Reorganisation objektorientierter Systeme mit Entwurfsmustern

Thomas Genßler und Benedikt Schulz

Forschungszentrum Informatik Karlsruhe

{genssler|bschulz}@fzi.de

Zusammenfassung In dieser Arbeit stellen wir eine Methode zur werkzeugunterstützten Reorganisation objektorientierter Software vor. Die zentrale Idee unseres Ansatzes ist, Entwurfsmuster nicht als Bausteine von Systemen sondern als Transformationsoperatoren zu betrachten. Solche Operatoren transformieren ein existierendes System in ein System mit einem durch Entwurfsmuster flexibilisierten Design. Wir präsentieren einen algorithmischen Prozeß zur Anwendung von Entwurfsmustern und zeigen, wie Entwurfsmusteroperatoren mit Hilfe von verhaltens-bewahrenden Refactorings implementiert werden können.

1 Einführung

Das Thema werkzeugunterstützte Sanierung objektorientierter Altsoftware gewinnt zunehmend an Bedeutung. Gerade im Telekommunikationsbereich existieren bereits heute objektorientierte Systeme mit mehren Million Zeilen Code[Rit98], die über mehrere Jahre hinweg gewachsen sind. Diese Systeme enthalten in der Regel unternehmenskritisches Know-how und müssen deshalb gepflegt, erweitert und an sich verändernde Anforderungen angepaßt werden. Auf der anderen Seite sind diese Systeme meist aufgrund ihrer Größe und ihrer über die Jahre hinweg gewachsenen Struktur kaum mehr handhabbar, so daß sogar kleine Änderungen am Code oder am Design unabsehbare Folgen für das Gesamtsystem haben. Eine Lösung für dieses Dilemma besteht darin, diese Systeme werkzeugunterstützt zu reorganisieren und ihre Struktur und ihr Design zu verbessern.

In dieser Arbeit präsentieren wir eine Technik zur werkzeugunterstützten Reorganisation objektorientierter Systeme. Die zentrale Idee unseres Ansatzes ist es, Entwurfsmuster in existierenden Quellcode einzuführen und dadurch das Design des betreffenden Systems zu verbessern. Im folgenden gehen wir auf die Grundlagen unseres Ansatzes ein und skizzieren, wie dieser implementiert und angewendet werden kann.

2 Grundlagen

Unser Ansatz ist eine Weiterentwicklung zweier bereits existierender Ansätze zur Reorganisation objektorientierter Systeme – Entwurfsmusteroperatoren und Refactorings.

In [Zim97] wird der Begriff *Entwurfsmusteroperatoren* eingeführt. Im Gegensatz zum ursprünglichen Verständnis von Entwurfsmustern als Bausteine, die einen Softwaresystem hinzugefügt werden [GHJV95][BMR+96], beschreibt der Autor sie als Transformationsoperationen, die das Design eines existierenden Systems verbessern können. Zu diesem Zweck definiert der Autor einen algorithmischen Prozeß, der die Anwendung eines Entwurfsmusters beschreibt und aus fünf Teilschritten besteht (vgl. Abbildung1):

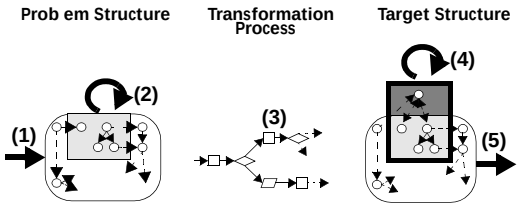

Abbildung1. Systematischer Prozeß der Entwurfsmusteranwendung

1. Identifikation der Problemstruktur: Die Problemstruktur umfaßt die Teile des Systems, die reorganisiert werden sollen. Dieser Schritt kann z.B. durch Problem-/Lösungsstrukturenkataloge unterstützt werden.
2. Prüfen der Vorbedingungen: Die verbal beschrieben Vorbedingungen eines Entwurfsmusteroperators müssen erfüllt sein.
3. Parametrierte Transformation: Hier findet die eigentliche Reorganisation statt. Parameter sind dabei z.B. neue Bezeichner oder die Variante des Entwurfsmusters.
4. Kontextreorganisation: Die Transformation der Problemstruktur zieht oft eine Veränderung ihrer Schnittstelle nach sich. Alle von dieser Änderung betroffenen Teile des Systems werden jetzt geeignet reorganisiert.
5. Prüfen der Nachbedingungen: Nach der Reorganisation muß unter der Voraussetzung, daß die Vorbedingungen erfüllt waren, eine Menge von Nachbedingungen erfüllt sein.

Die eigentliche Transformation des Systems wird als eine Sequenz von Operationen über einem Meta-Modell (z.B. "erzeuge Klasse") beschrieben. Eine detaillierter Beschreibung dieses Verfahrens findet man u.a. in [Zim97] und [SGMZ98].

Ein anderer Ansatz zur Reorganisation objektorientierter Systeme wird in [Opd92] beschrieben. Bei den dort beschriebenen *Refactorings* handelt es sich um verhaltensbewahrende Programmtransformationen. Für jede Refactoring-Operation ist eine Menge von Vorbedingungen definiert. Sind alle Vorbedingungen erfüllt, so wird garantiert, daß die Anwendung des Refactorings das Verhalten des Systems nicht ändert. Zunächst definiert er Autor eine Menge von Basisoperationen, deren Korrektheit bezüglich Verhaltensbewahrung bewiesen wird. Darüberhinaus definiert der Autor noch eine Reihe von Invarianten um syntaktische und semantische Korrektheit der Refactorings zu gewährleisten. Auf der Basis der einfachen Operationen werden anschließend komplexere Operationen definiert, die wiederum – die Erfüllung ihrer Vorbedingungen vorausgesetzt – verhaltensbewahrend sind.

3 Reorganisation mit Entwurfsmusteroperatoren

Wir haben, resultierend aus Erfahrungen vorangegangener Sanierungsprojekte, eine Reihe von Anforderungen identifiziert, die eine Reorganisationmethodik erfüllen muß, um praktikabel zu sein. Diese Anforderungen lassen sich in den folgenden Punkten zusammenfassen:

- **Sprachunabhängigkeit:** Eine Reorganisationsmethode sollte nicht auf eine spezielle Implementierungssprache ausgerichtet sein. Der Grund für diese Forderung ist die Tatsache, daß die meisten heutigen OO-Systeme mehrsprachig implementiert sind[1].
- **Richtige Abstraktionsebene:** Ein Großteil der Probleme eines objektorientierten Systems, betreffen das Design eines solchen Systems (z.B. fehlende Flexibilität). Aus diesem Grund muß eine Reorganisationsmethodik eher auf die Entwurfsebene statt auf die Implementierungsebene abzielen.
- **Verhaltensbewahrung:** Reorganisation dient nicht der funktionalen Erweiterung eines Systems. Stattdessen soll die Struktur eines Systems so verbessert werden, daß zukünftige funktionale Erweiterungen einfacher möglich sind. Dabei soll sich das Verhalten des transformierten Systems nicht ändern. Eine Reorganisationsmethodik muß das Verhalten des reorganisierten Systems beweisbar bewahren.
- **Werkzeugunterstützung:** Da die Reorganisation großer Systeme "von Hand" äußert aufwendig und fehleranfällig ist, muß sie durch Werkzeuge unterstützbar sein.

Beide im vorangegangen Kapitel vorgestellten Techniken zur Reorganisation erfüllen die gestellten Anforderungen nur partiell. Entwurfsmusteroperatoren sind sprachunabhängig und zielen auf die Entwurfsebene. Auf der anderen Seite bieten sie keine Möglichkeit, Verhaltensbewahrung konstruktiv – etwa durch automatische Überprüfung von Vorbedingungen[2] – sicherzustellen. Darüberhinaus ist das den Entwurfsmusteroperatoren zugrundeliegende Modell zu allgemein, als daß es direkt in Werkzeugen implementiert werden könnte. Demgegenüber bieten Refactorings zumindest einen semi-formalen Mechanismus, um Verhaltensgleichheit bei der Transformation von Software zu garantieren und zu beweisen. Darüberhinaus wurde bereits in mehreren Arbeiten nachgewiesen, das Refactorings in Form von Werkzeugen implementiert werden können [Opd92][RBJ97][SGMZ98]. Allerdings zeigt sich, daß Refactorings auf einer für Reorganisation zu niedrigen Abstraktionsebene ansetzen. Sie zielen vorrangig auf die Implementierungsebene eines Systems und bieten nur sehr beschränkte Möglichkeiten zur Reorganisation auf Entwurfsebene[SGMZ98]. Darüberhinaus sind Refactorings per se sehr stark von der Semantik der Implementierungssprache des zu reorganisierenden Systems abhängig.

Unser Ansatz kombiniert die Vorteile der beiden Ansätze. Wir implementieren *Entwurfsmusteroperatoren als Sequenz von Refactorings*. Während die bisherige Beschreibung von Entwurfsmusteroperatoren sehr informell war, sind wir jetzt in der Lage, diese Operatoren zu formalisieren, Vor- und Nachbedingungen ihrer Anwendung abzuleiten sowie Verhaltensgleichheit zu garantieren. Verhaltengleichheit wird durch die Tatsache gewährleistet, das die Anwendung einer Sequenz von für sich verhaltensbewahrenden Refactorings wiederum zu einem verhaltensgleichen System führt. Die Existenz von Refactorings für verschiedene objektorientierte Sprachen vorausgesetzt, sind wir in der Lage, Entwurfsmusteroperatoren jetzt als sprachunabhängige[3] Reorganisationswerkzeuge implementiert werden. Das

[1] Oft findet man z.B. Systeme, die einen in C++ implementierten Kern und eine in Java implementierte Oberfläche haben.

[2] Im ursprünglichen Ansatz werden Vorbedingungen informell beschrieben.

[3] Sprachunabhängigkeit kann durch den Austausch der zugrundeliegenden Refactoring-Implementierung erreicht werden

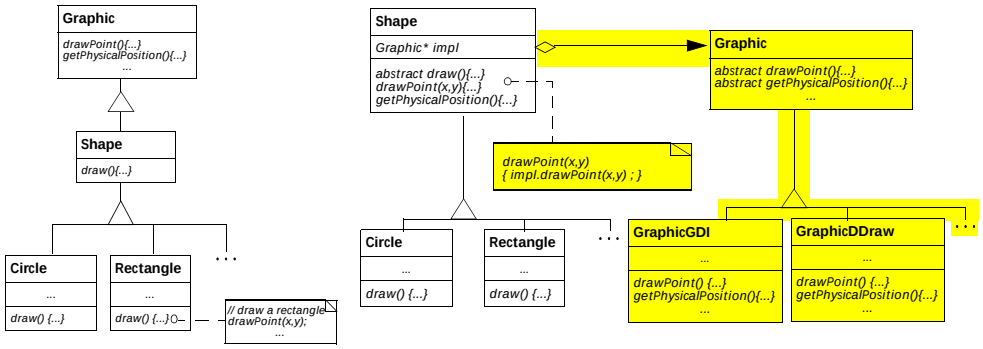

Abbildung2. Anwendung des Entwurfsmusteroperators *Bridge*

in [Zim97] beschriebene abstrakte Meta-Modell wird durch das in [Opd92] beschriebene Modell ersetzt. Die Vorbedingungen eines Entwurfsmusteroperators werden von den Vorbedingungen der für die Implementierung verwendeten Refactorings abgeleitet.

Beispiel Das folgende Beispiel stammt aus einer Fallstudie zur Reorganisation objektorientierter Software. Das untersuchte System dient der Visualisierung hydraulischer Daten. Ein Subsystem der Applikation ist verantwortlich für das Zeichnen geometrischer Objekte unter Benutzung der Windows GDI. Zu diesem Zweck wurde eine Klasse `Graphic` implementiert, die Zugriffe auf die eigentlichen GDI-Zeichenroutinen kapselt. Alle Domänenklassen erbten die notwendige Funktionalität von dieser Klasse. Der linke Teil von Abbildung2 illustriert die Ausgangssituation. In einer neuen Version der Software bestand die Anforderung, sowohl GDI als auch die modernere DirectDraw-Technologie zu unterstützen. Allerdings stellte sich das ursprüngliche Design aufgrund der enge Koppelung zwischen plattform- und domänenspezifischer Funktionalität als zu inflexibel für diese Anforderung heraus.

Eine Lösung dieses Problems stellt das Entwurfsmuster *Brücke* dar. Die Anwendung des Brücke-Operators transformiert die vormals enge Koppelung zwischen der Klasse `Graphic` und `Shape` respektive deren Unterklassen in eine flexiblere Delegationsbeziehung. Der Brücke-Operator wurde unter Zuhilfenahme der folgenden bereits in [Opd92] beschriebenen oder neu entwickelten Refactorings implementiert:

- Wandle Vererbung in Delegation (komposites Refactoring)
- Erzeuge leere Klasse
- Füge leere Methode oder neue Instanzvariable zu Klasse hinzu
- Setzte oder entferne Oberklassenbeziehung
- Füge Codefragmente (z.B. Initialisierungscode) ein

Eine detailliertere Beschreibung der Transformation findet sich in [SGMZ98]. Neben dem Brücke-Operator wurden bereits weitere Entwurfsmusteroperatoren (z.B. *Beobachter, Zustand, Stellvertreter*) implementiert.

4 Zusammenfassung

Dieses Papier beschreibt einen Ansatz zur werkzeugunterstützten Reorganisation auf der Basis von Entwurfsmusteroperatoren. Wir zeigen, wie diese Operatoren als Sequenzen verhaltensbewahrender Refactorings implementiert werden können, ohne dabei den Fokus auf die Entwurfsebene zu verlieren.

Literatur

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture – A System of Patterns.* Wiley and Sons Ltd, 1996.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[Opd92] W. Opdyke. *Refactoring Object-Oriented Frameworks.* PhD thesis, University of Illinois at Urbana-Champaign, 1992.

[RBJ97] D. Roberts, J. Brant, and R. Johnson. A refactoring tool for smalltalk. http://st-www.cs.uiuc.edu/users/brant/Refactory/RefactoringBrowser.html, 1997.

[Rit98] Fabian Ritzmann. Reverse Engineering of Large Scale Software Systems. Diplomarbeit, Universität Karlsruhe, June 1998.

[SGMZ98] B. Schulz, T. Genßler, B. Mohr, and W. Zimmer. On the Computer Aided Introduction of Design Patterns into Object-Oriented Systems. In Jian Chen, Mingshu Li, Christine Mingins, and Bertrand Meyer, editors, *Proceedings of the 27th TOOLS conference*, 1998.

[Zim97] W. Zimmer. *Frameworks und Entwurfsmuster.* PhD thesis, FZI, 1997.