4.13 Das Verhaltensmuster Vermittler

Das Verhaltensmuster Vermittler oder Mediator darf nicht mit dem Architekturmuster Broker verwechselt werden. Ein Broker stellt umgangssprachlich auch einen Vermittler dar, aber keinen Vermittler im Sinne des Vermittler-Musters.

4.13.1 Name/Alternative Namen

Vermittler, Mediator (engl. mediator).

4.13.2 Problem

Ein zentraler Vermittler soll es erlauben, das Zusammenspiel zwischen vielen Objekten im Vermittler zu kapseln und zu steuern. Objekte sollen sich wechselseitig nicht mehr kennen, sondern nur noch den Vermittler. Der Vermittler soll das gewünschte Gesamtverhalten durch die Benachrichtigung der Kollegen erzeugen. Das Zusammenspiel der Objekte soll an zentraler Stelle im Vermittler abgeändert werden können, damit die einzelnen Objekte voneinander entkoppelt werden. Damit soll die Wiederverwendbarkeit der Objekte erhöht und außerdem das System übersichtlicher werden.

4.13.3 Lösung

Der Vermittler ist ein objektbasiertes Verhaltensmuster. Bei Änderungen eines Objekts benachrichtigt dieses den Vermittler. Der Vermittler wiederum benachrichtigt die anderen Objekte über die erfolgte Änderung. Zwischen einem aufrufenden Objekt und den anderen Objekten wird also ein Vermittler eingeführt, der auf Grund von eingehenden Nachrichten andere, davon betroffene Objekte benachrichtigt.

Die n-zu-m-Beziehung (vermaschtes Netz) zwischen den Objekten wird auf eine 1-zu-n-Beziehung (Sterntopologie) zwischen Vermittler und Objekten reduziert. Somit kann jedes Objekt mit jedem anderen in indirekter Art und Weise über einen Vermittler reden. Dadurch sind die Objekte nicht mehr wechselseitig voneinander abhängig, allerdings sind sie stark von ihrem Vermittler abhängig. Änderungen bei der Zustellung der Nachrichten erfolgen im Vermittler.

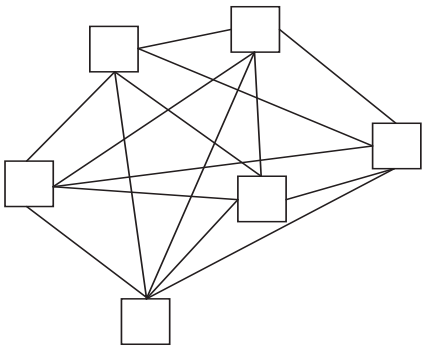


Bild 4-38 Vermaschtes Netz

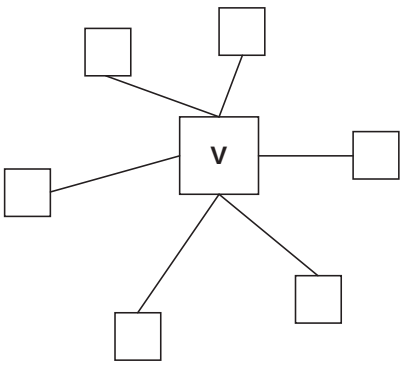


Bild 4-39 Sterntopologie

Beim Verhaltensmuster Vermittler kommunizieren Objekte nicht mehr direkt miteinander, sondern indirekt über einen Vermittler. Dieser informiert die von einer eingetroffenen Nachricht betroffenen Kollegen. Die wechselseitige Abhängigkeit der Objekte untereinander wird reduziert, die Objekte sind dafür vom Vermittler abhängig.

4.13.3.1 Klassendiagramm

Die miteinander kommunizierenden Objekte werden als **Kollegen** bezeichnet. Das folgende Klassendiagramm des Vermittler-Musters verwendet für die Abstraktion eines Kollegen eine abstrakte Klasse. Diese abstrakte Klasse kann den Code, der zwischen den konkreten Kollegen geteilt wird und der beispielhaft im folgenden Bild durch die Methode `aenderung()` angedeutet ist, enthalten. Aber prinzipiell kann für die Abstraktion eines Kollegen sowohl eine abstrakte Klasse als auch ein Interface verwendet werden. Das Gleiche gilt auch auf der Vermittlerseite: Auch die Schnittstelle eines Vermittlers kann ebenso in einem Interface definiert werden anstatt in einer abstrakten Klasse.

Wie in Bild 4-40 zu sehen ist, hat die abstrakte Klasse `Kollege` eine Assoziation zu einem abstrakten Vermittler. Diese Assoziation erben alle konkreten Kollegen. Jeder Kollege soll also den Vermittler kennen.

Im Fall der Veränderung eines konkreten Kollegen-Objektes wird der assoziierte konkrete Vermittler benachrichtigt. Dieser informiert dann alle betroffenen weiteren konkreten Kollegen. Ein konkreter Vermittler muss also wissen, welche konkreten Kollegen er benachrichtigen soll. Das Klassendiagramm in Bild 4-40 zeigt daher eine gerichtete Assoziation zwischen der Klasse `KonkreterVermittler` und den Klassen `KonkreterKollegeX (X = A..Z)`.

Damit die Kommunikation zwischen Vermittler und Kollegen reibungslos funktioniert, muss in beiden Klassenhierarchien das liskovsche Substitutionsprinzip eingehalten werden. Hier das Klassendiagramm:

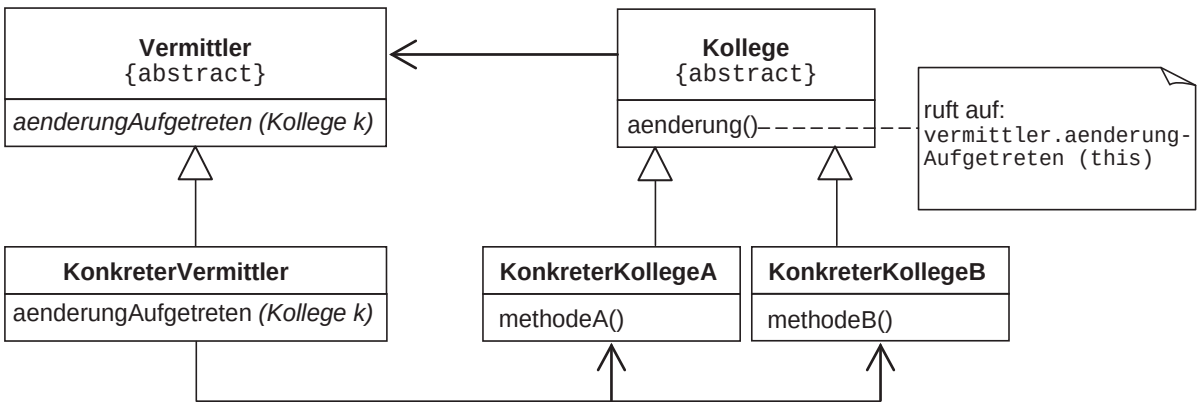


Bild 4-40 Klassendiagramm des Vermittler-Musters

Die Kommunikation aller konkreten Kollegen untereinander erfolgt nur zentral über den entsprechenden konkreten Vermittler.

Muss die Kommunikation zwischen den konkreten Kollegen abgeändert werden, weil beispielsweise Kollegen in eine andere Abteilung versetzt werden, muss nur der konkrete Vermittler geändert bzw. die Referenz eines Kollegen auf einen anderen konkreten Vermittler gesetzt werden. Ohne die Verwendung eines Vermittlers müsste man entweder die Kollegen-Klassen selbst abändern oder sie durch Unterklassenbildung an die neue Kommunikationsstruktur anpassen. Durch das Vermittler-Muster bleiben also die Kollegen-Klassen – bzw. deren Objekte – in unterschiedlichen Kommunikationsstrukturen unabhängig voneinander und damit wiederverwendbar.

4.13.3.2 Teilnehmer

Vermittler
Die Klasse `Vermittler` ist abstrakt und definiert die Schnittstelle, über welche die Kollegen-Objekte Änderungen zur Weiterleitung an andere Kollegen einem Vermittler mitteilen können.

KonkreterVermittler
Ein Objekt der Klasse `KonkreterVermittler` realisiert einen Vermittler. Der konkrete Vermittler implementiert die Kommunikationsstruktur zu den konkreten Kollegen. So besitzt er beispielsweise eine Referenz auf alle Kollegen und ruft die entsprechenden Methoden der anderen gewünschten Kollegen auf, wenn er von einem der Kollegen benachrichtigt wird. Es kann mehrere Klassen für konkrete Vermittler geben, die an die jeweils unterschiedlichen Gegebenheiten der Kollegen-Objekte und deren Zusammenspiel angepasst sind.

Kollege
Die Klasse `Kollege` ist eine abstrakte Basisklasse oder auch ein Interface für alle konkreten Kollegen. Ein Kollege hält eine Referenz auf einen konkreten Vermittler. Er informiert den Vermittler, wenn bei ihm eine Änderung eingetreten ist. Ein Kollege arbeitet mit seinem Vermittler zusammen. Er spricht nicht direkt mit den anderen Kollegen, sondern nur indirekt über den Vermittler.

KonkreterKollegeX
Eine Klasse `KonkreterKollegeX (X = A..Z)` leitet von der Klasse `Kollege` ab und definiert, wann eine Änderung eingetreten ist. Die Klasse verfügt über eine Methode `methodeX()`, die der Vermittler bei einer Aktualisierung aufruft, um einen Kollegen über eine erfolgte Änderung zu informieren.

4.13.3.3 Dynamisches Verhalten

Bild 4-41 zeigt nun das dynamische Verhalten der Beteiligten am Vermittler-Muster:

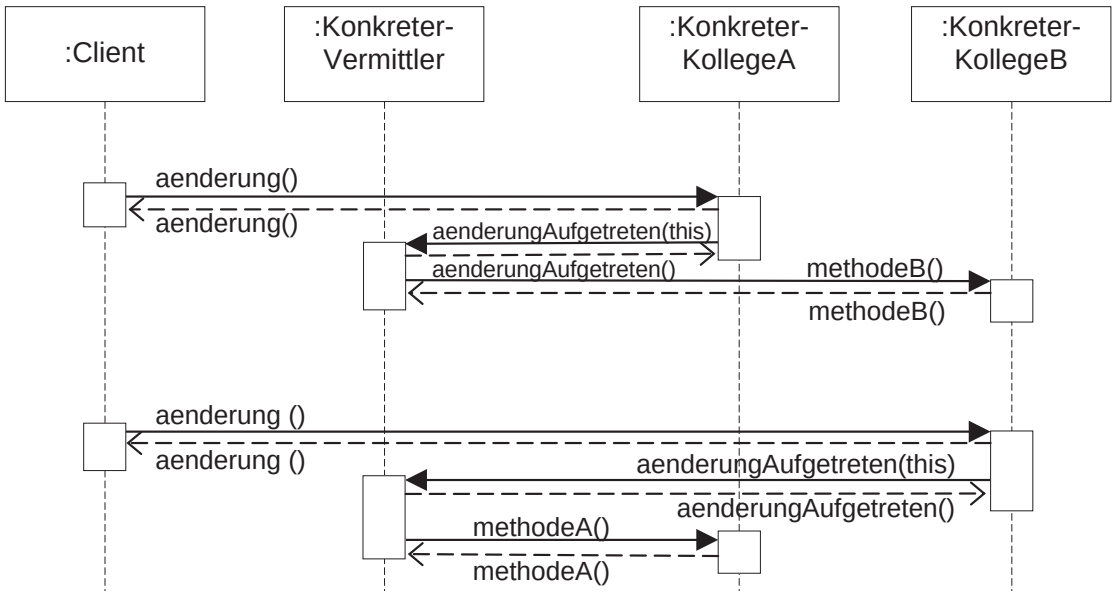


Bild 4-41 Sequenzdiagramm des Vermittler-Musters

In Bild 4-41 führt ein Client eine Veränderung am Objekt der Klasse `KonkreterKollegeA` durch. Danach teilt dieses Objekt seinem konkreten Vermittler mit, dass eine Änderung eingetreten ist. Das Objekt der Klasse `KonkreterVermittler` nimmt diesen Methodenaufruf entgegen und informiert dann die weiteren Kollegen über die Veränderung, in diesem Falle das Objekt der Klasse `KonkreterKollegeB`.

Ferner wird im Sequenzdiagramm gezeigt, dass, wenn der Client das Objekt der Klasse `KonkreterKollegeB` ändert, dieses wiederum den Vermittler informiert und dieser dann die Methode `methodeA()` des Objekts der Klasse `KonkreterKollegeA` aufruft, sodass dieser Kollege Kenntnis über die Änderung erhält.

Es sei hier noch erwähnt, dass das Client-Objekt, das in Bild 4-41 dargestellt ist, nicht Bestandteil des Musters ist. Es steht hier stellvertretend für Objekte, die eine Änderung bei einem der Kollegen-Objekte bewirken und dadurch eine Benachrichtigung anderer Kollegen über den Vermittler auslösen.

Weitergehende Lösungsvarianten

Die bisherige Beschreibung des Vermittler-Musters basierte auf einer sehr einfachen Form dieses Musters. Damit sollte das Prinzip des Musters zuerst einmal verständlich erklärt werden. Bei einer genaueren Betrachtung des Klassendiagramms in Bild 4-39 fällt aber ins Auge, dass Vermittler und Kollegen stark gekoppelt sind. Einmal gibt es eine Abhängigkeit auf der abstrakten Ebene und zum anderen gibt es eine umgekehrt gerichtete Abhängigkeit auf der konkreten Ebene. Hier ist der Vermittler sogar von jeder konkreten Kollegen-Klasse abhängig.

Weitere Punkte, die von der bisherigen Beschreibung nicht abgedeckt wurden, sind:

- Woher kennt der Vermittler die zu benachrichtigenden Kollegen?
- Woher weiß der Vermittler, welcher Kollege bei welchem Ereignis zu benachrichtigen ist?

- Warum müssen die Kollegen von einer gemeinsamen Basisklasse ableiten? Können die Objekte, die zu benachrichtigen sind, nicht von gänzlich unterschiedlichem Typ sein?

Diese Fragen müssen in der Regel anwendungsspezifisch beantwortet werden und sind nur schlecht durch eine allgemeine Herangehensweise im Rahmen des Vermittler-Musters zu lösen. In [Gr102] wird eine Lösung am Beispiel eines Vermittlers zwischen grafischen Objekten einer kleinen Anwendung gezeigt, bei der die oben genannten Punkte exemplarisch berücksichtigt sind.

4.13.3.4 Programmbeispiel

Es handelt sich im Folgenden um ein sehr einfaches Beispiel: der konkrete Vermittler kennt nur den Nachrichtenaustausch zwischen zwei Objekten – einem Objekt der Klasse `KonkreterKollegeA` und einem Objekt der Klasse `KonkreterKollegeB`. Selbst an diesem einfachen Beispiel ist aber bereits der Effekt des Vermittler-Musters zusehen, nämlich dass die Klassen `KonkreterKollegeA` und `KonkreterKollegeB` nicht voneinander abhängig sind, obwohl ihre Objekte miteinander kommunizieren. Sie sind nur von dem Vermittler abhängig, über den sie miteinander kommunizieren. Das Beispiel orientiert sich an der einfachen Lösungsvariante im Bild 4-39.

Die abstrakte Klasse `Kollege` ist die Basisklasse für alle Kollegen:

```
// Datei: Kollege.java
public abstract class Kollege
{
   // Instanzvariable
   private Vermittler vermittler; // Referenz auf den Vermittler

   // Konstruktor
   public Kollege (Vermittler v)
   {
      vermittler = v;
   }

   // Wird von den ableitenden Klassen ueberschrieben
   public void aenderung()
   {
      vermittler.aenderungAufgetreten (this); // Vermittler informiert
   }
}
```

Die Klasse `KonkreterKollegeA` ist von der Klasse `Kollege` abgeleitet. Über die Methode `aenderung()` informiert sie einen Vermittler über Zustandsänderungen:

```
// Datei: KonkreterKollegeA.java
public class KonkreterKollegeA extends Kollege
{
   // Konstruktor
   public KonkreterKollegeA (Vermittler v)
   {
      super (v);
```

```
      System.out.println ("KonkreterKollegeA: instanziiert");
   }

   // Wird aufgerufen, wenn sich ein anderer Kollege aendert
   public void methodeA()
   {
      System.out.println
         ("KonkreterKollegeA wird in methodeA() geaendert " +
          "als Folge der Aenderung eines Kollegen");
   }

   // Neuen Status setzen
   public void aenderung()
   {
      System.out.println
         ("KonkreterKollegeA wurde geaendert durch Aufruf" +
          " der Methode aenderung(). KonkreterKollegeA" +
          " informiert den Vermittler ");
      super.aenderung(); // informiert Vermittler
   }
}
```

Die Klasse `KonkreterKollegeB` ist ebenfalls von der Klasse `Kollege` abgeleitet und kann über die Methode `aenderung()` einen Vermittler informieren:

```
// Datei: KonkreterKollegeB.java
public class KonkreterKollegeB extends Kollege
{
   // Konstruktor
   public KonkreterKollegeB (Vermittler v)
   {
      super (v);
      System.out.println ("KonkreterKollegeB: instanziiert");
   }

   // Wird aufgerufen, wenn sich ein anderer Kollege aendert
   public void methodeB()
   {
      System.out.println
         ("KonkreterKollegeB wird in methodeB() geaendert" +
          " als Folge der Aenderung eines Kollegen");
   }

   // Neuen Status setzen
   public void aenderung()
   {
      System.out.println
         ("KonkreterKollegeB wurde geaendert durch Aufruf" +
          " der Methode aenderung(). KonkreterKollegeB" +
          " informiert den Vermittler ");
      super.aenderung(); // informiert Vermittler
   }
}
```

Die abstrakte Klasse `Vermittler` definiert die Schnittstelle, über die ein konkreter Vermittler über Änderungen von Kollegen informiert werden kann:

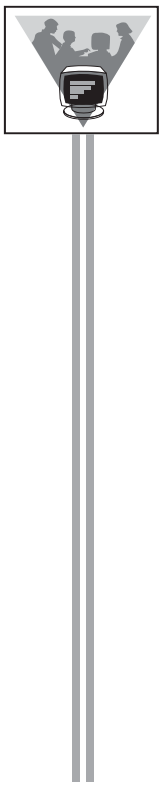
```
// Datei: Vermittler.java
public abstract class Vermittler
{
   // zur Information von Kollegen
   public abstract void aenderungAufgetreten(Kollege kollege);
}
```

Die Klasse `KonkreterVermittler` implementiert die abstrakten Methoden der Basisklasse `Vermittler`. Bei Änderung eines Kollegen informiert er den jeweils anderen Kollegen. Über set-Methoden erhält ein konkreter Vermittler Kenntnis über die beiden an der Kommunikation beteiligten Objekte. Hier nun der Quellcode der Klasse `KonkreterVermittler`:

```
// Datei: KonkreterVermittler.java
public class KonkreterVermittler extends Vermittler
{
   // Instanzvariablen
   private KonkreterKollegeA kollegeA;
   private KonkreterKollegeB kollegeB;

   // Konstruktor
   public KonkreterVermittler()
   {
      System.out.println("KonkreterVermittler: instanziiert");
   }

   // bei Aenderungen ruft der geaenderte Kollege diese
   // Vermittler-Methode auf
   public void aenderungAufgetreten (Kollege k)
   {
      if (k == (Kollege)kollegeA)
      {
         System.out.println
            ("KonkreterVermittler: informiere KollegeB");
         kollegeB.methodeB();
      }
      else if (k == (Kollege)kollegeB)
      {
         System.out.println
            ("KonkreterVermittler: informiere KollegeA");
         kollegeA.methodeA();
      }
   }

   //Set-Methoden fuer Kollegen
   public void setKollegeA (KonkreterKollegeA kka)
   {
      kollegeA = kka;
   }

   public void setKollegeB (KonkreterKollegeB kkb)
   {
      kollegeB = kkb;
   }
}
```

Das Client-Programm ist in zwei Phasen aufgeteilt. In der ersten Phase werden alle Objekte instanziiert. In der zweiten Phase führt der `Client` die Veränderungen an den Objekten der Klassen `KonkreterKollegeA` und `KonkreterKollegeB` durch:

```
// Datei: Client.java
public class Client
{
   public static void main (String[] args)
   {
      // Initialisierung
      System.out.println ("Initialisierung:");
      KonkreterVermittler konkreterVermittler =
         new KonkreterVermittler();
      KonkreterKollegeA kollegeA = new
         KonkreterKollegeA (konkreterVermittler);
      konkreterVermittler.setKollegeA (kollegeA);
      KonkreterKollegeB kollegeB = new
         KonkreterKollegeB (konkreterVermittler);
      konkreterVermittler.setKollegeB(kollegeB);

      // KollegeA aendern
      System.out.println ("\nKollegeA aendern:");
      kollegeA.aenderung();

      // KollegeB aendern
      System.out.println ("\nKollegeB aendern:");
      kollegeB.aenderung();
   }
}
```



Die Ausgabe ist:

```
Initialisierung:
KonkreterVermittler: instanziiert
KonkreterKollegeA: instanziiert
KonkreterKollegeB: instanziiert

KollegeA aendern:
KonkreterKollegeA wurde geaendert durch Aufruf der
Methode aenderung(). KonkreterKollegeA informiert den
Vermittler
KonkreterVermittler: informiere KollegeB
KonkreterKollegeB wird in methodeB() geaendert als
Folge der Aenderung eines Kollegen

KollegeB aendern:
KonkreterKollegeB wurde geaendert durch Aufruf der
Methode aenderung(). KonkreterKollegeB informiert den
Vermittler
KonkreterVermittler: informiere KollegeA
KonkreterKollegeA wird in methodeA() geaendert als
Folge der Aenderung eines Kollegen
```

Anzumerken ist, dass das gezeigte Beispiel untypisch ist, da es aus Platzgründen sehr vereinfacht ist. Die Kommunikation zwischen den beiden Kollegen ist hier im Vermittler hart codiert. Typischerweise muss der Vermittler eine Tabelle verwalten, in der die Kommunikationsverbindungen gespeichert sind, und die Verbindungen zwischen Objekten müssen tabellengesteuert durch Interpretation der Tabelle hergestellt werden.

4.13.4 Bewertung

4.13.4.1 Vorteile

Die folgenden Vorteile werden gesehen:

- Kollegen-Objekte sind untereinander lose gekoppelt und können wiederverwendet werden.
- Zwischen den Kollegen spannt sich ohne Vermittler eine n-zu-m-Beziehung auf. Der Vermittler reduziert dies jeweils auf eine 1-zu-n-Beziehung zwischen Objekten und Vermittler, was die Verständlichkeit, Verwaltung und Erweiterbarkeit verbessert.
- Die Steuerung der Kollegen-Objekte ist zentralisiert.
- Die Unterklassenbildung wird reduziert, da bei einer Änderung der Kommunikation zwischen Kollegen lediglich neue konkrete Vermittler erzeugt bzw. vorhandene Vermittler geändert werden müssen und keine neuen konkreten Kollegenklassen erzeugt werden müssen.
- Objekte können geändert werden, ohne den Kommunikationspartner anzupassen. (Rückwirkungsfreiheit auf Kommunikationspartner). Unter Umständen muss der Vermittler geändert werden. Damit sind Änderungen lokalisiert.

4.13.4.2 Nachteile

Die folgenden Nachteile werden festgestellt:

- Da der Vermittler die Kommunikation mit den Kollegen in sich kapselt, kann er unter Umständen sehr komplex werden.
- Der zentrale Vermittler ist fehleranfällig und bedarf fehlertoleranter Maßnahmen.
- Wenn sich die Kollegenschaft oder ihr Zusammenspiel ändert, muss der Vermittler angepasst werden.

4.13.5 Einsatzgebiete

Das Vermittler-Muster ist einzusetzen, wenn:

- Objekte zusammenarbeiten und miteinander kommunizieren, die Art und Weise der Zusammenarbeit und der Kommunikation aber flexibel änderbar sein soll, ohne dabei Unterklassen zu bilden (siehe folgendes Anwendungsbeispiel für grafische Oberflächen),
- ein Objekt aufgrund seiner engen Kopplung an andere Objekte schwer wiederzuverwenden ist oder
- Objekte ihre Kommunikationspartner nicht direkt kennen sollen.

Anwendungsbeispiel: Grafische Oberflächen

Das Vermittler-Muster wird häufig bei grafischen Oberflächen eingesetzt, bei denen mehrere Elemente wie Eingabefelder, Drop-Down-Menüs und Buttons beispielsweise in einer Dialogbox zusammenspielen. Nach der Eingabe von Zeichen in einem Eingabefeld müssen etwa in einer Dialogbox Buttons aktiviert oder deaktiviert werden, in einer anderen Dialogbox dagegen kann es nötig sein, dass der eingegebene Text dahingehend geprüft werden muss, ob es sich bei der Eingabe um einen gültigen Datensatz handelt. Löst man das Problem dadurch, dass man das spezifische Verhalten eines Eingabefeldes in einer Unterklasse realisiert, entsteht eine ganze Reihe von Unterklassen, die nur noch selten wiederverwendbar sind.

Durch die Einschaltung eines Vermittlers wird die Lösung einfacher und die Klassen bleiben wiederverwendbar: Jede Dialogbox bekommt einen eigenen Vermittler, der – um im Beispiel zu bleiben – von einem Eingabefeld benachrichtigt wird, dass eine Texteingabe vorliegt. Nun entscheidet der Vermittler einer Dialogbox, welche anderen grafischen Elemente darüber informiert werden müssen.

Die grafischen Elemente aus diesem Beispiel entsprechen den Kollegen im Sinne des Vermittler-Musters. Sie brauchen nicht über Unterklassen an ein spezifisches Verhalten angepasst werden, sondern nur der Vermittler einer Dialogbox muss das jeweilige Verhalten realisieren. Das Beispiel stammt aus [Gam95] und wird dort ausführlich beschrieben.

4.13.6 Ähnliche Entwurfsmuster

Ein Broker ist umgangssprachlich auch ein Vermittler. Das **Broker-Muster** hat auch eine gewisse Ähnlichkeit mit dem Vermittler-Muster, dadurch, dass die Kommunikation zwischen Komponenten (Kollegen) über einen Broker bzw Vermittler abläuft. Bei genauerer Betrachtung ergeben sich aber wesentliche Unterschiede:

- Beim Vermittler-Muster kommunizieren die Kollegen über den Vermittler, wobei der Vermittler auf Grund von bei ihm eingehenden Nachrichten betroffene Kollegen informiert. Alle Beteiligten befinden sich im selben System. Beim Broker-Muster spielt der Broker eine ähnliche Rolle wie ein Vermittler, der Broker leitet eine Nachricht nur an den einen gewünschten Empfänger weiter, jedoch können Clients, Server und Broker verteilt auf verschiedenen Rechnern ablaufen.
- Beim Architekturmuster Broker teilt eine Komponente dem Broker den Empfänger der Nachricht mit, der Broker ist für die Lokalisierung der Empfänger-Komponente im verteilten System, den Transport der Nachricht zum Empfänger und ggf. auch für den Rücktransport einer Antwort zuständig. Dieser letzte Aspekt fehlt beim Vermittler-Muster vollständig. Dies liegt auch daran, dass es beim Vermittler-Muster keine 1-zu-1-Zuordnung zwischen Sender und Empfänger wie bei der Anfrage eines Clients an einen Server beim Broker-Muster gibt, sondern eine 1-zu-n-Zuordnung. Außerdem erwartet ein Client in der Regel vom Server eine Antwort. Im Vermittler hingegen wird abgelegt, welche anderen Kollegen über eine Nachricht informiert werden sollen. Eine Antwort wird dabei von den benachrichtigten Kollegen nicht erwartet.

Sowohl über das Vermittler-Muster als auch über das **Beobachter-Muster** kann die Zusammenarbeit von Objekten gesteuert werden. Während beim Vermittler-Muster die Objekte (die Kollegen) gleichberechtigt sind und potentiell jedes Objekt mit jedem anderen über den Vermittler kommunizieren kann, haben die am Beobachter-Muster beteiligten Objekte bestimmte Rollen, die die Kommunikationsmöglichkeiten einschränken: nur beobachtbare Objekte können ihre Beobachter informieren und nicht umgekehrt. Das bedeutet, dass das Beobachter-Muster für einfachere Anwendungen besser geeignet ist. Würde man aber die komplexe Zusammenarbeit zwischen den Kollegen über das Beobachter-Muster realisieren, wäre jeder Kollege sowohl Beobachter als auch Beobachtbarer. Die daraus resultierende Kaskade von Benachrichtigungen wäre unüberschaubar und kaum nachzuvollziehen. Die Einschaltung eines Vermittlers "synchronisiert" in gewisser Weise auch die Zusammenarbeit. Denn ein benachrichtigter Kollege kann zwar sofort wieder den Vermittler anrufen, aber der Vermittler wird zuerst die alte Benachrichtigung noch komplett abarbeiten, bevor er sich dem neuen Anruf zuwendet.