Ein Editor zum Erstellen von Graphprogrammen

Christian Bley

2. Oktober 2008

1 Einleitung

Viele komplexen Probleme lassen sich mit Hilfe von Graphen lösen oder optimieren. Die Probleme werden auf Graphen abgebildet und können dann mit Verfahren der Graphentehorie bearbeitet werden. Um Graphen zu bearbeiten verwendet man Graphtransformationen.

Ein solches Problem ist die Korrektheit von Transformationssystemen, wie es Thema in [HP08] ist. Als ein Werkzeug, um die Korrektheit von Hochprogrammen zu gewährleisten dient ENFORCe, das in [AHPZ07] vorgestellt wird. Abbildung 1 zeigt vereinfacht die Vorgehensweise von ENFORCe zum Überprüfen eines Programs. Da ENFORCe bisher noch keine GUI für eine einfache Handhabung besaß, wurde das Programm ROOTS ([Jur07]) erweitert, um die ENFORCe-Funktionen zu verwenden. Ein Teil dieser Erweiterung stellt Karl Azab in [Aza08] vor, wo der Schwerpunkt auf verschachtelte Einschränkungen und Anwendungsbedingungen liegt. Diese Arbeit hat als Zielstellung einen Editor für Graphenprogramme in ROOTS, in Hinblick auf eine Verwendung mit ENFORCe, und basiert auf den Erweiterungen in [Aza08].

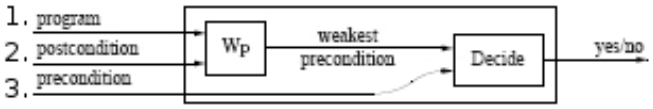

Abbildung 1: Prüfen eines Programms auf Korrektheit

2 Graphprogramme

Ein Graphprogramm verwendet Graphtransformationen, um Beziehungen zwischen Graphen zu verarbeiten und komplexe Operationen auf Graphen durchzuführen. Die formale Beschreibung eines Graphprogramms wird in [HP01], Kapitel 3 ausführlich behandelt. Kurz umschrieben umfasst ein Graphprogramm eine feste Menge von Regeln zur Graphtransformation. Eine Regel besteht aus der linken Seite und der rechten Seite der Transformation und einer Menge von Bedingungen. Die Regeln werden mit Bezeichnern versehen und sind so für das Graphprogramm zugreifbar.

Programme über eine Menge von Bezeichnern C werden wie folgt induktiv definiert:

1. Jede endliche Menge von Regeln über C ist ein Programm.
2. Wenn $P1$ und $P2$ Programme sind, so ist auch $\langle P1; P2\rangle$ ein Programm.
3. Wenn P ein Programm nach 1. und 2. ist, so ist auch $P\downarrow$ ein Programm.

Programme nach Punkt 1. sind elementare Programme, ein Programm $\langle P1; P2\rangle$ ist eine sequentielle Anordnung der elementaren Programme $P1$ und $P2$. $P\downarrow$ ist eine Wiederholung von P. Programme der Form $\langle\langle P1; P2\rangle; P3\rangle$ und $\langle P1; \langle P2; P3\rangle\rangle$ sind äquivalent und können auch als $\langle P1; P2; P3\rangle$ geschrieben werden.

Mit Graphprogrammen können so komplexe Transformationsketten in einfacher Form zusammengefasst werden. Ein Beispiel ihrer Anwendung wird in [AP08] gezeigt.

3 Ein Editor zum Erstellen von Graphprogrammen

Dieses Kapitel beschreibt die Konventionen und Anforderungen an diesen Editor. Als Basis dieser Implementierung dient das Programm ROOTS, das Stefan Jurack in seiner Diplomarbeit ([Jur07]) entwickelt hat. ROOTS dient als Modellierungsumgebung für regelbasierte Transformationssyteme und ermöglicht Definition und Anwendung von Transformationsregeln inklusive Anwendungsbedingungen. Es basiert auf Eclipse, EMF und GEF und ermöglicht so ein professionelles Arbeiten mit grafischen Editoren.

Verwandte Konzepte

Für die Ausführung von Graphprogrammen gibt es auch andere Ansätze. So hat Busatto in [Bus04] ein System für das Ausführen von Graphprogrammen vorgestellt. Hier werden in den Graphprogrammen die Graphen und Morphismen mitgespeichert und über einen Compiler für Graphprogramme in Javaklassen umgewandelt. Diese können dann über eine eigene virtuelle Maschine ausgeführt werden. Diese Implementierung ermöglicht das erzeugen von Javainstanzen der Graphprogramme, die in anderen Javaprogrammen Anwendung finden können. Es ist aber auch auf die Verwendung von Javaklassen beschränkt.

Das Werkzeug ENFORCe ([AHPZ07]) basiert zu einem Teil auf GraJ. Es besteht bereits ein Plugin, um ENFORCe in ROOTS mit einzubinden und zu verwenden.

3.1 Anforderungen

Dieser Abschnitt beschreibt zunächst die funktionalen Anforderungen an den Editor und gibt dann Auskunft über die Anwendungsfälle, die diese Erweiterung dem System hinzufügt.

Funktionale Anforderungen

Die funktionalen Anforderungen lassen sich in den folgenden Punkten zusammenfassen.

- Der Graphprogramm-Editor soll in das bestehende ROOTSprogramm eingebunden werden. Der Editor muss mit den vorhandenen Programmelementen interagieren und auf die bestehende Datenstrukur zugreifen können.
- Das vorhandene Datenmodell muss erweitert werden, um die notwendigen Datenstrukturen für Graphprogramme und deren Ausführung zu erhalten. Das erstellte Modell muss zu dem vorhandenen Modell kompatibel sein.
- Es soll ein textbasierter Editor erstellt werden, der in Aussehen und Nutzung Ähnlichkeit zu bestehenden Programmierumgebungen aufweisen soll. Das heißt, die Strukturelemente des Programms sollen hervorgehoben und fehlerhafte Eingaben angezeigt werden.
- Die in Abschnitt 2 dargelegte Notation für Graphprogramme ist in eine Programmiersprache für Graphprogramme umzusetzen. Die Konvention dieser Sprache wird in Abschnitt 3.2 definiert.

- Mit ROOTS kann man sowohl die Graphen, als auch die Transformationen erstellen und mit Bezeichnern versehen. Der Editor muss aus den im Programmtext verwendeten Bezeichnern die verwendeten Regeln und Programme ableiten und aus der Struktur des Textes das Graphprogramm erstellen.
- Das Graphprogramm soll mit dem Framework ENFORCe ausführbar sein.
- Bei Fehlern und Problemen sollen aussagekräftige Meldungen die Ursache anzeigen und ein Beheben des Fehlers ermöglichen. Kritische Funktionen dürfen mit einem fehlerhaften Pogramm nicht durchführbar sein.

Anwendungsfälle

Nachfolgend werden die Anwendungsfälle aufgezählt, die sich aus der Erweiterung ergeben. Zunächst die Fälle, die sich mit der Verwaltung der Graphprogramme im Datenmodell befassen:

Anwendungsfall	Beschreibung
Neu Erstellen	Ein Programm wird dem Modell hinzugefügt
Eigenschaften ändern	Name und Beschreibung des Programms werden geändert
Löschen	Das Programm wird aus dem Datenmodell gelöscht
Ausführung	Das Programm wird ausgeführt, das heißt, die Transformationen werden entsprechend der Programmsemantik durchgeführt, falls möglich
Speichern	Falls Änderungen vorlagen, werden diese in das persistente Modell übernommen

Die folgenden Fälle beziehen sich auf den eigentlichen Programmcodeeditor:

Anwendungsfall	Beschreibung
Editor öffnen	Aus dem Datenmodell heraus wird ein Pogramm zur Bearbeitung geöffnet. Jedes Programm hat maximal ein Editorfenster offen
Bearbeiten	Der Quellcode des Programms wird bearbeitet. Der veränderte Text wird erst durch eine explizites Speichern auch im Datenmodell persistiert. Die Verarbeitung des Quellcodes erfolgt im Hintergrund der Anwendung

Generell werden Änderungen am Modell nur durch einen expliziten Aufruf der Speichern-Funktion in ROOTS persistent. Wo möglich, sollen Änderungen auch rückgängig zu machen sein.

3.2 Die Programmiersprache

Die Programmiersprache für Graphprogramme ist eng an die Notation für Graphprogramme aus Abschnitt 2 angelehnt. Eine Menge von Regeln wird durch die Mengenklammern '{' und '}' zusammengefasst. Die einzelnen Regeln werden durch Kommata getrennt. Falls eine Menge genau eine Regel enthält, können die Klammern auch weggelassen werden. Programmsequenzen werden durch ';' getrennt. Iterationen werden wegen der einfacheren Notation durch ein '!' anstatt eines '↓' angezeigt. Die Syntax dieser Sprache wird durch die Grammatik in Abbildung 1 erzeugt.

$G = \{(P, S, R), (\{, \}, ;, !, a - z, A - Z, 0 - 9), Prod, P\},$
$Prod = \{$
$P \rightarrow \{S\}|P; P|P!|R$
$S \rightarrow P, S|P$
$R \rightarrow (a - z|A - Z|0 - 9) +$
$\}$

Tabelle 1: Grammatik für Graphprogramme

4 Implementierung

Der Implementierung ging zunächst die Identifizierung der betroffenen Teile des ROOTS-Editors vorraus.

Zuerst wurden dem bestehenden Modell des Editors die Teile für die Verarbeitung von Programmen hinzugefügt. Dann wurden diese Erweiterungen in den zentralen ROOTS-Editor eingebunden. Der nächste Schritt war die Implementierung des eigentlichen Quellcodeeditors und die Programmerzeugung aus dem Quelltext. Zuletzt steht die Verknüpfung des Programms zum Framework ENFORCe.

Modellerweiterung

Das in [Aza08] erweiterte Datenmodell von ROOTS wurde um drei weitere Elemente ergänzt.

Die Klasse `Program` stellt ein Graphrogramm im Modell des Editors dar. Es enthält den Text des Programms, einen Namen, eine Beschreibung und verweist auf den Wurzelknoten des Strukturbaumes, der aus Objekten der Klasse `ProgramTreeElement` besteht. Die einzelnen Knoten haben einen Typ, der in der Aufzählung `ProgramElementType` definiert wird. Ein Knoten kann einen Nachbar und einen Nachfolger haben. Außerdem kann der Knoten direkte Verweise auf referenzierte Transformationsregeln oder andere Graphprogramme enthalten. Ein Klassendiagramm zeigt Abbildung 2. Dieser Strukturbaum dient als ein Metamodell, aus dem dann verschiedene Implementierungen für Graphprogramme abgeleitet werden können.

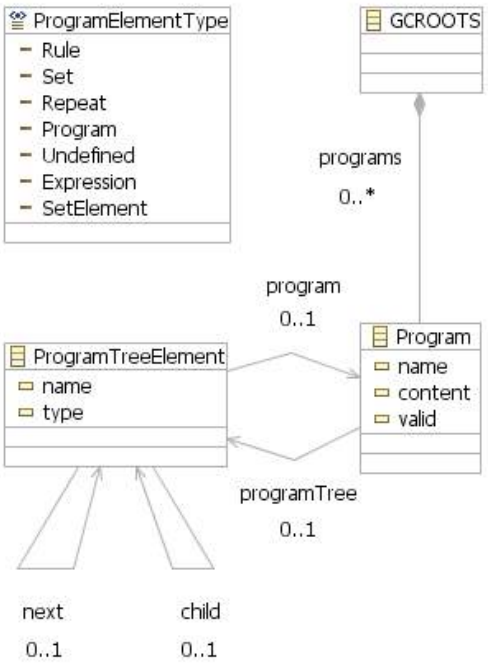

Abbildung 2: Programmmodell

Erweiterung des ROOTS-Editors

Der ROOTS-Editor wurde mit dem `ProgramTreeEditPart` erweitert, um dem Modell Programme hinzufügen zu können und deren Eigenschaften, wie Namen und Beschreibung zu verändern. Der eigentliche Programmeditor wird im `ProgramEditView` umgesetzt. Dieser besteht im wesentlichen aus einem Texteingabefeld, das vom Eclipseframework für Quellcodebearbeitung zur Verfügung gestellt wird. Damit erreicht man ein Erscheinungsbild, wie

man es vom Eclipse-Javaeditor her kennt. Das Aussehen dieses Editors ist in Abbildung 3 dargestellt.

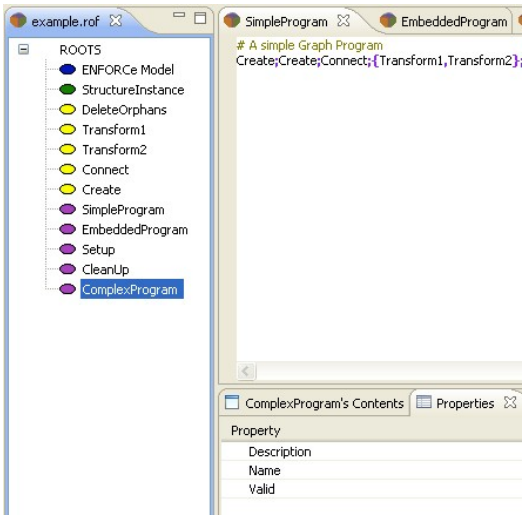

Abbildung 3: Der Programmeditor(links) im ROOTSeditor

Programmerzeugung

Der Programmcode wird durch einen `ProgramParser` analysiert. Dieser erzeugt aus dem Programmcode den Strukturbaum. Als Eingabe dient die in Abschnitt 3.2 beschriebene Sprache. Bei fehlerhafter Syntax oder fehlenden Regeln im Modell wird das Programm als fehlerhaft markiert und eine Fehlermeldung zurückgegeben.

Für den Teil des `ProgramParser`s ist ein Syntaxparser notwendig, der den Programmcode scannt und syntaktisch interpretiert. Für solche Zwecke gibt es Parsergeneratoren wie ANTLR oder JavaCC, die jedoch für große Sprachen ausgelegt sind, und einen hohen Einarbeitungsaufwand erfordern. Deshalb wurde für diese Arbeit RunCC ([Run]) verwendet. RunCC ist ein einfacher Parsergenarator, der für kleine Sprachen ausgelegt ist. Durch die Möglichkeit, Parser zur Laufzeit zu erzeugen, können Änderungen an der Syntax schnell getestet werden. Für diese Arbeit wurde aber ein statisch erzeugter Parser verwendet.

Mit RunCC kann auch eine Semantik an den Parserübergeben werden, um noch während des Parsens eine semantische Ordnung vorzunehmen. Diese `ProgramSemantic` erzeugt eine Rohform des Strukturbaumes. Ein `Resolver` verarbeitet diesen Rohbaum weiter und identifiziert die verwendeten Regeln

und Programme. Falls keine Fehelr auftreten, wird der Strukturbaum an das Programm geknüpft und steht dem Editor zur Verfügung.

Verknüpfung zu ENFORCe

Die Verknüpfung der Programme zum Framework ENFORCe steht noch aus. Zum jetztigen Zeitpunkt konnte dieser Arbeitschritt noch nicht durchgeführt werden.

5 Zusammenfassung und Ausblick

ENFORCe ist ein Werkzeug das viele Funktionen bietet, jedoch bislang nur umständlich zu Bedienen war. Die Erweiterung mit ROOTS als GUI gibt mehr Komfort beim Erstellen von Graphen und Transformationen. Die grafische Benutzeroberfäche erlaubt einfaches Arbeiten und eine gute Übersicht.

Es fehlen noch Elemente für eine bessere Benutzerfreundlichkeit, wie das Markieren von fehlerhaften Eingaben im Programmtext, dem hervorheben von verwendeten Regeln und dem Vorschlagen von Regelnamen, bei teilweiser Eingabe (Content Assist). Als letzter Schritt fehlt noch die Anknüpfung der Programme an ENFORCe oder andere Frameworks, um die Graphprogramme auch optimal nutzen zu können. Hier muss auch die Transformation durch ein Programm umgesetzt werden und die Änderungen der Transformation auf das eigentliche Editormodell zurückgeführt werden, um das Modell konsistent zu halten.

Literatur

[AHPZ07] Karl Azab, Annegret Habel, Karl-Heinz Pennemann, and Christian Zuckschwerdt. ENFORCe: A system for ensuring formal correctness of high-level programs. In *Proc. 3rd International Workshop on Graph Based Tools (GraBaTs'06)*, volume 1, pages 82–93, 2007.

[AP08] Karl Azab and Karl-Heinz Pennemann. Type checking C++ template instantiation by graph programs. In *Proc. Int. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'08)*, pages 249–262, Electronic Communications of EASST, 2008.

[Aza08] Karl Azab. Editing nested constraints and application conditions. In A. Habel and M. Mosbah, editors, *Proc. Int. Workshop on Graph Computation Models*, pages 35–42, 2008.

[Bus04] Giorgio Busatto. GraJ: A system for executing graph programs in java. *Berichte aus dem Fachbereich Informatik 3/04*, 2004.

[HP01] Annegret Habel and Detlef Plump. Computational completeness of programming languages basedon graph transformation. In *Proc. Foundations of Software Science and Computation Structures (FOSSACS 2001)*, volume 2030 of *Lecture Notes in Computer Science*, pages 230–245. Springer-Verlag, 2001.

[HP08] Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 2008. To appear.

[Jur07] Stefan Jurack. Konzeption und Implementierung einer Entwicklungsumgebung für regelbasierte Graphtransformationssysteme basierend auf AGG und Eclipse. Master's thesis, TU Berlin, 2007.

[Run] http://runcc.sourceforge.net/.