Design Pattern
Observer

Markus Schnalke
MatNr: 039131

Dies ist meine Ausarbeitung zum Design Pattern "Observer" im Rahmen der Vorlesung Softwaretechnik im Studiengang Wirtschaftsinformatik an der Hochschule Ulm.

Inhaltsverzeichnis

1. Einleitung **3**

2. Meine Vorgehensweise **3**

3. Erarbeitung des Patterns **4**
3.1. Ausgangssituation 4
3.2. Mögliche Ansätze 4
3.3. Erarbeitung einer Lösung 5
3.3.1. Eine (gute) Lösung: Pinnwand 5
3.3.2. Verbesserung: Pinnwand-Sekretärin 5
3.3.3. 2. Verbesserung: Pinnwand-Sekretärin mit Benachrichtigung 6
3.4. Zusammenfassung des Beispiels 6

4. Das Pattern **7**
4.1. Überleitung 7
4.2. UML-Diagramme 7
4.2.1. Klassifizierung nach GoF 8
4.3. Beispiele für den Observer in der Praxis 8
4.4. Erweiterungen des Patterns 9
4.4.1. Ein Observer und mehrere Subjects 9
4.4.2. Nur für bestimmte Informationen anmelden 9
4.4.3. ChangeManager 9
4.4.4. Wer ruft notify() auf? 9

5. Zusammenfassung **10**

A. Beispiel-Implementierung **11**

B. Verwendete Software **11**

1. Einleitung

Ich möchte im Folgenden einen Ansatz zum Verstehen von Design Patterns aufzeigen. Dazu nehme ich das Design Pattern "Observer", das ich Stück für Stück aus einer Situation der realen Welt herleiten werde. Es geht mir dabei nicht primär darum euch dieses Pattern zu erklären, vielmehr soll das Erfassen des Zusammenhangs zwischen Realität und Design Patterns gefördert werden.

Mein Ziel ist es die *Natürlichkeit* von Design Patterns darzustellen — weil sie dem Vorgehen in der Realität entsprechen!

2. Meine Vorgehensweise

> Design Pattern sind "Best Practices" (= Erfolgsrezepte), meist **nach Vorbildern in der Realität**.

Aus diesem Grund möchte ich nun anhand eines Beispiels aus dem täglichen Leben zeigen, dass das Design Pattern **Observer** ein absolut natürliches Vorgehen ist, das bei ähnlichen Situationen in Programmen deshalb ebenso verwendet werden sollte. Dass ich hier gerade das Pattern **Observer** verwende, hat keinen besonderen Grund; es kann wohl (fast) jedes Pattern auf diese Weise verständlich und logisch gemacht werden.

3. Erarbeitung des Patterns

3.1. Ausgangssituation

Meine Ausgangssituation von der ich mich zu einer möglichst optimalen Lösung vorarbeiten möchte ist Folgende:

- Es gibt Personen die Etwas verkaufen möchten
- Es gibt Personen die Etwas kaufen möchten
- Sie wollen/sollen sich nicht kennen

Gesucht ist natürlich ein möglichst effektives Vorgehen. Eben das ist der Grund, weshalb wir Patterns verwenden möchten: Wir suchen eine effektive Standardlösung für regelmäßig auftretende Problemstellungen.

3.2. Mögliche Ansätze

Um systematisch vorzugehen, überlegen wir uns zunächst, welche primitiven Lösungen für unser Problem in Frage kommen. Dies wären zum Beispiel:

- Personen direkt ansprechen
- Laut in die Menge rufen
- Den Freunden erzählen, die es dann weitererzählen

Vermutlich wäre der Eine oder Andere (unbewusst) gleich höher eingestiegen, dennoch zeigen auch (oder gerade) diese simplen Vorgehensweisen Probleme auf, die sonst kaum explizit wahrgenommen werden.

Dies sind unter anderem:

- Man weiß nicht wer Interesse hat
- nicht mal wieviele
- Manche Interessenten sind vielleicht nur zu bestimmten Zeiten da
- (... oder sie sind taub)

3.3. Erarbeitung einer Lösung

Im Folgenden möchte ich nun eine Lösung Schritt für Schritt erarbeiten und verbessern, bis sie das unsere Situation auf eine gute Weise löst.

Meine Lösung, die ich nun präsentieren möchte, ist eine Pinnwand. Pinnwände werden in der Realität normalerweise verwendet um derartige Problem zu lösen — kein Wunder, wie wir gleich sehen werden. Die Pinnwand bietet von sich aus schon eine gute Lösung für unsere Problemsituation.

3.3.1. Eine (gute) Lösung: Pinnwand

Die Funktionen die eine Pinnwand anbietet sind:

- Man kann Zettel anpinnen
- Hingehen und nach neuen Zetteln schauen
- Zettel lesen
- Zettel abnehmen

Nun sind zwar einige unserer Probleme (wie z.B. dass sich die Personen nicht kennen müssen) gelöst, doch es gibt auch welche die weiterhin bestehen. Dies sind vor allem:

- Man muss hingehen, nur um festzustellen, dass nichts Neues dabei ist
- Man kann wichtige Zettel verpassen
- Zettel sollten nicht weggenommen werden können

Diese Unzulänglichkeiten der jetzigen Lösung gilt es nun Schritt für Schritt zu eliminieren.

3.3.2. Verbesserung: Pinnwand-Sekretärin

Mit diesem Semester wurden bei uns Studiengebühren eingeführt. Die häufigen Diskussionen deswegen waren es wohl, die mich auf die Idee gebracht haben, das Geld doch sinnvoll(er) zu investieren. Und so erweitern wir unsere Pinnwand um eine Sekretärin die die Pinnwand verwaltet ... natürlich mit Studiengebühren finanziert ;-)

Nachfolgend möchten wir die Sekretärin und die Pinnwand als Einheit betrachten. Die neuen Features dieser Pinnwand-Sekretärin-Einheit sind folgende:

- Man kann bei ihr einen Zettel in Auftrag geben (auch telefonisch)

- (Sie schreibt mit lesbarer Schrift)
- Sie verhindert, dass Zettel abgenommen werden

...und wir sind der optimalen Lösung unserer Problemsituation wieder einen Schritt näher. Jedoch nur einen Schritt, denn nicht alle Probleme sind gelöst. Bestehen bleibt, dass man immer noch oft unnötig zur Pinnwand läuft.

3.3.3. 2. Verbesserung: Pinnwand-Sekretärin mit Benachrichtigung

Die Studiengebühren sollen uns an dieser Stelle noch nicht ausgedient haben —immerhin sind es 500 Euro— und so ist noch genug übrig um userer Sekretärin verlängerte Arbeitszeiten finanzieren zu können. In dieser zusätzlichen Zeit kann sie nun natürlich weitere Aufgaben übernehmen. Dies sind:

- Man kann sich bei der Sekretärin nun als "Interessierter" registrieren
- Die Sekretärin trägt die Telefonnummer in eine Liste ein
- Zukünftig ruft sie alle Personen der Liste an, wenn sie einen neuen Zettel anpinnt

3.4. Zusammenfassung des Beispiels

Wir haben nun eine Lösung die die meisten Probleme unserer Situation löst. Ich möchte hier die Funktionsweise nochmals aufzählen:

- Man kann neue Zettel anpinnen lassen
- Man kann sich als Interessierter anmelden (und auch abmelden)
- Interessierte werden bei Änderungen der Pinnwand benachrichtigt
- Sie können dann zur Pinnwand gehen und sie sich anschauen

Sind jetzt alle Anforderungen abgedeckt? Ist die geschaffene Struktur zufriedenstellend? Welche Wünsche sind noch offen? Was fehlt?

Es gibt natürlich weitere Anforderungen/Wünsche die über das jetzige Modell hinausgehen. Auf einige der verbreitendsten Erweiterungen des Observer-Modells werde ich weiter unten noch eingehen.

4. Das Pattern

Nun haben wir uns eine Lösung für unser Problem erarbeitet und der nächste Schritt ist es ein allgemein gültiges Lösungsmodell zu erstellen. Ein solches Modell wird “Pattern” genannt.

4.1. Überleitung

Um unsere Lösung in das Pattern zu überführen bedarf es ein paar anderer Bezeichnungen:

- Pinnwand-Sekretärin-Einheit → “Subject”
- Die Zettel auf der Pinnwand → “subjectState”
- Interessenten → “Observers”

Beim Programmieren sind besonders Interfaces (also Schnittstellen) wichtig. Diese entsprechen den Fähigkeiten die Pinnwand/Sekretärin und Interessenten haben oder anbieten. Dies wären zum Beispiel, dass Interessenten nicht taub sein dürfen, lesen und zur Pinnwand hingehen können müssen. Oder auch die leserliche Schrift der Sekretärin. (Siehe dazu auch die Erarbeitung der Pinnwand-Sekretärin.)

4.2. UML-Diagramme

Um das Pattern darzustellen bieten sich UML-Diagramme an.

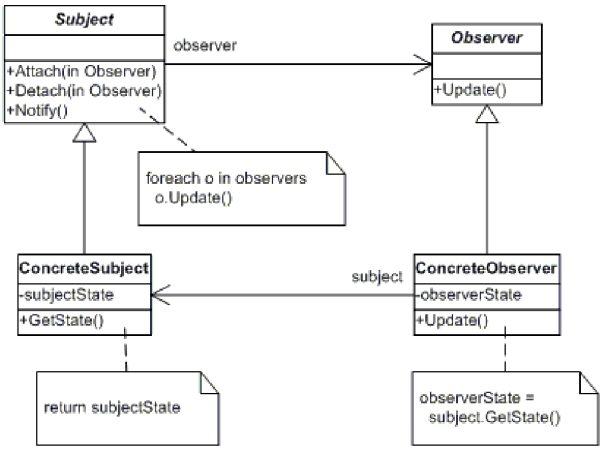

Abbildung 1: Struktur-Diagramm des Observers

Wer den ersten Teil der Ausarbeitung verstanden hat und UML kann, sollte hier keine Probleme haben die Diagramme zu verstehen — es ist quasi das Gleiche, nur in einer anderen Darstellungsform.

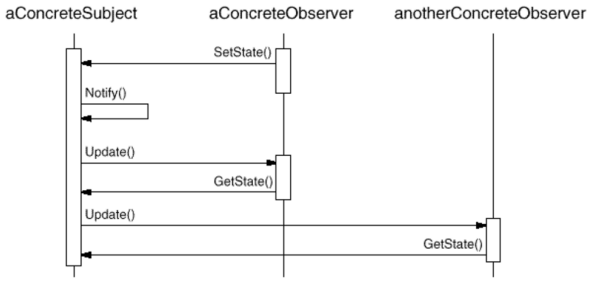

Abbildung 2: Interaktions-Diagramm des Observers

4.2.1. Klassifizierung nach GoF

Die "Gang of Four" (sie formulierten die ersten Design-Patterns für die Informatik) habe ein einheitliches Schema zu ihrer Klassifizierung erstellt. Anhand diesem ist der Observer folgendermaßen einzuordnen:

Klassifizierung: Verhaltensmuster, objektbasierend

Auch bekannt als: Publish-Subscribe, Dependents

Zweck: Abhängigkeiten zwischen Objekten erstellen, sodass sich abhängige Objekte ändern, wenn sich das Objekt selbst ändert.

Kurzbeschreibung: Schnittstellen anlegen, damit Abhängigkeiten zwischen Objekten registriert werden können, und um die abhängigen Objekte über Zustandsänderungen zu informieren.

4.3. Beispiele für den Observer in der Praxis

Wo ist das Observer-Pattern nun im täglichen Leben anzutreffen? Natürlich ist hier das Leben in der digitalen Welt gemeint, schließlich geht es uns ja um ein Design-Pattern für die Programmierung.

Zuerst einmal ist anzuführen, dass der Observer ein sehr verbreitetes Design-Pattern ist, das recht häufig bei passenden Problemstellungen eingesetzt wird.

Primär wäre das alles was mit Model-View-Controller (kurz: MVC) zusammenhängt. MVC wird vor allem für grafische Oberflächen eingesetzt. Dabei fungiert das Model als Subject und die View als Observer. Der Controller ist eher von untergeordneter Bedeutung. (MVC ist übrigens ein Architektur-Pattern.)

Aber auch Mailinglisten und Such-Abos (wie bei Ebay) sind optimalerweise nach dem Observer-Pattern implementiert.

Kein Beispiel für das Observer-Pattern ist aber der Weblog mit RSS-Feed! Denn hier findet kein Abonnement-Vorgang statt, der Client (Observer) meldet sich nicht bei der Website (Subject) an, und bekommt auch keine Änderungsinformationen zugeschickt. Stattdessen ruft der Client nur Informationen ab, die die Website ständig zur Verfügung stellt. (vgl. Pinnwand ohne Sekretärin)

4.4. Erweiterungen des Patterns

4.4.1. Ein Observer und mehrere Subjects

Oft ist es nicht nur ein einziges Subject, das beobachtet werden soll. Damit ein Observer mehrere Subjects beobachten kann, muss er den Namen des Subjects mitsenden. So kann festgestellt werden welches Subject betroffen ist.

4.4.2. Nur für bestimmte Informationen anmelden

Eine weitere kleine Erweiterung ist die Anmeldung am Subject für nur bestimmte Informationen. Dies ist sicher auch eine Ergänzung die unsere Pinnwand verbessert hätte. So wäre es dann möglich gewesen sich nur für Zimmerangebote oder ähnliches anzumelden. Auf diese Weise werden auch die unnötigen Updates verringert, was sich positiv auf die Performance auswirken kann.

4.4.3. ChangeManager

Bei komplexen Update-Zusammenhängen ist es empfehlenswert einen ChangeManager zwischen die verschiedenen Subjects und Observers zu stellen. Dieser vermittelt dann unter den Beteiligten und koordiniert die Update-Vorgänge. Der ChangeManager ist eine Instanz vom Mediator-Pattern und üblicherweise Singleton, da normalerweise nur ein ChangeManager für alle Observer-Beziehungen verwendet wird.

4.4.4. Wer ruft notify() auf?

Dies kommt stark auf das Programm an. Beide Alternativen haben ihre Vor- und Nachteile.

Das Subject: Auf diese Weise wird notify() sicher bei jedem setState() aufgerufen, jedoch können die Update-Kosten bei vielen Änderungen in kurzer Zeit sehr hoch werden.

Der Observer: Hier darf nun der Observer den Aufruf von notify() nicht vergessen, dieser kann jedoch zu einem günstigen Zeitpunkt erfolgen, was sich bei Änderungen en-block positiv auswirkt.

5. Zusammenfassung

Ich habe in meiner Ausarbeitung bisher ganz bewusst auf Quellcode verzichtet, denn ich wollte Design Patterns einmal von der anderen Seite her erklären. Ich wollte vermitteln weshalb das Observer-Pattern so aufgebaut ist wie es ist. Ich wollte Verständnis für Design Patterns entwickeln und zeigen, dass sie absolut logische Lösungen sind.

In drei Sätzen:

- Menschen denken basierend auf der Realität
- deshalb Design Patterns auf die Realität zurückführen
- Patterns anwenden weil man es in der Realität auch so machen würde

Design Patterns sind dabei Modelle wie Quellcode aufgebaut werden sollte. Sie sind kein Code — sie beschreiben nur wie Code sein sollte. Das ist auch ganz gut so, denn:

> **Implementierungen sind Schall und Rauch,**
> **Konzepte dagegen bleiben bestehen!**

Design Patterns sind Konzepte — Programmiersprachen kommen und gehen, Design Patterns überleben. Wenn man also in die Zukunft investieren möchte, dann sollte man sich Design Patterns aneignen, denn diese Investition ist risikofrei und zudem hoch rentabel!

Ich wollte euch die Natürlichkeit von Patterns nahebringen und euch dafür begeistern. Ich hoffe das ist mir gelungen :-)

markus schnalke

A. Beispiel-Implementierung

Ich möchte mich mit Quelltext auf dieses Beispiel im Anhang beschränken. Dennoch finde ich es wichtig, zumindest eine Beispiel-Implementierung vorzeigen zu können, da Quellcode sehr aussagekräftig sein kann. In jeden Fall wird er meine sonstigen Ausführungen gut abrunden. Hier also eine Realisierung in Java:

```
import java.util.Observable;
import java.util.Observer;

public class MessageBoard extends Observable {
  private String message;

  public String getMessage() {
    return message;
  }

  public void changeMessage(String message) {
    this.message = message;
    setChanged();
    notifyObservers(message);
  }

  public static void main(String[] args) {
    MessageBoard board = new MessageBoard();
    Student bob = new Student();
    Student joe = new Student();
    board.addObserver(bob);
    board.addObserver(joe);
    board.changeMessage("More Homework!");
  }
}

class Student implements Observer {
  public void update(Observable o, Object arg) {
    System.out.println("Message board changed: " + arg);
  }
}
```

Quellcode von http://java2s.com

B. Verwendete Software

- Debian GNU/Linux
- `pdflatex`
- Vim
- qiv und ImageMagick
- Mercurial