Expression Library

Studiengang: MAS BI BE Information Technology
Betreuer: Rolf Wenger
Experte: Dr. Beatrice Amrhein
Industriepartner: weroSoft AG, Ostermundigen

Die Expression Library ist eine eigenständige Funktionalität zur typsicheren Überprüfung und Ausführung von Expressions zur Laufzeit. Für das Parsing der Expressions wird ein rekursiver Top-Down Algorithmus mit Operatorpräzedenz auf Basis des Pratt-Parsers verwendet.

Stephan Müller
ste.muelle@gmail.com

Ausgangslage

Die weroSoft AG ist ein Software-Dienstleister auf Basis der Microsoft-Technologien. Die Produktepalette der Firma beinhaltet unter anderem ein serverseitiges Framework mit dem Namen Heron. Damit in Heron Workflows zur Laufzeit erstellt und ausgeführt werden können, hat die weroSoft eine allgemeingültige domänenspezifische Sprache entwickelt. In dieser Sprache sind Expressions ein wichtiges Element, weil sie die Möglichkeit zur Dateneingabe und die dynamischen Aspekte der Sprache abdecken. Expressions sind Zeichenfolgen, die anhand einer Benutzereingabe erstellt werden können. Nachfolgend sind drei zufällige Beispiele einer Expression gezeigt:

- 2 + 3 * 4
- Math.PI * Math.Sin(0.1)
- ((++integerVariable + 3.1) * 0x2) / arrayVariable[1]

Die in diesem Artikel beschriebene Arbeit hatte zum Ziel, die weroSoft Softwarebibliothek mit einer eigenständigen Funktionalität zu erweitern, welche Expressions zur Laufzeit überprüfen und auswerten kann.

Anforderungen

Wichtig ist, dass die Überprüfung der Syntax und die Auswertung einer Expression unabhängig voneinander durchgeführt werden können. Die Lösung muss Expressions beliebiger Grösse und Verschachtelungstiefe verarbeiten können. Solch eine Expression kann Konstanten, Variablen, Typen (grundlegende, .NET und dynamische), Typ-Varianten (Arrays, Aufzählungen) und Operatoren beinhalten, die allesamt von der Lösung unterstützt werden müssen. Operatoren muss die Lösung unter Beachtung ihrer Präzedenzregeln verarbeiten. Hat eine Expression eine fehlerhafte Syntax, so muss die Lösung eine exakte Beschreibung mindestens des ersten Fehlers ausgeben. Wichtig ist auch die Performance der Lösung. Die Zeitdauer für die Evaluation eines einzelnen Terms muss unterhalb einer Millisekunde liegen.

Algorithmus

Vor der eigentlichen Implementation wurde ein geeigneter Parsertyp evaluiert. Dabei wurde der sogenannte Pratt-Parser entdeckt, dessen Verwendung sich als sehr nützlich und geeignet erwies. Der Pratt-Parser ist ein rekursiver Top-Down Parser, der anhand der Präzedenz der Operatoren, welche eine Expression beinhaltet, die Expression selbst parsed. Zusätzlich zur Präzedenz braucht der Parser nur den Typ (Präfix oder Nicht-Präfix) von jedem Operator zu kennen. Die restliche Logik befindet sich auf dem Operator selbst. Damit ist dieser Algorithmus effizient, das heisst, er besteht aus wenig Code und ist zugleich dynamisch. Von dieser zweiten Eigenschaft konnte die Lösung profitieren, indem die Operatoren in eine oder mehrere eigenständige Programmbibliotheken ausgelagert und erst zur Laufzeit hinzugeladen werden können. Somit ist konfigurierbar, welche Operatoren die Lösung kennen soll.

Lösung

Als Kern innerhalb der Lösung gibt es einen Expression Manager, der die Anfragen für einen Syntaxcheck oder eine Evaluation erhält und dann die entsprechenden internen Verarbeitungselemente aufruft. Eines dieser Elemente ist ein Tokenizer. Mit dessen Hilfe wird eine einkommende Expression in anatomische Bestandteile der Sprache, in einzelne Token, zerlegt. Jedes Token wird einem von vier Token-Basistypen zugeordnet und erhält, falls nötig, einen Modifizierungswert. Danach nimmt der Tokenizer einen ersten, unvollständigen Syntaxcheck vor. Eine korrekte Tokenliste wird dem zweiten wichtigen Element, dem Parser, zur Verarbeitung übergeben. Während des Parsevorgangs wird zugleich ein Syntaxbaum (Parsetree) erstellt. Um Variablen, Typen und Typbestandteile zu erkennen kann der Parser auf zwei Schnittstellen zugreifen. Einerseits auf einen Variablenmanager, der für diese Lösung gemocked wurde, andererseits auf einen bereits bestehenden Typmanager. Nach Abschluss des Parsens folgt wiederum ein Checkdurchlauf. Ist dieser erfolgreich, kann die Expression evaluiert werden, indem der gebaute Syntaxbaum durchlaufen wird.
Entwickelt wurde die Lösung mit der Sprache C# auf Basis von .NET 4.5. Um die Korrektheit der Lösung zu gewährleisten, wurden Unit-Tests mit einer Codeabdeckung von 96 Prozent durchgeführt.