Test der semantischen Äquivalenz von Translatoren am Beispiel von CoJaC

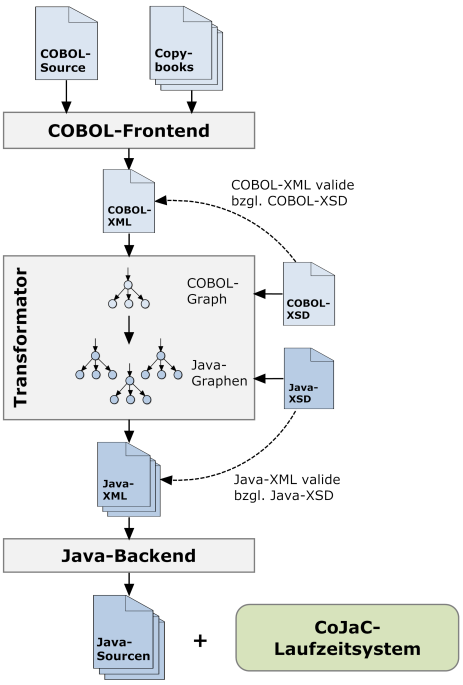
Christian Becker, Uwe Kaiser
pro et con Innovative Informatikanwendungen GmbH, Dittesstraße 15, 09126 Chemnitz
christian.becker@proetcon.de, uwe.kaiser@proetcon.de

Abstract

Im Rahmen des SOAMIG-Projektes[1] wurde u.a. der Translator CoJaC (COBOL to Java Converter) entwickelt. Entwicklungsziele von CoJaC waren, performanten und zum COBOL-Code semantisch äquivalenten Java-Code zu generieren. Der vorliegende Beitrag beschreibt die Testmethodik zum Nachweis dieser semantischen Äquivalenz.

1 CoJaC - COBOL to Java Converter

CoJaC konvertiert ein vollständiges COBOL-Programm (Hauptprogramm und Copybooks) in ein semantisch äquivalentes Java-Programm. CoJaC besteht aus einer Menge unabhängiger Werkzeuge, welche ihre Ergebnisse über Schnittstellen austauschen. Eine detaillierte Beschreibung von CoJaC liefert [1]. Eine für die Beschreibung der Testmethodik vereinfachte Darstellung der Architektur zeigt die nachfolgende Abbildung:



Ein COBOL-Frontend realisiert die aus dem Compilerbau bekannten Funktionen des Präprozessors, des Scanners und des Parsers. Der entstehende COBOL-Syntaxgraph wird in eine XML-Repräsentation serialisiert. Die XML-Schnittstelle entspricht einem definierten XSD-Schema (COBOL-XSD). Diese Repräsentation wurde gewählt, um die Interoperabilität verschiedener Migrationswerkzeuge der Firma pro et con zu gewährleisten. Das generierte COBOL-XML fungiert als Eingabe für ein weiteres Werkzeug (Transformator), welches auf Basis einer vorgegebenen Abbildungsbeschreibung eine *model-to-model*-Transformation aus dem COBOL-Syntaxgraphen in einen oder mehrere Java-Syntaxgraphen ausführt. Dieser Java-Syntaxgraph wird in Form einer serialisierten XML-Datei (Java-XML) dem Java-Backend zugeführt, welches aus dem Java-Generator und dem Java-Formatierer besteht. Ergebnis des Konvertierungsprozesses ist ein Java-Programm, das in seiner Funktionalität dem originalen COBOL-Programm entsprechen sollte. Zur Ausführung nutzen die generierten Java-Programme ein Laufzeitsystem. Dieses stellt Bibliotheksfunktionen für COBOL-Anweisungen und -Daten bereit, für die es in Java keine Entsprechung gibt.

2 Testmethode und Testabdeckung

COBOL besitzt die Eigenschaft, dass zu jeder syntaktischen Einheit (Anweisung, Datendefinition, ...) mehrere, zum Teil optionale Klauseln angegeben werden können. Diese sind in verschiedensten Kombinationen in kommerziellen Programmen enhalten. Deshalb wurde für jede mögliche Kombination von Anweisungen und Klauseln ein minimales Testprogramm (im Weiteren *Testfall*) erstellt, welches nur eine Variante der Anweisung sowie dafür notwendige Datendefinitionen enthält. Die systematische Spezifikation dieser Testfälle erfolgte anhand einer COBOL-Grammatik, welche dem aktuellen ANSI-85-Standard entspricht. Das folgende Listing enthält eine abgeschlossene Regel aus der COBOL-Grammatik für eine DISPLAY-Anweisung:

```
stmt   = DISPLAY id_lit {id_lit} upon;
id_lit= ident | literal;
upon   = [UPON name | env] [NO ADVANCING];
```

Die alternativen Klauseln (UPON, NO ADVANCING, ...) sind ersichtlich. Aus dieser Regel lassen sich u.a. die folgenden Testfälle systematisch ableiten:

- *DISPLAY ident*
- *DISPLAY literal*
- *DISPLAY ident UPON name*
- ...

Die Bezeichner der Grammatik (z.B. *ident*) stehen für einen COBOL-Bezeichner mit beliebigem Datentyp. Das heißt, dass die Anweisung *DISPLAY ident* in Kombination mit numerischen und alphanumerischen Datenelementen und Literalen getestet werden muss. Daraus ergibt sich eine große Anzahl von Testfällen. Die Gesamtheit aller Testfälle wird als Testabdeckung definiert. Diese kann jedoch aufgrund der Vielfalt der möglichen Kombinationen nie 100 % betragen. Im Verlauf der Entwicklung von CoJaC entstand auf diese Weise eine Testsuite mit aktuell mehr als 1.200 Testfällen. Bei neuen Projekten werden ausgewählte, kommerzielle Quellen auf ein Grundgerüst reduziert und ebenfalls konvertiert. Ziel ist, die Abdeckung

eines breiten Spektrums realer Problemstellungen aus historisch gewachsenen Programmen. Die Testsuite wächst folglich ständig mit der Nutzung von CoJaC, da vor allem in kommerziellen Programmen Spezialfälle auftreten, welche bis dahin noch nicht durch einen Testfall abgedeckt waren. Zur Abrundung der Testabdeckung wurden zusätzlich im Internet frei verfügbare, willkürlich ausgewählte COBOL-Programme verwendet.

3 Testrealisierung

Die Realisierung der Tests erfolgt in vier Phasen:
Compilierung und Ausführung der Testfälle in einer COBOL Enterprise Umgebung: Dadurch wird die syntaktische Korrektheit und Ausführbarkeit der Testfälle garantiert. Nur compilierbarer COBOL-Code soll von CoJaC konvertiert werden. Von jedem Testfall werden Dumps (z.B. Bildschirmausgaben) in einem Repository abgelegt.
Konvertierung der Testfälle von COBOL nach Java: In dieser Phase erfolgt die Verifikation der Funktionsfähigkeit des Translators sowie die Korrektheit der Schnittstellen zwischen den Werkzeugen. Der erzeugte Java-Code wird gegen die Konventionen der Abbildungsbeschreibung geprüft.
Ausführung der generierten Java-Programme: Alle generierten Java-Programme werden automatisiert compiliert und in einer Java-Umgebung ausgeführt. Dadurch erfolgt die Verifikation der Funktionalitäten des Laufzeitsystems. Zu jedem Testfall werden wiederum Dumps (z.B. Bildschirmausgaben) in Repositories abgelegt.
Vergleich der Repositories: In der letzten Testphase findet ein Vergleich der COBOL- und Java-Repositories eines Testfalls statt. Auf diese Weise werden semantische Fehler aufgedeckt. Sind die Dumps einer Anweisung in COBOL und in Java identisch, dann war auch die Transformation semantisch äquivalent. Nur in dieser Testphase kann die semantisch äquivalente Transformation von COBOL nach Java für einen Testfall nachgewiesen werden.
Jede Phase wurde über Batch-Prozesse automatisiert und ist somit beliebig oft wiederholbar.

4 Ergebnis des Testprozesses

Die Durchführung der Tests liefert eine Menge von Resultaten, welche zum Lokalisieren von Fehlern im Konvertierungsprozess und im Laufzeitsystem dienen. Die Fehler werden den einzelnen Translatorkomponenten zugeordnet und iterativ beseitigt.
Syntaktische Fehler in den COBOL-Programmen: Syntaktische Fehler treten auf, wenn ein Codefragment nicht vom COBOL-Frontend verarbeitet werden kann. Dieses wird seit Jahren stetig weiterentwickelt und wurde bereits an einer Vielzahl von kommerziellen Quellen ausgetestet. Daher sind echte syntaktische Fehler eher selten. Das Frontend arbeitet restriktiver als der Compiler, wodurch es zusätzlich die Sanierungsphase von COBOL-Programmen vor einer Software-Migration unterstützt.
Syntaktische Fehler in den Java-Programmen: Die erzeugten Java-Programme werden mit Hilfe eines Java-Compilers auf syntaktische Fehler untersucht. Fehler in der *model-to-model*-Transformation des Transformators oder der Codegenerierung des Java-Backends können auf diese Weise erkannt werden.
Korrektheit der COBOL- und Java-Syntaxbäume: Die vom COBOL-Frontend und dem Transformator gelieferten COBOL- bzw. Java-Syntaxbäume (COBOL- bzw. Java-XML) werden nach dem jeweiligen Verarbeitungsschritt gegen ein entsprechendes Schema (COBOL-XSD bzw. Java-XSD) validiert. Dies dient zur zusätzlichen Überprüfung der korrekten Verschachtelung von Anweisungen und Operatoren sowie zur Sicherstellung der fehlerfreien Verarbeitung in den nachfolgenden Transformationsprozessen.
Semantische Äquivalenz: Semantische Unterschiede in den COBOL- und Java-Programmen werden durch den Vergleich der Ergebnis-Repositories aufgedeckt. Erst wenn die Ausgaben der COBOL- und Java-Programme identisch sind, kann der Code als semantisch äquivalent betrachtet werden. Kann eine Anweisung von CoJaC nicht transformiert werden, dann ist das Programm (aus Sicht der gesamten Programmlogik) nicht semantisch äquivalent. Um solche Fälle zu erkennen, werden im Transformationsprozess so genannte „ToDo()“-Funktionsaufrufe für nicht übersetzbare Anweisungen erzeugt. Ein solches „ToDo“ führt zur Laufzeit zu einer Exception. In diesem Fall muss die Anweisung manuell konvertiert werden.
Laufzeitfehler: COBOL-Compiler tolerieren fehlerhafte Konstruktionen, die nicht unmittelbar zu einem Programmabbruch führen. Z.B. kann einer numerischen Variablen in COBOL eine Zeichenkette zugewiesen werden. In Java wird in diesem Fall durch das Laufzeitsystem eine Exception geworfen, da die Zuweisung eindeutig unkorrekt ist. Das Laufzeitsystem arbeitet somit sehr restriktiv.

5 Zusammenfassung

Trotz der umfangreichen Tests kann nicht gezeigt werden, dass für den gesamten COBOL-Sprachumfang semantisch äquivalenter Java-Code erzeugt wird. Dies gilt nur für jene Testfälle, welche den vollständigen Testprozess erfolgreich bestehen. Dazu gehören die Tests aus der Testsuite, die verwendeten kommerziellen Kundenlösungen und die willkürlich ausgewählten COBOL-Programme aus dem Internet. Der durch diese Kombination erreichte Querschnitt des COBOL-Sprachumfangs war für den bisherigen, kommerziellen Einsatz von CoJaC ausreichend.

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

[1] Das vom Bundesministerium für Bildung und Forschung geförderte Projekt SOAMIG beschäftigte sich mit der Migration von Legacy-Software in serviceorientierte Architekturen.