

Model-View-Controller

SE3 Team

Juni 2009

Model-View-Controller

Analyse

Entwurf und Entwicklung

Model-View-Controller

Allgemeine Betrachtungen

zu graphischen Bedienoberflächen

- ▶ interaktiv, nutzerfreundlich und komfortabel
- ▶ haben sich in Software-Systemen durchgesetzt
- ▶ heutige Akzeptanz und Verbreitung zeigt
 - ▶ wichtiger Bestandteil von Anwendungssystemen
 - ▶ interaktive SW-Systeme haben sehr hohen Stellenwert
- ▶ Architekturmuster **MVC**
 - ▶ grundlegende strukturelle Organisation
 - ▶ Unabhängigkeit des funktionalen Teils von der Bedienschnittstelle

Das MVC Muster

die Komponenten

Teilt eine interaktive Anwendung in 3 Komponenten auf.

► **Model**

- enthält die gesamte Daten, Zustands- und Anwendungslogik
- Zustandsänderung über Schnittstelle
- Benachrichtigungen über Änderungen an Beobachter

► **View**

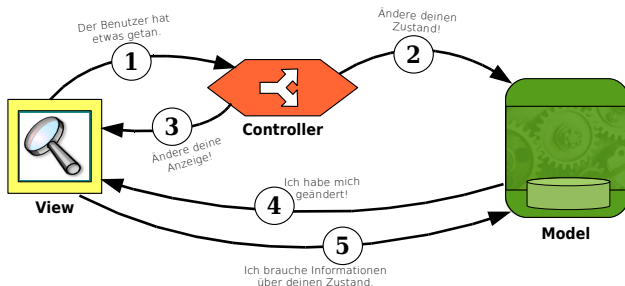
- Bildschirmrepräsentation des Anwendungsobjektes
- erhält Zustand und Daten direkt vom Model

► **Controller**

- nimmt Eingaben des Nutzers entgegen und verarbeitet sie

Das MVC Muster

die Komponenten



View- und Controller beschreiben die Benutzeroberfläche.

MVC etwas genauer betrachtet

Das Observer-Muster

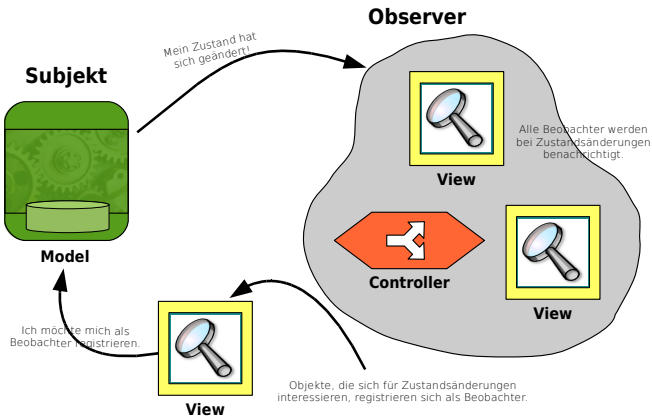
Wichtigstes Muster für Verständnis des MVC.

- ▶ **Zweck**

- ▶ 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.

MVC etwas genauer betrachtet

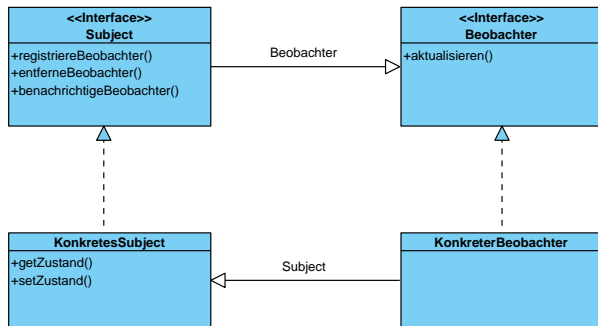
Das Observer-Muster



Es macht das Model völlig unabhängig von View und Controller.

MVC etwas genauer betrachtet

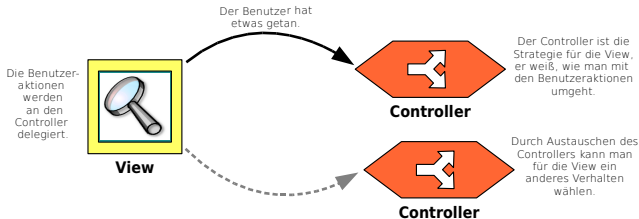
Das Observer-Muster als Klassendiagramm



- ▶ Beobachter registriert sich beim Subjekt
- ▶ Subjekt fügt es Liste seiner Beobachter hinzu
- ▶ Subjekt benachrichtigt alle registrierten Beobachter
- ▶ Subjekt bietet Zugriff über Schnittstelle an

MVC etwas genauer betrachtet

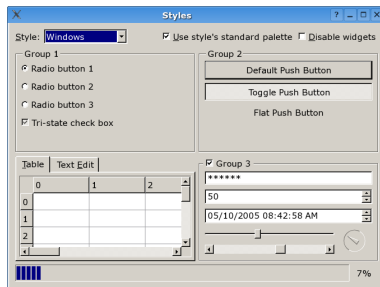
Das Strategy-Muster



- ▶ View ist mit einer Strategie konfiguriert
- ▶ Controller ist das Verhalten der View
- ▶ kann ausgetauscht werden
- ▶ View delegiert Benutzeraktionen an den Controller

MVC etwas genauer betrachtet

Das Composite-Muster



Die GUI ist ein Kompositum.

- ▶ besteht Label, Buttons, Texteingabefelder, ...
- ▶ Komponenten enthalten andere Komponenten
- ▶ wird intern verwendet um Bestandteile der Anzeige zu verwalten

Nachteile von MVC

in bestimmten Fällen

- ▶ größere Komplexität der Anwendung ohne Zugewinn an Flexibilität
- ▶ Potential für eine übermäßige Anzahl von Aktualisierungen
- ▶ enge Verbindung zwischen View- und Controllerkomponenten

Framework

Ein kurzer Überblick

- ▶ besteht aus einer Menge von zusammenarbeitenden Klassen
- ▶ Wiederverwendbarkeit für den Entwurf einer bestimmten Klasse von Software
- ▶ definiert
 - ▶ die Struktur im Großen
 - ▶ Unterteilung in Klassen und Objekte
 - ▶ die jeweiligen zentralen Zuständigkeiten
 - ▶ Zusammenarbeit und Kontrollfluß
- ▶ legt Entwursparameter im voraus fest
- ▶ Komponenten beinhalten Erfahrungen und sind erprobt

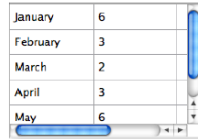
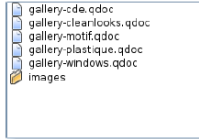
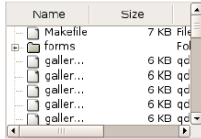
Model/View Programmierung mit dem Qt Framework

Was ist Qt?

- ▶ de facto Standard C++ Framework für die Entwicklung von Cross-Platform-Software
- ▶ enthält Widgets mit Standard GUI-Funktionalität
- ▶ Open Source Edition ist Grundlage von KDE

Model/View Programmierung mit Qt

Item Views



- ▶ Item-View-Widgets sind Standard GUI-Bedienungselemente
- ▶ List-, Tree-, Table-Views
- ▶ äquivalente Model/View Komponenten
 - ▶ *QListView*
 - ▶ *QTableView*
 - ▶ *QTreeView*

Model/View Programmierung mit Qt

Die Model/View Architektur

- ▶ Variante des MVC speziell angepaßt für Qt's Item Views
- ▶ verwendet Models um Daten anderen Komponenten zur Verfügung zu stellen
- ▶ Views präsentieren Daten
- ▶ Delegates behandeln Rendering- und Bearbeitungsprozesse

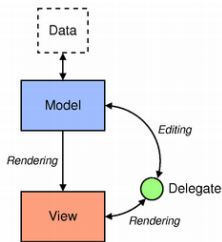
Model/View Programmierung mit Qt

Das Model/View Framework

- ▶ Variante des MVC speziell angepaßt für Qt's Item Views
- ▶ verwendet Models um Daten anderen Komponenten zur Verfügung zu stellen
- ▶ Views präsentieren Daten
- ▶ Delegates behandeln Rendering- und Bearbeitungsprozesse
- ▶ ermöglicht eine ganze Reihe Vorteile gegenüber den klassischen ItemViews

Model/View Programmierung mit Qt

Die Model/View Architektur



- ▶ resultiert aus der Kombination von View und Controller in einer Komponente
- ▶ dies ermöglicht einen Framework basierten Ansatz auf der Grundlage des MVC
- ▶ mit Delegates kann man individuell auf Benutzereingaben reagieren

Mit *Proxy Models* können Daten von Models transformiert werden.
Dies ermöglicht Sortierung und Filterung von Daten.

Model/View Programmierung mit Qt

Die Model/View Architektur

► **Model**

- kommuniziert mit Datenquelle
- bietet Standardinterface für Zugriff der anderen Komponenten

► **View**

- bekommt Model-Indizes vom Model
- diese referenzieren Daten-Items

► **Delegate**

- rendert die Daten-Items in View
- wird Item bearbeitet werden ebenfalls Model-Indizes verwendet

Komponenten werden von abstrakten Klassen definiert, welche Standardinterfaces anbieten.

Model/View Programmierung mit Qt

Die Model/View Architektur

Kommunikation der Komponenten mittels *Signals* und *Slots*¹.

- ▶ Signals vom Model informieren View über Datenänderungen
- ▶ Signals von der View bieten Informationen über Benutzeraktionen auf Daten-Items
- ▶ Signals vom Delegate während der Editierung verwendet, um Model und View über aktuellen Bearbeitungszustand zu informieren

¹Qt-Mechanismus für die Kommunikation zwischen Objekten

Model/View Programmierung mit Qt

weitere Informationen im Internet

<http://www.qtsoftware.com>

Analyse

Darstellung des Model-View-Controller- Konzeptes

- ▶ Welche Art der Applikation?
 - ▶ Fahrplan- Applikation?
 - ▶ Anzeige des Zugfahrplans gesamt
 - ▶ Anzeige des Fahrplans an bestimmter Haltestelle
 - ▶ *Gab es schon...*
 - ▶ Kinoinformation?
 - ▶ Anzeige der aktuell laufenden Filme
 - ▶ Anzeige der demnächst laufenden Filme
 - ▶ Wetterinformation? **weatherinfo**

Anforderungen an die Beispielapplikation

- ▶ Welche Wetterdaten sollen dargestellt werden, für welchen Zeitraum und für welche Städte? **Model**
- ▶ Welche Anzeigearten wollen wir implementieren? **View**
- ▶ Welche Funktionalitäten in den Views sollen implementiert werden? *Controller*

Das Model

- ▶ Was ist als Wetterinformation sinnvoll?
 - ▶ Temperatur
 - ▶ Bewölkung
 - ▶ Windstärke
 - ▶ Windrichtung
- ▶ Ein Zeitraum von 5 Tagen (*längere Vorhersagen grenzen an Wahrsagerei*)
- ▶ Welche Städte und welche Zusatzinformationen?
 - ▶ Dresden, Oslo, Springfield, ...
 - ▶ Weltkoordinaten (Längen- und Breitengrad) für die Ortsbestimmung

Die Views

- ▶ Welche Views?
 - ▶ Verlaufskurve der Temperatur: **temperature_view**
 - ▶ Wetterinformationen für eine bestimmte Stadt: **day_view**
 - ▶ Anzeige der Temperatur und Bewölkung in Tabellenform: **table_view**
 - ▶ Anzeige der Bewölkung auf einer Weltkarte: **world_view**
- ▶ Design der Oberflächen der Views
 - ▶ per Handzeichnung im ersten Schritt diskutiert und definiert
 - ▶ nachfolgend dann von Implementierer durch das Framework realisiert

Die Funktionalität der Views

- ▶ **temperature_view**
 - ▶ Auswahl der Stadt
- ▶ **day_view**
 - ▶ Auswahl der Stadt
 - ▶ Auswahl des angezeigten Tages
- ▶ **table_view**
 - ▶ Einschränkung der angezeigten Städte durch einen Filter
 - ▶ Filter soll case-insensitive sein
 - ▶ Änderung der Temperatur für eine Stadt und einen Tag
 - ▶ Temperatureintrag soll editierbar werden nach einem Doppelklick
- ▶ **world_view**
 - ▶ Auswahl des angezeigten Tages

Entwurf und Entwicklung

My Title

► foobar