

Rückgewinnung von Syntax und Semantik zur Analyse von Visual Basic 6 Programmen

Jan Harder

Arbeitsgruppe Softwaretechnik
Fachbereich 03, Universität Bremen

[http://www.informatik.uni-bremen.de/st/
harder@informatik.uni-bremen.de](http://www.informatik.uni-bremen.de/st/harder@informatik.uni-bremen.de)

20. April 2007

1 Einführung

Visual Basic ist eine BASIC-Variante, die für die Programmierung von Windows-Applikationen erweitert wurde und in der kommerziellen Softwareentwicklung breite Verwendung findet. Im Rahmen der Anpassung an das .NET-Framework erfolgte eine Neuausrichtung der Sprache mit so tief greifenden Veränderungen an Syntax und Semantik, dass die Abwärtskompatibilität zu den früheren Versionen gebrochen wurde. Die Migration von pre-.NET-Programmen ist nur zum Teil automatisiert möglich und erfordert die manuelle Anpassung von Programmsequenzen, deren Semantik sich geändert hat. Zudem sind die ehemals verwendeten Systembibliotheken durch Aufrufe der .NET-Klassenbibliothek zu ersetzen, die über eine andere API verfügt und damit auch andere Nutzungsprotokolle voraussetzt.

Um die Migration leisten zu können, ist daher eine umfassende Kenntnis der Software und der darin bestehenden Abhängigkeiten notwendig. Bei Legacy-Systemen ist das Wissen über die Implementierung jedoch oft begrenzt, da die Software vor geraumer Zeit geschrieben wurde, Dokumentation fehlt oder nie erstellt wurde. Ebenso können die einstigen Entwickler das Unternehmen mitsamt ihres Wissens verlassen haben. Der entstehende Aufwand für das Programmverstehen, der schon bei den üblichen Aufgaben in der Softwarewartung etwa die Hälfte der Zeit beansprucht [1], ist hier also besonders hoch und unterstützende Analysen und Werkzeuge um so wertvoller.

Das Bauhaus-Projekt der Universitäten Bremen und Stuttgart [5] hat es sich zum Ziel gesetzt Softwareingenieure bei den Aufgaben des Programmverstehens zu unterstützen. Der *Resource Flow Graph* (RFG) ermöglicht es, unabhängig von der Programmiersprache die architektonisch relevanten Elemente von Programmquelltexten selektiv darzustellen und verschiedene Analysen durchzuführen, die Aufschluss über Struktur und Qualitätsaspekte der Software geben. Hierbei nimmt insbesondere die Rückgewinnung der Systemarchitektur aus den Quelltexten eine zen-

trale Rolle ein. Um die Migration zu den neuen .NET-Versionen zu unterstützen, wurde ein Analyserwerkzeug realisiert, das den RFG für Visual-Basic-Programme der Version 6 – der letzten vor der .NET-Umstellung – erstellt.

2 Ausgangssituation

Zur Generierung des RFG müssen typische Aufgaben eines Compiler-Frontends durchgeführt werden, indem zunächst die Quelltexte geparkt und dann die darin verwendeten Typen und Bezeichner aufgelöst werden. Zwar ist reichlich Dokumentation zu Visual Basic 6 verfügbar, allerdings fehlen Informationen, die für die Analyse unverzichtbar sind. So gibt es weder eine vollständige und korrekte Grammatik – die einzige Referenz ist der Compiler, dessen Quelltexte nicht einsehbar sind. Noch sind die Regeln, denen die Namensauflösung zugrunde liegt, im Detail beschrieben. Dieses undokumentierte Wissen über die Sprache musste schrittweise in experimentellen Verfahren wiedergewonnen werden, um die Analyse zu ermöglichen. Die dazu eingesetzten Vorgehensweisen werden im folgenden beschrieben.

3 Rückgewinnung der Grammatik

Ralf Lämmel und Chris Verhoef haben eine Vorgehensweise zum *Grammar-Recovery* für einen COBOL-Dialekt beschrieben und angewendet [2]. Hierbei wird aus den verfügbaren Informationen, wie etwa den Handbüchern, eine Ausgangsgrammatik erstellt. Diese wird schrittweise verfeinert, indem ein Parser für die Grammatik erzeugt wird, mit dem Programme der Sprache, deren Korrektheit durch den offiziellen Compiler sichergestellt werden kann, analysiert werden. Dabei auftretende Fehler werden behoben und der Test mit den Beispielprogrammen so lange wiederholt, bis diese fehlerfrei analysiert werden können.

Im Fall von Visual Basic bot die offizielle Referenz [3] einen Ausgangspunkt zum Erstellen der initialen Grammatik. Hier ist die Syntax einzelner Anweisungen beschrieben, jedoch nicht wie sich diese zu einem korrekten VB6-Programm zusammensetzen. Die

ser fehlende Teil der Grammatik musste zunächst aufgrund von Annahmen und Erfahrungen mit anderen Sprachen gestaltet werden. Einige hilfreiche Informationen lieferten hierbei partielle Grammatiken anderer Autoren.

Als Testdaten diente eine große Menge industriellen Visual Basic 6 Codes mit mehr als 800.000 SLOC. Als besondere Erschwernis erwies sich bei der Verfeinerung der Grammatik die Unvollständigkeit und teils auch Fehlerhaftigkeit der offiziellen Dokumentation, die viele Relikte alter BASIC-Versionen, die noch immer unterstützt werden, verschweigt.

Vieles deutet zudem darauf hin, dass der VB6-Compiler nicht auf einer formalen Grammatikdefinition beruht sondern von Hand geschrieben wurde. Beispielsweise lässt sich kein allgemeines Regelwerk erkennen, nach dem sich entscheidet, ob ein Schlüsselwort reserviert ist oder nicht. Je nach Kontext, in dem ein Bezeichner verwendet wird, unterscheidet sich die Menge der reservierten Schlüsselwörter stark und ist offenbar willkürlich gewählt.

Dies spiegelt sich auch in einer Vielzahl von Mehrdeutigkeiten in der wenig restriktiven Syntax der Sprache wieder, die beim Parsen oft einen großen Lookahead erfordern. Daher wurde zur Implementierung des Parsers mit *ANTLR* [4] ein Generator verwendet, der es erlaubt, in Einzelfällen einen beliebigen Lookahead zu verwenden, der vom fest gewählten abweicht.

4 Nachbildung der Namensauflösung

Während für die Grammatik mit der Sprachreferenz ein hilfreicher Ausgangspunkt bestand, fehlte für die Rekonstruktion von Visual Basics Namensauflösung eine solche Quelle. Stattdessen fanden sich lediglich einige verstreute Hinweise in der Literatur, oft in Form von Anmerkungen, dass bestimmte Bezeichner eindeutig sein müssen. Bedeutende Hintergrundinformationen, etwa wie sich der globale Namensraum bei Namenskonflikten zusammensetzt, fehlten gänzlich.

Hier wurde zunächst ein initiales Modell auf der Grundlage von Beobachtungen, gezielten Tests und Erfahrungen mit anderen Sprachen erstellt. Die Identifikation von Namensräumen war durch die gezielte Generierung von Testfällen, die gleichnamige Bezeichner in unterschiedlichen Kombinationen gegenüberstellten, möglich. Testfälle, die vom VB6-Compiler abgewiesen wurden, beinhalteten Namensüberschneidungen, die die Sprache nicht erlaubt. Durch diese und weitere manuell erstellte Tests sowie dem Compiler als Validierungswerkzeug konnte so das initiale Modell konstruiert werden.

Dieses Modell wurde, analog zur Ausgangsgrammatik, anhand der Beispielprogramme getestet. Auftretende Fehler wurden schrittweise behoben. Hierbei ist generell zwischen zwei Arten von Fehlern zu unterscheiden: Namen, die nicht zugeordnet wurden, und

solche, die dem falschen Symbol zugeordnet wurden. Während erstere Laufzeitfehler verursachen und daher leicht festzustellen sind, fallen falsche Zuordnungen nur dann auf, wenn sie zu Folgefehlern in der Namensauflösung führen oder bei manuellen Stichproben entdeckt werden.

5 Ergebnisse

Die Realisierung des Analysewerkzeugs fand in einem Zeitraum von etwa drei Monaten statt. Zwar sind die angewandten Methoden nicht exakt, ebenso hängen die Ergebnisse stark von der Güte der herangezogenen Beispielprogramme ab. Weitergehende Tests mit anderen VB6-Programmen zeigen jedoch, dass das Analysewerkzeug mit nur geringen Modifikationen auch hierfür erfolgreich eingesetzt werden kann.

Die erzeugten RFGs liefern Informationen, die aus den Quelltexten nicht direkt ersichtlich, für die Migration zu .NET jedoch bedeutsam sind. So lassen sich etwa potentielle, globale Auswirkungen lokaler Änderungen, wie sie bei der Anpassung der Semantik nötig sind, abschätzen, indem die Kontroll- und Datenabhängigkeiten der geänderten Funktionen untersucht werden. Die Architektur-Rückgewinnung ermöglicht es zudem geeignete architektonische Komponenten für eine inkrementelle Migration zu identifizieren. Dies ist insbesondere bei größeren Legacy-Systemen hilfreich, bei denen eine vollständige Migration in nur einem Schritt nicht möglich ist.

Zukünftig ist geplant den RFG auch für .NET-Programme zu generieren. Hiermit wird es möglich sein die vorhandenen Bauhaus-Werkzeuge zu nutzen, um die Einhaltung der Architektur während der Migration durch Reflexionsanalysen zu überwachen.

Literatur

- [1] R. K. Fjeldstadt and W.T. Hamlen. Application program maintenance study: Report to our respondents. In *Proceedings of GUIDE 48*, 1983.
- [2] Ralf Lämmel and Chris Verhoef. Semi-automatic Grammar Recovery. *Software - Practice and Experience*, Vol. 31(15):1395–1438, Dezember 2001.
- [3] Microsoft Developer Network Library - Visual Basic 6. Webseite, April 2007. <http://msdn2.microsoft.com/en-us/library/ms950408.aspx>.
- [4] Terence Parr and R. W. Quong. Antlr: A Predicated-LL(k) Parser Generator. *Software - Practice and Experience*, Vol. 25(7), Juli 1995.
- [5] Aoun Raza, Gunther Vogel, and Erhard Plödereder. Bauhaus - a tool suite for program analysis and reverse engineering. In *Reliable Software Technologies - Ada-Europe 2006*, pages 71–82, Juni 2006.