Hochschule für angewandte Wissenschaften
Würzburg-Schweinfurt
Fakultät Informatik und Wirtschaftsinformatik

Schwerpunktseminar

Design Pattern

Erarbeitet im Seminar der Vertiefungsrichtung Technische Informatik des Studiengangs Informatik

Juliane Aulbach

Eingereicht: Oktober 2012

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Allgemeine Informationen zu Entwurfsmustern

Entwurfsmuster haben ihren Ursprung in der Architektur, konnten sich in diesem Bereich jedoch nie durchsetzen. Ende der Neunziger Jahre haben Entwurfsmuster in der Informatik Einzug erhalten. Das bekannteste Buch, das sich mit diesem Thema beschäftigt wurde von der “Gang of Four“, oft “GoF“ abgekürzt, verfasst. Die “GoF“ besteht aus Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides und ihr Buch trägt den Titel “Design Patterns - Elements of Reusable Object-Oriented Software“. Es ist ein Standardwerk der modernen Softwareentwicklung.

1.1 Definition

Der Architekt Christopher Alexander definiert Entwurfsmuster folgender Maßen: “Jedes Muster beschreibt ein in unserer Umwelt beständig wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, so dass Sie diese Lösung beliebig oft anwenden können, ohne sie jemals ein zweites Mal gleich auszuführen.“
Ergänzend zu dieser Definition ist die Aussage des Informatiker Erich Gamma, dass ein Entwurfsmuster drei Eigenschaften erfüllen muss, zu sehen. Es muss nützlich, anwendbar und benutzt sein (Useful, Useable and Used).

1.2 Motivation

- Entwurfsmuster vereinfachen die Wiederverwendbarkeit von erfolgreichen Entwürfen und Architekturen. [GHJV11]
- Entwurfsmuster bieten Entwicklern ein Vokabular, das es ihnen ermöglicht über Entwürfe zu sprechen, sie zu dokumentieren und über Alternativen zu diskutieren. [Pöt04]
- Entwurfsmuster können die Dokumentation und Wartung existierender Systeme verbessern, weil sie die Klassen- und Objektinteraktionen explizit spezifizieren und den ihnen zugrundeliegenden Zweck erklären. [GHJV11]

1.3 Organisation der Entwurfsmuster

Damit man einen besseren Überblick über die vielen Entwurfsmuster gewinnt, hat man sie nach verschiedenen Kriterien eingeteilt. Diese Organisation vereinfacht das Erkennen von verwandte Muster, das Lernen und das Finden neuer Muster. Die Abbildung 1.1 zeigt die

Gültigkeitsbereich	Aufgabe		
	Erzeugungsmuster	**Strukturmuster**	**Verhaltensmuster**
klassenbasiert	Factory Method	Adapter (class) Bridge (class)	Template Method
objektbasiert	Abstract Factory Prototype Singleton	Adapter (object) Bridge (object) Flyweight Facade Proxy	Chain of Responsibility Command Iterator (object) Mediator Memento Observer State Strategy
kompositionsbasiert	Builder	Composite Decorator	Interpreter Iterator (compound) Visitor

Abb. 1.1: Einteilung von Entwurfsmustern [Pöt04]

klassische Einteilung. [Pöt04] Die Entwurfsmuster sind nach zwei Hauptkriterien eingeteilt, der Aufgabe eines Muster und dem Gültigkeitsbereich. Der Gültigkeitsbereich beschreibt, ob sich das Muster auf Klassen, Objekte oder zusammengesetzte Objekte bezieht. Klassenbasierte Muster beschreiben Beziehungen, die statisch festgelegt sind. Sie beziehen sich auf die Beziehungen zwischen Klassen und ihren Unterklassen. Objektbasierte Muster hingegen beschreiben dynamische Beziehungen, die zur Laufzeit geändert werden können. Kompositionsbasierte Muster beschreiben rekursive Beziehungen. [Pöt04] Das zweite Kriterium, die Aufgabe eines Entwurfsmusters, wird danach unterteilt, ob ein Muster eine erzeugende, strukturorientierte oder verhaltensorientierte Aufgaben hat.

Erzeugungsmuster beschäftigen sich mit der Aufgabe, wann und wie bestimmte Objekte erzeugt werden. Darüber hinaus verbergen sie den eigentlichen Erzeugungsprozeß nach außen hin. Sie machen ein System unabhängig davon, wie seine Objekte erzeugt, zusammengesetzt und repräsentiert werden. [KRS+00]

Strukturmuster befassen sich mit der Zusammensetzung und der Granularität von Klassen und Objekten. [KRS+00]

Verhaltensmuster befassen sich mit den Zuständigkeiten und der Zusammenarbeit zwischen Klassen bzw. Objekten. Sie beschreiben komplexe Interaktionen zwischen Objekten, die zur Laufzeit nur schwer nachzuvollziehen sind. [KRS+00]

1.4 Beschreibung von Entwurfsmustern

Es gibt viele unterschiedliche Entwurfsmuster, die sich in ihrem Abstraktionsgrad und ihrer Granularität unterscheiden. In diesem Abschnitt wird das einheitliche Schema vorgestellt, nach dem die Entwurfsmuster im folgenden beschrieben werden. Es soll helfen Entwurfsmuster zu verstehen, zu vergleichen, zu verwenden und ihre Vor- und Nachteile aufzuzeigen.

Definition Beschreibt das Entwurfsmuster in einem Satz.

Motivation Erklärt, welche Ausgangssituation sich für den Einsatz des zu beschreibenden Musters eignet.

Beschreibung / Lösung Beschreibt die Implementierung in der Theorie

Implementierung Zeigt eine konkrete Implementierung des Musters auf

Bewertung des Entwurfsmusters Stellt die Vor- und Nachteile des Musters dar.

2 Adapter

2.1 Definition

Ein Adapter passt die Schnittstelle einer Klasse an eine andere von ihren Klienten erwartete Schnittstelle an. Das Adaptermuster lässt Klassen zusammenarbeiten, die andernfalls dazu nicht in der Lage wären. [GHJV11]

2.2 Motivation

Die Motivation für die Verwendung des Adapter-Musters ergibt sich aus der Definition:

- Es soll eine Komponente verwendet werden, deren Schnittstelle nicht mit der benötigten Schnittstelle übereinstimmt. [ES10]
- Klassen mit inkompatiblen Schnittstellen sollen zusammenarbeiten. [ES10]
- Eine Komponente besitzt die richtigen Daten und das richtige Verhalten, bietet aber eine unpassende Schnittstelle an. [ES10]

2.3 Beschreibung / Lösung

Es existieren zwei verschiedene Möglichkeiten einen Adapter zu Implementieren. Zum einen als Klassenadapter und zum anderen als Objektadapter. Ein Kassenadapter, in Abbildung 2.1 dargestellt, verwendet Mehrfachvererbung, um eine Schnittstelle an eine andere anzupassen.

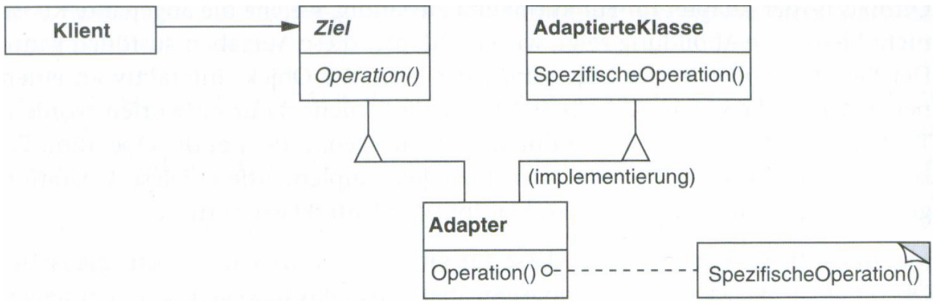

Abb. 2.1: UML-Darstellung des Klassenadapters [GHJV11]

Abbildung 2.2 zeigt einen Objektadapter. Dieser passt die Schnittstelle mit Hilfe von Objektkomposition an die erwartete Schnittstelle an.

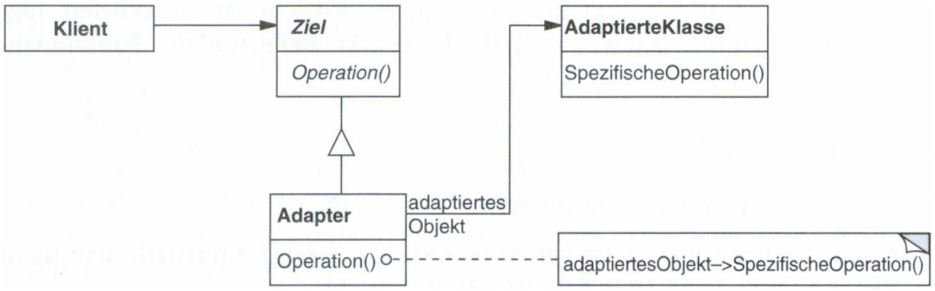

Abb. 2.2: UML-Darstellung des Objektadapters [GHJV11]

Beschreibung der in den UML-Diagrammen dargestellten Elemente:

Klient arbeitet mit Objekten, die der Zielschnittstelle entsprechen.

Ziel definiert die anwendungsspezifische vom Klienten verwendete Schnittstelle.

AdaptierteKlasse definiert eine existierende und anzupassende Schnittstelle.

Adapter passt die Schnittstelle der anzupassenden Klasse an die Zielschnittstelle an.

2.4 Implementierung

Die folgende Implementierung soll die Funktionsweise von Adapter-Klassen verdeutlichen. Um einen Klassenadapter sinnvoll einzusetzen ist eine Programmiersprache, die Sprachfeatures wie Mehrfachvererbung und private Vererbung unterstützt erforderlich. Aus diesem Grund wurde für das Beispiel die Programmiersprache C++ gewählt. Zur besseren Übersicht wurde die Deklaration der Elemente, nicht wie üblich in der Headerdatei, sondern im Programm-Quelltexte vorgenommen.
Eine Ampel soll durch ein neueres Modell ersetzt werden. Der Client konnte die zu ersetzende Ampel bisher über die Methoden setRot() und setGrün() steuern. Die neue Ampel stellt diese nicht zur Verfügung, bietet aber stattdessen eine Methode umschalten an. Nun muss ein Adapter geschrieben werden, der die Schnittstelle der neuen Ampel so anpasst, wie sie der Client erwartet.

Client

```
#include "Adapter.cpp"
#include "AmpelAlt.cpp"

class Client{
public:
        Client(){
```

```
                AmpelAlt *ampel = new Adapter();
                ampel->anschalten();
                ampel->setRot();
                ampel->setGruen();
                ampel->ausschalten();
                delete ampel;
        }
};
```

Implementiertes Interface der alten Ampel

```
#pragma once
class AmpelAlt{
public:
        void anschalten(){
        }
        void ausschalten(){
        }
        virtual void setGruen()=0;
        virtual void setRot()=0;
};
```

Die neue Ampel mit veränderten Schnittstellen

```
class AmpelNeu{
public:
        void umschalten(){
        //umschalten
        }
};
```

Der Klassenadapter

```
#include "AmpelAlt.cpp"
#include "AmpelNeu.cpp"

class Adapter:public AmpelAlt{
public:
        bool istRot;
        AmpelNeu *ampelNeuObject;

        Adapter(){
```

```
                ampelNeuObject = new AmpelNeu();
                this->istRot=false;
        }

        ~Adapter(){
                delete ampelNeuObject;
        }

        void setGruen(){
                if(this->istRot==true){
                        this->istRot=false;
                        ampelNeuObject->umschalten();
                }
        }

        void setRot(){
                if(this->istRot==false){
                        this->istRot=true;
                        ampelNeuObject->umschalten();
                }
        }
};
```

Der Objektadapter

```
#include "AmpelAlt.cpp"
#include "AmpelNeu.cpp"

class Adapter:public AmpelAlt, private AmpelNeu
{
public:
        bool istRot;

        Adapter(){
                this->istRot=false;
        }
        ~Adapter(){
        }

        void setGruen(){
                if(this->istRot==true){
                this->istRot=false;
                Adapter::umschalten();
                }
```

```
        }

        void setRot(){
                if(this->istRot==false){
                        this->istRot=true;
                        Adapter::umschalten();
                }
        }
};
```

2.5 Bewertung des Entwurfsmusters

2.5.1 Vorteile

Klassenadapter

Ermöglicht es einem Adapter Teile des Verhaltens der adaptierten Klasse zu überschreiben, weil der Adapter eine Unterklasse von ihr ist.
Verwendet lediglich ein einzelnes Objekt. Es wird keine zsätzliche Zeigerindirektion benötigt, um zum adaptierten Objekt zu gelangen. [GHJV11]

Objektadapter

Ermöglicht es einem Adapter, mit mehreren anzupassenden Klassen zu arbeiten. Neben der anzupassenden Klasse selbst kann auch auf ihre Unterklassen zugegriffen werden. Der Adapter kann zudem allen adaptierten Klassen aufeinmal neue Funktionalität hinzufügen.

2.5.2 Nachteile

Klassenadapter

Passt genau eine konkrete zu adaptierende Klasse an die Zielklasse an. Somit funktioniert ein Klassenadapter genau dann nicht, wenn eine konkrete Klasse mit all ihren Unterklassen angepasst werden soll. [GHJV11]

Objektadapter

Macht es schwerer, das Verhalten der anzupassenden Klasse zu überschreiben. Man muss eine Unterklasse der zu adaptierenden Klasse bilden und im Adapter mit dieser Unterklasse statt mit der ursprünglich anzupassenden Klasse arbeiten. [GHJV11]

Literaturverzeichnis

[ES10] EILEBRECHT, KARL GERNOT STARKE: *Patterns kompakt - Entwurfsmuster für effektive Software-Entwicklung*. Springer DE, Berlin, 3. Aufl. 2010 , 2010.

[GHJV11] GAMMA, ERICH, RICHARD HELM, RALPH JOHNSON JOHN VLISSIDES: *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*. Pearson Deutschland GmbH, München, 6. Auflage , 2011.

[KRS+00] KÜBLER, RIECK, STANULLO, VOLLMER WEISS: *Ferienkurs Design Patterns*. PDF, 2000. http://www.stz-softwaretechnik.de/index.php?id=630&tx_ttnews%5Bpointer%5D=2&cHash=196c4106a3.

[Pöt04] PÖTZSCH, SEBASTIAN: *Entwurfsmuster*. PDF, 2004. http://ebus.informatik.uni-leipzig.de/www/media/lehre/seminar-pioniere04/poetzsch-ausarbeitung-gamma.pdf.