Testautomatisierung am Beispiel des COBOL-to-Java-Converters CoJaC

Denis Uhlig
pro et con Innovative Informatikanwendungen GmbH,
Dittesstraße 15, 09126 Chemnitz
denis.uhlig@proetcon.de

Abstract

Im Rahmen von Migrationsprojekten steht immer die Frage, ob sich das migrierte System semantisch äquivalent zum Ausgangssystem verhält. Dabei liegen die Fehlerquellen nicht nur in den migrierten Programmen, sondern auch in deren Laufzeitumgebung. Der folgende Beitrag zeigt die Integration von automatisch erstellten Unittests am Beispiel der Migration von COBOL nach Java mit CoJaC. Das Ziel ist die Aufwandreduktion des Tests der migrierten Programme und der Laufzeitumgebung.

1 Migration mit CoJaC

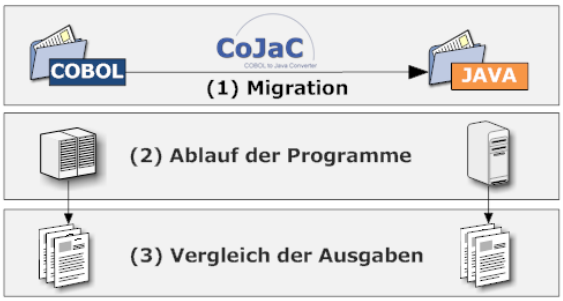
Mit dem COBOL-to-Java-Converter CoJaC[1] stellte pro et con einen Translator für die Migration von COBOL nach Java vor. Das Werkzeug migriert Quelltext aus COBOL in äquivalentes Java. Dabei erfolgt die Abbildung eines COBOL-Programms auf eine Java-Klasse. Zum Ablauf der Java-Programme wird das Laufzeitsystem CoJaC-RTS benötigt. Das dient dazu, spezielle Eigenschaften der Sprache COBOL (z. B. Typsystem und Systemfunktionen) in Java zu emulieren. Die im Ergebnis der Konvertierung entstehenden Java-Programme müssen nach der Migration auf semantische Äquivalenz zum Original getestet werden. Das Ziel war es, die Tests der migirerten Java-Programme, als auch des CoJaC-RTS zu automatisieren. Die dazu entwickelte Technologie basiert auf einem bereits vorgestellten Testverfahren[2].

2 Test durch Vergleich von Ausgaben

Die erste Ausbaustufe des automatisierten Testverfahrens besteht in einem textbasierten Vergleich der Ausgaben des originalen COBOL-Programmes und der konvertierten Java-Kopie. Dabei wird die Eigenschaft von COBOL genutzt, Variablen und Strukturen zu einer Zeichenkette serialisiert auszugeben. Dazu wird der COBOL-Befehl `DISPLAY` genutzt. Da dieser Befehl auch in die Java-Laufzeitumgebung CoJaC-RTS integriert wurde, besteht die Möglichkeit, die Ausgaben beider Programmwelten miteinander zu vergleichen. Damit ein solches Testverfahren funktioniert, muss natürlich gewährleistet sein, dass das COBOL-Programm aussagekräftige Ausgaben erzeugt. Ist das nicht der Fall, müssen diese noch in einem vorbereitenden Schritt in das Programm integriert werden.

Das Verfahren besteht auf der Grundannahme, dass ein Befehl semantisch äquivalent ist, wenn er zu identischen Eingabedaten identische Ergebnisse liefert. Diese Annahme lässt sich auch auf vollständige Programme als Folge von Anweisungen übertragen.

Das Testverfahren wird in der folgenden Abbildung dokumentiert:



Zunächst werden die COBOL-Programme mit CoJaC konvertiert (1). Anschließend werden die Programme in beiden Systemwelten ausgeführt (2), damit die Ausgaben der Programme entstehen. Der letzte Schritt besteht im Vergleich der erstellten Daten (3).

Trotz der Reduktion des Aufwandes gegenüber dem Test durch Debugging ist der manuelle Anteil insbesondere bei komplexen Programmen immer noch hoch. Dieser liegt vor allem im Vergleich der erzeugten Ausgabedaten und in der manuellen Bewertung der gefundenen Unterschiede.

3 Testtreiber und JUnit

Eine weitere Reduzierung des Testaufwandes war zu erwarten, wenn die Vergleichsmöglichkeiten und Testbewertungen zumindest teilautomatisiert erfolgen. Zunächst wurden dazu in einer Analyse des Quelltextes Methoden ermittelt, welche als Einstiegspunkte in die Verarbeitung dienen. Diese sollten in Folge genutzt werden, um die entsprechenden Teilprogramme/Teilsysteme zu testen. Dazu wurde in die konvertierten Java-Quellen eingegriffen und jede entsprechende Methode mit einem so genannten Testtreiber instrumentiert. Dieser erhält die Eingabedaten für den Test in einer standardisierten Form, ruft das eigentliche Java-Programm auf und vergleicht die Rückgabe mit einem Erwartungswert.

Damit auch die kommerziell häufig vorkommenden komplexen COBOL-Strukturen für die Ein- und Ausgabedaten behandelt werden können, wurde an dieser Stelle erneut auf die Möglichkeiten der Serialisierung von Zeichenketten durch das Laufzeitsystem zurückgegriffen.

Die notwendigen Daten für dieses Testverfahren wurden mit Hilfe des bestehenden COBOL-Systems ermittelt. Analog zur Stufe 1 wurden die Eingabe- und Ausgabedaten erstellt und in geeigneter Form im Testtreiber hinterlegt.

Da dieses Verfahren mit festen Erwartungswerten arbeitet, war es möglich, die Testtreiber als Unittests zu realisieren. Dazu wurde das in Java übliche JUnit eingesetzt.

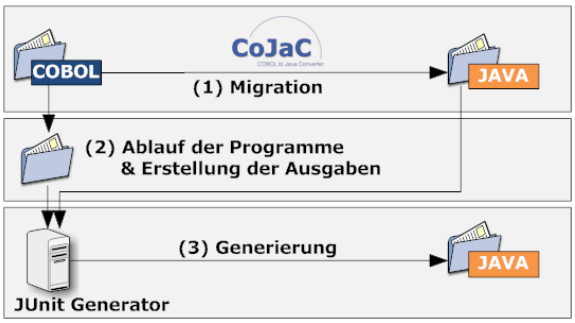
Dadurch stieg zwar die Geschwindigkeit für die Bewertung, ob ein (Teil-)System korrekt arbeitet. Allerdings ist bei diesem Verfahren der manuelle Aufwand zur Imple-

mentierung von Testtreibern der wesentliche Anteil am Testaufwand.

4 Generierte JUnit Tests

Die finale Version des Testverfahrens sollte beide Wege kombinieren. Die schnelle Bewertbarkeit durch JUnit sollte mit der einfachen Testmethode des textuellen Vergleichs verbunden werden.

Die Lösung für diese Anforderung sind Unittests, welche die Ausgaben des COBOL-Programms aus Stufe 1 nutzen, um daraus automatisiert JUnit-Testfälle zu generieren. Die folgende Abbildung zeigt das Verfahren.



Für die Umsetzung dieser Variante müssen die Ausgaben maschinell auswertbar sein. Dazu werden die bestehenden Meldungen standardisiert und erweitert. Darüber hinaus muss es möglich sein, die Ausgaben der COBOL-Programme mit den generierten Java-Programmen zu verknüpfen. Dazu wurden die bestehenden Ausgaben um die Präfixe/Tags `#CASE#` (normale Testausgaben) und `#METHOD#` (Methodenzuordnung) erweitert. Das folgende Beispiel zeigt Ausschnitte aus dem angepassten COBOL-Programm

```
01 PROCEDURE DIVISION.
02   DISPLAY "#METHOD#PROCEDURE"
03   MOVE SOME-TEXT TO SOME-NUMBER
04   DISPLAY "#CASE#", SOME-NUMBER
```

Im Beispiel wird im Hauptparagraphen des Programms eine Zeichenkette (`SOME-TEXT`) einer numerischen Variablen (`SOME-NUMBER`) zugewiesen. Mittels des `DISPLAY`s auf Zeile 02 werden die darauf folgenden Ausgaben dem Hauptparagraphen zugeordnet. Das `DISPLAY` auf Zeile 04 dient dem Test des `MOVE`-Statements auf Zeile 03. Dieses Programm liefert nach Ablauf folgende Ausgaben:

```
01  #METHOD#PROCEDURE
02  #CASE#345
```

Die so orchestrierten COBOL-Programme werden anschließend mit CoJaC konvertiert. Für das oben gezeigte Beispiel ergibt sich folgender Java-Code:

```
01   public void procedure() {
02      display("#METHOD#PROCEDURE");
03      someNumber.setValue(someText);
04      display("#CASE#"
05              + someNum.toString());
```

Der Hauptparagraph der `PROCEDURE-DIVISION` wird als Methode `procedure` in Java umgesetzt (Zeile 01). Für die Ausgaben bietet das CoJaC-RTS die Methode `display` an (Zeilen 02, 04). Das `MOVE`-Statement wird über die Methode `setValue` abgebildet (Zeile 03).

Die Ausgaben der COBOL-Programme und die Java-Programme bilden die Grundlage für einen in Perl entwickelten Generator, der die Informationen zusammenführt und daraus JUnit-Testprogramme generiert. Die Testprogramme basieren auf den erstellten Java-Programmen. Neben einigen globalen Anpassungen (z. B. Integration eines Einstiegspunktes für JUnit), müssen vor allem die modifizierten `display`-Aufrufe behandelt werden. Die Zuordnung der Ausgaben zu den einzelnen Methoden erfolgt über den `#METHOD#`-Tag im `display` und den Ausgaben. Im Beispiel wird die Verknüpfung über den Namen `PROCEDURE` hergestellt. Anschließend werden sequentiell alle `display`s mit `#CASE#`-Tag in Aufrufe der JUnit-Methode `assertEquals` umgewandelt. Den Vergleichswert entnimmt der Generator den standardisierten Ausgaben des COBOL-Programms entsprechend der Reihenfolge. Für das bereits gezeigte Beispiel ergibt sich der folgende JUnit-Testfall (Auschnitt):

```
01   public void procedure() {
02      someNumber.setValue(someText);
03      assertEquals("345",
04              someNumber.toString());
```

5 Fazit

Das beschriebene Verfahren erlaubt es, innerhalb eines überschaubaren Zeitfensters große Mengen von Code zu testen und die semantische Äquivalenz gegen "echte" Daten aus dem COBOL-System zu verifizieren. Da der Großteil der Testumgebung generiert wird und automatisiert ausführbar ist, wird der manuelle Aufwand für den Test wesentlich geringer als vorher.

Darüber hinaus bildet eine solche Testsuite auch die Möglichkeit, Erweiterungen und Änderungen am Laufzeitsystem CoJaC-RTS schnell verifizieren zu können, da es die Entwicklung der Laufzeitumgebung durch die erstellten Testfälle von den COBOL-Programmen entkoppelt. Damit kann auch auf neue Anforderungen in diesem Bereich kurzfristig reagiert werden.

Literaturverzeichnis

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

[2] C. Becker, U. Kaiser: Test der semantischen Äquivalenz von Translatoren am Beispiel von CoJaC 14. Workshop Software-Reengineering (WSR 2012), 2.-4. Mai 2012 , Bad Honnef GI-Softwaretechnik-Trends, Band 32, Heft 2, ISSN 0720-8928