iX im Gespräch mit Scott Meyers

Die Zukunft von C++

Seit Jahren ist Scott Meyers als C++-Experte bekannt, und sein Buch „Effective C++" dürfte in den Regalen vieler C++-Programmierer zu finden sein. Im Gespräch mit *iX* gibt er eine Einschätzung der Bedeutung, die die objektorientierte Sprache heute noch hat, zusammen mit einem Ausblick auf den kommenden Standard C++0x.

iX: Kann es sein, dass C++ mittlerweile etwas in die Jahre gekommen ist?

Scott Meyers: In gewisser Weise handelt es sich um eine altmodische Sprache, die mehr darauf abzielt, vorhandene Ressourcen effizient zu nutzen und Programmierern eine weitgehende Kontrolle zu gewähren, als darauf, die Produktivität zu steigern. Das steht im Gegensatz zu neueren Sprache wie Java, C# und Ruby/Rails. Die kontrollierten Umgebungen und umfangreichen Funktionen einer Standardbibliothek, die diese Sprachen zur Verfügung stellen, sind für viele Anwendungen sehr reizvoll. Gleichzeitig sind sie im Vergleich zu C++ weniger leistungsfähig, wenn es um Dinge geht wie Hardwareschnittstellen, Konformität zu Schnittstellen von Legacy-Applikationen oder Data Layouts, den initialen Speicherbedarf möglichst gering und die Performance möglichst hoch zu halten. Ich denke, es wird immer Bedarf nach einer Sprache bestehen, die ihre Stärken in diesen Bereichen hat. Daher ist es vielleicht besser, C++ unter dem Gesichtspunkt zu betrachten, dass sie den klassischen Vorstellungen einer schönen Programmiersprache entspricht, statt zu versuchen, sie in das aktuell vorherrschende Modeschema für Programmiersprachen zu pressen.

iX: Was denken Sie über andere Sprachen wie Java oder C#, die immer wichtiger werden. Glauben Sie, dass C++ wegen Java/C# auf einen begrenzten Markt beschränkt wird?

S. M.: Schon seit Jahren vertrete ich die Ansicht, dass C++ seine natürliche Nische in der Entwicklung anspruchsvoller Systemanwendungen hat und meiner Meinung nach noch viele Jahre behalten wird. Dieser Einsatzbereich ist begrenzt, aber natürlich hat auch der Part, den die auf den ersten Blick produktiveren Sprachen übernehmen, seine Grenzen. Ich denke, dass C++ für einige Gebiete weniger wichtig sein wird (beispielsweise für die Enwicklung gängiger Desktop-Anwendungen). Gleichzeit wird C++ auf anderen Gebieten zunehmend an Bedeutung gewinnen (etwa für die Entwicklung von Embedded-Systemen, wo C++ mehr und mehr zur Sprache der Wahl wird und C ersetzt).

iX: Wird der Embedded-Markt der neue Markt für C++ werden?

S. M.: Ich weiß nicht, ob er der Hauptmarkt sein wird, aber er wird ganz sicher eine sehr wichtige Rolle spielen. C++ wird jedoch auch weiterhin sehr stark auf den Gebieten vertreten bleiben, auf denen die Performance maßgeblich ist, beispielsweise Videospiele,

Schon seit Jahren vertrete ich die Ansicht, dass C++ seine natürliche Nische in der Entwicklung anspruchsvoller Systemanwendungen hat.

Bildverarbeitung, Finanz-Applikationen (etwa die Echtzeitanalyse von Investitionsmärkten) sowie Simulationen diverser Art. Außerdem wird C++ weiterhin den Markt für Template-Metaprogrammierung (siehe Kasten) beherrschen. Zurzeit ist das ein winziger Markt, und ich glaube auch nicht, dass er jemals groß sein wird, aber meiner Meinung nach wird er sicherlich immer mehr an Bedeutung gewinnen.

iX: Was halten Sie von dem kommenden Standard C++0x?

S. M.: Soweit ich das sagen kann, wird er wahrscheinlich wenigstens ein paar recht interessante neue Sprachmittel bieten, darunter statische Assertions, *auto* und Lambda-Funktionen. Er wird aber voraussichtlich auch Konzepte anbieten, die ich persönlich nicht sonderlich reizvoll finde, über die andere jedoch ganz aus dem Häuschen zu sein scheinen. Dazu kommen sehr schöne Verbesserungen der Standard Library, die durch TR1 bereits abzusehen sind. Meine Favoriten dabei sind *shared_ptr*, Hash-Tabellen und reguläre Ausdrücke. Aber auch die anderen neuen Bibliotheksfunktionen werden sehr hilfreich sein, besonders für diejenigen, die die STL viel nutzen oder mit Template-Metaprogrammierung zu tun haben.

iX: Glauben Sie, dass C++ durch C++0x einfacher werden wird?

S. M.: Nicht in irgendeiner Weise, die von Bedeutung ist. C++ ist eine komplexe Sprache. Das lässt sich nicht ändern.

iX: In der dritten Auflage von „Effective C++" sprechen Sie häufig über TR1. Was ist TR1 und warum ist es für C++-Programmierer von Bedeutung?

S. M.: TR1 ist ein Technikbericht, der 14 neue Arten von Bibliotheksfunktionen spezifiziert, die voraussichtlich mit C++0x Teil der Standard Library werden. Von diesen 14 sind bereits 13 Teil des C++0x-Entwurfs. Es ist jedoch wahrscheinlich, dass das, was schließlich in C++0x eingehen wird, sich in einigen Punkten von der TR1-Spezifikation unterscheiden wird. Das liegt daran, dass TR1 darauf beschränkt ist, C++ so zu nutzen, wie es der Standard von 1998/2003 definiert. Die Standard Library für C++0x hingegen kann von all den neuen, von C++0x hinzugefügten Sprachmerkmalen profitieren.

TR1 ist für C++-Programmierer von Bedeutung, weil es ihnen einen Vorge-

schmack auf wichtige Funktionen gibt, die im Lauf der nächsten Jahre Teil jeder C++-Implementierung sein sollten. Für diejenigen, die das Funktionsspektrum bereits jetzt nutzen möchten, gibt es Implementierungen der meisten TR1-Komponenten von Anbietern wie Boost oder STLPort. Solche Implementierungen unterscheiden sich gelegentlich von der TR1-Spezifikation, allerdings im Allgemeinen nur geringfügig.

iX: Neue Hardwarearchitekturen und Multithreaded-Anwendungen werden an die Sprache C++ neue Anforderungen stellen. Gibt es da Hoffnung?

S. M.: Es ist sehr wahrscheinlich, dass C++0x ein Speichermodell definieren wird, auf dessen Basis sich Anwendungen programmieren lassen, deren Ausführung über mehrere Plattformen hinweg in mehreren Threads erfolgt. In erster Linie ist ein Speichermodell für Compiler-Entwickler und für Programmierer, die Bibliotheken schreiben, relevant, aber auch jeder andere, der C++ nutzt, wird die Auswirkungen spüren. Allerdings habe ich bisher nichts in der Richtung gehört, dass die Threading-Funktionen, die Endbenutzern standardmäßig zur Verfügung stehen werden, über das hinausgehen werden, was zurzeit über Bibliotheken wie Posix verfügbar ist. Daher scheint es mir momentan nicht wahrscheinlich, dass sich die Programmierung mit Threads in C++0x groß von der im jetzigen C++ unterscheiden wird. Andererseits bin ich nicht im Standardisierungskomitee. Es ist also möglich, dass dieses letztendlich viel ambitioniertere Pläne verfolgt.

C++ wird jedoch auch weiterhin sehr stark auf den Gebieten vertreten bleiben, auf denen die Performance maßgeblich ist, beispielsweise Videospiele, Bildverarbeitung, Finanz-Applikationen.

iX: Wird es ein neues Buchprojekt geben (beispielsweise eins über „Keyholes“) oder ein Update auf CD von „Effective C++“?

S. M.: Ich plane ständig neue Bücher, allerdings werden die meisten nie geschrieben :-). Dennoch hoffe ich, nächstes Jahr ein neues Buch zu schreiben, und ein Punkt auf meiner To-do-Liste für 2007 ist, meine „Keyhole“-Arbeit in irgendeiner Weise schriftlich festzuhalten. Das kann ein Buch, es kann aber auch etwas anderes sein. Das weiß ich noch nicht.

Mein Herausgeber und ich haben darüber gesprochen, eine überarbeitete elektronische Version meiner Bücher zu veröffentlichen, die sowohl die neueste Ausgabe von „Effective C++“ als auch „Effective STL“ enthält (das nicht auf der aktuellen CD ist). Wir würden das beide gern tun, aber es gibt noch offene Fragen bezüglich des Formats (HTML, PDF, beide, keins von beiden). Ich hoffe, diese Punkte lassen sich möglichst bald klären, damit wir mit dem Projekt vorankommen.

ix: Haben Sie vor, sich neben C++ noch mit etwas anderem zu befassen?

S. M.: Ich beschäftige mich sowohl mit „Keyholes“ als auch mit sprachunabhängiger Softwarequalität, und ich will auch beides weiterverfolgen. Aber ich gehe davon aus, dass ich mich weiterhin sehr für C++ engagieren werden. Ich finde zunehmend Interesse daran, Programmierern von Embedded-Anwendungen dabei zu helfen, von C++ zu profitieren, und ich gehe davon aus, dass ich zukünftig in dem Bereich viel Arbeit investieren werde. (ka)

Das Interview für iX führte Bernhard Merkle. Die Übersetzung stammt von Kersten Auel.

Etwas Hintergrund

Template-Metaprogrammierung: Sie bezeichnet die Technik, mit Hilfe von C++-Templates neuen C++-Programmcode zu generieren. Im Prinzip machen C++-Templates den Compiler zum „freikonfigurierbaren“ C++-Codegenerator. Normalerweise produziert der Compiler nur fest vorgegebene Funktions- oder Klassen-Rümpfe aus den C++-Templates. Die Template-Metaprogramming erweitert diesen Mechanismus extensiv.

Dies hat diverse Vorteile: Da die Auswertung von Variablen oder Funktionen während der Kompilierung und nicht zur Laufzeit erfolgt, hat das Programm eine bessere Performance. Durch die Typesicherheit ist es sehr flexibel (die Boost Library verwendet diesen Mechanismus ebenfalls).

Demgegenüber stehen einige Nachteile: Die Fehlermeldungen des Compilers sind schwer verständlich (nicht genügend Sprachmittel). Das gilt selbst für einfach aussehende Konstrukte, wenn man sie im Detail betrachtet.

C++0x: C++0x wird der „nächste“ C++-Standard sein (nach dem 2003 erschienenen Standard ISO/IEC 14882:2003). Der Name C++0x (manchmal auch C++200x) ist eine inoffizielle Abkürzung und deutet das geplante Zieldatum an. Der Standard wird verschiedene neue Sprachmittel bieten, darunter statische Assertions, *auto*- und Lambda-Funktionen und sogenannte „Concepts“. Außerdem soll er einige „Defekte“ der Sprache beheben (beispielsweise „> >“ bei geschachtelten Templates).

Darüber hinaus wird den Standard-Bibliotheken – nicht nur wie bisher der STL (Standard Template Library) – wesentlich mehr Beachtung geschenkt werden. Diese Arbeit läuft momentan separat unter TR1.

TR1: Dieser Technical Report spezifiziert eine Familie von Bibliotheksfunktionen. Momentan findet man 14 verschiedene Arten, die sich in unterschiedliche Bereiche gliedern, darunter:

– Algorithmen und Datenstrukturen. Im Vergleich zur STL sind sie komfortabler und wesentlich erweitert.

– Bibliotheken für reguläre Ausdrücke, wiederkehrende Aufgaben, beispielsweise Smart Pointer .

– Die Bereiche Mathematik und Numerik.

– Generische Programmierung und Template-Metaprogrammierung sind integraler Bestandteil von TR1.

Momentan bieten Boost (www.boost.org) und STLPort (www.stlport.org) Implementierungen der TR1-Funktionen an. Insgesamt zehn der Boost Libraries decken den TR1-Bereich ab. Aller Voraussicht nach wird das C++ Standards Committee diese Bibliotheken in den nächsten C++-Standard übernehmen. Weitere Boost-Bibliotheken sind schon für TR2 vorgeschlagen.

Keyholes: Hierbei handelt es sich um Beschränkungen in Software-Systemen, die deren Qualität und Benutzbarkeit stark beeinträchtigen. Die Metapher des Schlüssellochs verdeutlicht dies (der Benutzer sieht nur einen unzureichenden Teilbereich). Beispiele für sichtbare Keyholes sind „fixed size Windows“ oder – allgemeiner – GUI-Elemente, die nur einen kleinen Teil ihrer Werte anzeigen, obwohl Platz für mehr vorhanden ist. Die Folgen sind unter anderem, dass Anwender die unsichtbaren Werte nie auswählen, wenn sie nicht umständlich über den Scroll-Balken in den verdeckten Bereich wechseln.

Es gibt aber auch zahlreiche unsichtbare Keyholes, beispielsweise unsinnige Beschränkungen von Wertebereichen, Eingabeformate oder -werte, die ein Programm nicht akzeptiert (obwohl sie fachlich sinnvoll wären), und so weiter.

(Bernhard Merkle)