Model-View-Controller-Pattern

Wechseln zu:Navigation, Suche

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

Ein Model-View-Controller-Pattern, kurz MVC-Pattern, bezeichnet ein Entwurfsmuster, das auf dem Model-View-Controller-Paradigma beruht, das also für die MVC-Module Model, View und Controller geeignete Klassen- und Objekt-Strukturen definiert.

2 Verschiedene

Realisierungsmöglichkeiten

Es gibt mehrere Möglichkeiten, geeignete Klassen- und Objekt-Strukturen zu definieren, d. h., es gibt mehrere MVC-Pattern-Varianten.

Zwei wichtige Aspekte, die dabei berücksichtigt werden müssen, sind die

Kommunikation: Auf welche Arten kommunizieren die Module miteinander?
Initialisierung: Wie werden die Module initialisiert?

Basierend auf diesen Vorüberlegungen sollen drei mögliche MVC-Pattern vorgestellt werden:

Initialisierung mittels Singleton(s) (weniger empfehlenswert)
Initialisierung mittels Dependency Injection (empfehlenswert)
Kommunikation ausschließlich mittels des Observer Patterns (empfehlenswert)

3 Kommunikation zwischen MVC-Modulen

VCM-Prozess

Ein wichtiges Programmierprinzip ist es, so wenige Abhängigkeiten wie möglich zu erzeugen (*few interfaces*), da Abhängigkeiten die Komplexität und die Fehleranfälligkeit erhöhen. Die Abhängigkeiten entstehen dadurch, dass die MVC-Module miteinander kommunizieren müssen.

Für die Kommunikation zwischen zwei Modulen gibt es mehrere Möglichkeiten. Die wichtigsten davon sind:

Unicast-Nachrichten: Modul A ruft Methoden von Modul B direkt auf.
Multicast-Nachrichten: Modul A informiert alle Module, die als Nachrichtenempfänger angemeldet wurden, mit Hilfe von Signalen (Observer-Pattern).
    Der Sender informiert mit einer Nachricht die Empfänger lediglich, dass sich etwas geändert hat. Die Empfänger müssen sich daraufhin die für sie wichtigen Informationen per direktem Zugriff vom Sender holen.
    Der Sender schickt in der Nachricht detailierte Informationen mit, so dass der Empfänger danach nicht mehr auf den Sender zugreifen muss.

Weitere Kommunikationsmöglichkeiten, wie z. B. Broadcast-Nachrichten, die Kommunikation über globale Variablen (insbesondere über Klassenattribute) etc., sollten i. Allg. vermieden werden. Bei Broadcast-Nachrichten, die an eine Vielzahl von nicht-beteiligten Objekten gehen, muss jeder einzelne Empfäger entscheiden, ob die jweils empfangene Nachricht für ihn überhaupt von Interesse ist. Bei globalen Variablen kann i. Allg. nur schwer nachvollzogen werden, wer welche Variablen zu Kommunikation nutzt oder ob bestimmte Variablen überhaupt noch zur Kommunikation benötigt werden. Der zugehörige Code wird im Laufe der Zeit im unwartbarer.

4 Initialisierung von MVC-Modulen

Je nach Art der Kommunikation müssen die verschiedenen Module unterschiedlich initialisiert werden:

Unicast-Nachrichten: Modul A muss das Modul B kennen, bevor es mit diesem Modul kommunizieren

kann.
Multicast-Nachrichten: Modul B muss sich bei Modul A als Empfänger anmelden oder es muss von dritter Seite angemeldet werden, bevor es Nachrichten von A empfängt.
Modul B muss Modul A kennen, wenn Modul B sich sebst als Empfänger bei Modul A anmelden muss oder wenn Modul B nach Empfang einer Nachricht weitere Informationen von Modul A erfragen muss.
Wenn Modul B von einem Dritten als Empfänger angemeldet wird und Modul A alle notwendigen Informationen in den Multicast-Nachrichten mitschickt, braucht Modul B das Modul A nicht zu kennen.

5 Das MVC-Pattern als 3-Schichten-Pattern

Das MVC-Pattern als Spezialfall des Layer-Patterns

Man kann die drei MVC-Module in Schichten anordnen: In der untersten Schicht ist das Modell angesiedelt, darüber liegt der Controller und zuoberst die View.

Das Layer- oder Schichtenpattern erlaubt nur, dass höhere Schichten direkt auf tiefere zugreifen, aber nicht umgekehrt. Ordnet man die drei MVC-Module so an, wie zuvor beschrieben, darf die View sowohl auf den Controller als auch auf das Modell zugreifen und der Controller auf das Modell. Das Modell darf dagegen auf kein anderes Modul direkt zugreifen.

Diese Schichteneinteilung ist sinnvoll. Wenn ein Modell, d. h. eine Abbildung eines Realitätsauschnitts auf ein Datenmodell, erstellt wird, ist weder festgelegt, wie die Manipulationen an diesem Modell realisiert werden, noch, auf welche Arten diese Daten den Benutzern präsentiert werden. Auch der Controller, der die Benutzereingaben verarbeitet, braucht nicht zu wissen, welche Views das Modell darstellen. Eine View dagegen muss das Modell kennen, dessen Daten sie visualisiert. Außerdem muss sie einen geeigneten Controller kennen, wenn sie Benutzeraktionen, wie z. B. Mausklicks auf bestimmte Buttons, verarbeiten lassen will.

5.1 Heterogene Kommunikationswelt: Unicast- und Multicast-Kommunikation

Die Drei-Schichten-Architektur legt folgendes Kommunikationsverhalten nahe:

Ein View-Objekt kommuniziert direkt mit den ihr zugeordneten Controller- und Modell-Objekten sowie evtl. mit anderen View-Objekten.
Ein Controller-Objekt kommuniziert direkt mit den ihr zugeordneten Modell-Objekten sowie evtl. mit anderen Controller-Objekten.
Ein Modell komminiziert höchstens mit anderen Modellobjekten direkt. Mit allen anderen Objekten kommuniziert sie nur per Multicast.

Auf diese Weise ist sichergestellt, dass zunächst alle Modell-Objekte erzeugt und initialisert werden können, da dafür keine anderen Objekte bekannt sein müssen. Danach werden die Controller-Objekte erzeugt und initialisiert. Die Modell-Objekte, mit den die Controller kommunizieren, existieren zu

diesem Zeitpunkt schon. Zu guter Letzt werden die View-Objekte erzeugt und initialisiert.

5.2 Homogene Kommunikationswelt: Reine Multicast-Kommunikation

Ein zweite Möglichkeit wäre, alle Objekte per Multicast-Nachrichten kommunizieren zu lassen. Allerdings sollten in diesem Fall die Empfänger-Objekte jeweils von dritter Stelle, das heißt vom Initialisierungs-Modul, beim Sender registriert werden. Außerdem müssten in den Multicast-Nachrichten alle wesentlichen Informationen mitgeschickt werden. Anderenfalls müsste beispielsweise ein Modell-Objekt alle Controller kennen, die mit diesen Modell-Objekt kommunizieren wollen: Entweder, um sich als Nachrichtenempfänger bei „seinen“ Controllern zu regisrieren, oder, um, sobald Nachrichten vom Controller eintreffen, nähere Informationen vom Controller einholen zu können, welche Änderungen am Modell eigentlich vorgenommen werden sollen.

6 Drei verschiedene Arten von MVCS-Pattern

6.1 Initialisierung mittels Singleton(s)

Bei diesem Pattern kommunizeren ebenfalls nur die Modell-Objekte mittels Multicast-Nachrichten. Die anderen Objekte kommunizieren mittels Unicast-Nachrichten.

Initialisiert werden alle Module mittels Zugriffen auf ein oder mehrere Singleton-Klassen.

Der Nachteil dieses Patterns ist, dass die Singleton-Klassen wie globale Variablen wirken. Daher sollte auf diese Art der Realisierung im Allgemeinen verzichtet werden.

Details siehe: DMVC-Pattern: Initialisierung mittels Singleton(s)

6.2 Initialisierung mittels Dependency Injection

Bei diesem Pattern kommunizeren nur die Modell-Objekte mittels Multicast-Nachrichten. Die anderen Objekte kommunizieren mittels Unicast-Nachrichten.

Initialisiert werden alle Module mittels Dependency Injection.

Der Nachteil dieses Patterns ist, dass View- und Controller-Module ihre Kommunikationspartner kennen müssen. Dies widerspricht dem Programmierprinzip, die Schnittstellen möglichst klein zu halten (small interfaces). Allerdings kann man dieses Problem abmildern, wenn man für jeden Kommunikationskanal eigene, möglichst schlanke Kommunikationsschnittstellen definiert.

Details siehe: MVC-Pattern: Initialisierung mittels Dependency Injection

6.3 Kommunikation ausschließlich mittels des Observer-Patterns

Bei diesem Pattern kommunizieren alle MVC-Objekte mittels Multicast-Nachrichten (die alle notwendigen Informationen beeinhalten, so dass der Empfänger nie beim Sender rückfragen muss).

Die Zuweisung der Nachrichten-Sender zu ihren Nachrichten-Empfängern erfolgt während der Initialisierungsphase von außerhalb. Das heißt, die einzelnen MVC-Objekte kennen einander nicht.

Der Nachteil dieses Patterns ist, dass die eine Multicast-Kommunikation weniger effizient ist, als eine Unicast-Kommunikation. In zeitkritischen Anwendungen kann dies evtl. zu Problemen führen.

Details siehe: MVC-Pattern: Kommunikation ausschließlich mittels des Observer Patterns

7 Quellen

Gamma et al. (1995): Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides; Design Patterns – Elements of Reusable Object-Oriented Software; Auflage: 1; Verlag: Addison-Wesley Longman; Adresse: Amsterdam; ISBN: 0201633612; 1995; Quellengüte: 5 (Buch)
Kowarschick (MMProg): Wolfgang Kowarschick; Vorlesung „Multimedia-Programmierung“; Hochschule: Hochschule Augsburg; Adresse: Augsburg; Web-Link; 2018; Quellengüte: 3 (Vorlesung)

8 Siehe auch

Model-View-Controller-Paradigma

Kategorien:
MVC
Objektorientierte Programmierung
Glossar
Kapitel:Multimedia-Programmierung

Diese Seite wurde zuletzt am 22. September 2017 um 16:50 Uhr bearbeitet.
Inhalt verfügbar unter CC BY-SA 4.0.