Zusammenfassung

These

Durch funktionale Programmiersprachenkonzepte inspirierte Entwurfsmuster können objektorientierten Entwurf verbessern.

Problem

Das objektorientierte Paradigma hat ohne Zweifel unsere Möglichkeiten, komplexe Softwaresysteme zu entwerfen verbessert. Allerdings scheint es, daß die Erwartungen an Wiederverwendung und wartbare Software, die seit des kommerziellen Erfolges an die Objektorientierung bestehen, nicht eingelöst wurden.

Viele mögliche Faktoren, wie z.B. unqualifiziertes Personal, unreife Implementierungssprachen, ungenügende Entwicklungsmethodiken oder unangemessene Organisationsprozesse, können hier ursächlich sein.

Die hier eingenommene Sicht konstatiert, daß das objektorientierte Paradigma zwar eine exzellente Basis darstellt, aber wahrscheinlich unvollständig in seinen Entwurfsdimensionen ist. Dadurch scheint der Entwurfsraum auf unangemessene Lösungen eingeschränkt. Sowohl Software- als auch Sprachentwurf könnten eine größere Potentialentfaltung erleben, wenn eine Sicht über rein objektorientierte Konzepte hinaus eingenommen wird.

Lösung

Da das komplementäre Paradigma zur Objektorientierung durch die funktionale Programmierung repräsentiert wird, untersuche ich dort verwandte Konzepte auf ihre Verwendbarkeit im objektorientierten Kontext. Ich erforsche die softwaretechnische Relevanz jeden Konzepts und präsentiere seine Absicht, Anwendbarkeit, Implementierung und Konsequenzen in der literarischen Form eines Entwurfsmusters.

Mein Ansatz motiviert den Einsatz von funktionalen Konzepten innerhalb objektorientierten Entwurfs aus einer softwaretechnischen Perspektive heraus. Dies steht im Gegensatz zu einer Zusammenwürfelung objektorientierter und funktionaler Sprachkonstrukte, um eine neue Sprache zu erhalten. Letzterer Ansatz erfordert Sprachentwurfskompetenz und macht es schwierig, Anwendungen für die neue Sprache zu finden, bzw. diese zu bewerten.

Im Gegensatz dazu ist der Einsatz von Entwurfsmustern zur Verbesserung von Softwareentwürfen bereits weit verbreitet. Da funktionale Konzepte bereits ein in sich geschlossenes mächtiges Paradigma konstituieren, scheint es berechtigt anzunehmen, daß funktionale Entwurfsmuster das objektorientierte Paradigma zu neuen Möglichkeiten erweitern werden.

Beitrag

Durchführbarkeit

Ich zeige die Machbarkeit der Nutzung funktionaler Konzepte in objektorientiertem Entwurf bei Benutzung von herkömmlichen objektorientierten Implementierungssprachen. Dies geschieht einerseits theoretisch auf der Kalkülebene und andererseits praktisch mit konkreten Entwurfsmusterbeschreibungen. Ich zeige synergetische Effekte der Konzeptintegration auf, die zusammen mit den überzeugenden Vorteilen funktionaler Entwurfsmuster die Nützlichkeit des Ansatzes verdeutlichen.

Softwareproduktion

Objektorientierte Praktiker könnten die Namen funktionaler Entwurfsmuster als Vokabular in Diskussionen innerhalb eines neuen Entwurfsraums benutzen. Ich präsentiere ein System durch Relationen miteinander verbundener Entwurfsmuster, das beschreibt wie einzelne Muster miteinander interagieren und kollaborieren. Dieses, aus qualitativ hochwertigen Miniarchitekturen bestehende System, ermöglicht es über traditionelle, dogmatisch eingeschränkte objektorientierte Lösungen hinaus zu denken und zu entwerfen. Dementsprechend könnte eine Qualitätsverbesserung von Software erwartet werden.

Sprachentwurf

Die Benutzung von funktionalen Entwurfsmustern in einem objektorientiertem Kontext kann als Hybridparadigmenansatz angesehen werden. In diesem Licht erscheinen funktionale Entwurfsmuster als implementierungssprachliche Idiome, die eine objektorientierte Sprache zu einer Hybridparadigmensprache aufwerten.

Es ist sehr instruktiv zu bewerten, inwieweit eine spezifische Sprache die Umsetzung dieser Idiome unterstützt, da limitierende Eigenschaften auch in anderen Versuchen flexible und wartbare Software zu erstellen auftreten werden.

Freie Wahl der Implementierungssprache vorausgesetzt, ist es allerdings nur natürlich, direkte sprachliche Unterstützung der Idiome zu fordern, um deren wiederholte manuelle Anwendung zu vermeiden. Eine ganzheitlich konzipierte Sprache, die sowohl das objektorientierte als auch das funktionale Paradigma umfaßt, ist leichter anwendbar, bietet erhöhte Sicherheit, bessere initiale Ausführungseffizienz und ein höheres Optimierungspotential.

Ich betrachte jedes Entwurfsmuster bezüglich seiner Impulse auf Sprachkonstrukte, die eine Hybridparadigmensprache unterstützen. Das Heranziehen der softwaretechnischen Eigenschaften der Entwurfsmuster hilft, Sprachentwurf nicht auf eine "Featureliste" sondern auf konzeptuell fundierte Sprachprinzipien zu gründen.

Letztendlich führen die durch das Entwurfsmustersystem initiierten Impulse zu einer Neubewertung der Rollenverteilung von Programmiersprachen und deren zugeordneten Entwicklungsumgebungen. Der herausgearbeitete Ausblick erlaubt die Transzendierung der respektiven Paradigmenlimitationen durch die Möglichkeit, die jeweils optimale Paradigmensicht dynamisch einzunehmen.

Teil I

Vor dem ersten Teil dieser Arbeit leitet der Prolog in die Thematik ein. Der Handlungsbedarf für die Erweiterung des objektorientierten Paradigmas wird motiviert, funktionale Konzepte werden als vielversprechend angeführt und Paradigmenintegration wird als von allgemeinen Interesse identifiziert.

Nach einer Leseanleitung folgt der erste Teil, der das Fundament für die Arbeit setzt. Zum einen werden in drei Kapiteln die Grundlagen von funktionaler Programmierung, objektorientierter Programmierung, und Entwurfsmustern gelegt. Neben einer Begriffsbestimmung steuern die ersten beiden Kapitel auch eine Analyse der jeweiligen Vor- und Nachteile bei.

Kalkülvergleich

Vor der praktischen Integration der Paradigmen wird auf theoretischer Ebene überprüft, ob der Ansatz, die funktionalen Konzepte mit Objektorientierung zu subsumieren, vernünftig ist. Die Ergebnisse einer von mir konzipierten und betreuten Diplomarbeit [Thierbach96] bestätigen die gewählte Integrationsrichtung und schälen fundamentale Paradigmenunterschiede heraus. Während der λ-Kalkül freien Zugriff auf Umgebungsvariablen bereitstellt, bietet der objektorientierte Kalkül atomare Namensuche.

Paradigmenintegration

Im praktischen Integrationskapitel werden zunächst scheinbar unvereinbare Gegensätze aufgezeigt, die dann im nachfolgenden Ansatz aufgelöst werden. Mit der Entscheidung für seiteneffektfreie Funktionen wird referentielle Transparenz und auch verzögerte Auswertung für die Welt der Funktionen erhalten. Pattern Matching wird als "syntaktischer Zucker" ohne softwaretechnische Relevanz, aber mit einer Reihe von subtilen Problemen abgelehnt. Dieser Abschnitt schließt mit einer Untersuchung, wie sich scheinbare Gegensätze — Parametrisierung und Vererbung bzw. funktionale versus objektorientierte Dekomposition — synergetisch ergänzen können.

Teil II

Der zweite Teil, das Entwurfsmustersystem, wird vom Katalog der Muster und einem Abschnitt über das Zusammenwirken der Muster eingerahmt.

Katalog

Der Musterkatalog benutzt sogenannte “Pattlets”, um die Hauptidee der nachfolgenden Muster, in einem prägnanten Aphorismus auszudrücken. Die Muster werden im Kontext des ersten Teils motiviert und ihr Wirkungsprinzip wird kurz umrissen.

Function Object

Absicht

Kapsele eine Funktion mit einem Objekt. Dies ist nützlich für die Parametrisierung von Algorithmen, partielle Parametrisierung von Funktionen, verzögerte Berechnungen, Funktionen 1. Klasse und für die Separation von Funktionen von Daten.

Funktionsobjekte implementieren Funktionen höherer Ordnung als sogenannte Closures. Im Motivationsabschnitt werden die Vorteile gegenüber Vererbung genannt und eine Reihe wichtiger Variationen (z.B., Prozedurobjekt, Schlüsselwortparameter, “multi-dispatch” Funktionsobjekt) werden aufgezählt.

Funktionsobjekte können gegenüber Klienten sowohl die Anzahl der Argumente als auch der Resultate verstecken. Mit ihrer guten Kapselung sind sie ideale Bausteine für das sogenannte komponentenorientierte Programmieren. Sie bereichern das objektorientierte Paradigma mit gekapselter Parametrisierung und funktionaler Erweiterbarkeit.

Lazy Object

Absicht

Verzögere Berechnungen bis zum spätmöglichsten Zeitpunkt. Dies ist nützlich, um Abhängigkeiten aufzulösen, Berechnungskomplexität zu reduzieren, Modularität zu erhöhen und unendliche Datenstrukturen zu modellieren.

Die Verzögerung von Berechnungen durch Objektsuspensionen ermöglicht eine bedarfsgetriebene Ausführungsreihenfolge. Dadurch wird ein deklarativer Programmierstil ermöglicht, da auf künstliche obere Grenzen für Datenmengen verzichtet werden kann, Berechnungen — ohne deren verfrühte Ausführung — vorab spezifiziert werden können und Datenabhängigkeiten ohne explizite Angabe deren Auflösung ausgedrückt werden können.

Lazy Object ermöglicht es, Datengenerierung von Datenkonsummierung zu trennen, ohne dabei die Terminierungskontrolle auf einer Seite zu verlieren. Folglich sind modulare Lösungen möglich, die jedoch keinen Effizienzverlust implizieren. Im Gegenteil sind die bedarfsgesteuerte partielle Auswertung und die verschachtelte Ausführungscharakteristik von verzögerten Prozessen sogar effizienzsteigernd.

Ein Entwurf zur allgemeinen Behandlung von verzögerten Datenströmen wird ausführlich beschrieben.

Value Object

Absicht

Benutze unveränderbare Objekte mit Generatoroperationen als Schutz gegen Seiteneffekte und Mehrfachreferenz.

Dieses Entwurfsmuster legt den Finger auf zwar effizienzsteigernde aber im allgemeinen unintuitive Referenzsemantik in objektorientierten Sprachen. Der Unveränderbarkeit und der fehlenden Identität von Werten wird mit diesem Entwurfsmuster Rechnung getragen.

Auf diese Weise wird der seiteneffektfreie Bereich von Funktionen auf Objekte — wenn angemessen — ausgeweitet.

Transfold

Absicht

Bearbeite die Elemente eines oder mehrerer zusammengesetzter Objekte, ohne ihre Repräsentierung freizulegen und ohne explizite Schleifenbenutzung.

Unter Zuhilfenahme von Funktionsobjekten wird es möglich, Strukturen intern abzuarbeiten, d.h., Strukturiterationen müssen nicht von Klienten gesteuert werden. Die normalerweise mit interner Iteration verbundene Inflexibilität wird durch das gleichzeitige Bearbeiten mehrerer Strukturen umgangen. Die Verwendung von verzögerter Auswertung ermöglicht eine Trennung von Strukturexplorationsprozeß und Datenbearbeitung. Folglich können die Vorteile von interner und externer Iteration, d.h., struktureigene Erforschung und Flexibilität in der Bearbeitungstrategie, kombiniert werden.

Ein Iterationskonzept, das spezielle Strukturoperationen durch allgemeines “Falten” verfügbar macht, wird mit seinen Implikationen auf Bibliothekenentwurf vorgestellt.

Void Value

Absicht

Werte "Nil" zu einem Wert 1. Klasse auf. Dies erlaubt, nicht initialisierte Daten und initialisierte Daten uniform zu behandeln. Weiterhin kann hiermit Standardverhalten bereitgestellt werden, die Formulierung von rekursiven Methoden wird vereinfacht und Fehlersituationen können ohne Systemabbruch behandelt werden.

Gemäß der Beobachtung, daß die Existenz eines allgegenwärtigen "Nil" Wertes "pattern matching"-ähnliche Abfragen in Klientenkode verursacht, wird "Nil" durch typspezifische Initialisierungswerte ersetzt.

Durch die konsequente Nutzung prozeduraler Abstraktion wird einerseits Klientenkode stark vereinfacht und andererseits eine große Quelle von Ausnahmebehandlungsfehlern ausgeschaltet.

Translator

Absicht

Stelle Interpretationen auf Strukturen mit heterogenen Elementen bereit, ohne die Elemente zu ändern. Separiere Interpretationen voneinander und benutze Teilinterpretationen, die eine inkrementelle Neuauswertung erlauben.

Das Entwurfsmuster Translator benutzt die "multi-dispatch" Funktionsobjektvariante (generische Funktionsobjekte), um externe Funktionalität auf Datenstrukturen zu definieren. Dieses Entwurfsmuster benutzt homomorphe Übersetzungen, um inkrementelle Auswertung zu erreichen und weiterhin eine Zwischendatenstruktur, um Übersetzung und Interpretation von einander zu trennen.

Eine herausragende Charakteristik dieses Entwurfsmusters ist, daß Datenstrukturen völlig von Interpretationen unbehelligt bleiben. Da generische Funktionsobjekte durch Zustandsänderung an neue Daten angepaßt werden können, ist sogar die Erweiterbarkeit von Datentypen bei funktionaler Dekomposition gegeben.

Zusammenwirken

Schon in den individuellen Beschreibungen werden die Entwurfsmuster durch Querverweise und die entsprechenden "Verwandte Muster"-Abschnitte miteinander vernetzt. Dieses Kapitel faßt alle Relationen in drei Sparten zusammen:

- *Kategorisierung:* Welches Muster kann die Rolle eines anderen spielen?
- *Implementierung:* Welche Muster helfen bei der Realisierung eines anderen?
- *Kollaboration:* Welche Muster passen wie Puzzlestücke ineinander, um gemeinsam eine Lösungskonfiguration zu bilden?

Drei entsprechende Diagramme und vor allem ein weiteres viertes, das alle Relationen enthält, erlauben eine Analyse der Relationen und letztendlich eine relative Positionsbestimmung der Muster innerhalb ihres Systems. Hierdurch sind schon erste Hinweise auf mögliche Formen der sprachlichen Unterstützung, die im dritten Teil angesprochen wird, sichtbar.

Das Kapitel über die Zusammenwirkung der Entwurfsmuster schließt mit einem Beispiel, das alle Entwurfsmuster im Zusammenspiel zeigt. Ein Schaubild visualisiert das Zusammenspiel, und die durch funktionale Entwurfsmuster erzielten wünschenswerten Softwareeigenschaften werden tabellarisch aufgeführt.

Teil III

Der letzte Teil der Arbeit hat zwei Funktionen. Erstens erlauben die bei der Realisierung der Entwurfsmuster gemachten Erfahrungen Rückschlüsse auf die Implementierungssprache (in diesem Fall EIFFEL). Für EIFFEL wurde im Wesentlichen festgestellt, daß Objekterzeugung ein Ausdruck sein sollte, universal quantifizierte Typvariablen zur Spezifizierung von Typzusammenhängen verfügbar sein sollten und außerdem eine Unterscheidung zwischen schreibbaren und unveränderlichen generischen Parametern möglich sein müßte, um Subtypenbeziehungen zwischen Werten zu ermöglichen.

Die zweite Funktion des dritten Teils ist es, die Impulse der funktionalen Entwurfsmuster für ihre sprachliche Unterstützung aufzunehmen und ausblicksweise geeignete Umsetzungen zu identifizieren. Insbesondere nicht realisierbare Details oder ermüdende Tätigkeiten bei der Realisierung der vorgestellten Entwurfsmuster, werden hier als Motivation für grundlegende sprachliche Unterstützung benutzt. Zusammenfassend ergeben sich die Forderungen nach unterstützter partieller Parametrisierung, statisch getypten “multi-dispatch” Funktionen, verzögerter Auswertung, standardmäßiger Wertezuweisung mit zusätzlicher Benachrichtigungszuweisungsanweisung, automatische Referenzeninitialisierung und eine algebraische Bibliotheksorganisation.

In den entsprechenden Sprachentwurfsdiskussionen wird immer wieder die Beobachtung gemacht, daß sowohl die funktionale als auch die objektorientierte Perspektive — je nach momentanem Kontext — wünschenswert wäre. Das Entwurfsmuster Translator unterstützt zwar gleichzeitige Daten- und Funktionserweiterbarkeit, erlaubt aber keine objektorientierte Sicht der Operationen.

Folgerichtig schlage ich einen Ansatz vor, der die Paradigmenwahl von der Programmiersprache in die Programmierumgebung verlagert. Programmiersprachlich existiert dafür der Begriff der Kachel, einer Einheit, die das Verhalten einer Funktion auf einem Konstruktor definiert. Eine Kachelmatrix spezifiziert daher eine Schnittstelle auf einem Datentyp. Die Entwicklungsumgebung kann durch Reihen- oder Spaltenauswahl eine funktionale oder objektorientierte Sichtweise generieren. Das Kachelkonzept wird im weiteren Ausblick auf sich selbst angewandt, d.h., eine dreidimensionale Kachelhierarchie wird zur Organisation von Softwaresystemen angeregt.

Schlußfolgerung

Der Epilog dieser Arbeit versucht die erzielten Ergebnisse in ein Gesamtbild einzuordnen. Ich argumentiere wie das Entwurfsmustersystem helfen kann, die bestehende objektorientierte Praxis besser zu verstehen. Weiterhin benenne ich die Bereicherungen, die durch die funktionalen Entwurfsmuster an bisherige objektorientierte Lösungen gemacht werden. Als Beispiel sei hier das Entwurfsmuster Transfold herausgehoben, welches mit Hilfe von Funktionsobjekten und verzögerter Auswertung eine bisher nicht erreichte Kombination von positiven Eigenschaften erzielt. Aber auch Translator besticht mit seinen Eigenschaften inmitten zahlreicher Konkurrenten von “Visitor”-Varianten.

Nach einer Betrachtung über die Auswirkungen des Entwurfsmustersystems auf Endbenutzer, folgen abschließende Reflexionen über sprachliche Unterstützung, Nachteile und mögliche Werkzeugunterstützung.

Einer Betonung der Relevanz von Paradigmenintegration folgend, gehe ich auf das Verhältnis zwischen Entwurfsmustern und Paradigmen ein. Der enge Zusammenhang wird deutlich durch die

- signalisierende (warum sind Muster in einem Paradigma nötig?),
- beschreibende (durch was wird ein Paradigma konstituiert?) und
- integrative (Paradigmeneinbettung durch Entwurfsmuster)

Wirkung von Entwurfsmustern auf Paradigmen.

Der Epilog schließt mit einer Betrachtung über die Bedeutung von Entwurfsmustern für den Sprachentwurf und einer Diskussion über verwandte Paradigmenintegrationsversuche.