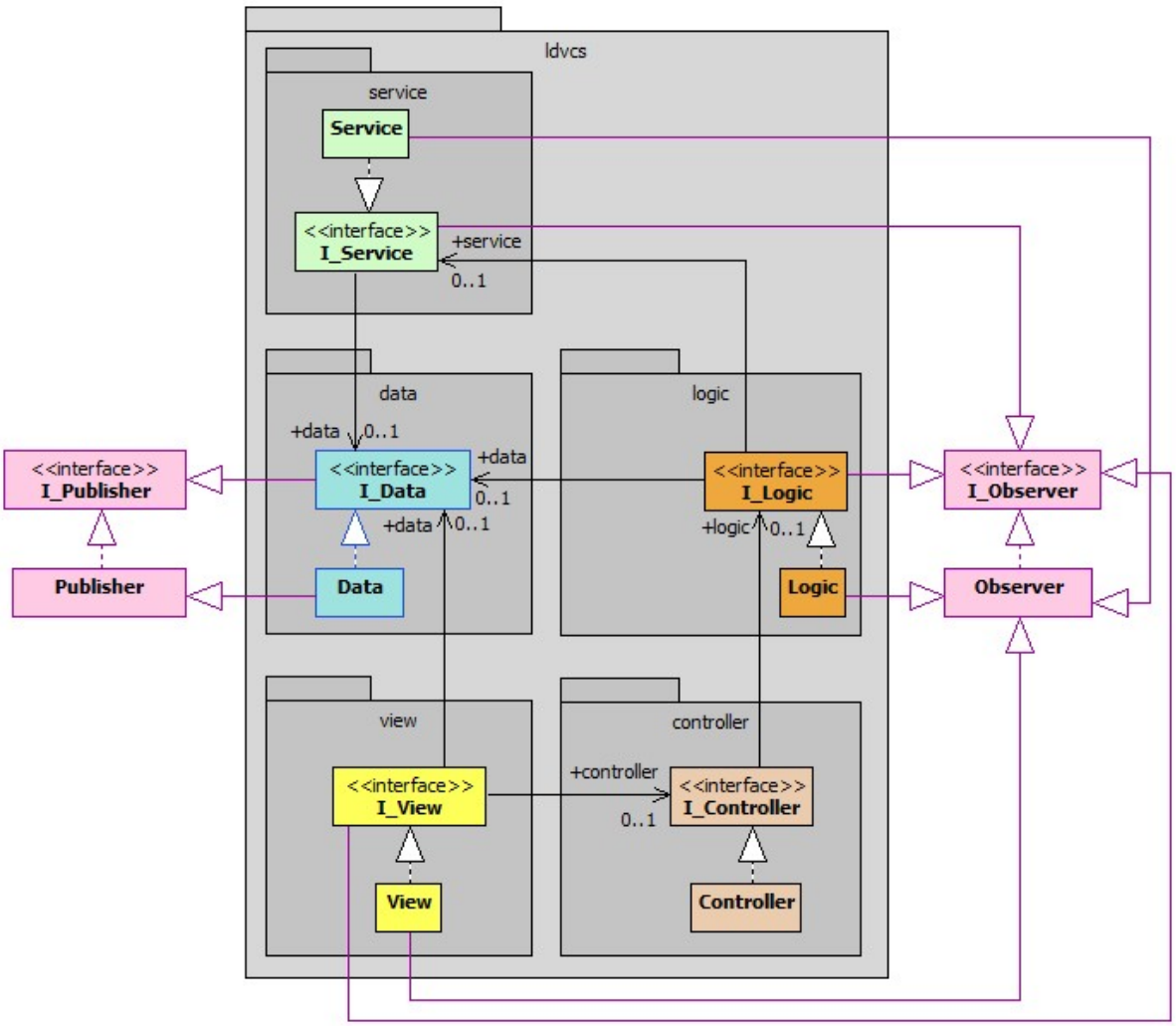
View-Controller-Logic-Service-Data-Pattern/Implementierung: Dependency Injection

Wechseln zu:Navigation, Suche

Inhaltsverzeichnis

1 Implementierung

Folgendes Klassendiagramm zeigt, wie eine Anwendung, die gemäß dem VCLSD-Pattern implementiert wird, prinzipiell aufgebaut sein kann.



VCLSD-Klassendiagramm

Für jedes der fünf Module (View, Controller, Logic, Service, Data) gibt es ein Interface, das die Methoden und Attribute deklariert, die diese Komponente unterstützt, sowie eine Klasse, die das zugehörige Interface implementiert. Durch die Verwendung von Interfaces ist es ganz einfach, eine Modulimplementierung durch eine andere zu ersetzen.

Jedes Interface mit Ausnahme des Interfaces `I_Data` defniert Beziehungen zu anderen VCLSD-Klassen. Das Datenmodul kommuniziert mit den anderen Modulen über Signale (siehe Observer-Pattern).

Ein wichtiger Aspekt der Implementierung ist daher die Initialisierung der Module. Insbersondere müssen die Module die anderen Module kennen, mit denen sie kommunizieren sollen. Dies kann auf mehrere Arten geschehen.

Zum Beipiel kann es Singleton-Klassen geben, die alle Module enthalten und über die die einzelnen Module die jeweils benötigten Kommunikatonspartner ermitteln. Dieses Vorgehen hat allerdings den Nachteil, dass die Singleton-Klassen im Prinzip globalen Variablen entsprechen: Man sieht ihnen nicht an, welche Klassen darauf zugreifen. Daher sollte man auf Singleton-Klassen besser verzichten.

Eine weitere Möglichkeit wäre, alle Module über Signale miteinander kommunizieren zu lassen. Dann müssten die Module sich gegenseitig gar nicht kennen (sofern alle wichtigen Informationen direkt in den Signalen übermittelt werden). Hier müssten bei der Initialisierung die Signaler und die Observer

geeignet miteinander vernetzt werden. Allerdings sind die Datenmodule und die Logikmodule i. Allg. eng miteinander verknüpft, da die Logikmodule sowohl lesend, als auch schreiben auf die Datenmodule zugreift. Hier ensteht durch das Observer-Pattern ein unnötiger Overhead.

Eine letzte Möglichkeit ist es, die benötigten Module per Dependency Injection den anderen Modulen bekannt zu geben.

1.1 Beispiele

Beispielsimplementierungen, die das VCLSD-Prinzip umsetzen, finden Sie in den ActionScript-Tutorien:

AS3-Tutorium: Flash: Calculator
AS3-Tutorium: Flash: Butterfly 10 vclsd

Darüber hinaus gibt es für Flex 4, Flash 11 (CS5) und Flash 10 (CS4) Rahmen-Implementierungen, die Sie als Basis für die Implementierung eigener Anwendungen verwenden können:

Umsetzung des Klassendiagramms in Flex 4	[1] (SVN-Repository)
Umsetzung des Klassendiagramms in Flash 11 (CS5)	[2] (SVN-Repository)
Umsetzung des Klassendiagramms in Flash 10 (CS4)	[3] (SVN-Repository)

In den obigen Beispielen wurden alle im obigen Diagramm definierten Klassen und Beziehungen implementiert, jedoch keine weitere Funktionalität.

Wenn Sie eine eigene Anwendung auf Basis des VCLSD-Patterns implementieren wollen, sollten Sie sich die gewünschte Rahmen-Anwendung per Subclipse in Eclipse (bzw. Flash Builder 4) laden. Heben Sie anschließend die Verknüpfung zum SVN-Repository auf:

> In Eclipse/Flash Builder: Klick mit rechter Maustaste auf das Projekt → Team → Disconnect → Also delete the SVN meta information from the file system. → Ja/Yes

Nun können Sie die Klassen und Pakete (mittels Refactoring) umbenennen. Ersetzen Sie am Besten in allen generischen Namen der Art `Domain...` den Begriff `Domain` durch den Namen Ihrer Anwendung. Und ersetzen Sie das Paket `domain` durch ein Paket mit dem (kleingeschriebenen) Namen Ihrer Anwendung.

2 Quellen

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

3 Siehe auch

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

Dieser Artikel ist GlossarWiki-konform.

Kategorien:
- MVC
- Objektorientierte Programmierung
- Glossar
- Kapitel:Multimedia-Programmierung