Programmierung mit Entwurfsmustern
Eine Fallstudie in einem industriellen Umfeld

Christian Gruber[1], Nils Gruschka[2] und Friedemann Simon[2]

[1] RWE-DEA AG Hamburg, gvocg@rwedea.de
[2] Christian-Albrechts-Universiät Kiel, {ngr|fs}@informatik.uni-kiel.de

Zusammenfassung Am Beispiel einer Dialogsteuerung wurde gezeigt, wie der Einsatz vom Entwurfsmustern zur Lösung von konkreten Entwurfsproblemen in der industriellen Praxis beitragen kann. Die Ergebnisse wurden bei RWE-DEA AG als Basis für eine mit Visual Basic erstellte Anwendung verwendet.

1 Einführung

Die Grundlage dieses Artikels ist die Diplomarbeit von Nils Gruschka bei der Firma RWE-DEA AG, in der es um den Einsatz von Entwurfsmustern (engl. Design Patterns) für typische Aufgaben beim Softwareentwurf ging. Ausgehend von einer systematischen Katalogisierung von wiederkehrenden Entwurfsproblemen wurden entwurfsmusterbasierte Software-Design-Lösungen erarbeitet, mit der Standardentwicklungsumgebung der RWE-DEA AG implementiert und im industriellen Umfeld erprobt. Die dafür vorgegebene Programmiersprache war Visual Basic. Für Entwurfsmuster wurde die Standardreferenz [1] bzw. die amerikanische Originalfassung [2] benutzt.

2 Das Umfeld der Fallstudie

2.1 Das Unternehmen

Die Firma RWE-DEA Aktiengesellschaft für Mineralöl und Chemie, eine Tochtergesellschaft der RWE AG, ist ein deutsches Mineralölunternehmen mit weltweiten Chemieaktivitäten. Sitz der RWE-DEA AG ist in Hamburg. Sie hat über 10.000 Mitarbeiter und einen Jahresumsatz vom 27 Mrd. DM (Geschäftsjahr 1997/98).

Die beiden größten Tochtergesellschaften sind die DEA Mineraloel Aktiengesellschaft und die CONDEA.

Die Aufgabenfelder der DEA Mineraloel AG sind: Rohölversorgung, Mineralölverarbeitung und Verkauf von Mineralöl- und petrochemischen Produkten. Die Raffinerien der DEA haben eine Verarbeitungskapazität von 20 Mio. Tonnen Rohöl pro Jahr. Die DEA hat ein Tankstellennetz mit über 1600 Tankstellen, außerdem diverse Tochtergesellschaften zum Vertrieb von Kraftstoffen, Heizöl und Schmierstoffen.

Die CONDEA produziert vor allem Grundstoffe für die weiterverarbeitende chemische Industrie. Sie verfügt über Produktionsstätten in Deutschland und im Ausland (u. a. USA und China).

2.2 Die Abteilung ITAT

Der IT-(Informationstechnik-)Bereich der RWE-DEA AG besteht aus 5 Abteilungen: ITAT (technische Anwendungen), ITBK (betriebswirtschaftliche Anwendungen), ITBZ (Betriebszentrum), ITKC (Konzepte & Controlling) und ITST (Systemtechnik).

In der Abteilung ITAT werden technische Anwendungen für die gesamte RWE-DEA AG entworfen und programmiert. Manche Systeme werden von Grund auf selbst entwickelt, häufig werden aber auch Teile der Software eingekauft und gegebenenfalls modifiziert bzw. um Funktionalität erweitert.

Beispiele für typische Anwendungen, die bei ITAT entwickelt wurden, sind: Planungssysteme, Optimierungssysteme, Laborinformationssysteme, Managementinformationssysteme, Versandsysteme, Bestandsinformationssysteme.

Die existierenden Anwendungen laufen auf ganz unterschiedlichen Plattformen: einige Systeme noch auf IBM Host-Rechnern, andere unter UNIX, Windows NT oder Windows 95. Dementsprechend vielfältig sind die Programmiersprachen, mit denen Anwendungen geschrieben wurden. So wurde z. B. unter Windows u. a. Smalltalk, Visual C++, JAM und Visual Basic verwendet.

Die RWE-DEA AG hat entschieden, die eingesetzten Arbeitsplattformen zu vereinheitlichen. Dabei hat man sich im wesentlichen für Microsoft Produkte entschieden. Im einzelnen bedeutet dies: als Betriebssystem wird nur noch Windows 95 (für Einzelplatzsysteme) und Windows NT (für Server) verwendet. Auf Einzelplatzsystemen werden als Anwendungen die Microsoft Office-Programme (Word, Excel, Access, Outlook) verwendet und als Programmiersprache Visual Basic. Auf den Servern laufen u. a. SAP-Anwendungen und relationale Datenbanken (z. B. Oracle).

2.3 Visual Basic

Microsoft Visual Basic (VB) ist zur einfachen Entwicklung von Dialogfenstern für graphische Benutzeroberflächen (GUI) entwickelt worden. Die Entwicklungsumgebung von VB bietet die Möglichkeit die Elemente solcher Dialoge (z. B. Buttons, Eingabefelder, List-Boxen, usw.) zu erzeugen, mittels Drag-and-Drop zu positionieren und Eigenschaften zu verändern. Der mit dem jeweilige Ereignissen eines Dialogelementes (z. B. Klicken auf einen Button oder Eingabe in ein Textfeld) assoziierte Code kann direkt editiert werden.

Seit der Version 5.0 ist VB außerdem eine objektbasierte Sprache mit Klassen, Objekten und Methoden. Klassen definieren die Methoden und Daten der Objekte. Objekte sind Instanziierungen einer Klasse, kapseln ihre Daten und erlauben nur den Zugriff über öffentliche Methoden.

Im Gegensatz zu objektorientierten Sprachen (wie z. B. JAVA) gibt es aber weder Klassenmethoden noch Klassenvariablen. Es gibt auch keine Implementierungsvererbung. Die einzige Form der Vererbung in VB ist die Schnittstellenvererbung.

Visual Basic ist relativ leicht zu lernen, die elementaren Sprachkonstrukte sind wie im „klassischen“ BASIC. Allerdings sind Elemente, die man in modernen Programmiersprachen nicht mehr erwartet, entfernt worden (Zeilennummern, das `GOTO`-Konstrukt, implizite Typdeklaration durch Variablennamen).

Visual Basic erlaubt dem Programmierer ein *rapid Prototyping*. So kann man z. B. Methoden durch einen schnell erstellten Button testweise ausführen. Während der Entwicklung wird Code nicht komplett kompiliert, sondern interpretierend ausgeführt. Das komplette Kompilieren geschieht nur auf einen gesonderten Befehl hin oder beim Schreiben des Programms in eine ausführbare Datei.

Eine weitere wichtige Eigenschaft von VB ist die Entwicklung und Benutzung von Microsoft COM-Komponenten und damit insbesondere die Einbindung von VB-Programmen in die Microsoft Office-Palette.

2.4 Zur Schnittstellenvererbung

Im folgenden werden die Begriffe Schnittstellenvererbung und Implementierungsvererbung immer wieder auftauchen. Der Begriff der Implementierungsvererbung, genauso wie andere grundlegende Begriffe der objektorientierten Programmierung wie Klasse, Objekt oder Methode, werden als bekannt vorausgesetzt. (siehe z. B. [4])

Eine *Signatur* ist ein Tupel aus Methodenname, Parameter(n) und Rückgabewert. Eine *Schnittstelle* ist eine Menge von Signaturen. Der *Typ* eines Objektes ist der Name der Schnittstelle, die es implementiert. Ein Objekt kann dabei mehrere Typen haben. Umgekehrt können viele, ganz unterschiedliche Objekte den selben Typ haben. Bei der *Schnittstellenvererbung* erbt der Subtyp von seinem Supertyp nur die Schnittstelle, nicht aber die dazugehörige Implementierung. Einige objektorientierte Programmiersprache bieten die Definition und Vererbung von Schnittstellen direkt als Konstrukt an. In JAVA gibt es dafür z. B. das `interface`-Konstrukt. In anderen Sprachen, wie z. B. C++, entspricht die Schnittstellenvererbung dem Erben von rein abstrakten Klassen.

In Visual Basic definiert man die Schnittstelle als Klasse, wobei die Rümpfe der Methoden leergelassen werden. Soll eine Klasse diese Schnittstelle implementieren, so geschieht das mit der Anweisung `implements <interface>`. Der Compiler zwingt einen dann, jede Methode, die die Schnittstelle enthält, zu implementieren.

Das Verwenden von Schnittstellenvererbung führt zu flexibleren Entwürfen. So müssen Klienten nicht die Klasse eines Objektes, auf das sie zugreifen, kennen, sondern nur die Schnittstelle, die das Objekt implementiert. Man dann dieses Objekt unmittelbar durch ein anderes Objekt ersetzen, das nur dieselbe Schnittstelle implementieren muß.

3 Entwurfsmuster

3.1 Motivation

Der Entwurf objektorientierter Software allein ist schon schwer, aber noch schwerer ist es, einen Entwurf so zu gestalten, daß die Software auch *wiederverwendbar* ist. Die Erwartung, daß Teile von objektorientierten Programmen problemlos wiederzuverwenden sind, hat sich als Irrtum herausgestellt. Es ist in der Regel sehr schwierig, eine zweckmäßige Granularität von Klassen zu finden. Zu umfangreiche oder zu kleine Klassen (bezüglich Methoden und Zustandsvariablen) oder zu viele Relationen zwischen Klassen können die Wiederverwendbarkeit stark einschränken oder gar zunichte machen. Entwurfsmuster repräsentieren die Erfahrungen, die Programmierer beim Entwurf von objektorientierter Software gemacht haben, und zeigen „gute“ Lösungen für typische Probleme, die beim Entwurf auftreten. So ist die vorrangige Verwendung von Schnittstellenvererbung anstelle von Implementierungsvererbung eine dieser Erfahrungen. Insbesondere darf man erwarten, daß eine klare Strukturierung von Software durch Entwurfsmuster leichter zu wiederverwendbaren Klassen führt; Entwurfsmuster selbst sind natürlich auch immer wieder anwendbar.

3.2 Was sind Entwurfsmuster?

„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“ – Christopher Alexander [7]

Diese Beschreibung von Mustern in der Architektur trifft genauso zu auf die Muster über die wir hier sprechen, Muster zum Entwurf von objektorientierter Software. Ein Entwurfsmuster besteht aus folgenden Elementen:

– Mustername (Stichwort zur Benennung des Problems und seiner Lösung)
– Problemabschnitt (Problem, Kontext, Vorbedingungen, „Negativbeispiel“)
– Lösungsabschnitt (Lösungselemente und deren Beziehung und Interaktion)
– Konsequenzabschnitt (Vor- und Nachteile der Musteranwendung)
– Implementierungshinweise (Beispielcode)

Ein Entwurfsmuster ist eine abstrakte Beschreibung von Standardlösungen beim Softwareentwurf, weniger bei der Anwendungsmodellierung. Man findet die Lösung in Form der wesentlichen Klassen und Objekte und deren Schnittstellen und Interaktionen. Konkreten Code findet man (normalerweise) nur zur Illustration der Lösung, die i. a. sprachunabhängig ist.

3.3 Erwartungen des Unternehmens

Die RWE-DEA AG möchte Entwurfsmuster beim Softwaredesign einsetzen und dadurch die Produktivität und Qualität der Softwareentwicklung steigern und damit eine Reduzierung von Kosten erreichen. Für gewisse, wiederkehrende Probleme, deren Lösung mit den Standardmustern [1], [2] nur unzulänglich beschrieben werden kann, wird man eigene Entwurfsmuster entwerfen, erproben und in zukünftige Projekten benutzen.

Neben der prinzipiellen Frage, ob Entwurfsmuster beim Design das leisten, was sie versprechen, zwei weitere wichtige Fragen: Lassen sich die üblichen Entwurfsmuster auf die Probleme der industriellen Praxis anwenden? Können diese Lösungen dann auch in Visual Basic umgesetzt/implementiert werden?

4 Fallstudie: Ablaufsteuerung

4.1 Entwurfsproblem

Die *Ablaufsteuerung* genannte Aufgabe ist ein Teil des Entwurfs von graphischen Benutzeroberflächen (GUI) für Anwendungen, konkret für ein Raffinerie-Informationssystem im Werk Heide der DEA Mineraloel AG. Da diese Aufgabe zu den häufig anfallenden Routineaufgaben der Abteilung ITAT gehört, wurde sie zusammen mit weiteren Teilaufgaben als Objekt der Fallstudie ausgewählt. Hier wird nur ein kleiner, charakteristischer Ausschnitt der Studie vorgestellt.

Ein GUI besteht typischer Weise aus einer Menge von Dialogfenstern, die jeweils eine Menge von graphischen Kontrollelementen (z. B. Knöpfe, Menüs, Eingabefelder), die sogenannten Widgets, enthalten. Zwischen diesen Widgets können vielfältige Interaktionen bestehen (so soll vielleicht eine Auswahl in einer Liste ein Textfeld füllen). Programmiert man diese Aktionen direkt in die Widgetklassen, führt dies zu einer Reihe von Problemen. So muß im schlimmsten Fall jedes Widget alle anderen Widgets kennen. Wenn man ein Widget löscht, müssen die dazugehörigen Aktionen aus vielen Klassen gelöscht werden. Für jeden Dialog müssen neue Unterklassen des selben Widgettyps gebildet werden. Auf diese Abhängigkeiten, Kopplungen und Interaktionen trifft man auf der Ebene des ganzen GUI wieder bei der Organisation verschiedener Dialoge.

4.2 Lösung mit Entwurfsmustern

Ein Designvorschlag für das Zusammenspiel der Widgets eines Dialog wird durch das Entwurfsmuster *Vermittler* (engl. Mediator) geliefert durch die Einführung eines Objektes, das die Kommunikation zwischen den Widgets übernimmt, den DialogDirektor. Jedes Widget kennt nur den DialogDirektor. Findet auf einem Widget eine Benutzeraktion statt (z. B. Anklicken oder Tastatureingabe), dann schickt das Widget eine Nachricht an seinen DialogDirektor. In diesem ist der Code für die Auswirkung dieses Ereignisses (z. B. die Aktivierung oder Deaktivierung eines anderen Widgets) implementiert.

Jetzt muß man für jedem Dialog nur eine neue DialogDirektor-Klasse schreiben und bei Änderungen der Abhängigkeiten nur den DialogDirektor ändern.

Die Interaktion der Dialoge ist prinzipiell nichts anderes als die Interaktion der Widgets. Dies führt zu der Idee, das Gesamtproblem dadurch zu lösen, daß man das Muster des Vermittlers ein zweites Mal anwendet, dieses Mal auf der Ebene der gesamten Applikation[1].

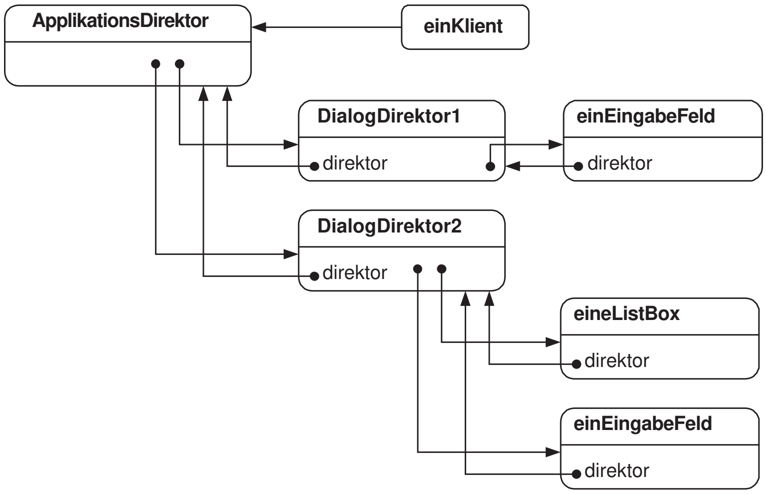

Abb. 1. Ablaufsteuerung: Objektdiagramm

In Abbildung 1 sieht man das Beispiel einer Objektstuktur, die sich bei der Einführung eines ApplikationsDirektors ergibt. Die Dialoge (genauer: deren DialogDirektoren) sind nicht mehr voneinander abhängig, sondern nur vom ApplikationsDirektor. Von Modifikationen der Anwendungsablaufsteuerung ist durch diese Entkopplung nur der ApplikationsDirektor betroffen.

Die Interaktion dieser Objekte kann man in Abbildung 2 erkennen. Der ApplikationsDirektor erzeugt den ersten DialogDirektor und schickt ihm die Nachricht zur Erzeugung seiner Widgets. Der DialogDirektor erzeugt dann seine Widgets. Findet auf einem Widget (hier eine ListBox) eine Benutzeraktion statt, so meldet das Widget diese Aktion seinem DialogDirektor. Der DialogDirektor schickt dann im Normalfall eine Nachricht an andere Widgets, für die diese Aktion Auswirkungen hat. In diesem Fall ist die Änderung aber auf Applikationsebene wichtig, deshalb schickt der DialogDirektor eine Nachricht an seinen ApplikationsDirektor. Dieser „entscheidet“, daß der Dialog geschlossen wird, und schickt die entsprechende Nachricht an den DialogDirektor. Danach erzeugt der ApplikationsDirektor einen neuen DialogDirektor und der 2. Dialog ist entstanden.

4.3 Umsetzung in Visual Basic

Wie man in dem Klassendiagramm der Lösung (Abb. 3.) erkennt, beruht das Prinzip dieser Lösung auf der Vererbung von Schnittstellen. Die Methoden, deren

[1] Die Kombination mehrerer Entwurfsmuster ist eine häufig angewandte Vorgehensweise.

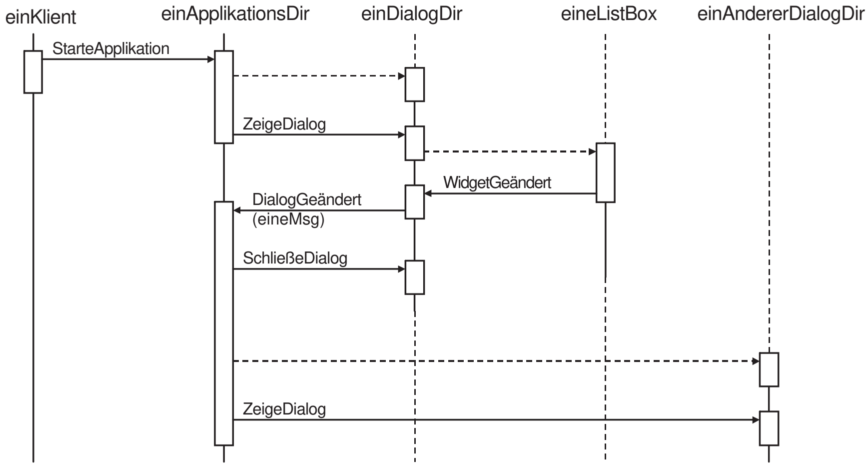

Abb. 2. Ablaufsteuerung: Interaktionsdiagramm

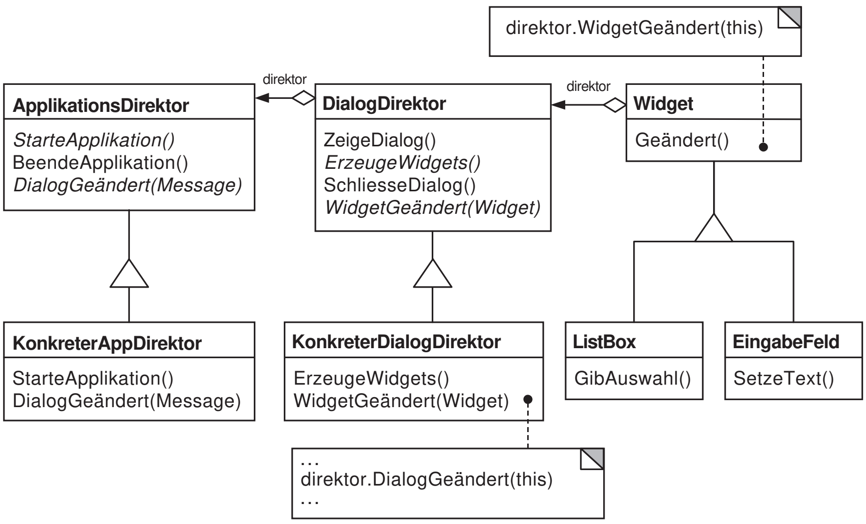

Abb. 3. Ablaufsteuerung: Klassendiagramm

Implementierungen vererbt werden (z. B. ZeigeDialog), sind für das Funktionieren dieser Lösung ohne Bedeutung. Da man in Visual Basic aber Schnittstellenvererbung zur Verfügung hat ist die Umsetzung dieser Lösung in Visual Basic unter Erhaltung der Struktur problemlos möglich.

5 Ergebnis

Im Zusammenhang mit der Entwicklung eines neuen Raffinerie-Informationssystems wurde die Erstellung und Pflege von GUIs auf der Basis von Entwurfsmustern erprobt und qualitativ bewertet. Die folgenden, auch in der Literatur immer wieder als Vorteile genannten Aussagen konnten ohne Abstriche bestätigt werden:

- Klare Strukturen auf höherem Abstraktionsniveau als Klassen und Objekte führen zu „Robustheit“ von Code, besserer Wartbarkeit und besserer Dokumentation.
- Kostenreduktion durch höhere Produktivität ist realistisch.
- Arbeitsteilige Entwicklung von umfangreichen Anwendungen wird gefördert durch Strukturierung entsprechend Entwurfsmustern und durch bessere Kommunizierbarkeit von Programmentwürfen bzw. Programmen.

Nach einer breiteren Ausbildung von Mitarbeitern der Abteilung ITAT sollen Entwurfsmuster zukünftig bei der Entwicklung neuer Anwendungen zielgerichtet eingesetzt werden.

Literatur

1. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*, Addison-Wesley, 1996
2. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994
3. Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M.: *Patternorientierte Software-Architektur*, Addison-Wesley, 1998
4. Budd, T.: *An Introduction to Object-Oriented Programming*, Addison-Wesley, 1997
5. Oestreich, B., *Objektorientierte Softwareentwicklung – Analyse und Design mit Unified Modelling Language*, Oldenburg, 1999
6. Pree, W., *Komponentenbasierte Softwareentwicklung mit Frameworks*, dpunkt, 1997
7. Alexander, Ch.; Ishiakwa, S.; Silverstein, M.; Jacobson, M.; Fiksdahl-King, I.; Angel, S.: *A Pattern Language*, Oxford University Press, New York, 1977
8. *Microsoft Developer Network Library, Visual Studio 97 Edtion*, Microsoft Corporation, 1997
9. Buschmann, F., Meunier, R.: *Software-Konstruktion mit Entwurfsmustern*, S. 48–56, Objektspektrum, 5/94
10. Buschmann, F.: *Welche Arten von Entwurfsmustern gibt es?* S. 68–69, Objektspektrum, 4/95
11. Sommerlad, P.: *Entwurfsmuster für Software-Architektur,* S. 16–20, Objektspektrum, 2/96