Neues bei C# 4.0

Deklarativ, dynamisch, parallel

Was wird C# 4.0 Neues bringen? dotnetpro hat demjenigen zugehört, der es am besten wissen muss: Anders Hejlsberg, leitender Architekt für C# bei Microsoft, hat auf der PDC 2008 in einem Vortrag die kommenden Änderungen bei C# 4.0 vorgestellt. dotnetpro fasst die wesentlichen Neuigkeiten zusammen.

Auf einen Blick

Neno Loje ist strategischer Berater für den Softwareentwicklungsprozess bei der AIT AG und erster europaweiter Microsoft Most Valuable Professional für Visual Studio Team System. Sie erreichen ihn über seine Website www.team-systempro.de.

Inhalt

Neuigkeiten bei C# 4.0:

- dynamisch typisierte Objekte,
- optionale und benannte Parameter,
- verbesserte Interoperabilität mit COM,
- Ko- und Kontravarianz.

dnpCode

A0902Csharp40

Schon zehn Jahre ist es her, dass sich Hejlsberg im Dezember 1998 mit seinem Team zurückzog, um eine neue Sprache speziell für das Programmieren von verwaltetem Code zu entwerfen. Dabei herausgekommen ist C# 1.0. Das Folge-Release, C# 2.0, brachte die Funktionalitäten, für deren Umsetzung es in der ersten Version einfach nicht mehr genügend Zeit gab.

Mit der dritten Fassung, C# 3.0, bot sich die erste Möglichkeit, neue Dinge in die Programmiersprache mit aufzunehmen. Der Schwerpunkt war schnell gefunden, man wollte die Kluft zwischen der Welt der universellen Programmiersprachen und der Datenbanken verringern. Das Resultat war LINQ, die Language Integrated Query. Dafür wurden neue Funktionen in die Sprache eingebaut.

Was kommt als nächstes? Zur grundsätzlichen Orientierung kann man einen Blick auf diejenigen Themen werfen, die derzeit die Softwareindustrie und auch die Sprachdesigner bewegen. Hier zeichnen sich drei Trends ab:

- *Bewegung zu einem mehr deklarativen Programmierstil.* Die klassische, strikte Einteilung von Programmiersprachen in „funktional", „strukturiert", „imperativ" oder ähnliches beginnt zu bröckeln. Künftig werden die Sprachen eine Mischung aus verschiedenen Paradigmen beinhalten. C# 3.0 hat beispielsweise auch Bausteine, die eine dynamische Sprache ausmachen. Traditionell ist die imperative Programmierung der Mainstream. Dabei legen Entwickler einerseits fest, was gemacht werden soll. Andererseits beschreiben sie aber auch sehr detailliert, *wie* sie an das Ziel kommen wollen, etwa indem sie Variablen deklarieren, Schleifen durchlaufen und so weiter. Damit war die Lösung für das eigentliche Problem häufig überdimensioniert. Beim Blick auf den Code war die Logik nur noch mühsam erkennbar, und auch die Infrastruktur kann nicht erkennen, wie sie diesen Code am effizientesten ausführen kann. Der Just-in-time-(JIT)-Compiler kann dann nur Zeile für Zeile ausführen und kaum optimieren. Verwendet man hingegen deklarative Anweisungen, wie etwa bei LINQ, dann ist die Chance viel größer, dass diese auf eine clevere Art und Weise ausgeführt werden. Auch in C# wird es künftig mehr deklarative Programmierung geben. C# 3.0 war der erste Schritt in diese Richtung.
- *Dynamische Programmiersprachen.* Die Frage nach statischer Programmiersprache versus dynamischer Sprache ist nicht selten eine "religiöse". Fakt ist: Beide Arten haben gute Eigenschaften, und mit beiden werden heute produktive Anwendungen entwickelt. Am liebsten hätte man das Beste aus beiden Welten, und dieser Trend hat schon begonnen. So bieten diverse dynamische Sprachen die Möglichkeit, statische Typen zu verwenden.
- *Nebenläufigkeit und Parallelisierung.* Das Mooresche Gesetz hat es Entwicklern bisher einfach gemacht, sich nur wenig um das Problem der Nebenläufigkeit zu kümmern, wurden doch die Prozessoren regelmäßig schneller. Doch damit ist nun Schluss – irgendwo zwischen 2,5 und 3 GHz ist die Grenze erreicht worden. Zwar funktioniert das Mooresche Gesetz immer noch, aber es gilt nun für die Anzahl der Prozessoren – die für sich genommen nicht mehr schneller werden. Nebenläufigkeit und Parallelarbeit zu programmieren ist heute keine triviale Aufgabe. Bei Mehrkernprozessoren besteht die Herausforderung darin, eine einzelne logische Aufgabe in kleine Arbeitspakete aufzuteilen, sodass diese von mehreren Prozessoren zeitgleich bearbeitet werden können. Die schlechte Nachricht lautet also: Eine Anwendung muss explizit so programmiert worden sein, damit diese parallel ablaufen kann. Einen Compilerschalter */parallel*, der uns die Arbeit abnimmt, wird es leider nicht geben.

Visual C# versus Visual Basic

Eine interessante Randbemerkung von Hejlsberg betrifft die Weiterentwicklung der beiden Sprachen C# und VB (für .NET). Nach unterschiedlichen Begründungen seitens Microsoft, wofür man die eine oder andere Sprache nehmen solle, akzeptiere man nun die Realität, dass beide Sprachen für alle Arten von Anwendungen genutzt werden und die Wahl eher damit zusammenhängt, was man vorher programmiert hat. Damit fördert Microsoft die Co-Evolution: Beide Sprachen werden parallel weiterentwickelt. Es wird