Logic-Data-View-Controller-Service-Paradigma

Wechseln zu:Navigation, Suche

Dieser Artikel erfüllt die GlossarWiki-Qualitätsanforderungen **nur teilweise**:

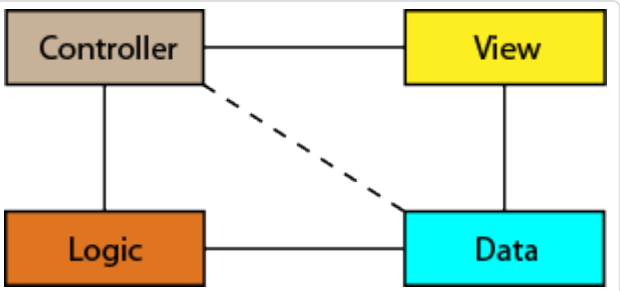
Korrektheit: 4 (großteils überprüft)	**Umfang**: 3 (einige wichtige Fakten fehlen)	**Quellenangaben**: 4 (fast vollständig vorhanden)	**Quellenarten**: 4 (sehr gut)	**Konformität**: 4 (sehr gut)

Diese Bewertungen beziehen sich auf alle im nachfolgenden Menü genannten Artikel gleichermaßen.

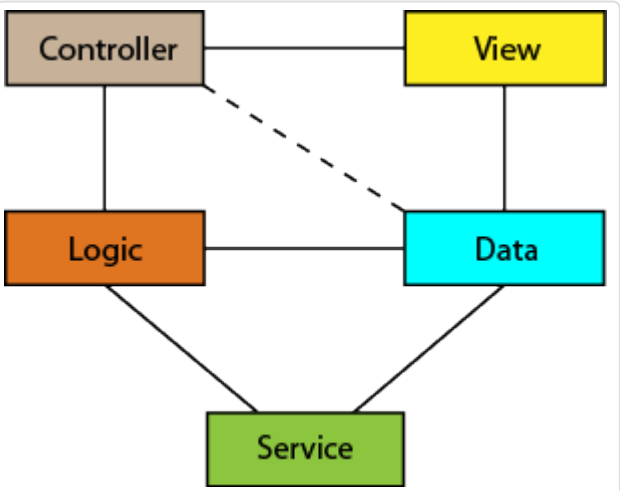
MVC-Paradigma:	Model (Data) | View | Controller	MVC-Pattern:	Singletons | Dependency Injection | Observers
MVCS-Paradigma:	Service	MVCS-Pattern:	Singletons | Dependency Injection | Observers
LDVCS-Paradigma:	Logic	VCLSD-Pattern:	Singletons | Dependency Injection | Observers

Inhaltsverzeichnis

1 Definition (Kowarschick (MMProg))



LDVC-Module



LDVCS-Module

Als Logic-Data-View-Controller-Service-Paradigma (engl. Logic-data-view-controller-service paradigm) oder -Architektur (engl. architecture), kurz LDVCS-Paradigma, LDVCS-Architektur oder auch nur LDVCS bezeichnet man ein Architekturmuster, bei der eine Anwendungs-Komponente in fünf eigenständige Module unterteilt wird: View (Darstellung, Präsentation), Controller (Steuerung), Logic (Logik), Service (Service, Zugriff auf externe Daten) und Data (Daten).

1.1 Datenmodul

Ein Datenmodul (engl. LDVCS data module) einer LDVCS-Anwendung dient zur Speicherung bestimmter Daten, d. h. zur Speicherung von Teilen des aktuellen Zustands der Anwendung.

Ein LDVCS-Datenmodul kann weitere Aufgaben übernehmen:

anderen Modulen Zugriff auf die Zustandsdaten gewähren
andere Module über Änderungen informieren (meist mittels des Observer-Patterns)

1.2 View (Darstellung, Präsentation)

LDVCS-Views (engl. LDVCS views) sind die grafischen, akustischen, haptischen und olfaktorischen Schnittstellen einer LDVCS-Anwendung. Sie „visualisieren“ Daten, die in Datenmodulen der Anwendung enthalten sind.

Eine LDVCS-View kann weitere Aufgaben übernehmen:

bei Daten-Änderungen in den zugehörigen Datenmodulen der Anwendung die View-Repräsentation dieser Daten automatisch anpassen
Benutzeraktionen, die über grafische Eingabeelemente – wie Textfelder oder Buttons – erfolgen, an einen Controller weiterleiten

1.3 Controller (Steuerung)

LDVCS-Controller (engl. LDVCS controllers) dienen zur Steuerung einer LDVCS-Anwendung. Dazu nimmt ein Controller Eingaben aus verschiedensten Quellen entgegen (z. B. Sensor-Daten oder Daten, die ein Benutzer über eine beliebige Benutzer-Schnittstelle wie eine Tastatur oder eine Maus eingibt) und leitet diese bereinigt und normalisiert an geeignete Logikmodule weiter.

Ein LDVCS-Controller sollte keine weiteren Aufgaben übernehmen.

1.4 Logikmodul

LDVCS-Logik-Module (engl. LDVCS logic modules) realisieren die Anwendungslogik einer LDVCS-Anwendung.

Ein Logikmodul reagiert auf Ereignisse, wie z. B. Benutzereingaben, Timer Events oder Ähnliches, indem es Daten in Datenmodulen entweder indirekt mit Hilfe von Servicemodulen oder direkt manipuliert.

Die Aktionen, die ein Logikmodul ausführt, hängen i. Allg. vom aktuellen Zustand der Anwendung ab. Dieser Zustand kann entweder in Datenmodulen gespeichert werden (so dass auch andere Module darauf zugreifen können) oder im Logikmodul selbst (so dass ein Zugriff durch andere Module nicht möglich ist).

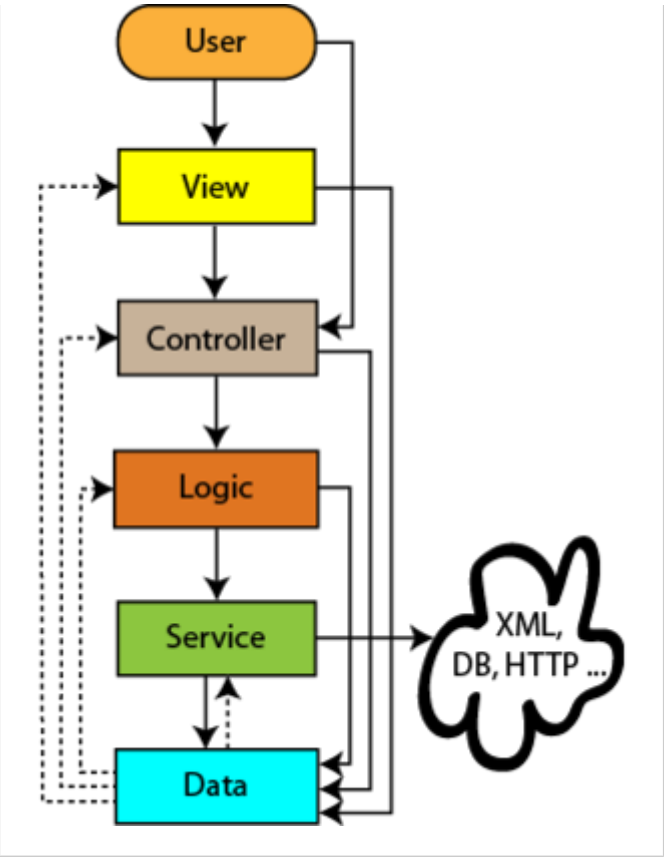
Ein LDVCS-Logik-Modul sollte keine weiteren Aufgaben übernehmen.

1.5 Service

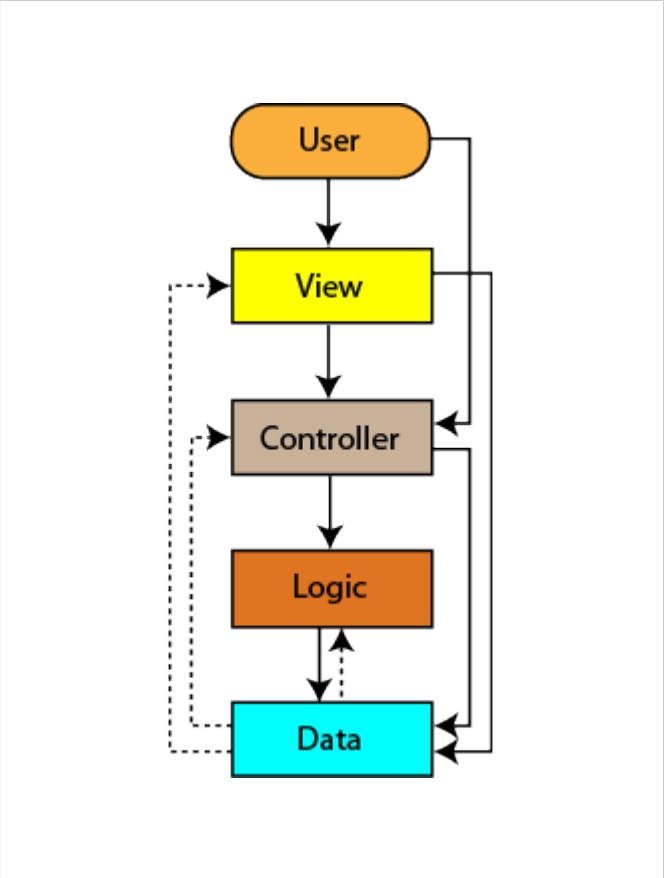
Ein Service (engl. service) dient einer LDVCS-Anwendung zur Kommunikation mit der Außenwelt, d. h. mit Dienste-Anbietern wie Web-Servern, Datenbanksystemen oder auch Dateisystemen. Die Kommunikation kann in beide Richtungen erfolgen: Services können sowohl Daten aus Datenmodulen auslesen und in externe Repositories schreiben, als auch Daten aus externen Repositories lesen und in Datenmodule einfügen.

Ein LDVCS-Service-Modul sollte keine weiteren Aufgaben übernehmen.

1.6 LDVCS-Paradigma: Varianten (Definitionen nach Kowarschick (MMProg))



Beispiel für eine VCLSD-Schichtenarchitektur



Beispiel für eine VCLD-Schichtenarchitektur

1.6.1 VCLSD-Paradigma

Ein LDVCS-Paradigma wird **VCLSD-Paradigma** genannt, wenn die fünf zugehörigen Module folgende Schichtenarchitektur bilden: *View*, *Controller*, *Logic*, *Service*, *Data*.

Die fünf Module einer VCLSD-Komponente sind gemäß dem Schichtenparadigma in Ebenen angeordnet. Die höheren Ebenen können auf tiefer gelegene Ebenen zugreifen, aber nicht umgekehrt.

Die unterste Ebene ist Data. Das zugehörige Modul weiß nichts von den über ihr liegenden Module und kann mit diesen nur mit indirekt – z. B. durch Antworten auf Nachrichten oder mit Hilfe des Observer-Patterns – kommunizieren.

Ein Servicemodul kennt nur die darunterliegende Datenmodule und kann nur auf diese zugreifen (sowie i. Allg. auf externe Datenquellen, sofern es nicht nur Filterfunktionen wahrnimmt).

Ein Logikmodul kann entweder direkt auf ein Datenmodul zugreifen oder indirekt mit Hilfe eines Servicemoduls (zum Beispiel um Daten aus einer externen Quelle zu laden, oder um zu überprüfen, ob die entsprechenden Zugriffsrechte für eine Datenmanipulation überhaupt gegeben sind).

Ein Controller-Modul kann nur auf Logikmodule zugreifen, um Benutzeraktionen an dieses (bereinigt und normalisiert) weiterzuleiten. Direkte Zugriffe auf Service- und Datenmodule wären zwar möglich, sind aber nicht vorgesehen.

Eine View kommuniziert i. Allg. direkt nur mit Controller-Modulen, um diesen Benutzeraktionen, die über die View erfolgen, mitzuteilen. Ein direkter Zugriff auf ein Datenmodul ist nur dann notwendig, wenn die Nachrichten, die von den Datenmodulen verschickt werden, nicht alle relevanten Informationen enthalten. Zugriffe auf andere Module sind nicht vorgesehen.

Der Benutzer stellt das „oberste Modul“ dar. Er kommuniziert nur mit View- und Controller-Modulen.

1.6.2 LDVCS-Multicast-Paradigma

Ein LDVCS-Paradigma wird LDVCS-Multicast-Paradigma genannt, wenn alle Module nur indirekt, d. h. mit Hilfe von Multicast-Nachrichten kommunizieren.

1.7 LDVCS-Pattern (Definitionen nach Kowarschick (MMProg))

Wenn für ein LDVCS-Paradigma – im Sinne eines Entwurfmusters – eine konkrete Klassen- und Objekt-Struktur für die fünf Module Data, Logic, View, Controller und Service vorgegeben ist, spricht man von einem LDVCS-Pattern.

1.7.1 VCLSD-Pattern

Ein LDVCS-Pattern, das ein VCLSD-Paradigma realisiert, wird auch **VCLSD-Pattern** genannt.

1.7.2 LDVCS-Multicast-Pattern

Ein LDVCS-Pattern, das nur auf Multicast-Nachrichten basiert, wird auch **LDVCS-Multicast-Pattern** genannt.

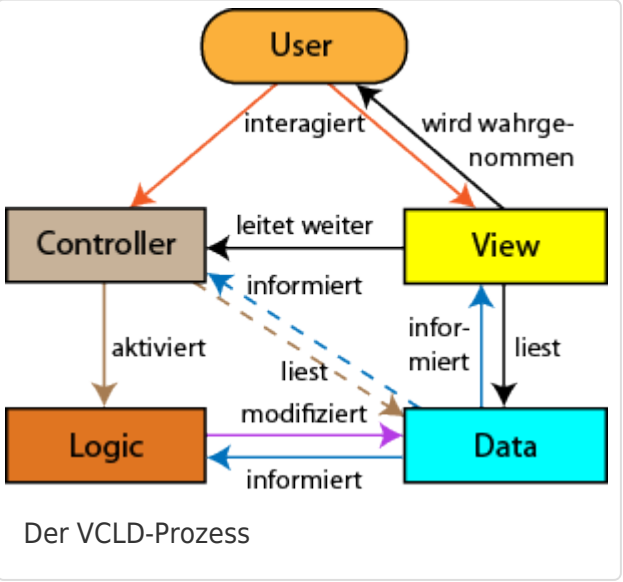
2 Das VCLSD-Paradigma als Verfeinerung des MVCS-Paradigmas

Dieses Pattern ist eine Verfeinerung des Model-View-Controller-Service-Paradigmas (MVCS-Paradigma).

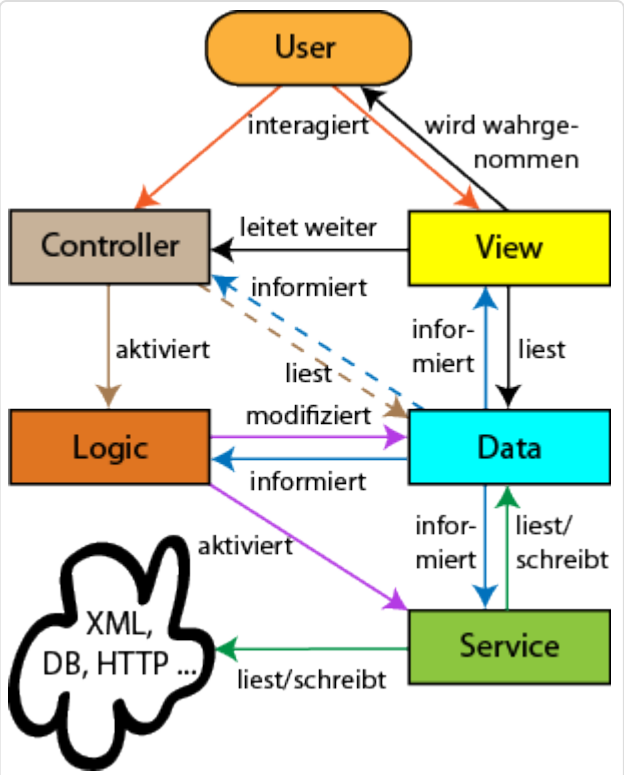
Eine VCLSD-Komponente besteht im Gegensatz zum MVCS-Komponenten nicht aus vier, sondern aus fünf Modulen. Der Grund dafür ist, dass das Modell des MVCS-Paradigmas normalerweise zwei Aufgaben übernimmt: Die Speicherung der Anwendungsdaten und die Realisierung der Anwendungslogik. Häufig übernimmt auch der Controller die Realisierung der Anwendungslogik.

Im VCLSD-Paradigma wird ein Modellmodul daher in zwei Module aufgeteilt: ein Datenmodul (Data) und ein Logikmodul (Logic). Der Controller darf nun nicht mehr direkt schreibend auf ein Datenmodul zugreifen. Dies ist jetzt die Aufgabe der Logikmodule (und evtl. der Servicemodule).

3 Der VCLSD-Prozess



Der VCLD-Prozess



Der VCLSD-Prozess

3.1 Beispiel Jump 'n' Run

In einem Jump-'n'-Run-Spiel werden Daten (**Data**) über die Spielfigur (Position, Laufrichtung, Geschwindigkeit ...), die Gegner, die Gegenstände etc. gespeichert.

Die **View** visualisiert die Elemente des Spiels mit Hilfe von Bildern und Animationen (Walk cyles etc.). Jede Änderung an den Daten hat, sofern sie sich im für den Spieler sichtbaren Bereich befindet, eine Anpassung der View zur Folge.

Der Spieler steuert die Spielfigur mit Hilfe der Tastatur. Jeder Tastendruck wird vom Controller analysiert und zur Manipulation der Spielfigur an die Logik-Komponente (**Logic**) weitergeleitet.

Man beachte, dass die Logik-Komponente i. Allg. aktiv sein kann: Sie kann die Daten selbstständig, d. h. auch ohne Manipulation durch den Controller verändern. Beispielsweise werden die gegnerischen Figuren, sofern es welche gibt, direkt von der Logik-Komponente gesteuert.

Mit Hilfe einer Service-Komponente (**Service**) kann das Spiel zu einem Multiuser-Spiel erweitert werden. Die Service-Komponente hat dann zwei Aufgaben:

1. Die Position des Avatars (d. h. der eigenen Spielfigur) an einen zentralen Server zu übertragen.
2. Die Position anderer Spielfiguren und möglicher Gegner vom zentralen Server zu holen und in die eigene Datenkomponente zu schreiben.

Eine weitere typische Aufgabe der Service-Komponente ist es, die Daten-Komponente bei Programmstart zu initialisieren. Zu diesem Zweck sollten die Service-Komponente eine Initialisierungsmethode implementieren, die vom Hauptprogramm direkt aufgerufen wird, nachdem die Daten- und die Service-Komponente erzeugt wurden.

3.1.1 Modularität

Der große Vorteil des VCLSD-Patterns ist die Modularität. Das Prinzip „Separation of Concerns“ wird hierbei beachtet: Jede Aufgabe wird von einer eigenen Komponente bearbeitet.

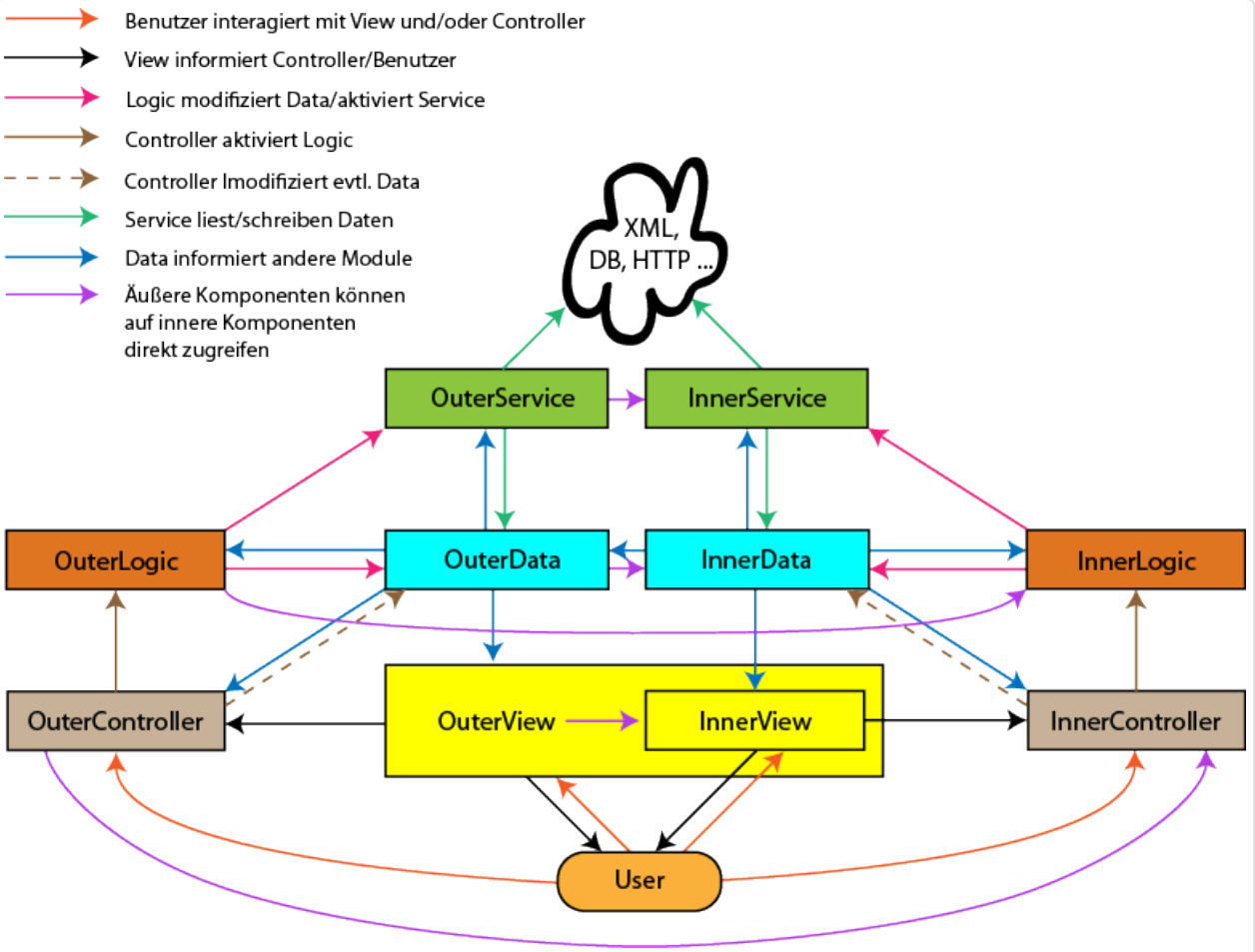
Beispielsweise kann ein Jump-'n'-Run-Spiel, das derartig modular aufgebaut ist, sehr leicht an neue Gegebenheiten angepasst werden. Jede der folgenden Änderungen hat lediglich Auswirkungen auf eines der fünf Module:

1. Änderung der Darstellung (d. h. der View): Die View kann jederzeit durch neue Skins aktualisiert werden. Es ist auch möglich, die 2D-Ansicht des Spiels durch eine Pseudo-3D-Ansicht zu ersetzen. Bei einem Multiuser-Spiel hat sowieso jeder Spieler seine eigene Sicht auf die Szenerie.
2. Änderung der Spielsteuerung (d. h. des Controllers): Anstelle einer Steuerung der Spielfigur durch die Tastatur kann beispielsweise eine Steuerung der Spielfigur durch eine WiiMote erfolgen. Hier wäre es auch denkbar, dass der WiiMote-Controller bestimmte Datenänderungen (wie z. B. Kollisionen) per Force Feedback an den Benutzer zurückmeldet. Dazu muss sich der Controller ebenfalls über Datenänderungen informieren lassen.
3. Änderung der Spiellogik (d. h. der Logikkomponente): Die KI-Engine zur Steuerung der Gegner kann beispielsweise durch eine bessere Engine ersetzt werden.
4. Änderung der Dateninitialisierung (d. h. der Service-Komponente): Wenn die Dateninitialisierung künftig beispielsweise nicht mehr über XML, sondern über SQL erfolgen soll, muss lediglich die entsprechende Service-Komponente ausgetauscht werden.
5. Änderung des Spielservers (d. h. der Service-Komponente): Es ist jederzeit möglich, ein Multi-User-Spiel über einen anderen Server (mit einer anderen Architektur) laufen zu lassen.

Auch die Datenkomponente kann ausgetauscht werden, wenn z. B. andere Datenstrukturen zur Speicherung der Daten eingesetzt werden sollen. Es ist darüber hinaus möglich, einer Datenkomponente z. B. mit Hilfe von speziellen Filter-Services Filter vorzuschalten (vgl. Aspektorientierte Programmierung). Zum Beispiel kann auf diese Weise ein Login-Mechanismus realisiert werden: Der Zugriff auf bestimmte Daten wird erst dann erlaubt, wenn ein Login-Vorgang erfolgreich abgeschlossen wurde.

4 Verfeinerung des VCLSD-Prozesses

Genauso wie der MVC-Prozess und der MVCS-Prozess kann auch der VCLSD-Prozess verfeinert werden. Auch hier kann eine „äußere VCLSD-Komponente“ eine „innere VCLSD-Komponente“ verwenden: Die Module der äußeren Komponente dürfen auf die Module der inneren Komponente zugreifen. Der umgekehrte Weg sollte allerdings vermieden werden, damit die innere Komponente problemlos in andere Umgebungen integriert werden können.



Der VCLSD-Prozess mit Trennung in eine äußere und eine innere Komponente

4.1 Beispiel „Spiel mit Highscore-Verwaltung“

Wenn man ein „Spiel mit Highscore-Verwaltung“ realisieren möchte, sollte man zwei unabhängige Komponenten implementieren: Das Spiel selbst und eine spielunabhängige Highscore-Verwaltung.

Eine dritte Komponente, die sogenannte Rahmenkomponente verknüpft diese beiden Anwendungen. Die Rahmenkomponente liest beispielsweise bei Spielende die erreichten Punkte aus dem Modell der Spielkomponente aus und leitet diese an das Highscore-Modell weiter. Eine weitere Möglichkeit wäre, dass die Rahmenkomponente über die Highscore-Anwendung ermittelt, ob der Spieler schon mindestens fünf Level erfolgreich gespielt hat. Nur in diesem Fall gewährt sie den Zugang zum Trainingsmodus des Spiels.

5 Implementierung

Das VCLSD-Pattern lässt sich auf mehrere Arten implementieren:

Initialisierung mittels Dependency Injection
Initialisierung mittels Singletons
Kommunikation ausschließlich mittels des Observer Patterns

5.1 Data Access Object

TO BE DONE

6 Quellen

1. **Kowarschick (MMProg)**: Wolfgang Kowarschick; Vorlesung „Multimedia-Programmierung“; Hochschule: Hochschule Augsburg; Adresse: Augsburg; Web-Link; 2018; Quellengüte: 3 (Vorlesung)

7 Siehe auch

Model-View-Controller-Paradigma
Model-View-Controller-Service-Paradigma

Kategorien:
MVC
Objektorientierte Programmierung
Glossar
Kapitel:Multimedia-Programmierung