Jedes COBOL-Java-Migrationsprojekt birgt neue Überraschungen

Uwe Erdmenger, Uwe Kaiser
pro et con Innovative Informatikanwendungen GmbH, Reichenhainer Straße 29a, 09126 Chemnitz
{uwe.erdmenger,uwe.kaiser}@proetcon.de

Abstract

Der Wunsch nach Modernisierung von Legacy-Software ist aktuell ungebrochen [1]. Die toolgestützte Software-Migration hat sich dabei als eine Technologie für diese Modernisierung etabliert. Werkzeuge für die Software-Migration weisen heute einen beachtlichen Automatisierungsgrad auf. Der von pro et con entwickelte COBOL-Java-Converter CoJaC konvertiert z.B. mehr als 90 % aller COBOL-Programme semantisch äquivalent nach Java. Der vorliegende Beitrag beschreibt ausgewählte Aspekte eines Migrationsprojektes bei der SüdLeasing GmbH. Die SüdLeasing GmbH ist eine der größten Leasing-Gesellschaften Deutschlands und agiert im Verbund der LBBW-Gruppe. Das zu migrierende Legacy-System beinhaltete 1.500 COBOL-Programme mit ca. 2,2 Millionen Codezeilen. Diese wurden mit CoJaC automatisiert nach Java konvertiert. Im Zielsystem ersetzt ein Spring-Boot-Server die proprietäre Middleware des originalen Systems. Die Benutzeroberfläche bestand aus 1.300 ASCII-orientierten Bildschirmmasken, welche über Messages mit den COBOL-Programmen kommunizierten. Die Maskenmigration war nicht Projektbestandteil. SüdLeasing entwickelte dafür mit Angular Weboberflächen identischer Funktionalität, welche an den Spring-Boot-Server angebunden wurden. Das Projekt wurde im geplanten Zeitraum abgeschlossen. Auch in diesem Migrationsprojekt existierten neue Herausforderungen, welche die aktuellen Migrationstechnologien und -werkzeuge noch nicht unterstützten. Dieser Beitrag beschreibt einige der teils unkonventionellen Lösungen.

1 COBOL ist nicht gleich COBOL

Die COBOL-Programme des Legacy-Systems lagen nicht in einem COBOL-Standarddialekt vor. Vielmehr wurde eine COBOL-Erweiterung verwendet, welche zusätzliche Sprachkonstrukte für die strukturierte Programmierung bereitstellt. Das folgende Codefragment zeigt exemplarisch die Definition einer Prozedur (`#entry`) sowie eine `#while`- und eine `#if`-Anweisung und den Prozeduraufruf mit `#pass`:

```
VERARBEITUNG #entry
   initialize v01daten.
   #while not ende-loop #do
      #pass LESE-URTV.
      #if fcode not = zero #then
         #pass LESE-UD0.
      #bend
   #bend
#end
```

Ein Precompiler *precob* übersetzt diesen Code in den Micro-Focus-Dialekt. Dabei werden die strukturierten Anweisungen in Folgen von `if`-Anweisungen und `GO-TO`-Sprüngen konvertiert. Diese COBOL-Programme sind für eine toolgestützte Migration nach Java ungeeignet. Es entsteht durch die große Anzahl von `GO-TO`-Anweisungen schwer wartbarer Java-Code im Zielsystem. Eine Erweiterung des CoJaC um den originalen Dialekt wurde geprüft, aber verworfen. Der zu realisierende Aufwand hätte in keinem Verhältnis zur Nutzung in einem einzelnen Projekt gestanden. Alternativ wurde folgende Lösung gewählt: Es wurde ein neuer Precompiler *precob2* entwickelt. Dieser erzeugt im Gegensatz zu *precob* strukturierten Standard-COBOL-Code, der von CoJaC ohne weitere Anpassungen nach Java konvertiert werden kann. Das oben beschriebene Codefragment liefert nach der Bearbeitung durch *precob2* das nachfolgende Ergebnis:

```
VERARBEITUNG SECTION.
   initialize v01daten
   PERFORM WITH TEST BEFORE
           UNTIL NOT (not ende-loop)
      PERFORM LESE-URTV
      IF fcode not = zero
         PERFORM LESE-UD0
      END-IF
   END-PERFORM.
```

Dieses Vorgehen ist auch auf andere Migrationsprojekte übertragbar. Insbesondere in den Anfangszeiten von COBOL, als die Sprache selbst noch keine strukturierte Programmierung unterstützte, wurden eine Reihe von „Vorübersetzern“ entwickelt, um diese Funktionalität bereitzustellen. Da diese heute nur noch vereinzelt im Einsatz sind, lohnt es nicht, neue Konvertierungswerkzeuge zu entwickeln oder existierende anzupassen, welche diese Programme direkt nach Java konvertieren. Hier ist ein Precompiler die bessere und kostengünstigere Wahl. Gleiches gilt auch für die Behandlung eingebetteter Sprachen wie bspw. die Transaktionssteuerung mit CICS von IBM oder `ENTER-TAL`-Anweisungen von HPE. Auch dafür existieren spezielle Precompiler, welche z.B. bei CICS die Transaktionsbefehle in `CALL`-Befehle von COBOL übersetzen. Bei der Konvertierung nach Java werden diese `CALL`-Befehle zu Aufrufen einer Bibliothek für die Transaktionssteuerung im Zielsystem.

2 Durchblick ordnet Durcheinander

Eine Aufteilung des Gesamtprojektes in Pakete hat sich bei vergangenen Projekten als erfolgreiche Migrationsstrategie erwiesen [2]. Diese Aufteilung wird auf Basis der toolbasierten Analyse von *vertikalen* Geschäftsprozessen (zusammenhängende Komponenten von Masken, Programmen und Daten) und dem Fachwissen des Kunden vorgenommen. Sie reduziert die Komplexität, sowohl bei der eigentlichen Migration als auch beim Test einzelner Pakete parallel zur Migration. Voraussetzung ist das Identifizieren von separat konvertier- und testbaren Paketen.

Beim zu migrierenden System handelte es sich um ein

Online-System für das Leasinggeschäft, welches unterschiedliche Geschäftsbereiche strukturiert abbildet (Sollstellung, Benutzerverwaltung, Angebotskalkulation, ...). Die Nutzeroberfläche bestand aus einer Menge von Masken, welche über mehrstufige Menüs erreichbar sind. Diese Menüstruktur wurde für das Identifizieren von Paketen genutzt. SüdLeasing erstellte eine Programmliste mit Menüpunkten und dazugehörenden Einstiegsprogrammen pro Geschäftsbereich. Diese bildete die Basismenge der im jeweiligen Paket zu migrierenden Programme. Ausgehend davon erfolgte eine toolgestützte Callgraph-Analyse. Dynamische `CALL`-Aufrufe, bei denen der Name des zu rufenden Programms über eine Variable übergeben wird, wurden durch zusätzliche Datenflussanalysen ermittelt. Die durch `CALL` referenzierten Programme erweiterten die Basismenge. Das wurde in einem ersten Ansatz solange fortgesetzt, bis eine transitive Hülle ermittelt war und keine weiteren Programme mehr gefunden wurden. Die identifizierten Pakete waren im Ergebnis sehr komplex und so für eine Migration ungeeignet. Die Ursache bestand darin, dass zwischen den Paketen viele Querverweise und gemeinsam genutzte Komponenten existierten.

Um Komplexität zu reduzieren, wurde ein pragmatischer Ansatz gewählt: Dieser bestand darin, nur bis zu einer `CALL`-Tiefe von drei aufzulösen. Programme, die nur über mehr als drei Stufen erreichbar waren, wurden für das jeweilige Paket ignoriert. Existierten Aufrufe für diese Programme, dann wurden diese in der Migration als Dummy-Programme konvertiert, welche lediglich eine `NotImplemented`-Exception warfen. Erst in nachfolgenden Paketen ersetzten die eigentlichen, migrierten Programme diese Dummy-Programme. Das hatte zur Folge, dass in den betroffenen Paketen nicht die vollständige Funktionalität realisiert und demzufolge auch keine vollständige Testabdeckung pro Paket zu erreichen war. Die Praxis bestätigte jedoch, dass dieser pragmatische Ansatz sowohl für die Paketgröße als auch für den paketweisen Test einen vertretbaren Kompromiss darstellte, mit welchem ein praktikables Ergebnis erzielt werden konnte.

3 Eine Benutzeroberfläche – zwei Systeme

Bei der Benutzeroberfläche des Legacy-Systems handelte es sich um eine SüdLeasing-Eigenentwicklung auf Windows-Basis mit ca. 1.300 ASCII-Masken. COBOL-Server und Windows-Client tauschten ihre Informationen in Form von COBOL-Messages aus. Bereits vor dem eigentlichen Migrationsprojekt entwickelte SüdLeasing mit dem Framework Angular die gesamte Benutzeroberfläche als Webapplikation neu. Der Informationsaustausch zu den unveränderten COBOL-Servern erfolgte nach wie vor über COBOL-Messages. Diese wurden durch Zusatzinformationen in Form von Kommentaren angereichert. Der COBOL-Precompiler wertete diese zusätzlichen Informationen aus, so dass diese dem Client zur Laufzeit zur Verfügung standen. So konnte die Darstellung der Informationen in der Weboberfläche feingranular gesteuert werden.
Das folgende Beispiel zeigt den Ausschnitt einer typischen COBOL-Message mit einem zusätzlichen Kommentar als Erweiterung:

```
10 bld1-b30e0117 pic 9(02).       02 B
```

Der Kommentar `02 B` steuert die Breite der Anzeige und die Ausrichtung der angezeigten Zahl.
Die Webapplikation sollte im Zielsystem auch zukünftig genutzt werden. Es bestand somit die Aufgabe, eine Schnittstelle zwischen der Webapplikation und den im Ergebnis der Migration entstandenen Java-Webservices zu entwickeln:
Bei der Migration entstehen aus den COBOL-Servern Java-Webservices. Der Informationsaustausch mit der Weboberfläche erfolgt über Java-Klassen, welche durch Konvertierung der COBOL-Messages entstanden. Die zusätzlich notwendigen Informationen werden durch Annotationen an den Java-Attributen realisiert. CoJaC wurde dazu um nutzerspezifische Übersetzungsregeln erweitert. Das nachfolgende Beispiel dokumentiert eine solche Annotation:

```
@LeascoDataField(
name = "bld1-b30e0117", length = 2,
feldtyp = GanzzahligRechtsbuendig)
public CobolNumber bld1B30e0117 =
                    createNumber(2);
```

Die Annotation beinhaltet z.B. zusätzlich den originalen Message-Namen, der für die Maskensteuerung benötigt wird. Zur Laufzeit stehen diese Informationen der Webapplikation zur Verfügung. Somit wird eine Kompatibilität der Webapplikation sowohl zum originalen COBOL-System als auch zum generierten Java-System erreicht.

Diese Kompatibilität wurde im Testprozess ausgenutzt. SüdLeasing entwickelte eine komplexe Sammlung von Gherkin-Testszenarien unter Nutzung des Frameworks Cucumber. Mit jeder neuen Version wurden diese Testszenarien automatisch unter der Steuerung von Jenkins sowohl für das COBOL-System als auch für das migrierte Java-System abgearbeitet und die Ergebnisse wurden ausgewertet. Die Möglichkeit des automatisierten Vergleichs der Ergebnisse des COBOL- und Java-Systems reduzierte den Testaufwand wesentlich.

4 Das nächste Projekt wird wieder anders

Kein Projekt gleicht dem anderen. Trotz hoch entwickelter Werkzeuge und Technologien bietet jedes neue Herausforderungen. Das vorgestellte Projekt hat bestätigt, dass fertige Werkzeuge und Technologien nicht existieren. Der Aufwand für deren Anpassung bei einem neuen Migrationsprojekt muss bei der Planung berücksichtigt werden.
Im Projekt ist die Kompetenz des Projektpartners hervorzuheben, sowohl beim Aufbau der Infrastruktur, beim Test unter Nutzung moderner Technologien als auch beim „Leben“ dieser Prozesse im Projektverlauf. Dies und die sehr gute Zusammenarbeit der Projektpartner waren ein Garant für den Projekterfolg.

Literaturverzeichnis

[1] Moussavi-Amin, W.: Kosten senken und Innovationen schaffen. Computerwoche 2019, 51 - 52

[2] Becker, C.; Kaiser, U.: Toolbasierte Software-Migration nach Plan. Softwaretechnik-Trends, Band 36, Heft 2, Mai 2016