Vom COBOL-Server zum Java-Webservice

Uwe Erdmenger
pro et con Innovative Informatikanwendungen GmbH, Dittesstraße 15, 09126 Chemnitz
uwe.erdmenger@proetcon.de

Abstract

Der Einsatz von Konvertierungswerkzeugen (Translatoren) beschleunigt ein Migrationsprojekt wesentlich und gestaltet es kostengünstiger. Gegenstand der folgenden Ausführungen sind ausgewählte Entwicklungsaspekte für einen COBOL → Java-Translator, an dem die Firma pro et con GmbH gegenwärtig arbeitet. Dabei werden neben der reinen 1:1-Sprachmigration auch insbesondere die Anpassungen des Quellcodes an die Middleware (Transaktionsmonitor, Application-Server) und an die Behandlung von embedded-Systemen (z.B. SQL) berücksichtigt.

1 Motivation

In vielen Unternehmen sind heute noch große, in COBOL entwickelte Legacy-Systeme im Einsatz, welche teilweise über Jahrzehnte gewachsen sind, viel betriebliches Know-How beinhalten und daher nicht einfach durch Standardsoftware abgelöst werden können. Um deren Weiterentwicklung und Betrieb kostengünstiger zu gestalten, wird insbesondere in letzter Zeit vermehrt der Wunsch geäußert, diese Systeme auf modernere Programmiersprachen und Ablaufumgebungen zu migrieren. Ein häufig genanntes Migrationsziel ist die Java Enterprise Edition, für die viele Werkzeuge (z.B. Eclipse), Klassenbibliotheken und auch Softwareentwickler zur Verfügung stehen [1].

2 Abbildungsvorschrift COBOL → Java

Ausgangspunkt einer jeden Sprachmigration, ob toolgestützt oder manuell, ist eine Abbildungsbeschreibung, welche den syntaktischen Konstrukten der Ausgangssprache entsprechende, semantisch äquivalente Konstrukte der Zielsprache zuordnet. Dabei muss die Menge der Ausdrücke, Anweisungen, Definitionen, ... der Ausgangssprache möglichst vollständig erfasst werden. Entsprechend wird zunächst eine solche Vorschrift für die Migration von COBOL nach Java entwickelt.

Bei der Migration sind die folgenden Grundprinzipien einzuhalten: *Erhaltung der Wartbarkeit* und *Erzeugung Java-typischen Zielcodes*. Das erstere ist eine Notwendigkeit, wenn die Programme nach der Migration weiterentwickelt werden sollen. Auch das zweite Grundprinzip dient diesem Ziel. Allerdings widersprechen sich die beiden Prinzipien teilweise: Aus Sicht der Wartbarkeit ist es sicher am sinnvollsten, den Zielcode möglichst nahe am Original zu halten, damit die Programmierer sich leichter zurechtfinden. Aus Sicht der Erzeugung Java-typischen Zielcodes ist es sinnvoll, sich soweit wie möglich an die Zielsprache Java anzunähern. Im Projekt wird versucht, einen ausgewogenen Kompromiss zu realisieren. Im Weiteren werden exemplarisch einige Designentscheidungen vorgestellt:

Alphanumerische Daten werden in COBOL in folgender Art beschrieben:

```
01 USER-ID PIC X(08).
```

Im Gegensatz zu Java-Strings haben die alphanumerischen COBOL-Datenfelder eine feste Länge, die alle Arten von (ASCII- bzw. EBCDIC-) Zeichen aufnehmen können. Um diese Besonderheit auch in Java geeignet abzubilden, scheint sich auf den ersten Blick die Implementation einer eigenen Klasse anzubieten. Allerdings hat diese Lösung Nachteile: Alle für `java.lang.String` bereits vorhandenen Methoden müssten nachimplementiert werden. Und die dafür mögliche Sondersyntax (z.B. Verketten von Strings mit dem +-Operator) steht für selbstdefinierte Klassen nicht zur Verfügung und muss durch Methodenaufrufe realisiert werden. Das erzeugt komplexeren, schwerer verständlichen Code. Desweiteren sind String-Literale vom Typ `java.lang.String` und müssen erst konvertiert werden. Daher wird vorgeschlagen, alphanumerische Felder direkt als Java-Strings zu übersetzen und bei allen Zuweisungen (z.B. durch `MOVE`) eine Längenkorrektur vorzunehmen. Die dazu notwendige Feldlänge wird in Annotationen oder als zusätzliche Konstante definiert:

```
String u_id; static final int U_ID_LEN=8;
```

Damit wird eine `MOVE-Anweisung` der Art

```
MOVE "U3156" TO U-ID
```

in

```
user_id = adjust("U3156", U_ID_LEN);
```

konvertiert. Die Methode `adjust` wird dabei über ein Laufzeitsystem bereitgestellt.

Strukturierung wird in COBOL durch Stufennummern realisiert. Niedrigere Stufennummern gruppieren die ihnen untergeordneten Datenelemente:

```
01 PERSON.
   05 U_ID     PIC X(8).
   05 DATUM.
      10 TAG   PIC 9(2).
      10 MONAT PIC 9(2).
      10 JAHR  PIC 9(2).
```

In Java ist die Verschachtelung von Datenelementen nur durch die Definition von Klassen darstellbar. Im Unterschied zu COBOL-Strukturen, welche direkt Datenelemente darstellen, sind Java-Klassen Typen. Das bedeutet, wenn davon eine Instanz mit konkreten Werten benötigt wird, muss sie zunächst mit `new` angelegt und initialisiert werden:

```
class Person {
   String u_id; static final int U_ID_LEN=2;
   class Datum {
      byte tag, monat, jahr;
   };
   Datum datum = new Datum();
};
Person person = new Person();
```

Eine weitere Besonderheit von COBOL ist die Verwendung von (Sub-)Strukturen wie Zeichenketten (alphanumerische Felder), wobei lesende und auch schreibende Zugriffe möglich sind. Dieses wird durch die Bereitstellung zweier Methoden `toString()` und `fromString()` für jede migrierte COBOL-Struktur realisiert. Unter Verwendung von *Reflection* wird der größte Teil der Implementation dieser Methoden ebenfalls vom Laufzeitsystem übernommen, so daß der obige Code nicht wesentlich vergrößert wird. Wichtig dabei ist, daß der entstehende bzw. gelesene String genau den gleichen Aufbau hat wie die COBOL-Struktur.

Andere COBOL-Konstrukte (z.B. Überlagerungen mit `REDEFINES`) erfordern mehr zusätzliche eingefügte Anweisungen und Konvertierungen.

3 Migration der Middleware

COBOL-Programme können in *Batch-Programme* und *Server* unterteilt werden. Letztere laufen unter Steuerung einer Middleware (z.B. *TUXEDO* oder Pathway) und sind Gegenstand der folgenden Betrachtungen. Sie arbeiten wie folgt:

Ein Client schickt eine Message an die Middleware, welche daraufhin den geeigneten Server aktiviert und ihm die Message übergibt. Dieser wertet sie aus und verarbeitet sie. Danach erstellt er eine Antwort-Message, gibt sie an die Middleware zurück und wird wieder inaktiv. Die Middleware überträgt sie dann an den Clienten, welcher in dieser Zeit gewartet hat.

Bei der Migration von COBOL nach Java bietet sich die Ausprägung der Server als Webservices an. Diese arbeiten prinzipiell ähnlich. Aus technischer Sicht handelt es sich dabei um speziell annotierte Java-Klassen, die unter der Steuerung eines Application-Servers (z.B. JBoss) laufen. Über eine Schnittstelle, welche in der Web Services Description Language (WSDL) beschrieben ist, bietet der Service seine Dienste im Internet an. Aus dieser Schnittstellenbeschreibung kann z.B. mit `wsimport` ein Client-Stub generiert werden. Dessen Methoden realisieren die Berechnung nicht selber, sondern sprechen, transparent für den Client-Entwicker, über Netz die entsprechenden Methoden des Webservice an.

Der konvertierte COBOL-Server wird als ein Webservice aufgefasst, der nur eine Web-Methode bereitstellt. Dabei ergibt sich die Frage, welche Parameter der Service erhält. In COBOL sind das Messages, also komplette COBOL-Strukturen. Es scheint zunächst naheliegend, diese, wie oben angedeutet, nach Java zu konvertieren und verschachtelte Java-Klassen als Parameter zu verwenden. Das führt jedoch zu sehr komplexen Parameter-Strukturen, da Messages in COBOL üblicherweise groß sind. Außerdem sind dann nennenswerte Änderungen an den bisher vorhandenen (nicht notwendig in COBOL geschriebenen) Clients notwendig.

Daher wird vorgeschlagen, die Parameter und Rückgabewerte des migrierten Services als einen String, also unstrukturiert, zu übertragen. Dieser muss im Aufbau mit der entsprechenden COBOL-Message übereinstimmen. Im Service selber besteht nun die Notwendigkeit, die Daten aus dem String wieder zu extrahieren und in die Java-Datenfelder zu übertragen. Dazu kann die oben bereits erwähnten Methode `fromString()` genutzt werden. Der Service arbeitet dann über diesen Daten und erstellt am Ende mit Hilfe der Methode `toString()` die zurückzusendende Message. Auf diese Weise sind Änderungen an den Clients geringer und im Server sind lediglich die middleware-spezifischen Aufrufe (z.B. `TPSVCSTART`, `TPRETURN`, ... bei Verwendung von TUXEDO) zu ändern und durch Aufrufe entsprechender Laufzeitsystem-Methoden zu ersetzen. Die notwendigen Transformationen sind dabei weitgehend automatisierbar.

4 Werkzeugunterstützung und Ausblick

Ziel ist es, die Konvertierung mit Werkzeugen zu unterstützen. Wesentliche Teile des COBOL → Java-Translators sind das bereits vorhandene, ausgereifte COBOL-Frontend der Firma pro et con, welches die Informationen über den COBOL-Quelltext als XML-Daten aufbereitet, sowie ein im Projekt SOAMIG (www.soamig.de) entwickelter Java-Unparser, welcher umgekehrt aus XML-Daten, die dem Java-Schema entsprechen, wieder strukturierten Java-Code erzeugt. Der Translator, welcher XML-Daten im COBOL-Schema in Java-XML transformiert, befindet sich aktuell in Entwicklung. Er entspricht dem allgemeinen Translatormodell von pro et con ([1]) und wird unter Verwendung der firmeneigenen Metawerkzeuge, z.B. des Parsergenerators BTRACC2 ([2]), entwickelt.

Eine weitere Aufgabe ist die Vervollständigung der Abbildungsbeschreibung um die COBOL-Anweisungen (`PROCEDURE DIVISION`) und die *compiler-directing statements*, z.B. `COPY`. Diese stellen eine besondere Herausforderung dar, da entsprechende Konstrukte (also Präprozessoranweisungen) in Java nicht vorhanden sind. Hier steht zu erwarten, daß eine komplette 1:1-Migration nicht möglich ist. Nicht konvertierbare Konstrukte müssen allerdings auch aufgeführt werden, damit sie in den konkreten Migrationsprojekten in einer vorgelagerten Sanierungsphase beseitigt bzw. in unkritische Konstrukte umgewandelt werden können.

Literaturverzeichnis

[1] Erdmenger, U.; Kaiser,U.; Loos, A.; Uhlig, D.:
Methoden u. Werkzeuge für die Software Migration.
In: Gimnich, R.; Kaiser, U.; Quante, J.; Winter, A. (Eds.): 10th Workshop Software-Reengineering (WSR 2008), 5.-7. May 2008, Bad Honnef.
Lecture Notes in Informatics, (LNI)-Proceedings, Volume P-126, S. 83-97

[2] Erdmenger, U.: Der Parsergenerator BTRACC2.
11. Workshop Software-Reengineering (WSR 2009), Bad Honnef 4.-6. Mai 2009
In: GI-Softwaretechnik-Trends, Band 29, Heft 2, ISSN 0720-8928, S. 34 und 35