dradio.de

URL: http://www.dradio.de/dlf/sendungen/computer/461232/

COMPUTER & KOMMUNIKATION

21.01.2006 · 16:30 Uhr

Auch hinter spielerisch anmutenden Anwendungen verbirgt sich ein undurchschaubarer Kode-Dschungel. (Bild: sigs-datacom.de)

Programmiertes Chaos

Riesige Softwareprojekte unterliegen natürlichen Grenzen

Von Mariann Unterluggauer

Wer einmal eine Maschinensprache lernen musste, weiß, wie mühsam es sein kann, selbst kleinste Funktionalitäten zu programmieren. Erst das Objekt orientierte Programmieren macht die Sache zu einem überschaubaren Prozess. In München widmete sich eine Tagung mit dem Problem.

In München fand diese Woche die Konferenz "OOP2006" statt. OOP steht für Objekt-orientiertes Programmieren. Vorherrschende Themen waren Software-Design, -Entwicklung und -Architektur. Unter den Vortragenden war auch der Amerikaner Richard Gabriel. Er hat nicht nur die Programmiersprache LISP mitentwickelt, die vornehmlich im Bereich "künstliche Intelligenz" eingesetzt wird, sondern galt früher auch als "Zar der Java Community". Er arbeitet derzeit mit einem sehr kleinen Team für Sun Microsystems in Kalifornien an einem militärischen Projekt mit dem Ziel, hochproduktive Computer Systeme zu entwickeln. In München kritisiert er das Festhalten an klassischen Denkstrukturen bei der Software-Entwicklung: Man träume zwar heute davon, alles mit allem zu verbinden, übersehe dabei aber, dass sich mit dem derzeit praktizierten Programmierstil derartige ultragroße Systeme gar nicht verwirklichen ließen.

Allein schon das Eintippen von Billionen Kode-Zeilen würde zu lange dauern, aber heute programmieren wir auf diese Art. Wir sind noch nicht soweit, um große Systeme zu bauen.

Man träume von Städten und baue doch nur Häuser. Im Gegensatz zur physikalischen Welt unterliegen Software-Entwickler im Grunde genommen keinerlei Sachzwängen. Man kann heute mit jeder Programmiersprache beliebige Aufgaben lösen. Das Problem sei daher nicht der Kode, sondern die mangelnde Phantasie der Entwickler. Zum Beispiel wäre für ein ultragroßes System die derzeitige Praxis der Abstraktion ein Problem.

Wir haben eine beschränkte Auffassung von Abstraktion. Die höchste Form der Abstraktion sind heute bei objektorientierten Systemen die Objekte. Denen kann man eine gewisse Verhaltensform zuweisen, aber das war es dann auch schon. Keine Programmiersprache erlaubt derzeit das Schreiben einer eigenen Kontrollstruktur. Man kann eigene Objekte definieren und auch eine eigenen Datenstruktur entwickeln, aber es ist dem Programmierer untersagt, über Kontrolle zu abstrahieren.

Diese Einschränkung müsse nicht sein, meint Gabriel, allerdings fehle es dem Menschen an Phantasie, wie man diese Grenze überschreiten und ein funktionierendes System mit Billionen Zeilen an Kode erschaffen kann. Es gäbe selbst auferlegte Zwänge wie zum Beispiel "Schönheit des Kodes", die die Softwareentwickler oft behindern würden. Nur dem ausführenden Computer selbst ist - wie vieles andere auch - der Begriff "Schönheit" fremd. Daher fallen Lösungen, die Computer selbst generieren - wie im Bereich des "genetischen Programmierens" - auch so erstaunlich anders aus. Man könnte langsam und in kleinen, abgegrenzten Bereichen, dazu übergehen es, mit "Genetic Programming" zu versuchen, so Richard Gabriel.

Wenn sie ein ultragroßes System bauen wollen, so überschreitet das das menschliche Vorstellungsvermögen: Es ist zu komplex

und zu groß, als dass es der Mensch selbst bewältigen könnte. Der Computer hat den Vorteil, dass er sehr viel Zeit und Leidenschaft für kleine Dinge aufbringt. Genetic Programming hat bereits bewiesen, dass es in der Lage ist, ausgeklügeltere Designs zu entwickeln als der Mensch. Der beste Sortier-Algorithmus den wir derzeit haben, wurde mit Hilfe von Genetic Programming geschrieben.

Aber auch die Ansicht, dass Fehler in der Software etwas grundsätzlich Schlechtes und Beseitigungswürdiges wären, sei problematisch. Weder ist der Mensch bis heute in der Lage, fehlerfreie Software herzustellen, noch gestehen Software-Hersteller ein, dass ihre Produkte fehlerhaft sind. Manche Systeme würden ohne diese Fehler erst gar nicht funktionieren. Per se seien Fehler in der Software also nichts Schlechtes, trotzdem würde man an diesem Schwarz-Weiß-Denken festhalten. Sinnvoller, so Richard Gabriel wäre es, Informatikstudenten die Kenntnis zu vermitteln, wie man zwischen lebensbedrohenden und lässlichen Fehlern unterscheiden könne. Denn sie werden einmal definieren, was ein ultragroßes System sein soll.

Wir haben gerade erst mit unseren Forschungsarbeiten begonnen, Ich versuche dabei, das Fundament, auf dem Computing basiert, zu überprüfen und es zu erweitern, damit wir neue, besser Wege entwickeln können, um über Software nachzudenken, sie zu entwerfen und schließlich zu implementieren.