Migration prozeduraler Anwendungssysteme in eine objektorientierte Architektur

von

Harry M. Sneed (MPA)
Case Consult, Wiesbaden
eMail: Harry.Sneed@CaseConsult.com

Abstrakt:

Der folgende Beitrag fasst die bisherige Forschungsarbeit zum Thema „objektorientierte Softwaremigration“ zusammen und erläutert die Gründe, warum es zu dem erwarteten Erfolg noch nicht gekommen ist. Die Forscher haben sich nur mit der Ableitung abstrakter Objektmodelle aus dem alten Source befasst, während die Praktiker sich nur mit einer Source-zu-Source-Transformation beschäftigt haben. Weder das eine noch das andere führt zu einem annehmbaren Ergebnis. Der Autor schlägt einen dritten Weg vor, der die Architektur von der Implementierung und die Daten von den Funktionen trennt. Obwohl dies gegen die klassische objektorientierte Lehre verstößt, verspricht dieser komponentenbasierte Reengineering-Ansatz mit der Verwendung von Wrapping-Technologie und zustandsloser Klassen eine praktikable Lösung zum Problem der objektorientierten Softwaremigration zu werden.

1. Bisherige Forschungsansätze zur Ableitung von Objekten aus prozeduralem Code

Zum Thema der Überführung prozeduraler Programme in eine objektorientierte Architektur gibt es seit Anfang der 90er Jahre immer mehr Beiträge sowohl theoretischer als auch praktischer Art. Einer der allerersten Ansätze kam vom Autor selbst und befasste sich mit der Migration prozeduraler COBOL-Programme in eine objektorientierte Architektur. Der REORG-Ansatz, der im Rahmen des ESPRIT-Forschungsprojektes DOCKET entwickelt wurde, war ein Prozess mit 10 Schritten, um Objekte mittels einer Datenflussanalyse aus den COBOL-Programmen abzuleiten und in eine OO-Repository abzulegen, aus der später Klassen generiert wurden. Das Ergebnis waren OO-Klassen, die im Rahmen einer Neuentwicklung wiederverwendet werden konnten. [Sne 92]

Diese Forschungsarbeit wurde an ein weiteres ESPRIT-Forschungsprojekt angelehnt – das REDO-Projekt –, in dem Breuer und Lano ein Reverse-Engineering-Verfahren entwickelten, um aus alten COBOL-Programmen eine Z++-Spezifikation abzuleiten. [Breu 91] In dieser formalen Spezifikation wurden COBOL-Dateien, -Sätze und –Tabellen als Objekte und alle Zugriffsoperationen auf sie als Methoden dargestellt. Das daraus resultierende Objektmodell sollte als Basis für eine mögliche Neuentwicklung dienen.

Der Autor hat seine Forschungsarbeit fortgesetzt und drei Ansätze zur Zerlegung monolithischer Programme konzipiert – einen prozeduralen Ansatz basierend auf der Cluster-Analyse der Programmablaufgraphen, einen funktionalen Ansatz basierend auf der fachlichen Funktionsauflösung und einen objektorientierten Ansatz basierend auf der Datenverwendung. Alle drei Ansätze wurden in einem Migrationsprojekt der Firma Wella vom Host zu AS 400 in den Jahren 1993/94 ausprobiert. [Sne 94]

Eine intensive Beschäftigung mit der Umsetzung prozeduraler Programme in ein Objektmodell begann erst ab 1996. Im Anschluss an die Pionierarbeiten des Autors haben zwei Mitarbeiter von Nortel-Telecom ein Verfahren – SCORE/RM – entwickelt, um ein Objektmodell aus alten FORTRAN-Codes zu gewinnen. Sie setzten einen kooperativen Prozess zwischen Werkzeug und Entwickler ein zur Identifikation von Objekten und Methoden aufgrund der Datenverwendung. Schritt für Schritt wurden Objekte und Funktionen

in neue Klassenrahmen herübergezogen. Danach wurden die Klassen in eine Klassenhierarchie geordnet. Zum Schluss ergab sich ein Objektmodell mit über- und untergeordneten Klassen samt Attributen und Methoden. Diese wurde wiederum als Basis für die Reimplementierung des Codes in C++ verwendet. [Byrne 96]

Eine weitere Variante der objektorientierten Migration präsentierte der Autor auf der Reverse-Engineering-Konferenz in Monterey im Jahre 1996. Zu diesem Zeitpunkt hatte der Autor bereits ein voll automatisiertes, werkzeuggestütztes Verfahren entwickelt, um COBOL-74- bzw. COBOL-85 Programme in OO-COBOL zu transformieren. Das Verfahren umfasste fünf Schritte

- Objektauswahl,
- Operationsextraktion,
- Attribute- und Funktionsvererbung,
- Eliminierung von Redundanz und
- Syntaxkonversion.

Am Ende der Prozesskette kamen kompilierbare OO-COBOL-Klassen heraus. [Sne 96]

In deutscher Sprache erschien 1995 die erste Dissertation über objektorientiertes Reverse Engineering von Klösch und Gall an der Universität Wien. Sie beschrieb ein umfangreiches Verfahren basierend auf zahlreichen Fallstudien für die Generierung eines statischen objektorientierten Anwendungsmodells aus prozeduralen C-Sourcen. Das Verfahren mit dem Namen Capsule-oriented Reverse Engineering Method (COREM) erzeugte zunächst die klassischen SSD-Diagramme – Structure Charts, Data Flow Diagrams und Entity/Relationship Diagrams – aus dem Source Code und den Datenbankschemen. Anschließend wurden die Entitäten in Objekte, die Datenflüsse in Schnittstellen und die Prozeduren in Methoden umgewandelt. Das resultierende Objektmodell müsste zwar manuell verfeinert werden, bietet jedoch eine Basis für eine objektorientierte Neuentwicklung mit der Wiederverwendung einzelner alter Codeabschnitte. Interessant am Ansatz von Klösch und Gall ist, dass sie sich darauf beschränkten, einen Entwurf für die Reimplementierung zu

schaffen und nicht dem Versuch unterlagen, gleich einen neuen Code zu generieren. [Klösch 95]

Einem ähnlichen Ansatz zum Problem der Objektextraktion folgte ein italienisches Forschungsteam aus Neapel. Sie stellten persistente Data Stores – also Stammdaten und Datenbanken – in den Mittelpunkt der Modellierung und benutzten sie als Objkete, zu denen die IO und Verarbeitungsanweisungen zugewiesen wurden. [DeLucia 97]

An der Universität Aachen schrieb K. Cremer eine Dissertation zum Thema „Objektorientiertes Redesign von Legacy-Anwendungen". Während Klösch und Gall sich mit C-Programmen für die Prozesssteuerung befassten, befasste sich Cremer mit COBOL-Programmen aus der kommerziellen Datenverarbeitung. Ihr PROGRESS-Verfahren zielte daraufhin, die bestehenden COBOL-Abschnitte als Methoden wiederzuverwenden, die gemeinsamen Datengruppen als Objekte zugewiesen wurden. [Crem 98]

Zur gleichen Zeit entstand an der DeMontfort-Universität in England ein Ansatz, der alle globalen Daten in einer gemeinsamen Basisklasse sammelte – der Gottklasse –, aus der alle weiteren Klassen geerbt haben. Die untergeordneten Klassen enthalten nur die Methoden und die lokalen Variablen. Die Methoden entsprechen den Prozeduren in den bisherigen C-Progammen. [Luker 98]

2. Kommerzielle Ansätze zur Transformation prozeduraler in objektorientierte Programme

Alle erwähnten Forschungsansätze und mehr wurden in dem Buch „Objektorientierte Softwaremigration", von diesem Autor erschienen im Jahr 1999, zusammengefasst. [Sne 99] Es folgten bis Ende 2000 noch weitere Vorschläge für die Gewinnung von Objekten aus prozeduralem Code. Danach ebbte die Forschung ab und es folgten die ersten kommerziellen Werkzeuge, allen voran die Produkte ROCOCO von der GMD [GMD 99] und Rescue Ware von der Firma Relativity [CW 98]. Während ROCOCO aus altem COBOL-Code OO-COBOL erzeugt, generiert Rescue Ware als altem COBOL-Code neue JAVA-Klassen. In beiden

Fällen entsprechen die Klassen bisherigen Codeabschnitten, also ist das sogenannte Objektmodell von dem bestehenden Funktionsmodell nicht weit entfernt.

Was letztendlich aus der Source-zu-Source-Transformation herauskommt, ist das alte Programm in einer neuen Syntax. Die Objekte sind die bisherigen Datenstrukturen, die Methoden sind die bisherigen Prozeduren. Man betreibt einen großen Aufwand, um den alten Inhalt in einer nur quasi neuen Form zu bekommen. Die daraus resultierenden Klassen sind nicht besser als die Module, aus denen sie abgeleitet wurden. Für OO-Programmierer sind sie ebenso unverständlich wie für die konventionellen Programmierer. Sie bleiben ein Zwitter zwischen den beiden Programmiermethoden.

Der Grund dafür ist offensichtlich: Objekte sind semantische Elemente aus dem Anwendungsmodell – Elemente wie Artikel, Konten, Verträge usw. Diese Objekte werden in Programmen verarbeitet, meistens mehrere auf einmal. Die Anweisungen, die die Objekte verändern bzw. ihre Zustandsübergänge regeln, sind kreuz und quer vermischt. Anweisungen, die völlig andere Objekte verarbeiten, sind in einem prozeduralen Zweig zusammen und werden von einer Entscheidung angesteuert. Wenn man die Anweisungen nach Objekten neu ordnet, muss man die Steuerungsstruktur für jedes Objekt duplizieren. Dann gäbe es praktisch die gleiche Kontrollstruktur in jeder Klasse. Eine Klasse wäre die Summe aller Teilprogramme für ein bestimmtes semantisches Element. Dies führt zu einer exponentiellen Aufblähung des Codes, der danach unwartbar wäre. Auch wenn es gelingt, echte objektorientierte Klassen aus den prozeduralen Programmen zu gewinnen, ist der Code daher nicht mehr handhabbar.

Deshalb bleibt nichts anderes übrig, als den alten Code weitgehend so zu lassen wie er ist, nur zu zerstückeln und die Stücke als Methoden zu definieren. Demnach ist das Objekt die Summe aller Daten, die von allen Methoden einer Klasse verarbeitet werden. Die Klasse ist equivalent zum alten Programm. Dies ist jedoch weit entfernt vom Ziel der Objektorientierung, denn nun werden die gleichen Attribute und die gleichen Methoden in etlichen Klassen auftreten, so dass die sogenannten objektorientierte Lösung qualitativ schlechter ist als die bisher prozedurale Lösung. Außerdem sind die Codemaße noch

umfangreicher. Es darf nicht wundern, wenn die bisherigen kommerziellen Ansätze abgelehnt werden.

3. Komponentenbasiertes Reengineering als Alternativstrategie

Eine Alternative zu der Source-Transformation ist, aus jedem Codeabschnitt, CSECT in ASSEMBLER, interner Prozedur in PL/I und Section in COBOL eine eigene Klasse zu machen. Die Methoden sind die labeled Codeblöcke bzw. Paragraphen insofern, als es sie überhaupt gibt. Attribute gibt es nicht, sondern nur Parameter. Das heißt, die Klassen haben keine eigenen Daten, sie sind zustandslos. Objekte werden von übergeordneten Klassen initialisiert, verwaltet und aufbewahrt. Die Objektveränderung wird aber an die untergeordneten Klassen delegiert, die die Objekte nur als Schnittstellen zu sehen bekommen. [Sne 00]

Die Vorteile dieser Methode sind, dass es möglich wird, ein neues objektorientiertes Framework zu implementieren oberhalb der alten Programme. Die Architektur dieses Frameworks ist unabhängig von der Struktur der bisherigen Anwendungsprogramme. Sie orientiert sich nach den Anforderungen einer komponentenbasierten Entwicklung. Zwischen dieser übergeordneten Superarchitektur und den darunter liegenden Altprogrammen bzw. Altklassen befindet sich ein Wrapper oder eine Zugriffsschicht, die die Aufträge der übergeordneten Systeme in Schnittstellen zu den Altprogrammen umsetzt. Hier werden Datentypen konvertiert und Daten neu geordnet, damit sie zu den Parametern des alten Codes passen.

Auf diese Weise lässt sich der alte prozedurale Code unverändert weiter verwenden. Nur der Datenteil wird restrukturiert. Alle globalen Daten, einschließlich aller Masken und Kommunikationsbereiche, werden in Parameter umgewandelt. Möglicherweise kann die Schnittstelle mit allen Daten der Zielklasse als XML-Dokument oder als IDL-Schnittstelle implementiert werden. Auf jeden Fall gilt es, im Gegensatz zur klassischen Objektorientierung, die Daten von der Methoden zu trennen und die alten Programmabschnitte als zustandslose Klassen zu betrachten.

Die übergeordneten Klassen im neuen Komponenten-Framework haben demnach zwar Attribute, aber keine Methoden. Ihre Methoden sind lediglich Proxy-Methoden, welche die untergeordneten echten Methoden in den Altprogrammen über die Wrapper aufrufen und dabei die Objekte übergeben. Damit haben die Entwickler des Frameworks die Wahl, Methoden gleich in den Framework-Klassen einzubauen oder die Methoden der untergeordneten Altklassen zu verwenden.

Am Anfang wird es so sein, dass die Verarbeitung der Objekte weitgehend an die untergeordneten Klassen delegiert wird. Die Framework-Klassen beschränken sich darauf, die Objekte zu bilden und zu verwalten. Im Laufe der Zeit können immer mehr alte Methoden in den neuen Klassen reimplementiert werden. Aber, und dies ist ausschlaggebend, es muss nicht alles auf einmal reimplementiert werden. Man hat Zeit.

Diese evolutionäre Vorgehensweise passt mehr zur Organisation der IT in konventionellen Anwendungsbetrieb. Auf der einen Seite gibt es eine junge Mannschaft, die moderne objektorientierte und komponentenbasierte Methoden und Sprachen beherrscht. Sie kann für die Entwicklung des übergeordneten Komponenten-Frameworks – der Frontend – zuständig sein. Sie ist damit auch für die Implementierung der Client-Software verantwortlich. Auf der anderen Seite gibt es eine ältere Mannschaft, die die bisherigen Methoden und Sprachen beherrscht. Sie kann für die Wartung und Weiterentwickung der untergeordneten Backend-Klassen zuständig sein. Außerdem ist sie für die Zugriffe auf die bestehenden Dateien und Datenbanken verantwortlich.

Dazwischen gibt es die Umsetzungsschicht oder Wrappers, die im Sinne der Middleware von Kommunikationsspezialisten aufgebaut und gepflegt wird. Sie sind für die Schnittstellen zwischen Frontend und Backend sowie für die Umsetzung der Objekte zuständig. Ihre Funktion wird natürlich auslaufen in dem Maße, wie die alten Backend-Methoden und neuen Frontend reimplementiert werden. Es wird aber noch lange eine Zugriffsschnittstelle zu den alten Datenbanken bestehen bleiben.

Erste Erfahrungen mit dieser komponentenbasierten Reengineering unter Verwendung zustandsloser Backend-Klassen zeigen, dass dieser Ansatz ein sinnvoller und praktikabler Weg zu einer objektorientierten Software-Migration sein könnte.

Literaturhinweise

[Sne 92] Sneed, H.: „Migration of procedurally-oriented COBOL Programs in an object-oriented Architecture“, in Proc. of IEEE ICSM-92, IEEE Press, Orlando, Nov. 1992, p. 105 to 116

[Breu 91] Breuer/Lano: „Creating object-oriented Specifications from Code Reverse Engineering Techniques“, in Journal of Software Maint., Vol. 3, No. 3, Sept. 1991

[Sne 94] Sneed, H.: „Downsizing large Application Programs“, in Journal of Software Maint., Vol. 6, No. 5, Sept. 1994

[Byrne 96] Byrne/Subramaniam: „Deriving an Object Model from Legacy FORTRAN Code“, in Proc. of ICSM-96, IEEE Press, Monterey, Nov. 1996, p. 3 to 12

[Sne 96] Sneed. H.: „ Object-oriented COBOL Recycling“, in Proc. of IEEE WCRE-96, IEEE Press, Monterey, Nov. 1996, p. 169 to 178

[Klösch 95] Klösch, R./Gall, H.: „Objekt-orientiertes Reverse Engineering“, Springer Verlag, Berlin, 1995

[DeLucia 97] DeLucia/Lucca/Fasolini: „Migrating Legacy Systems towards object-oriented Platforms“, in Proc. of IEEE ICSM-97, IEEE Press, Bari, Oct. 1997. p. 114 to 121

[Crem 98] Cremer, K.: „A Tool supporting the Redesign of Legacy Applications“, in Proc. of European Conference on Software Maintenance and Reengineering, IEEE Press, Florence, March 1998, p. 142 to 150

[Luker 98] Luker, P./Zedan, H.: „Conceptual Foundations for the Design Transformation of procedural Software to object-oriented Architecture“, in Proc. of IEEE ICSM-98, IEEE Press, Washington, Nov. 1998, p. 284 to 294

[Sne 99] Sneed, H.: „Objekt-orientierte Software-Migration“, Addison-Wesley Verlag, Bonn, 1999, p. 167 to 192

[GMD 97] Gesellschaft für Mathematik und Datenverarbeitung: Bericht des GMD-Forschungszentrums Informationstechnik, Sankt Augustin, Juli 1997

[CW 98] Computerwoche: „COBOL- zu JAVA-Umsetzung“, in CW Nr. 4, München, Jan. 1998

[Sne 00] Sneed, H.: „Generation of stateless Components from Procedural Programs for Reuse in Distributed Systems“, in Proc. of European Conference on Software Maintenance and Reengineering, IEEE Press, Zürich, March 2000, p. 183 to 189