Graphgrammatiken als Basis von Programmsystemen und -entwicklungsumgebungen

Rudolf Freund Herbert Pötzl Tatjana Svizensky

Technische Universität Wien, Resselg. 3, A-1040 Wien
Tel.: 0043 1 58801 4084, email: {rudi,herbert,tatjana}@logic.tuwien.ac.at

Seit mehr als 25 Jahren sind Graphgrammatiken in der Theorie der Formalen Sprachen vertreten, in der praktischen Informatik wurden sie jedoch erst geraume Zeit später als Alternative zu konventionellen Ansätzen entdeckt. Praktische Anwendungen existieren erst sehr vereinzelt (e.g. [5]), obwohl eine Vielzahl von Ideen und Konzepten zur Verwendung von Graphgrammatiken in Programmieranwendungen ausgearbeitet und zur Verfügung gestellt wurde ([1]).

Als Hauptkritikpunkte am praktischen Einsatz von Graphgrammatiken für Programme und Programmentwicklungen werden oft mangelnde Effizienz, geringe Übersichtlichkeit und unzureichende Flexibilität genannt.

Andererseits gibt es eine Vielfalt von Anwendungen für Graphgrammatiken: In vielen Problemstellungen der Informatik existieren bereits Strukturen, welche oftmals im Aufbau ohnehin graphenähnlich sind oder sich leicht in geeignete Form bringen lassen, sodass sich die Beschreibung, Darstellung und Bearbeitung in Form von Graphen geradezu anbietet. In solchen Fällen ist es unbestreitbar sinnvoll, das Prinzip der Graphgrammatiken zur Modellierung und Problemlösung heranzuziehen ([2], [3]).

Bekannte Beispiele für geeignete Bereiche sind etwa Bilderkennung, Datenbanken, Steuerungen oder neuronale Netze. In diesen Bereichen bringt der Ansatz mittels Graphgrammatiken eine erhöhte Übersichlichkeit und verbesserte Möglichkeit zur Visualisierung komplexer Zusammenhänge, welche im konventionellen Programmieransatz nicht in dieser Form gegeben ist.

Die Effizienz kann aus Mangel an praktischen Beispielen schwer bewertet werden, jedoch hängt die Gesamteffizienz eines Programmes oder Algorithmus nicht nur von der Ausführungseffizienz des Programmiersystems ab, welche zweifelsfrei optimiert werden kann, sondern auch von der Entwicklungseffizienz. Eine problemorientierte Entwicklungsumgebung kann daher die gesamte Effizienz erheblich erhöhen, sofern sie die Bedürfnisse des Entwicklers berücksichtigt. Eine Effizienzsteigerung in der Behandlung graphenähnlicher Strukturen eben in Form von Graphen ist sicherlich unbestreitbar.

Für jede Art von Programmentwicklung (ob auf Basis von Graphgrammatiken oder nicht) ist eine geeignete, durchschaubare, klar definierte Entwicklungsumgebung essentiell. Wir wollen nun eine mögliche Entwicklungsumgebung vorgestellen, die für die spezifischen Notwendigkeiten für die Entwicklung von auf Graphgrammatiken basierenden Programmsystemen konzipiert ist.

UPGraDE ([4]) ist ein Beispiel für eine Entwicklungsumgebung, die den Entwickler durch eine universelle aber einfach zu verstehende Graphgrammatikprogrammiersprache befähigt, komplexe Graphgrammatikprogramme zu entwickeln und zu überprüfen.

Die zugrundeliegenden Graphproduktionen erlauben das Anlegen bzw. Löschen eines Knotens und das Abändern des Labels und der Attribute eines Knotens sowie das Anlegen bzw. Löschen einer Kante zwischen zwei Knoten und das Abändern des Labels und der Attribute einer Kante und der damit verbundenen Knoten.

Das Programm wird in Form eines Kontrollgraphen in das Graphsystem eingebracht und zur Laufzeit von einem Graphgrammatikinterpreter abgearbeitet. Dies ermöglicht unter anderem die Abänderung des Programmes zur Laufzeit durch das Programm selbst. Die Programmerstellung wird durch einen flexiblen Compiler unterstützt, der eine den gängigen höheren Programmiersprachen angepasste Syntax (Pascal, C, etc.) aufweist und damit auf dieser abstrakten Ebene strukturierte Programmierung ermöglicht.

Attributfunktionen werden in Form von Funktionsmodulen zur Verfügung gestellt, die bei Bedarf einfach erweitert oder mit beliebiger Komplexität kombiniert werden können.

Ein- und Ausgabekomponenten können einfach gestaltet und erweitert werden, sodass die Entwicklung einer geeigneten graphischen Benutzerschnittstelle sehr vereinfacht wird.

Ein spezielles Modul unterstützt durch Einzelschrittsimulation und ausführliche Darstellung des Graphsystems sowie der an einer Produktion beteiligten Komponenten das Auffinden von Programmierfehlern. Auch ist durch dieses Modul eine Präsentation und Erläuterung eines Programmes zu Lehrzwecken leicht möglich.

Alle Komponenten, selbst Compiler und Graphgrammatikinterpreter, sind modular aufgebaut und können daher bei Bedarf erweitert oder ausgetauscht werden.

Die Entwicklungsumgebung selbst basiert auf einem äusserst stabilen aber flexiblen Betriebsystem (OPENSTEP) und ist in einer objektorientierten Programmiersprache gestaltet (Objective-C). In Planung befinden sich folgende Erweiterungen:

1. Eine Makrosprache, die es dem Entwickler ermöglicht, sich seine eigenen Befehle selbst zu gestalten (eigene Programmiersprache).

2. Ein flexibles Konzept für Unterprogramme mit Parameterübergabe.

3. Ein erweitertes System zur raschen Erstellung von Attributfunktionen, sodass die Entwicklung von Funktionsmodulen weitgehendst überflüssig wird.

4. Ein graphisches Darstellungsmodul zur besseren Visualisierung der Graphstrukturen.

Literatur

[1] G. Engels, C. Lewerentz, and W. Schäfer: *Graph Grammar Engineering - A Software Specification Method.* H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld (editors): Graph Grammars and Their Applications to Computer Science. LNCS 291, Springer, 1987.

[2] R. Freund, B. Haberstroh, and C. Stary: *Applying graph grammars for task-oriented user interface development.* Proceedings IEEE Conference on Computing and Information ICCI'92, 361-365, 1992.

[3] R. Freund, *Software spezification based on graph grammars,* Fifth International Workshop on Graph Grammars, Williamsburg, USA, November 1994

[4] H. Pötzl, UPGraDE, Diplomarbeit, September 1997.

[5] A. Schürr: *A VHL-Language Based on Graph Grammars.* H. Ehrig, H.-J. Kreowski, and G. Rozenberg (editors): *Graph Grammars and Their Application to Computer Science.* LNCS 532, Springer, 1991.