VIERTE DIMENSION

3&4/1994

10. Jahrgang 1994 Doppelheft DM 20,-

Forth im Fenster

Echtzeit & Automatisierung

Objektorientierte Programmierung

und, und, und,...

FORTH MAGAZIN

Organ der Forth Gesellschaft e.V.

ultra- und volksFORTH (Diskettenformat)	**Handbuch**		**Disketten**		**Komplettpaket**	
ultraFORTH 3.8 für Commodore C64 (2*5¼"; beidseitig)	C64uF-H	40,00	C64uF-D	25,00	C64uF	55,00
volksFORTH 3.8 für Schneider CPC (2*3" für CP/M)	CPCvF-H	55,00	CPCvF-D	25,00	CPCvF-HD	70,00
volksFORTH 3.80 für Atari ST (3*360K; 3.5")	STvFH	65,00	STvF-D	35,00	STvF-HD	75,00
volksFORTH 3.81.41 für PC (1*360K; 5¼")	PCvF-H	65,00	PCvF-D	10,00	PCvF-HD	65,00
8087-Floatingpointpacket für PC-volksFORTH (nur Diskette: 1*360K; 5¼")					PCFloat	10,00
KK-FORTH (Alle Versionen mit PC-Terminalprogramm und Tools)	**Ergänzung**				**Komplettpaket**	
Allgemeines Handbuch zum KK-FORTH (für alle Versionen, ist im Komplettpaket enthalten)					KKFHB	65,00
KKF V1.2/1 für PC (Zusatzbeschreibung und 2*360K; 5¼")	KKFPC-E	70,00			KKFPC	110,00
KKF V1.2/0 für PC-Einplatinencomputer (V20; SIO0, 32K EPROM; 32K RAM)	KKFV20-E	70,00			KKFV20	110,00
KKF V1.2/0 für EMUF Z80mini3 (84C015; 32K EPROM; 32K RAM)	KKF8415E	70,00			KKF8415	110,00
KKF V1.2/0 für RTX-2000/1a (RTX-Board der FG-e.V. und RTX_EMUF)	KKFRTX-E	70,00			KKFRTX	110,00
Sonstiges						
Mikroprozessor Super8 mit ROM-FORTH (8KByte)					S8Chip	46,00
Handbuch zum Super8-FORTH mit Diskette (PC; 360K; 5¼")					S8HD	34,50
Leerplatine, Bausatzbeschreibung und Bauteile für S8-Bausatz (nur GAL, Quarz und SMD´s)					S8B	57,50
Komplettes Super8-System nach Bausatz aus VD					S8Sys	228,00
FIFTH 2.0 (PC-Diskette 1*360K; 5¼")					FIFTH	10,00
FPC V3.56 (PC-Diskette 4*1.2M; 5¼")					FPC356	25,00
Sourcen zur Vierten Dimension (PC-Diskette; bitte Ausgabe angeben: z.B. 03/94 -> xxxx=0394)					VD-xxxx	10,00
Sourcen zu EFORTH V1.0 für 8098, PC, 68HC11 oder 8051 (PC-Diskette; z.B 8096 -> xx=96)					EF-xx	10,00
DPANS6-Dokumentation					DPANS6	40,00

Weitere PD-FORTH-Versionen, Literatur oder andere Diskettenformate auf Anfrage
Alle Preise in DM inklusive 15% Mehrwertsteuer
Preisnachlaß für FG-Mitglieder: 10%
Verpackung, Versandt und Nachnahme: 9,00 DM

Postfach 1173
86406 Mering
Tel. 08233/30524
Fax 08233/9971

IMPRESSUM

NAME DER ZEITSCHRIFT
VIERTE DIMENSION
FORTH MAGAZIN
Organ der FORTH-Gesellschaft e.V.

Herausgeber
FORTH-Gesellschaft e.V.
Postfach 110
85701 Unterschleißheim
Tel.: 0 89 / 3 17 37 84

Redaktion
Birgit Steffenhagen (bs)
Tel.: 03 81 / 57 - 392
Ralf Neuthe (rn)
Tel.: 01 72 / 428 65 55

Illustrationen
Rolf Kretzschmar

Redaktionsschluß
Feb., Mai, Aug., Nov.

Erscheinungsweise
vierteljährlich

Auflage
1000

Preis
Einzelheft DM 10,-, Abonnementpreis DM 50,-, bei Auslandsadresse DM 55,- inklusive Versandkosten

Anmerkungen der Redaktion und des Direktoriums in Personalunion

von Birgit Steffenhagen

Nun steht das Weihnachtsfest vor der Tür und ein recht turbulentes Jahr 1994 geht zu Ende. Leider hat es nicht mehr ganz geklappt, das neue Heft auf den Gabentisch unserer treuen Mitglieder zu legen. Aber als Überraschung im neuen Jahr ist es ja auch ganz nett. Wir sind Euch auch nichts schuldig geblieben und haben ein Doppelheft "produziert" und hoffen Euch damit wieder versöhnen zu können nach den Turbulenzen und Terminsäumigkeiten der VD im letzten Jahr. Wir hoffen, daß mit dem Übergang der Redaktion in die Hände von Claus Vogt aus Berlin, die VD wieder in ruhige See kommt und Ihr pünktlich neues aus der Forth-Welt erfahrt. Hier noch mal an alle Praktiker, Wissenschaftler und Hobbyisten der Forth-Gemeinde den Aufruf, über alles zu berichten was Euch mit Forth widerfährt und an die Redaktion zu schicken.

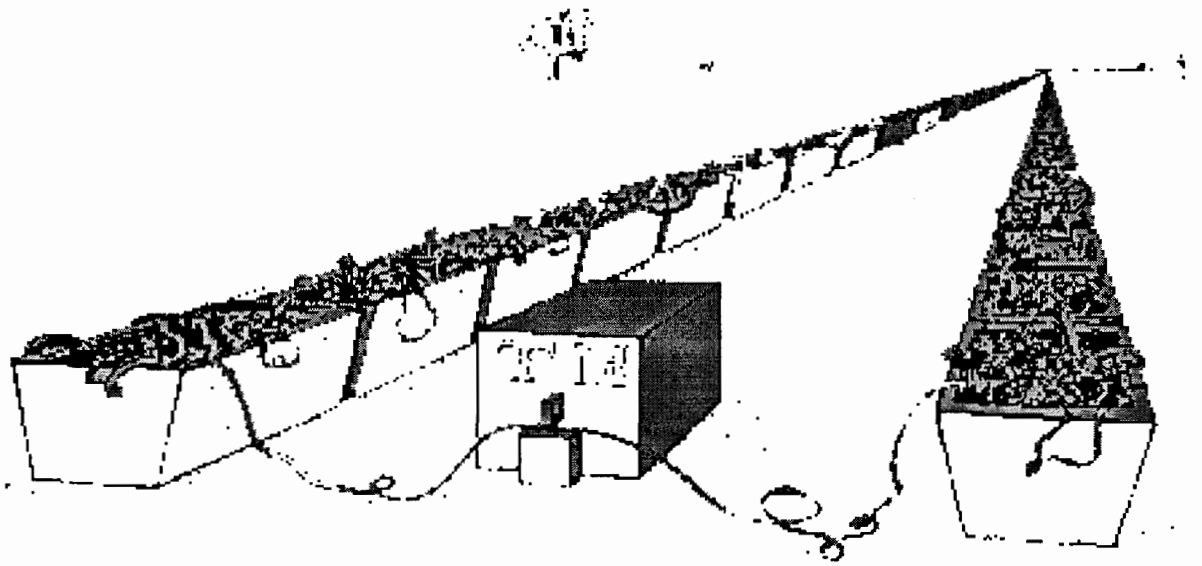
Ich möchte mich bei allen bedanken, die uns diesmal so fleißig Artikel zukommen ließen und auch den angesprochenen Firmen für ihre umfangreichen Produktvorstellungen. Wo wir gleich bei einem Thema unseres Heftes wären. Es geht um Forth-Systeme unter Windows. Ja auch wir Forthler kommen da nicht mehr umhin, uns mit dieser Problematik auseinanderzusetzen. Welche Möglichkeiten uns jetzt offenstehen, sollen die Artikel zu dem Thema "Forth im Fenster" aufzeigen.

Ein weiterer Schwerpunkt unseres Doppelheftes wird wiedereinmal die "Echtzeit und das Drumherum" sein. Hier geht es um Timer- und Automatenprogrammierung, eingebettete Systeme, Windows- und Echtzeitsysteme sowie um objektorientiertes Programmieren und paralleles "Speichermüllsammeln". Na viel Vergnügen. Auch sollen mal einige "philosophische" Betrachtungen und ein Interview mit Herrn Moore über seine Streiche, die uns heute noch beschäftigen, nicht fehlen.

Zum Schluß wollen wir natürlich allen Mitgliedern und Abonnenten unseres Heftes ein erfolgreiches Jahr 1995 wünschen .

Euer Direktorium

Eine Turing-Maschine braucht strenggenommen ein unendlich langes Band..



... so erklärt es uns zumindest Roger Penrose in seinem Buch 'Computerdenken'. Die 'Vierte Dimension' stellt da geringere Anforderungen. Ihr endlich langes Manuskript sollte allerdings in endlicher Zeit bei uns eintreffen. Die nächste Ausgabe wird erstmalig in Berlin erscheinen und soll pünktlich im März in Ihrem Briefkasten liegen. Ein kleines, hochqualifiziertes Team mit modernster Kommunikationstechnik steht für den Erfolg dieser zweifachen Weltpremiere bereit. Alles, was noch fehlt, sind die Artikel. Trotz modernster Technik können wir nicht mehr veröffentlichen, als Sie - liebe Forth-Freunde - uns einsenden.

Artikel, die im Januar eintreffen, werden gerne berücksichtigt. Kürzere Informationen, Leserbriefe und Kurzkritiken der Weihnachtspräsente haben bis in die erste Februarwoche Zeit. Warten Sie nicht ab, bis die Alarmsirenen ertönen, schreiben Sie jetzt! Dafür erlassen wir Ihnen strenge Autorenrichtlinien. Eine PC-Diskette wäre uns natürlich am liebsten. Kurze Meldungen passen auch auf den Anrufbeantworter. Faxen Sie Endlosstreifen à la Penrose, mailen Sie, wenn Sie können!

... und wenn gerade nichts anderes zur Hand ist, beherzigen Sie Joseph Weizenbaums Vorschlag für die Konstruktion einer Turing-Maschine. Sein Streifen wird bis zur Länge von fünf ungebrauchten Blatt anstandslos entgegengenommen, findet sich in jedem Haushalt und wird nach Veröffentlichung einer stofflichen Wiederverwertung zugeführt. Ein Beitrag unserer Redaktion zum Umweltschutz.

Echtes Forth.
Ein Geschenk für gute Freunde.
In Ihrem Forth-Magazin

Neue Adresse ab sofort

Forth Magazin - Vierte Dimension
c/o Claus Vogt
Ebersstraße 10
D-10827 Berlin
030 / 782 81 79 pa(flvb)
clv@BBS.forth-ev.de

Abkürzungen:

p	- privat
a	- Anrufbeantworter
f	- Fax
l	- Faxpollen
v	- Voicebox
b	- Mailbox
()	- Betrieb unregelmäßig, ggf. nach fünf Minuten noch mal probieren oder Zeitpunkt des nächsten Versuchs kurz ankündigen

Noch Fragen?

Ruf doch mal an!

Inhalt / Info

FORTH-Gesellschaft intern

Wunschzettel des FORTH-Büros für 1995

Bitte benachrichtigen Sie das FORTH-Büro rechtzeitig bei Änderungen Ihrer Anschrift oder Bankverbindung, da fehlgeleitete Postsendungen und Überweisungen unnötige Kosten verursachen. Der VIERTEN DIMENSION, Jahrgang 10, Nr. 3/4, liegt ein Mitgliedsantrag bei. Mit dem unteren Abschnitt des Antrags haben Sie die Möglichkeit, der FORTH-Gesellschaft eV eine Einzugsermächtigung für den Mitgliedsbeitrag zu erteilen, oder benützen Sie diesen Antrag zum Werben neuer Mitglieder.

Mitgliedsbeiträge:

Der Mitgliedsbeitrag gilt immer für das laufende Kalenderjahr. Beachten Sie bitte, daß die ermäßigten Beiträge nur gegen Nachweis gewährt werden können. Wenn Sie also für 1995 den ermäßigten Beitragssatz in Anspruch nehmen möchten, bitten wir Sie eine Kopie Ihrer Studienbescheinigungen, Ausbildungs-, Wehrpflicht- oder Ersatzdienstnachweise an das FORTH-Büro zu schicken. Wir benötigen nur einen Nachweis pro Kalenderjahr.

Ihren Mitgliedsbeitrag entrichten Sie bitte bis Februar 1995. Als kleine Erinnerung liegt dieser VIERTEN DIMENSION ein Überweisungsformular bei.

Wurde der FG eine Einzugsermächtigung erteilt, werden wir den neuen Mitgliedsbeitrag Ende Februar abbuchen.

Hier die gültigen Mitgliedsbeiträge für 1995:

- Schüler, Studenten, Rentner u. Arbeitslose 64,00 DM
- Ordentliche Mitglieder, Auslandsadresse 96,00 DM
- Fördernde Mitglieder, Firmen u. Institutionen 176,00 DM

Bei Auslandsadressen ist wegen der erhöhten Versandkosten keine Ermäßigung möglich.

Für weitere Fragen stehen wir Ihnen gerne zur Verfügung.

Telefonisch erreichen Sie uns unter 089-317 37 84 (Anrufbeantworter und Fax, wir rufen gerne zurück, aus Kostengründen lieber nach 18:00 Uhr).

Allen Mitgliedern und Lesern wünschen wir ein erfolgreiches, gesundes 1995.

Ulrike Schnitter
-FORTH-Büro-

LMI WINFORTH

Eine Vorstellung von FS FORTH-SYSTEME GmbH
Postfach 1103, 79200 Breisach

LMI WINFORTH bietet Ihnen eine komfortable Entwicklungsumgebung, um mit der Programmiersprache FORTH Standalone Windows Programme zu entwickeln. Es unterstützt dabei alle Windows-Funktionen und erleichtert mit Hilfe von integrierten Multitaskingfunktionen den Aufbau von typischen Windows Anwendungen. Dabei besteht Zugriff auf die gesamte Windows-API. Es können allerdings auch andere, beispielsweise C-Libraries, benutzt und integriert werden. Alle dokumentierten Funktionen lassen sich direkt als FORTH-Worte aufrufen, aber auch undokumentierte sind einfach zu integrieren. Die komplizierten Windows API-Calls sind in den für den FORTH-Entwickler geläufigen FORTH-Worten enthalten. Somit ist auch die Ein- und Ausgabe von Text sehr leicht realisierbar. Es genügt bereits eine Zeile FORTH-Code, um eine Messagebox am Bildschirm darzustellen.

Im Handbuch finden Sie mehrere einführende Kapitel über die Grundelemente von Windows und die Programmierung mit WINFORTH. Beginnend von der Erstellung eines Fensters finden Sie nach und nach alle Informationen, die Sie zur Erstellung einer Windows-Applikation benötigen.

Menüstrukturen und selbstdefinierte Dialogboxen können entweder durch FORTH-Worte zusammengestellt werden, oder Sie können hierzu auch den Microsoft Resource Compiler oder den Borland Resource Workshop verwenden. Durch Verwendung der "Common Dialogs"-Schnittstelle können Sie sich erheblichen Programmieraufwand einsparen. Eine Dateiauswahlbox oder ein Fenster zur Auswahl eines Druckers kann z. B. über ein einziges FORTH Wort aufgerufen werden.

Durch über 40 verschiedenen Beispielprogramme wird Ihnen der Einstieg in die Windows-Programmierung erleichert und Sie erhalten dadurch auch erweiterbare Grundgerüste für nahezu alle Themenbereiche.

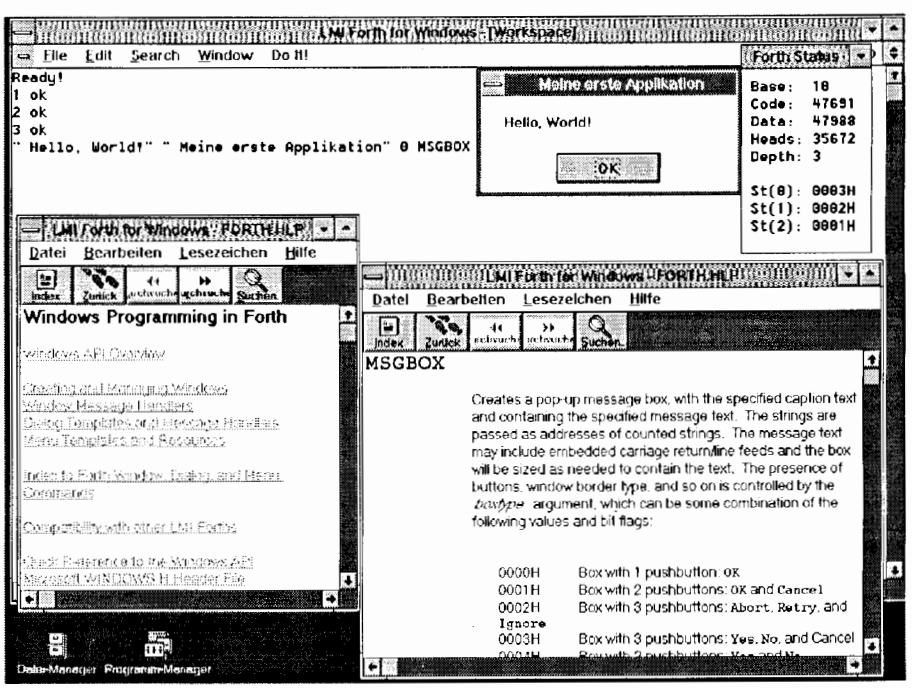
Die Benutzeroberfläche von LMI WINFORTH ist vollständig in die Windows-Umgebung integriert. Durch Verwendung eines MDI-Fensters (Multiple Document Interface) können Sie mehrere Quelltexte laden und zwischen diesen umschalten. Über das Windows-Clipboard können Sie auf einfache Weise Quelltexte kopieren oder auch Zeilen in den Kommandozeileninterpreter übernehmen. Alternativ hierzu steht Ihnen auch ein verbesserter Screen-Editor zur Verfügung. Quelltextdateien von UR/FORTH können dadurch ohne Änderung weiterverwendet werden.

Über eine Menüleiste können alle wichtigen Funktionen, wie z. B. Dateien laden und speichern, kompilieren, ausführen, drucken etc. komfortabel aufgerufen werden. Zu Debuggingzwecken können Sie sich den aktuellen Stack-Zustand in einem Fenster anzeigen lassen.

Die komplette Dokumentation steht Ihnen als Windows-Hilfe zur Verfügung, was die Arbeit erheblich erleichtert. Definitionen der einzelnen Wörter können durch die Suchfunktion schneller gefunden werden als durch Blättern im Handbuch. Durch Querverweise können Sie sich sehr einfach zwischen Überblick und den detaillierten Funktionen hin und her bewegen.

LMI WINFORTH bietet Ihnen die Möglichkeit, Ihre FORTH-Programme in kurzer Zeit auf Windows umzustellen. Durch das sowohl kompilierend wie interpretierend arbeitende Sprachkonzept von FORTH eignet sich WINFORTH auch für den Windows-Anfänger, um verschiedene Windows-Aufrufe experimentell auszuprobieren.

❑



Programmierung mit comFORTH für Windows

von Udo Schütz, FORTecH-Software GmbH
Joachim-Jungius-Straße 9, 18059 Rostock

In diesem Artikel soll das Programmiersystem comFORTH für Windows vorgestellt werden. Das System comFORTH wird schon seit Jahren als Haussystem der Firma FORTecH Software GmbH Rostock gepflegt und weiterentwickelt. Ziel der Umgebung unter Windows war die Beibehaltung der von den Vorgängersystemen bekannten Forth-Sprachoberfläche. Sie sollte sich jedoch die Vorteile der graphischen Betriebssystemerweiterung Windows verstärkt zunutze machen und so dem Programmierer eine komfortable Entwicklungsumgebung bieten. Nicht zuletzt sollte aber darauf geachtet werden, dem Nutzer den vollen Zugriff auf das System zu gewähren und ihm dazu alle sprachlichen Mittel zur Verfügung zu stellen.
Im Folgenden werden die Eigenschaften des Systems kurz umrissen und anhand einiger einfacher Programmierbeispiele der Umgang mit comFORTH für Windows erläutert. Dies mag ebenfalls als kleiner Einstieg in die Windows-Programmierung dienen und den Leser ermutigen, selbst weiter in dieser Richtung zu testen und zu probieren.

Das Entwicklungssystem

Überblick

comFORTH für Windows stellt ein segmentiertes 16-Bit Forthsystem dar. Es ist voll kompatibel zum Standard FORTH83. Als Codeart wird indirekt gefädelter Code unterstützt. Die Oberfläche ist Windowskonform gestaltet und gestattet eine komfortable Erstellung und Compilation der Quellen. Die Quellen werden mittels eigener Editoren sowohl in Text- als auch in Block-Files unterstützt. Zur Erleichterung der Arbeit verfügt es über Knopfleisten, die auch anwendungsspezifisch nachgeladen und wie die Fenstermenüs frei über eine INI-Datei konfiguriert werden können. Auch zahlreiche andere Systemeinstellungen, wie zum Beispiel Suchpfade, Stapelgrößen, Fonts und viele andere, können über die INI-Datei eingestellt werden. Zur besseren Kontrolle des Systems ist es mit einer Statuszeile versehen. Das Windows-API (Applikation Programmers Interface) wird in vollem Umfang mit allen Funktionen, Strukturen und Konstanten unterstützt. Zur Entlastung des Forth-Bereiches ist das API in einem eigenen Segment untergebracht.

Das System befindet sich momentan noch in der Betatestphase. Das Release, welches zum jetzigen Stand technisch keine wesentlichen Veränderungen mehr erfahren wird, ist demnächst verfügbar.

Werkzeuge

Im Lieferumfang ist das System in drei Ausbaustufen enthalten. So gibt es das Entwicklungssystem (CFWX.EXE) mit allen Komponenten für ein komfortables Editieren und Debuggen. Nach der Entwicklungsphase kann die Applikation dann auf das Basissystem geladen werden, das zur Speicherplatzeinsparung auf die Entwicklungskomponenten verzichtet, aber das API voll unterstützt. Sollen sehr kleine Applikationen erzeugt werden, kann man das System ohne residentes API verwenden und dort nur die benötigten Funktionen aus den mitgelieferten Headerdateien laden.

comFORTH für Windows stellt zur Applikationsentwicklung unter Windows eine Entwicklungsumgebung zur Verfügung, mit der man Quelltext erstellen, compilieren, debuggen und in Forth natürlich auch interpretieren kann. Eine Besonderheit von Windows-Programmen sind die in ihnen enthaltenen Ressourcen. Diese Ressourcen sind z. B. Ikonen, Dialog-Templates, Bitmaps und andere graphische Oberflächenelemente einer Applikation oder auch nutzerdefinierte Daten. Die Ressourcen werden mit Hilfe geeigneter Editoren erstellt und in einem gesonderten Compilationslauf an das entsprechende Programm-Modul gebunden. Das Modul ist dann in der Lage, diese Ressourcen zur Laufzeit zu laden und in der Applikation zu verwenden. Aus dieser Tatsache ergibt sich die Notwendigkeit eines Resource-Toolkits zu deren Erzeugung und Compilation. Da derartige Toolkits zur Zeit in verschiedenen Varianten am Markt separat oder als Bestandteil anderer Windows-Entwicklungswerkzeuge verfügbar sind, wurde bei comFORTH für Windows auf eine Nachentwicklung verzichtet. Es verfügt also über kein eigenes Resource-Toolkit, unterstützt jedoch die Arbeit mit den Werkzeugen des Windows-SDK (Software Development Kit), des Microsoft-Visual C++ oder des Borland-Resource-Workshops.

Oberfläche

Zentrales Element der Forth-Oberfläche ist der Workspace. Er ersetzt die traditionelle Eingabezeile. Über ihn werden die Kommandos eingegeben und gelangen zur Interpretation. Auch Ausgaben erfolgen über den Workspace. Man kann mehrere Workspaces gleichzeitig öffnen. Dies ist zum Beispiel sinnvoll, wenn man in einem Fenster einen Dump erzeugt hat und in einem anderen weiterarbeitet, ohne den Dump aus den Augen zu verlieren. Im Workspace hat man über die Scrollbars auch eine History-Funktion. Jeder Teil des abgearbeiteten Workspaces kann wiederholt ausgeführt oder zum nochmaligen Editieren an die aktuelle Eingabeposition geholt und dort ausgeführt werden. Natürlich kann man den so getesteten Quelltext dann per Cut&Paste in ein Quellfile übernehmen und sichern. Aber auch der umgekehrte Weg ist möglich: Man schreibt seine Ent-

würfe in ein Quellfile und compiliert selektiv aus diesem Fenster.

Bei den Quellfiles werden, wie bereits erwähnt, sowohl Block- als auch Textfiles durch jeweils eigene Windows-Editoren unterstützt. Der Block-Editor verfügt dabei noch über eine automatische Darstellung der Kommentare in einer separaten Farbe. Beide Editoren gestatten zur besseren Verwaltung der Quellen das Setzen eines automatischen Datumsstempels. Neue Textfiles können auf Basis einer nutzerdefinierten Vorlage erstellt werden. Eine Verriegelung der Dateien im Netzbetrieb erfolgt durch read-only-Öffnen bereits benutzter oder schreibgeschützter Dateien.

Zur Vereinfachung der Verwaltung bereits erstellter Quellen werden über die INI-Datei einstellbare Suchpfade unterstützt. Ihre Zahl ist nicht begrenzt. Ein weiterer Mechanismus in dieser Richtung wird über den Ladeautomatismus EXTERNAL verfügbar gemacht. In der speziellen INI-Datei EXTERNAL.INI können für einzelne Worte Ladeanweisungen definiert werden, die bei erfolgloser Wörterbuchsuche ausgeführt werden. So wird zum Beispiel ein fehlendes Wort nach der Eingabe geladen und danach sofort ausgeführt.

Die Windows-Programmierung wird durch zahlreiche Werkzeuge unterstützt. Einige von ihnen werden in den folgenden Abschnitten an konkreten Beispielen näher erläutert. Stellvertretend seien die Unterstützung der Bibliothek DDEML zur DDE-Programmierung oder des kompletten Multimedia-APIs, die Bereitstellung von sogenannten WHEN-Listen zur komfortablen Callback-Programmierung oder vereinfachender Worte zum Zugriff auf INI-Dateien oder die Common-Dialoge genannt.

Windows-Programmierung

In diesem Abschnitt sollen anhand einiger einfacher Beispiele einerseits der Umgang mit comFORTH für Windows andererseits aber auch erste Schritte in der Windows-Programmierung gezeigt werden. Etwas Theorie, die zum Verständnis dienlich sein soll, findet der Windows-Neuling im Kasten. Einige Themen zur weiterführenden Programmierung, wie z. B. DDE oder DLLs,

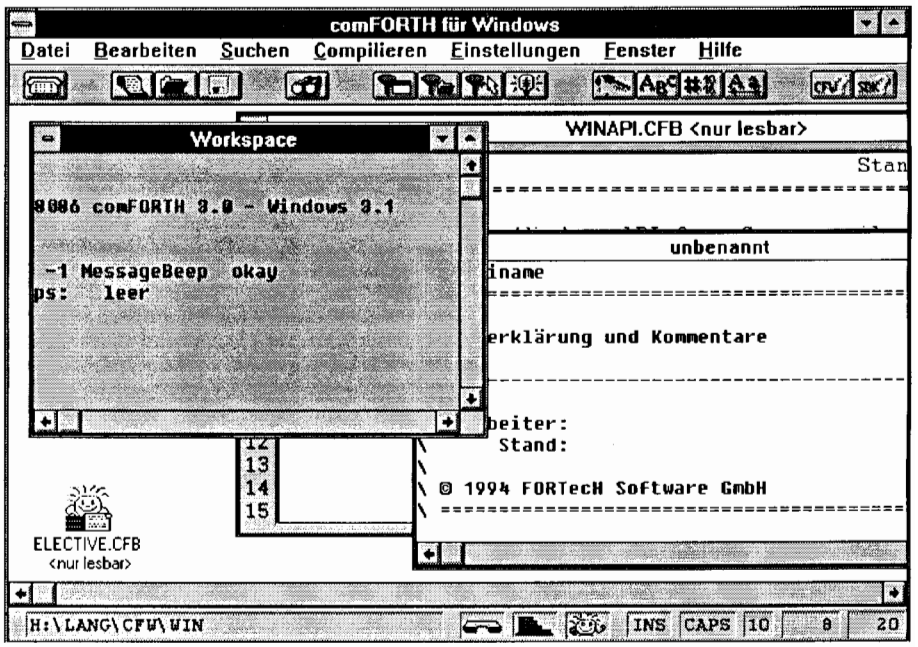


Bild 1 Oberfläche comFORTH für Windows

werden Thema weiterführender Artikel sein.

Folgende Vereinbarungen sollen für die folgenden Beispiele gelten:
Nutzereingaben: fett gedruckt
Rechnerausgaben: unterstrichen

Abkürzungen:
ps: Parameterstack
"s: Stringstack
ib: Eingabepuffer (Input-Buffer)
===> Stackbilanz (vor und nach Ausführung)

Eine MessageBox

Wie schon erwähnt, wird unter comFORTH für Windows das gesamte Windows-API unterstützt, d. h. alle Funktionen, die in der SDK-Hilfe beschrieben sind, können auch unter comFORTH verwendet werden. In der SDK-Hilfe sind diese Funktionen in C-Notation aufgeführt. Beim Aufruf in Forth erfolgt der Parameteraustausch natürlich über den Stack, wobei aber die Anzahl und die Reihenfolge der Parameter beibehalten wird. Dies kann man sich an einem einfachen Beispiel mit der Funktion MessageBeep verdeutlichen. Diese Funktion gibt einmal kurz Laut und wird folgendermaßen aufgerufen:

C-Notation (SDK-Hilfe):
```
void MessageBeep (uAlert)
```

Forth-Aufruf:
```
MessageBeep
( ps: uAlert ===> )
```

uAlert ist dabei ein Parameter, der (bei Installation eines entsprechenden Treibers) den Klang bestimmt. Mit -1 gibt es einen kurzen Pieps:

```
-1 MessageBeep ↵ okay
```

Das war unser erster API-Aufruf unter Forth. Diese Funktion hat einen Parameter verbraucht und keinen zurückgeliefert. Als nächstes soll eine eigene Messagebox erzeugt werden, wie sie von anderen Programmen hinlänglich bekannt sein dürfte. Die erforderliche Funktion heißt MessageBox. Man kann (bei korrekt installiertem comFORTH) diesen Befehl schon einmal eintippen (vielleicht in ein neues Quellfile), den Cursor daraufsetzen und mit F1 die kontextbezogene On-Line-Hilfe für die Funktion aktivieren. In Forth ausgedrückt, ergibt sich folgender Aufruf:

```
MessageBox
( ps: hWndParent lpszText
      lpszTitle fuStyle
      ===> int )
```

Dabei kann für den Test das Fensterhandle des Parent mit 0 angegeben werden (kein Elternfenster). Die beiden Long-Pointer auf den Text und den Titel

müssen erst erzeugt werden. Dazu dient das Wort:

```
LPCSTR ( ps: ===> lpsz )
       ( "s: cs ===> cs+0)
```

Der Style legt das Erscheinungsbild der Messagebox bezüglich der Knöpfe und der Ikone fest. Zum Test soll ein einfacher OK-Knopf und ein Stopzeichen verwendet werden. Nun wird die Box erzeugt:

```
0 ↵ okay     \ hWndParent
" Das ist der Text" LPCSTR
↵ okay       \ lpszText
" Nachricht" LPCSTR ↵ okay
             \ lpszTitle
MB_OK MB_ICONSTOP OR ↵
okay
             \ fuStyle
MessageBox ↵ okay
```

Es sollte eine Messagebox erscheinen. Nach Betätigen des OK-Kopfes erhält man als Rückgabewert eine 1, die anzeigt, daß der OK-Knopf gedrückt wurde (1 entspricht IDOK). IDOK ist eine der API-Konstanten, die unter comFORTH symbolisch zur Verfügung stehen. Sie repräsentieren jeweils einen Zahlenwert und sind im Original in der WINDOWS.H definiert. Den erhaltenen Wert und die beiden nicht mehr benötigten Strings wirft man weg mit

```
DROP "DROP "DROP ↵ okay
```

Jetzt kann man ein einfaches Wort definieren, das eine vordefinierte Messagebox mit einem variablen Text zeigt und dabei mit einem Laut auf sich aufmerksam macht. Eine solche Box ist beispielsweise für die Ausgabe von Fehlermeldungen sinnvoll und soll deshalb ".FEHLER heißen:

```
: ".FEHLER  ( ps: ===> )
            ( "s: text ===> )
  -1 MessageBeep          \ Laut geben
  0                       \ hWndParent
  LPCSTR                  \ lpszText
  ["] Fehler" LPCSTR      \ lpszTitle
  MB_OK MB_ICONSTOP OR    \ fuStyles
  MessageBox              \ Box zeigen
  DROP "DROP "DROP        \ aufräumen
;
```

Bild 2 erste Messagebox

```
" Das war ein Fehler"
.FEHLER" ↵ okay
```

Diesem Wort übergibt man einen Text auf dem Stringstack, welcher dann in einer Box mit Ton angezeigt wird. Diese Box war das erste Fenster, das wir unter Windows erzeugt haben. Dabei wurden alle Routinen zur Verwaltung des Fensters, denn man kann es ja z. B. auch ohne unser besonderes Zutun verschieben, vom System übernommen. Wie man Fenstern ein eigenes Verhalten zuordnet, werden wir in den folgenden Abschnitten sehen.

API-Strukturen und das Record-Paket

Viele der komplexeren API-Funktionen erfordern die Verwendung der ebenfalls im API definierten Strukturen. Man kann diese ebenso wie die Funktionen in der SDK-Hilfe nachschlagen. Das Recordpaket von comFORTH stellt eine wesentliche Erleichterung des Umganges mit strukturierten Daten, insbesondere der API-Strukturen dar. Es wurde in /WOI93/ vorgestellt und soll hier nur noch einmal kurz in den Bestandteilen erläutert werden, die zum Verständnis der Beispiele notwendig sind. Es werden eine Reihe von Definitionsworten bereitgestellt, mit denen man nahezu jede Art von Strukturen erzeugen kann. Dazu wird zunächst immer ein Datentyp definiert. Basierend auf dieser Typdefinition kann man dann Speicherobjekte erzeugen.
Zur Erzeugung atomarer Datentypen dienen die Definitionsworte:

```
n ATOM: name
```

Beispiel:

```
2 ATOM: UINT ↵ okay
```

Es wird ein Datentyp mit dem Namen UINT erzeugt, der 2 Byte im Speicher beschreibt und seinen Typ bei Ausführung aktiviert.

Aufbauend auf derartige Typen können nun mit:

```
RECORD: name
            typ FIELD: name
(oder)      typ FIELD
      .
      .
;RECORD
```

Beispiel:

```
RECORD: RECT
      UINT  FIELD: left
            FIELD: top
            FIELD: right
            FIELD: bottom
;RECORD
```

auch strukturierte Datentypen definiert werden. In diesem Beispiel wird ein Typ mit dem Namen RECT definiert, der vier Integer-Variablen mit den Namen left, top, right und bottom enthält. Die Namen der Mitglieder sind dabei strukturlokal, d. h. in diesem Fall, daß left, top... auch noch in anderen Typdefinitionen verwendet werden könnten. Diese Typdefinition ist eine Definition des API und daher bereits im System enthalten. Sie kann in der SDK-Hilfe nachgesehen werden.

Die Typdefinitionen kann man beliebig schachteln und mischen. So können die Mitglieder eines Records wiederum Records oder auch Arrays sein.

Nach der Typdefinition sollen nun die Worte zur Erzeugung von Speicherobjekten auf Basis dieser Typen genannt werden:

```
typ OBJECT: name
```

Beispiel:

```
RECT OBJECT: FENSTER ↵
okay
```

In dem Beispiel wird ein Speicherobjekt mit dem Namen FENSTER erzeugt, d. h. es wird physikalisch Speicher für

seine Variablen reserviert. (Dies ist bei der reinen Typdefinitionnoch nicht passiert!) Jetzt kann man auf die Struktur FENSTER folgendermaßen zugreifen:

```
FENSTER top @ . ↵ okay
```

oder

```
100 FENSTER right ! ↵ okay
```

Man sieht, daß jeweils durch Ausführen eines Mitgliedes der Stuktur der zugehörige Typ aktiviert wird. In diesem Beispiel aktiviert FENSTER seinen Typ. Nun stehen die Mitglieder top, right, ... als Worte zur Verfügung. Sie übernehmen die Offsetberechnung und liefern damit die eigentliche Adresse.

Auf diese Weise kann man symbolisch in gut lesbarer Form auf jede Art von Datenstruktur zugreifen bzw. eigene Strukturen erzeugen.

Ein praktisches Beispiel soll die Verwendung mit einer API-Funktion verdeutlichen. GetWindowRect ist eine Funktion des Windows-API (SDK-Hilfe), mit deren Hilfe man die Größe und Position eines Fensters aus einem Fensterhandle ermitteln kann. Sie benötigt als Parameter eine Variable des Typs RECT, in der die Koordinaten abgelegt werden. Unter Verwendung der oben definierten Variable FENSTER und des Handles des comFORTH-Hauptfensters, erhältlich über hWndFrame, beschaffen wir Position und Ausdehnung des Forth-Fensters:

```
hWndFrame DSEG FENSTER ↵
okay   \ hwnd lprc
GetWindowRect ↵ okay
       \ Größe holen
```

DSEG liefert hier die Adresse des Forth-Datensegmentes, die zur Erzeugung des Long-Pointers benötigt wird. Die Struktur FENSTER kann jetzt, wie oben beschrieben, ausgelesen werden. In ein Wort verpackt, sieht das folgendermaßen aus:

```
: @RECT ( ps: hWnd ===> x0 y0 xd yd )
  ( Fenstergröße/position bestimmen )
     DSEG FENSTER GetWindowRect
     FENSTER left @    FENSTER top @
     FENSTER right @   FENSTER bottom @
;
```

```
hWndFrame @RECT . . . . ↵
407 513 10 25 okay
```

Lokale Variablen

Wie zu erkennen ist, wird die Variable FENSTER in @RECT nur temporär benötigt. Sie belegt nur unnötig Speicher im System. Da ähnliche Fälle oft auftreten, ist die Verwendung lokaler Variablen sinnvoll, die jetzt kurz eingeführt werden sollen. Sie gehören nicht zum Standard, sind aber für die Übersichtlichkeit und Handhabbarkeit von Windows- (und anderen) Programmen sehr nützlich. Die Lebensdauer lokaler Variablen ist auf die Aufrufebene eines Wortes beschränkt. Danach sind sie nicht mehr verfügbar. Realisiert wird das durch Aufbau eines temporären Frames auf dem Returnstack. In /WOI94/ wird eine genauere Beschreibung gegeben. Es existieren in comFORTH verschiedene Typen lokaler Variablen, deren Definitionsworte folgende sind:

- lokale Variablen mit Entsprechungen zum Standard:

```
n VAL: name
d 2VAL: name
VAR: name
2VAR: name
```

- lokale Variablen mit Typbindung:

```
typ OBJ: name
addr typ PTR: name
sel:off typ LPTR: name
```

- generische Operatoren für lokale Variablen:

```
TO name       (ps: n ===>)
2TO name      (ps: d ===>)
[PTR] name    (ps: ===> addr)
[LPTR] name
              (ps:===>sel:off)
```

Mit VAL: bzw. 2VAL: werden lokale Values erzeugt, die bei Ausführung ihren Wert auf dem Stapel hinterlassen und denen man mit der Operation TO einen neuen Wert zuweisen kann. Mit VAR: bzw. 2VAR: kann man sich entsprechend lokale Variablen erzeugen, die mit @ und ! bearbeitet werden können. Im Beispiel:

```
: AB/A+B ( ps: a b ===>
                    ab/a+b )
     VAL: b  VAL: a
     a b *  a b +  / ;
```

kann man gut die Benutzung erkennen. Eine Besonderheit stellen die typgebundenen Variablen dar. Mit ihrer Hilfe kann man das Wort @RECT redefinieren, so daß keine residente Strukturvariable FENSTER mehr benötigt wird. Dazu aktiviert man zunächst den Redefinierer mit

```
+REDEFINE ↵ okay
```

Sollte das Paket noch nicht geladen sein, wird es über die EXTERNAL geladen und aktiviert. Ab jetzt werden vorhandene Definitionen bei Redefinition gepatcht (und zwar so, daß sie auch in schon definierten Worten wirksam werden). Wir (re)definieren also:

```
: @RECT ( ps: hWnd ===>
              x0 y0 xd yd )
( Fenstergröße/position
                 bestimmen )
     RECT OBJ: Fenster
     [LPTR] Fenster
         GetWindowRect
     Fenster left @
     Fenster top @
     Fenster right @
     Fenster bottom @ ;
```

Das neue @RECT arbeitet mit lokalen Variablen und benötigt FENSTER nicht mehr. Weitere Beispiele zu den lokalen Variablen findet man in den Samples zu comFORTH für Windows.

Die erste Dialogbox

Da nun bekannt ist, wie man eine API-Funktion aufruft und wie man auf die Windows-Strukturen zugreift, kann es jetzt an den Aufruf einer eigenen Dialogbox gehen. Dialogboxen werden mit den schon erwähnten Resource-Editoren erstellt und mit Hilfe des Resource-Compilers an das ausführbare Modul compiliert. In den Samples findet man Beispiele für den Aufruf des Compilers in Form von Batch-Dateien. Wir werden für den Aufruf einer Dialogbox eine schon vor-

handene Box des comFORTH-Systems verwenden - den Info-Dialog. Natürlich ist auch der Aufruf selbsterstellter Dialoge möglich.

Eine Dialogbox ist eine spezielle Art von Fenstern. Da diesem Fenster ein eigenes Verhalten zugewiesen werden soll, d. h. daß es z. B. auf Eingaben reagieren soll, muß sie mit einem Callback (s. Kasten) beim System bekannt gemacht werden. Allgemeine Callbacks kann man mit Hilfe der Worte CALLBACK0, CALLBACK1 bzw. CALLBACK2 definieren. Ihre Verwendung kann man in der On-Line-Hilfe nachlesen. Um die Programmierung spezieller häufig benutzter Callbacks, wie in diesem Beispiel einer Dialogbox-Fensterfunktion, zu erleichtern, wurde comFORTH für Windows mit den sogenannten WHEN-Listen ausgestattet. Ihre genaue Funktionsweise kann man in der On-Line-Hilfe unter der Rubrik Tools erfahren. Die WHEN-Listen für Dialoge erhält man durch Laden der Datei WHENDLG.CFW:

```
@:WHENDLG WINCLUDE ↵
...
okay
```

(Man kann auch über die Fileauswahl laden). Jetzt steht das Definitionswort DLGPROC: zur Definition des Callbacks zur Verfügung. Wir definieren also:

```
DLGPROC: DEMODP ↵ okay
```

Mit der API-Funktion CreateDialog können wir nun den Dialog aufrufen. CreateDialog erzeugt einen sogenannten modeless Dialog, d. h. man kann im rufenden Fenster bei sichtbarem Dialog weiterarbeiten (im Gegensatz zum modalen Dialog, der mit DialogBox erzeugt wird). Zum Laden des Info-Dialoges muß man noch seinen Identifier kennen, den man aus dem Resource-Script entnehmen kann:

```
hInstance ↵ okay
      \ Instance
0 300 ↵ okay
      \ Template-Res.

hWndFrame ↵ okay
      \ hwndParent
DEMODP lpDlgProc ↵ okay
      \ lpdlgproc
CreateDialog ↵ okay
      \ Dialogaufruf
```

Jetzt ist der Dialog erzeugt. Auf dem Stapel haben wir sein Handle erhalten. Es ist aber nichts zu sehen. Das liegt daran, daß der Dialog die Eigenschaft besitzt, initial nicht sichtbar zu sein. Man kann dies in den Resource-Editoren einstellen. Um ihn anzuzeigen, folgt ein weiterer API-Ruf:

```
SW_SHOWNORMAL ShowWindow
DROP ↵ okay
```

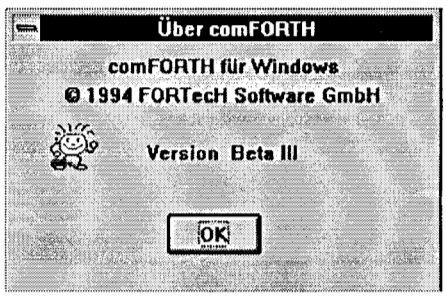


Bild 3 Info-Dialogbox

Jetzt sollte der Dialog erscheinen. Momentan reagiert er jedoch auf keinerlei Aktionen. Das soll nun geändert werden, indem der definierte Callback DEMODP schrittweise mit Leben erfüllt wird. Vorher ist es sinnvoll, mit

```
DEMODP +Debug ↵ okay
```

den Debug-Modus zu aktivieren. Da ein Stackfehler im Message-Handler i. a. zu üblen Abstürzen führt, wird so die Stackbilanz überwacht und in begrenztem Umfang korrigiert, indem die betroffene Aktion aus der Liste gestrichen wird. Man kann sie dann überarbeiten und neu übersetzen. Zunächst soll nun das Schließen des Dialoges möglich werden. Zu diesem Zweck reagieren wir auf die Nachricht WM_CLOSE, die nach Anwahl des Menüpunktes Schließen gesendet wird:

```
WM_CLOSE DEMODP ONMSG :WHEN
      hWnd DestroyWindow
      DROP
      ;WHEN ↵ okay
```

Das liest sich etwa so: "WHEN an ein Fenster mit dem Messagehandler DEMODP die Message (ONMSG) gesendet wird, führe hWnd DestroyWindow DROP aus". Jetzt kann man den Dialog über das Systemmenü schließen. Das Wort hWnd stellt eine der lokalen Values dar, die bei Benutzung der WHEN-Listen symbolisch die Parameter des Messagehandlers zur Verfügung stellen. (Es gibt außerdem noch Msg wParam und lParam - s. CFW-Hilfe).

Der OK-Knopf funktioniert aber nach erneutem Dialogaufruf (man kann sich dafür auch ein Wort schreiben) immer noch nicht, da wir ihm keine Aktion zugewiesen haben. Da er das Schließen des Dialoges bewirken soll, ist das Senden einer WM_CLOSE-Nachricht möglich, deren Verhalten wir ja gerade definiert haben.

```
IDOK DEMODP ONCMD :WHEN
      hWnd WM_CLOSE 0 0,
      SendMessage 2DROP
      ;WHEN ↵ okay
```

Das liest sich etwa so: "WHEN an ein Fenster mit dem Messagehandler DEMODP die Commando-Message (ONCMD) mit dem Parameter IDOK gesendet wird, führe ... aus". Die Message WM_COMMAND kann verschiedene Parameter besitzen, die in den Messagehandler verzweigen. In diesem Fall wird sie mit dem Identifier IDOK des OK-Knopfes bei dessen Betätigung gesendet. Mit dem Ruf SendMessage kann man eine Nachricht verschicken. Das Senden der Nachrichten kann man übrigens mit den Werkzeugen des SDK (Message-Spy) verfolgen. Nach dieser Definition sollte auch der OK-Knopf seinen Dienst versehen.

Als kleines Zusatzbonbon kann man den Dialog nun mit:

```
WM_RBUTTONDOWN DEMODP ONMSG
:WHEN
      -1 MessageBeep
      ;WHEN ↵ okay
```

einen Druck mit der rechten Maustaste (WM_RBUTTONDOWN wird geschickt) einmal piepsen lassen.

Ausblick

Mit diesem Artikel sollten erste Schritte in der Programmierung von Windows vermittelt werden, die hoffentlich zum weiteren Probieren anregen. Auch einige typische Vorgehensweisen, wie zum Beispiel die intensive Arbeit mit der SDK-Hilfe während der Programmierung oder die Definition neuer Eigenschaften der Ziel-Applikationen während ihres Laufes sollten verdeutlicht werden. Dabei wurde versucht, einige der Besonderheiten des Systems comFORTH für Windows sowie den praktischen Umgang mit der Umgebung zu zeigen.

Bei der Forth-Programmierung unter Windows wird besonders deutlich, daß die Interktivität von Forth durch die Multitaskumgebung Windows eine Renaissance erlebt. Hier hat man die Möglichkeit, am laufenden Objekt der Programmierung Veränderungen vorzunehmen und sie zu debuggen. Abgesehen davon ist der interaktive Aufruf einer API-Funktion an der Kommandozeile nicht in jeder Programmierumgebung möglich.

Da mit diesem Artikel keine umfassende Einführung in die Windows-Programmierung gegeben werden konnte, ist es vorgesehen, in den folgenden Heften Beiträge zu anderen weiterführenden Themen, wie zum Beispiel die Programmierung von eigenen Fenstern (auch als Turnkey-Anwendung), des Graphikinterfaces oder auch der DLL und DDE-Programmierung zu liefern.

Literatur

MS92:
Dokumentation zum Microsoft Software Development Kit for Windows; Band "Guide to Programming". Copyright Microsoft Corporation 1987-1992

PETZ92:
Petzold, Charles: "Programmierung unter Microsoft Windows 3.1." München: Microsoft Press Deutschland 1992

WOI93:
Woitzel, Egmont: "Datenstrukturen unter Forth - Eine alte Lösung für ein altes Problem" Vortrag auf der deutschen FORTH-Tagung in Nürnberg 1993

WOI94:
Woitzel, Egmont: "Typisierte lokale Variablen" Vortrag auf der deutschen FORTH-Tagung in Malente 1994

Etwas Windows-Theorie

Microsoft Windows stellt eine Multitaskumgebung dar, die unter einer grafischen Benutzeroberfläche arbeitet. Die Oberfläche ist fensterorientiert. Die einzelnen Fenster stellen Objekte dar, an die eine Applikation gebunden sein kann. Durch die Darstellung eines Applikationsfensters am Bildschirm wird sie dem Anwender zugänglich gemacht. Über das Fenster werden alle Interaktionen mit dem Benutzer abgewickelt. Dabei kann es sich zum Beispiel um Tastatur- oder Maseingaben oder entsprechende Bildschirmausgaben handeln. Das kooperative, ereignisgesteuerte Multitasking ermöglicht die quasiparallele Bearbeitung mehrerer Programme. So kann der Nutzer mit mehreren Applikationen gleichzeitig arbeiten. Die Verwaltung der Fenster und die Kommunikation der Anwendungen untereinander wird durch Mechanismen ermöglicht, die im Folgenden kurz umrissen werden. Dabei werden nur einige Punkte betrachtet, die zum grundlegenden Verständnis eines Einstieges in die Windows-Programmierung von Bedeutung sind. Eine weiterführende Behandlung der Thematik ist beispielsweise in /PETZ92/ gegeben.

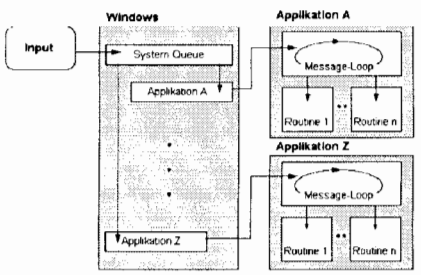
Callback

Unter Callbacks sind Funktionen eines Anwenderprogrammes zu verstehen, die nicht von ihm selbst, sondern vom Betriebssystem aus aufgerufen werden. Dies stellt die Grundlage der Ereignissteuerung unter Windows dar. Jede Applikation unter Windows ist mit wenigstens einer Callbackfunktion beim System angemeldet. Tritt ein für diese Anwendung relevantes Ereignis auf, so ruft das System diesen angemeldeten Callback mit den dem Ereignis entsprechenden Parametern auf. Die Auslegung von Callbackfunktionen ist vom jeweiligen Anwendungsfall abhängig. In ihnen kann der Programmierer Ereignisse des Systems behandeln. Ist dies nicht beabsichtigt, kann für jedes Systemereignis eine vordefinierte Standardbehandlung aufgerufen werden.

Ereignissteuerung

Die Ereignissteuerung bzw. -synchronisation wird durch den Austausch von Messages realisiert. Sie basiert auf dem Callbackmechanismus. Die zentralen Callbackfunktionen sind die in jedem Fenster zu realisierenden Messagehandler. Windows als System verwaltet eine globale Message-Queue, in welche zunächst alle einlaufenden Nachrichten plaziert werden, wie es im Bild zu sehen ist. Quelle dieser Nachrichten sind meist Aktionen des Nutzers mittels Tastatur oder Maus. Aus diesen Messages können dann vom System weitere abgeleitet werden. Aber auch Funktionen, wie zum Beispiel Timer, können Messages liefern. Aus der Message-Queue verteilt das System die Nachrichten entsprechend ihrer Zugehörigkeit zu den Fenstern auf die einzelnen Applikationen. In diesen existiert eine ähnliche, aber applikationsspezifische Message-Queue, in die die Nachrichten plaziert werden. Nach dem Start einer Applikation wird in eine Schleife, Message-Loop genannt, gesprungen, die erst beim Ende des Programms wieder verlassen wird. In ihr erfolgt die Überprüfung der Queue, wobei - dem Prinzip des kooperativen Multitasking gemäß - eine zyklische Übergabe der Steuerung an das System erfolgt. Dieses ruft für den Fall, daß Nachrichten detektiert wurden, den Messagehandler der Applikation auf. Andernfalls erfolgt der Aufruf der Message-Loop einer anderen Anwendung.

Der Messgehandler übernimmt, den Aufgaben der Applikation entsprechend, eine Bearbeitung der Nachricht. Hier ist vom Programmierer der Anwendung die gewünschte Funktionalität bereitzustellen. Soll die betreffende Message nicht behandelt werden, erfolgt der Aufruf des Default-Messagehandlers von Windows, der sie korrekt bearbeitet.



Durch dieses Messageprinzip kann man eine sehr gute Modularisierung von Applikationen erreichen, wobei der Daten- und Informationsaustausch zwischen den Modulen über den Austausch von Nachrichten abgewickelt wird.

Eine Besonderheit in der Programmierung stellt im kooperativen Multitasking die zyklische Übergabe der Steuerung an das System dar. Dies wird im allgemeinen in der Message-Loop gehandelt. Es kann vorkommen, daß man Routinen zu implementieren hat, die über einen großen Zeitraum die Steuerung für sich beanspruchen. Als Beispiel sei hier das Polling einer bestimmten Hardware in Treibermodulen genannt. Für diesen Ausnahmefall stellt Windows Funktionen zur Verfügung, die eine einmalige Abgabe der Steuerung an das System veranlassen und danach in das rufende Modul zurückkehren. So ist auch für diese Fälle eine Kooperation mit anderen Anwendungen möglich.

MPE ProForth für Windows

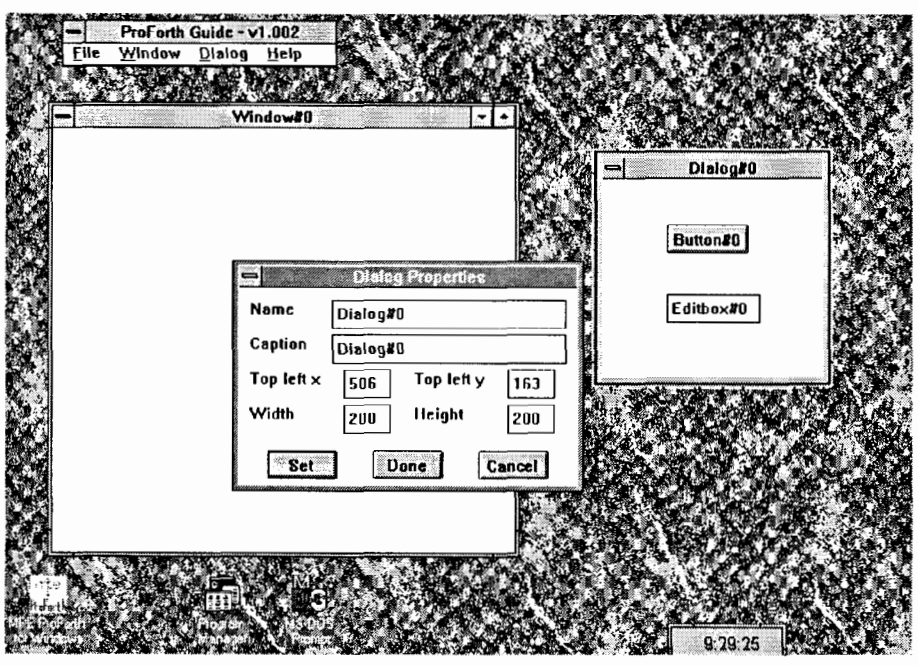
"More real - less time"

Roy Goddard BSc MIAP, Senior Engineer,
MicroProcessor Engineering Ltd.
übersetzt von Jörg Plewe

ProForth für Windows von MicroProcessor Engineering Ltd. ist ein 32-bit Forth-System für Windows 3.1 und Windows NT. Es bietet eine Entwicklungsumgebung, um große, benutzerorientierte Echtzeit-Applikationen zu erstellen, wobei die Forth-typischen, kurzen turn-around und time-to-market Zeiten erhalten bleiben. Dieser Artikel beschreibt einige der Hauptfeatures des Systems und seiner Tools.

ProForth GUIDE

Der Quelltext einer ProForth für Windows Applikation ist leicht verständlich und schnell geschrieben. Um dabei den Entwurf von Fenstern und Dialogen zu vereinfachen, gibt es ProForth GUIDE. Dabei handelt es sich um einen GUI-Editor mit dem Fenster und Dialoge, Menüs und Controls gestaltet werden können. Das ganze wird dann als Forth-Quelltext abgespeichert, der alle nötigen Definitionen sowie einen vollständigen, ausführbaren Programmrahmen enthält. In diesen Rahmen muß dann der Code für die entsprechenden Nachrichten und Bedienelemente eingefügt werden. Dieser so modifizierte Quelltext kann wieder von GUIDE zurückgelesen werden, so daß damit grafische Veränderungen leicht durchgeführt werden können. Wenn GUIDE das neue Layout speichert, bleibt der von Hand hinzugefügte Code erhalten. Mit GUIDE, dem Texteditor und dem Debugfenster können so Applikationen interaktiv in kürzester Zeit erstellt werden.



ProForth Features

Erweiterte Windows-Unterstützung

ProForth für Windows unterstützt alle Aspekte des Windows-GUI - Fenster, Menüs, Dialoge und 'Common Dialogs'. Damit erzeugt man Benutzerschnittstellen, die verständlich und einfach handhabbar sind. Fenster und Dialoge können leicht in Form von Sourcecode erstellt werden. Dazu stehen High-Level-Konstruktionen in einer erweiterten Forth-83-Umgebung zur Verfügung. Dieser Teil einer Applikation ist natürlich bei Verwendung von GUIDE noch einfacher zu erstellen. Wenn eine Applikationen auch grafische Darstellungen verlangt, kann man auf die Grafikunterstützung von ProForth für Windows zurückgreifen. Diese ist kompatibel zum Grafikpaket von MPEs Modular Forth und ProForth für DOS.

ProForth für Windows hat High-Level-Strukturen, um 32-bit DLL interaktiv ausführen zu können. Damit können DLL von anderen Autoren/Herstellern als auch mit anderen Sprachen erstellte DLL genutzt werden. Es eröffnet sich hier ein einfacher Weg, Code von Drittanbietern, der mit nicht-interaktiven Sprachen erstellt wurde, interaktiv zu testen!

DDE-Kommunikation mit anderen Applikationen ist ebenfalls enthalten. ProForth-Applikationen können so Daten mit anderen Windows-Programmen austauschen. So kann z. B. eine Spreadsheet-Tabelle in Echtzeit gefüllt werden.

Echtzeitunterstützung

Das Design von ProForth für Windows erlaubt die Entwicklung von Echtzeit-Anwendungen für Windows NT und, soweit die Resourcen das zulassen, für Windows 3.1. Dazu werden die Windows-Timer mit einer Zeitauflösung vom 1 ms unterstützt. Das System unterstützt Multitasking sowohl unter Windows NT (preemptiv) als auch unter Windows 3.1 (kooperativ und Timer-gestützt).

Da viele Echtzeitanwendungen auf dem PC den Anschluß von externen Geräten und Controllern verlangen, bietet ProForth ein einfach zu nutzendes Interface zu der recht komplexen API für

die serielle Kommunikation von Windows.

ProForth ist Forth

Zusätzlich zu der Unterstützung von Windows- und Echtzeitfunktionen bietet ProForth natürlich die 'normalen' Features eines Forth-Systems. Dazu gehören Hardwarefloatingpoint für schnelle und genaue Berechnungen, Overlays um den Code klein zu halten oder Bibliotheken zu verwalten, headerlose Kompilation für spezielle Vokabulare und eine dynamische Speicherverwaltung.

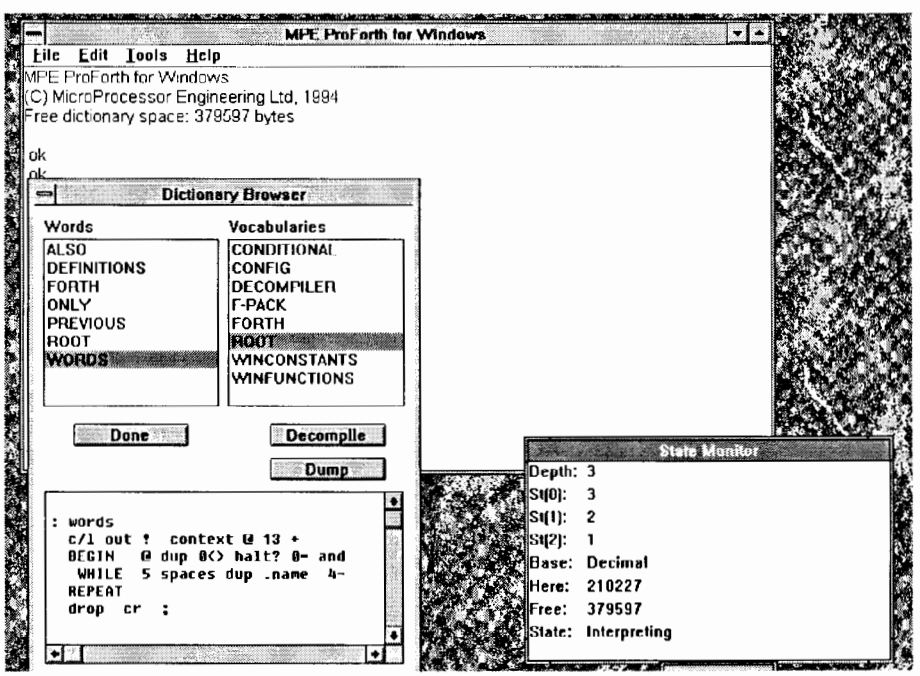
Ein 386/387-Assembler, um zeitkritische Routinen zu verfassen, gehört selbstverständlich zum System. Dabei handelt es sich um einen Prefix-Assembler, der zu Intel Assemblern und denen anderer MPE-Systeme kompatibel ist.

Debugging Tools

ProForth für Windows hat ein traditionelles Forth Interface: Eine Kommandozeile und 'Debug-Konsole'. Auf dieser Konsole erscheinen Meldungen des Programms während der Laufzeit oder während des Debuggens.

Die meisten Forth-Programmierer sind mit der Kommandozeile vertraut. Ein Entwicklungssystem besteht aber nicht nur aus einer Programmiersprache und einigen Bibliotheken, sondern auch aus Entwicklungshilfen wie z. B. einem Debugger.

```
\ Catalogue example
pto

Copyright (C) 1994

MicroProcessor Engineering Ltd
133 Hill Lane
Southampton
UK
SO15 5AF

tel +44 1703 631441
fax +44 1703 339691
email mpe@cix.compulink.co.uk

File            ABOUT.FTH
Started         08/06/94
Author          GIE
Function        Provide example about box

(p)
\ draw the icon

[forth]

create iconfile$       ", appfile.ico"
create about-caption$  ", Example About Box"
create about-line1$    ", Example application for catalogue"
create about-line2$    ", (c) MicroProcessor Engineering Ltd"
create about-line3$    ", June 1994"

#200 constant about-x                \ left
#150 constant about-y                \ top
#300 constant about-w                \ width
#200 constant about-h                \ height

: get-icon                \ icon# -- handle ;
  iconfile$ $>asciiz swap
  winextracticon                     \ open file and get handle
;
```



Folgende Tools gibt es in ProForth für Windows:

- Der 'Dictionary Browser' zeigt eine sortierte Liste aller Worte im System. Von hier aus können sie dekompiliert oder als Dump angezeigt werden.
- Der Statusmonitor zeigt wichtige Systemparameter wie den Stackinhalt, die Zahlenbasis, den freien Speicher sowie den Zustand des Interpreters. Diese Daten sind wichtig, um die Ergebnisse im interaktiven Debugfenster interpretieren zu können.
- Der 'Message Spy'. Mit diesen Spion sieht man, was im Windows los ist und wer wann was mit wem macht.

```
: draw-app-icon          \ --
  curr-window wingetdc dup          \ get and preserve device context
  #136 #10                          \ x y position
  0 get-icon                        \ handle of 1st icon in file
  windrawicon drop                  \ display it
  curr-window winreleasedc drop     \ release context
;

: about-paint-handler  \ hwnd message wparam lparam -- status
  draw-app-icon                         \ draw our icon
  windefwindowproc                      \ pass to default handler
;

: about-finished       \ hwnd message wparam lparam exitcode -- sta-
tus
  curr-window dialog-done !         \ flag as finished
  4drop 0                           \ no need to pass on
;

: ok-button-action     \ hwnd message wparam lparam -- status
  1 about-finished                  \ done (exit-code 1)
;

: about-command-handler \ hwnd message wparam lparam -- status
  case
    dup 1 nth-control = ?of ok-button-action  endof  \ ok button
\   insert any extra control handlers here
    windefwindowproc
  end-case
;

: about-close-handler   \ hwnd message wparam lparam -- status
  2 about-finished                      \ done (exit-code 2)
;

: about-winproc          \ hwnd message wparam lparam -- status
  2 pick                          \ copy of message so we can choose
  case                            \ which ones we want to process
    WM_PAINT    of about-paint-handler      endof
    WM_COMMAND  of about-command-handler    endof
    WM_CLOSE    of about-close-handler      endof
    drop windefwindowproc
  end-case
;

: about-box-init         \ --
  draw-app-icon
  about-line1$ $>asciiz #2 nth-control winsetwindowtext
  about-line2$ $>asciiz #3 nth-control winsetwindowtext
  about-line3$ $>asciiz #4 nth-control winsetwindowtext
;

dialog: about-box
  initialises: about-box-init
  winproc: about-winproc
  4 controls:
    #120 #130 #60  #25 z" OK"   bs_pushbutton "button" control
    #10  #54  #280 #20 z" ..."  ss_center     "static" control
    #10  #74  #280 #20 z" ..."  ss_center     "static" control
    #10  #94  #280 #20 z" ..."  ss_center     "static" control

: draw-about-box         \ --
  set-brush                         \ use white brush
  about-caption$ $>asciiz           \ set caption
  about-x about-y about-w about-h
  about-box drop                    \ draw dialog (ignore exit code)
  handle-of about-box off           \ mark as closed
;
```

WinTed

WinTed unterstützt die Arbeit mit dem Debugfenster und GUIDE. WinTed ist ein Multi-Dokumenteditor zum Bearbeiten von Forth-Quelltexten. WinTed unterstützt die normalen Cut/Copy/Paste und Suchfunktionen. Darüberhinaus bietet er die Möglichkeit, alle veränderten Dateien auf einen Schlag zu speichern, was die Arbeit an umfangreichen Projekten vereinfacht.

Zusammenfassung

MPE ProForth für Windows ist ein komplett ausgestattetes 32-bit Forth für Windows 3.11 und Windows NT. Mit der Interaktivität von Forth und speziellen Werkzeugen wie ProForth GUIDE und WinTed unterstützt es einen schnellen Entwicklungsprozeß von Echtzeit-Windowsapplikationen. ProForth für Windows - More real, less time.

Contacts

MicroProcessor Engineering Ltd
133 Hill Lane, Southampton
SO15 5AF, UK
Tel.: (+44) 1703 631441
Fax: (+44) 1703 339691
email: sales@mpeltd.demon.co.uk

In Deutschland:

Polygon GmbH
Basler Straße 103
D-79115 Freiburg
Tel.: (+49) 761 406692
Fax: (+49) 761 402621

In den USA:

FORTH Inc.
111 N. Sepulveda Blvd
Manhattan Beach
CA 90266-6847
USA
Tel.: (+1) 310 372-8493

❑

Forth als Metasprache

von Michael Symonds
Brandenbaumer Landstr. 48, 23564 Lübeck

Allgemeine Aspekte der Forthprogrammierung diskutiert anhand des AOP-Lösungsansatzes aus "Vorwärts - und dann kreuz und quer II"

Stichworte

Metasprache
Portierbarkeit
Forthstandards

Es ist eine Binsenweisheit - Forth ist irgendwie anders als andere Programmiersprachen. Auf die Frage nach den signifikanten Unterschieden kann jeder einigermaßen "Eingeweihte" einiges zum Besten geben. Bei den *rational und logisch begründeten* Argumenten für Forth wird es da schon etwas dünner. Gerade mal die Vorteile der hohen Interaktivität und des Open-Box-Charakters lassen sich einigermaßen leicht nachweisen. Wer ist aber schon in der Lage, die Vorteile eines Datenstacks *logisch* zu begründen; ich finde, mit dessen Nachteilen hat man es tatsächlich sehr viel einfacher.

Ich möchte daher in diesem Artikel einmal versuchen, ein Forth-spezifisches Leistungsmerkmal zu diskutieren und dessen Vorteile und Problematiken aufzuzeigen: Eignung und Einsatz von Forth als Metasprache. Damit sich dabei nicht Wunsch und Wirklichkeit, wie es so häufig bei allgemeinen Aussagen der Fall ist, zu einem verklärenden Brei vermischen, möchte ich meine Ausführungen anhand des Lösungsansatzes der Automatenorientierten Programmierung aus [VORWÄRTS94] konkretisieren. Andererseits bin ich mir durchaus der Unmöglichkeit bewußt, allgemeine Aussagen an nur einem Beispiel quasi "beweisen" zu wollen.

Begriffsdefinition

Der Informatik-Duden sagt zum Begriff "Metasprache" folgendes:

> *Eine Metasprache ist eine Sprache, in der es möglich ist, eine andere Sprache zu definieren und zu beschreiben.*

Hiermit sind jedoch in erster Linie formale Sprachen gemeint und keine Programmiersprachen. So ist die Definition noch auf Forth umzumünzen:

> Forth ist eine Meta(programmier)-sprache, weil es mit ihr möglich ist, andere Programmiersprachen zu programmieren.

Was bedeutet das aber nun ganz praktisch, welche Konsequenzen hat das - und überhaupt, warum sollen sich daraus Vorteile und Probleme ergeben?

Erweiterungen des Wortschatzes

Forth sieht, in den Augen eines Nichtforthlers - und dementsprechend als reine Programmiersprache betrachtet, relativ schwach auf der Brust aus. Es stehen nur die rudimentärsten Funktionen zur Verfügung, und man sieht sich eher in eine Assemblerwelt versetzt als in eine "richtige" Hochsprache. Und in der Tat ist es ein unsinniges Unterfangen, ernsthafte Applikationen als "reines" Forthprogramm schreiben zu wollen. Vielmehr ist Forth so zu erweitern, daß die Sprachelemente entstehen, in denen die eigentliche Aufgabe wirklich elegant formuliert werden kann.

Aber wer würde denn in seiner Forthapplikation keine neuen Wörter kreieren und damit den Wortschatz seines Forth erweitern? Jedoch alleine mit der Definition von Colon-, Constant- und Variable-Wörtern verwendet man in Forth gerade das, was in jeder anderen Programmiersprache auch zur Verfügung steht. Colon-Definitionen tragen dort nur einen anderen Namen: Subroutinen, Prozeduren, Funktionen und dergleichen. Zugegebenermaßen mit einigen Qualitätsunterschieden zu der erreichten Orthogonalität in Forth, aber das alleine kann nicht der Maßstab sein.

Wenn man die Definition eines Colon-Wortes als eine Spracherweiterung von Forth klassiert, dann heißt das nichts anders, als daß die Definition eines Assemblerunterprogrammes einer Erweiterung der Maschinensprache des Prozessors äquivalent ist. Und das ist nicht so scherzhaft gemeint, wie es sich vielleicht anhört - ich halte diese Klassifikation tatsächlich im Prinzip für gerechtfertigt.

Wenn es das also nicht ist, was Forth von anderen Programmiersprachen unterscheidet, was ist es dann?

Syntax und Semantik ...

Eine Sprache alleine als eine Ansammlung von Wörtern zu betrachten, ist bestenfalls ein Ausschnitt der ganzen Wahrheit. Eine Sprache besteht vielmehr auch aus Syntax und Semantik - den Regeln also, welche Wortkombinationen erlaubt sind und welche Bedeutungen sich daraus ergeben. Und erst genau hier beginnt die metasprachliche Ebene, wie sie in dieser einfachen und unmittelbaren Form nur in Forth zur Verfügung steht.

... am Beispiel der AOP

Was die abstrakten Begriffe Syntax und Semantik ganz praktisch bedeuten, kann man an dem Beispiel der Automatenorientierten Programmierung in [VORWÄRTS94] deutlich machen.

Die Syntax legt die gültigen Wortkombinationen fest, das sind in der AOP im wesentlichen die Ausdrücke AUTOMAT: ... CANAL ... ;AUTOMAT und STATE: ..... ;STATE. Wie in Forth üblich, wird de facto nur eine äußerst spartanische Syntax-Überprüfung durchgeführt. Daß der Compiler eine falsche Syntax nicht erkennt heißt aber nicht, daß keine festgelegt wäre. In der Tat existieren implizit eine Reihe von Syntaxregeln, die uns aber angesichts der Beispiele zu selbstverständlich sind, um sie überhaupt noch erwähnen zu müssen. Für ihre Nichteinhaltung wird sich das Run-Time-Verhalten schon früh genug bedanken...

Die Syntax ist aber nur Mittel zum Zweck - und dieser Zweck heißt Semantik. Die Semantik legt fest, welche inhaltliche Bedeutung die Ausdrücke haben, bzw. was will und kann der Programmierer mit einer bestimmten Syntax zum

Ausdruck bringen? Um aufzuzeigen, daß dem Programmierer beispielsweise mit der AOP tatsächlich eine ganz neue Klasse von Ausdrucksmöglichkeiten zur Verfügung steht, möchte ich hier noch einmal die Funktionsweise der AOP beleuchten - diesmal aber aus einem anderen Blickwinkel, als dies schon in [VORWÄRTS94] geschehen ist.

Das, was ich in [VORWÄRTS94] als "Automat" bezeichnet habe, kann man genauso gut auch als "virtuelle Maschine" betrachten. Auf diese Betrachtungsweise kommt man schnell, wenn man sich vor Augen führt, was denn die eigentliche Forth-Maschine noch zum Gelingen der ganzen Anwendung beiträgt. Dann stellt man fest, daß sie ihr nunmehr kümmerliches Dasein in einer Endlosschleife zubringt, um dort im Falle einer Tastatureingabe eines von fünf möglichen Wörtern aufzurufen. Und wenn wir hunderte von Menüs programmieren würden - es würde sich nichts an dieser Tätigkeit ändern. Das "richtige Leben" scheint sich aber woanders abzuspielen, in eben der virtuellen Maschine und in seinem Herzstück, der Zustandsvariablen. Die Zustandsvariable ist der Instructionpointer (IP) der Maschine. Die Strukturen sind uns von sequentiellen Programmen wohlbekannt: Endlosschleifen von Menüs (siehe Applikation I in [VORWÄRTS94]) Verzweigungen (siehe Applikation II [VORWÄRTS94]) und auch Untermenüs (wie Unterprogramme) wären durch Rettung (Nest) und Restauration (Unnest) der Zustandsvariablen einfach zu realisieren. Was aber tatsächlich bei der Anwendung dieser Maschine vorherrscht ist das GOTO!!! Unser guter alter Bekannter aus den Zeiten, wo man noch ohne Gewissensbisse den größten Schund programmieren durfte. Und um die Verwunderung beim Leser auf einen vorläufigen Höhepunkt zu treiben, behaupte ich auch noch, daß wir es hier mit einem strukturierten GOTO zu tun haben. (Und wenn nun einer sagt, daß ich mir mehr einbilde als da eigentlich ist, dem sage ich, daß er sich dann auch Forth nur einbildet und in Wirklichkeit ständig in Maschinensprache programmiert.)

Jede Instruktion dieser Maschine wird durch die Definition eines seiner Zustände programmiert. Die Definition eines solchen State-Wortes erfolgt in erster Linie durch eine Aneinanderreihung von Forthwörtern. Diese Codefeldadressenreihe ist uns aus den Colon-Definitionen bereits gut vertraut. Daher konnte der schon in Forth zur Verfügung stehende Compilationsmechanismus mit minimaler Änderung verwendet werden, um die State-Wörter zu kompilieren. Aber, und jetzt komme ich endlich zu dem entscheidenden Punkt: Trotz äußerlicher Ähnlichkeit sind zwischen einer Colon-Definition und einer State-Definition keinerlei semantische Gemeinsamkeiten zu finden. Auf der einen Seite wird eine Folge von Instruktionen festgelegt, die nacheinander auszuführen sind - auf der anderen Seite werden Reaktionen auf mögliche Ereignisse definiert.

Ich hoffe, ich konnte hinreichend deutlich machen, was ich unter der Programmierung einer Programmiersprache verstehe. Es geht dabei letztlich immer um die Anpassung des Compilers an die gegebene Aufgabe und an die Zurverfügungstellung neuer Strukturen und Mechanismen. Die herkömmliche strukturierte Programmierung stellt in erster Linie hierarchisch ineinander verschachtelte Prozeduren zur Verfügung. Wirklich effektiv programmieren lassen sich damit nur hierarchisch ineinander verschachtelte Strukturen wie z. B. einen Seitenausdruck (Hierarchie: Seite / Abschnitt / Zeile / Wort / Buchstabe).

Portierbarkeit

Nachdem ich die potentielle Mächtigkeit metasprachlicher Möglichkeiten aufgezeigt habe, möchte ich nun auch zu den potentiellen Problemen kommen.

Eine im Prinzip gerechtfertigte Kritik an AOP.SCR und LINK.SCR wäre die weitgehende Abhängigkeit von der jeweiligen Forth-Implementation und deren inneren Strukturen. (Ich werde diese Eigenschaft im folgenden "Maschinenabhängigkeit" nennen - nicht ganz korrekt - aber anschaulich). Denn ohne jeden Zweifel ist die Portierbarkeit von Software eines der wichtigsten Kriterien und Maßstäbe heutiger Softwareentwicklung.

Ich bin allerdings der Meinung, daß hier ein grundsätzliches Problem von Forth bzw. in der Festlegung von Forthstandards vorliegt. **Es ist die Verwendung von Forth als Metasprache, die zur Zeit eine grundsätzliche Schwierigkeit bei der Portierbarkeit entsprechender Forthprogramme darstellt.** Da metasprachliche Elemente wie schon ausgeführt ein mächtiges und effektives Werkzeug sind, haben vermutlich gerade die nicht ungeschicktesten Basis- und Universallösungen dieses Problem gemeinsam.

Für diese Meinung habe ich folgende Begründung: Ein nicht unbeträchtlicher Teil des Forth-Stammvokabulars bezieht sich mehr oder weniger explizit auf diese metasprachlichen Fähigkeiten und Funktionen. Sie stehen unter anderem deshalb zur Verfügung, weil sie als elementarer Bestandteil des bestehenden Forthinterpreters und -compilers offenliegen. Daraus folgert aber auch die funktionale Abhängigkeit dieser Worte vom inneren Aufbau, sprich der Implementation des jeweiligen Forth-Systems. **Die Forthelemente, die die Eignung als Metasprache und damit eine der größten Stärken von Forth ausmachen, sind genau die Elemente, die am stärksten implementationsabhängig sind.**

In diesem Sinne stellt sich die Frage, inwieweit es denn überhaupt sinnvoll ist, Forth-Standards ins Leben zu rufen, in denen auf implementationsspezifische Details weitgehend und sogar bewußt verzichtet wird. Man hat dann sicherlich Forth als Programmiersprache standardisiert aber als Metasprache kastriert. Ich nenne das eine zu "oberflächliche" Spezifikation - "oberflächlich" nicht im Sinne von "zu wenig Mühe gegeben" sondern im Sinne von "nicht genug Implementierungsdetails spezifizierend".

Portierbarkeit von AOP

Aus der eben beschriebenen allgemeinen Problematik darf aber nicht gefolgert werden, daß man als Forthprogrammierer a priori gegen Windmühlen kämpft, wenn eine möglichst weitgehende Portierbarkeit erreicht werden soll. Denn in der Tat können eine Reihe von Maßnahmen getroffen werden, um das Problem wesentlich einzuschränken. Um auch dieses Thema nicht im abstrakt Theoretischen zu behandeln, werde ich mich wieder weitgehend auf [VORWÄRTS94] beziehen.

Die sicherlich wichtigste und effektivste Maßnahme, die getroffen werden kann, ist das Ausfaktorieren der Maschinenabhängigkeit aus der eigentlichen Anwendungsebene. In unserem Beispiel

beschränkt und konzentriert sich die Maschinenabhängigkeit auf die Teile AOP.SCR (und LINK.SCR). Die Programmierung dieser Spracherweiterung wird immer einen fest umrissenen und relativ kleinen Umfang haben - egal wieviel Megabyte Anwendungssoftware auf Basis dieser Spracherweiterung geschrieben werden. AOP.SCR stellt quasi das Schnittstellenmodul bzw. die Treibersoftware zur jeweiligen Forthimplementation dar.

In meinen Argumenten scheint sich ein kleiner Widerspruch aufzutun, denn habe ich nicht gerade zu Beginn die metasprachlichen Elemente als die Ursache schlechter Portierbarkeit identifiziert? Das ist auch immer noch richtig, denn AOP.SCR ist tatsächlich äußerst implementationsabhängig. Das Problem liegt meines Erachtens darin, daß Spracherweiterungen i. d. R. nicht bewußt als solche programmiert und separiert werden. Dadurch kommt es zu einer intensiven Verstrickung mit der Anwendungssoftware und diese wird geradezu mit implementationsspezifischen Details überschüttet.

Syntax der AOP

Erste Voraussetzung für die Gestaltung einer Spracherweiterung ist sicherlich ein tiefes Verständnis dafür, was und wie damit eigentlich programmiert werden soll. Meiner Erfahrung nach ist gerade hier die Technik des "Rückwärtsarbeitens", wie es in [BRODIE] beschrieben ist, sehr gewinnbringend. Unabhängig davon, was realisiert werden könnte, überlege ich mir, wie die ideale Programmiersprache aussehen müßte, um meine Aufgabe ideal zu formulieren. Das hört sich einfach an, ist aber tatsächlich sehr schwierig, weil man selten eine *neuartige* Aufgabe wirklich versteht und man sich nur sehr schwer von alten Denkmustern lösen kann.

Neben dieser grundsätzlichen Festlegung von Syntax und Semantik einer Spracherweiterung gibt es jedoch noch eine Menge Kleinarbeit zu leisten, denn der Teufel steckt (auch) im Detail (Stichwort: Schnittstellenspezifikation).

Z. B. hätte ich statt einer angepaßten, neuen Semikolon-Definition '`;STATE`' auch das Semikolon der Colon-Definition '`;`' benutzen können. In der Forth-Implementation, die ich benutze, hätte das zufällig gepaßt und hätte auch so funktioniert (Wenn auch mit einer kleinen Unsauberkeit, es wäre nämlich jedesmal unnötigerweise ein UNNEST compiliert worden.) Das muß aber nicht zwangsläufig in allen Implementationen *zufällig* passen.

Ein noch besseres Beispiel ist '`;AUTOMAT`'. Dieses Wort überhaupt einzuführen scheint auf den ersten Blick völlig unnötig zu sein - denn ist nicht mit '`2DROP`' schon alles getan? Und dennoch erfüllt '`;AUTOMAT`' zwei wichtige Funktionen: Zum einen erhöht es die Lesbarkeit erheblich - denn was soll es einem schon sagen, wenn irgendwo im Quelltext mehr oder weniger zusammenhanglos ein '`2DROP`' rumsteht? Dagegen ist die Kombination aus '`AUTOMAT:`' und '`;AUTOMAT`' selbsterklärend.

Die zweite Aufgabe von '`;AUTOMAT`' ist die des Information-Hidings. Vor dem Benutzer dieses Konstruktes werden für ihn unwichtige Implementierungsdetails versteckt. Erst dadurch werden überhaupt Änderungen an den Internas von AOP.SCR und Portierungen auf andere Forth-Implementierungen problemfrei möglich gemacht. Es liegt hier das gleiche Prinzip vor, wie bei einer Konstantendeklaration. Es wird eine unnötige Redundanz innerhalb der Software vermieden. Statt der Information "Lösche die obersten beiden Stackwerte" steht nun "Tue alles, was notwendig ist, um die `AUTOMAT:`-Definition abzuschließen". (Um das ganze theoretisch zu formulieren, kann man auch sagen, daß erst der Name `;AUTOMAT` den richtigen semantischen Inhalt wiedergibt. Hier wird also der Begriff der Semantik wieder ganz praktisch.) Was auf die Aussage "Tue alles, was notwendig ist, um die `AUTOMAT:`-Definition abzuschließen" hin zu tun ist, ist da definiert, wo es hingehört, nämlich in AOP.SCR und nicht hundertfach kopiert in der Anwendungssoftware, die auf Basis von AOP.SCR geschrieben wird. Es ist z. B. auch denkbar, daß '`;AUTOMAT`' auch nur ein '`NOOP`' ausführt. Dann nämlich, wenn es dem Programmierer eines alternativen AOP.SCR gefällt, die beiden Werte, die sich in meiner Version auf dem Stack befinden, in zwei Variablen zu halten. Nicht, daß ich das für eine bessere Vorgehensweise halten würde, aber man weiß nie, was in einer anderen Umgebung und unter anderen Zielsetzungen sinnvoller wäre.

Fazit

Die Eignung von Forth als Metasprache ist einer der wesentlichsten Unterschiede zu anderen Programmiersprachen. Daß mit metasprachlichen Elementen nicht nur die Schaffung von einfachen Colon-Definitionen gemeint sein kann, habe ich aufgezeigt. Vielmehr gehört dazu die Schaffung neuer syntaktischer und semantischer Ausdrucksmöglichkeiten, die der jeweiligen zu formulierenden Aufgabe angepaßt sein müssen. In dem Moment, wo die Sprache ideal an das Problem angepaßt ist, ist aus Forth eine überaus lesbare und damit wartbare Programmiersprache geworden. (Ob allerdings diese ideale Anpassung unter den *realen Randbedingungen* (Zeitrespektive Geldnot) einer Softwareentwicklung überhaupt in der Mehrzahl der Fälle erreicht werden kann, ist meiner Erfahrung nach mehr als fraglich.)

Eines ist hoffentlich auch deutlich geworden: Die OOP ist noch lange nicht der Stein der Weisen. Eine wertvolle Anregung sicherlich - aber das Paradigma bzw. das Bild des Objektes alleine ist für viele Probleme nicht genügend aussagefähig. Sicherlich kann man das, was ich eine "Virtuelle Maschine" genannt habe auch als "Objekt" bezeichnen - das ist dann in etwa so, als wenn man Marilyn Monroe *ausschließlich* als "junge Frau mit blonden Haaren" beschreiben würde.

Verweise:

[BRODIE]
"Thinking Forth" von Leo Brodie
[VORWÄRTS94]
Artikel "Vorwärts - und dann kreuz und quer" in dieser VD

❑

Vorher <--> Nachher

von Arndt Klingelberg
Strassburger Straße 12, 52477 Alsdorf

F-PC portiert Code!
PREfix - POSTfix Wandlung fix gemacht.

Stichworte

CODE - mASM
8086
F83
ZF , TurboForth
F-PC
F-PC-ak

Gerade die Meßtechnik erfordert viel Lowlevel Code. Sei es, daß Geschwindigkeit gefordert ist, sei es, daß es einfacher ist, mit Registern parallel zu arbeiten als am Stack seriell sich zu vertricksen, oder sei es, daß Interrupt-Prioritäten usw. verbogen werden (z. B. für die serielle Schnittstelle oder den Timer). Da aber für einen PC eigentlich alles schon mal da war, zumindest in der großen Familie von F83 bis F-PC, kann alles übernommen werden. Wenn ... ja, wenn man es denn findet (das logistische Thema stellen wir uns mal gelöst vor) und wenn ... ja, wenn nicht immer alles im gerade falschen ASM-Format vorliegt und zudem eine (sinnvolle) Beschreibung des notwendigen Formats fehlt.

Ein F-PC-Nutzer kann hier nur erstmal schmunzeln. Er steht über solchen Problemen. Einerseits bereitet es kaum Probleme irgendwelche Assembler Quelltexte aus üblicher (Nicht-Forth) Literatur zu übernehmen. Da die F-PC Syntax MASM-ähnlich ist (PREfix) müssen nur hier und da einige Leerzeichen 'eingestreut' werden. Soll anderseits Code von F83 und dessen direkten Abkömmlingen wie TurboForth (Jedi) oder ZF übernommen werden, so wird auf F83 (typische reverse Forth-) Syntax umgeschaltet, und zwar mit POSTfix. POSTfix heißt, daß zuerst die (Stack-) Argumente kommen und dann später (lateinisch, nicht postalisch: POST) die eigentliche Instruktion. Dem Rest der Welt ist es gebräuchlich, daß zuerst (lat. PRE) die Instruktion kommt. So sehr ich beim Eintippen in den Rechner Anhänger der UPN (siehe unten) bin, so sehr ziehe ich die normale Schreibweise auf dem Papier vor und auch die PREfix MASM Notation. Ich halte es für deutlich schneller lesbar bzw. fehlerfreier erfaßbar.

Nachher --> Vorher

Aufgund dieser Umschaltbarkeit eignet sich F-PC aber auch vorzüglich dazu, Code 'als Dienstleistung' für kleine Forthe zu konvertieren. Ich lernte das zu schätzen, als ich 'die Uhr' (VD 92Q4, 93Q1, 93Q2, 93Q3) enterashen wollte. Die Konvertierung von F83 zu Masm ist Standard in F-PC. Es wird mit PREfix kompiliert und dann in eine Datei hinein dekompiliert bzw. genauer disassembliert. Dekompiliert wird typisch mit DIS8086.seq in prefix.

```
>B see code-word
```

Nach geringfügigem Edieren steht dann der Code übersetzt zur Verfügung. Typisch muß 'nur' HEX davor gesetzt werden, nicht vergessen! Schließlich ist INT 15h == INT 21d. Dann wird JMP 108 gegen NEXT ausgetauscht (JMP 107 == 1push, JMP 106 == 2push). Das schnelle inline-NEXT ist definiert als:

```
LODSW ES:
JMP   AX
```

Es wird disassembliert zu:

```
ES:
LODS  WORD
JMP   AX
```

was das gleiche bedeutet. Das könnte natürlich so stehen bleiben, ist ja nicht falsch, normale F-PC-Schreibweise ist aber INLINEON NEXT. Hochsprachenartige Strukturen wie CX-DO .. CX-LOOP, HERE .. LOOP, IF .. THEN, BEGIN .. UNTIL werden natürlich in JMPs übersetzt, wobei nur jeweils eines der BranchWorte in irgendeinen JMP gewandelt wird. Auch in Highlevel Forth ist es ja gar nicht so einfach, diese 'Branch'-Strukturierung sauber beim dekompilieren zurückzuzaubern. Auch das ist wieder eine Frage des Stils, ediert werden muß es NICHT.

In der Originalversion von F-PC muß dann noch END-CODE ergänzt werden. Dafür heißt es statt JMP 108 nun besser interpretierbar JMP NEXT. Das muß aber auch ediert werden, und zwar entweder zu nur NEXT oder zu JMP >NEXT.

Der Trick mit dem >BROWSER, um Forth-Ausgaben in eine Datei umzuleiten, bereitete manchmal Probleme. Ich nutze dann auch hier SNIPP.com, um den Code aus dem Bildschirm auszuschneiden und in eine Datei zu 'pasten'. Sicherlich nutzen die 'Aufnehmer' (VD 4/93) hier.

Vorher --> Nachher

Umgekehrt ist es (etwas) schwieriger (umgekehrt zum realen Leben). Es wird wie gewohnt kompiliert, dann aber muß das 'alte' DISASSEM.seq geladen werden. Am besten wird vor dem Laden das Dateiende, und zwar alles nach

```
DIS:   ( <name> -- )
```

per \S deaktiviert. Dann steht das 'alte' SEE (mit dem wohlmöglich vorher geladenen prefix DIS) noch zur Verfügung. Zuerstmal wird man über das Ergebnis des Disassemblierens enttäuscht sein. Das Disassemblieren geschieht auf Tastendruck im Einzelschritt, das Ende des Codewortes muß man kennen (typisch hier auch 108 .. JMP,). Es geht sonst endlos weiter.

Nun ist einiges zu edieren. So enden alle Instruktionen mit Komma (zwar Forth typisch, zumindest für F-PC aber falsch), zudem sind alle am Zeilenanfang voreilenden und auch nachkommenden Adressen und OP-Code-Bytes zu löschen oder auszukommentieren. Aber mit Key-Macros geht auch das teils voll-, teils halbautomatisch. Auch muß rund um das W bzw. BX -Register ediert werden. Immerhin führt es aber zum richtigen Ergebnis. Ausnahmen bestätigen die Regel. Bei [FAR] CALLs bzw. JMPs und selteneren OP-Code-Kombinationen ist

probieren angesagt. (Einige seltene Instruktionen aus der 8086-Vielfältigkeit kennt auch die letzte DIS8086.seq Version noch nicht.) 'Damals' zu F83 und DISASSEM - Zeiten wurde/mußte kaum FAR-geCALLt werden. Rechner waren klein und die Systeme auch, das konnte vernachlässigt werden. Und in solchen Fällen wird dann auch typisch unter Einbeziehung des roten Knopfes 'hardwarenah' am Rechner probiert.

Bei allem Edieren sollte natürlich beachtet werden, daß ja wohl nicht für F-PC sondern eine anderes Zielsystem ediert wird. Und wenn zurück in Blöcke gewandelt wird, so ist das natürlich dank SEQtoBLK.seq prinzipiell kein Problem. Erforderlich sind einige Layout Änderungen, wie eben ein Zeilenumbruch nach max. 64 Zeichen und Quelltext-Einheiten von 14 Zeilen Länge.

Help ! Code !

Nicht nur für den Anfänger wichtig ist sicherlich, daß nur in der ak-Version von F-PC, das Hilfesystem für ASM-Instruktionen zur Verfügung steht. Im Original-F-PC ist nur ein recht rudimentärer Hilfetext vorhanden. Zudem zeigt dort das Anklicken der Worte (VIEW HELP) in den Disassembler hinein, statt in den Assembler-Hilfe- oder Quelltext.

VolksForth

VolksForth ist hier leider völlig verschieden. VolksForth verwendet die Register deutlich anders. Hier ist wieder eine gewisse Übereinstimmung mit tCOM vorhanden. Die Schreibweise ist für Anfänger kritisch. Eingefleischte Forther werden aber die kurze, klassisch forthige Schreibweise lieben, die sich für Screens gut eignet, etwas einzigartiges in beiderlei Sinn.

SEE ANS

Da in Forth das Dekompilieren und Disassemblieren doch meist sehr schön läuft, kann ich nur empfehlen, es doch auch öfter zu nutzen. Viele Fehler können vor dem Ausführen entdeckt werden, indem SEE genutzt wird. Und während oben POSTfix in PREfix vice versa umgesetzt wurde, kann Highlevel in F-PC-ak ab v.4.2 auch Forth83/F-PC weitgehend in ANS gewandelt werden. Alles dekompiliert zu ANS-Namen! Auch ein sinnvoller ANS-Lerneffekt: Wort mit Alt-D anklicken, und schon steht die ANS-Dekompilierung unter dem Forth83 Quelltext. Umgekehrt geht das mit der Original F-PC Version und Ulrich Hoffmanns ANSI.seq: ANS rein, F-PC bzw. Forth83 raus. Das brauchen wir alles nicht immer, aber wohl erstmal deutlich öfter.

❑

UPN

immer wieder genannt,
immer wieder verkannt.

(UPN == RPN == reversed polish notation)

Jan Lukasiewics stellte die Rechenzeichen (auch mehrfach gruppiert, ohne Nutzung von Klammern) vor die Zahlen.

Bill Hewlett drehte die polnische Schreibweise um, und gab die Zahlen immer zuerst ein.

Beispiele zur DISassemblierung (RTC-Start aus NEW-MSZF.seq)

Die Adressen (z. B. $E606, $82B0 bzw. $8EFC) sind abhängig vom Ladezustand.

F-PC POSTfix per DISASSEM.seq (single-step), eingefangen per SNIPP.com.

```
see start-rtc ESC_TO_EXIT
  82B2 59   CX POP,                 59                Y
  82B3 5A   DX POP,                 5A                Z
  82B4  6   ES PUSH,                06                .
  82B5 8C   CS AX MOV,              8C C8             .H
  82B7 8E   AX ES MOV,              8E C0             .@
  82B9 BB   82B0 # W MOV,           BB B0 82          ;0.
  82BC B8   8300 # AX MOV,          B8 00 83          8..
  82BF CD   15 INT,                 CD 15             M.
  82C1  7   ES POP,                 07                .
  82C2 E9   108 NEXT JMP,           E9 43 7E          iC~
  82C5 E9   289 JMP,                E9 C1 7F          iA.
  82C8  2   0 [W] DL ADD,           02 17             ..
  82CA E8   108 NEXT CALL,          E8 3B 7E          h;~
```

...kein automatisches Ende! Hier schon das FolgeWort MS disassembliert.

F-PC org. PREfix per DIS8086.seq, eingefangen über >B in browse.prn.

```
CODE START-RTC
   POP      CX                    \ 167A 8EFE  59
   POP      DX                    \ 167A 8EFF  5A
   PUSH     ES                    \ 167A 8F00   6
   MOV      AX, CS                \ 167A 8F01  8C C8
   MOV      ES, AX                \ 167A 8F03  8E C0
   MOV      BX, # 8EFC            \ 167A 8F05  BB FC 8E
   MOV      AX, # 8300            \ 167A 8F08  B8  0 83
   INT      15                    \ 167A 8F0B  CD 15
   POP      ES                    \ 167A 8F0D   7
   JMP      NEXT                  \ 167A 8F0E  E9 F7 71    iwq
```

Im Fall von >B automatischer Stop, sonst Zwischenstop per Leerzeile (<ESC> stoppt endgültig). Das Forth Kommando lautete:
`' start-rtc dis` bzw. `see start-rtc`.

F-PC-ak PREfix per DIS8086.seq, eingefangen über >B bzw. >B+.

```
see start-rtc
^^^^^
---> command line interpreted  94feb22   22:29'46".810

\ START-RTC         voc.: FORTH          file: NEW-MSZF    35 (loaded interact.)
\ instruc|operators          \ _address_|+0|+1|+2|+3|+4|  _byte_+0_|_byte_+1_

CODE START-RTC
   POP      CX               \ 167A:E608 59              0101-1001
   POP      DX               \ 167A:E609 5A              0101-1010
   PUSH     ES               \ 167A:E60A 06              0000-0110
   MOV      AX, CS           \ 167A:E60B 8C C8           1000-1100 1100-1000
   MOV      ES, AX           \ 167A:E60D 8E C0           1000-1110 1100-0000
   MOV      BX, # E606       \ 167A:E60F BB 06 E6        1011-1011 0000-0110
   MOV      AX, # 8300       \ 167A:E612 B8 00 83        1011-1000 0000-0000
   INT      15               \ 167A:E615 CD 15           1100-1101 0001-0101
   POP      ES               \ 167A:E617 07              0000-0111
   JMP      108              \ 167A:E618 E9 ED 1A        1110-1001 1110-1101
     END-CODE
```

Mit automatischem Stop und END-CODE. Die Bitweise Darstellung von bis zu zwei OP-Code-Bytes erlaubt die komfortable exakte Analyse der umfangreichen 8086 Instrutionen. Bei Nutzung von >B erfolgt eine automatische Protokollierung der Forth Eingabezeile und des Zeitpunktes in browse.prn. SEE gibt zudem das Vocabulary an, wie auch den Namen des Quelltextes (und die Zeile) und die Stelle, von dem aus dieses File geladen wurde.

Stilles Forth

Standardausgabe weggeleitet

von Claus Vogt
Eberstraße 10, 10827 Berlin

Die Ausgabe von Zahlen und Texten auf dem Bildschirm leistet wohl jede Programmiersprache. In Forth ist sie besonders pfiffig gelöst, unter anderem kann Sie auf den Drucker oder in eine Datei umgeleitet werden. Das vorliegende Prgrämmchen leitet alle Ausgaben wahlweise in einen Speicherbereich um oder unterdrückt sie ganz. Das ist bei aufwendigem Bildschirmaufbau manchmal ganz nützlich. Gleichzeitig ist es ein einfaches Beispiel für die Umleitung der Standardausgabe in F-PC 3.56.

```
\\ BUFOUT.SEQ    redirecting Output to a Buffer.                  clv1991

-------------------------------------------------------------------------

Purpose:This Utility puts subsequent output to a buffer in memory
        instead of the console.

        Set the desired standard-output vector prior to load this file
        to make NOBUFOUT work as intended.
Usage:  SETBUFOUT ( seg addr len -- )     \ set buffer to len bytes at seg:addr
                                          \ Allocate buffer yourself
                                          \ if len==0 all output is suppressed.
                                          \ if buffer is full, chars are lost.
        BUFOUT                            \ redirect output to buffer
        NOBUFOUT                          \ output back to console.
        GETBUFOUT ( seg addr len -- )     \ filled output buffer

see also: BUFOUT.HLP for a standard remark form

Author:  1993 Claus Vogt, Mitglied der Forth-Gesellschaft
         Bülowstr. 67, D-10783 Berlin, Germany, 0(049)30/216 89 38

-------------------------------------------------------------------------
```

Zur Umleitung der Standardausgabe im F-PC können die deferred-Worte EMIT, TYPE und TYPEL umgeleitet werden. Ein einziges deferred-Wort TYPEL würde im Prinzip reichen. Trotzdem empfiehlt es sich, alle drei umzuleiten, auch wenn die Worte EMIT und TYPE durch Worte wie BUFTYPE und BUFEMIT ersetzt werden, die universell auf jedes TYPEL aufsetzen können. Denn einerseits verlangt die lange Historie von Forth uns dieses Opfer ab, und andererseits kann uns für die noch längere Zukunft von Forth kein Mensch versprechen, daß die im System gerade aktiven EMIT und TYPE diesen Trick der Rückführung auf TYPEL kennen.

Wer sich durch mein scheußliches Touristenenglich durchgequält hat, wird im Listing keine großen Stolpersteine mehr finden. Hinzuweisen ist noch auf das sehr pfiffige SP@ in BUFEMIT. Den Vorschlag hörte ich erstmalig von Klaus Schleisieck vor ca. 5 Jahren. Er funktioniert auf sehr vielen Forthsystemen.

```
{ \ compiler setup

Only Forth also definitions  decimal

}


{

2VARIABLE       BOADDR          \ seg:addr of outputbuffer
VARIABLE        BOLEN           \ length of buffer
              0 BOLEN     !     \ a buffer of length zero suppresses all output
VARIABLE        BOYET           \ bytes free

: BUFTYPEL ( seg addr len -- )
        BOLEN @ 0= IF 3DROP EXIT THEN
        >R
        BOLEN @ BOYET @ - R@ U<          \ full?
        IF
                BOADDR 2@ R@ +  BOADDR 2@  BOLEN @ R@ -  CMOVEL
                BOLEN @ R@ -  BOYET !
        THEN
        BOADDR 2@ BOYET @ + R@ CMOVEL
        R> BOYET +! ;

: BUFTYPE ( addr len -- )  ?CS: -ROT TYPEL ;

: BUFEMIT ( key -- ) SP@ 1 TYPE DROP ;

: NOOUT
        ['] drop  IS emit
        ['] 2drop IS type
        ['] 3drop IS typel
        ;

: BUFOUT
        ['] bufemit  IS emit
        ['] buftype  IS type
        ['] buftypel IS typel
        ;


: NOBUFOUT              \ returns to previous output
        [ ' emit  >body @ up @ + @ ] Literal IS emit
        [ ' type  >body @ up @ + @ ] Literal IS type
        [ ' typel >body @ up @ + @ ] Literal IS typel
        ;

: SETBUFOUT ( seg addr len -- ) \ set output buffer
        BOLEN !  BOADDR 2!  BOYET OFF ;

: GETBUFOUT ( -- seg addr len ) \ get address and length of filled output
        BOADDR 2@  BOYET @ ;

}
```

Der Block mit den Testroutinen ist wegkommentiert. Ich hebe sie gerne auf, man weiß ja nie, wofür man sie nochmal braucht. Wer sie laden möchte, kann z. B. MLOAD benutzen, sollte aber vorsichtig sein, weil das POINTER-Konzept von F-PC 3.56 manchmal mit ALLOC und DEALLOC nicht so gut zusammenarbeitet.

❑

Das vorgestellte Programm wurde für ein **Mixed-Language-Interface** entwickelt, das Aufrufe von F-PC aus Microsoftsprachen und anderen heraus ermöglicht. Für eine der nächsten Ausgaben würde ich gerne einen Artikel über **gemischtsprachige Programmierung** schreiben. Wer kann mir dazu noch Artikel, Literaturhinweise und/oder Quellen geben? Bitte an meine Adresse:

Claus Vogt, Ebersstr.10, D-10827. Tel. 030/782 81 79 pa(flb) oder in die KBBS für clv.

Alle Einsendungen werde ich ordnen und dem Archiv stiften.

❑

```
{ \\ Test:

: oda
        GETBUFOUT 2DROP DEALLOC
        0 0 0 SETBUFOUT ABORT" cannot dealloc " ;

: .o
\       NOBUFOUT
        BOADDR 2@ BOLEN @ 0
        ?DO     CR
                2DUP I + 40 TYPEL
                ?keypause
        40 +LOOP 2DROP
        \ BOADDR 2@ BOYET @ LDUMP CR
        \ ." after:" BOADDR 2@ BOYET @ + 20 (TYPEL)
\       BUFOUT
        ;
: oba
        1000 ALLOC ABORT" no memory"
        0 ROT SETBUFOUT
        ;
```

```
: t BUFOUT
    BEGIN
        query interpret
        0 0 at .o
    AGAIN ;

: tt BUFOUT
     BEGIN key dup emit 27 = UNTIL NOBUFOUT ;


: ttt BUFOUT      WORDS NOBUFOUT ;
)
```

Vorwärts - und dann kreuz und quer II

von Michael Symonds
Brandenbaumer Landstraße 48, 23564 Lübeck

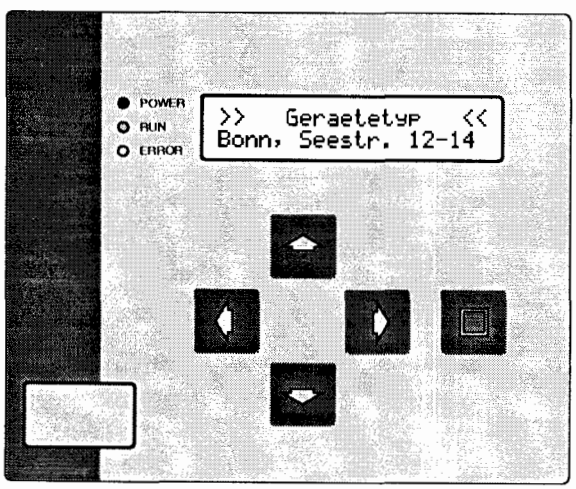
AOP - Automatenorientierte Programmierung
Paradigma, Konzept und Technik zur Programmierung von Menüsteuerungen (... und weiterer Aufgaben)

Stichworte

Automatentheorie
OOP
Menüprogrammierung

In [VORWÄRTS93] wurden Problematik und Realisation einer bestimmten Klasse von Benutzeroberflächen behandelt. Da dieser Artikel schon grundsätzliche Aspekte gut nachvollziehbar herausgearbeitet hat und die aufgezeigte "Minimalapplikation" gut gelungen ist, da sie tatsächlich schon alle wichtigen Elemente enthält, möchte ich an diesen Artikel anschließen und hier meinen Vorschlag einer generellen Lösung vorstellen.

Ich habe mich sehr intensiv mit dieser Aufgabenstellung beschäftigt und ein überaus komplexes Thema vorgefunden. Da die Ebenen, die ich aufzeigen möchte, entsprechend vielschichtig sind, habe ich daraus gleich drei Artikel gemacht, die alle in dieser VD zu finden sind.



Aufbau des Bedienfeldes Abb. 1

Sie gehen von
- einer ganz praktischen Seite (diesem Artikel hier)
- über eher Philosophisches ("Forth als Metasprache")
- zu dem ernsten Teil ("Die Neuentwicklung des Bits")

über.

Vom letzteren mag sich jeder selber fragen, was ich daran ernst gemeint habe und was nicht.

Das Verfahren, das ich hier vorstellen möchte, hat einen grundsätzlich anderen Ansatz als die Varianten von Friedrich Prinz und Michael Major. Und so möchte ich, vor der Erläuterung der Implementierungsdetails, die Aufgabenstellung und das Anforderungsprofil aufzeigen, unter dem ich dieses Verfahren entwickelt habe.

Aufgabenstellung

Die Aufgabe bestand in der Programmierung eines Bedienfeldes wie es in Abb. 1 zu sehen ist. Es beinhaltet im wesentlichen fünf Eingabetasten und eine LCD-Anzeige mit zwei Zeilen zu je 20 Zeichen. Das Bedienfeld, das als Folientastatur mit Klarsichtfenstern für LCD- und LED-Anzeigen sowie einer Einschubtasche für ein kundenspezifisches Logo ausgerüstet ist, hat einen bewußt einfachen und klar strukturierten Grundaufbau. Es dient zur manuellen Ablesung und Bedienung von Energiecontrolling-Geräten und soll in diesem Zuge auch von "Nichtinformatikern" (z. B. einem Hausmeister) ohne große Schwellenangst und lange Einarbeitungszeit bedient werden können.

Ein wichtiger Aspekt für die Gestaltung des Bedienfeldes war die Vermeidung applikationsspezifischer Details. Diese Folientastatur wird nicht nur in einem spezifischen Gerät eingesetzt sondern in verschiedenen Geräten eines ganzen Systems, in dem unterschiedliche Leistungsklassen und Ausbaustufen realisiert sind. Dieser Vielfalt der Hardware stehen ebenso vielfältige Softwarevarianten gegenüber. Zudem sind immer wieder kundenspezifische Erweiterungen der Software und damit auch der Programmierung der LCD-Anzeige erforderlich. Und so kann und soll Abb. 2 nur die Prinzipdarstellung bzw. eine mögliche Variante des Menüaufbaues wiedergeben.

Anforderungsprofil

Die universelle Lösung, die zu finden war, sollte erst einmal allen üblichen Kriterien genügen: Sparsamer Umgang mit Rechner-Ressourcen, sehr gute Lesbarkeit und dergleichen mehr ...

Autorenprofil

Michael Symonds, 34 Jahre, ledig
Seine beruflichen Wurzeln liegen, als gelernter Werkzeugmacher und Maschinenbautechniker, im Maschinenbaubereich. Anfang 1990 machte er sein jahrelang mehr oder weniger intensiv ausgeübtes Hobby zum Beruf und ist seitdem geschäftsführender Gesellschafter der TEXEM GmbH, Lübeck. TEXEM entwickelt und produziert komplette Gerätesysteme zur Verbrauchsdatenerfassung und führt kundenspezifische Entwicklungen durch. In diesem Zuge programmiert M. Symonds die Firmware von Energiecontrollinggeräten auf Basis von Forth.
Ansonsten ist ihm Forth das, was dem Physiker sein Teilchenbeschleuniger ist, das geeignetste Experimentierfeld seiner eher philosophischen Zielsetzungen, der Erlernung und Erforschung des Wesens von Software - seinem Traum von einem ultimativen Paradigma der Softwareentwicklung.

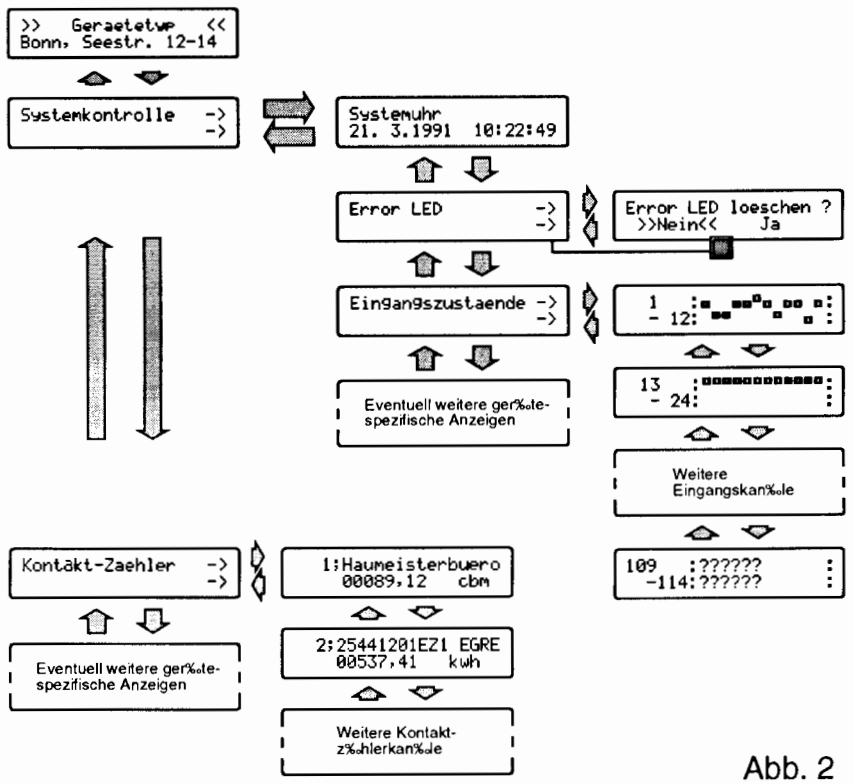
Ein schon eher anwendungsspezifisches Problem stellte das Timing-Verhalten dar. Das Handling des Bedienfeldes ist notwendigerweise einer der am niedrigsten priorisierten Prozesse im Gesamtsystem. Trotzdem, sollte die Tastatur eine saubere, also nicht nachlaufende, Repeatfunktion besitzen, da teilweise über hundert Anzeigefelder (z. B. Verbrauchszähler einer Wohnanlage) durchzublättern sind. Zudem sollten einige Anzeigen zyklisch refreshed werden, um immer den aktuellen Wert anzuzeigen.

Die eigentliche Hauptaufgabe aber und das wesentliche Kriterium, dem eine generelle Lösung gerecht werden mußte, war eine vollständige Modularisierungsmöglichkeit der einzelnen Anwendungsmodule bzw. Komponenten, die die LCD-Anzeige verwenden, quasi "beliefern" sollten. D. h. beispielsweise, daß die Anzeige von Analogeingängen auf der LCD-Anzeige wirklich auch im "Analogeingangsmodul" definiert werden sollte - oder die Anzeige von Digitaleingängen im "Digitaleingangsmodul" - oder die Anzeige eines XYZ-Stromzählers im "XYZ-Stromzählermodul". Mir schwebte von Anfang an ein ähnliches Verfahrensprinzip vor, wie es in idealer Form auch im Forth-Vokabular angelegt ist. Das Forth-Vokabular ist nach außen hin (gegenüber dem Benutzer) eine einheitliche, geschlossene Ressource, die von jeder Stelle des Quelltextes her erweitert werden kann. Also die Ausgabewörter der Analogeingänge werden im Analogeingangsmodul definiert, die Anzeigewörter von Digitaleingängen im Digitaleingangsmodul usw. usf.. Der Benutzer merkt nichts mehr von den ursprünglichen Modulen. Die Module können nach Belieben eingebunden oder weggelassen werden. Universelle Handhabungen der Ressource wie `WORDS`, `FIND` und die `QUIT`-Schleife sind Bestandteil des Betriebssystems und in ihrer Ausführung unabhängig vom Inhalt und Umfang des Vokabulars. Um es populär zu formulieren: Meine Zielsetzung war eine objektorientierte Betriebssystemerweiterung. Eine Formulierung, die ebenso schwammig wie nichtssaussagend ist - hört sich aber sehr elegant an.

Problemanalyse

Was ist nun aber die Aufgabe des Betriebssystems und was die Aufgabe der Anwendungsprogrammierung? Da gibt es auf der einen Seite die Tastenbetätigungen - sie sind zu entprellen (und zu repeaten) und fein säuberlich in eine Empfangsqueue zu schieben - das kennt man ja schon. Wir haben es also mit einem kontinuierlichen Strom von Ereignissen (Events) zu tun, auf die das Gerät zu reagieren hat. Die Anzahl der Ereignisse ist recht klein und übersichtlich - in unserem Fall (siehe Applikation II) fünf Tastenbetätigungen + Refresh = sechs Events. Die Aktionen (Event-Handler), die auf diese Events erfolgen sollen, sind jedoch ganz unterschiedlich - je nachdem an welchem Menüpunkt sich die LCD-Anzeige befindet - präziser formuliert - in welchem Zustand sich die LCD-Anzeige befindet. In jedem Zustand der LCD-Anzeige stehen also den fünf möglichen Events immer jeweils eine Gruppe von fünf Event-Handlern gegenüber. Somit sind wir schon zur sogenannten wahren Anforderung der Aufgabe vorgedrungen.

Beispiel einer LCD-Men¸-Struktur



Beispiel einer LCD-Menüstruktur

Realisation

Jedem Tasten-Event wird ein Forth-Wort zugeordnet, das aufgerufen wird, sobald das jeweilige Ereignis eintritt: `'DO_UP'`, `'DO_DOWN'`, `'DO_LEFT'`, `'DO_RIGHT'`, `'DO_ENTER'` und `'DO_REFRESH'`. Außerdem wird eine Variable benötigt, in der sich der momentane Zustand der LCD-Anzeige befindet: `LCDISPLAY`. Diese Variable muß nun auf jeweils eine 6er-Gruppe von Forth-Worten, den Event-Handlern zeigen (z.B. `'(DO_UP'`, `'(DO_DOWN'`, `'(DO_LEFT'` ...), die tatsächlich zur Ausführung kommen sollen, wenn ein Tasten-Event auftritt.

Implementation

Die Implementation aller Aufgaben und Funktionen, die bisher genannt worden sind (und noch einige mehr, die ich erst im Zuge der Details erläutern werde), können auf Basis des Quelltextes AOP.SCR in Abb. 3 programmiert werden. Obwohl AOP.SCR praktisch nur drei dünn beschriebene Screens umfaßt, realisiert es doch ein recht ausgeklügel-

tes Konzept einer speziellen Programmiertechnik. Ich nenne sie Automatenorientierte Programmierung (AOP).

LINK.SCR in Abb. 4 ist als *spezialisierte* Erweiterung von AOP.SCR zu sehen. Eine spezialisierte Erweiterung deswegen, weil damit Vorwärtsreferenzen erzeugt werden können, wie sie bei *bestimmten* Anwendungen (z. B. eines LCD-Menues) von AOP.SCR benötigt werden. Für gewisse andere Anwendungen ist LINK.SCR nicht geeignet (siehe z. B. [BITERFINDUNG]). Eine ausführliche Erläuterung von LINK.SCR erfolgt weiter unten.

Die Wirkungsweisen von AOP.SCR und LINK.SCR sollen anhand des sehr gut vereinfachten Beispiels aus [VORWÄRTS93] erläutert werden.

Applikation I

In Abb. 5 ist zu sehen, wie die Aufgabenstellung aus [VORWÄRTS93] auf Basis von AOP.SCR und LINK.SCR zu programmieren ist. Dieser und alle anderen Quelltexte dieses Artikels sind auf dem Atari-volksFORTH-83 3.80 entwickelt worden und sollten dort unproblematisch lauffähig sein.

In Abb. 7 ist der dazugehörige Zustandsgraph dargestellt, der in kürzester Form die wahren Anforderungen der Aufgabenstellung wiedergibt: Aufgrund von zwei möglichen Ereignissen muß in drei verschiedene Zustände gewechselt werden.

Im Folgenden werde ich, angefangen vom allgemeinen Konzept der AOP, immer detaillierter in die konkrete Realisation der inneren Wirkungsweise von Applikation I und damit von AOP.SCR eindringen.

AOP

Dreh- und Angelpunkt der AOP ist der Begriff des Automaten. Es ist damit exakt der Automat gemeint, wie er aus der Automatentheorie der theoretischen Informatik bekannt ist. Ein Automat besteht aus einer Menge von separaten Zuständen, einem Eingabe- und einem Ausgabealphabet und einer Menge von Regeln. Diese Regeln bestimmen, wie der Automat in jedem seiner Zuständen auf die Eingabezeichen zu reagieren hat. Die möglichen Reaktionen auf ein Eingabezeichen bestehen in der Änderung seines Zustandes und/oder der Ausgabe eines Ausgabezeichens oder daß der Automat gar nicht reagiert.

Dieses Modell, aus sehr abstrakten Elementen zusammengesetzt, erscheint erst einmal sehr theoretisch und unanschaulich. Wie ich aber hoffe, aufzeigen zu können, paßt diese Konstruktion sehr genau auf unser Problem der Menuesteuerung. In der Realisation der Automatenorientierten Programmierung werden sich auch alle Elemente (***Menge der Zustände, Eingabealphabet, Ausgabealphabet, Funktionsregeln***) des theoretischen Automaten wiederfinden lassen.

Aufbau eines AOP-Automaten

In Abb. 8 ist anhand des Applikationsbeispieles aus [VORWÄRTS93] einmal exemplarisch der komplette Aufbau eines konkreten Automaten der AOP aufgezeichnet. Er entspricht genau dem Applikationsbeispiel in Abb. 5. Bis zum Absatz "AOP zusammengefaßt" wird auf Abb. 8 Bezug genommen.

Der Automat wird mit Hilfe der definierenden Wörter `AUTOMAT:`, `CANAL` und `STATE:` aus AOP.SCR programmiert. Die Definitionswörter, durch die die einzelnen Elemente des Automaten kreiert wurden, sind in Abb. 8 zusätzlich mit angegeben (grau unterlegt). Ein Automat besteht, wie könnte es in Forth auch schon anders sein, aus einer Reihe von Wörtern.

Aktueller Zustand

Das zentrale Element des Automaten ist eine Speicherstelle, die den jeweils aktuellen Zustand des Automaten enthält. Die Adresse dieser Speicherstelle wird bei Aufruf des Forthwortes, das den Automatennamen darstellt, auf den Stack gelegt; in unserem Beispiel das Wort `MENUE`. Das Verhalten des Automatennamens ist also äquivalent mit dem Verhalten einer Forthvariablen. Die Speicherstelle der Zustandsvariablen wird daher auch als "Identifizierungstoken", als *die* signifikante Adresse des Automaten angesehen; auch wenn er, wie zu sehen ist, aus wesentlich mehr Elementen besteht.

Der Inhalt der Zustandsvariablen spezifiziert den Automatenzustand dadurch, daß er auf ein Array von Codefeldadressen (normalerweise das Parameterfeld eines State-Wortes) zeigt, die das Verhalten der Canal-Wörter des Automaten festlegen.

Im Prinzip könnte auch die Zustandsvariable des Automaten von "außen" direkt manipuliert werden. Jedoch ist dies nur in Ausnahmefällen sinnvoll und notwendig. Eine sinnvolle Anwendung wäre z. B. dann gegeben, wenn der aktuelle Zustand gesichert werden muß, um ihn zu einem späteren Zeitpunkt wieder restaurieren zu können, d. h. ihn wieder in den ursprünglichen Zustand zu versetzen.

Kanäle

Dem Automaten zugeordnet sind seine Kanäle. Das sind die Worte, über die mit dem Automaten kommuniziert werden kann. Sie stellen eine Schnittstelle zum Automaten dar und entsprechen damit den Methoden der objektorientierten Programmierung. In Abb. 8 sind das die Worte `DO_LEFT` und `DO_RIGHT`. Da beide Wörter ohne Parameterübergabe fungieren, also quasi immer nur ein Bit übertragen, heißt das, daß auf den dargestellten Automaten nur zwei Ereignisse (Events) eintreffen können. In das Vokabular des theoretischen Automaten übersetzt bedeutet das, daß das ***Eingabealphabet*** des Automaten nur zwei Zeichen umfaßt.

Das Parameterfeld eines Kanal-Wortes besteht aus zwei Parametern. An der ersten Position befindet sich der Offset des Kanals. Durch ihn unterscheidet sich der Kanal von allen anderen Kanälen, die zu diesem Automaten definiert sind. An zweiter Position des Parameterfeldes befindet sich die Adresse des Automaten, zu dem dieser Kanal gehört. Mit Hilfe dieser beiden Parameter berechnet der `DOES>`-Teil eines Kanal-Wortes letztlich die Speicherstelle, in der die Codefeldadresse des Wortes steht, das in Abhängigkeit des Automatenzustandes zur Ausführung kommen soll. In Abb. 8 ist dieser virtuelle Weg für `DO_RIGHT` im Automatenzustand "Eins" durch eine Strichpunktlinie aufgezeichnet.

Zur Namensgebung des Begriffes "Kanal" möchte ich noch einige Bemerkungen machen: Wenn ich statt des Begriffes "Kanal" den Begriff "Methode" gewählt hätte, würden die Leser, die sich in der OOP auskennen, die Funktion und

```
Screen # 0
 0 AOP.scr   Automatenorientierte Programmierung   \ Symds 23.11.94
 1           fuer Atari-volksFORTH 3.80
 2
 3 History:
 4 23.11.94  Ersterstellung
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen # 1
 0                                                 \ Symds 17.11.94
 1 ! : (s_entry    ( pfa_of_state -- )
 2       dup 2+ under swap @ !
 3              perform    ;
 4
 5 : s_entry  ( cfa_of_state -- )  >body (s_entry  ;
 6
 7 : a_exit   ( addr_of_automat -- )    @ 2+ perform   ;
 8
 9
10 : Automat:     ( -- addr_of_automat offset )
11                ( -- addr_of_automat )
12    here 2 allot  dup  Constant  4  ;
13
14   ' 2drop  Alias  ;automat   ( addr_of_automat offset_+_n -- )
15                                                 -->
```

```
Screen # 2
 0                                                 \ Symds 16.11.94
 1
 2 : Canal       ( addr_of_aut offset -- addr_of_aut offset_+_2 )
 3      Create
 4          dup  ,
 5          2+
 6
 7          over ,
 8
 9      Does>
10          dup  @
11          swap 2+ @ @ +
12          perform
13
14        ;
15                      -->
```

```
Screen # 3
 0                                                 \ Symds 18.11.94
 1 : State:    ( addr_of_automat -- $efde )
 2      Create
 3         ,
 4         hide  current @ context ! ]
 5
 6         $efde
 7      Does>
 8         dup @ a_exit
 9             (s_entry
10    ;
11
12
13 : ;state   $efde ?pairs  [compile] [   reveal ;
14                                    immediate restrict
15                                                 -->
```

```
Screen # 6
```

```
Screen # 7
                                                   \ Symds 16.11.94
Zustandsvariable auf neuen Zustand setzen...
...und ENTRY des neuen Zustandes ausfuehren

Initialisiere den (ersten) Zustand

Fuehre das S_EXIT des momentanen Zustandes des Automaten aus,
dessen Adresse auf dem Stack uebergeben wird.

Stackverhalten während der Compile-Time
Stackverhalten des definierten Wortes


;AUTOMAT macht nur den Datenstack wieder sauber
```

```
Screen # 8
                                                   \ Symds 14.11.94


Word-Header erzeugen
Ablegen des (Zaehl-)Offsets...       (So dass jede CANAL:-De-    )
 ...und weiterzaehlen des Offsets    (finition eines Automaten   )
                                     (ihren eigenen Offset erhaelt)
Ablegen von ADDR_OF_AUTOMAT, also der Adresse der Zustands-
 variablen, zu der diese CANAL:-Definition gehoert.

Lese deinen eigenen Offset...
 ...und addiere den Inhalt deiner Zustandsvariablen hinzu...
 ...diese Speicherstelle spezifiziert das Wort, das eigentlich
    ausgefuehrt werden soll.
```

```
Screen # 9
                                                   \ Symds 15.11.94

Word-Header erzeugen
Die Adresse des Automaten ablegen, zu dem dieser Zustand gehoert
Ein wenig Forth-Verwaltungsarbeit, weil hier eine (Quasi-)Colon-
Definition eingeleitet wird... (siehe also Definition von ':' )
..nur die Magic-Number ist nicht 0 sondern (willkuerlich) $EFDE

EXIT des alten Zustandes wird ausgefuehrt...
...und der neue Zustand wird aktiviert.


;STATE entspricht im wesentlichen dem ';' nur dass kein UNNEST
kompiliert wird.
```

Abbildung 3 - AOP.SCR

```
Screen # 0
 0 LINK.scr     Erweiterung des AOP                  \ Symds 23.11.94
 1              fuer Atari-volksFORTH 3.80
 2 History:
 3 23.11.94   Ersterstellung
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen # 1
 0                                                   \ Symds 16.11.94
 1 : Link    ( -- )
 2    here     0 ,   0 ,    8 allot
 3    Constant  immediate  ;
 4
 5
 6   \needs Assembler                -->
 7
 8 : !Link    ( -- )                   \ !LINK kann wegen >LABEL
 9    8 hallot                         \ nur geladen werden, wenn
10    2 hallot    heap  off            \ der Assembler vorhanden ist.
11    2 hallot    heap  off
12    heap  >label  Does>  @    ;
13
14                                    -->
15
```

```
Screen # 2
 0                                                   \ Symds 17.11.94
 1 ! : pointer_switch   ( addr -- )      dup @  4 xor  swap !  ;
 2
 3
 4 : fw  ( addr_of_link -- )
 5   here             over dup @ + 6 +     !
 6   compile noop
 7
 8   last @  name>    over dup @ + 4+      !
 9
10   pointer_switch
11
12  ;  immediate restrict
13
14                                                      -->
15
```

```
Screen # 3
 0                                                   \ Symds 17.11.94
 1
 2
 3
 4 : bw  ( addr_of_link -- )
 5   2+
 6   dup  dup @ + 2+  @  ,
 7
 8   last @  name>    over dup @ + 4+  @  !
 9
10   pointer_switch
11
12  ;  immediate restrict
13
14
15
```

```
Screen # 4
```

```
Screen # 5

LINK legt eine Spezialvariable bzw. Datenstruktur an:
  FW-Pointer:=0  BW-Pointer:=0  Registerpaar0  Registerpaar1

!LINK ist funktionsgleich mit LINK, nur dass alles auf den Heap
      kommt.
```

```
Screen # 6
                                                    \ Symds 12.11.94
Hilfsdefinition (Toggled Inhalt von ADDR von 0 <-> 4 )

Merke dir HERE in dem Link-Wort,
    dann kompiliere vorerst ein NOOP - spaeter wird an dieser
    Stelle der Aufruf des Ziel-Zustandes nachgepatched.
Merke dir das Wort, das gerade compiliert wird, es wird als
    Sprungziel benoetigt, wenn BW aufgerufen wird.
Lege die naechsten Informationen in das andere Registerpaar
    ab, das sich in dem Link-Wort befindet.
FW muss waehrend der Compile-Time ausgefuehrt werden und macht
    auch nur Sinn, wenn es zur Compile-Time ausgefuehrt wird.
```

```
Screen # 7


Arbeite mit dem BW-Pointer
Compiliere das Wort, dass in dem Link-Wort gespeichert wurde.

Trage das momentan in der Definition befindliche Zustandswort
    in die Adresse eine, in die FW ein NOOP kompiliert hat.
Lese die naechsten Informationen aus dem anderen Registerpaar,
    das sich in dem Link-Wort befindet.
```

Abbildung 4 - LINK.SCR

```
Screen # 0
 0 APPLIK_1.scr                                    \ Symds 23.11.94
 1           Anwendungsbeispiel von AOP.scr
 2           ==============================
 3 Problemstellung siehe VIERTE DIMENSION 3/1993 Seite 23-27
 4
 5 Autor:     Michael Symonds
 6           Brandenbaumer Landstr. 48, 23564 Luebeck, Germany
 7 History:
 8 23.11.94  Ersterstellung
 9
10
11
12
13
14
15
```

```
Screen # 1
 0 \  Loadscreen                                   \ Symds 16.11.94
 1
 2
 3     \needs  Automat:   include  aop.scr
 4     \needs  Link       include  link.scr
 5
 6
 7
 8
 9
10
11                                        -->
12
13
14
15
```

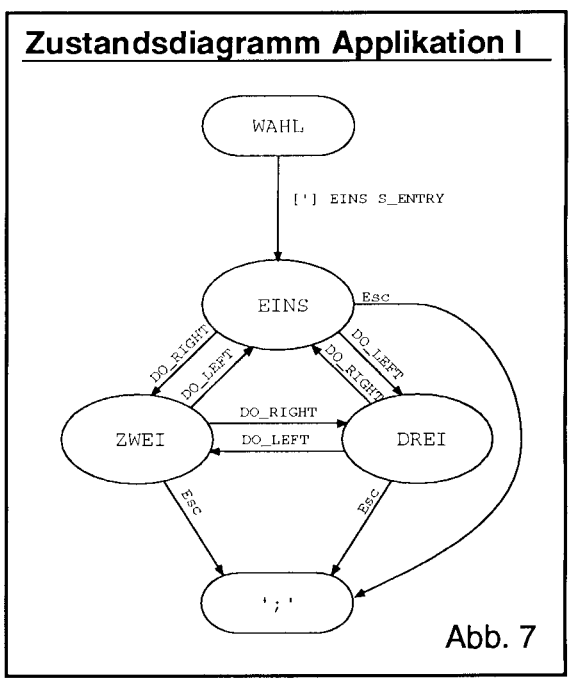
```
Screen # 2
 0 \ Symds 16.11.94
 1 Automat: menue   Canal  do_left
 2                  Canal  do_right     ;automat
 3
 4 : .eins   ( -- )  0 0 at   ." Definition EINS (A)"  ;
 5 : .zwei   ( -- )  0 0 at   ." Definition ZWEI (B)"  ;
 6 : .drei   ( -- )  0 0 at   ." Definition DREI (C)"  ;
 7
 8  Link lnk0   Link lnk1
 9
10                  \  Entry   Exit    do_left   do_right
11 menue State: eins    .eins   noop    lnk0 fw   lnk1 fw   ;state
12 menue State: zwei    .zwei   noop    lnk1 bw   lnk1 fw   ;state
13 menue State: drei    .drei   noop    lnk1 bw   lnk0 bw   ;state
14                                  -->
15
```

```
Screen # 3
 0 \ Symds 15.11.94
 1 $011b Constant escape_key
 2 $4B00 Constant left_key
 3 $4D00 Constant right_key
 4
 5 : wahl      ( -- )   page
 6    ['] eins s_entry          \ Der Anfangszustand ist EINS
 7      BEGIN   key
 8              left_key    case?  IF  do_left            ELSE
 9              right_key   case?  IF  do_right           ELSE
10              escape_key    =    IF  exit
11                                   THEN THEN THEN
12      REPEAT  ;
13
14 : applikation ( -- )    ( .... )    wahl    ( .... )    ;
15
```

Abbildung 5 - Applikation I

```
Screen # 1
 0                                                 \ Symds 23.11.94
 1
 2 \ Zum besseren Verstaendnis hier die State-Definitionen
 3 \ der Applikation I, wenn keine expliziten Vorwaertsreferenzen
 4 \ noetig waeren:
 5
 6                  \  Entry   Exit    do_left   do_right
 7 menue State: eins    .eins   noop    drei      zwei      ;state
 8 menue State: zwei    .zwei   noop    eins      drei      ;state
 9 menue State: drei    .drei   noop    zwei      eins      ;state
10
11
12
13
14
15
```

Abbildung 6 - Applikation I ohne Vorwärtsreferenzen



Zustandsdiagramm Applikation I

den Einsatzzweck der Kanal-Wörter unmittelbar erkennen. Auch der Begriff "Event" hätte seine Berechtigung, denn es handelt sich um die Deklaration eines möglichen Ereignisses, das auf den Automaten einwirken kann. Aber aus den Erfahrungswerten, die ich bei der Benutzung von AOP bei anderen Anwendungen gemacht habe, kann ich sagen, daß erst der Name CANAL die notwendige Allgemeingültigkeit besitzt. (Die Namensgebung von Bezeichnern ist eines meiner liebsten Steckenpferde.) Man betrachte also das, was CANAL deklariert als jeweils einen Zugriffspfad, eben als einen Kanal, der zum Automaten hinführt und/oder von ihm kommt. Was denn nun auf diesen Kanälen stattfindet, ob also Events übertragen werden, ob Methoden aufgerufen oder Abfragen gemacht werden, das sei der Problematik des Einzelfalles und der Intuition des Programmierers überlassen.

Zustände

In Abb. 8 sind die Wörter EINS, ZWEI und DREI die möglichen Zustände des Automaten. (Hier haben wir also das Äquivalent zur ***Menge der Zustände*** des theoretischen Automaten.) Das Parameterfeld eines solchen State-Wortes beginnt mit der Adresse des Automaten, zu dem dieser Zustand gehört. Die Automatenadresse wird für das Run-Time-Verhalten des State-Wortes benötigt. Unter anderen Aktionen, die es ausführt, trägt es bei seiner Ausführung einen Zei-

Aufbau eines AOP-Automaten (Beispiel gem‰fl Applikation I)

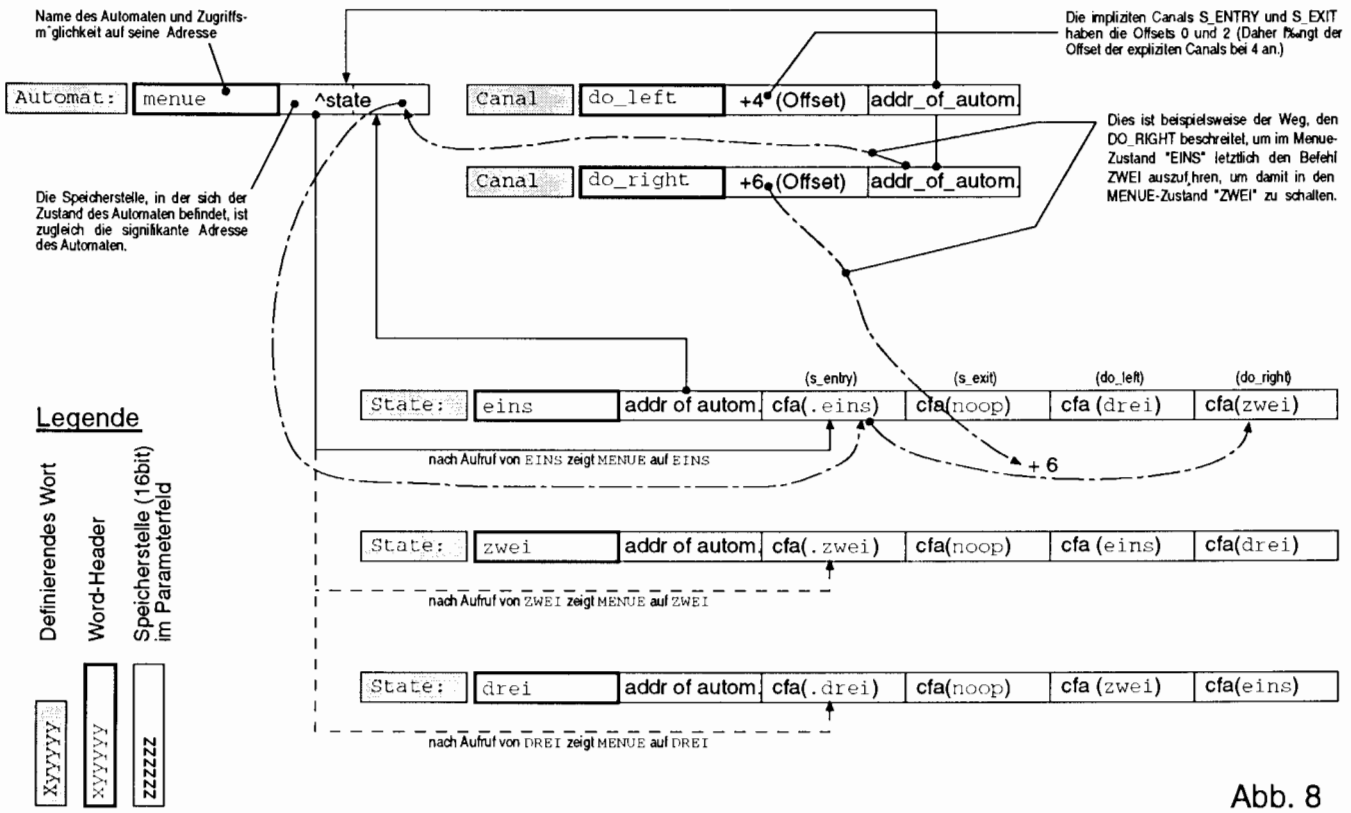


Abb. 8

Aufbau eines AOP-Automaten (Beispiel gemäß Applikation II)

```
Screen # 0
 0 APPLIK_2.scr                                    \ Symds 23.11.94
 1            Anwendungsbeispiel von AOP.scr
 2            ==============================
 3 Demo fuer einen Artikel in der VIERTEN DIMENSIOn
 4
 5 Autor:     Michael Symonds
 6            Brandenbaumer Landstr. 48, 23564 Luebeck, Germany
 7 History:
 8 23.11.94   Ersterstellung
 9
10
11
12
13
14
15
```

```
Screen # 1
 0 \  Loadscreen                                   \ Symds 23.11.94
 1
 2
 3     \needs  Automat:   include  aop.scr
 4     \needs  Link       include  link.scr
 5
 6
 7
 8           1 +load
 9
10
11
12
13
14
15
```

```
Screen # 2
 0                                                 \ Symds 21.11.94
 1
 2 Automat: lcdisplay
 3              Canal  do_up        Canal  do_down
 4              Canal  do_left      Canal  do_right
 5              Canal  do_return    Canal  do_refresh
 6          ;automat   .
 7
 8
 9 Link  up_link       Link  down_link
10 Link  left_link     Link  right_link
11
12
13                                              -->
14
15
```

```
Screen # 3
 0                                                 \ Symds 21.11.94
 1 : home  ( -- )      0 0  at  ;
 2 : 1home  ( -- )     1 0  at  ;
 3
 4
 5 : .root  ( -- )    home  ." >->      ROOT-     <-<"
 6                    1home  ." >->      MENUE     <-<"      ;
 7
 8
 9 lcdisplay State: root_menue
10    .root           noop                .      \ Entry    Exit
11    up_link fw      down_link fw             \ Up       Down
12    left_link fw    right_link fw            \ Left     Right
13    noop            noop                      \ Return   Refresh
14  ;state
15                                                  -->
```

Abbildung 9a - Applikation II praxisnah

```
Screen # 4
 0                                                \ Symds 15.11.94
 1 $011b Constant escape_key
 2 $4800 Constant up_key           $5000 Constant down_key
 3 $4B00 Constant left_key         $4D00 Constant right_key
 4 $1c0d Constant return_key
 5
 6
 7 : ?execute_key    ( key -- )
 8     up_key       case?  IF  do_up       exit   THEN
 9     down_key     case?  IF  do_down     exit   THEN
10     left_key     case?  IF  do_left     exit   THEN
11     right_key    case?  IF  do_right    exit   THEN
12     return_key     =    IF  do_return          THEN
13   ;
14                                                        -->
15
```

```
Screen # 5
 0                                                \ Symds 21.11.94
 1 : wahl      ( -- )
 2    page
 3    ['] root_menue s_entry
 4    BEGIN
 5       BEGIN  key? 0=  WHILE  do_refresh  REPEAT
 6       key
 7       dup ?execute_key
 8           escape_key  =
 9    UNTIL
10  \ menue a_exit   Bei dieser Applikation kann A_EXIT entfallen
11    &2 0 at ;
12
13 \ ***      Hier ist der 'Betriebssystemteil' beendet       ***
14 \ ************************************************************
15                                                          -->
```

```
Screen # 6
 0 \ ******************************************  \ Symds 21.11.94
 1 \ ***   Ab jetzt nur noch 'Applikation'     ***
 2
 3 : .menue_0  ( -- )  home    ."        RAM-View     ->"
 4                     1home   ." <-    Refresh-Demo    "   ;
 5
 6 Link  llink_1    Link  rlink_1
 7
 8 lcdisplay State: menue_0
 9    .menue_0        noop                    \ Entry   Exit
10    down_link bw    down_link fw            \ Up      Down
11    llink_1 fw      rlink_1 fw              \ Left    Right
12    root_menue      noop                    \ Return  Refresh
13  ;state
14                                                        -->
15
```

```
Screen # 7
 0   Variable rcounter                            \ Symds 21.11.94
 1
 2 : .run_header  ( -- )
 3       home      ." Refresh-Demo          "
 4      1home      ."              Exit -> "    ;
 5
 6 : run_refresh  ( -- )   1 rcounter +!
 7     1home   rcounter @  6 .r    ;
 8
 9 lcdisplay State: refresh_run
10    .run_header    noop                    \ Entry   Exit
11    noop           noop                    \ Up      Down
12    noop           llink_1 bw              \ Left    Right
13    root_menue     run_refresh             \ Return  Refresh
14  ;state
15                                                        -->
```

```
Screen # 8
 0  Variable pointer                          \ Symds 21.11.94
 1
 2 : .ram_view    ( -- )   base push  hex
 3    home   pointer @  5 .r  ."  :  "  pointer @  c@ 2 .r
 4    9 spaces     1home  ."   <ENTER> =  Exit   "   ;
 5
 6 : up     ( -- )     -1  pointer +!  .ram_view  ;
 7 : down   ( -- )      1  pointer +!  .ram_view  ;
 8 : left   ( -- ) -$100  pointer +!  .ram_view  ;
 9 : right  ( -- )  $100  pointer +!  .ram_view  ;
10
11 lcdisplay State: ram_view
12    .ram_view      noop                    \ Entry   Exit
13    up             down                    \ Up      Down
14    left           right                   \ Left    Right
15    rlink_1 bw     noop     ;state         \ Return  Refresh
```

Abbildung 9b - Applikation II praxisnah

ger auf sich selbst in die Zustandsvariable des Automaten ein. Das hört sich komplizierter an als es ist - der Effekt ist einfach: Ein Automat wird dadurch in einen Zustand geschaltet, indem der Name des Zustandes aufgerufen wird. Nach Ausführung des Wortes `EINS` ist der Automat im Zustand "Eins" - nach Aufruf von ZWEI ist der Automat im Zustand ...?... na?, wer hat's kapiert?

Die weiteren Werte im Parameterfeld eines Zustandes definieren das Verhalten des Automaten in diesem Zustand - das Äquivalent zu den ***Funktionsregeln*** des theoretischen Automaten. Insofern die darin spezifizierten Aktionsworte eine externe Wirkung haben, z. B. eine Ausgabe auf das LCD-Display, entsprechen diese Aktionen dem ***Ausgabealphabet*** des theoretischen Automaten. (Auch wenn bei dieser etwas freizügigen Interpretation so manch ein theoretischer Informatiker die Hände über dem Kopf zusammenschlagen würde.)

Die ersten beiden Aktionen, die zu jedem Zustand spezifiziert sind (S_ENTRY und S_EXIT) haben eine spezielle Aufgabe. Sie legen das Verhalten für zwei Ereignisse fest, die bei jedem Zustand auftreten: Das erste Ereignis ist das Eintreten in den Zustand und das zweite das Verlassen desselben. Häufig, wie auch in diesem Fall, ist in dem S_ENTRY-Feld die eigentliche externe Reaktion des Automaten festgelegt. Das S_EXIT wird sehr selten benötigt; in Regel nur dann, wenn ein externer Systemzustand wieder restauriert werden muß. Z. B. kann eine Speicherallokierung in S_ENTRY durch S_EXIT wieder freigegeben werden.

Der Zustandswechsel des Automaten, bzw. das komplette Run-Time-Verhalten eines State-Wortes, erfolgt immer nach folgendem Schema:

1) Aufruf des S_EXIT des momentanen (alten) Zustandes
2) Automat in den neuen Zustand schalten
3) Aufruf des S_ENTRY des neuen Zustandes

Die auf S_ENTRY und S_EXIT folgenden Werte legen die Aktionen des Automaten fest, mit denen er im jeweiligen Zustand auf dessen Kanal-Worte, also auf die eigentlichen Eingabeereignisse, reagieren soll. In unserem Beispiel sind das die Umschaltungen in die jeweiligen Folgezustände. Beispiel: Im Zustand "Eins" reagiert der Automat auf `DO_RIGHT` dadurch, daß er in den Zustand "Zwei" schaltet und damit auch den S_ENTRY von `ZWEI` ausführt.

AOP zusammengefaßt

Der Automat der AOP ist vergleichbar mit der Konstruktion des Objektes der OOP. So wie die Methoden der OOP die Schnittstelle zum Objekt bilden, bilden die Kanäle die Schnittstelle zum Automaten. Der wesentliche Unterschied zur OOP besteht darin, daß innerhalb des Automaten, in Abhängigkeit seines globalen Zustandes, den äußeren Kanal-Wörtern ganz verschiedene Wörter zuge-

ordnet sein können. Die Kanal-Wörter sind also eine Gruppe von DEFER-Wörtern, deren Run-Time-Verhalten gemeinsam umgeschaltet werden kann. Ich selber stelle mir diesen Vorgang gerne als eine Kulisse im Innern des Automaten vor, die beliebig verschoben werden kann und die von außen nicht zu bemerken ist.

AOP.scr

`Automat:`
leitet die Definition eines Automaten ein.

`Canal`
deklariert einen Kanal des Automaten

`;automat`
beendet die Definition eines Automaten.

`State:`
definiert einen möglichen Zustand eines Automaten. Damit der Zustand einem konkreten Automaten zugeordnet werden kann, erwartet `STATE:` die Adresse dieses Automaten auf dem Stack. Es können zu jedem Automaten beliebig viele Zustände definiert werden und auch zu einem späteren Zeitpunkt hinzukompiliert werden. Im Falle der Programmierung der LCD-Anzeige, wo jeder Menüpunkt durch einen Zustand des Automaten realisiert ist, bedeutet das, daß die geforderte modulare Programmierung der Applikationssoftware realisiert werden kann.

`;state`
beendet die Definition eines Zustandes des Automaten. Es erfolgt hierbei keine Überprüfung, ob auch zu jedem Kanal des Automaten eine Aktion aufgeführt worden ist.

`s_entry ( cfa_of_state -- )`
ist notwendig, um den Anfangszustand des Automaten während der Run-Time zu initialisieren. Die Initialisierung ist nicht durch das direkte Aufrufen des gewünschten Zustandes möglich. Es würde sonst das S_EXIT des "alten" Zustandes ausgeführt werden, was bei einer Zustandsvariablen, die irgendwo hinzeigt nur in's Nirgendwo führen kann.

`a_exit (addr_of_automat --)`
führt das S_EXIT des momentanen Zustandes des Automaten aus, der auf dem Stack übergeben wurde. Dieses Wort dürfte nur in ganz seltenen Fällen zur Anwendung kommen. Es macht nur dann Sinn, wenn mindestens ein S_EXIT eines Zustandes definiert worden ist und wenn der Automat z. B. bei Austritt aus einer Endlosschleife quasi deaktiviert werden soll.

LINK.SCR

Die Programmierung der LCD-Anzeige bringt noch das Problem der Vorwärtsreferenzierung mit sich. Wenn zwei Menüfelder programmiert werden sollen, wie es dem Standardfall entspricht, daß nämlich von Menü A in Menü B und von B wieder nach A gesprungen werden soll, dann kann man seinen Quelltext drehen und wenden wie man will, es muß immer irgendwo "nach vorne" gesprungen werden. Diesem Problem kann man mit Hilfe von Defer-Worten begegnen. Im ersten Ansatz hatte ich diese Vorgehensweise auch gewählt, aber feststellen müssen, daß durch die Vielzahl der Verbindungen zwischen den Menues schnell unübersichtliche Quelltexte entstehen. Daher habe ich einen speziellen Worttyp entwickelt, die Link-Wörter.

Die Link-Wörter dienen ausschließlich als Compilationshilfe und werden daher nur während der Compilationsphase aufgerufen und verwendet; während der Run-Time haben sie keinerlei Bedeutung mehr. Wenn also auf Speicherplatzersparnis Wert gelegt wird, können die Link-Wörter komplett, d. h. inkl. Header und Parameterfeld, auf den Heap gelegt werden. Dazu existiert das definierende Wort "`|LINK`", mit dem insbesondere alle Link-Wörter definiert werden können, die nur innerhalb eines Moduls Verwendung finden.

Zur Benutzung der Link-Wörter gehören die Wörter `FW` (Forward) und `BW` (Backward). Sie werden innerhalb einer Zustandsdefinition zusammen mit einem Link-Wort aufgerufen. Jede FW/BW-Kombination stellt die Verbindung zwischen zwei Menues her.

Die Funktionsweise von `FW` ist wie folgt: Es kompiliert vorerst ein `NOOP`, so daß es später durch den Aufruf des korrekten Folgezustandes wieder überschrieben werden kann. Damit dies später von `BW` getan werden kann, legt es die Speicherstelle, in der das `NOOP` eingetragen wurde, im Link-Wort ab. (Link-Wörter sind im Grunde genommen Spezialvariablen.) Außerdem legt `FW` das Zustandswort in dem Link-Wort ab, in dessen Definition es aufgerufen wurde.

Das korrespondierende `BW` kompiliert das Zustandswort, das im Link-Wort abgelegt wurde. Und es überschreibt die durch das `NOOP` belegte Speicherstelle mit dem Zustandswort, in dessen Definitionsphase das `BW` aufgerufen wurde. Damit ist die Verbindung zwischen den beiden Menues in beiden Richtungen komplett.

Applikation II

Eine etwas praxisnähere Anwendung, als sie Applikation I darstellt, möchte ich mit Abb. 9, dem zweiten Anwendungsbeispiel, vorstellen. Sie entspricht in ihren Grundzügen schon weitgehend der tatsächlich programmierten Anwendung, wie sie eingangs beschrieben worden ist. Trotzdem kann auch hier nur von einem Schulbeispiel die Rede sein, denn "richtige" Software hat immer eine Vielzahl von Nebenbedingungen zu erfüllen, bei deren Realisierung das eigentlich Wesentliche in den Hintergrund tritt. Die reale Anwendung läuft z. B. innerhalb einer Task unter Umlenkung des Ausgabekanals dieser Task. Der Refresh der LCD-Anzeige wird nur relativ sparsam einmal pro Sekunde ausgeführt, wofür eine entsprechende Timerfunktion benötigt wird. Diese Spezifika und noch einige mehr, habe ich zur besseren Übersichtlichkeit weggelassen.

Andere AOP-Varianten

Die in AOP.SCR realisierte Variante einer "Automatenorientierten Programmierung" ist bei weitem nicht die einzige Möglichkeit, diese Programmiertechnik anzuwenden. Ausgehend von meinen eigenen subjektiven Erfahrungen und Problemstellungen, die ich tagtäglich in meiner Arbeit erfahre, habe ich mich aber bemüht, hier eine Variante vorzustellen, die ein möglichst breites Anwendungsgebiet hat.

Eine grundsätzlich andere interessante Variante wäre gegeben, wenn die Zustandsvariable herausfaktoriert werden würde. Mit den Wörtern `CANAL` und `STATE:` würde nicht nur ein Automat definiert werden, sondern eine ganze Klasse von Automaten. Ich werde diese

Variante im folgenden Klassen-AOP (K-AOP) nennen; im Gegensatz zur Einzel-AOP (E-AOP). Anstelle der einen Zustandsvariablen könnte dann z. B. ein Array von Zustandsvariablen treten. Den Canal- und den State-Wörtern müßte jeweils die Speicherstelle eines dieser Automaten auf dem Stack übergeben werden. Obwohl ich noch keine Anwendung für diese Konstruktion hatte, habe ich zu "Forschungszwecken" diese Variante einmal programmiert und dabei eine interessante Beobachtung gemacht: ***Trotz größerer Universalität der zweiten Variante wird AOP.SCR einfacher.*** (Merke: Je einfacher desto besser.)

Die K-AOP hat eine potentiell größere Mächtigkeit, d. h. es können alle Applikationen der E-AOP programmiert werden und darüber hinaus eben auch Anwendungen mit mehreren Automaten einer Klasse. Trotzdem wird der Umfang der Basisdefinitionen (AOP.SCR) geringer. Im K-AOP wird im Kern quasi weniger getan und damit weniger festgelegt. Allerdings müssen diese Festlegungen dann in der Applikationssoftware erfolgen, was die Programmierung einer E-AOP-Applikation aufgrund einer K-AOP aufwendiger gestaltet, als wenn von vornherein eine E-AOP-Basis verwendet wird. Da ich, wie gesagt, in meiner Praxis noch keine Anwendung einer K-AOP hatte, war dies auch der Grund, warum ich mich für die E-AOP als den Standard entschieden habe.

Alles schon mal dagewesen

Die Bildung einer Gruppe von Defer-Worten ist nichts grundsätzlich neues in der Forthwelt. In dem volksFORTH, das ich seit einigen Jahren mit großem Vergnügen benutze, können die Gruppen der OUTPUT-Wörter (`EMIT`, `TYPE` ...) und der INPUT-Wörter (`KEY?`, `KEY` ...) mit Hilfe dieser Technik auf verschiedene Ausgabekanäle umgeleitet werden. Und in der Tat war dies einer meiner wertvollsten Ideenlieferanten. Allerdings habe ich in dem Fall der LCD-Programmierung den Spieß gewissermaßen um 180° gedreht. Bei den OUTPUT / INPUT - Wörtern ruft die Anwendungssoftware die Defer-Wörter auf und der "Betriebssystemteil" kümmert sich darum, was sich tatsächlich hinter diesen Wörtern befindet. In dem Fall der LCD-Steuerung ruft der Betriebssystemteil die Defer-Wörter auf und die Anwendungsebene spezifiziert, was daraufhin tatsächlich zu geschehen hat.

In einem gewissen Sinne ist durch die Implementation der AOP die vorgefundene Technik zu einem ganzen Konzept weiterentwickelt worden. Die AOP stellt unter anderem eine Verallgemeinerung dieser Technik dar - eine Defaktorierung der universell anwendbaren Methode. Die I/O-Umleitung des volksFORTH könnte nun auch auf Basis der AOP programmiert werden.

... zum Schluß

Als sich mir die Aufgabe der Programmierung eines LCD-Menues stellte, war ich nicht gerade ein Anfänger und trotzdem habe ich geschlagene drei Jahre mehr oder weniger intensiv an diesem Problem herumgebrütet, bis mir der erste vernünftige Einfall gekommen ist. Ein weiteres Jahr habe ich gebraucht, um zu verstehen, was ich da eigentlich programmiert hatte. Aber ich habe dabei mehr über das Wesen von Software gelernt, als ich in einer anderen Programmiersprache jemals begriffen hätte. Es ist schon ein eigentümlich' Ding - dies' Forth . . .

Literatur:

[VORWÄRTS93]
Artikel "Vorwärts - und dann kreuz und quer" VD 3/93
[BITERFINDUNG]
Artikel "Die Neuentwicklung des Bits" in diesem Heft.

NEU: TDS9092 ControllerBoard

Seit 1994 April bietet Triangle Digital Services, London nun den neuen embedded Forth Controller TDS9092 an. Es ist die modernisierte und preisgünstigere Version des bewährten TDS9090 und der 'kleine Bruder' des in Vierte Dimension 1994 Q4 vorgestellten schnellen 16bit Controllers TDS2020.

Die Auslegung der Hard- und Software ermöglicht eine schnelle und flexible Lösung über ein breites Anwendungsspektrum. Das vollständige Forth 'on board' und die damit verbundene Interaktivität während der Entwicklung, beim Service und -- wenn gewünscht -- auch in der Anwendung heben dieses Board von den üblichen Target-Lösungen ab.

Der TDS9092 ist mit einem Maskenprogrammierten 8bit Controller Hitachi HD6301 ausgestattet. Das 16 kB große interne Forth enthält Assembler, Interrupt Service per Assembler und Forth, ferner Worte für LCD und Keyboard, Timer (Auflösung 814 ns), I²C-Bus, WatchDog, LowPowerMode, vectored serial I/O. Dazu kommen bis zu 29 kB Anwender-spezifische Forth.

Erweiterungen im (gepufferten SRAM oder) PROM, 16 kB RAM für Variablen und Datenfelder, 256 Bytes (... 8 kB) EEPROM und insgesamt 35 In/Out.

Erwähnenswert ist der 3 mA standby Mode, und zwar bei sofortiger Tastenreaktion und weiterhin verfügbarer Uhr. Auf den seriellen Port reagiert das System mit nur max. 100 ms Verzögerung.

Weiter Daten: Versorgung 6...16V bei 3...30 mA, 1 oder 2 full-featured RS232, BatterLow Output, geschaltete +5 V, +9 V / -9 V Ausgänge, 'Intel-' als auch 'Motorola-style' Peripherie.

Die Startkosten betragen £150 für Hard- und Software (plus Versand, plus VAT/MwSt).

Arndt Klingelnberg

Computer, die man nicht sieht: Eingebettete Systeme und Echtzeit-Datenverarbeitung

von Ralf Kern

Auch wenn man es oft nicht ahnt, spielen im Betrieb von fast jedem modernen technischen Gerät Mikroprozessoren mit. Sind Vorgänge zu automatisieren oder optimal zu steuern oder ist auch nur die Bedienung eines Geräts zu vereinfachen, werden die "intelligenten Winzlinge" zusammen mit anderen integrierten Schaltungen eingesetzt. Die dabei eingesetzten Komponenten unterscheiden sich trotz oberflächlicher Ähnlichkeit teilweise gravierend von denen, die in handelsüblichen PCs oder Workstations verwendet werden, und es gibt auch beträchtliche Unterschiede in der Aufgabenstellung, die sich sowohl in der Verschaltung als auch in der Programmierung dieser Bausteine bemerkbar machen. Die Programmiersprache Forth spielt bei eingebetteten Systemen wegen ihrer hohen Code-Dichte und ihrer guten Anpassungsfähigkeit an spezielle Umgebungen eine besondere Rolle.

Beispielsweise übernimmt ein Mikroprozessor in einer Waschmaschine die Aufgaben, die Wassereinlaßventile und die Abflußpumpen zu steuern, mit Temperaturfühlern den Aufheizvorgang zu überwachen und den Trommelmotor regelmäßig zwischen Rechts- und Links-Lauf hin- und herzuschalten. Darüber hinaus muß er zum richtigen Zeitpunkt die Spül- und Waschmittelzugabe veranlassen. Das Zusammenspiel dieser Aktionen kann dann von der Wäschesorte, dem Waschmittel, der Waschphase (Vor- und Hauptwaschgang, Schleuderphase) und dem Härtegrad des Wassers abhängig gemacht werden.

Als zweites Beispiel mag die Einspritz- und Zündungssteuerung eines Benzinmotors herhalten. In modernen Autos der oberen Leistungsklasse ist es heutzutage üblich, diese Steuerung und andere Funktionen von einem Mikrocomputer durchführen zu lassen.

Die Mikroprozessor-Steuerung erfaßt über Sensoren (Meßfühler) Größen wie die angesaugte Luftmenge, Drehzahl, Kolbenstellung, Motor- und Ansaugluft-temperatur, Drosselklappenstellung, Batteriespannung und die Stellung einiger Schalter (z. B. zum Starten und zur Anwahl von Leerlauf- und Vollastbetrieb). Aus diesen Eingabewerten ermittelt das Steuergerät laufend die optimalen Werte von Zünd- und Schließwinkel sowie die Kraftstoffmenge. Zur Zeitersparnis werden die Ausgabegrößen nicht berechnet, sondern Kennlinienfeldern entnommen, die in einem Festwertspeicher, (ROM, Read-Only-Memory) gespeichert sind. Mittels dieser Ausgabewerte werden u. a. Zündspule, Einspritzventile und Elektrokraftstoffpumpe als sogenannte Aktoren (Stellglieder) gesteuert.

Kennlinienfelder im ROM

Auf diese Weise kann der Motor immer im optimalen Betriebsbereich gehalten werden, wodurch die Leistung maximiert und Kraftstoffverbrauch sowie Abgaswerte minimiert werden können, und zwar bis zu einem Grad, den die Steuerung durch einen menschlichen Fahrer nicht annähernd erreichen könnte.

Das Steuersystem besteht aus einem Mikroprozessor, einem ROM zur Aufnahme des Steuerprogramms und der Kennfelddaten, einem RAM (Random Access Memory) als Zwischenspeicher von Meßwerten, Rechenergebnissen usw., und schließlich einem Ein/Ausgabe-Schaltkreis zur Erfassung und Wandlung der analogen Meßwerte in digitale Größen sowie zur Ausgabe der wiederum analogen Steuergrößen.

Bei einem Sechszylindermotor und einer Drehzahl von 6666 U/min ist alle 3 ms eine Zündung durchzuführen. Die tatsächlich benötigte zeitliche Auflösung des Gesamtsystems muß allerdings viel höher sein, da z. B. der Zündzeitpunkt (Zündwinkel) nur um einen geringfügigen Teil einer Umdrehung variiert werden darf. Allerdings betrifft diese hohe zeitliche Auflösung des Gesamtsystems (in der Größenordnung von einigen µs) nicht unbedingt auch die Reaktionszeit des Mikroprozessor-Systems, da die CPU zur Ausgabe Analogkomponenten über Digital-/Analog-Wandler ansteuert und diesen nur die Betriebsparameter vorgibt. Die Analogkomponenten lösen dann z. B. die Zündung gemäß dieser Parameter zu den erforderlichen Zeitpunkten aus. Für die CPU genügt es, wenn sie etwa Drehzahl-Änderungen rechtzeitig erkennt und darauf durch Änderung der Betriebsparameter reagiert. Damit läßt sich vor allem auch der CPU-Takt von der ständig wechselnden Drehzahl entkoppeln.

Einbettung und Echtzeit

Diese Beispiele verdeutlichen, daß für derartige technische Geräte und die darin eingebauten Mikrocomputer zweierlei charakteristisch ist:

- Die eingebauten Mikrocomputer treten nach außen gar nicht wie übliche Computer (also wie z. B. PCs oder Workstations mit Tastatur und Bildschirm, Disketten-Laufwerk usw.) in Erscheinung. Oftmals merkt der Benutzer gar nicht, daß er es mit einen Mikrocomputer zu tun hat, der zwischen ihm und dem technischen Gerät vermittelt. Man spricht daher von *eingebetteten Systemen.*
- Die Berechnungsvorgänge in den eingebetteten Mikrocomputern sind zeit-

bezogen. Sie stehen in engem zeitlichem Zusammenhang zu Vorgängen in der "Außenwelt", also im umgebenden technischen System. Man bezeichnet dies als *Echtzeit-Datenverabeitung*.

Natürlich darf man bei eingebetteten Systemen nicht nur an hochwertige Konsumgüter denken: so gehören Weltraumraketen zu den ersten technischen Systemen, die eingebaute Computer enthielten, und viele andere Geräte der Luftfahrt- und Rüstungstechnik zählen ebenfalls dazu; darüber hinaus auch Roboter oder numerisch gesteuerte Werkzeugmaschinen.

Eingebettete Systeme sind von der Hardware her durch eine Reihe von Unterschieden zu konventionellen Computersystemen gekennzeichnet:

- Sie werden normalerweise nicht über computer-übliche Peripheriegeräte (Tastatur, Bildschirm) bedient, sondern stattdessen, sofern überhaupt nötig, mittels Schaltern, Einzeltasten, Drehknöpfe, Leuchtdioden, Flüssigkristall-Anzeigen (LCDs) usw. Da die meisten Daten mit dem umgebenden technischen System ausgetauscht werden, bestehen dorthin die wichtigsten Ein/Ausgabe-Schnittstellen in Form von Sensoren und Aktoren.
- Da die üblichen Bediengeräte nicht vorhanden sind, ist für Diagnosezwecke manchmal eine Service-Schnittstelle (Stecker für ein Analyse- oder Programmiergerät oder einen externen Computer) vorgesehen. Vorteilhaft ist es, wenn das System sich in Fehlerfällen mittels "eingebauter Intelligenz" selbst testen und Fehler diagnostizieren kann.
- Magnetische Massenspeicher (Diskette, Festplatte) sind nicht vorhanden; stattdessen ist die Software in einem nichtflüchtigen Festwertspeicher (PROM = Programmable Read-Only-Memory) abgelegt oder "gepromt". Alternativ dazu wird in bestimmten Fällen (sogenannte speicherprogrammierbare Steuerungen) die Software aus einem Programmiergerät (das ist im Prinzip ein spezialisierter PC) in das ausführende Rechnersystem geladen.
- Oftmals unterliegen eingebettete Systeme besonderen Sicherheitsanforderungen, da Eingriff oder Alarmierung des Bedieners bei Systemstörungen nicht möglich ist. Oder der Ausfall des Systems kann katastrophale Folgen (z. B. Flugzeugabsturz) haben. Hier behilft man sich häufig mit verschiedenen Formen der Redundanz, also des mehrfachen Einsatzes derselben Komponenten zur Steigerung der Ausfallsicherheit (und nicht der Leistungsfähigkeit).

Diese Merkmale, aber auch Unterschiede in der Einsatzweise beeinflussen die Programmierung dieser Systeme erheblich. In der klassischen Datenverarbeitung früherer Jahre herrschte die sogenannte Stapelverarbeitung (batch processing) vor. Sie hat ihren Namen davon bekommen, daß die Programmierer ihre Rechenaufträge in Form von Lochkartenstapeln im Rechenzentrum abgaben und die Ergebnisse nach einigen Stunden wieder abholen konnten.

Heute arbeitet man zwar nicht mehr mit Lochkarten, aber diese Betriebsform ist durchaus noch verbreitet. Bei ihr kommt es nicht darauf an, ob die Abarbeitung des Programms eine Stunde früher oder später stattfindet; insbesondere ist die Richtigkeit der Ergebnisse nicht vom Zeitpunkt des Zustandekommens abhängig. Stattdessen hat der Rechenvorgang keinerlei Bezug zum Zeitablauf in der Außenwelt.

Schritt halten

Ganz andere Verhältnisse herrschen da bei der Echtzeit-Datenverarbeitung. Die Rechenvorgänge im Computer müssen mit den Vorgängen in der Außenwelt Schritt halten können. Dazu genügt es eben nicht, daß die Ergebnisse richtig berechnet werden, sondern sie müssen auch noch rechtzeitig abgeliefert werden, damit etwa das Wassereinlaßventil in der Waschmaschine vor dem Überlauf geschlossen oder der Zündzeitpunkt des Motors rechtzeitig angepaßt werden kann. Um es an einem anderen, drastischen Beispiel zu illustrieren: Es reicht nicht, wenn eine Mikroprozessor-Steuerung Überdruck in einem Dampfkessel diagnostiziert; der Überdruck muß außerdem rechtzeitig festgestellt werden, bevor der Kessel explodiert.

Man unterscheidet daher im Echtzeit-Betrieb zwischen Werte- und Zeit-Korrektheit der Ergebnisse. In eingebetteten Systemen ergeben sich durch die enge Kopplung mit dem umgebenden technischen System automatisch Echtzeit-Anforderungen an die Mikrocomputersteuerung.

Eine gewisse Mittelstellung zwischen Stapelverarbeitungs- und Echtzeitsystemen nehmen Dialogsysteme oder interaktive Systeme ein. Zwar stehen auch sie in einem engen zeitlichen Zusammenhang zur Außenwelt, aber da der Interaktionspartner kein technisches System, sondern ein (hoffentlich geduldiger) Mensch von vergleichsweise langsamer Reaktion ist, dem man zur Not auch einige Augenblicke Wartezeit zumuten kann, brauchen Dialogsysteme nicht so strengen Zeitanforderungen zu genügen wie ("harte") Echtzeit-Systeme im eigentlichen Sinn. Im englischen Sprachgebrauch werden Dialogsysteme daher auch als "weiche" Echtzeit-Systeme bezeichnet.

Die Echtzeit-Anforderungen bedingen besondere programmiertechnische Verfahren, um die Rechtzeitigkeit, also die Zeit-Korrektheit der Ergebnisse zu gewährleisten. So bekommen besonders wichtige Ereignisse (z. B. drohender Überlauf in der Waschmaschine) bei der rechnerinternen Behandlung Vorrang vor anderen, weniger zeitkritischen Aktionen (etwa Abschalten der Wasserheizung oder Richtungsumkehr des Trommelmotors).

Rechtzeitigkeit und Gleichzeitigkeit

Es ist typisch für die Echtzeit-Datenverarbeitung, daß für die vorrangigen Ereignisse bei der Reaktion des Mikroprozessor-Systems konkrete zeitliche Schranken, die aus der technischen Aufgabenstellung hergeleitet werden, eingehalten werden müssen. Wie die Beispiele zeigen, können diese Schranken ziemlich weit gefaßt sein (z. B. einige Minuten für die Richtungsumkehr der Waschmaschine), in anderen Fällen aber auch sehr knapp, bis in den Mikrosekundenbereich hinein. Dies trifft vor allem dann zu, wenn aus dem technischen System in großer Menge Daten einlaufen und in Echtzeit erfaßt werden müssen.

Da außerdem meistens mehrere miteinander verflochtene Vorgänge der Außenwelt gleichzeitig zu überwachen

und zu steuern sind (z. B. in der Waschmaschine der Aufheizvorgang und der Rechts-Links-Lauf der Trommel sowie der richtige Zeitpunkt der Waschmittelzugabe), werden die Vorgänge in der Realität durch mehrere parallele Rechnerprozesse (Tasks) modelliert, die nur an bestimmten Punkten ihres Ablaufs aufeinander warten oder untereinander Daten austauschen und ansonsten zeitlich völlig unabhängig voneinander ablaufen.

Eine besondere Rolle in der Echtzeit-Datenverarbeitung spielen die Echtzeit-Betriebssysteme, die dem Programmierer durch spezielle Dienste die Arbeit wesentlich erleichtern können. Sie müssen mehrprozeßfähig (multitaskingfähig) sein, leistungsfähige Dienste zur schnellen Bearbeitung von Unterbrechungen (Interrupts) anbieten, die Einteilung der Prozesse in Prioritäten (Vorrangstufen) sowie die Kommunikation zwischen den Prozessen ermöglichen. Natürlich wird bei allen Betriebssystem-Diensten Geschwindigkeit groß geschrieben, doch Geschwindigkeit allein macht noch kein Echtzeit-Betriebssystem aus, sondern es muß schon geeignete Strukturen aufweisen und die benötigten Dienste anbieten.

Echtzeit-Betriebssysteme, die speziell in eingebetteten Systemen eingesetzt werden, werden häufig auf einen Minimalumfang abgemagert, weil Funktionen, die man bei konventionellen Betriebssystemen gewöhnlich antrifft, hier entfallen. Da keine Disketten oder Festplatten verwendet werden, lassen sich nicht nur die zugehörigen Treiber, sondern auch die gesamte Komponente zur Verwaltung des Dateisystems einsparen. Ebenso brauchen keine Terminaltreiber, die häufig viel Aufwand verursachen, integriert zu werden.

Knauserei mit Speicherplatz

Man spricht bei solchen Minimal-Betriebssystemen, die selten eine Größe von mehr als ca. 10 KByte benötigen, von Echtzeit-Kernen oder -Monitoren bzw. von real-time executives. Obwohl ihr Leistungsumfang gewöhnlich nur die Task- oder Prozeßverwaltung sowie die Interprozeßkommunikation und -synchronisation umfaßt, bieten sie den Systementwicklern doch wertvolle Hilfestellung bei der Strukturierung des Anwendungssystems. Darüber hinaus müssen sie durch komfortable Schnittstellen das Einbringen von Treibern für die systemspezifischen Aktoren und Sensoren erleichtern. Die Programmierung solcher sehr spezialisierter Treiber ist normalerweise nämlich Aufgabe der Systementwickler, da sie nicht standardmäßig zum Lieferumfang des Betriebssystems gehören.

Bei der Echtzeit-Anwendungsprogrammierung herrschen weniger die komfortablen als die effizienteren Sprachen vor. C ist verbreitet, und die Sprache Forth spielt speziell bei eingebetteten Systemen eine besondere Rolle, denn die Erweiterbarkeit von Forth erlaubt die flexible Anpassung an die unterschiedlichsten Sensoren und Aktoren. Aus Geschwindigkeitsgründen und wegen der besonderen Hardware-Nähe wird allerdings noch überdurchschnittlich viel in Assemblersprachen programmiert, obwohl in der sonstigen Informatik ein klarer Trend zu höheren und damit komfortableren Programmiersprachen zu beobachten ist. Darüber hinaus gibt es auch ausgesprochene Echtzeit-Sprachen wie etwa PEARL, die dem Programmierer echtzeitbezogene Sprachmittel vor allem zur Prozeßsynchronisierung bieten. Solche Dienste müssen sonst vom Betriebssystem angefordert werden.

Während man in konventionellen Computern heutzutage mit Hauptspeicher megabyteweise klotzt, bildet vor allem in Massenkonsumgütern jeder überflüssige Speicherchip einen nicht vernachlässigbaren Kostenfaktor. Außerdem müssen eingebettete Systeme häufig mit sehr wenig Platz und bei Batteriebetrieb mit geringer Leistung auskommen, und auch da stört jeder entbehrliche Chip. Der geringe Speicherbedarf für System- wie Anwendungssoftware ist also für eingebettete Systeme lebenswichtig, und auch da kann Forth wegen der hohen Code-Dichte seine Stärken ausspielen.

❑

Windows-Applikationen und sicherere Echtzeitsysteme

eine Chance für Forth durch autonome Controller

Die Mär, daß Echtzeitanforderungen nicht zusammen mit WINDOWS genutzt werden könnten, hat sich überholt, zumindest wenn die Meßlatte nicht gar zu streng angesetzt wird. Allerdings würde ich in keinem Fall ein kritisches Echtzeitsystem (soetwas auf 'Leben und Tod' einem PC, geschweige denn einem PC unter WINDOWS überlassen).

Windows NT 3.5 und OS/2 v.≥2.1 erlauben zwischen den Programmen echtes preemptives Multitasking, darüber hinaus sind auch in ein 'normales' WINDOWS Echtzeitmodule integrierbar. Die Workshops auf der MeßComp 1993 und 1994 (September, Wiesbaden) zeigten jedoch, daß Echtzeit unter Windows keinesfalls trivial ist. So nutzen einige der vorgestellten Lösungen PC-externe(!) Controller, andere solche innerhalb des PC's auf Steckkarten, um den RealTime-Part zu übernehmen. Und hier speziell liegt die Chance für Forth. Forth war schon immer eine Domäne für 'embedded control'. Es bietet sich an externe Controller über eine normale serielle RS 232 oder über eine multidrop RS 485 oder sogar über ein Feldbus-Protokoll an einen PC anzubinden, die Controller mehr oder weniger zu synchronisieren und die Echtzeitprozesse unabhängig vom PC und damit auf einem höheren Sicherheitsniveau laufen zu lassen. Solche externen Controller können über weite Strecken autark operieren. Sie sind auch leichter über Accu zu puffern. Sie könnten äußerst schnell wieder starten (watchdog) und sind sehr viel einfacher kontrollierbar gegenüber EMV- Abstrahlung und/oder Einstreuung.

Das Debuggen der Einzelprozesse bzw. Einzelcontroller kann als deutlich unkristischer eingestuft werden, insbesondere wenn es Forth-Systeme sind. Ein Gesamtsystem auf einem PC bzw. einer Workstation sind degegen äußerst komplex und weniger robust.

Windowsapplikationen (oder andere GUIs) sind 'schön' und verwirren immer wieder die Geldgeber. (Über den wirklichen Nutzeffekt kann gestritten werden). Sie können aber vorteilhaft die nicht in Echzeit notwendigen Prozesse wie Visualisierung und Parametereingabe übernehmen.

Arndt Klingelnberg

Paralleles Garbage Collecting in objektorientierten Softwaresystemen

von Birgit Steffenhagen und Malte Köller

In objektorientierten Softwaresystemen besteht die Notwendigkeit des dynamischen Erzeugens und Freigebens von Objekten. Um den Programmierer von der Verwaltung des dynamischen Heaps bzw. konkret der expliziten Freigabe von dynamischen Objekten zu entlasten, was eine nicht zu unterschätzende Fehlerquelle darstellt, ist es lohnenswert, einen automatisierten garbage collection - Mechanismus zu verwenden.
Solch ein Mechanismus wird an Hand eines objektorientierten Experimentalsystems basierend auf einem gekoppelten Reference-counting und Mark & Sweep - Verfahren vorgestellt.

1 Motivation

Seit ungefähr zwei Jahren arbeitet im Institut Automatisierungstechnik der Universität Rostock eine Arbeitsgruppe an der Entwicklung von Algorithmen für die echtzeitfähige Implementierung objektorientierter Laufzeitsysteme. Das Attribut echtzeitfähig steht hier für den anvisierten Einsatzbereich solcher Systeme, die Prozeßautomatisierung und Prozeßmeßtechnik, wobei die Rolle objektorientierter Techniken in diesem Bereich komplexeren Applikationen wie z. B. intelligenten Regelungssystemen [1], Meßvorverarbeitung oder sogar echtzeitfähigen objektorientierten Betriebssystemen [2] vorbehalten sein wird. Als Implementierungsprache des Prototyps eines solchen Systems wurde die Programmiersprache Forth gewählt, wegen ihrer hardwarenahen und vor allem interaktiven Programmiermethodik. Forth ist eine in ihren Compilerstrukturen einfach zu erweiternde Programmiersprache und am Institut standen mehrjährige Erfahrungen mit Forth sowie ein eigens am Institut entwickeltes Forthsystem zur Verfügung. Also bot es geeignete Voraussetzungen für unser Vorhaben.

Ein besonderer Schwerpunkt bei der Entwicklung des Systems lag in der optionalen Hardwareimplementierung spezieller Teilsysteme der Laufzeitumgebung, wie z. B. die dynamische Speicherverwaltung oder ein Modul für die späte Bindung von Methoden an Objekte. Durch eine derartige Modularisierung können die, den objektorientierten Systemen eigenen Effizienzverluste, mittels Hardwarecodierung der Algorithmen kompensiert werden.

Im folgenden soll nun eine dynamische Speicherverwaltung vorgestellt werden, wie sie im Prototyp zum Einsatz kommt. Deren Algorithmen und Schnittstellen zum objektorientierten Laufzeitsystem wurden so ausgelegt, daß sie ohne weiteres gegen eine effizientere Hardwareimplementierung ersetzt werden kann. Hier sollen vor allem die Verfahren des, wegen der Echtzeitanforderungen parallelen, Garbage Collecting (automatisiertes Auffinden und Zurückgeben von verlorenem Speicherplatz) und die durch die Nebenläufigkeit bedingten Besonderheiten näher erläutert werden. Ein zweiter Artikel in diesem Heft wird Sie mit der objektorientierten Arbeitsweise vertraut machen.

2 Überblick über Garbage Collection - Mechanismen

Bei der Implementierung von für die Echtzeitverarbeitung geeigneten Garbage-Collecting-Mechanismen sind eine Reihe von Besonderheiten zu beachten. Auf diese soll im Folgenden näher eingegangen werden. Anschließend werden grundlegende GC-Verfahren im Hinblick auf die Echtzeitverarbeitung betrachtet.

Für Echtzeitapplikationen scheinen fein gegliederte inkrementelle GC-Verfahren notwendig zu sein. Das Garbage Collecting kann hier nicht als zusammenhängende Aktion ausgeführt werden, während das Programm unterbrochen wird. Es müssen sich kleine Einheiten des Garbage-Collectors und des ausführenden Programms überlappen. Dafür ist eine anspruchsvolle Kooperation zwischen Anwendungsprogramm und GC erforderlich, um die Konsistenz der Objektreferenzen sicherzustellen.

Collectoren, nach dem noch zu erläuternden Referenz-Counting-Verfahren, lassen sich sehr einfach inkrementell realisieren. Es gibt dort aber Probleme mit der Effizienz und zyklischen Referenzen. Deshalb ist es wünschenswert, auch kopierende oder markierende Collectoren inkrementell zu gestalten. Dabei treten folgende Schwierigkeiten auf, die durch den Umstand bedingt sind, daß sich der Graph der erreichbaren Objekte ändert während er vom Collector durchsucht wird. Deshalb müssen dem GC Veränderungen am Graphen mitgeteilt werden, um eventuell Aktionen des GC's rückgängig zu machen.

Da Sicherheit oberstes Gebot ist, kann es aber auch passieren, daß unter Umständen ein Objekt, welches innerhalb eines Zyklusses zu "Müll" wird, während des aktuellen Zyklus nicht als solcher erkannt wird. Es wird dann im nächsten Zyklus zurückgegeben.

Zum besseren Verständnis inkrementeller Collectoren wurde das Drei-Farbenmodell [3] eingeführt. Hier wird der GC als Prozeß des Durchsuchens des Graphen der erreichbaren Objekte und Einfärben dieser beschrieben. Objekte, die während eines Zyklusses noch nicht vom GC erreicht wurden sind weiß. Alle Objekte, die am Ende eines Zyklusses noch weiß sind, werden als freier Spei-

cher zurückgegeben. Solche die am Zyklusende schwarz eingefärbt sind, werden noch benötigt. Zusätzlich wurde eine dritte Farbe grau eingeführt. Sie zeigt an, daß das Objekt selbst erreichbar ist, aber seine Nachkommen noch überprüft werden müssen. Diese Graueinfärbung geschieht ausgehend von den Wurzelobjekten. Sobald die Untersuchung der Zeiger der Nachkommenschaft erfolgte, werden sie zu schwarzen und die Nachkommen zu grauen Objekten. Es darf kein Zeiger von einem schwarzen Objekt direkt zu einem weißen zeigen. Es müssen immer graue Objekte dazwischen liegen. Will das Anwendungsprogramm so eine Verbindung von einem schwarzen zu einem weißen Objekt aufbauen, muß das mit dem Garbage Collector koordiniert werden. Dazu gibt es zwei Basis-Methoden:

- Lese-Barriere
  Erkennt, wenn das Anwendungsprogramm (Mutator) versucht Zugang zu einem Zeiger auf ein weißes Objekt zu bekommen. Dann wird es wieder zu einem grauen Objekt.
- Schreib-Barriere
  Wenn ein Programm versucht Zeiger in ein Objekt zu schreiben, wird dies erkannt und aufgezeichnet.

Ausgehend von diesen Basismethoden zur Koordination von Anwendungsprogrammund GC, dem Drei-Farben-Modell sowie verschiedenen "statischen" GC-Verfahren wurden verschiedene inkrementelle Verfahren entwickelt [3].

Im folgenden sollen nun einige Basis-GC-Verfahren im Hinblick auf ihre "Echtzeiteignung" betrachtet werden.

Reference-counting Verfahren (Referenzenzählen)

Jedes auf dem Heap erzeugte Objekt besitzt einen Referenzzähler, der die Verweise auf ein Objekt verwaltet. Wird einem Zeiger, die Adresse bzw. ein Handle (implementationsabhängig) eines Objektes zugewiesen, muß der Referenzzähler dieses Objektes inkrementiert werden. Falls dieser Zeiger zuvor die Adresse eines anderen Objektes enthielt, wird dessen Referenzzähler dekrementiert. Wird ein Zeiger gelöscht, wird der Referenzzähler des referenzierten Objektes ebenfalls dekrementiert. Diese Verkettung wird bei der Freigabe von Objektspeicher benötigt.

Nimmt ein Refenzzähler den Wert Null an, so wird das entsprechende Objekt automatisch gelöscht, d. h. der belegte Speicherplatz freigegeben. Außerdem muß durch Auswertung der Referenzen dafür gesorgt werden, daß die Referenzzähler aller Objekte, auf die das gelöschte Objekt Referenzen besaß, dekrementiert werden.

Dieses Verfahren ist inkrementeller Natur, d. h. es wird der Speicher sofort freigegeben, wenn er nicht mehr benötigt wird. Das impliziert aber nicht automatisch Echtzeitfähigkeit. Wenn Daten mit variabler Größe verwendet werden, kann es beim Freigegeben großer Datenstrukturen zu nicht vorherbestimmbaren Antwortzeiten kommen. Durch geeignete Modifikationen, z. B. indem große Datenstrukturen "scheibchenweise" freigegeben werden, kann das RC-Verfahren leicht komplett inkrementell implementiert werden. "Komplett inkrementell" sollen hier für ein definiertes Zeitfenster stehen, indem der GC seine Arbeit verrichten kann.

Beim RC-Verfahren kann es aber zu Effizienzproblemen kommen auf Grund des Overheads durch Erhöhen und Erniedrigen des Referenzzählers, vor allem bei Programmen die intensiven Gebrauch von Zeigern machen. Durch Modifikationen dieses Basisverfahrens kann die Effizienz aber erhöht werden. Des weiteren werden zyklische Referenzen nicht erkannt (s. Abb. 1).

Trotz des zusätzlichen Referenzfeldes sind sie speichereffektiv gegenüber z. B. kopierenden Verfahren.

Bei verteilten bzw. vermischten GC-Implementationen kann man das Referenzenzählen auch in Kombination mit anderen Verfahren verwenden. Zwischen den Knoten werden Referenzen gezählt und Mark & Sweep oder kopierende Verfahren können dann innerhalb der Knoten benutzt werden. Dieses Prinzip findet auch beim cooS Verwendung.

Mark & Sweep

Dieses Verfahren ist kein inkrementelles Verfahren. Der GC wird aktiviert, wenn im Heap kein Speicher mehr zur Verfügung steht. Dabei werden zunächst alle existierenden Objekte demarkiert. Danach werden alle Objekte, die über Zeigerbeziehungen von der Wurzel aus erreicht werden können, markiert. Jetzt kann der Speicherplatz aller unmarkierten Objekten zurückgegeben werden z. B. über Einketten in eine oder mehrere freie Listen.

Ein großes Problem bei diesem Verfahren und auch beim Referenzenzählen ist, daß es bei der Verwaltung von Objekten variabler Größe zu einer Fragmentierung des Speichers kommt. Dies wird noch begünstigt, da junge, eher zum "sterben" neigende, und ältere Objekte zusammen stehen. Des weiteren muß der Speicher erst vollständig durchsucht werden, ehe Objekte freigegeben werden können. Dies ist nachteilig für eine inkrementelle Realisierung. Der Aufwand für das Sammeln des "Mülls" ist proportional der Heapgröße.

In der klassischen Realisierung sind sie nicht für die Echzeitverarbeitung geeignet. Sie müssen nach den oben erwähnten Verfahren modifiziert und einer Fragmentierung des Speichers entgegengewirkt werden.

Kopierende Verfahren

Zu dieser Gruppe zählen:

1. Das Mark & Compact Verfahren
   In der ersten Phase werden hier alle erreichbaren Objekte markiert und anschließend alle lebenden Objekte so verschoben bis sie benachbart sind. Dieses Verfahren arbeitet in einem Heap.
   Es beseitigt das Problem der Fragmentierung. Ebenfalls stehen langlebige Objekte beieinander. Es macht aber mehrere Phasen nötig und ist langsamer als das Mark & Sweep-Ver-

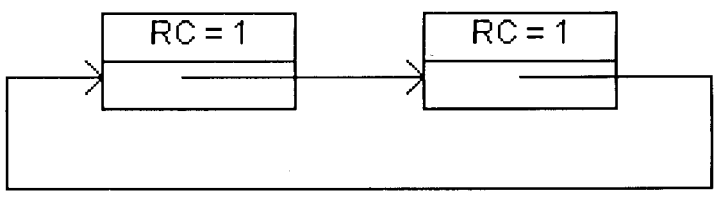


Abb. 1: Zyklische Referenz

fahren. Für Echtzeit-Anwendungen ist dieses Verfahren am wenigsten geeignet, da beim Umkopieren Objektdaten überschrieben werden können. So kann während des Umkopiervorganges nicht auf gültige Objektdaten zugegriffen werden.

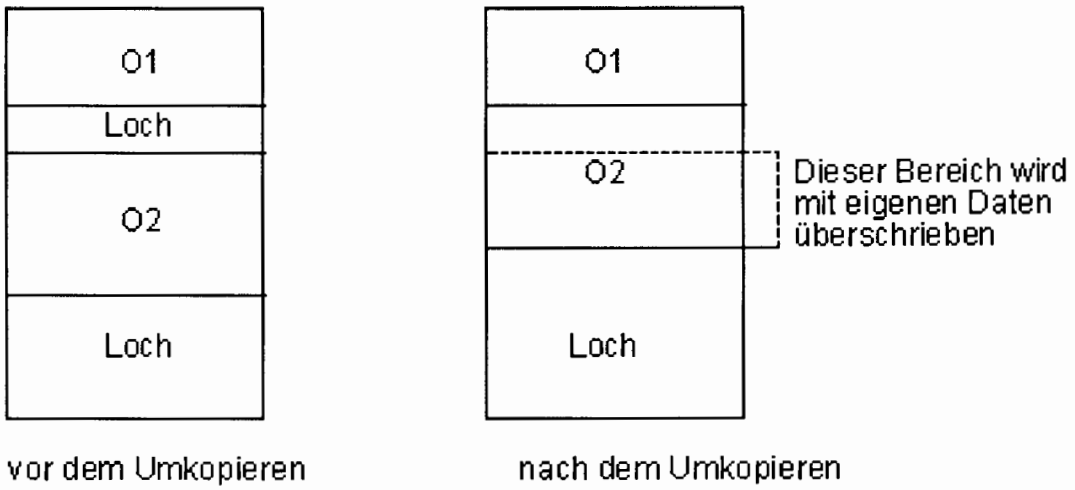


Abb. 2: Mark & Compact Verfahren

2. Kopierendes Garbage Collecting mit zwei Heaps
Der Heap wird hier in zwei aufeinanderfolgende Semiheapbereiche geteilt. Während der normalen Programmausführung wird jeweils ein Semiheap verwendet. Der Speicher wird linear bereitgestellt. Ist nun bei einer Speicheranforderung keiner mehr vorhanden, wird das Programm gestoppt und der kopierende GC wird aktiviert. Alle lebenden Daten werden vom aktiven zum leeren Semiheap kopiert. Danach werden die Arbeits- und Zielhälfte vertauscht, und es können weitere Objekte allokiert werden. So wird einer Fragmentierung des Speichers entgegengewirkt.
Für interaktive und Echtzeitprogrammierung sind solche sekundenlangen Unterbrechungen nicht akzeptabel. Dieses Verfahrensprinzip ist deshalb für inkrementelle Collectoren mit Lese- oder Schreib-Barrieren zu modifizieren. Als ein Basisverfahren fand es u. a. im eooS Verwendung.

3. Garbage-Collecting mit Lebensdauerabschätzung (generation scavenning)
Dieses Verfahren geht davon aus, daß "jüngere" Objekte eher zum Sterben neigen als "ältere". Daher werden die Objekte verschiedenen Generationen zugeteilt. Die älteren werden seltener überprüft als die jüngeren. Dieses Verfahren spart Zeit, da weniger Objekte umkopiert werden müssen.
Diese Basisidee läßt sich noch beliebig vaiieren. So kann man den Heap in zwei Genarationen (alt und neu) unterteilen. Jeder Generationsspeicher wird noch mal in zwei Semiheaps unterteilt. Objekte werden in der neuen Generation allokiert bis der Arbeitsheap voll ist. Dann wird die neue Generation durchsucht. Die lebenden Daten werden in den anderen Semiheap geschrieben. Wenn ein Objekt lange genug lebt, wird es in eine ältere Generation geschrieben. Da nicht viele Objekte alt werden, füllt sich der Semiheap der älteren Generation nicht so schnell. Wenn die ältere Generation voll ist, muß diese ebenfalls durchsucht werden. Die Anzahl der Generationen kann auch größer sein.
Es ist zu beachten, daß alle Zeiger von der alten in die neue Generation zur Scavening-Zeit auffindbar sind und als Wurzelobjekte genutzt werden können. Das sichert, daß erreichbare Objekte der jüngeren Generation gefunden werden und daß Zeiger bewegter Objekte aktualisiert werden können.
Diese GC-Methode ist von Natur aus nicht echtzeitfähig, kann aber durch spezifische Modifikationen mittels einer Indirektionstabelle ähnlich einer Lesebarriere oder durch Einsatz einer Schreibbariere als inkrementelles Verfahren implementiert werden. Ein großer Vorteil ist die Effizienz, da wenige Objekte während eines Zyklusses kopiert werden müssen.

In Hochleistungssystemen werden oft hybride Techniken benutzt, um sich an verschiedene Objektkategorien anzupassen. In Echtzeitsystemen werden immer inkrementelle oder parallele Verfahren zu finden sein. Beim Entwurf solcher inkrementellen Collectoren sollte man immer das drei-Farben-Modell im Hinterkopf behalten. Die Wahl der Basis-GC-Algorithmen und der inkrementellen Modifikationen wird zum einen von der Hardware und zum anderen vom übrigen Systementwurf des objektorientierten Systems abhängen. Hier steht die Frage, wer mit dem Koordinationsaufwand zwischen GC und objektorientierten System "belastet" werden soll. Auf diese Fragen versuchen die folgenden Abschnitte eine Antwort zu geben.

3 Bestandteile des eooS

Das echtzeitfähige, objektorientierte System dient zur Entwicklung prozeßnaher Applikationen. Folgende Forderungen wurden in der Designphase des eooS aufgestellt:

- Erfüllung der Echtzeitforderungen des angekoppelten Prozesses, d. h. determinierte Reaktionszeiten auf externe Ereignisse und effiziente Algorithmen.
- Ermöglichung einer interaktive Arbeitsweise mit extrem kurzen turnaround Zyklen wie in Smalltalk unter Einschluß interaktiver Testmöglichkeiten am realen Prozeß zur effektiven Programmentwicklung und -inbetriebnahme.
- Realisierung einer hybriden Struktur, um die bewährten Vorteile der prozeduralen Programmierung mit den Möglichkeiten der objektorientierten Programmierung zu verbinden.
- Modularisierung des objektorientierten Sprachsystems derart, daß die

optionale Verwendung objektorientierter Coprozessoren, insbesondere Speicherverwaltungseinheiten, unter Beibehaltung der guten Wiederverwendbarkeitseigenschaften der Software ermöglicht wird.

Aus diesen Gründen stellt sich der Aufbau des eooS wie nebenstehend dar.

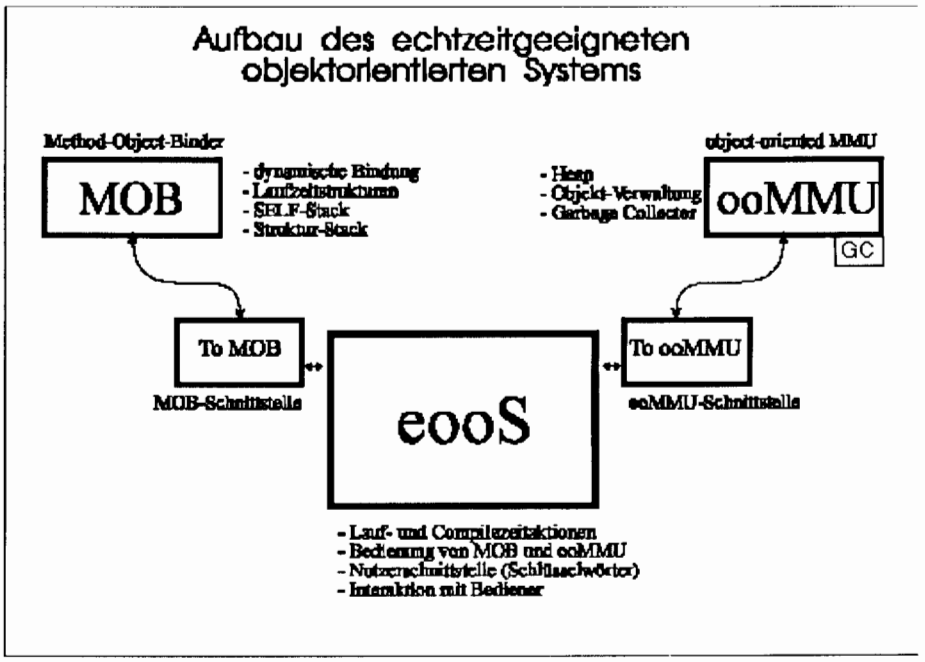


Abb. 3: Aufbau des echtzeitgeeigneten objektorientierten Systems

4 Zusammenspiel vom Laufzeitsystem, Objektverwaltung und Garbage Collektor

Objekte, deren Methoden aufgerufen bzw. die als Parameter übergeben werden sollen, befinden sich auf dem sogenannten Objektstack (als Analogie zum Parameterstack eines konventionellen Forthsystems). Genauergenommen liegen hier nur Deskriptoren, die das entsprechende Objekt identifizieren. Im weiteren werden diese Deskriptoren als Objektreferenzen bezeichnet.

Es werden statische und dynamische Objekte unterschieden. Erste sind während der gesamten Laufzeit eines Programms verfügbar. Zweite werden während der Programmabarbeitung für unbestimmte Zeitdauer erzeugt. Sie werden von der dynamischen Objektverwaltung gehandelt.

Damit der GC Speicherbereiche verschieben kann, ohne daß das Laufzeitsystem darüber in Kenntnis gesetzt werden muß, enthalten Objektreferenzen für die Adressierung des Objektspeichers nur Handle. Diese werden von der Handleverwaltung zur Verfügung gestellt.

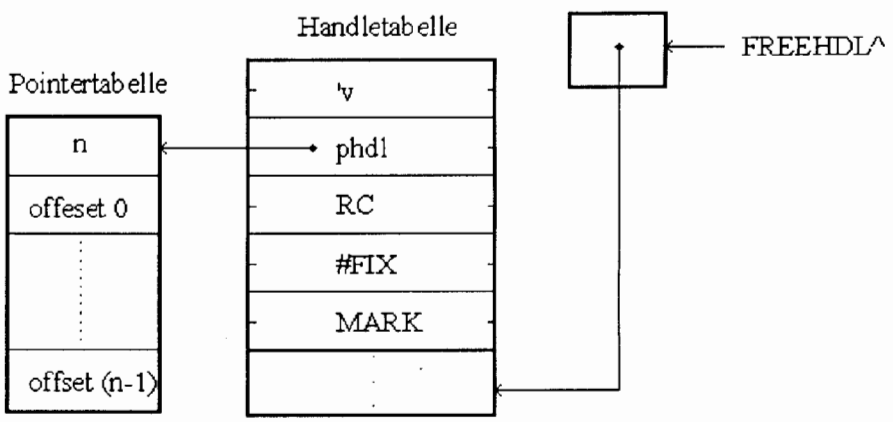


Abb. 4: Aufbau von Handle- und Pointertabelle

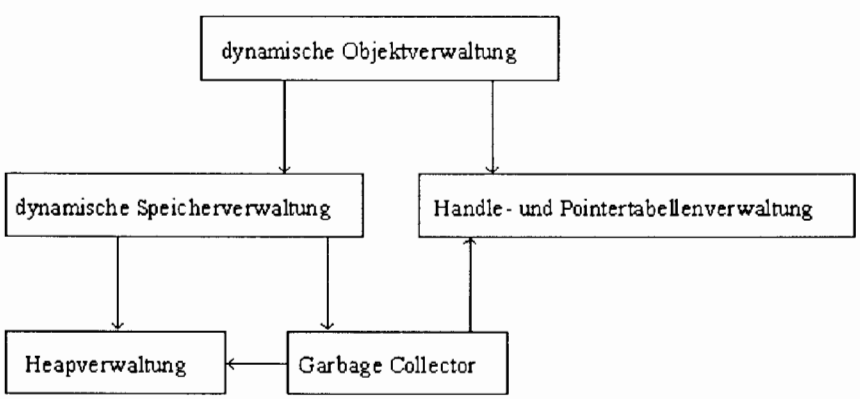


Abb. 5: Aufbau der dynamischen Objektverwaltung

Um dem GC ein Durchmustern des dynamischen Speicherraumes nach lebenden Objekten (entlanghangeln an den Objektreferenzen) zu ermöglichen, hat der GC auf sogenannte Pointertabellen Zugriff (s. Abb. 4). In diesen sind die Positionen von Objektreferenzen im Objektspeicher enthalten. Diese Pointertabellen lassen sich eindeutig aus der Klassenbeschreibung eines Objektes ableiten und werden bei der Definition von Klassen vom Laufzeitsystem an die Objektverwaltung weitergereicht.

5. GC-Prinzip des eooS

Der Garbage Collecting-Mechanismus wird nicht vom System aufgerufen bzw. gesteuert, sondern verrichtet asynchron zur Heapverwaltung seine Arbeit. Das bedingt logischerweise einige Besonderheiten in der Kommunikation zwichen Heap und Laufzeitsystem, damit es nicht zu Kollisionen kommt.

So verändert zum Beispiel das Umsetzen oder Neuhinzufügen von Referenzen in dynamischen Objekten durch das Laufzeitsystem den Graphen der lebenden Objekte. Der Garbage Collector benötigt jedoch einen konsistenten Graphen der Objektreferenzen, um sämtliche lebende Objekte ausgehend von den statisch referenzierten Objekten (Wurzelobjekte) auffinden zu können [4].

Von der Struktur her wird dieses Problem durch den einheitlichen Zugriff auf Objektspeicher über Handle sowie die stackorientierte Arbeitsweise des Systems gelöst. Der Objektstack gehört zum statischen Speicherbereich. Dadurch fungieren sämtliche Objektreferenzen, die im statischen Bereich abgelegt werden, als Wurzelobjekte aus Sicht des Garbage Collectors. Das Ändern von Referenzen in Objekten erfolgt wiederum ausschließlich über entsprechende Methoden, deren Parameter (die jeweiligen Objektreferenzen) auf dem Objektstack übergeben werden. Nach dem Einspeichern der Referenz in ein (natürlich lebendes) Objekt, zählt das referenzierte Objekt zwar nicht mehr zu den Wurzelobjekten, wird aber nun über das lebende Objekt gefunden. Dadurch ist der GC trotz Änderung des Graphen immer in der Lage, sämtliche lebende Objekte aufzufinden.

Der Speicherverwalter des eooS selbst arbeitet nach einem kombinierten Verfahren. Die außerhalb des dynamischen Speichers angelegten Referenzen auf dynamische Objekte werden nach dem Reference-Count-Verfahren behandelt, alle Referenzen innerhalb des dynamischen Speichers werden jedoch nicht gezählt, sondern in der Mark-Phase eines Mark&Copy Algorithmus erkannt.

Ausgangspunkt für die Suche des GC's nach Referenzen sind alle Objekte, auf die extern verwiesen wird, d. h. deren Reference-Count größer Null ist. Weitere Referenzen auf dynamische Objekte werden über die Pointertabellen gefunden. Als benutzt erkannte Objekte werden in einen alternativen Speicherbereich kopiert (Scavening).

Soll aber der Speicherinhalt von dynamischen Objekten geändert werden, kann es passieren, daß der GC gerade beim Umkopieren des entsprechenden Objektes ist. Die Adressierung der entsprechenden Speicherzelle würde fehlschlagen. Aus diesem Grund kann vom Laufzeitsystem für ein Objekt im Heap eine Fixierung angemeldet werden. Dadurch wird dieses Objekt vom Umkopieren ausgeschlossen. Sollte das Objekt während des Umkopierens fixiert werden, wird dieser Vorgang ungültig gemacht, d. h. die alte Objektadresse verbleibt im Handle und der für das Umkopieren angeforderte Speicher wird zurückgegeben. So ist ein ständiger Zugriff auf die Objekte gewährleistet. Praktisch wird die Fixierung für jedes Objekt angewendet, das gerade von einer Methode bearbeitet wird. Natürlich wird man durch diese Vorgehensweise nie eine vollständige Defragmentierung erreichen. Objektorientierte Coprozessoren für die Heap-Verwaltung bzw. für das Garbage-Collecting hingegen brauchen durch ihre spezielle Adreßarithmetik keine Fixierung der gerade bearbeiteten Objekte (siehe auch [5]).

Ein weiteres Problem stellt die Neuerzeugung dynamischer Objekte dar. Dazu wird der entsprechende Speicher von der Heapverwaltung allokiert und eine Objektreferenz auf dem Stapel abgelegt. Wäre das neue Objekt zu diesem Zeitpunkt dem GC bekannt, obwohl es keine gültigen Daten enthält, könnte der Graph der Objektreferenzen inkonsistent werden. Deshalb nehmen dynamische Objekte im Heap nach der Erzeugung einen als ungültig markierten Zustand ein. So werden sie nicht vom GC angefaßt und beeinflussen damit auch nicht den Graphen der Objektreferenzen. Erst nach ihrer vollständigen Initialisierung über die entsprechenden Methoden, wird das Objekt in den gültigen Zustand überführt und wird damit auch für den GC sichtbar.

Die Objekte werden in der schon beschriebenen Handletabelle verwaltet. Für den Garbage Collector ist weiterhin die Arbeitstabelle erforderlich. Sie dient zur Speicherung der Objekte, die als notwendig erkannt wurden, aber noch nicht auf weitere Referenzen durchmustert wurden. Das entspricht den grauen Objekten.

Um die Arbeitsschritte des GC zu verstehen, sollen im folgenden alle Zustände eines Objektes, die die Handleverwaltung auch erkennt, aufgezählt werden:

- nicht existent
- ungültig, d. h. auch es ist extern (statisch) referenziert
  In so ein Objekt dürfen nur NIL-Pointer eingetragen werden. Diesen Zustand besitzt ein Objekt unmittelbar nach der Erzeugung
- gültig und extern referenziert
  Es existiert mindestens eine statische Referenz auf dieses Objekt und sein Referenzzähler ist größer als Null.
- gültig und intern (dynamisch) referenziert
  In einem dynamischen Objekt ist ein Zeiger auf dieses Objekt vermerkt.
- gültig und sowohl extern als auch intern referenziert
- gültig und weder extern noch intern referenziert
  Es existiert weder ein statischer noch ein dynamischer Verweis auf dieses Objekt. Auf solche Objekte können später keine Referenzen mehr aufgebaut werden. Sie müssen vom GC als Müll erkannt werden und in den nicht existenten Zustand überführt werden.

Während der Garbage Collector läuft, soll aber weiterhin nach Anforderung durch das System Speicher durch die OOMMU in Echtzeit bereitgestellt werden. Deshalb muß der Garbage Collector im Hintergrund arbeiten und unterbrechbar sein. In der Prototypvariante wurde dies durch eine Timerinterruptsteuerung realisiert.

Arbeitsschritte des Garbage Collectors

Der Zustand des Garbage Collector wird in einer Variablen festgehalten. Damit die Objektverwaltung der ooMMU diesen auslesen und beachten kann.

1. Alle Objekte in der Handletabelle werden markiert.
2. Alle Objekte, deren Referenzzähler größer als Null ist und die gültig sind, werden in die Arbeitstabelle eingetragen und demarkiert. Alle Objekte, die ungültig sind, werden nur demarkiert.

3. Nun wird die Arbeitstabelle schrittweise bearbeitet:
   Es wird ein Objekt entnommen. Dann wird nach dynamischen Referenzen dieses Objektes auf andere gesucht. Wird so eine Referenz gefunden, wird dieses Objekt, wenn es nicht demarkiert ist, demarkiert und in die Arbeitsliste eingetragen. Ist dieses Objekt schon demarkiert, wird nichts getan. Das Bearbeiten der Arbeitsliste erfolgt solange bis die Arbeitsliste leer ist. Falls der Garbage Collector Zeiger auf statische Objekte findet, verfolgt er diesen Pfad nicht weiter.
4. Ergebnisse:
   Alle noch benötigten Objekte sind demarkiert und alle nicht mehr erforderlichen Objekte (der Müll) sind markiert.
5. Markierten Müll zurückgeben und die demarkierten Objekte umspeichern:
   Alle markierten Objekte werden gelöscht, d. h. das Handle und der Speicher werden intern zurückgegeben.
   Alle demarkierten Objekte werden in die Zielhälfte kopiert. Dieses Umspeichern ist aber nur unter bestimmten Bedingungen möglich:
   - Das betreffende Objekt muß gültig sein.
   - Es darf nicht fixiert sein, d. h. es darf keine Methode über ihm arbeiten.
   - Das Objekt muß sich in der Arbeitshälfte befinden. Dies ist nicht immer gegeben, da es bei früheren Umspeicherschritten fixiert gewesen sein konnte und nicht in der aktuellen Arbeitshälfte stehen muß. Des weiteren muß ein Umkopiervorgang rückgängig gemacht werden, wenn das betreffende Objekt während dieses Vorganges fixiert wurde.

Parallelarbeit des GC - Time Slice- Verfahren

Da das System in einer reinen Softwarelösung ohne objektorientierte Coprozessoren auskommen muß, steht für die Abarbeitung des Programms und die Arbeit des Garbage Collectors nur ein Prozessor zur Verfügung. Die Echtzeitforderung des Gesamtsystems bedingt jedoch eine parallele Arbeit beider Prozesse, weshalb ein Time-Slice-Verfahren zum Einsatz kommt. Jeder Prozeß bekommt eine feste Zeitscheibe zugeteilt, in der er abgearbeitet wird. Die Zeitscheibe des GC's ist dabei jedoch um Größenordnungen kleiner als die des Anwenderprozesses, da das Garbage Collecting nur einen kleinen Teil der Gesamtlaufzeit des Programms ausmacht. Die Performance des Systems wird dadurch nicht zu stark beeinträchtigt.

6 Aussichten - weitere Arbeiten

Die Definition der Schnittstelle zwischen ooMMU und Laufzeitsystem steht eindeutig fest. Änderungen in der Struktur der ooMMU bzw. Erweiterungen des Laufzeitsystems haben dadurch keine Konsequenzen für die Kommunikation beider Teilsysteme.

Die ooMMU kann zum Beispiel durch Hinzufügen mehrerer Teilheaps auf ein Generation-Scavenning-Verfahren beim Umkopieren erweitert werden. Dadurch werden länger lebende Objekte vom aufwendigen Umkopieren verschont.

Durch die Unterbrechung des Laufzeitsystems durch den Garbage-Collector wird das gesamte Zeitverhalten des Systems beeinträchtigt. Vorteilhaft ist hier die Einführung eines Heap-Fragmentierungsfaktors, durch den das Time-Sharing zwischen beiden Teilsystemen dynamisch geregelt wird. Bei starker Fragmentierung des Heaps bekommt der GC eine größere Zeitscheibe zugeteilt. Bei keiner oder kleiner Belastung und Fragmentierung kann dagegen der GC-Prozeß ruhen. Dabei muß jedoch für die Berechnung von Reaktionszeiten im System die maximale Zeitscheibe des GC-Prozesses zugrunde gelegt werden.

Zur Zeit wird die ooMMU unter MS-Windows implementiert, da hier die parallele Arbeit des GC sehr gut unterstützt werden kann. Die Echtzeitfähigkeit des Gesamtsystems kann dadurch zwar beeinträchtigt werden, diese Arbeiten dienen jedoch der Verifizierung und Validierung der parallelen Algorithmen.

Weitere Arbeiten konzentrieren sich auf die Portierung des Systems (einschließlich ooMMU und MOB) auf ein abgesetztes Rechnersystem, daß zur Prozeßsteuerung sowie Meßwertverarbeitung eingesetzt wird. Die Bedienung des interaktiven Systems erfolgt dabei von einem Hostrechner aus.

Literaturverzeichnis

[1] Steffenhagen, B.:
Integrierte Softwareumgebung zur Realisierung Intelligenter Regelungssysteme.
Dissertation, Universität Rostock, 1993.

[2] Fiedler, J.:
Unterstützung der objektorientierten Softwareentwicklung unter Berücksichtigung von Echtzeitbetriebssystemen.
Dissertation, RWTH Aachen, 1994.
VDI-Verlag GmbH.

[3] Wilson, P. R.:
Uniprocessor Garbage Collection Techniques.
Proceedings of 1992 International Workshop on Memory Managment.
St. Malo, France, September 1992

[4] Pleßmann K.W.:
Schlußbericht zum Forschungsvorhaben "Konzeption und Überprüfung einer Objekt-orientierten Speicherverwaltungseinheit für echtzeitgeeignete Systemumgebungen.
Lehr- und Forschungsgebiet für Verfahren der Prozeßdatenverarbeitung und Prozeßführung der RWTH Aachen, 1993.

[5] Schrader, M.:
Garbage-Collection: Grundlagen und Implementierungen in C++.
OBJEKTspektrum 4/94, S.78-82.

Objektorientierte Programmierung in der Automatisierungstechnik mit der Option auf zusätzliche Hardwareunterstützung

von Malte Köller
Universität Rostock, Fachbereich Elektrotechnik, Institut für Automatisierungstechnik

Objektorientierte Programmierung wird heute in vielen Bereichen eingesetzt. In der Automatisierungstechnik bzw. allgemein in der hardwarenahen Programmierung sind diese Techniken jedoch selten zu finden, da trotz der großen Vorteile, die sie bieten, auch einige Nachteile zum Tragen kommen. Das sind einmal der um einiges größere Speicherbedarf objektorientierter Programme sowie die dynamische Verwaltung dieses Speichers, zum anderen verschlechtern sich die Laufzeiteigenschaften der Programme durch das dynamische Binden von Methoden. Diese Nachteile mögen ein Grund für die geringe Verbreitung in diesem Bereich sein.
Der Artikel soll Möglichkeiten aufzeigen, wie mit Hilfe von Forth objektorientierte Techniken angewendet werden können und wie eine zusätzliche Hardwareunterstützung möglich ist, um die angesprochenen Nachteile zu kompensieren.

1 Merkmale der objektorientierten Programmierung

1.1 Vererbung

Durch die Vererbung ist es möglich, schon definierte Programmteile (Objekte) wiederzuverwenden. Durch das Vererbungskonzept erhält der Nachkomme eines Objektes sämtliche Eigenschaften und die Funktionalität des geerbten Objektes. Dabei unterscheidet man einfache und mehrfache Vererbung. Bei der einfachen Vererbung entsteht eine Hierarchie unter den Objektklassen, wie man sie zum Beispiel aus der Biologie (Stammbaum) her kennt. Es existiert also für eine Objektklasse nur eine Superklasse. Bei der Mehrfachvererbung hingegen können für eine Objektklasse mehrere Objektklassen als Eltern fungieren, so daß ein nicht eindeutig gegliederter Stammbaum entsteht, sondern ein Graph.

1.2 Kapselung

Da sich ein Objekt von anderen Objekten durch bestimmte Eigenschaften und Verhaltensweisen unterscheidet, kann man es einfach aus der Gesamtheit der Objekte einer Applikation abgrenzen. Um eine gewisse Sicherheit des Zusammenspiels der Objekte untereinander zu gewährleisten, ist es sogar nötig, die Einzel-Objekte als abgegrenzte Einheiten betrachten zu können. Diese Abgrenzung wird als Kapselung bezeichnet. Bei der Definition einer neuen Objektklasse werden deshalb nur die Methoden nach außen hin sichtbar, die als Schnittstelle zum Objekt fungieren sollen. Sämtliche Datenfelder und Hilfsfunktionen sind nach außen hin unsichtbar. Auf sie kann nicht direkt zugegriffen werden. Auf diese Art und Weise erreicht man eine Abstraktionsebene, die noch über der einer streng typisierten prozeduralen Programmiersprache, wie zum Beispiel Pascal, liegt. Die Datenfelder eines Objektes können nur durch das Objekt selbst beeinflußt werden. Dadurch erreicht man eine Einheit zwischen Daten und Programmcode, der diese Daten manipulieren soll und der Programmierer wird von häufigen Fehlerquellen befreit: Die Änderung einer Quelltextstelle hat nicht mehr "zerstörerische" Wirkung im ganzen Programm.

1.3 Polymorphismus

Das Wort Polymorphismus bedeutet soviel wie Vielgestaltigkeit. Diese Eigenschaft objektorientierter Systeme hängt eng mit der Vererbung zusammen. Wird eine neue Objektklasse definiert, erhalten sämtliche Instanzen dieser Klasse die gleichen Methoden. Ihre Datenfelder werden also alle auf die gleiche Art und Weise manipuliert. Nun kommt es jedoch auch vor, daß sich bestimmte Objekte zwar ähneln, aber nicht völlig gleich sind. Für sie muß also eine neue Klasse definiert werden. Wie in 1.1. erläutert, kann man dafür die Vererbung anwenden. Die neue Objektklasse erbt die Eigenschaften seiner Eltern. Nun können entweder neue Eigenschaften hinzugefügt oder schon bestehende Eigenschaften modifiziert werden. Für den zweiten Fall müßten also nur schon existierende Methoden umgeschrieben werden. Diese Methoden werden sozusagen mit neuem Programmcode "überladen". Im Extremfall schreibt man für eine neue Objektklasse nur die bestehenden Methoden um und erhält somit Objekte, die absolut identische Schnittstellen wie ihre Eltern aufweisen, jedoch grundverschieden reagieren.

1.4 Dynamische Instantiierung

Es wurde schon angeführt, daß viele Objekte gleiche Eigenschaften besitzen und deshalb in Klassen zusammengefaßt werden. Soll mit einem Objekt dieser Klasse gearbeitet werden, muß es erzeugt werden. Es wird instantiiert. Dynamische Instantiierung heißt nun, daß ein Objekt erst zur Laufzeit des Programms erzeugt wird. Während der Übersetzungszeit steht also nicht fest, wieviel Objekte zur Laufzeit im System vorhanden sein werden. Außerdem besteht die Möglichkeit, Objekte zur Laufzeit zu entfernen. Bei der Konzeption einer objektorientierten Programmiersprache muß auf diese Eigenschaften geachtet werden. Das betrifft zum Beispiel die Speicherverwaltung sowie die Überwachung des Zugriffs auf nicht mehr existierende Objekte.

1.5 Statische und dynamische Bindung

In einfachen prozeduralen Programmiersprachen tritt durchgängig nur die statische Bindung auf. Das heißt, daß dort, wo im Quelltext ein Prozeduraufruf vorgesehen ist, der Compiler im Programmcode ein Unterprogrammaufruf zu dieser Prozedur einfügt. Dadurch ist eine einfache Syntaxprüfung (Typprüfung) während der Compilationsphase möglich. Komplexere Programmiersprachen wie z. B. **C** oder auch **Turbo-Pascal** bieten jedoch auch die Möglichkeit der dynamischen Bindung. Hier steht zur Übersetzungszeit noch nicht fest, welcher Code aufgerufen wird. Erst zur Laufzeit wird der auszuführende Code mit dem Aufruf gebunden. Dadurch verkompliziert sich die Überprüfung der korrekten Programmierung nicht unerheblich, macht sie manchmal sogar unmöglich.

Im engen Zusammenhang mit der dynamischen Bindung steht der Polymorphismus, der im Kapitel 1.3 beschrieben wurde. Methoden, die für verschiedene Objektklassen überschrieben wurden, können auch ohne Bezug auf eine bestimmte Klasse aufgerufen werden. Dadurch ist zur Übersetzungszeit nicht eindeutig klar, welcher Code zur Ausführung kommt. Erst zur Laufzeit wird der Bezug auf eine spezielle Klasse hergestellt und der entsprechende Code ausgeführt.

2 Vorteile der OOP in Verbindung mit Forth

Durch die starke Modularität der objektorientierten Technik wird die Gesamtapplikation überschaubarer. Auch die Wartung komplexer Programme vereinfacht sich, da Änderungen in Teilkomponenten nur lokal wirken. Durch die Wiederverwendbarkeit und Modifizierbarkeit von Teilkomponenten wird der Entwicklungsprozeß für neue Applikationen verkürzt.

Da Forth eine interaktive Sprache ist, können neue Objektklassen bzw. auch deren Instanzen interaktiv erzeugt bzw. gelöscht werden. Forth gibt sich mit wenig Speicher zufrieden und erlaubt eine effiziente Implementierung von Programmen.

Vereinigt man diese Vorteile, erhält man ein leistungsfähiges Programmiersystem, das gerade im Bereich der hardwarenahen Programmierung mit objektorientierten Techniken eingesetzt werden kann.

3 Beseitigung von Schwachstellen der OOP

Objektorientierte Systeme arbeiten normalerweise mit einer dynamischen Speicherverwaltung, die aus Effizienzgründen nicht immer einfach zu realisieren ist. Außerdem bringt das späte Binden von Methoden Laufzeitprobleme mit sich, die kompensiert werden müssen. Diese Probleme können durch die Einbindung zusätzlicher Hardware beseitigt werden. Dazu wird das objektorientierte Programmiersystem in drei Teilkomponenten untergliedert:

- eine echtzeitfähige Speicherverwaltungseinheit für dynamische Objekte (Hardwareplattform oder Softwareemulation)
- eine echtzeitfähige Komponente zur Verwaltung der Strukturen des Systems sowie für das dynamische Binden der Methoden (Hardwareplattform oder Softwareemulation)
- das interaktive Programmiersystem

4 Implementierung eines echtzeitfähigen OOP-Systems mit optionaler Hardwareunterstützung

Das vorliegende System besteht im wesentlichen aus den in Abschnitt 3 genannten Komponenten:

- ooMMU (**o**bjekt**o**rientierte MMU)
- MOB (**M**ethod-**O**bject-**B**inder)
- eooS (**e**chtzeitfähiges **o**bjekt**o**rientiertes **S**ystem)

ooMMU und MOB wurden zunächst als TSR-Programme (auf Grundlage von comFORTH 2.3) implementiert, die dem eooS eine Reihe von Funktionen zur Verfügung stellen.

Das eooS selbst basiert auf einem comFORTH 3.0-System, das um einen zusätzlichen OOP-Wortschatz erweitert wurde. Inzwischen ist die Portierung auf ein comFORTH für Windows in Arbeit.

4.1 Aufgaben der ooMMU

Die ooMMU dient zur Verwaltung dynamischer Objekte, d. h. Objekte, die erst zur Laufzeit einer Applikation erzeugt und wieder vernichtet werden. Das Vernichten solcher Objekte (also Zurückgeben des Speicherplatzes an die ooMMU) braucht dabei nicht vom Anwender explizit ausgeführt werden, da die Referenzen auf dynamische Objekte von der ooMMU verwaltet werden. Um einer Fragmentierung des dynamischen Speicherraums (Heap) entgegenzuwirken, wurde in der ooMMU ein Garbage-Collector (Speicherbereinigung) implementiert, der nach dem Scavenning-Verfahren (Umkopieren von Objekten) arbeitet. Dazu ist der Heap der ooMMU in zwei Hälften geteilt, in denen noch referenzierte Objekte so umkopiert werden, daß eine Defragmentierung des Heaps erreicht wird. Dieses Verfahren wurde in der Softwaresimulation für die parallele Arbeitsweise modifiziert, um Echtzeitanforderungen gerecht zu werden [siehe auch Artikel über parallele GC-Verfahren in diesem Heft].

4.2 Aufgaben des MOBs

Die hauptsächlichste Arbeit des MOBs besteht darin, zur Laufzeit Objekte und eine aufzurufende Methode so zu binden, daß der richtige Code zur Ausführung gelangt. Da dazu eine ganze Menge Strukturen des eooS notwendig sind, obliegt dem MOB zusätzlich auch noch die Verwaltung dieser Strukturen. Außerdem befinden sich im MOB der SELF-Stack und der Struktur-Stack, da diese vom MOB am meisten beansprucht werden. Der MOB kommuniziert mit dem eooS wie die ooMMU über eine FAR CALL-Schnittstelle mit Funktionsnummern. Die Prozessorregister werden zur Parameterübergabe benutzt. Zusätzlich zu den beiden Stacks besitzt der MOB einen Cache für den obersten Eintrag des Objekt-Stacks, um die Methoden-Bindung zu beschleunigen. Der Vorteil eines solchen Caches ist natürlich nur in einer Hardwarelösung relevant, bei der der Mob asynchron zum eooS die Tabellen umschalten kann, sobald sich der Wert des Caches ändert und sich damit der eigentliche Bindeprozeß auf ein Minimum reduziert. Das Binden der Methoden wird über sogenannte virtuelle Methodentabellen (VMT's) erreicht, mit denen ein echtzeitfähiges Binden unabhängig von der Vererbungshierarchie möglich ist. Außerdem existiert noch eine Strukturtabelle, in der Informationen über die Vererbungshierarchie gespeichert sind. So arbeiten z. B. die Schlüsselwörter `OWNER` und `BUILDER` mit dieser Strukturtabelle. Wird eine Methode nach Vererbung nicht redefiniert, muß irgendwie der neue Offset in den Datenbereich des Teilobjektes bekannt gemacht werden, damit die Methode korrekt auf die Variablen des Teilobjektes zugreifen kann. Das ist Sache des Struktur-Stacks. Auf ihm wird bei jedem Eintritt in eine Methode ein Index in die Strukturtabelle abgelegt, wo Informationen über den aktuellen Datenoffset ins Objekt zu finden sind.

4.3 Das eooS

4.3.1 Schlüsselwörter

Das eooS arbeitet mit zusätzlichen Stacks, dem Objektstack zur Übergabe von Objektpointern als Parameter bzw. zum Binden an Methoden, dem SELF-Stack, auf dem die Objektpointer gehalten werden über denen gerade eine Methode arbeitet sowie dem Strukturstack, auf dem Indizes in die Strukturtabelle gespeichert werden, um Datenfelder von Teilobjekten korrekt adressieren zu können. (SELF-und Strukturstack werden dabei vom MOB verwaltet.) Parameterstack sowie Returnstack des Forth-Systems sind natürlich weiterhin gültig und notwendig.

Für die Programmierung der Klassendefinitionen wurden eine Reihe von Schlüsselwörtern eingeführt. Das sind vor allem:

- `CLASS:`
  leitet eine neue Klassendefinition ein
- `INHERIT:`
  vollzieht einen Erbvorgang einer Klasse innerhalb einer neuen Klassendefinition
- `VAR:`
  definiert atomare Variable innerhalb einer Klassendefinition
- `PART:`
  definiert ein Variable vom Klassentyp innerhalb einer Klassendefinition
- `METHOD:`
  definiert neue Methode, Methodennamen sind systemglobal
- `REDEFINE:`
  redefiniert eine Methode in einer Klassendefinition
- `;REDEFINE`
  beendet die Redefinition
- `;CLASS`
  beendet Klassendefinition

In dieser Reihenfolge werden sie auch bei einer neuen Klassendefinition angewendet.

Weiterhin existieren einige Schlüsselwörter für den Zugriff auf geerbte Methoden bzw. Datenfelder innerhalb einer neuen Methodenredefinition. Diese Wörter sind:

- `PART`
  dient dem Zugriff auf Teilobjekte, die durch Vererbung eingebunden wurden
- `SUPER`
  auch für den Zugriff auf Teilobjekte, das SELF-Objekt wird jedoch nicht geändert
- `OWNER`
  berechnet für ein Teilobjekt das übergeordnete Objekt
- `BUILDER`
  berechnet für ein Objekt die zugehörige Klasse

Nur wo interaktiv draufsteht, ist auch interaktiv drin. Das wissen und schätzen alle FORTHler. Natürlich kann auch solch eine Klassendefinition interaktiv eingehackert und sofort ausprobiert werden. `OWNER` und `BUILDER` sind noch nicht einmal an Klassendefinitionen gebunden. Sie können auch interaktiv benutzt werden, um für ein beliebiges Objekt die entsprechenden Informationen einzuholen, dessen Objektpointer auf dem Objektstack liegt.

Für den Objekt-Stack des eooS gibt es einige Stapelmanipulationswörter:

- `OSWAP`
  vertauscht die beiden obersten Objektpointer
- `ODUP`
  dupliziert den obersten Eintrag (Top)
- `OOVER`
  der zweite Eintrag wird auf den Top kopiert
- `ODROP`
  der oberste Eintrag (Top) wird entfernt

Mit Hilfe von Klassen werden Objekte erzeugt, sogenannte Instanzen dieser Klasse. Da jedoch auch die Klasse irgendwie erzeugt wird (muß ja irgendwo herkommen), liegt die Vermutung nahe, daß auch Klassen Instanzen anderer Klassen sind. Und genauso ist es auch. Klassen sind auch nur Objekte, an die Methoden gesendet werden, nur daß diese Methoden eben Objekte erzeugen (bzw. andere Klassen). Um sich in diesem Kauderwelsch auch irgendwie zurechtzufinden, müssen vier Sachen beachtet werden.

- Klassen, die Klassen erzeugen, werden Meta-Klassen genannt
- Den Urknall des Ganzen bildet die Klasse META
- Klassen sind auch nur Objekte und werden genauso wie Objekte gehandhabt
- Klassen sind immer statische Objekte

4.3.2 Erzeugung neuer Klassen

Zur Definition einer Klasse werden die in Kapitel 4.3.1 aufgeführten Schlüsselwörter benutzt. Eine Klassendefinition könnte z. B. so aussehen:

```
DYNAMIC CLASS: LOCATION
16B   VAR: X
16B   VAR: Y

METHOD: !XY (ps: x y ===>)
METHOD: @XY (ps: ===> x y)

REDEFINE: !XY   Y O!   X O!
;REDEFINE
REDEFINE: @XY   X O@   Y O@
;REDEFINE
REDEFINE: INIT
( ps: x y ===> )
SELF !XY ;REDEFINE

;CLASS
```

CLASS: leitet die Klassendefinition ein. Vorangestellt wird der Name der Meta-Klasse, von der eine Instanz erzeugt werden soll. Diese Meta-Klasse landet auf dem Objektstack. CLASS: führt eine Methode Namens NEW: aus, die an die Meta-Klasse gesendet wird. Dadurch wird eine neue Klasse mit Namen LOCATION definiert. Es gibt vier vordefinierte Meta-Klassen:

- META
  der Anfang von allem
- STATIC
  wurde mit META definiert, erzeugt Klassen, deren Instanzen nur statisch sein können, sich also im Wörterbuch befinden.
- DYNAMIC
  wurde mit Meta definiert, hat STATIC geerbt, erzeugt Klassen, deren Instanzen auch dynamisch sein können
- ARRAY
  wurde mit META definiert, baut Klassen, deren Objekte Arrays mit fester Länge sind, deren Elemente wiederum Objekte einer Klasse sind -- Das Besondere an diesen Arrays ist die Speicheraufteilung. Die Elemente des Arrays werden nicht durch Zeiger adressiert, die im Array gespeichert sind, die Elemente liegen als Teilobjekte nacheinander eingekapselt im Arrayobjekt. Dadurch braucht die dynamische Speicherverwaltung nur einen Speicherblock verwalten, wobei bei einer Zeigervariante jedes Element als einzelner Speicherblock verwaltet werden müßte. (Anwendung von OWNER auf ein Element führt also zum umschließenden ARRAY-Objekt)

Kommen wir zurück zur Klassendefinition. Es werden jetzt atomare Variablen X und Y definiert, die die Bildschirmposition speichern. Dazu wird ein Typbeschreiber des RECORD-Konstruktes des comFORTH-Systems benutzt - 16B. Dieser Typ beschreibt eine einfache 16b-Variable. Es sind an dieser Stelle auch zusammengesetzte Typen (Records) möglich. Die Namen X und Y sind dabei lokal für diese Klasse und erscheinen nirgendwo im Wörterbuch. Als nächstes folgt die Definition der Methoden, die auf Objekte der Klasse LOCATION anwendbar sein sollen. Hier werden die beiden Methoden @XY und !XY definiert. Werden diese Methoden in der Klassendefinition nicht redefiniert, führen sie bei ihrer Anwendung auf ein Objekt dieser Klasse eine Fehlerbehandlung durch, die mit der Ausschrift "Methode ist nicht redefiniert" endet.

Als letztes folgt nun die Redefinition der Methoden. Hier kann genauso programmiert werden, als wenn es sich um normale Forth-Definitionen handelt. Achtgeben muß man lediglich beim Zugriff auf Datenfelder von Objekten. Da die Objekte irgendwo im dynamischen Speicherraum liegen können, reicht eine Adressierung mit 16b-Breite nicht aus. Das Segment des Objektes muß mit adressiert werden. Dazu dienen die beiden Zugriffswörter O@ und O!. Sie greifen korrekt auf das Segment des derzeit aktiven Objektes zu. Wird eine Klasse mit Hilfe der Meta-Klasse STATIC definiert, können die beiden Forth-Wörter @ und ! verwendet werden, da sich das Objekt im Forth-Segment befinden muß. Innerhalb der Klassendefinition können auch Forthwörter definiert werden, die von den Methoden als Hilfsdefinitionen verwendet werden. Die Namen dieser Hilfswörter sind außerhalb der Klassendefinition nicht mehr sichtbar.

Von der eben definierten Klasse können jetzt Objekte erzeugt werden. Dazu dienen die Methoden NEW:, NEW, NEW^ und NEW^:. Der Doppelpunkt zeigt an, daß ein benanntes Objekt erzeugt wird. Ruft man ein Objekt mit dem erzeugten Namen auf, wird dessen Objektpointer auf den Objektstack gelegt. NEW und NEW^ hinterlassen jedoch nur den Objektpointer auf dem Stack, und der Anwender ist selbst dafür verantwortlich, diesen zu sichern. NEW^ erzeugt dabei ein dynamisches Objekt. Diese und die Methode NEW^: sind also nur auf Klassen anwendbar, die mit der Meta-Klasse DYNAMIC erzeugt wurden.

4.3.3 Vererbung und Einbettung

4.3.3.1 Einfachvererbung

In Kapitel 4.3.2 wurde ein kleines Beispiel angegeben, wie eine Klasse erzeugt werden kann. Dort wurde jedoch keine andere Klasse geerbt bzw. mit PART: eingebettet. Wollen wir nun eine Klasse Punkt definieren, deren Objekte außer der Bildschimposition noch eine Farbe besitzen und an und ausgeschaltet werden können, brauchen wir nicht noch mal von vorn anfangen. Wir können die Klasse LOCATION erben.

```
DYNAMIC CLASS: PUNKT
LOCATION INHERIT: _LOC

16B VAR: FARBE
16B VAR: VISIBLE

METHOD: VIEW
  ( ps: ===> )
METHOD: ?VISIBLE
  ( ps: ===> ? )
  ( ?t wenn an )
METHOD: AN
  ( ps: ===> )
METHOD: AUS
  ( ps: ===> )

REDEFINE: INIT
  ( ps: farbe x y ===> )
  _LOC PART INIT FARBE O! 0
  VISIBLE O!
  ;REDEFINE

REDEFINE: ?VISIBLE
  VISIBLE O@ ;REDEFINE
```

```
REDEFINE: VIEW
 VISIBLE O@
 IF   FARBE O@ _LOC PART X
      O@ _LOC PART Y O@
      POINT
 ELSE BLACK SELF @XY POINT
      ( oder BLACK _LOC
      SUPER @XY POINT )
 THEN ;REDEFINE

REDEFINE: AN
 TRUE  VISIBLE O! SELF
 VIEW ;REDEFINE
REDEFINE: AUS
 FALSE VISIBLE O! SELF
 VIEW ;REDEFINE

;CLASS
```

Jede geerbte Klasse bekommt einen für die zu definierende Klasse lokalen Namen, unter dem sie während der Redefinition der Methoden angesprochen werden kann. So können auch gleiche Klassen mehrfach geerbt werden, ohne das es zu Namenskonflikten kommt. Die Redefinition der Methoden könnte man bestimmt noch effizienter gestalten, es geht jedoch hier mehr darum, die Möglichkeiten zu zeigen, die es gibt, um geerbte Klassen anzusprechen. Die wichtigsten Schlüsselwörter dafür sind PART und SUPER. Mit ihnen können Methoden bzw. mit PART auch Datenfelder von geerbten Klassen angesprochen werden. Der lokale Name der geerbten Klasse wird dabei vor das Schlüsselwort geschrieben. Es ist auch eine Hintereinanderreihung von SUPER oder PART Sequenzen möglich, so das eine komplette Vererbungshierarchie entlanggelaufen werden kann. Nötig dafür sind natürlich die lokalen Namen der geerbten Klassen. Der Unterschied zwischen PART und SUPER besteht in der Art der Bindung des Objektes an die aufzurufende Methode der geerbten Klasse. PART berechnet zur Laufzeit immer das entsprechende Teilobjekt innerhalb des umschließenden Objektes, so daß die Methode wirklich an ein Objekt des geerbten Klassentyps gebunden wird. Dadurch hat man mit PART auch direkten Zugriff auf Variablen der geerbten Klasse. Anders verhält es sich bei SUPER. Hier wird die geerbte Methode an das umschließende Objekt, also das derzeit aktive SELF-Objekt gebunden. Das hat tiefgreifende Konsequenzen für den Aufruf von Methoden innerhalb der geerbten Methode. Nehmen wir ein neues Beispiel.

Wir wollen ein Ventil steuern, dessen Zustand grafisch dargestellt werden soll. Wir erben die Klasse PUNKT, da dort schon ein Teil der zu erreichenden Funktionalität definiert wurde.

```
DYNAMIC CLASS: VENTILCLASS
PUNKT INHERIT: _PUNKT

METHOD: DISPLAY_AN
METHOD: DISPLAY_AUS

REDEFINE: INIT ...
 ;REDEFINE

: ANSCHALTEN
  ( schaltet Ventil ein )
  ... ;
: AUSSCHALTEN
  ( schaltet Ventil aus )
  ... ;

REDEFINE: AN   ...
 ANSCHALTEN   ... ;REDEFINE
REDEFINE: AUS ...
 AUSSCHALTEN ... ;REDEFINE

REDEFINE: VIEW
 ( visualisiert den
 Zustand oder schaltet
 DISPLAY ab)
 SELF ?VISIBLE
 IF ... ELSE ... THEN
 ;REDEFINE

REDEFINE: DISPLAY_AN
 _PUNKT SUPER AN
 ;REDEFINE

REDEFINE: DISPLAY_AUS
 _PUNKT SUPER AUS
 ;REDEFINE

;CLASS
```

In den Methodenredefinitionen für DISPLAY_AN und DISPLAY_AUS wurde SUPER benutzt. Dadurch können die Methoden AN und AUS der geerbten Klasse PUNKT aufgerufen werden. Diese wiederum rufen die Methode VIEW auf, die in der neuen Klasse redefiniert wurde. Durch die Verwendung von SUPER wird aber gerade diese redefinierte Methode benutzt, da das SELF-Objekt von der Klasse VENTILCLASS stammt, und die Anzeige des Ventilzustandes funktioniert korrekt. Hätte man an dieser Stelle das Wort PART benutzt, würde an Stelle des Ventilzustandes ein Punkt auf dem Bildschirm erscheinen, da AN und AUS die Methode VIEW der Klasse PUNKT aufrufen würden.

4.3.3.2 Mehrfachvererbung

Genauso wie Einfachvererbung funktioniert auch die Mehrfachvererbung. Nur das hier eben mehrfach die Anweisung INHERIT: verwendet wird. Was passiert jedoch, wenn verschiedene Klassen Methoden mit ein und demselben Namen besitzen und diese Klassen zusammen in eine Neue geerbt werden? Das Problem löst sich durch die Vorschrift, daß sämtliche geerbte Klassen lokale Namen bekommen. Dadurch ist jede geerbte Methode, auch wenn es noch eine mit dem gleichen Namen gibt, über PART bzw. SUPER ansprechbar. Soll in der neuen Klasse die Methode, die mehrmals geerbt wurde, die Funktionalität sämtlicher geerbter Methoden gleichen Namens aufweisen, kann man sich eine Automatik des Systems zugute machen. Beim Erbvorgang werden alle Methoden gleichen Namens zusammengekettet. Das heißt, wird die entsprechende Methode in der neuen Klassendefinition nicht redefiniert, führt sie beim Aufruf sämtlichen geerbten Code aus und zwar in der Erbreihenfolge. Sinnvoll ist diese Verkettung des Methodencodes z. B. bei der INIT-Methode. Benötigt man in der neuen Klasse keine zusätzlichen Initialisierungsmaßnahmen, braucht die INIT-Methode auch nicht redefiniert zu werden.

4.3.4 Einbettung

Eine komplexe Klasse besteht aus vielen einzelnen einfacheren Klassen, die einmal durch Vererbung eingebunden wurden, andererseits aber auch über Einbettung dazu kommen können. Diese Klassen dienen dann nur als Hilfsmittel, die Funktionalität der neuen Klasse intern zu erweitern. Das Wort intern ist hier das Schlüsselwort. Die Methoden von eingebetteten Klassen werden nämlich nicht geerbt und können somit auch nicht als Schnittstelle der neuen Klasse benutzt werden. Sie dienen lediglich dem Zugriff auf Teilobjekte der eingebetteten

Klasse. Dadurch ist eine optimale Kapselung gewährleistet. Es ist bei Einbettung nämlich auch nicht, wie bei Vererbung, der direkte Zugriff auf Datenfelder der eingebetteten Klasse möglich. Diese sind nur über deren Methoden ansprechbar.

Um Klassen in einer neuen Klasse als eingebettet zu deklarieren, wird das Wort PART: benutzt. Diese Klassen gehören damit sozusagen zu den Variablen der neuen Klasse, wie die VAR: -Variablen, nur das sie vom Klassentyp sind und auf sie keine Speicheroperationen sondern Methoden angewendet werden.

```
Beispiel:

DYNAMIC CLASS: 1LINKABLE
  \ OPTR ist eine Klasse,
  \ deren Objekte andere
  \ Objektpointer
  \ aufnehmen können
  OPTR PART: NEXT^

METHOD: @NEXT
METHOD: !NEXT
METHOD: OLINK
METHOD: ODELINK

REDEFINE: @NEXT
  ( os: ===> ob' )
  NEXT^ O^@   ;REDEFINE
REDEFINE: !NEXT
  ( os: ob' ===> )
  NEXT^ O^!   ;REDEFINE

REDEFINE: OLINK
  ( os: ob' ===> )
  ODUP NEXT^ O^@! OSWAP
  !NEXT ;REDEFINE

REDEFINE: ODELINK
  ( os: ===> ob' )
  NEXT^ O^@ ODUP NIL O^<>
  IF   ODUP @NEXT NEXT^ O^!
  THEN
  ;REDEFINE

;CLASS
```

Zugegriffen wird auf Objekte der eingebetteten Klasse über deren lokalen Namen, wie im Beispiel NEXT^. Da die Methoden der eingebetteten Klassen nicht geerbt werden, gibt es auch keine Konflikte mit geerbten Methoden gleichen Namens. Es wird also auch keine dieser Methoden mit einer anderen verkettet. Eine Ausnahme bildet hier die INIT-Methode. Sie wird auch bei eingebetteten Klassen mit verkettet. So das eine korrekte Initialisierung gewährleistet ist. Gut zu erklären auch am letzten Beispiel. Hier wird die INIT-Methode nicht redefiniert, obwohl eine Initialisierung von NEXT^ mit NIL unbedingt nötig ist. Das geschicht nämlich automatisch bei der Neuerzeugung eines Objektes der neuen Klasse.

4.3.5 Frühe - Späte Bindung

```
: NEUER_PUNKT
  ( os: ===> ob' )
  PUNKT NEW^   ;
```

PUNKT sei die neu definierte Klasse aus Kapitel 4.3.3.1. In der Forth-Definition NEUER_PUNKT haben wir einen klaren Fall von früher Bindung. Zur Compilationszeit sind Objekt und Methode bekannt. Der zu bindende Methodencode kann zur Compilationszeit gesucht und ein entsprechender Ruf compiliert werden. Und genau das passiert auch, weil dadurch eine Menge Laufzeit gespart wird. Noch eine nicht objektorientierte Forthdefinition:

```
: NEUER_????
  ( os: classob' ===> ob' )
  NEW^ ;
```

Das ist nun eine echte späte Bindung, weil beim Compilieren nicht feststeht, welches Objekt denn nun zur Laufzeit die Methode NEW^ ausführen soll. Also erst zur Laufzeit liegt das entsprechende Objekt auf dem Objekt-Stack und erst dann kann auch der entsprechende Methodencode gesucht werden. Das ist dann eine Aufgabe des MOBs. Das Ganze dauert dann natürlich etwas länger als frühes Binden. Späte Bindung findet man auch während Redefinitionen, wenn mit SELF gearbeitet wird, oder wenn der Programmierer einfach einen Methodennamen hinschreibt, weil er angeblich weiß, welches Objekt dort zur Laufzeit auf dem Objekt-Stack zu liegen hat. Das böse Erwachen kommt dann erst, wenn irgendwann zur Laufzeit eine Fehlermeldung ausgelöst wird:

```
Objekt .... und Methode ...
gehören nicht zusammen.
```

Neben früher und später Bindung gibt es noch eine Art dazwischen, bei der der Methodencode zwar schon feststeht, also auch schon der Klassentyp des zu bindenden Objektes, jedoch die konkrete Instanz erst zur Laufzeit feststellbar ist. Diese Art der Bindung tritt bei Definitionen mit SUPER bzw. PART auf und dauert etwas länger als frühes Binden, jedoch nicht so lange wie späte Bindung.

Mit diesem Abriß ist nun nicht das gesamte eooS bis ins kleinste Detail beschrieben. Es gibt jedoch einen Überblick, was mit dem System machbar ist und wie es angewendet wird.

❑

OOCOBOL auch für oder gerade für Forthler

Natürlichsprachlich!

MicroFocus entwickelt ein Object-Cobol, das an ANSI X3J4 (Standardisierung von Cobol geplant für 1997) orientiert ist. Zu einem besonderen Feature schreibt MicroFocus in einem PresseText:

ObjectCOBOL für UNIX enthält ein einzigartiges Feature, welches es von anderen objektorientierten Sprachen unterscheidet: die benutzerorientierte Syntax. Von MicroFocus entwickelt, vereinfacht dieses Feature das Lesen und Schreiben von Programmen, indem die Anweisungen direkt in den Programmen aufgerufen werden können. Die benutzerdefinierte Syntax ermöglicht dem Autor einer neuen Klasse zu spezifizieren, wie die Befehle für die neue Klasse geschrieben werden sollen. Folglich verfügt der Code von ObjectCOBOL-Applikationen über Anweisungen, die denen der natürlichen Sprache ähnlich sind und auch von Nicht-Programmierern verstanden werden können.

Nun ja ¡ gut !
aber einzigartig ?¿

Arndt Klingelnberg

Der STREICH feiert seinen 25. Geburtstag

von Bernd Paysan

Vor einiger Zeit wurde in comp.lang.forth ein "Interview" mit Charles Moore gepostet. Der Interviewtext ist von ftp.wsoy.com, die Datei pub/interviews/1994/cmoore.Z.
Disclaimer: WSOY ist als notorischer Fälscher von News und Interviews bekannt. Der untenstehende Artikel ist (ohne Erlaubnis) textgetreu übersetzt und enthält nicht notwendigerweise die Wahrheit.
Ich habe "you" mit "du" übersetzt, weil ich von einer persönlichen Umgebung und einem lockeren Interviewstil ausgegangen bin. Auch habe ich versucht, Computerslang soweit unübersetzt zu lassen, wie es in einer entsprechenden Umgebung in Deutschland üblich ist.

New York, 18.2.1994, 16:50 Uhr

Dies ist die Mitschrift eines Interviews auf WSOY, des schlechtesten Radiosenders im bekannten Universum und darüber hinaus.

Interviewer: Johnny Hacker
Interviewter: Charles Moore

JH: Charles, wann hast du deine Karriere als Streichespieler begonnen, und warum?

CM: Es muß Ende der 60er gewesen sein, ich arbeitete als Programmierer für ein großes Teppichgeschäft. Es gab nicht viel zu tun, die meisten Programme arbeiteten wie gewünscht und sie waren gut in COBOL geschrieben. Ich war die meiste Zeit müßig, aber ich hatte Zugriff zu einem alten IBM-Mainframe. Damals hatten nicht viele Leute Zugang zu einem Computer, so war ich einer der ersten, die viel Zeit hatten, mit einem Computer zu spielen. Es gab kein Solitair für Windows oder Tetris, nicht einmal "256 einfache Spiele in TSR80 BASIC", so begann ich meine eigenen Spielzeuge zu schreiben.

Einmal laß ich etwas in den Zeitschriften über Programmiersprachen der vierten Generation. Diese Sprachen waren interaktiv, man war also die ganze Zeit online, beim Tippen, Testen und Debuggen des Programms. Nun, das ist eine Verschwendung von Rechenzeit. Das mußte ein höllisches Spielzeug sein. Die Zeitschrift erwähnte APL, aber es hatte zwei Nachteile: Es verwendete alle möglichen seltsamen griechischen Zeichen, Pfeile etc. die unser Computer nicht hatte, und es konnte nützliche Dinge wie Matrix-Inversion mit ein paar Befehlen. Ich beschloß, mein eigenes "Programmiersprache der vierten Generation"-Spielzeug zu basteln, so unlesbar wie APL, aber nur ASCII-Zeichen und nur elementare Integerrechnung. Ich wollte es FOURTH nennen, aber es gab nur Fünf-Zeichen-Dateinamen auf der Maschine, also nannte ich es FORTH.

JH: Aber die Teppichgesellschaft schloß wegen Computerproblemen, nicht wahr?

CM: Ja, sie schlossen, ihr Bestellungssystem war ziemlich buggy, und ich konnte dem nicht abhelfen. Kein großes Problem für mich. Programmierer hatten damals kurzfristige Verträge und ziemlich schnell bekam ich einen Job am Kitt Peak Observatorium. Ich fand einen ungenutzten Data General Minicomputer, und ich schrieb meine zweite Version von FORTH für ihn. Alles lief gut, bis einige meiner Kollegen auch mit FORTH zu spielen begannen. Eines Tages kam mein Boss herein. Keiner war am Teleskop und vier von uns spielten tic-tac-toe am Computer. Wir erklärten die Situation und sagten, daß wir eine revolutionäre neue Programmiersprache, FORTH, erfanden, und daß sie unsere Programmierjobs deutlich beschleunigen würde. Zu unserer großen Überraschung glaubte mein Boss das tatsächlich. In diesem Moment war der Streich geboren. Es muß Ende 1969 gewesen sein, vor beinahe 25 Jahren.

JH: Von diesem Zeitpunkt an wurde FORTH ein offizielles Projekt des Observatoriums? Hat man dich angewiesen, all die Teleskopsteuerungssoftware in FORTH zu schreiben?

CM: Von diesem Augenblick an wurden wir angewiesen, FORTH-Systeme für alle Minicomputer am Observatorium zu entwickeln. Das waren ein paar wenige verschiedene Maschinen. Während wir dies taten, fügten wir mehr und mehr absurde Features hinzu, von denen wir hofften, sie würden das Projekt sehr schnell wieder killen. Wir fügten ein DO..LOOP-Konstrukt dazu, einen sichtbaren Returnstack mit >R und R>, eine Menge schräger Divisionsworte, wie U/, M/ und /MOD, von denen keines sinnvoll erschien. Wir hatten Disk Blocks, keine Dateien. Wir hatten ein Wort namens ;: (später bekannt als DOES>), das so sonderbar war, daß man es unmöglich erklären konnte. Wir hatten Assembler unter FORTH, und wir taten unser Bestes, um die Syntax so furchtbar wie möglich zu machen. Am Ende hatten wir ein Multiuser-Multitaskingsystem. Jeder Benutzer konnte einfach >R oder ! eingeben und das ganze Ding abschießen, aber es schien zu laufen. Wir waren erstaunt, daß unser Boss niemals >R oder ! ausprobierte. So dachte er, das Ding war ernsthaft. Als wir FORTH auf all den Minis hatten, wurden wir angewiesen, alle nötige Programmieraufgaben darin zu erledigen.

JH: So warst du schwer enttäuscht, daß das Projekt nicht gekillt wurde und du angewiesen wurdest, tatsächlich FORTH zu nutzen ?

CM: Ja, tatsächlich. Es macht keinem Spaß, FORTH zu nutzen, wenn Assembler so viel freundlicher ist. Viele unserer Aufgaben brauchten sowieso Assembler, aber nun waren wir gezwungen, unseren Forth-Assembler zu verwenden, den wir so schrecklich wie möglich gemacht hatten. Nur die Länge und die ersten drei Buchstaben jedes Wortes waren gespeichert. So gab es eine Menge unerwarteter Wortkollisionen, zum Beispiel CONVERT und CONTEXT waren für das System dasselbe. Wir nutzten das ausgiebig, um unseren Sourcecode unklar zu gestalten. Ein paar Monate später war das Projekt immer noch nicht gekillt. Sogar etwas nützliche Arbeit war in FORTH erledigt. Andere Observatorien rund um die Welt bestellten und nutzten zu unserem Schrecken sogar unsere Software.

JH: Wenn FORTH nicht von deinem Job verschwand, hast du entschieden zu gehen?

CM: Genau. Nach einigen Monaten konnte ich nicht länger mit ansehen, daß mein Streich alle die guten Software-Tools ersetzte, die vorher da waren. Sie hatten all die guten COBOL- und FORTRAN-Compiler und die Assembler in den Müll geworfen. Ich tat mein möglichstes, nicht zu kooperieren, aber sie lobten mich trotzdem. Dann verließ ich das Observatorium. Eine Kollegin, Elizabeth Rather, ging mit mir. Zu dieser Zeit war der Arbeitsmarkt für Programmierer nicht mehr so gut. Da ich gezeigt hatte, daß ich ein ziemlich lausiger Programmierer bin, wollte mich keiner einstellen. Ich dachte, sie hatten recht. Nach einigen Monaten Arbeitslosigkeit beschloß ich, meine eigene Company zu gründen. Ich hatte keine Erfahrung außer mit FORTH, so wurde meine Company FORTH Inc. E. Rather war meine Angestellte. Nun waren wir gezwungen, unseren eigenen Streich in unserer eigenen Gesellschaft auszubeuten.

JH: Und die Company lebt immer noch?

CM: Traurig, aber wahr.

JH: Dein erstes Produkt, PolyFORTH, war ziemlich erfolgreich, nicht wahr?

CM: Ja, war es. Es hatte einen lächerlichen Preis, etwa $4000 für ein Single-User-System. Trotzdem verkauften sich viele tausend Kopien. Es machte aus dem einfachsten Computer, sagen wir einem 8080, ein Multiuser-Multitasking-System. Dieses einzige Feature verkaufte es, zweifellos. Daß das System so schnell abstürzte, wurde erst nach dem Kauf bemerkt. Unser Hauptmarkt war die PDP11. Unix wurde zu dieser Zeit nicht kommerziell vertrieben. Es verlangte eine Harddisk.

JH: Und dann hast du ein freies FORTH an die FIG gegeben?

CM: Die FIG gab es damals noch nicht. Einige Leute hatten Forth gekauft und wollten es verbreiten. Ich dachte, das sei keine gute Idee, so beschloß ich, eine kastrierte Version von FORTH der DECUS, der DEC User Group zu überlassen. Ich entfernte die Multitaskingfähigkeit, ich schraube 'raus was ich konnte (sowas wie die Drei-Zeichen-Namen und Vokabulare) und ich gab keinerlei Dokumentation mit. Dieses Stück Scheiße wurde benutzt, zu meiner großen Überraschung. Seine Anwender gründeten die FIG und entwickelten es zu FIG-Forth.

JH: Als der TRS80 herauskam, wollten seine Entwickler FORTH einbauen? Wäre das wirklich passiert, hätte es dich verdammt reich gemacht?

CM: Sie fragten mich wirklich, ein FORTH für den TRS80 zu schreiben. Ich lehnte ab oder ich verlangte einen lächerlich hohen Preis dafür. Es hätte mich sehr reich gemacht. Ich hätte sein können was Bill Gates jetzt ist, aber ich habe meine Lektion aus dem FIG-Vorfall gelernt. Wäre FORTH in den TRS80 eingebaut worden, es wäre zu Millionen Haushalten gekommen. Ich glaubte sicher, daß hätte einen verwüstenden Effekt auf unsere Gesellschaft. Besser von BASIC verdorben als FORTH an den Haken gegangen.

JH: Glaubst du wirklich, die Gesellschaft gerettet zu haben, indem du Tandy ein FORTH für den TRS80 verweigert hast?

CM: Ja, habe ich. Forth ist nicht im Mainstream. Nur ein kleiner Prozentsatz unserer Gesellschaft kommt dazu, FORTH zu nutzen. Unsere Gesellschaft kann einen kleinen Prozentsatz Querköpfe vertragen. Sie bereichern unsere Kultur irgendwie. Unsere Gesellschaft kann auch einen kleinen Prozentsatz Leute vertragen, die an absurde Religionen wie Scientology glauben. Wäre FORTH im Mainstream, es hätte einen schädlichen Einfluß auf unsere Gesellschaft.

JH: Warum denkst du, wurde FORTH von so vielen Leuten verwendet, trotz seiner Unannehmlichkeit?

CM: Nun, zum einen scheint FORTH sehr einfach zu implementieren zu sein. Jeder Idiot auf der Welt denkt, er kann seine eigene FORTH-Implementierung schreiben. Und das ist in der Tat wahr. Ich entwarf FORTH so, daß es einfach zu implementieren war. Damals in den 60ern wollte ich mein Vierte-Generations-Sprachen-Spiel so schnell wie möglich. Deshalb. Und sobald sie ihr FORTH implementiert haben, entdecken sie, daß es nicht einfach ist, etwas nützliches damit anzufangen. Aber dann sind sie bereits davon eingenommen und machen weiter. Nicht, daß irgendetwas nützliches erreicht wird, aber die Arbeit, die sie in Lernen und Implementieren von FORTH investiert haben, ist nicht einfach abgeschrieben. Sie machen weiter für eine lange Zeit, um zu sehen, ob ihre Investitionen längerfristig belohnt werden. Und sie geben vor (und denken manchmal sogar), daß sie sehr zufrieden damit sind. Und wenn sie sich FORTH kaufen, statt es zu schreiben, nutzen sie es aus dem exakt gleiche Grund weiter.

Kurz, FORTH ist nicht viel mehr als Pappmaché und heiße Luft. Einfach zu implementieren und einfach zu verkaufen. Nicht mehr als das.

JH: Nach FORTH Inc. bist du Chipdesigner geworden?

CM: Ja, ich hatte genug von Forth, so versuchte ich einen neuen Job zu finden. Ich versuchte, einen Job im Chipdesign zu finden, aber ich hatte keine Erfahrung. Dann las ich etwas über spezielle Mikroprozessoren für bestimmte Programmiersprachen. Es gab die WD Pascal Micro Engine und die MIT LISP Machine. Bei-

de waren echt beschissen. Ich dachte, es sei ein leichtes, einen speziellen Prozessor für FORTH zu designen. Ich designte ihn als 4000-Gate-Array, komplett mit Multipliziererbug, so daß niemand ihn benutzen würde. Es war ausschließlich eine Übung für mich, um mehr Erfahrungen im Chipdesign zu sammeln. Der Chip wurde der NOVIX NC4000. Er wurde tatsächlich produziert. Es gab sogar einen Nachfolger, den Harris RTX2000. Er war erfolgreicher als viele echte Micros. Ich dachte, das war ein echtes Desaster.

JH: Und dann verließt du FORTH? Du hast etwas Namens OK gemacht?

CM: Ja, mein nächstes Übungsprojekt am Computer sollte so nutzlos sein, wie es nur ging. Ich beschloß, einen 20-Bit-Prozessor zu entwerfen, mit Stacks zu klein für traditionelles FORTH und vielen dummen Restriktionen. Er sollte auch Video generieren, aber nur NTSC, was höchstens für den Spielmarkt brauchbar wäre. Und da der Chip direkt auf dynamisches RAM, aber nur auf 1K Worte ROM zugreifen konnte, würde es auch für den Spielmarkt nutzlos sein. Das ist der uP20. Niemand hat das Ding bislang produziert, das ist eine gute Sache.

Ich habe alle meine Chipdesign-Software selbst geschrieben. Es sollte unbedingt ein Spielzeug sein und niemand sonst sollte jemals in Versuchung geraten, es zu nutzen. Deshalb entwarf ich eine neue Sprache, OK. Es ist im wesentlichen ein Programmiersystem, in dem man direkt binären Objektcode für eine FORTH-artige virtuelle Maschine entwirft. Um seine Benutzung noch schrecklicher zu machen, entwarf ich ein menügesteuertes User-Interface mit nur drei Tasten dafür. Man hat nur drei Tasten. Man muß Programme eingeben, Chipdesigns zeichen, etc. etc. Wenn jemand sonst so verrückt ist, es zu benutzen, denke ich, ist es Zeit, mich zu erschießen.

JH: Letzte Frage heute: Was denkst du über den neuen ANSI FORTH Standard?

CM: Es ist eine Verschwendung von Geld, Hirn und Zeit. Ich finde es traurig, daß immer noch soviel Geld und Zeit für meinen Witz verschwendet wird. Die alten FORTHer werden es eh nicht verwenden. Sie denken ANSI FORTH ist nicht mehr FORTH. Sie mögen irgendwie das Fehlen von Standards, daß man unterschiedlichen Code für 16-Bit byteadressierte Maschinen und wortadressierte 32-Bit-Maschinen schreiben muß. Die jüngere Generation würde FORTH sowieso nicht nutzen, Standard oder nicht. Nun, das ist, was ich eh hoffe. Ich hoffe, daß es letzten Endes ausstirbt. Die Generation dazwischen, die werden den neuen Standard nutzen. Aber der Standard wird nicht verhindern, daß jeder sein eigenes FORTH schreibt, und daß es mehr FORTH-Implementationen als FORTH-Applikationen gibt. Und die wenigen Applikationen in ANSI FORTH werden den Standard auf etliche komplizierte Weisen verletzen, sodaß sie auch überhaupt nicht portabel sind. Die meisten (oder alle) ANSI FORTH-Implementierungen werden auch Mist sein. So wird es dasselbe Chaos sein wie jetzt, trotz (oder dank) des Standards.

JH: Hoffen wir, daß dein Streich nicht seinen 50. Geburtstag feiert.

CM: Das hoffe ich auch.

❑

Die Neuentwicklung des Bits

von Michael Symonds

Wer keinen Spaß versteht,versteht auch keinen Ernst.

Liebe Forthgemeinde,

die Tage werden kürzer (und die Nächte länger!). Nach einem sinnlos heißen und langen Sommer, manch einem sind seine, in letzter Minute und panischer Hektik angeschafften Hamsterkäufe an Selters und Bourbon schon wieder verdorben, neigt sich nun das Jahr dem Ende entgegen. Es ist wieder die Zeit, in der wir uns beschaulich in unser Inneres zurückziehen, in der wir das Erreichte betrachten und Abstand gewinnen können. Und so scheint es mir, daß gerade jetzt die Zeit gekommen ist, einige Dinge der Informatik, die uns scheinbar schon vollkommen vorkommen, uns so mir nichts dir nichts in Fleisch und Blut übergangen sind, - ja, wie soll ich es sagen - zu überdenken.

Und so habe ich, unermüdlich wie es meine Art ist, den ersten Stein geworfen und mich an das kleinste Ding aller Dinge herangewagt, diesem in schier endloser Zahl unermüdlich für uns in Knechtschaft schuftendem und doch fast Vergessenem - dem Bit. Nach dem Motto, daß es nichts gäbe, was nicht noch einmal erfunden werden könnte, und gerade dafür ist ja Forth wie nichts anderes prädestiniert, habe ich mich an die Neuentwicklung des Bits als solchen gemacht. Nunmehr möchte ich meinen jetzigen Entwicklungsstand durch die Abb. 1 einer breiten Öffentlichkeit vorstellen. (Vorraussetzung ist AOP.SCR aus [VORWÄRTS94]!). Bescheidenheit war schon immer einer meiner größten Schwächen und gerade deswegen muß ich hier einmal über meinen Schatten springen: Ist es mir nicht in einzigartiger Weise gelungen ein Minimun an Funktion mit einem Maximum an Eleganz zu verbinden?

Des weiteren kann ich mathematisch beweisen, daß das Verhältnis aus Effektivität/Funktion gegen unendlich strebt. Wie in wissenschaftlichen Kreisen unbestritten bekannt ist, ist das Bit die kleinstmögliche Informationseinheit. Daraus leite ich die kühne These ab, daß es auch kleinstmögliche Funktionseinheit ist. So gesehen ist das Bit, ich will es hier einmal bewußt laienhaft ausdrücken, nur ein ganz ganz klein bißchen mehr als Null. Bekanntermaßen strebt ein Quotient, dessen Divisor annähernd Null ist, gegen Unendlich. Demzufolge strebt auch das Verhältnis von Effektivität/Funktion gegen Unendlich; was zu beweisen war.

```
Screen # 0
 0 BIT.SCR    Bloedelbeispiel fuer AOP.scr \ Symds 23.11.94
 1
 2
 3 History:
 4 23.11.94  Ersterstellung
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen # 1
 0                                    \ Symds 18.11.94
 1
 2      \needs  Automat:  include aop.scr
 3
 4
 5
 6
 7
 8
 9                          -->
10
11
12
13
14
15
```

```
Screen # 2
 0                                    \ Symds 19.11.94
 1
 2 Automat: bit
 3           Canal  show
 4           Canal  get      Canal  store
 5           Canal  set      Canal  clear    Canal  toggle
 6         ;automat
 7
 8
 9 : show0    ( -- )   ." Mein Zustand ist jetzt 0"  cr   ;
10 : show1    ( -- )   ." Mein Zustand ist jetzt 1"  cr   ;
11
12 : store0   ( flag -- )  0= ?exit  set    ;
13 : store1   ( flag -- )     ?exit  clear  ;
14                                                    -->
15
```

```
Screen # 3
 0  Defer 'eins                       \ Symds 19.11.94
 1
 2 bit State: null  show0   noop
 3                  show0
 4                  false          store0
 5                  'eins          noop           'eins
 6          ;state
 7
 8 bit State: eins  show1   noop
 9                  show1
10                  true           store1
11                  noop           null           null
12          ;state
13
14   ' eins  Is 'eins       ' null s_entry
15
```

Als nächste Entwicklungsschritte sind geplant, das Bit echtzeitfähig auf einem Transputer zu parallelisieren, es netzwerkfähig zu machen, um es schließlich mit einer MS-Windows-Benutzeroberfläche in C++ zu versehen. Letzteres werde ich aber erst zu Beginn meiner Pension in Angriff nehmen, weil ich noch weitere Projekte in meinem Leben vollenden möchte. Ich denke da z. B. an eine Neuentwicklung des Nicht, einem objektorientierten Goto oder einem Sortierverfahren, das auch auf Nullmengen oder Mengen mit einem Element angewendet werden kann. (Sollte mir dieser wegweisender Algorithmus gelingen, wäre das Verhältins aus Effektivität/ Funktion auf jeden Fall Unendlich.)

Wer nun noch in dem Irrglauben verharrt, daß man in Forth keine ernsthaften Anwendungen programmieren könne, der soll fortan meinen Zorn fürchten.

Im übrigen geht der Quelltext in Abb. 1 ausdrücklich nicht ins Public Domain über. Alle Rechte sind vorbehalten, denn auch dem schnöden Mammon muß gefrönt sein.

Verweise:

[VORWÄRTS94]
Artikel "Vorwärts - und dann kreuz und quer II" in dieser VD

❑

VD Splitter

von Arndt Klingelnberg gesammelt

Die beste Zeit einen Computer zu kaufen, ist immer der nächste Tag.
Nick Donatiello

(...bei einem neuen Processor das nächste Jahr
intel, 1994)

"Das neue Computer Modell ist wesentlich schneller, damit kann ich in der gleichen Zeit doppelt so viele Fehler machen."
ein Forth Programmierer

Das war der Programmierwettbewerb auf der Echtzeit ´94

von Jörg Plewe

Es ist bereits gute Tradition auf Computermessen, dort Programmierwettbewerbe abzuhalten, auf denen sich Programmierer nach Herzenslust darstellen und zur Not auch blamieren dürfen.

In Forthkreisen berühmt sind die Wettbewerbe auf der Echtzeit-Messe, die nun schon seit einigen Jahren immer wieder für Gesprächsstoff in der VD und zum Teil sogar in der echten Presse sorgen.

Dieser Bericht soll nun die Geschichte zweier Helden erzählen, die angetreten sind, das Forth-Fähnchen hochzuhalten und der Welt zu zeigen, daß auch Forth-Nichtprofis in der Lage sind, mit Nichtforth-Profis mitzuhalten.

Dabei hatte die größte Hürde nichts mit der gestellten Aufgabe zu tun. Der kritischste Augenblick war die Entscheidung, überhaupt teilzunehmen. War es eine Schnapsidee oder unfaßbarer Übermut? Wir wissen es nicht mehr. Auf jeden Fall waren da zwei sehr aufgeregte Jörgs, obwohl der Wettbewerb noch gar nicht in Sicht war.

In der nächsten Zeit legte sich die Aufregung etwas bis zu dem Tag, da die Beschreibung der Schnittstelle des zu steuernden Gerätes bekanntgegeben wurde: Eingänge M,N,S,O,W und Ausgänge RI und RS. Außerdem sollte man einen 9V-Block und ein Teelicht mitbringen. Hmm?! Der Name des Gerätes: Die Pusteblume.

Irgendetwas mit Thermik bestimmt. Wahrscheinlich dreht es sich auch. Wozu sonst der 9V-Block? M,N,S,O,W können doch eigentlich nur Mitte, Norden, Süden, Osten und Westen sein. Oder eine Falle. Wir tippten, daß ein warmer Aufluftstrom durch eine Scheibe oder ähnliches gesteuert werden muß, so daß irgendein leichter Gegenstand, wie z.B. ein Pusteblumensamen, in der Schwebe gehalten werden kann.

Zunächst wollten wir den Wettbewerb mit einer (früher) weitverbreiteten Spielkonsole bestreiten, für die ein ganz exzellentes PD-Forthsystem existiert. Doch leider, leider hatte man bei Atari beim Zusammenlöten so den ein oder anderen Draht am Printerport vergessen, so daß wir wohl oder übel auf den Industriestandard setzen mußten.

Das größte Problem für uns war zunächst: Das Kabel. Da kommt man als professioneller Nichtprofi schon ins Schleudern. Doch gut ist es, wenn man einen Profi kennt. Ein ganzes Ingenieurbüro hat uns in stundenlanger Feinarbeit zu einem solchen technischen Wunderwerk verholfen. Für zwei hardwarefremde Forth-Enthusiasten war es spannend zu sehen, welchen auch philosophischen Aufwand man mit einem einfachen Kabel treiben kann. Es soll halt top sein und muß funktionieren, wenn´s ernst wird. Deshalb wurden alle Leitungen mit einem Ohmmeter auf Durchgang geprüft und von zwei unabhängigen Qualitätsprüfern mit der Beschreibung der Pusteblumenschnittstelle verglichen.

Auf so einen Wettbewerb geht man natürlich nicht unvorbereitet (es sein denn, man ist bereits mehrfacher Gewinner). Einige nützliche Routinen sollten schon im Koffer sein. Wichtig ist es vor allem, die 7 Pins und die Zeit im Griff zu haben. Das erste Vorbereitungstreffen war von der neu erschienenen REBEL-ASSAULT-CD geprägt und gipfelte im Feuersturm auf den Sternenkreuzer des Imperiums. Und in der Erkenntnis, daß ein Joystick ideal zum interaktiven Konzept von Forth und zur Aufgabenstellung der Pusteblume paßt. Mit einem Joystick kann man sich sehr gut auf vier Himmelrichtungen und eine Mitte einstellen. Zu diesem Zeitpunkt waren wir sogar entschlossen, nötigenfalls des Pusteblumenmodell während des Wettbewerbes nicht zu programmieren, sondern einfach mit dem Joystick zu steuern. Was die Jury wohl dazu gesagt hätte...

Die zweite harte Anforderung der Pusteblume lag in der Kürze der Zeit, in der die Signale von und zur Pusteblume behandelt werden mußten. Die Spezifikation schrieb vor, binnen 1 ms ein Signal erkennen zu können. Damit gerieten wir in den Strudel der Zeitmessung auf dem PC. Die Tücken der Millisekunde sind hier in der VD schon hinreichend behandelt worden. Die MS von Prof. Dr. T. Beierlein ist und bleibt die erste Wahl. Leider stellte sich beim Studium der reichlich vorhandenen Literatur zur PC-Systemprogrammierung heraus, daß diese MS die Millisekunde der Langzeitmessung ist. Zwar läßt sich die Echtzeituhr der PC/AT-Klasse (XT´s dürfen bei uns nicht mit zur Echtzeit) noch zu kürzeren Zeitauflösungen hin beeinflussen, hier wird aber die verfügbare Literatur schnell dünn und widersprüchlich.

Neben der Echtzeituhr mit ihren vielfältigen Funktionen gibt es noch einen zweiten, klassischen Weg zur Zeitmessung, der auch noch XT´s offensteht: Die Manipulation des TimerChips 8253. Da altbewährt, ist zu diesem Chip Literatur bis in´s letzte IC-Gatter vorhanden. Dort erfährt man, daß die 18,2 Hz des Systemtickers dadurch erreicht werden, daß man die größtmögliche 16Bit-Integerzahl in einen (Herunter-)Zähler schreibt. Ist dieser Herunterzähler bei Null angekommen, wird ein Zeitgeberinterrupt ausgelöst. Schreibt man einen kleineren Wert in den Zähler, so wird eben häufiger der Zeitgeberinterrupt ausgelöst.

Das ganze funktioniert erfreulicherweise Rechnermodell-abhhängig (je Pentium, desto kürzer) hinunter bis zu 10µSekunden. Der Rechner löst dann fast nur noch Zeitgeber-Interrupts aus, aber wenn´s denn der Sache dient...

Damit war das Problem der Zeitauflösung erschlagen, der geplante Einsatz eines Pentium/60 ermöglichte Zeitauflösungen bis zu 50 µSekunden in Forth-Hochsprache!!

Da man die schnell zu erfassenden Daten erst einmal haben muß, gehörte die Aufmerksamkeit nun dem parallelen Port des PC; auch dieser ist hinreichend in Büchern dokumentiert und in seinen Besonderheiten in EDV-Magazinen diskutiert worden. Mangels Pusteblumenmodell mußte der Drucker herhalten, glücklicherweise ein Tintenstrahler. Streng nach Hardware-Handbuch wurden Routinen entworfen, die über Portzugriffe einzelne Bits der parallelen Schnittstelle setzen und lesen konnten. Zudem wurde für diese Bits eine Anzeige entworfen, um Veränderungen an der Schnittstelle sofort während des Programmierens auf dem Monitor sehen zu können.

Getestet wurden die Routinen, indem wir die Druckerschnittstelle zum Drucken benutzen. Und ein "Hallo Welt" auf dem Drucker mittels direktem Schnittstellenzugriff reichte uns als Bestätigung für die Qualität unserer Arbeit.

Oft steht der Begriff "Echtzeit" für "Kurzzeit". Bei der Steuerung der Pusteblume war abzusehen, daß zum Erfassen von Signaländerungen (Sie erinnern sich an die vier Himmelsrichtungen?) die parallele Schnittstelle möglichst gleichmäßig abgefragt werden mußte. Damit wurde "Echtzeit" synonym mit "Genauzeit". Obwohl wir den Systemticker drastisch erhöht hatten, wurde der Zeitgeberinterrupt weiterhin gleichmäßig, nur schneller ausgelöst; so bot es sich an, die Portabfrage hier einzuhängen. Nun bekommt ein Forth-Programmierer beim Anblick von Assemblercode nicht unbedingt eine Gänsehaut, aber Forth-Hochsprache ist nun mal transparenter, weniger fehleranfällig und einfach schön. Für das F-PC existiert eine Lösung für Interrupts in Hochsprache von Prof. Hendtlass, Australien, bei deren Einsatz man seinem PC wieder etwas näher kommt. Da merkt man schnell bei Rechnerstillstand, daß die Tips (wie STACKS 0,0 in CONFIG.SYS) aus der EDV-Regenbogen-Presse nicht immer etwas taugen und ein Pentium sowieso ein ganz anderes Wesen ist. Die Hendtlass-Lösung läuft mit eingerichteten Stacks zwar bis hin zum 486er, aber nicht mehr auf dem Pentium. Bleibt nur noch bitteres Brot: Interruptprogrammierung in Assembler. Öh?!

Da haben wir lieber ausprobiert, ob man statt einer eigenen Dienstroutine im Zeitgeberinterrupt nicht ganz auf Interruptdienste verzichtet und lieber die parallele Schnittstelle in gleichmäßigen Abständen selbst abfragt. Fazit: Ein Pentium kann eine Schnittstelle sehr schnell, sehr oft und sehr gleichmäßig abfragen (pollen). Damit war Abfragen und Erkennen der Pusteblume-Daten erledigt.

Nun ist das ständige Abfragen eines Ports eine nervige Aufgabe für den Programmierer; das überträgt man besser als selbständige Aufgabe an den Rechner. So kommt das Forth-Multitasking in´s Spiel, damit der Rechner die ganze Zeit von alleine pollt, aber trotzdem noch für andere Dinge zu haben ist.

Auf der Echtzeit:

Wie nervös man doch sein kann! Die anderen schienen einigermaßen ruhig. Wir waren es nicht. Da war das C-Team, das auch schon mal in den Vorjahren ziemlich weit vorne war. Und dann andere, die sehr seriös aussahen. Konkurrenz war ausreichend in Sicht.

Als der Wettbewerb dann losging, waren wir aber ganz cool und ruhig, fast schon wie echte Profis. Ullrich Hoffmann führte das Modell vor: Überraschung!

Nix mit Thermik. Vom Teelicht brauchte man nur die dünne Alu-Hülle (gut, daß wir ein schweres Teelicht aus Glas hatten; und noch einen zweiten Wachseinsatz als Reserve!). Es gab ein quadratisches Brett mit je einem Lüfter an den Kanten. In der Mitte rotiert ein Galgen, so daß er eine sich drehende Lichtschranke bildet. Die Aufgabe: Mit der rotierenden Lichtschranke die Position des Teelichtes erfassen und die Lüfter derart steuern, daß das Teelicht sich auf dem Brett im Kreis bewegt. Drei mal herum!

Ein Pappstreifen, der fest mit der Achse des Galgens verbunden war, erzeugte über eine zweite Lichtschranke bei jeder Umdrehung ein Indexsignal.

Damit war nun auch die Bedeutung der Pinbezeichnungen geklärt:

- N,S,O,W zum Einschalten jeweils eines der Lüfter,
- M zum Einschalten des Galgenmotors,
- RS zur Abfrage der rotierenden Lichtschranke und
- RI zur Abfrage des Indexsignals.

Damit hatten wir nicht gerechnet. Aber trotzdem: Los ging´s!

Leider hatten durch die Verlosung der Pusteblumen manche Gruppen ihr Gerät früher als andere, und während wir noch aufgeregt unser Gerät zum Tisch transportierten, konnte man schon drehende Galgen sehen. Und das, wo es doch um Sekunden geht!!

Obwohl wir uns das Gerät ja ganz anders vorgestellt hatten, war die Joystick-Idee ein Volltreffer! Spannung anschliessen, ein Druck auf den Feuerknopf und schon drehte sich der Galgen! Recht, links, vor und zurück und die Lüfter schnurrten. Damit konnte man das Verhalten des Gerätes ganz gut studieren.

Jetzt die Eingänge lesen. Fehlanzeige! Lichtschranke zu oder offen, am Pin des Printerport rührte sich nichts. An dieser Stelle wären wir eigentlich am Ende gewesen. Doch dann hat uns der Mitveranstalter Andreas Dobbertin mit ein wenig Messen an der richtigen Stelle wieder auf die Beine geholfen. Ergebnis: Die Treiber der Pusteblume hatten nicht genug Leistung, um die Pins am Port meines super-günstig-Rechners auf Low zu ziehen! Je nach Zustand der Lichtschranke schwankte die Spannung zwischen 5V und 2.5V, also zwischen High und High. Wieder wären wir am Ende gewesen, wenn nicht derselbe Veranstalter in weiser Voraussicht Widerstände im Gepäck gehabt hätte. Ein wenig Löten, und schon gab es 3.5V und 1.5V. Das war am Printerport zu erkennen.

Erst eine dreiviertel Stunde war rum, und schwupp, schon konnte es losgehen. Mit Sorge konnten wir beobachten, wie an den anderen Tischen die Aluhütchen schon hübsch durch die Gegend rutschten! Na wenigstens hatten wir schon mal die Ausrede fertig!

Jetzt erstmal kurz überlegen. Wenn man das Brett in zwölf Winkelsegmente aufteilt (Warum gerade zwölf? Ich sah auf die Uhr.) und für jedes Segment eine bestimmte Lüfterkonfiguration einschaltet? Mit Sicherheit das einfachste Verfahren. Also machen wir´s. Wenn´s nicht geht, können wir uns immer noch etwas

Komplizierteres ausdenken.

Doch woher weiß man, in welchem Segment sich das Teelicht befindet?? Ganz einfach! Mit dem Indexsignal mißt man die Zeit, die der Rotor für eine Umdrehung braucht. Dadurch hat man ein willkürliches Winkelmaß, aber noch ohne Ursprung. Dann stellt man das Teelicht an eine definierten Position (intuitiverweise zum 'Nord'-Lüfter), und schaut nach, zu welchem Zeitpunkt der Rotordrehung das Teelicht gesehen wird. Und das ist dann der Nullpunkt! Die Einheit ist 'Ticker', da die wirkliche Zeit ja gar keine Rolle spielt!

Wir stellen unseren Ticker so ein, daß eine Umdrehung so etwa 600 dauert. An sowas denkt man, wenn man mit 16 Bit herumkaspern muß. Das Teelicht in den Norden stellen und warten, bis es von der Lichtschranke erfaßt wird. In der Gegend von 50 Tickern war es meist soweit. Also nehmen wir die Tickerposition des Teelichtes, ziehen die 50 ab, skalieren mit 12/600 und schon haben wir unser Segment. Ach nein, wenn die erhaltenen Zahl negativ ist, wird nochmal zwölf addiert, wegen der Periodizität.

Dank unserer Vorbereitung war das schnell gemacht. Die Hintergrundroutine, die gleichmäßig Ticker zählte und beim Ändern der Signalleitungen DEFER-Worte ausführt, hatten wir schon. Da wurden also nur noch kleine Worte reingehängt, die sich die Umdrehungsdauer und den Nullpunkt merken konnten. Fertig.

Damit hatten wir das Gerät quasi kalibriert und es war nun ebenso einfach, die Winkelposition des Teelichtes zu bestimmen. Was nun?

Da es nur zwölf mögliche Lüfterkonfigurationen geben kann (zwölf Segmente!), haben wir einfach eine kleine Tabelle gemacht. Dort haben wir für jedes Segment und jeden Lüfter den passenden Zustand eingetragen.

Jedesmal, wenn nun eine neuen Teelichtposition gemessen wurde, also einmal pro Umdrehung, wurden einfach mit Hilfe der Tabelle die richtigen Lüfter ein- oder ausgeschaltet. Bingo!

Dieses Programm war recht schnell fertig. Jetzt ging es aber erst los! So ein Teelicht auf einem Brett ist nämlich ein ziemlich eigenwilliges Ding. Also experimentiert man mit allen möglichen Segeln oder der Lüfterstärke! Ob wir wirklich die Lüfterstärke per Pulsweitenmodulierung steuern müßten? Ach was, erstmal probieren!

Wir spielten etwas mit dem regelbaren Netzteil, bis wir eine günstige Spannung für die Lüfter gefunden hatten, bei der das Teelicht sich vorsichtig bewegte. Und tatsächlich: Das Teelicht fuhr unter Zeugen dreimal um den Mast herum! Leider waren es die falschen Zeugen. Als wir die Jury gerufen hatten, fuhr unser Teelicht einmal herum, dann irgendwo in die Ecke und blieb dort liegen. Mist.

So ging es dann eine ganze Weile. Vor uns gab es plötzlich Applaus! Das erste Team hatte es geschafft. Die Rostocker von FORTecH. Na ja, wer sonst.

Ob wir doch das Programm aufwendiger gestalten müssen? Vielleicht die Lüfter in steigender Folge immer länger laufen lassen, bis man eine Bewegung erkennt? Die Lüfter laufen langsam an. Damit wäre man sehr feinfühlig in der Dosierung. Ist ja einfach zu machen. Auf der anderen Seite: Wir sind immer so nah dran! Also weitertunen!

Hinter uns gibt es Applaus! Der Vorjahressieg war wohl doch kein Zufall.

Die Wettbewerbszeit schreitet fort, und die Anforderung wird von drei auf zwei vollständige Umkreisungen reduziert. Das hatten wir oft! Bei der nächsten Gelegenheit holen wir sofort die Jury und die Statistik läßt uns nicht im Stich: Brav fährt das Ding zweimal rum, zwei-dreiviertel-mal sogar. Wieder Applaus, diesmal um uns. Es ist geschafft und wir sind in den Medaillenrängen!! Stolz und Erleichterung. Das war gut, das hat richtig hingehauen. Wir sind die Größten! Denken wir.

Später fanden wir die beste Lösung: Alle Lüfter voll an, und das Teelicht fährt, vom rotierenden Mast gelegentlich leicht angestoßen, rund und rund und rund...

Wir haben einen ADA-Compiler gewonnen. Wahrscheinlich richtig teuer. Sieht so aus wie ein Compiler von 1980. Dafür langsam, undurchschaubar und verdonglet (ADA-Compiler zu verkaufen!!). Das ist natürlich so, als ob man jemandem, der gerade ein Autorennen gewonnen hat, ein Dreirad gibt.

❑

Einladung zur Forth Jahrestagung '95

vom 21. bis 23. April 1995 in Berlin
Hotel am Dämeritzsee, Kanalstraße 38/39 in 12589 Berlin-Köpenick

Organisation der Forth Jahrestagung:
Bernd Hinze
Waiblinger Weg 6
12487 Berlin
Telefon: 0 30 / 6 36 90 96 (ab 17:00 Uhr)

Einladung zur außerordentlichen Mitgliederversammlung am Sonntag, den 23. April 1995

Tagesordnungspunkte:
1. Rechenschaftsbericht des Direktoriums
2. Kassenbericht und Bilanz 1994
3. Aussprache, Entlastung, Wahl des Direktoriums
4. Neue Ziele der Gesellschaft
5. Änderung des Wahlrechtes für fördernde Mitglieder
6. Verschiedenes

Ergänzende Tagesordnungspunkte sind bis zum 1. März über das Forth-Büro dem Direktorium einzureichen.

Ein Anmeldeformular liegt dieser VD bei.

Der totale Patch

von Egmont Woitzel und Udo Schütz
FORTecH Software GmbH

In einem Standard-Forth-System führt das mehrfache Definieren von Worten zu einem Verdecken der jeweils zeitlich vorher in demselben Vokabular definierten gleichnamigen Worte. Typischerweise meldet sich das System bei so einer Redefinition auch, um die Verwirrung in Grenzen zu halten. Das sieht beispielsweise so aus:

```
1 CONSTANT FIRST↵  okay
FIRST .↵ 1  okay
2 CONSTANT FIRST↵
FIRST gab's schon  okay
FIRST .↵ 2  okay
```

Allerdings wirkt diese Überdeckung nur für den Textinterpreter und keinesfalls an den Stellen im Fadencode von :-Worten, wo das früher definierte Wort mal benutzt wurde. Diese bleiben von der Überdeckung unbehelligt, der Code für das ältere Wort existiert ja auch noch ganz unverändert. Also passiert zum Beispiel folgendes:

```
1 CONSTANT FIRST↵  okay
: SECOND FIRST . ;↵ okay
FIRST .↵ 1  okay
SECOND↵ 1  okay
2 CONSTANT FIRST↵
FIRST gab's schon  okay
FIRST .↵ 2  okay
SECOND↵ 1  okay
```

Dieses Verhalten ist in der Regel auch erwünscht, man möchte ja den früher definierten Code nicht ständig wieder durcheinanderwirbeln. Aber manchmal wäre es doch schon schön, wenn man bei einer Redefinition das Verhalten der Benutzer der früheren Version gleich mitändern könnte. Besonders häufig passiert das beim Debuggen. Das sitzt man beim Ausprobieren eines Programms, das - guter Forth-Stil - aus vielen kleinen aufeinander aufbauenden Worten bestehen, richtig schön top-down entworfen und bottom-up dahincodiert. Beim Testen hat man auch sein Bestes gegeben, aber dann passiert es doch. Man entdeckt einen Fehler in einem der Worte, die ganz früh geladen werden, weil sie ganz oft benutzt werden. Und dann wünscht man sich den *totalen Patch*. Es wäre so einfach: irgendwie in die Betriebsart *Patchen* schalten, den korrigierten Sourcecode im Editorfenster markieren und diesen per Knopfdruck neu compilieren. Das wäre so viel schneller als das ganze Programm zu FORGETten und alles noch mal neu zu übersetzen.

Irgendwann kommt dann der Punkt, wo man sich sagt, eigentlich könnte man das ja mal probieren - den *totalen Patch*. Auch wenn es nicht ganz fein ist, Zeit sparen kann man wahrscheinlich ja genug damit. Tja und dann reicht ein kleiner Anstoß und dann probiert man es wirklich. Es ist ja im Prinzip auch alles ganz einfach. Eigentlich genügt es, wenn man während der Definition eines neuen Wortes beim Feststellen einer Überdeckung dafür sorgt, daß die alte Codefeldadresse so verändert wird, daß sie auch das neu zu definierende Wort aufruft. Das geht zum Beispiel mit einem Stück Assemblerprogramm, das so ähnlich wie EXECUTE funktioniert. In der ersten Euphorie schätzt man den Aufwand auf höchstens eine Stunde. Allerdings kommt dann wie üblich eins zum anderen und es wird doch ein halber Programmiertag draus. Dann kann man aber stolz mit +REDEFINE einen universellen Patcher aktivieren und siehe da:

```
1 CONSTANT FIRST↵ okay
: SECOND FIRST . ;↵ okay
FIRST .↵ 1  okay
SECOND↵ 1  okay
+REDEFINE↵  okay
2 CONSTANT FIRST↵ Codefeld
39414 von FIRST überschrie-
ben  okay
FIRST .↵ 2  okay
SECOND↵ 2  okay
```

Auch SECOND bemerkt jetzt die Änderung von FIRST!

Wie funktioniert nun diese Zauberei? Da man ganz tief im System herumstöbern muß, wird man den Patcher kaum portabel implementieren können. Der abgedruckte Quelltext dürfte daher ausschließlich auf comFORTH für Windows laufen (die einzige für comFORTH 3.0 nötige Änderung kann man im Sourcecode finden). Allerdings sollten die Prinzipien leicht auf andere Systeme wie F-PC übertragen werden können.

Beginnen wir zunächst damit, uns den Aufbau der noch unberührten Worte FIRST und SECOND unter comFORTH für Windows anzusehen (vergleiche Bild 1). Als erstes fällt die Trennung in verschiedene Segmente auf. Im Headersegment HSEG befinden sich die Köpfe der Worte. Diese beginnen mit zwei Zeigern, die von FORGET benötigt werden, um Speicher im Fadencode- und Assemblersegment freizugeben. Danach kommt das Linkfeld, das auf den Kopf des Vorgängerwortes in diesem Hashzweig zeigt. (Apropos Hashzweig: Um die Suche von Worten zu beschleunigen, besitzt in comFORTH jedes Vokabular nicht nur eine Linkkette, sondern 8 verschiedene. Zu welcher ein Name gehört, wird aus seiner Länge und dem ersten Zeichen ermittelt.) Es folgen der Name des Wortes als abgezählte Zeichenkette sowie der Zeiger auf das Codefeld. Alter Tradition gemäß dienen die oberen Bits des Längenbytes zur Speicherung besonderer Informationen, richtig benutzt wird aber nur Bit 6 durch IMMEDIATE. Bit 7 ist immer auf 1 gesetzt, damit man gegebenenfalls das Wörterbuch auch "von hinten" aufräufeln kann, was allerdings nur funktioniert, wenn ausschließlich 7-Bit-ASCII-Zeichen in den Namen verwendet werden. Bei aufmerksamer Betrachtung von SECOND fällt außerdem noch auf, daß comFORTH den erzeugten Code sorgfältig auf Wortgrenzen ausrichtet. So wird zwischen dem letzten Zeichen des Wortnamens und dem Zeiger auf das Codefeld gegebenenfalls ein Leerzeichen eingefügt, so daß (FIND) seine Vergleiche ohne Geschwindigkeitsverlust wortweise ausführen kann.

Das Codefeld sowie alle Parameterfelder befinden sich im Datensegment DSEG. Da comFORTH indirekt gefädelten Code benutzt, enthalten die Codefelder einen Zeiger auf den zugehörigen Maschinencode, der im Assemblerseg-

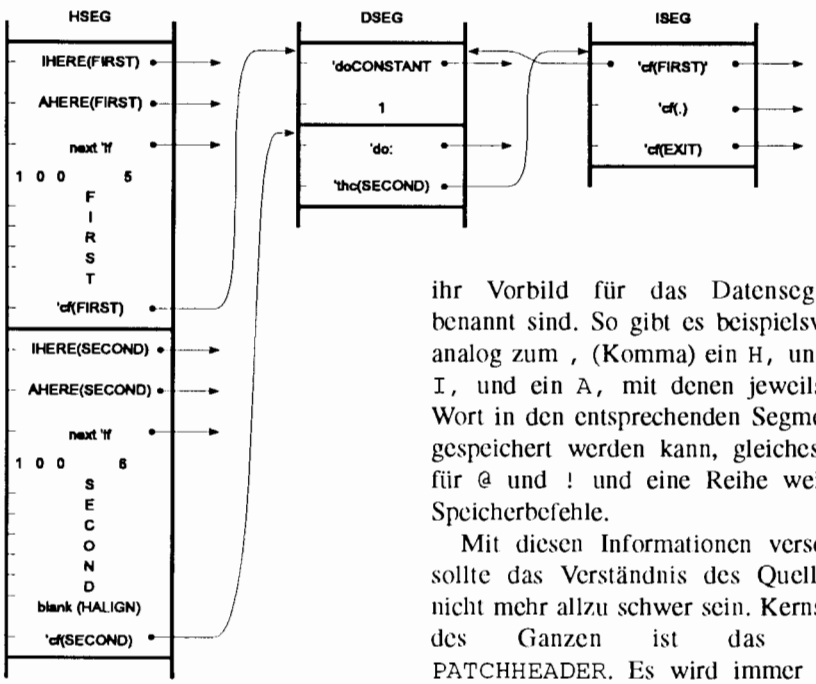


Bild 1: Wörterbuchaufbau vor der Redefinition von FIRST

ment ASEG gespeichert wird. CONSTANTen verweisen daher auf doCONSTANT, Doppelpunktworte auf do:. Hinter dem Codefeld beginnt dann das Parameterfeld, auf dem bei FIRST der Zahlenwert selbst liegt. Abweichend von üblichen Implementierungen enthält das Parameterfeld von Doppelpunktworten nicht den Fadencode, sondern nur einen Zeiger darauf. Um die Auslastung des Speichers zu verbessern, wird nämlich dieser in einem eigenen Instruktionssegment ISEG gespeichert. In üblichen Programmen wächst dieses Segment am schnellsten. Durch die Trennung von Daten- und Instruktionssegment kann comFORTH größere Programme übersetzen und ausführen als andere segmentierte 16-Bit-Forth-Systeme, die nur die Köpfe und den Maschinencode auslagern. Ein netter Nebeneffekt dieser Implementation ist, daß Doppelpunktworte und Brodie'sche DOER miteinander identisch sind. Dadurch können sie unter Verwendung von MAKE einfach geändert werden, was im Quelltext z. B. beim Patch von HIDE, REVEAL und IMMEDIATE ausgenutzt wird.

Für den Speicherzugriff auf die neuen Segmente HSEG, ISEG und ASEG implementiert comFORTH eine ganze Reihe spezialisierter Worte, die mit Ausnahme eines einbuchstabigen Präfix wie ihr Vorbild für das Datensegment benannt sind. So gibt es beispielsweise analog zum , (Komma) ein H, und ein I, und ein A, mit denen jeweils ein Wort in den entsprechenden Segmenten gespeichert werden kann, gleiches gilt für @ und ! und eine Reihe weiterer Speicherbefehle.

Mit diesen Informationen versehen, sollte das Verständnis des Quelltexts nicht mehr allzu schwer sein. Kernstück des Ganzen ist das Wort PATCHHEADER. Es wird immer dann aufgerufen, wenn beim Bauen eines neuen Wörterbuchkopfs (vgl. die Redefinition von HEADER) eine Überdeckung festgestellt wird, das Redefinieren aktiv ist und das betreffende Wort nicht aus dem Kern stammt. Die letztgenannte Bedingung ist zwar nicht zwingend nötig, schützt den Benutzer aber doch wenigstens ein bißchen vor verstörtem Verhalten des Systems beim Redefinieren von Kernbestandteilen.

Die Implementation von PATCHHEADER selbst ist absolut geradlinig. Die Linkfeldadresse des gefundenen und zu patchenden Wortes wird für (;CODE) in der Variablen LAST gespeichert. Dann wird eine Systemnachricht über die erfolgte Redefinition ausgegeben, wobei man ein letztes Mal die gepatchte Codefeldadresse zu Gesicht bekommt. Im Anschluß daran wird mit einer Menge A,'s ein Stück Maschinencode generiert, das wie ein EXECUTE für die neue Codefeldadresse (zu diesem Zeitpunkt das HERE des DSEG) wirkt. Dazu muß diese in comFORTH in die CPU-Register AX und DI geladen werden. Da der Inhalt des Codefelds einen Zeiger auf Maschinencode darstellt, kann anschließend einfach über [ES:DI] gesprungen werden. Die Adresse dieses Codestückchens wird dann nur noch als neue Codeadresse in das alte Codefeld eingetragen. Auf diese Weise brauchen die Definitionsworte von altem und neuem Wort nicht übereinstimmen. Auch das Überladen von CODE-Definitionen, die im DSEG nur das Codefeld und kein Parameterfeld besitzen, ist problemlos möglich. Schließlich und letztendlich muß nur noch die im Kopf gespeicherte Codefeldadresse auf die neue gebogen werden, da sonst) (;CODE) bei der Zuweisung des Ausführverhaltens danebenschlägt. Das neue Code- und Parameterfeld von FIRST wird nach dem Abschluß von HEADER durch CREATE bzw. CONSTANT angelegt. Bild 2 zeigt den Zustand des Wörterbuchs nach erfolgter Redefinition. Falls man ein Forth-System ohne Segmentierung verwendet, wird man im Gegensatz zu der implementierten Lösung um ein doppeltes Anlegen des Wörterbuchkopfes wohl nicht herumkommen, der prinzipiellen Realisierbarkeit steht aber nichts im Wege.

Der verbleibende Quelltext ist nicht mehr besonders spannend. Der erste Abschnitt dient nur dem Ein- und Ausschalten des Patch-Modus. Daran folgen die Redefinitionen von HIDE und REVEAL, welche in comFORTH das Verbergen fehlgeschlagener Definitionen im Wörterbuch steuern und im Fall einer Redefinition unwirksam gemacht werden müssen (das PREDEFINED sorgt jeweils für den Aufruf des vorher zugeordneten Codes). Eine nette Überraschung bereitete NAME>" (wird von NAME. benutzt), das den Wörterbuchbereich ab HERE zum Umkopieren benutzt und dabei den von NAME gelieferten String zerstörte. Die Redefinition ist etwas seriöser und kopiert den Namen direkt auf den Kettenstapel. Die Definitionen von MAKEHEADER und HEADER entsprechen mit Ausnahme des Factoring weitestgehend dem Original, wovon man sich leicht mit SEE HEADER überzeugen kann. Und so stellt man am Ende fest: Eigentlich ist wirklich alles ganz einfach!

Bei Anwendung des Patchers gibt es verblüffend wenig Probleme. Die im Wörterbuch verbleibenden Restbestände unbenutzten Codes lassen sich leider nicht beseitigen, dazu würde man einen Heap als Wörterbuch benötigen. Der bei jeder Redefinition generierte Maschinencode läßt sich leicht verschmerzen, das Assemblersegment wird ohnehin am wenigsten belastet. Der einzige echte Seiteneffekt wurde bei WORDS beobach-

tet. Da sich WORDS bei der Auflistreihenfolge der Worte zwischen den verschiedenen Hash-Zweigen nach der Codefeldadresse richtet, "springen" redefinierte Worte ab und zu ein Stück "nach vorne". Aber deswegen auch noch WORDS zu patchen, war uns dann doch zuviel.

Viel Spaß beim Patchen!

PS: Praktiziert werden diese Techniken schon lange,in der VD 1/94 stellte Wolfgang Schemmert in " Aus ALT mach Neu" ein ähnliches Verfahren vor, auch im System HOLON von Wolf Weijgaard muß ein ähnliches Verfahren implementiert sein.

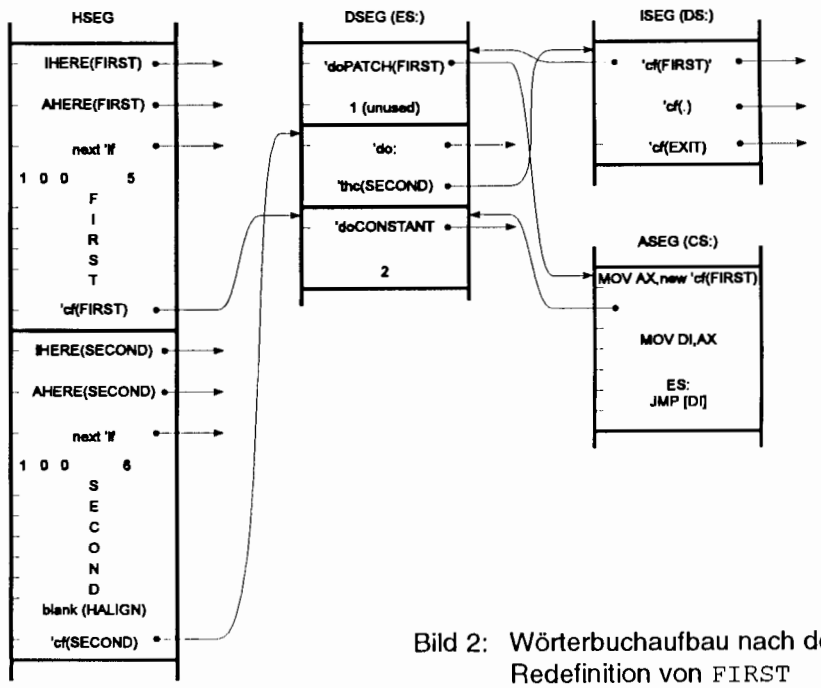


Bild 2: Wörterbuchaufbau nach der Redefinition von FIRST

Der Quelltext des universellen Patchers:

```
\ Universeller Patcher
\ E. Woitzel, U. Schütz
\ FORTecH Software GmbH 1994

ONLY STRING ALSO DICTIONARY ALSO COMFORTH ALSO FORTH DEFINITIONS

\ --- Zustandsverwaltung

2VARIABLE 'REDEFINE? ( ps: ===> addr )( die erste Zelle enthält ?t, wenn )
          ( Definitionen gleichen Namens hart redefiniert werden dürfen )
          ( die zweite Zelle enthält ?t, wenn das tatsächlich geschah )

: +REDEFINE ( ps: ===> )( schaltet in harten Redefinemodus )
          'REDEFINE? ON ;

: -REDEFINE ( ps: ===> )( schaltet in harten Redefinemodus )
          'REDEFINE? OFF ;

: REDEFINED ( ps: ===> )( markiert letzte Definition als Redefinition )
          'REDEFINE? 2+ OFF ;

: DEFINED ( ps: ===> )( markiert letzte Definition als Neudefinition )
          'REDEFINE? 2+ ON ;

: DEFINED? ( ps: ===> ? )( ?t, wenn letztes Wort eine Neudefinition war )
          'REDEFINE? 2+ @ ;

-REDEFINE   \ erst mal ganz normal bleiben
DEFINED               \ und auf Neudefinitionen stellen



\ --- HIDE und REVEAL-Verwaltung robust machen

MAKE HIDE ( ps: ===> )( nur bei Neudefinition Hiden )
          DEFINED? IF  PREDEFINED  THEN ;

MAKE REVEAL ( ps: ===> )( nur bei Neudefinition Revealen )
          DEFINED? IF  PREDEFINED  THEN ;


\ --- Seiteneffekte zu NAME vermeiden

: LSTR@ ( ps: sel off len ===> )( "s: ===> s )( auf Kettenstapel holen )
          255 UMIN DUP 1+ NEGATE "P+!        \ freihalten
          "P@ C!                             \ Länge
          DSEG "P@ COUNT LCMOVE ;

MAKE NAME>" ( ps: 'nf ===> )( "s: ===> name )( keine Funktionsänderung )
          ( es werden nur Seiteneffekte via HERE vermieden )
          HSEG SWAP HCOUNT 31 AND LSTR@ ;


\ --- Behandlung neuer Köpfe

HEX
: PATCHHEADER ( ps: 'nf ===> )( altes Wort umpatchen )
          DUP N>LINK LAST !       \ 'lf für (;CODE) setzen
          DUP DUP HC@ 2DUP        \ Längenbyte lesen
          BF AND HSEG ROT LC!     \ IMMEDIATE-Bit zurücknehmen
          1F AND                  \ Namenslänge bestimmen
          1+ + HALIGN             \ Adresse von 'cf
          DUP H@                  \ alte 'cf bestimmen
          MOOD>R >ERROR           \ ausführlich anmaulen
          ." Codefeld " U.        \ alles in der Fehlerfarbe
          ." von " SWAP .NAME
          ." überschrieben " R>MOOD
          AHERE                   \ special EXECUTE generieren
          B8 AC,  HERE A,         \ MOV AX,'cf
\         comFORTH für Windows Beta II: Forth-DSEG steht in ES:
          F88B A, 26 AC, 25FF A,  \ MOV DI,AX  JMP [ES:DI]
\         comFORTH 3.0 unter MS-DOS: Forth-DSEG steht in SS:
\         F88B A, 36 AC, 25FF A,  \ MOV DI,AX  JMP [SS:DI]
          OVER H@ !               \ 'ca in altem cf patchen
          HERE SWAP H!            \ Zeiger im Header auf neues cf biegen
          REDEFINED ;             \ HIDE und REVEAL beide Augen zudrücken
```

Der Quelltext des universellen Patchers: (Fortsetzung)

```
: MAKEHEADER ( ps: 'hash 'str ===> )( neuen Kopf bauen )
          HALIGN HHERE H, AHERE H,          \ für 'FORGET'
          HHERE DUP LAST !                  \ 'lf für HIDE und REVEAL
          2 HALLOT                          \ lf reservieren
          SWAP DUP C@                       \ Namenslänge bestimmen
          1+ HHERE SWAP >HCMOVE             \ Name kopieren
          HHERE CACHE>                      \ Cacheplatz auspusten
          SWAP HLINK                        \ Linkfeld einbinden
          80 HHERE DUP HC@ 1+ HALLOT        \ Namensplatz reservieren
          HCSET                             \ und Traverse-Bit setzen
          HALIGN                            \ ausrichten
          ALIGN HERE H,                     \ Zeiger auf Codefeld
          DEFINED ;                         \ HIDE und REVEAL aktivieren
DECIMAL

\ --- im Kernel herumbiegen, Achtung: hier nicht Modulkomprimieren!!!

MAKE HEADER  ( ps: ===> )( ib: name )( baut oder patcht Wörterbucheintrag )
          CURRENT @ CURRENT 2+ !                \ Vokabular für HIDE/REVEAL
          BL NAME                               \ neuen Namen lesen
          CURRENT @ +HASH 2DUP @ (FIND)         \ in CURRENT suchen
          IF        \ im aktuellen Vokabular gefunden
                    DUP HFENCE @ U>             \ Finger weg vom Kern
                    'REDEFINE? @ 0<> AND
                    IF        \ es soll redefiniert werden
                              PATCHHEADER       \ alten Kopf verbiegen
                              2DROP EXIT  \ aufräumen und verschwinden
                    THEN      \ ps: 'str 'hash 'nf
                    MOOD>R >ERROR DUP .NAME R>MOOD      \ 'gab's schon' Meldung
                    1 MESSAGE
          THEN      \ ps: 'str 'hash 'str|'nf
          DROP SWAP                             \ ps: 'hash 'str
          MAKEHEADER ;                          \ neuen Kopf anlegen
```

Brief aus der Provinz Moers, am 29. 11. 1994

Ich hatte lange nichts von der FG gehört, keine VD mehr gesehen und bei den DFÜ 'treibenden' Forthern nur die gleiche Ratlosigkeit gefunden, die ich selber gefühlt hatte. Ich hatte Briefe, Aufsätze und Berichte an die Redaktion der VD geschrieben, ohne die bisher übliche Antwortkarte zurück zu bekommen. Vereinskameraden, die ich seit Jahren immer wieder auffordere die VD selbst mit einigen Zeilen zu schmücken, haben mich angeschrieben und angerufen und sich bitter beklagt, daß ihre Briefe an die VD ohne jedes Echo geblieben sind. So macht 's natürlich keinen Spaß.

Und es macht auch keinen Spaß, wenn man immer wieder, meist mit einem hämischen Grinsen verbunden, gefragt wird, ob es denn die Forthgesellschaft überhaupt noch gibt.

Als ich schon fast bereit war aufzugeben, rief meine Frau mich eines späten Herbsttages im Büro an, um mir mitzuteilen, daß "die Zeitung, auf die Du schon so lange wartest, heute gekommen ist." Na, das war eine gespannte Heimfahrt. Ich will gar nicht darüber klagen, daß die letzte Ausgabe der Zeitschrift meines Vereins 'etwas schwach auf der Brust' scheint. Ich habe mich einfach gefreut, daß sie überhaupt noch gekommen ist - und das es die Forthgesellschaft offensichtlich doch noch gibt !

Etwas verärgert bin aber schon noch.

Und dann schreibt mich unser aller, neuer Direktor via DFÜ an. Der Jörg möchte von mir einen 'Brief aus der Provinz' haben. Vermutlich kurzfristig. Die nächste Ausgabe der VD soll ja noch in diesem Jahr herauskommen. Und das Direktorium überlegt, in den nächsten VDs einen Anfängerkurs anzubieten. Die Moerser machen so etwas doch schon seit Jahren. Ob es dazu nicht 'druckbares' Material gibt, möglichst in Fortsetzungen... ?

Menschenskind, Jörg ! Das habe ich alles schon mehrfach der VD geschickt. Aber bitte - ich stell's gerne zur Verfügung - und wenn es noch 10 Mal sein sollte. (...kann man den Seufzer 'hören', den ich gerade ausstoße ?)

Na gut - an die Arbeit. Daß ich den Brief, sehr kurzfristig (!), sofort in das XPoint Terminal 'hacke' ist doch Ehrensache. Die Kurstexte kann Jörg, oder wer auch immer die VD damit füllen will, gerne haben - und ich versuche meinen Ärger endgültig herunter zu schlucken und mich auf das zu konzentrieren, was vor uns liegt. Und das ist in Moers, wie immer, reichlich.

Zunächst kommen wir ganz gut mit unserer Graphik-Bibliothek voran. Ich habe bereits im letzten Brief... kurz davon geschrieben, daß wir eine 'spezialisierte' Graphikbibliothek für das ZF und die VGA-Karte im Modus 620 x 480 Pixel bei 16 Farben 'basteln'. Erklärtes Ziel ist es, einige Primitives zu haben, die jeweils mit maximaler Geschwindigkeit arbeiten.

Wir gehen dabei so vor, daß wir in der Gruppe (nach dem Anfängerkurs, den Michael Major natürlich fortsetzt) gemeinsam Graphikroutinen 'entwerfen', in HigLevel kodieren und austesten. Edgar Prellwitz, der einige Zeit in England Informationswissenschaften studiert hat, und Ulrich Richter, der als Bauingenieur die notwendige Mathematik gewissermassen 'aus dem Ärmel' beisteuert, sind dabei ebenso unschätzbare Hilfen wie die Christel Schulz, die grundsätzlich 'vorweg arbeitet' und für den nötigen Antrieb sorgt. Natürlich sind an dieser Arbeit noch ein paar andere Kameraden beteiligt, die allesamt dafür sorgen, daß ich am Ende die jeweilige Routine mit ZF's Assembler 1:1 auf die VGA Karte portiere.

Vom Erfolg dieser Arbeit konnten sich Rolf Kretzschmar und Klaus-Peter Schleisiek kürzlich überzeugen. Die beiden haben bei uns die 'schnellste Grafik diesseits und jenseits des Rheins' zu sehen bekommen ;-)

Wir machen damit natürlich weiter, ebenso wie mit Michael's Kurs. Aber wir wollen noch mehr machen. Mit Rolf und Klaus wollen wir in Moers so etwas wie ein 'Niederrhein Forum Forth' ins Leben rufen. Wir stellen uns dazu regelmäßige Veranstaltungen vor, die jeweils irgendwo am Niederrhein stattfinden und Forther zusammenrufen sollen, die in einem Radius von ca. 250 km um Moers herum leben. Das schließt unsere Nachbarn in den Niederlanden natürlich ebenso mit ein, wie die Forther in Belgien. Diese Veranstaltung soll, wenn wir das so 'hinkriegen', und wenn der Bedarf entsprechend ist, bis zu vier Mal im Jahr stattfinden, jeweils ein Wochenende dauern und 'rund um Forth' sowohl den 'notorisch ernsthaften' etwas bieten, als auch denen, die einfach nur gerne mal zusammensitzen und 'quatschen'. (Wer Hans Dieter Hüsch kennt, sollte allerdings vor unserem 'Quatschen' gewarnt sein ;-))

Wir haben damit begonnen, eine erste Veranstaltung für den Sommer 1995 zu planen. Die soll, wenn mit den Räumlichkeiten alles klappt, in Steinwurfweite vom Rhein stattfinden, gewissermaßen in einer Art Camp. Mehr will ich dazu aber noch nicht verraten, außer, daß wir uns bemühen werden, die Kosten so gering wie nur möglich zu halten, was vor allem die jüngeren Forther (aber nicht nur) dazu ermuntern soll, sich eine Teilnahme zu überlegen. Interessenten sollten sich in jedem Fall schon jetzt bei mir melden. Das würde die ganze Sache planbarer machen...

So, das war der ad hoc Brief, via DFÜ an den Jörg 'gepostet'. Ich wünsche, im Namen der Forthgruppe Moers allen Forthern ein frohes Fest und ein glückliches Neues Jahr. Letzteres wünsche ich ganz besonders der Forthgesellschaft, und, wie das in meinem Berufstand so üblich ist, einfröhliches

Glückauf
Glückauf
der Fritz

Anzeigen

Berlin wird Forth-Hauptstadt

Berliner lokales Treffen am 26.1.95

Forthige Zeiten stehen in Berlin bevor. Die diesjährige Forth-Tagung wird im April in Berlin stattfinden. Die Vierte Dimension ist nach Berlin umgezogen. Nur die Mitglieder, die die Forth-Gesellschaft in Berlin hat, kennen sich nicht, sind weg-, zu- und umgezogen oder weigern sich, vereinsmeierisch tätig zu werden. Daher laden wir alle Forth-Interessierten ein:

Treffen der Berliner Forthler:
Am Donnerstag, 26.1.1995 um 19.00 Uhr in der Rudower Chaussee 5 (eh. Akademiegelände). S-Bhf. Adlershof. Um Voranmeldung wird gebeten! Herr Haase (639 23 220) beantwortet auch Rückfragen. Bei Interesse findet eine Präsentation statt. Raum für gegenseitige Information und Meinungsaustausch vorhanden. Kekse bitte selber mitbringen.

FORTH-SYSTEME GMBH

Postfach 1103
D-79200 Breisach

Telefon (0 76 67) 5 51
Telefax (0 76 67) 5 55

WinFORTH

- UR/FORTH kompatibel
- Windows Funktionen werden voll unterstützt
- Erweiterte Debugging-Hilfsmittel
- Online Windows Hilfe
- Coprozessor Unterstützung möglich
- Software-Gleitkomma-Paket
- Viele Beispielprogramme
- Upgrades von UR/FORTH Systemen auf WinFORTH sind preisgünstig zu erhalten

UR/FORTH

- FORTH-83 Standard
- Für MS-DOS, OS/2, 80386
- Direkt gefädelte Code Implentationen mit dem obersten Stackwert im Register um größtmögliche Ausführungsgeschwindigkeit zu erreichen
- Segmentiertes Speichermodell mit Programm, Daten, Headers und Dictionary Hash Table jeweils in einem getrennten Segment
- Komplett gehashtes Dictionary führt zu extrem schneller Übersetzung
- Mächtige neue String Operatoren (Suche, Extraktion, Vergleich und Addition) sowie einen dynamischen String-, Speichermanager
- Kann mit Objektmodulen, die in Assembler oder anderen Hochsprachen erzeugt wurden, gelinkt werden
- Native Code Optimizer zur direkten Umsetzung in 80 x 86 Code im Lieferumfang

Eprom Emulator

- SRS-II – Serieller-ROM/RAM-Simulator
- Flex ROM plus – Lowcost Eprom-Emulator
- Econo ROM

ModuNORM

CPU-Steck-Module im Scheckkartenformat:

- 8 Bit z.B. 6303
- 16 Bit z.B. V25
- Highspeed RTX-2000/1
- 80C166
- C 167 CAN
- 68332
- Softwareunterstützung durch SwissFORTH
- Thermodrucker und Controller
- LCD Grafik-Controller

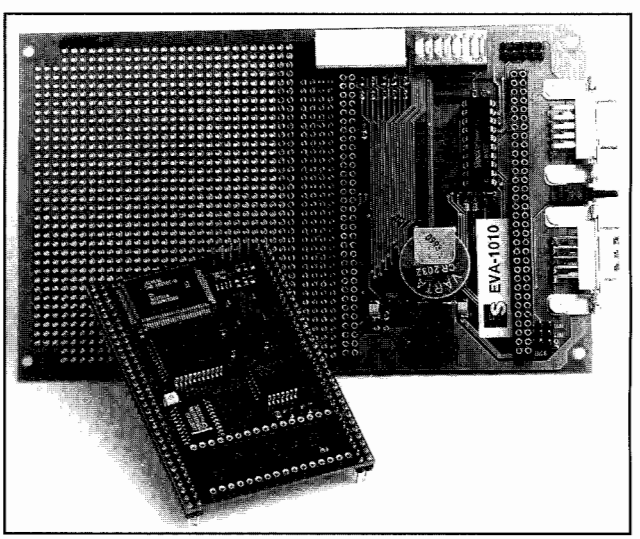
FORTH-83 Metacompiler

Der LMI FORTH Metacompiler wird mit komplettem Quellcode für ein ausführlich ausgetestetes, Hochgeschwindigkeits FORTH 83 Kern ausgeliefert, wobei Sie die Auswahl aus folgenden Zielprozessoren haben:

- 8086/8088
- Z80/HD64180
- 8080/8085
- 68000
- Z8
- 1802
- 6809
- 8096/97
- 68 HC 16
- 78310
- 8031/32/535
- 6303
- 6502
- V25
- 68HC11
- RTX 2000
- 80C166
- 68332

Sie erzeugen schnelle und kompakte Anwendungen, indem Sie Ihre Quellprogramme mit unserem Forth Nucleus zusammenstellen und ihn mit dem LMI FORTH Metacompiler übersetzen.

Scheckkartenmodule



CPU Module für Entwicklungen im Embedded Controller Bereich:

- 80386 EX
- 80C166-CAN
- Evaluations-Boards

Bitte fordern Sie unseren Produktkatalog und die Preisliste an. FORTH-Gesellschaftsmitglieder erhalten bis zu 10% Rabatt (artikelabhängig).