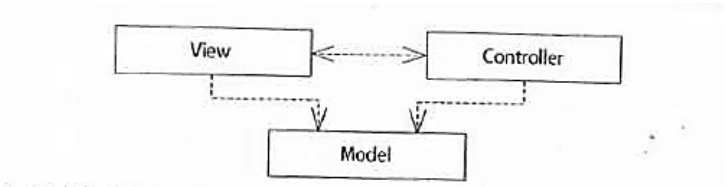
Das Model View Controller (MVC) Entwurfsmuster

Zweck

Die Verantwortlichkeiten beim Aufbau von Benutzerschnittstellen werden auf drei verschiedene Rollen verteilt, um die unterschiedliche Präsentation derselben Information zu er leichtern. Sie entwickeln eine Software zur Medienverwaltung (CDs, DVDs etc.). Die Benutzerschnittstelle (User Interface, UI) soll möglichst flexibel gehalten werden. Neben konventionellen grafischen Benutzeroberflächen kommen beispielsweise auch HTML -Seiten oder eine iOS-App infrage. Sie müssen die grafische Darstellung unabhängig von anderen Systemteilen austauschen können. Code-Redundanz wollen Sie dabei weitgehend vermeiden.

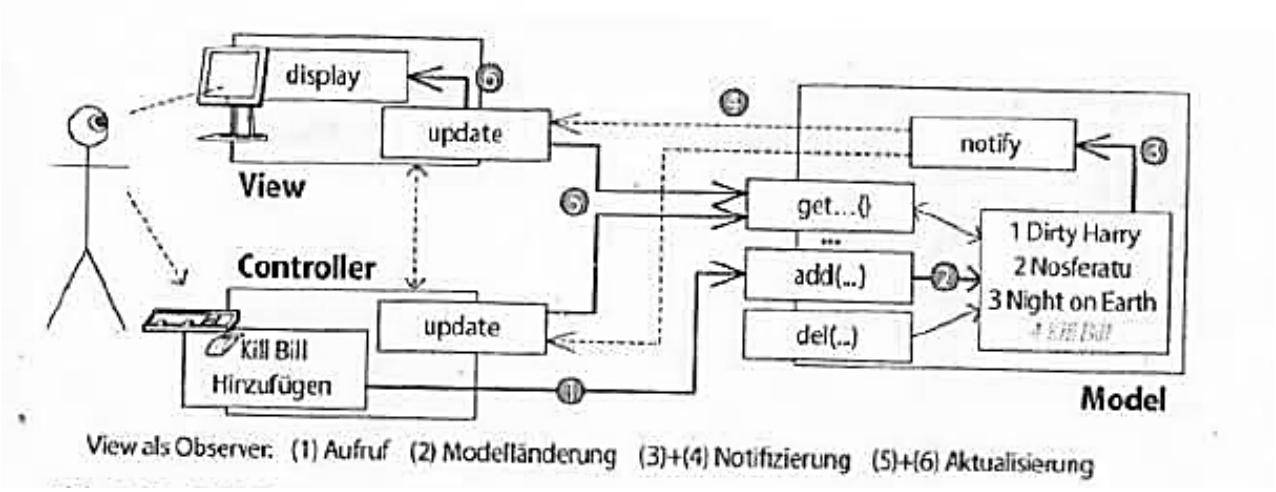


Problem/Kontext

Sie möchten eine Benutzerschnittstelle von der Anwendungsfunktionalität entkoppeln. Das User-Interface soll häufiger als die Geschäftslogik angepasst oder in verschieden Formen angeboten werden.

Lösung

Trennen Sie die Verantwortlichkeiten (Separation of Concems) für die Darstellung (View), die Verwaltung der darzustellenden Daten (Model) und die Kontrolle der Benutzereingaben bzw. der Änderung von Daten (Controller). Die View beinhaltet die visuellen Elemente (Fenster, Buttons oder auch Frames einer HTML-Seite). Das Modell hingegen verwaltet Daten unabhängig von deren Präsentation. Die dritte Komponente bildet der Controller, der Benutzereingaben verarbeitet, die Modelldaten ändert und für die Aktualisierung der visuellen Darstellung (View) sorgt. Sie können die View auch als ➔ Observer (siehe Observerpattern) für das Modell realisieren. So vermeiden Sie Inkonsistenzen bei mehreren Views desselben Modells.



Vorteile

Ein Austausch der Benutzeroberfläche (auch zur Laufzeit wird unabhängig vom Modell möglich. Innerhalb der Anwendung können verschiedene Benutzerschnittstellen dasselbe Modell präsentieren. Zudem fallen Aufwandsabschätzungen für Anpassungen genauer aus, weil leicht feststellbar ist, welche der drei Teile betroffen sind. Sie können das Modell unabhängig von Controller und View testen. Bei einer Mischimplementierung müssen in der Regel alle drei Teile aufgrund zyklischer Abhängigkeiten gemeinsam getestet werden.

Nachteile

Der Implementierungsaufwand erhöht sich. Eine Stärke des Ansatzes, nämlich die Unabhängigkeit des Modells, ist ein potenzieller Nachteil bei der gleichzeitigen Darstellung (und Manipulation) derselben Daten in unterschiedlichen Views. In diesem Fall muss dafür Sorge getragen werden, dass bei Änderung der Daten alle Views konsistent bleiben. Eine komplexe Aufgabe liegt in der Festlegung des Grundgerüsts (z. B. Interfaces, Events) mit dem Anspruch, allen eventuellen Anforderungen gerecht zu werden.

Verwendung

In vielen Frameworks für Benutzerobflächen (z. 8. Java Swing) wird dieses Pattern reduziert umgesetzt. View und Controller werden dabei vereint. In herkömmlichen Web-Frontends ist es dagegen üblich, diese Funktionalitäten zu trennen. Häufig ist dort das verwandte Muster Front Controller anzutreffen (z.B. Spring). Dieser nimmt alle Requests entgegen und transformiert sie in Aufrufe auf den zuständigen Komponenten der Applikation. Ein anderer Trend ist in der voranschreitenden Ajaxifizierung zu erkennen, bei der ein Teil des Controllers in Form von JavaScript-Logik in den Browser verlagert wird. Offlinefähige Webanwendungen verlagern sogar das Modell (virtuell) auf die Clientseite, sodass im Browser eine lokale Anwendung mit hybrider Datenhaltung entsteht.

Varianten / Strategien

Passive View: Die View hält keine Referenz auf das Modell und kann sich daher auch nicht selbst aktualisieren. Der Controller überwacht neben Benutzeraktionen auch Modelländerung und ist exklusiv für Ul-Updates zuständig. Dies vereinfacht das Testen.

Quelle: Eilbrecht & Straker (2013): Patterns kompakt, Springer Verlag, S. 92ff.