Werkzeuggestützte Überführung prozeduraler Programme in eine objektorientierte Architektur – Erfahrungen aus der Praxis

von
Harry M. Sneed
Universitäten Passau & Regensburg, Bayern
AneCon GmbH, Wien
Harry.Sneed@T-Online.de

1. Motivation für eine Objektorientierte Migration

Schon seit etlichen Jahren sind die Anwenderbetriebe bemüht ihre alten, prozedurale Anwendungssysteme in eine objektorientierte Architektur zu migrieren. Der Aufbau einer Service Oriented Architektur mit Web Services setzt dies voraus. Große monolitische Programme haben darin keinen Platz. Die Alternativen zur Migration sind dreierlei. Zum Einen kann der Anwender aufgrund einer Nachdokumentation der Altsysteme die Programme in eine objektorientierte Sprache neu entwickeln. Zum Zweiten, kann er die Altprogramme mit Hilfe eines Werkzeuges in eine objektorientierte Sprache transformieren. Zum Dritten kann er die Altprogramme oder Teile davon kapseln und unverändert in die neue OO-Architektur integrieren. Nicht wenige Anwender verbinden die drei Alternativen miteinander – Reimplementierung, Konvertierung und Kapselung. Wichtig ist, die Migration möglichst kostengünstig und risikolos durchzuführen und zwar ohne den laufenden Betrieb zu stören. D.h. man ist gezwungen die Migration schrittweise, inkrementell durchzuführen. Während einige Subsysteme bereits in der neuen Umgebung funktionieren, laufen andere weiterhin in der alten Umgebung. Sie müssen über Schnittstellen mit einander verbunden werden. Die Vorausplanung und Implementierung dieser Schnittstellen ist eine wichtige Voraussetzung für eine erfolgreiche Migration. [Sned99]

2. Objektorientierte Nachdokumentation prozeduralen Anwendungssysteme

Für die erste Alternative, ist es erforderlich, aus dem bestehenden Code, bzw. aus der vorhandenen Dokumentation, ein Objektmodell abzuleiten. Da es fast nie eine vollständige und aktuelle Dokumentation gibt, bleibt nur übrig das Objektmodell aus dem Code zu gewinnen. Dafür hat der Autor ein Werkzeug entwickelt und in einigen Industrieprojekten bereits eingesetzt. Das Tool „*RepoTran*“ setzt auf die Ergebnisse des statischen Analysators „*SoftAnal*“ auf. SoftAnal analysiert den Source Code zahlreicher prozeduralen Sprachen, u.a. Assembler, COBOL, PLI, Natural, ABAP und APS. Neben den üblichen Funktionen eines statischen Analysators – Prüfung, Messung und Spezifikation – erzeugt SoftAnal eine Tabelle binärer Beziehungen zwischen den Programmstrukturelementen – Module, Codeabschnitte, Datenstrukturen, Daten, Masken, Dateien, und sonstigen Schnittstellen. Module beinhalten Codeabschnitte und Datenstrukturen. Sie senden und empfangen Masken, greifen auf Dateien und Datenbanken zu, und rufen andere Module auf. Codeabschnitte verwenden Daten als Prädikate, Argumente und Ergebnisse, und rufen andere Codeabschnitte auf. Diese Beziehungen sind alle in der Tabelle erfaßt.

Um daraus ein Objektmodell zu erzeugen, invertiert RepoTran die Programmhierarchie und stellt die untergeordneten Codeabschnitte, die von mehreren übergeordneten Codeabschnitte aufgerufen werden an der Spitze einer neuen Klassenhierarchie. Die tiefsten Knoten in dem prozeduralen Baum werden zu den Basisklassen im Klassenbaum. Die steuerenden Codeabschnitte an der Spitze der Prozedurhierarchie bilden die untersten Klassen von denen nichts weitervererbt wird. Hinzu kommen Klassen für alle Datenstrukturen. Falls eine Datenstruktur nur in einem Abschnitt verwendet wird, wird sie in der entsprechenden Klasse eingebettet. Sonst, wird sie in eine Superklasse mit allen globalen Daten integriert.

Diese neuen Klassenstrukturen werden anschließend in die SoftRepo Repository geladen und dort zusammengeführt. Ähnliche Klassen, bzw. Klons, werden auf eine reduziert. Überflüssige Daten sowie redundante Funktionen werden entfernt. Dadurch entsteht ein gestrafftes Objektmodell, das durch das Visualisierungswerkzeug RepoView graphisch dargestellt werden. Es werden neben Modul und Klassenbäume auch Klassen-

diagramme, Sequenzdiagramme und Aktivitätendiagramme generiert. Mit Hilfe dieser Diagramme, sind die Entwickler in der Lage die neuen Klassen zu programmieren und zu integrieren. [Sned02]

3. Transformation prozeduraler Programme in objektorientierte Komponente

Falls der Anwender es vorzieht, die Programme direkt 1:1 in OO-Komponente zu transformieren stellt der Autor das Werkzeug „*SoftTran*“ zur Verfügung. SoftTran geht von den Datenstrukturen aus und erstellt eine Klasse für jede Datei, jede Datenbanktabelle, jede Maske, jede Tabelle und jede Arbeitsdatengruppe. Aufgrund einer Datenverwendungsanalyse wird festgestellt welche Codeabschnitte welche Datenobjekte verwenden. Codeabschnitte, bzw. Prozeduren, die eine Datenstruktur benutzen, werden der entsprechenden Klasse als Methode zugeordnet. Sollte ein Codeabschnitt mehrere Datenobjekte verwenden, wird er dupliziert und als Methode allen zugeordnet.

Auf dieser Weise wird als erstes eine neue Programmstruktur mit mehreren abstrakten Datentypen – ADTs – in der bisherigen Sprache erzeugt. D.h., es entsteht eine Zwischenversion, eine objektbasierte Version des ursprünglichen prozeduralen Programmes. Erst in dem zweiten Schritt wird die alte Syntax in die neue Syntax überführt. Aus COBOL-85 Code wird OO-COBOL Code generiert und anschließend Java. Aus PLI Code wird CPP Code generiert. Der generierte Code ist allerdings unvollständig und muß manuell von dem Entwickler vervollständigt und integriert werden. Diese Lösung ist zwar schneller und billiger als die Reimplementierung, liefert aber eine qualitativ minderwertigere Lösung, die eigentlich nur objektbasiert ist.

4. Kapselung prozeduraler Programme als Objekte

Eine dritte Alternative ist die der Kapselung. Sie ist die aller billigste und schnellste, denn der alte Code wird nur minimal verändert. Es geht darum neue Schnittstellen zu den alten Programmen zu schaffen. Aus den Masken, Dateien und Call-Schnittstellen werden XML Schnittstellen kreiert. Statt von einem TP-Monitor, einem Dateiverwaltungssystem oder einem anderen Programm gefüttert zu werden, empfangen die gekapselten Programmen ihre Eingaben von einem XML Wrapper. Ihre Ausgaben geben sie auch an ein anderes XML Wrapper ab. Somit können sie als Geschäftsobjekte in jede beliebige Umgebung wiederverwendet werden.

Das Tool „*SoftWrap*“ dient dazu, den Source Code der Kapselkandidaten zu verändern und die XML Schnittstellen zu generieren. Für jede Programmeingabe wird eine Eingabeschnittstelle und für jede Programmausgabe eine Ausgabeschnittstelle mit einem XML Schema erzeugt. Die Schemen beschreiben die Struktur und den Inhalt der Schnittstellen. Gleichzeitig wird für jede Schnittstelle eine Treiberklasse erzeugt, die jene Schnittstelle bedient. Bei den Eingabeschnittstellen werden die Aufträge, bzw. Requests, umgesetzt und dem gekapselten Programm zugeführt. Bei den Ausgabeschnittstellen werden die Antworten, bzw. Responses, umgesetzt und dem Clientprogramm gesendet. Auf dieser Weise können alte Programme oder Prozeduren als Web Services wiederverwendet werden. [Sned03]

5. Objektorientierte Softwaremigration in der Praxis

Trotz der Verfügbarkeit und Praktizierbarkeit der obengenannten Ansätze, werden sie in der industriellen Praxis nur selten angewendet. Die meisten Anwender ziehen es vor, ihre alte Software so zu lassen wie sie ist, um sie irgendwann durch Standardanwendungssysteme zu ersetzen. Andere, die alte Systeme durch neue ablösen, beginnen mit einem völlig anderen Konzept und entwickeln die neuen Systeme vom Grund auf neu. Nur die wenigsten versuchen ihre Altsysteme zu migrieren. Es wäre interessant zu erforschen, warum dies so ist.

Literatur

[Sned99] Sneed, H.: Objektorientierte Softwaremigration, Addison-Wesley Verlag, Bonn, 1999

[Sned02] Sneed, H.: „Transforming procedural program structures to object-oriented class structures“, Proc of IEEE-ICSM-2002, IEEE Computer Society Press, Montreal, 2002, p. 286

[Sned03] Sneed, H.: „Web-basierte Systemintegration, Vieweg Verlag, Wiesbaden, 2003