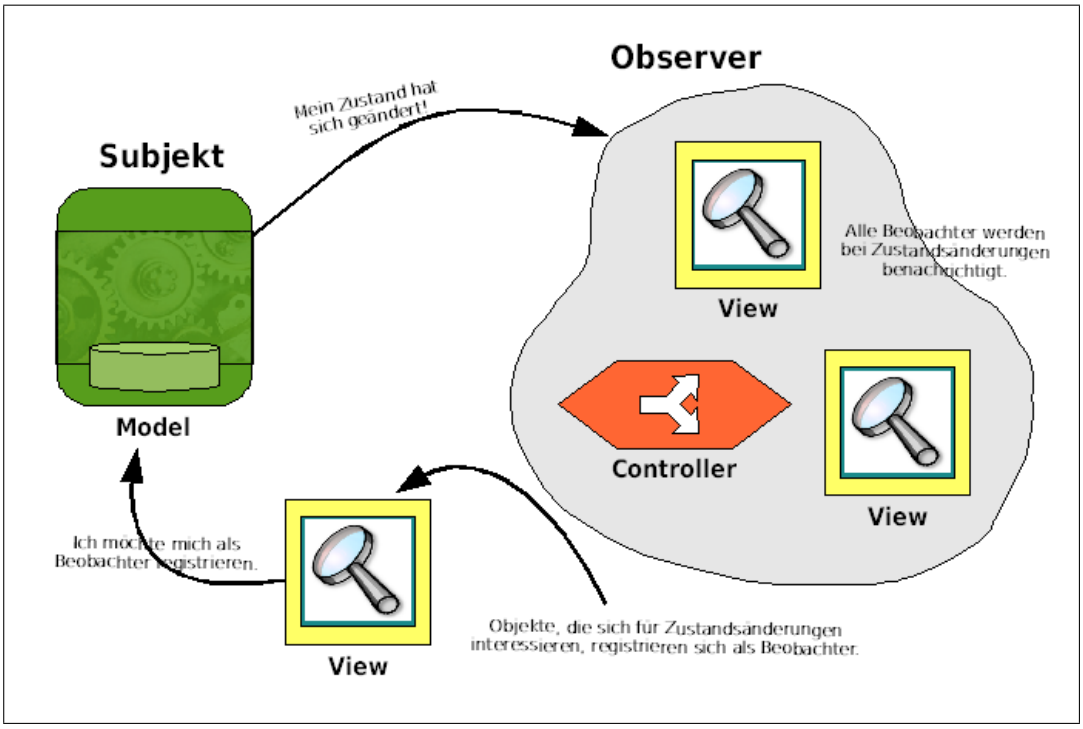

Abbildung 2.2: Das Observer-Muster in Aktion

Das Model nutzt das Observer-Muster, um View und Controller auf dem aktuellen Stand über die letzten Zustandsänderungen zu halten. Es macht das Model völlig unabhängig von View und Controller. So können für das gleiche Model unterschiedliche, oder sogar mehrere Views auf einmal verwendet werden.

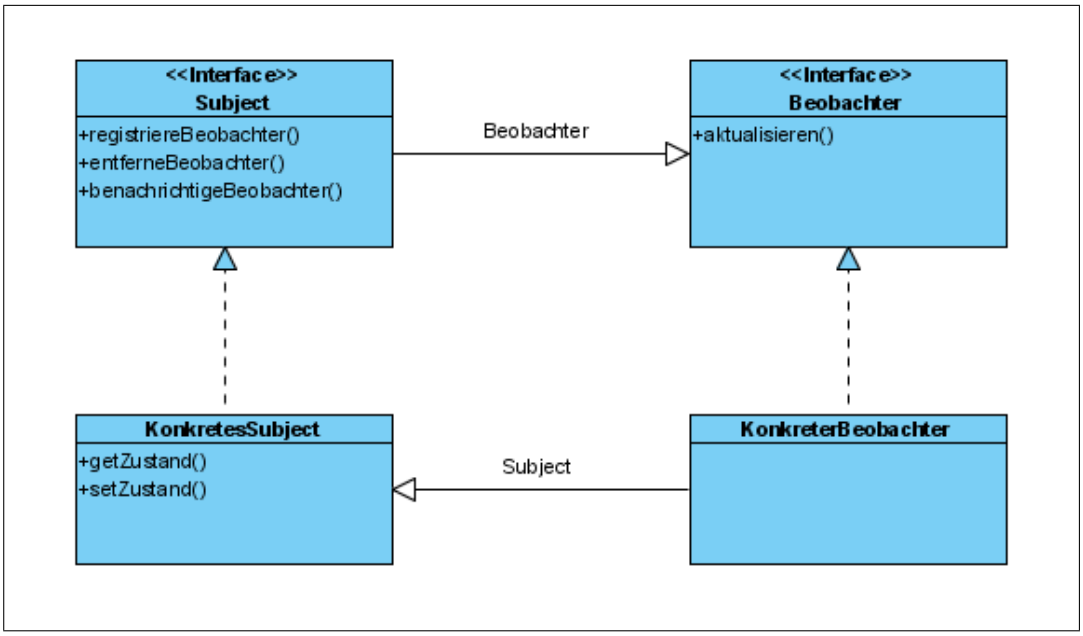

Abbildung 2.3: Das Observer-Muster Klassendiagramm

2.2.2 Strategy-Muster

Def. Definiere eine Familie von Algorithmen, kapsele jeden einzelnen und mache sie austauschbar. Das Strategy-Muster ermöglicht es, den Algorithmus unabhängig von ihn nutzenden Klienten zu variieren.

Die View und der Controller implementieren das Strategy-Muster. Der Controller ist das Verhalten der View und kann leicht gegen einen anderen Controller ausgetauscht werden, wenn ein anderes Verhalten gewünscht wird.

Die View ist ein Objekt, das mit einer Strategie konfiguriert ist; der Controller liefert diese Strategie. Die View ist nur für die Anzeige der Anwendung zuständig; alle Entscheidungen über das Verhalten der Schnittstelle *delegiert* sie an den Controller. Durch die Verwendung des Strategy-Muster bleibt die View vom Model *entkoppelt*, denn der Controller ist ja bei der Bearbeitung der Benutzeraktionen für die Interaktion mit dem Model zuständig. Die View weiß nicht wie das vor sich geht.

Die View delegiert die Verarbeitung der Benutzeraktionen an den Controller. Der Controller übersetzt die Eingaben des Benutzers in Aktionen auf dem Model.

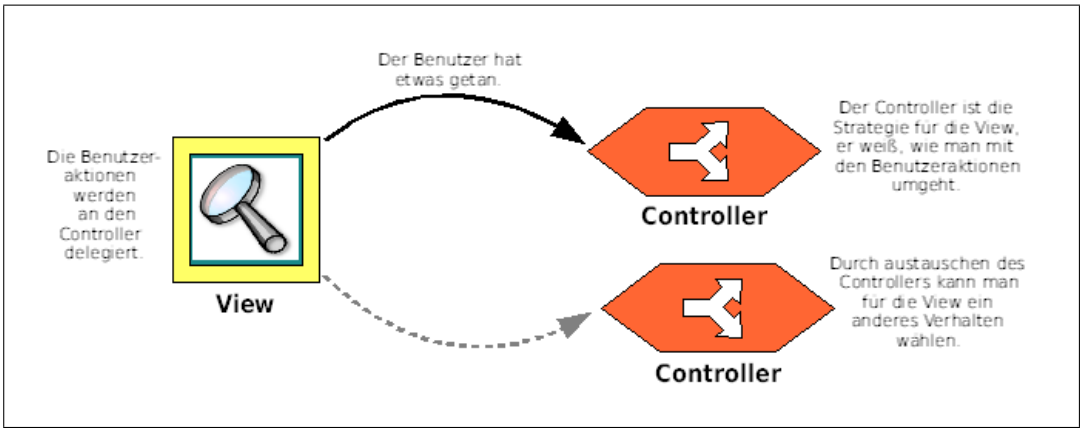

Abbildung 2.4: Das Strategy-Muster in Aktion

2.2.3 Composite-Muster

Die View ist ein Kompositum aus GUI-Komponenten (Labels, Buttons, Texteingabefentern, usw.). Die oberste Komponente enthält andere Komponenten, die wiederum weitere Komponenten enthalten, bis man beim Blattknoten angelangt ist. Sie verwendet dieses Muster intern, um die Bestandteile der Anzeige zu verwalten.

2.3 Nachteile von MVC

Größere Komplexität. Nicht immer ist die strikte Einhaltung der Model-View-Controller-Struktur die beste Art, eine interaktive Anwendung zu entwickeln. Es kann sein das die Verwendung vom MVC die Komplexität der Anwendung erhöht ohne den Zugewinn an Flexibilität.

Potential für eine übermäßige Anzahl von Aktualisierungen. Hat zum Beispiel eine einfache Aktion des Anwenders viele Aktualisierungen zur Folge, sollte das Model unnötige Benachrichtigungen über Änderungen auslassen. Eine View die gerade nicht sichtbar ist, braucht nicht benachrichtigt zu werden.

Enge Verbindung zwischen View- und Controllerkomponenten. View und Controller sind eigene, aber eng gekoppelte Komponenten, was deren jeweilige Wiederverwendung behindert. Es ist unwahrscheinlich, dass eine View ohne ihre Controllerkomponente oder umgekehrt verwendet wird.

3 Framework

3.1 Ein kurzer Überblick

Ein Framework besteht aus einer Menge von zusammenarbeitenden Klassen, die einen wiederverwendbaren Entwurf für eine bestimmte Klasse von Software darstellen. Das Framework bestimmt die Architektur der Anwendung. Es definiert:

- die Struktur im Großen
- Unterteilung in Klassen und Objekte
- die jeweiligen zentralen Zuständigkeiten
- die Zusammenarbeit der Klassen und Objekte sowie den Kontrollfluß

Ein Framework legt diese Entwurfsparameter im voraus fest, so dass der Entwickler sich auf die spezifischen Details seiner Anwendung konzentrieren kann. Verwendet man ein Framework, schreibt man den Code, der vom Framework gerufen wird. Dies wird erreicht indem man Operationen mit bestimmten Namen und Aufrufkonventionen schreiben muß. Dies reduziert die zu treffenden Entwurfsentscheidungen. Diese sind bereits von anderen getroffen worden.

Speziell bei GUI-Frameworks ermöglicht diese Vorgehensweise ein sehr viel schnelleres Entwickeln von Anwendungen[1]. Diese haben eine ähnliche Struktur und sind einfacher Wiederverwendbar. Auf der einen Seite schränkt dies die Kreativität ein, aber man erhält Komponenten in denen Erfahrung steckt und die schon erprobt sind. Dem Benutzer der Anwendung sind diese Verhaltensweisen in der Regel schon bekannt.

[1] in diesem Fall von den View-Komponenten der Anwendung

4 Das Model/View Programming Framework von Qt

4.1 Was ist Qt?

Qt ist das de facto Standard C++ Framework für ein sehr schnelles Entwickeln von Cross-Platform-Software. Zusätzlich zu einer sehr Umfangreichen C++ Klassenbibliothek enthält Qt Werkzeuge, die das Schreiben von Anwendungen erleichtern und beschleunigen.

Qt enthält eine große Menge von Widgets, welche Standard GUI Funktionalität zur Verfügung stellen. Qt ist ein weltweit verwendetes und ausgereiftes C++ Framework.

Zu den vielen kommerziellen Verwendungen von Qt, ist die Open Source Edition von Qt die Grundlage von KDE, der Linux Desktop Umgebung.

Literaturverzeichnis

[1] Frank Buschmann. *Pattern-orientierte Software-Architektur . Ein Pattern-System.* Addison-Wesley, 2. aufl. edition, 1 1998.

[2] Eric Freeman, Elisabeth Freeman, Kathy Sierra, and Bert Bates. *Entwurfsmuster von Kopf bis Fuß.* O'Reilly, 1 edition, 12 2005.

[3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software.* Addison-Wesley, München, 1 edition, 2 2009.