Applikationswissen in der Sprachkonvertierung am Beispiel des COBOL-Java-Converters CoJaC

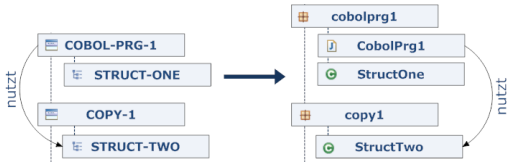
Christian Becker, Uwe Kaiser
pro et con Innovative Informatikanwendungen GmbH, Dittesstraße 15, 09126 Chemnitz
christian.becker@proetcon.de, uwe.kaiser@proetcon.de

Abstract

Die Firma pro et con hat in der Vergangenheit einen **CO**BOL to **Ja**va **C**onverter (CoJaC) entwickelt. CoJaC konvertiert aus einem einzelnen, vollständigen COBOL-Programm einschließlich der Copy-Books ein Java-Programm mit zugehöriger Package-Struktur. Kontextinformationen aus der Gesamtapplikation blieben bei dieser Arbeitsweise unberücksichtigt. Der vorliegende Beitrag beschreibt ein Verfahren, programmübergreifendes Applikationswissen, welches durch eine werkzeugbasierte Analyse gewonnen und während der automatischen Sprachkonvertierung verarbeitet wird, zur Optimierung des Ziel-Codes einzusetzen.

1 Ausgangssituation

In kommerziellen Programmen werden häufig COBOL-Strukturen in Copy-Books ausgelagert, um redundanten Code zu reduzieren und die Architektur des Programmsystems durch weitere Strukturierung übersichtlicher zu gestalten. CoJaC konvertiert einzelne, vollständige COBOL-Programme einschließlich der dazugehörigen Copy-Books 1:1 in semantisch äquivalente Java-Programme[1] und ordnet diese, wie in Java üblich, in einer hierarchischen Java-Package-Struktur (*Packages*) ein.



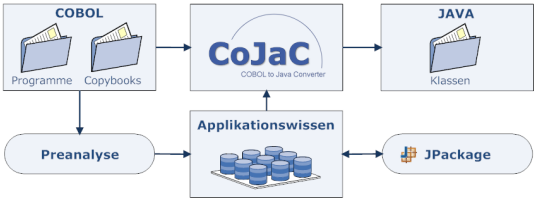
Die Grafik zeigt die Abbildung von COBOL-Programmen und -Strukturen auf Java-Klassen und -Packages. Das Programm *COBOL-PRG-1* wird nach der Transformation in einem gleichnamigen Package *cobolprg1* eingeordnet. Die enthaltenen COBOL-Datenstrukturen (hier *STRUCT-ONE*) werden als eigenständige Java-Klassen transformiert (*StructOne*) und im selben Package (*cobolprg1*) eingebunden. Die Umsetzung von Copy-Books einschließlich der darin enthaltenen COBOL-Strukturen erfolgt analog. Die COBOL-Struktur *STRUCT-TWO*, welche im Copy-Book *COPY-1* enthalten ist, wird in eine äquivalente Java-Klasse *StructTwo* transformiert und im Package *copy1* eingeordnet. Alle Java-Programme (hier *CobolPrg1*), welche zuvor Copy-Books verwendeten (*COPY-1*), nutzen nun die nur einmal existierenden, korrespondierenden Java-Klassen (z.B. *copy1.StructTwo*). Auf diese Weise wird bei der werkzeuggestützten Sprachkonvertierung eine zum COBOL-System äquivalente Hierarchie in Java aufgebaut. Zum Zeitpunkt der Konvertierung mit CoJaC stehen nur die Informationen zur Verfügung, welche aus dem zu konvertierenden COBOL-Programm und den verwendeten Copy-Books ermittelt werden können. Duplikate bzw. Klone im COBOL-Code, welche nicht durch den Einsatz von Copy-Books eliminiert wurden, führen durch die 1:1-Migration ebenfalls zu Duplikaten im generierten Java-Code. Die tatsächliche Anzahl dieser Klone kann nicht pauschal beziffert werden, da sie ausschließlich von Architektur und Programmierstil abhängt.
Bei der Migration von Kundensourcen wurde festgestellt, dass die Aufteilung von COBOL-Strukturen auf Copy-Books nicht immer optimal vom Kunden durchgeführt wurde. Der umfangreiche Einsatz von Copy-Books zur Eliminierung aller Duplikate würde deren Anzahl stark erhöhen und somit die Übersichtlichkeit des gesamten Programmsystems beeinträchtigen. Die Firma pro et con erhielt z.B. Programmpakete, in denen keine Copy-Books enthalten bzw. in denen diese bereits aufgelöst waren. Nach einer Konvertierung mit CoJaC enthielt der generierte Java-Code viele identische Klassen. Solche führen ihrerseits zu einer erhöhten Anzahl von Lines of Code (*LOC*) und somit implizit zu einem erhöhten Wartungsaufwand des migrierten Programmsystems.

2 Preanalyse

Bei kommerziellen Migrationsprojekten verursachte die beschriebene Arbeitsweise des CoJaC einen hohen manuellen Aufwand bei der Realisierung einer redundanzfreien Java-Package-Struktur. Daraus resultierte die Notwendigkeit einer weiteren Automatisierung des Migrationsprozesses mit folgenden Zielen:

- *Verringerung des Analyseaufwandes des gesamten Programmsystems*
- *Minimierung des Aufwandes zur Erstellung einer homogenen Java-Package-Struktur*
- *Beseitigung von Redundanzen*

Ausgehend von der genannten Zielsetzung wurde dem Migrationsprozess eine programmübergreifende Preanalysephase (*Preanalyse*) vorangestellt. Diese erweitert den CoJaC um Applikationswissen des zu migrierenden Programmsystems. Die Stellung der Preanalyse im Sprachkonvertierungsprozess dokumentiert die nachfolgende Abbildung:



Die Preanalyse analysiert alle COBOL-Programme inklusive der Copy-Books. Sie arbeitet auf Basis von abstrakten Syntaxbäumen (*AST*) und nutzt bereits vorhandene Metawerkzeuge (z.B. COBOL-Frontend), welche auch bei der

Entwicklung des CoJaC zum Einsatz kamen. Durch Verarbeitung der AST werden Informationen z.B. über den Aufbau und die Verwendung von COBOL-Datenstrukturen im gesamten Programmsystem gesammelt. Das Ergebnis, ein programmübergreifendes Applikationswissen, wird im Verlauf des Migrationsprojektes durch weitere Werkzeuge weiterverarbeitet. Die Preanalyse bietet auch die Möglichkeit, vom Kunden nachträglich gelieferte bzw. veränderte Programme inklusive Copy-Books zu analysieren und zu einem bestehenden Applikationswissen hinzuzufügen respektive zu aktualisieren. Dies ist dann der Fall, wenn der Kunde im Verlauf eines Projektes Sanierungsarbeiten durchführt und die betroffenen Quellen erst nachträglich liefert. Solche Nachlieferungen sind gängige Praxis.

3 Applikationswissen in der Sprachkonvertierung

Eine Anwendung des Applikationswissens ist die programmübergreifende Erkennung und Beseitigung von Duplikaten in Java-Packages, die aus Datenstruktur-Klonen in den COBOL-Programmen resultieren. Für den Konvertierungsprozess sind dabei *Typ 1*- und eingeschränkt auch *Typ 2*-Klone relevant. Es wird das *Baxter-Verfahren*[2], welches auf abstrakten Syntaxbäumen basiert, angewendet. Zu jedem Knoten (entspricht einem Teilbaum) des AST wird ein Hashwert berechnet. Da identische Teilbäume identische Hashwerte liefern, können Klone durch einen Vergleich aller Hashwerte gefunden werden.
Die Reduktion von Duplikaten erfolgt durch Mapping von identischen COBOL-Datenstrukturen auf eine einzige Java-Klasse in einem einzigen Package. Zur Minimierung des Aufwandes wurde für diesen Mapping-Prozess ein Werkzeug zur teilautomatischen Erstellung und Pflege der Package-Struktur namens *JPackage* entwickelt. JPackage stellt einen COBOL-Baum (enthält COBOL-Datenstrukturen der gesamten Applikation) und einen Java-Baum (entspricht der korrespondierenden Java-Klassenhierarchie) gegenüber. Durch die Manipulation beider Bäume lassen sich Klone reduzieren und der Aufbau der Zielstruktur mit geringem Aufwand überarbeiten. Spätere Nachlieferungen geänderter Kundensourcen werden ebenfalls berücksichtigt. Diese Technologie führt zu einer deutlichen Reduktion des Aufwandes gegenüber der manuellen Pflege der resultierenden Java-Package-Struktur.
Nachfolgend werden die Ergebnisse anhand von zwei sehr unterschiedlichen COBOL-Programmpaketen dokumentiert. Beide Pakete wurden einerseits mit den Standardeinstellungen des CoJaC ohne Preanalyse und andererseits mit einer Optimierung durch Preanalyse und JPackage konvertiert. Die erste Variante ermöglicht die Eliminierung von Klonen in einem Migrationsprojekt nur durch Sanierung des COBOL-Codes oder durch nachträgliches Refactoring des Java-Codes. Beides erfordert Aufwand, der sich potenziert, wenn Konvertierungen von COBOL-Programmen mehrfach durchgeführt werden müssen. Die zweite Variante entspricht der optimalen Reduktion von Duplikaten im Konvertierungsprozess mit JPackage.
Das Paket *Legacy 1* besteht aus 32 Programmen, nur einem einzigen Copy-Book und insgesamt 157.000 LOC.

Legacy 1	Standard	Preanalyse	Reduktion
Java LOC	325.000	244.000	25%
Java-Klassen	946	565	40%

Da nur ein Copy-Book verwendet wurde, hat sich die Anzahl der LOC in Java nach einer Konvertierung mit den Standardeinstellungen des CoJaC gegenüber dem originalen COBOL-Code verdoppelt. Nach einer Reduktion der Klone mit der Preanalyse und JPackage konnten die LOC um ca. 25% von 325.000 auf 244.000 reduziert werden. Die Anzahl der Klassen verringerte sich um ca. 40% von 946 auf 565.
Das Paket *Legacy 2*, welches bereits in COBOL gut strukturiert war, bestand aus insgesamt 44 Programmen, 216 Copy-Books und 196.000 LOC. Es diente zur Verifikation des Ansatzes.

Legacy 2	Standard	Preanalyse	Reduktion
Java LOC	331.000	155.000	53%
Java-Klassen	521	282	45%

Gegenüber einer Konvertierung ohne Preanalyse konnten 53% der LOC reduziert werden. Die Reduktion der Klone führte zu einer Verringerung der Java-Klassen um 45% von 521 auf 282. Dieses Ergebnis zeigt, dass auch gut strukturierte Programmsysteme mit vielen Copy-Books Duplikate enthalten und in der Programmtransformation von Preanalyse und Klone-Reduktion profitieren.

4 Zusammenfassung

Zusammenfassend kann gesagt werden, dass die Erweiterung des Migrationsprozesses um eine werkzeugbasierte Preanalysephase sinnvoll ist und den CoJaC um programmübergreifendes Applikationswissen ergänzt. Bei pro et con bereits vorhandene Metawerkzeuge ermöglichten eine einfache Realisierung und Integration in die bestehende Werkzeugkette. Basierend auf den Informationen des Applikationswissens und deren Verwendung in JPackage konnte bei der Erstellung der Zielstruktur (Java-Packages) und der Reduktion von Duplikaten eine nennenswerte Zeitersparnis und Verbesserung des generierten Java-Codes erreicht werden.

Literaturverzeichnis

[1] Erdmenger, U.; Uhlig, D.: Ein Translator für die COBOL-Java-Migration. 13. Workshop Software-Reengineering (WSR 2011), 2.-4. Mai 2011, Bad Honnef. In: GI-Softwaretechnik-Trends, Band 31, Heft 2, ISSN 0720-8928, S. 73-74

[2] Koschke R.: Software Reengineering Grundlagen der Softwareanalyse und -transformation. 3. November 2011. Vorlesungsunterlagen der Universität Bremen (eBook), S. 154-158