Konvertierung der Jobsteuerung am Beispiel einer BS2000-Migration

Uwe Erdmenger, Denis Uhlig
pro et con Innovative Informatikanwendungen GmbH, Annaberger Straße 240, 09125 Chemnitz
Uwe.Erdmenger@proetcon.de, Denis.Uhlig@proetcon.de

Software-Migration beschränkt sich nicht nur auf die Konvertierung von Programmen, Bildschirmmasken und Dateien/Datenbanken. Zu einem produktiven System gehören auch Skripten bzw. Jobs zur Ablauforganisation und Protokollierung.

Der vorliegende Beitrag beschreibt eine toolgestützte Konvertierung der unter BS2000 eingesetzten Jobsteuersprache **SDF (System-Dialog-Facility)** nach **Perl**. Es werden daraus Schlußfolgerungen zur toolgestützten Konvertierung von Jobsteuersprachen allgemein gezogen.

1 Eigenschaften der Quellsprachen

Sprachen zur Jobsteuerung unterscheiden sich in einigen wesentlichen Eigenschaften von Programmiersprachen. Die wichtigsten Unterschiede sollen in den folgenden Punkten erläutert werden.

Syntaktische Komplexität: Jobs werden meist interpretativ abgearbeitet. Dabei wird typischerweise eine befehlsweise Abarbeitung durchgeführt. Dadurch ist die syntaktische Komplexität geringer als bei üblichen Programmiersprachen.

Abhängigkeit vom Betriebssystem: In Jobs wird ungekapselt auf Funktionen des Betriebssystems zugegriffen (keine Bibliotheken, wie bei Programmiersprachen üblich). Ein Beispiel ist bei SDF die direkte Nutzung von Jobvariablen zur Inter-Prozess-Kommunikation.

Einbettung externer Programme: Da Jobs häufig zur Automatisierung von Abläufen eingesetzt werden, besteht die Notwendigkeit, externe Programme in die Jobs einzubetten, um sie zu steuern und um Ergebnisse zu kontrollieren. Eingebettet sein können z.B. Systemprogramme bzw. Utilities wie SORT oder EDT, oder Aufrufe und Parameter von Programmen.

2 Wahl der Zielsprache

Die genannten Eigenschaften beeinflussen die Auswahl der Zielsprache. Dafür sind folgende Kriterien wesentlich:

Sprachumfang: Die Zielsprache soll möglichst viele der im ersten Abschnitt genannten Eigenschaften unterstützen. Alle nicht unterstützten Funktionalitäten müssen in der Zielsprache nachgebildet werden können.

Erweiterbarkeit: Es muss möglich sein, die Zielsprache durch Bibliotheken zu erweitern. Auf diese Weise ist es einfach und effizient möglich, für die konvertierten Jobs Funktionen, welche die Zielsprache nicht bietet, verfügbar zu machen. Des Weiteren sollen Module für die zukünftige Weiterentwicklung zur Verfügung stehen (z. B. Datenbankanbindung).

Portabilität: Die Zielsprache soll auf mehreren Hardwareplattformen zur Verfügung stehen, um zukünftige Migrationen zu erleichtern.

Für die Konvertierung von SDF fiel die Wahl auf Perl als Zielsprache. Perl ist ebenso eine interpretative Sprache, welche die wesentlichen Eigenschaften von Jobsteuersprachen bietet (Variablensubstitution, Einbettung externer Programme, ...). Perl ist auf den verschiedensten Plattformen verfügbar. Damit ist auch das Kriterium der Portabilität erfüllt. Auch die Erweiterbarkeit ist gegeben. Perl kann über spezielle Schnittstellen auf Bibliotheken und eine Vielzahl von Modulen im Comprehensive Perl Archive Network (CPAN) zurückgreifen. Ebenso erfüllt der Sprachumfang die geforderten Kriterien.

3 Werkzeug

Für die Konvertierung entwickelte pro et con ein Werkzeug **SDF-to-Perl-Translator (S2P)**. S2P besteht aus drei Komponenten. Die erste Komponente *Parse* führt eine Syntaxanalyse durch, deren Ergebnis ein Syntaxbaum der SDF-Prozedur ist. Dieser ist mittels verschachtelter Hashes und Arrays implementiert. Dieser Baum wird anschließend in ein Repository abgelegt, um den Aufwand für zukünftige Schritte zu minimieren.

Die zweite Komponente *Analyse* erlaubt Analysen über die SDF-Prozeduren. Dabei kann die Analysebasis eine einzelne Prozedur (z.B. Analyse von Befehlen oder Parameterlisten) oder auch das gesamte Repository sein. Diese Batch-Analysen erstellen z. B. die Aufruf-Hierarchie oder führen eine Suche nach Clones, also ähnlichen Prozeduren, durch. Die Ergebnisse der Analysen werden als .csv-Dateien oder als HTML-Dokument angeboten.

Die dritte Komponente *Convert* realisiert die eigentliche Übersetzung der SDF-Prozedur. Bei der Konvertierung der sequentiell abgearbeiteten Befehle werden zwei Typen unterschieden: Befehle, die ein Äquivalent in Perl besitzen, werden durch diese abgebildet. So wird aus dem Sprungbefehl `SKIP-COMMANDS` ein `goto` in der generierten Perl-Funktion. Alle anderen Befehle ohne Perl-Äquivalent erfordern eine Nachbehandlung. Bei der Konvertierung wird der Name in Kleinbuchstaben gewandelt und die Parameterübergabe wird in eine für Perl typische Notation

überführt. Die Implementierung dieser Befehle erfolgt in einem Laufzeitsystem. Das folgende Beispiel zeigt eine solche Konvertierung.
Original:

```
/  ADD-FILE-LINK LINK-NAME=SORTOUT -
/                FILE-NAME=I.TST.DAT
```

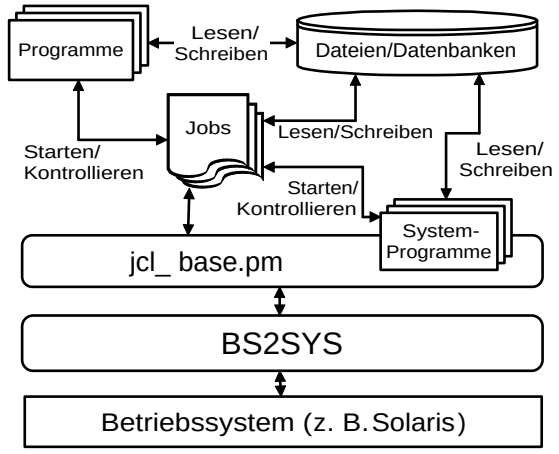
generiertes Perl:

```
add_file_link (LINK_NAME => 'SORT_OUT',
               FILE_NAME => 'I.TST.DAT');
```

Das Ergebnis der Komponente *Convert* ist eine zur SDF-Prozedur semantisch äquivalente Perl-Funktion, die alle konvertierbaren SDF-Befehle beinhaltet. Trotzdem bleibt immer ein gewisser Teil Handarbeit. Nicht konvertierbare SDF-Kommandos werden als Perl-Kommentar in den Zielcode übernommen, um diese manuelle Arbeit zu erleichtern.

4 Laufzeitsystem

Zur Ausführung der konvertierten Skripten wird zusätzlich ein Laufzeitsystem benötigt, welches die von Perl nicht unterstützen Befehle und Funktionalitäten nachbildet. Beim Laufzeitsystem der aus SDF generierten Perl-Prozeduren wurde ein zweischichtiger Aufbau gewählt (Vgl. Abbildung). Die systemnahen Funktionalitäten sind in der Komponente BS2SYS konzentriert. BS2SYS ist eine in C implementierte Bibliothek, die als Shared Object bereitgestellt wird. Eine Funktion aus dieser Bibliothek ist z.B. die Konvertierung von Jobvariablen und Dateien.



Die übrige Funktionaliät des Laufzeitsystems wird im Modul jcl_base.pm in Perl realisiert.

Der Quellcode des Laufzeitsystems wurde in maschinell auswertbarer Form kommentiert. Das Werkzeug **PerlDoc2** von pro et con erstellt daraus eine Nutzerdokumentation in HTML.

5 Konvertierungsbeispiel

Das folgende Beispiel demonstriert einen Fall, bei dem eine SDF-Funktionalität in Perl nachgebildet wurde.

SDF verfügt über eine spezielle Technik, um auf Fehler zu reagieren. Schlägt ein Kommando fehl, wird ein so genannter Spin-Off ausgelöst. Dieser kann als eine Ausnahme (Exception) betrachtet werden. Das bedeutet, es werden nach einem Fehler alle Folgebefehle übersprungen. Gestoppt wird der Spin-Off erst, wenn ein Befehl erreicht wird, der als Aufsetzpunkt dienen kann (z. B. `SET-JOB-STEP`). Das nachfolgende Beispiel zeigt einen solchen Fall:

```
/ REMARK ******************************************
/ REMARK TEST AUF EXISTENZ DER DATEI I.TST.DAT
/ REMARK ******************************************
/ .FILETEST SHOW-FILE-ATTRIBUTES FILE-NAME=I.TST.DAT
/      WRITE-TEXT 'FILE I.TST.DAT IST VORHANDEN'
/      SKIP-COMMANDS TO-LABEL=WEITER
/      SET-JOB-STEP
/      WRITE-TEXT 'FILE I.TST.DAT IST NICHT VORHANDEN'
/ .WEITER
```

Der Befehl `SHOW-FILE-ATTRIBUTES` gibt die Eigenschaften der im Parameter `FILE-NAME` angegebenen Datei aus. Ist die Datei vorhanden und der Befehl damit erfolgreich, wird die nachfolgende Textausgabe ausgeführt und mittels `SKIP-COMMANDS` die Fehlerbehandlung übersprungen. Ist die Datei allerdings nicht vorhanden, werden alle Befehle bis zu `SET-JOB-STEP` übersprungen. Im Beispiel wird einfach eine Meldung ausgegeben und die Verarbeitung wird fortgesetzt.

Eine solche Technik existiert in Perl nicht und wurde deshalb wie folgt emuliert: Im ersten Schritt werden alle Aufsetzpunkte im Job ermittelt und mit einer Sprungmarke versehen, sofern sie noch keine besitzen. Zusätzlich muss nun an jeder Sprungmarke ein Befehl eingefügt werden, der im Laufzeitsystem den nächsten Aufsetzpunkt für den Spin-Off setzt. Wird nun durch einen Befehl eine Ausnahme ausgelöst, ermittelt das Laufzeitsystem den nächsten Aufsetzpunkt und springt zur entsprechenden Sprungmarke. Der resultierende Perl-Text sieht dann wie folgt aus:

```
REMARK("******************************************");
REMARK("TEST AUF EXISTENZ DER DATEI I.TST.DAT     ");
REMARK("******************************************");
FILETEST:
   NEXT_STEP('SPIN_OFF_LAB_1');
   show_file_attributes(FILE_NAME=>'I.TST.DAT');
   write_text('FILE I.TST.DAT IST VORHANDEN');
   goto WEITER;
SPIN_OFF_LAB_1:
   NEXT_STEP('SPIN_OFF_LAB_2');
   write_text('FILE I.TST.DAT IST NICHT VORHANDEN');
WEITER:
   NEXT_STEP('SPIN_OFF_LAB_2');
```

Wie beschrieben, wird mittels `NEXT_STEP` der nächste Aufsetzpunkt bekannt gemacht. Tritt in der Perl-Variante in dem im Laufzeitsystem nachgebildeten Befehl `show_file_attributes` ein Fehler auf, wird das zuletzt registrierte Spin-Off-Label ermittelt (in diesem Fall `SPIN_OFF_LAB_1`) und angesprungen. Ist dieses erreicht, wird der nächste Aufsetztpunkt gesetzt und die Fehlerbehandlung durchgeführt.

6 Zusammenfassung

S2P wurde erfolgreich in einem BS2000-Migrationsprojekt eingesetzt. Unsere praktischen Erfahrungen besagen, dass es möglich und sinnvoll ist, nicht nur Programme und Daten, sondern auch die Jobs (teil-)automatisch mit Toolunterstützung zu migrieren.