Ein Metatool für die model-to-model Transformation

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

Abstract

Ein Kernbestandteil von Softwaremigrationen ist die Konvertierung von Quelltexten in eine neue Programmiersprache oder einen neuen Dialekt. Der Einsatz von Konvertierungswerkzeugen (Translatoren) beschleunigt dabei ein Migrationsprojekt wesentlich und gestaltet es kostengünstiger. Gegenstand der folgenden Ausführungen ist ein Metatool der Firma *pro et con* für die *model-to-model* Transformation, welches die Erstellung von Translatoren unterstützt und transparenter gestaltet.

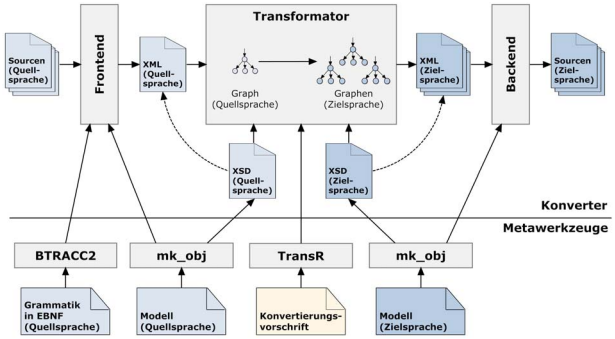
1 Motivation

Bei der Entwicklung von Translatoren und Compilern kommen eine Reihe von Generatoren, Metatools und Bibliotheken zum Einsatz. Bereits in diesem Rahmen vorgestellt wurden *BTRACC2*, ein Parsergenerator von pro et con [1] und *mk_obj*, ein Tool für die Sprachmodelldefinition. Ebenso kommen für die Ausgabe von Programmcode bei Translatoren Generatoren (*CGen*, *JGen*) zum Einsatz [2]. Mit diesen und ähnlichen Werkzeugen wird die Erstellung von Front- und Backends umfangreich unterstützt. Die Erstellung des eigentlichen Transformators, welcher das interne Modell der Quellsprache in das Modell der Zielsprache überführt, ist hingegen meist noch Handarbeit. Gerade an dieser Stelle ist jedoch eine Toolunterstützung wünschenswert, da es sich hierbei um die aufwendigste Komponente eines Konvertierungswerkzeugs handelt.

2 Anforderungen an das Metatool

Ausgangspunkt einer jeden Sprachmigration ist eine Abbildungsbeschreibung, welche den syntaktischen Konstrukten der Ausgangssprache entsprechende, semantisch äquivalente Konstrukte der Zielsprache zuordnet. Diese (meist recht umfangreiche) Abbildungsvorschrift liegt zunächst als Menge verbal formulierter Konvertierungsregeln vor. In eine formale Notation gebracht, bilden sie den Kernbestandteil, die Verarbeitungslogik des Transformators. Dieses Werkzeug muss also *regelorientiert* arbeiten und dabei eine *kompakte, übersichtliche Regelnotation* verwenden. Allerdings gibt es auch Bestandteile einer Konvertierung, welche durch eine algorithmische Beschreibung besser als mit Hilfe von Regeln abgebildet werden können. Die Umsetzung der Schreibweise von Bezeichnern ist hierfür ein Beispiel. Aus diesem Grund müssen auch *prozedurale Sprachelemente* integriert sein.

Das Translatormodell der Firma *pro et con* (s. Abb.) sieht eine lose Kopplung von Frontend, Transformator und Backend vor. Umfangreiche XML-Dateien, die den jeweiligen Quell- und Zielsprachenschemata (XSD) entsprechen, bilden die Schnittstelle. Daher muss der Konverter die *effiziente Ein- und Ausgabe von XML* beherrschen.



Vorhandene Lösungsansätze dazu, speziell XSLT und diverse *tree/graph rewriting tools*, wurden untersucht. Sie konnten jedoch nicht alle der oben genannten Anforderungen erfüllen. Daher wurde eine Eigenentwicklung favorisiert.

3 Entwicklungsaspekte

Es wurde keine von Grund auf neue Sprache entworfen, da der dafür erforderliche Aufwand zu hoch gewesen wäre. Stattdessen wurde die Kombination zweier etablierter Sprachen bevorzugt. Als Ausgangspunkt für die Regelnotation und - abarbeitung diente dabei die aus dem Umfeld der Künstlichen Intelligenz (KI) bekannte Sprache *Prolog*. Diese Sprache zeichnet sich durch eine einfache, kompakte und trotzdem durch eigene Operatoren flexibel erweiterbare Syntax aus. Aufgrund ihres rekursiven Aufbaus aus *Atomen*, n-stelligen *Funktoren* und logischen *Variablen* sind Prolog-Terme gut als interne Abbildung der Syntaxbäume und -graphen geeignet. Das Regelwerk, welches in einer ähnlich einfachen Syntax notiert wird (s. [3]), bildet die Grundlage für die formale Notation der Abbildungsvorschrift. Die Abarbeitung der Regeln greift auf Mechanismen wie *Unifikation* (pattern matching), *Backtracking* (Nichtdeterminismus) und *Rekursion* zurück. Im Verlaufe dieses Prozesses entsteht aus dem Term des Syntaxgraphen der Quellsprache, welcher die Regelanwendung steuert, der Graph der Zielsprache.

Als prozedurale Sprache wurde *Perl* gewählt. Ausschlaggebend dafür waren neben den langjährigen Erfahrungen mit dieser Sprache auch die flexiblen Möglichkeiten der Stringverarbeitung (z. B. mit regulären Ausdrücken). Perl bildet die „Wirtssprache“ des Tools, in welche der Prolog-Interpreter als in C geschriebenes und über die XS-Schnittstelle angebundenes Modul eingebettet wurde. Bei diesem Interpretermodul wurde aus Flexibilitätsgründen eine Eigenentwicklung bevorzugt, welche die notwendigen Eigenschaften (Parser, virtuelle Maschine [warren abstract machine – WAM], wesentliche Standardprädikate, ...) besitzt, auf andere Teile (z. B. garbage collection) jedoch verzichtet. Dadurch wird eine schnelle Regelabarbei-

tung erreicht.

Wesentlich für den Erfolg einer solchen Sprachkombination ist der sorgfältige Entwurf einer ausreichend flexiblen Schnittstelle. Das Prolog-Modul stellt die Klassen `Converter` und `TermType` bereit. `Converter` besitzt Methoden für die Erstellung und Initialisierung des Interpreters (*new*), die Definition des Regelwerks (*compile*), den Aufruf einer Abfrage *(query)*, die Definition von Termen (z. B. *mkAtom*) usw. `TermType` dient dem Zugriff auf Prolog-Terme von Perl aus, z. B. der Abfrage des Term-Typs (*isAtom*, ...) oder der Ermittlung des Zahlenwertes (*toInt*). Um den Komfort zu erhöhen, wurde in Perl mit Hilfe des Filter-Standardmoduls („Perl-Präprozessor“) eine spezielle neue Syntax für die Regelangabe definiert. Das folgende Beispiel der Konvertierung einer COBOL-OPEN-Anweisung verdeutlicht das:

```
my $conv = Converter->new("input.xml");
ENTER_RULES($conv, DBG=>0)
...
conv_open_stmt(cobol_open_stmt_node(
                  files=FileList,
                  comments=Comments),
               ResultStmtList) :-
   conv_open_list(FileList,ResultStmtList),
   add_comments(ResultStmtList,Comments).
...
LEAVE_RULES
```

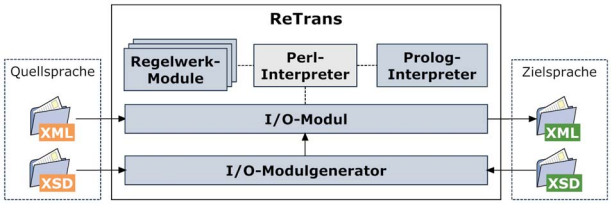
Um den Anwendungskomfort zu erhöhen, wurde die Prolog-Syntax um benannte Parameter erweitert (z.B. `files` und `comments` in `conv_open_stmt`). Die Parameternamen stehen für eine bestimmte Position des Subterms. Beim Übersetzen der Regel werden sie entfernt, die Terme nach dem =-Zeichen an die richtige Position gesetzt und die fehlenden Subterme durch anonyme Variablen ersetzt. Die Zuordnung des Namens zur Position wird aus dem Sprachmodell (XSD) abgeleitet.

Die so definierten Regeln können auch Perl-Funktionen (*Callbacks*) aufrufen. Diese werden ebenfalls durch eine neue Syntax in Perl definiert und vom Filter in Perl-Subroutinen übersetzt:

```
callback($conv)
pl_get_atom_len(atom $str, var $len) {
   $conv->bindVar($len,
              $conv->mkInt(length($str)));
   return 1;
}
```

Ist der erwartete Term-Typ dabei bekannt, kann er in der Parameterliste angegeben werden und wird automatisch in einen äquivalenten Perl-Typ konvertiert, z. B. Atome in Strings. Es ist auch möglich, Callbacks zu erstellen, die im Falle eines Backtrackings erneut gerufen werden (backtrackbare Callbacks).

Aktiviert wird die Regelabarbeitung durch eine *query*, wobei der Syntaxgraph der Ausgangssprache als Parameter übergeben wird. Er liegt als XML-Datei vor, welche eingelesen und in die interne Prolog-Term-Notation überführt werden muss. Diesem Zweck dient ein weiteres, in C geschriebenes Perl-Modul, das I/O-Modul. Sein Quelltext wird aus den als XSD vorliegenden Sprachmodellen mit Hilfe des *I/O-MODUL-Generators* – seinerseits ebenfalls ein Perl-Programm – generiert. Mit Hilfe eines XML-Parsers (*expat*) liest er die Eingabe und baut die interne Darstellung auf. Umgekehrt beinhaltet er auch Ausgabefunktionalität, die es gestattet, eine interne Darstellung eines Syntaxgraphen wieder im XML-Format auszugeben. Die folgende Skizze verdeutlicht diese Zusammenhänge:



4 Erfahrungen

Das beschriebene Werkzeug kam bei der Entwicklung des **CO**BOL to **Ja**va-**C**onverters *CoJaC* zum Einsatz. Die kompakte Regelnotation und seine mächtigen Funktionen bewirken, dass der entsprechende model-to-model-Konverter nicht mehr als 24.500 Codezeilen umfasst (inkl. Leer- u. Kommentarzeilen) und in nur 1,5 Personenjahren erstellt werden konnte. Durch die Aufteilung dieses Codes auf verschiedene Perl-Module und die insgesamt geringe Codegröße wird die Wartbarkeit des Translators und seine Anpassbarkeit an neue COBOL-Dialekte und weitere Kundenanforderungen positiv beeinflusst. Besonders hervorzuheben ist die gute Performance des Werkzeugs. Durch eine Reihe von Optimierungen ist die Laufzeit von *CoJaC* durchaus mit der Laufzeit von in C++ geschriebenen Translatoren vergleichbar. Auch der Speicherverbrauch ist moderat. Ein handelsüblicher PC mit vier GB Hauptspeicher genügt vollständig, selbst sehr umfangreiche COBOL-Programme nach Java zu konvertieren.

Zusammenfassend kann festgestellt werden, dass mit dem vorgestellten Metatool ein weiteres Werkzeug zur Verfügung steht, welches es pro et con ermöglicht, die Entwicklungszeit von Translatoren zu verkürzen und die Projekte kostengünstiger zu gestalten.

Literaturverzeichnis

[1] 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

[2] Kaiser, U.; Uhlig, D.; Zimmermann, Y.: Tool- und
Schnittstellenarchitektur für eine SOA-Migration.
12. Workshop Software-Reengineering (WSR 2010),
Bad Honnef 3.-5. Mai 2010
In: GI-Softwaretechnik-Trends, Band 30, Heft 2,
ISSN 0720-8928, S. 66 und 67

[3] Richard A. O'Keefe: The Craft of Prolog.
The MIT Press, 1 edition, 1990
ISBN 0-262-15039-5