Von Anwendungen zu Komponentensystemen

M. Bauer[1] *bauer@fzi.de*
R. Neumann[2] *Rainer.Neumann@gmx.de*
B. Schulz[1] *bschulz@fzi.de*

[1]*Forschungszentrum Informatik Karlsruhe (FZI)*
[2]*Universität Karlsruhe (TH)*

Einleitung

Die zunehmende Verbreitung von Software für immer komplexere Aufgaben führt zu immer umfangreicheren Softwaresystemen, die mit herkömmlichen Softwareerstellungsprozessen und Methoden kaum noch zu beherrschen sind. Die Folgen sind bekannt: Softwareprojekte überschreiten regelmäßig die Zeit- und Ressourcenvorgaben, die entstehenden Systeme sind unzuverlässig, ineffizient und schwierig zu warten und weiterzuentwickeln.

Komponenten als Bausteine für komplexe Softwaresysteme versprechen hier Abhilfe: Die Dekomposition eines Systems in Komponenten mit definierten Schnittstellen erlaubt eine verteilte Entwicklung und die Wiederverwendung von bestehenden Komponenten und damit kürzere Entwicklungszeiten. Durch die Modularisierung wird auch der Betrieb der Software flexibilisiert: Anwendungen können auf verschiedene Rechner verteilt werden und damit die zu Verfügung stehenden Systemressourcen besser ausnutzen. Schließlich eignen sich Komponenten als wiederverwendbare Softwarebausteine auch für eine eigenständige Vermarktung.

Bei den Systemen, die neu entwickelt werden, greift sowohl die Industrie als auch die akademische Welt die komponentenorientierte Softwareentwicklung begeistert auf. Doch neben diesen neuen Systemen existieren unzählige, oft mehrere Millionen Codezeilen umfassende Altsysteme, die noch nicht von der Komponententechnologie profitiert haben. Die fehlende Modularisierung der Systeme erweist sich dabei als zentrales Problem der Softwaresanierung. Um auch diesen alten Systemen die Vorteile der Komponententechnologie zu erschließen, müssen Verfahren entwickelt werden, um die monolithischen Systeme in flexible Komponentensysteme zu überführen. Dieses Papier beschreibt, welche Einzelaufgaben zur Lösung dieses Problems anstehen, und wie diese durchgeführt werden müssen.

Von Anwendungen zu Komponentensystemen

Die Vorgehensweise bei der Überführung eines gegebenen objektorientierten Systems in ein komponentenorientiertes System verläuft in der Regel nach einem strengen Schema:

- Zunächst wird die Zielarchitektur bestimmt, d.h. es werden die zentralen Anforderungen ermittelt, denen das neue System genügen muss. Oftmals geben diese Anforderungen bereits eine grobe Aufteilung des Systems vor.
- Danach werden Kandidaten für die zukünftigen Komponenten ermittelt. Dieser Schritt ist oftmals der schwierigste, da er nur bedingt automatisierbar ist und vom Erfahrungsschatz desjenigen abhängt, der ihn durchführt.
- Der technisch aufwendigste Schritt ist die anschließende tatsächliche Umsetzung der Anwendung im Hinblick auf die zuvor festgelegte Komponentenstruktur – die Komponentifizierung. Das zentrale Problem hierbei ist das derzeitige Fehlen von Werkzeugen, die helfen, die Umsetzung zu automatisieren und Fehler zu vermeiden.

Festlegung der Zielarchitektur

Bei diesem Schritt kommt es darauf an, die Anforderungen an das Zielsystem zu identifizieren und aus diesen Anforderungen und der aktuellen Systemarchitektur eine Zielarchitektur zu entwickeln, deren Umsetzung mit akzeptablem Aufwand machbar ist. Eine Beispiel einer solchen Anforderung ist die Aussage: „Unser Warenwirtschaftssystem soll mit einer Web-Schnittstelle versehen werden“.

Hierbei ist es wichtig festzustellen, inwiefern sich durch die neue Aufgabenstellung architekturelle Änderungen ergeben und inwiefern diese Änderungen realistisch durchführbar sind.

Finden von Komponenten

Das Auffinden von Komponenten erfolgt üblicherweise in zwei Schritten:

1. Aufstellen von Hypothesen – Ziel dieses Schrittes ist es, Kandidaten für Komponenten im zukünftigen System aufzufinden. Die Hypothesen enthalten u.a. Aussagen über Abhängigkeiten und Antiabhängigkeiten einzelner Systemteile.
2. Validierung der Hypothesen – die zuvor aufgestellten Hypothesen müssen auf ihre Plausibilität und ihre technische Machbarkeit hin untersucht werden. Dieser Schritt erfordert in der Regel mindestens einen fachlichen und einen technischen Gutachter.

Sinnvollerweise sollten die Hypothesen nicht von demjenigen validiert werden, der sie aufstellt. Vielmehr sollte die Überprüfung analog zu Design-Reviews durchgeführt werden.

Sowohl zum Aufstellen, als auch zur Validierung der Hypothesen stehen verschiedene Hilfsmittel zur Verfügung:

- Das wichtigste Hilfsmittel ist dabei das Know-how der ursprünglichen Systementwickler, oder zumindest der Personen, die das System zur Zeit warten. Oftmals ergibt sich aus Gesprächen mit diesen Personen bereits eine Aufteilung des Systems in geeignete modulare Strukturen.
- Ebenso ist es meist nützlich, die existierenden Subsystemstrukturen zu analysieren. Diese zeigen sich oftmals bereits in der Organisation der Programmquellen.
- Das zentrale technische Hilfsmittel ist die Vermessung des Quellcodes. Dadurch ist es möglich, die Kopplung und Kohäsion von Systemteilen zu ermitteln. Diese Technik ermöglicht es zum einen, schnell einen Überblick über umfangreiche Systeme zu erhalten, zum anderen stellt sie ein wertvolles Mittel zur Validierung von Hypothesen dar: Eine Komponente, die sehr eng mit den umliegenden Systemteilen verwoben ist, lässt sich im allgemeinen nur schwer isolieren.

Komponentifizierung

Sind die Komponenten erst einmal gefunden, dann erfolgt der technisch aufwendigere Schritt der Komponentifizierung – die Auslösung der identifizierten Teile aus dem Altsystem heraus und die Verpackung dieses Teils in eine Komponente. Dabei müssen die technischen Randbedingungen der zu Grunde liegenden Middleware beachtet werden – sei es (D)COM, CORBA oder JavaBeans. Jede dieser Basisplattformen stellt bestimmte Anforderungen an die Komponenten, denen bei der Umsetzung nachgekommen werden muss.

Die Aktivitäten bei der Herauslösung der zukünftigen Komponenten sind:

1. Bildung von Schnittstellen – die Funktionalität, die eine Komponente erbringt, muss durch eine Schnittstelle geeignet beschrieben werden. Die Bildung dieser Schnittstelle ist essentiell für die folgende Aufgabe:
2. Auflösen von Abhängigkeiten – dabei wird dafür gesorgt, dass das Geheimnisprinzip der Komponente eingehalten wird, d.h. es wird sichergestellt, dass eine Komponente mit externen Systemteilen nur über die zuvor gebildete Schnittstelle kommuniziert.
3. Schließlich muss die Komponente sowie das verwendende Umfeld auf die zu Grunde liegende Middleware und die gewünschte Zielarchitektur abgestimmt werden. Erfahrungsgemäß erfordert dieser Schritt ein großes Maß an Erfahrung mit der Technologie, da die Überführung existierender Systemteile in entsprechende Komponenten oftmals ungleich schwieriger ist, als die Entwicklung neuer Komponenten, für die meist gute Werkzeuge existieren.

Fallstudie

Die soeben geschilderte Vorgehensweise haben wir teilweise in einer Fallstudie erprobt. In dieser Fallstudie sollte ein vorgegebenes Subsystem eines umfangreichen, in C++ implementierten Systems in eine wiederverwendbare COM-Komponente überführt werden. Diese COM-Komponente soll dann als Basis für die Entwicklung zusätzlicher Anwendungen zur Erweiterung des Grundsystems in Visual Basic dienen. Zur Bearbeitung der Fallstudie verwendeten wir *Goose* [CIUPKE 1999], eine Sammlung von Reengineering-Werkzeugen zur Visualisierung und Analyse von Altsystemen.

Zunächst haben wir mit Hilfe von Visualisierungen, Metriken und automatisch prüfbaren Heuristiken die Vermutung bestätigt, dass das umzuarbeitende Subsystem bestimmte Charakteristika aufweist, die als Voraussetzung für die Komponentenbildung gelten können.

- Mäßige Kopplung (Abhängigkeiten) zum Rest des Systems.
- Hohe Kohäsion der Klassen innerhalb der Komponente [FAMOOS 1999]
- Nur ein Teil der Klassen des Subsystems bildet die nach außen hin sichtbare Schnittstelle des Subsystems.

In einem weiteren Schritt haben wir die Abhängigkeiten zwischen dem Komponentenkandidaten und dem Rest des Systems ermittelt und klassifiziert. Dabei konnten wir zwei Arten von Abhängigkeiten unterscheiden:

- Einfache Dienstnutzung: Andere Teile des Systems erzeugen Objekte von Klassen des Subsystems und nutzen durch Methodenaufrufe die Funktionalität des Subsystems.
- Nutzung von Basisimplementierungen: Andere Teile des Systems leiten Unterklassen von Klassen des Subsystems ab und erweitern deren Verhalten oder passen es an.

Um das Subsystem in eine COM-Komponente umzuarbeiten, müssen diese Abhängigkeiten auf die Erfordernisse der COM- Komponenteninfrastruktur angepasst werden. Zum einen müssen Klassen, die Dienstgeber im Sinne der einfachen Dienstnutzung darstellen, mit COM-Schnittstellen versehen werden. Zum anderen bietet COM keine Vererbung von Implementierungen [SZYPERSKI 1997]. Dies erfordert die Auflösung der Vererbungsbeziehungen zwischen den Basisklassen im Subsystem und den davon abgeleiteten Klassen in anderen Teilen des Systems. Diese Beziehungen können dann in Delegationsbeziehungen umgewandelt werden [GENSSLER, SCHULZ 1999].

Mit Hilfe von *Goose* konnten wir genau bestimmen, welche Klassen dazu umgearbeitet werden mussten. Diese Information ermöglicht es uns einerseits, den Aufwand für die Komponentifizierung des Subsystems zu bestimmen, andrerseits bildet sie einen guten Ausgangspunkt für die eigentlichen Restrukturierungsmaßnahmen.

Für unsere Fallstudie hat sich beispielsweise ergeben, dass das zu komponentifizierende Subsystem starken Framework-Charakter hat und zahlreiche Basisimplementierungen bereitstellt. Demzufolge müssen zahlreiche Vererbungsbeziehungen umgestaltet werden; der Aufwand dafür ist beträchtlich. Hierbei fällt der bereits erwähnte Mangel an Werkzeugen zur Transformation deutlich ins Gewicht.

Zusammenfassung

In diesem Papier haben wir eine systematische Vorgehensweise bei der Überführung einer Anwendung in ein komponentenorientiertes System vorgestellt. Dieses Verfahren besteht im wesentlichen aus den drei Schritten Festlegung der Zielarchitektur, Finden von Komponenten und der eigentlichen Komponentifizierung.

Teile dieses Verfahrens wurden anhand einer Fallstudie demonstriert. Im Rahmen derzeit laufender Projekte wird die Vorgehensweise auf umfangreiche Systeme mit über 2000 Klassen angewandt. Die bisherigen Ergebnisse scheinen den vorgestellten Prozess zu bestätigen.

Als eines der wichtigsten Probleme wurde das Fehlen von Werkzeugen zur systematischen, semi-automatischen Transformation erkannter Problemstellen identifiziert. Die Entwicklung solcher Werkzeuge ist ebenfalls Inhalt derzeit laufender Projekte.

Literatur

[CIUPKE 1999] Oliver Ciupke, *Automatic Detection of Design Problems in Object-Oriented Reengineering*. .In Proceedings of the *TOOLS USA99*, 1999.

[FAMOOS 1999] FAMOOS Project Team, *The FAMOOS Handbook of Object-Oriented Reengineering*. FZI-Report 1999. Download from: http://www.fzi.de/bauer.html

[GENSSLER, SCHULZ 1999] Thomas Genßler, Benedikt Schulz. *Transforming Inheritance into Composition – A Reengineering Pattern*. In *Proceedings of the 4th European Conference on Pattern Languages of Programming and Computing*, 1999.

[SZYPERSKI 1997] Clemens Szyperski, Component Software – Beyond Object-Oriented Programming. Addison-Wesley 1997.