M:OOP

Autor: Grischa Ekart

Publikation: ST-Computer

Veröffentlicht: September 1990

M:OOP

Magic OOP?

Während der PC-Markt mit objektorientierten Programmiersprachen überflutet wird, hat der ST-Anwender nur wenige Möglichkeiten, sich mit diesem neuen Programmierparadigma auseinanderzusetzen. M:OOP, ein Objective-C-kompatibler Precompiler, verspricht Abhilfe.

Da sicherlich nur die wenigsten etwas über Objective-C wissen, gliedert sich der Testbericht in zwei Teile: eine kurze Übersicht der Sprache und der OOP (objektorientierte Programmierung) und den Test selber.

Objective-C

In seinem Buch [1], beschreibt Cox einen Weg, wie man die objektorientierten Elemente von Smalltalk-80 mit der Ausdrucksstärke von C verbinden kann. Das Ergebnis ist eine Sprache, die leider sehr wenig nach C aussieht, dafür aber sehr einfach zu lernen und zu implementieren ist - vorausgesetzt man kann bereits C und hat auch einen entsprechenden Compiler. Diese neue Sprache soll, wie auch OOP im allgemeinen, große Programmierprojekte erleichtern und Software wiederverwendbar machen. Cox spricht dabei von dem Software-IC-Modell. Wie die aus der Elektronik bekannten Chips, sollen Programmteile, die in sich völlig abgeschlossen sind, in große Programme eingesetzt werden. Damit wird verhindert, daß ‘Software-Räder’ zum hundertsten Mal neu erfunden werden, und man stellt sicher, daß Teile von Programmen, hier Objekte genannt, fehlerfrei sind. Objekte sind von C her bekannte Strukturen, die zusammen mit den Funktionen, die auf sie zugreifen (Methoden genannt), innerhalb eines Moduls, d.h. einer Datei definiert werden. Ausgehend von dieser Veränderung, erleichtern einige Begriffe aus der OOP-Literatur dem Programmierer das Arbeiten am Computer.

Data Encapsulation

Die in den Objekten definierten Variablen können von außen her nicht verändert werden, wohl aber durch die Methoden, die im selben Objekt zu finden sind. Das wirkt sich sehr positiv auf den Programmierstil aus, denn dadurch werden globale Variablen vermieden und das Programm spaltet sich zwangsweise in kleine, überschaubare Teile. Vorteile hat die Verkapselung der Daten auch in sehr großen Projekten, wenn die Gefahr besteht. daß sich Namen in verschiedenen Modulen gleichen.

Single Inheritance

Eine der besonderen Stärken von OOP ist die Vererbung: haben Sie zwei oder mehrere Objekte. die sich in einigen Eigenschaften gleichen, so definieren Sie erst eine Klasse (so heißen Objekte) mit genau diesen Eigenschaften und vererben diese dann weiter auf Ihre eigenen Objekte. Die Vorteile liegen auf der Hand; Redundanz, entfällt. und wenn Sie ein Objekt ergänzen: so ergänzen Sie seine Nachfolger mit. In Objective-C ist die Basisklasse das ‚Object', ein Objekt mit der minimalen Funktionalität, d.h. es enthält nur das notwendigste. Alle von Ihnen definierten Objekte erben Variablen und Methoden von dieser Klasse. Sie können nun zusätzliche Variablen und Methoden definieren, bestehende Überlagern oder löschen. Im Gegensatz zur multiplen Vererbung können hier die Objekte nur einen Vorgänger haben. Dies erhöht die Übersicht jedoch erheblich.

Polymorphism

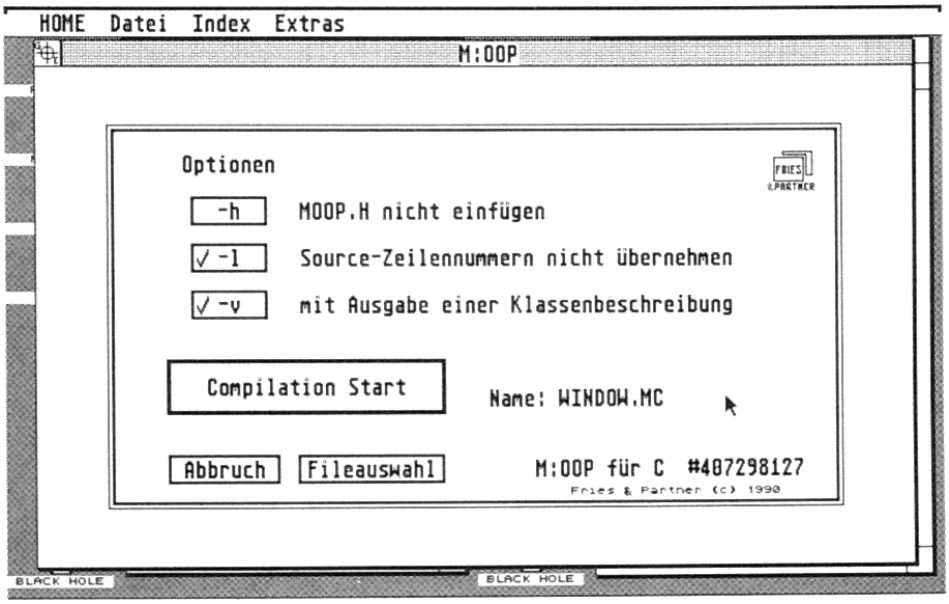
Zusätzlich zu den in C bereits bestehenden Typen wird in Objective-C ein neuer eingeführt: 'id'. Dieser Typ kann alles mögliche sein: vom Integer bis zu ganzen Objekten. Mit Hilfe dieses Typs können die sogenannten generischen Funktionen geschrieben werden. Ein Beispiel dazu: Sie schreiben eine Sortierroutine, die Elemente vom Typ 'id' sortiert. Es ist Ihnen dabei völlig egal, was 'id' nun wirklich ist; Hauptsache, es hat eine Vergleichsmethode. mit der Sie entscheiden können, welches von zwei Objekten größer ist. Nun können Sie mit Ihrer Routine Integer-Arrays. Namen und beliebige andere Objekte sortieren. Die gesamte Verwaltung übernimmt die Laufzeitumgebung des Objective-C.

Late Binding...

... ist eines der stärksten Features von OOP. Der Prozeß des Linkens wird hierbei auf die Laufzeit des Programms aufgeschoben. Erst so wird Typunabhängigkeit möglich. Eine weitere Anwendung findet sich in sogenannten Collections, einer vordefinierten Klasse. Dieser Klasse können Sie Objekte zuweisen, die zum Beispiel alle eine gemeinsame Methode haben. Wenn Sie nun diese Methode über eine besondere Funktion der Collection übergeben, wird sie jedes darin enthaltene Objekt ausführen. Insbesondere eignet sich diese Eigenschaft zur Bewältigung von Aufgaben, die mit der graphischen Oberfläche zu tun haben.

M:OOP

Alle oben aufgeführten 'Features' werden auch von M:OOP, einem Produkt der Berliner Firma Fries und Partner, unterstützt. Neben der Programmdiskette befinden sich in der Verpackung noch ein 85 Seiten starkes Handbuch und ein Lizenzvertrag. Da M:OOP nur ein Precompiler ist und C-Code generiert, braucht man auch einen C-Compiler. Die Version 2.0 unterstützt Mark-Williams-C und Turbo C. Neben den beiden Anpassungen befinden sich auf der Diskette die Sourcecodes für die acht Standardklassen und für ein größeres Programmbeispiel, eine Bibliothek, die die Standardklassen und die Laufzeitumgebung enthält, und, wie sollte es anders sein, der M:OOP-Compiler selbst. Dieser wird in zwei Ausführungen geliefert: einer Commandline- und einer Accessory-Version (Bild 1).



Das Handbuch geht in der ersten Hälfte auf das Programmieren in M:OOP ein. In diesem Teil werden auch die mitgelieferten Klassen detailliert erklärt. In der zweiten Hälfte ist das größere Beispiel abgedruckt, das man auch auf der Diskette findet. Der Index des Buches ist erstaunlich gut ausgefallen: alle wichtigen Begriffe sind aufgeführt.

Der Compiler...

... wird, wie oben schon erwähnt, in zwei Versionen geliefert. Wenn Sie öfters unter einer Text-Shell arbeiten, werden Sie keine Probleme beim Integrieren des Compilers in Ihre Programmierumgebung haben. Die einzige Änderung: Bevor die Source-Dateien an den C-Compiler kommen, werden sie an M:OOP geschickt. Jede Klasse, die Sie neu definieren, wird in einem besonderen Ordner in Form einer '.DAT'-Datei gespeichert. Diese entspricht ungefähr den üblichen Header-Dateien und muß in die Module, die diese Klasse benutzen, eingebunden werden. Der Ordner muß dem Compiler mitgeteilt werden, indem eine entsprechende Environment-Variable gesetzt wird. Hier treten die ersten Probleme mit der Accessory-Version auf. Es gibt hier keine Möglichkeit, diese Variable zu setzen. Der Compiler sucht seinen Ordner dann auf dem aktuellen Laufwerk, wo er sich mit größter Wahrscheinlichkeit nicht befindet. Die zu übersetzenden Dateien müssen manuell per File-Selector bestimmt werden. Das kann besonders bei größeren Projekten lästig werden. Ein weiteres Manko: Obwohl das Accessory nur um die 50kB lang ist, verbraucht es im Speicher ca. 250kB. Hier kann es einem leicht passieren, daß man den C-Compiler wegen Speichermangel nicht mehr laden kann. Was die Geschwindigkeit angeht, liegt M:OOP hinter Turbo C. Beim Übersetzen der Klasse 'Set' ergaben sich folgende Werte:

M:OOP:	271 Zeilen in 8030 ms
Turbo C:	401 Zeilen in 6140 ms

Der erzeugte Code ist kompakt und stützt sich auf die Laufzeitumgebung. Diese besteht im wesentlichen aus der Funktion *_msg*. Ihr Programm erhält dadurch einen Overhead von nur etwas über 2kB (plus die benötigten Klassen). Jede von Ihnen definierte Methode bekommt ihre eigene Nummer. Diese wird dann an *_msg* weitergegeben und intern auf die richtige Adresse umgewandelt. Die

dabei entstehenden Verzögerungen sind je nach der Programmgröße unterschiedlich, aber durchaus als gering zu bezeichnen: eine leere Methode ist ungefähr zehnmal langsamer als eine leere Funktion in Turbo C. Bei Methoden mit ‘Inhalt’ fällt dies jedoch nicht ins Gewicht.

Sechs im Handbuch beschriebene globale Variablen bilden eine sehr sinnvolle Möglichkeit, den Dispatcher zu beobachten und zu erweitern. So können alle Methodenaufrufe protokolliert, die Methodentabelle erweitert oder neue objekt-orientierte Speicherverwaltungsroutinen integriert werden.

Die Fehlermeldungen sind zwar nicht in Unmengen vorhanden, dafür aber deutsch. Leider untersucht M:OOP die Quelle nicht besonders gründlich nach Fehlern, so daß sie erst beim Compilieren mit dem C-Compiler zutage treten. Das verzögert die Arbeit beträchtlich.

Wie oben schon erwähnt, bauen alle Klassen in Objective-C auf einer Standardklasse, nämlich ‘Object’, auf. Diese und sieben andere befinden sich im Quellcode auf der Diskette. Sie stellen das Fundament der Programmierung in M:OOP dar. So sind zum Beispiel einige Array- und Collection-Klassen enthalten, die man als Anwender nicht implementieren könnte. Bevor man sie benutzen kann, muß man sie erst einmal übersetzen. Dies ist um so lästiger, da diese bereits in übersetzter Form in der Bibliothek enthalten sind.

Was die Kompatibilität zu Objective-C von Stepstone angeht, so muß gesagt werden, daß M:OOP die ‘@-Direktiven’ nicht kennt. Daher ist eine Portierung von M:OOP auf Objective-C leichter zu bewerkstelligen als umgekehrt. Alle anderen Inkompatibilitäten sind eher nebensächlich und im Handbuch beschrieben.

Fazit

Mit 198 DM können Sie nun endlich anfangen, objektorientiert zu programmieren. Die Frage ist nun sicherlich, für wen sich dieses Produkt eignet. Einerseits ist der Preis etwas hoch ausgefallen, vor allem wenn man bedenkt, daß man für den C-Compiler extra zahlen muß. Andererseits ist der Lieferumfang ziemlich spartanisch: ein Browser und Profiler sollten in einem professionellem Paket schon enthalten sein. Und dann ist da ja noch die Accessory-Version; sie stellt eine Art Verlegenheitslösung dar. Eine schöne GEM-Shell mit einer Make-Funktion würde dem Programm gut zu Gesicht stehen. Das Handbuch stellt zwar keinen Ersatz für [1] dar, läßt sich aber trotz orthographischer Fehler gut lesen und bildet für den Anfang eine solide Grundlage. Wenn der Hersteller sein Versprechen, die Programmierer mit Klassen zu unterstützen, hält, dann kann ich das Programm jedem empfehlen, der unbedingt die Vorzüge von OOP kennenlernen will. Der Weisheit letzter Schluß ist M:OOP jedoch nicht.

Grischa Ekart

M:OOP

Bezugsquelle:

Fries & Partner
Eislebener Str. 7
1000 Berlin 30
Tel.: 030/69 41 114

Literatur:

[1] B. J. Cox: *Object-Oriented Programming*, Addison-Wesley, 1986