DIE PEARL-IMPLEMENTATION VON AEG-TELEFUNKEN FÜR UND AUF AEG 80-20

Dipl.Ing. S. Eichentopf
ATM Computer GmbH 7750 Konstanz

1. Einleitung

AEG-TELEFUNKEN hat von Anfang an - also seit 1969 - bei der Erarbeitung der Sprachdefinition von PEARL mitgewirkt.
Als sich die Sprachdefinition in der Mitte der 70er Jahre nach einer größeren Überarbeitung zu stabilisieren schien, hat AEG-TELEFUNKEN auf der Basis des bis dahin Erarbeiteten mit vermehrter Kapazität die Implementierung der neuen Sprache in Angriff genommen. Dabei wurden nachträgliche Änderungen, die im PEARL-Arbeitskreis einvernehmlich an PEARL vorgenommen wurden, jeweils möglichst umgehend in die Implementierung einbezogen, was durch das gewählte halbautomatische Implementierungs-Verfahren begünstigt wurde. So kann heute eine PEARL-Implementation angeboten werden, deren Sprachumfang auf dem neuesten Stand der offiziellen Full-PEARL-Sprachdefinition ist und der weit über Basic PEARL hinausgeht.

2. Sprachumfang

Bei der Auswahl des Sprachumfangs stand der Nutzen aus Benutzersicht im Vordergrund. Es wurde nur auf die Sprachelemente von Full PEARL verzichtet, deren Implementierung mit unverhältnismäßig hohem Aufwand möglich gewesen wäre oder die nicht auf ausreichend effiziente Weise mit dem vorhandenen Betriebssystem hätten realisiert werden können. Aufgrund dieser Randbedingungen wurde sehr frühzeitig ein zu implementierender Sprachumfang festgelegt, der gegenüber Full PEARL nur wenig eingeschränkt ist.
Die wesentliche Einschränkung ist wohl der Verzicht auf Subtasks, d.h., daß die Einführung von Tasks nur durch entsprechende Deklarationen "auf Modulebene" zugelassen ist, - eine Einschränkung, mit der der Benutzer jedoch gut leben kann. Eine genauere Abgrenzung des AEG 80-20-PEARL-Sprachumfangs gegenüber Basic PEARL und Full PEARL findet sich im Anhang der AEG-80-20-PEARL-Sprachbeschreibung /1/.

Außerhalb von Basic PEARL gibt es eine Reihe von Sprachelementen, die nicht nur den Programmierkomfort erhöhen und zur besseren Lesbarkeit und Selbstdokumentation der Programme beitragen, sondern die überdies die Effizienz der Programme wesentlich erhöhen können. Hierzu einige Beispiele:

- Speicheroptimierung durch Arrays,deren Indexgrenzen dynamisch auf Eingabewerte zugeschnitten sind, bei minimaler Laufzeiterhöhung
- Verminderung der Zahl erforderlicher Tasks durch Einplanungs-Listen bei Aktivierungsanweisungen zusammen mit der Standardfunktion ORIGIN zur Abfrage des auslösenden Einplanungselements
- Zuweisung von Strukturen und Arrays als Ganze statt nur elementweise:

```
DCL (AR1, AR2) (50)FIXED;
DCL (ST1, ST2) STRUCT [E1 TYP1, E2 TYP2, E3 TYP3];

/* FULL PEARL */                    /* BASIC PEARL */

AR1 := AR2;                         FOR I TO 50
                                    REPEAT  AR1 (I) := AR2(I);
                                    END;

ST1 := ST2;                         ST1.E1 := ST2.E1;
                                    ST1.E2 := ST2.E2;
                                    ST1.E3 := ST2.E3;
```

- Vermeidung von Indextransformationen durch beliebige untere und auch negative obere Arrayindexgrenzen,
- Verminderung der Zahl von Adreßberechnungen mit Hilfe von Identitätsspezifikationen (SPC ... IDENT...), z.B. bei Arrays von Strukturen:

```
TYPE STR STRUCT [E1 TYP1, E2 TYP2, E3 TYP3];
DCL A (12,10)STR;
DCL (I, J) FIXED;

/* FULL PEARL*/                     /* BASIC PEARL */
```

```
BEGIN
SPC AIJ STR IDENT(A(I, J));
AIJ.E1 := ...                    A(I,J).E1 := ...
... AIJ.E2 ...                   ... A(I,J).E2 ...
AIJ.E3 := ...                    A(I,J).E3 := ...
END;
```

3. Compiler

Der Compiler läuft auf AEG 80-20/4 und AEG 80-20/5 mit mindestens 96 KBytes, besser 128 KBytes Hauptspeicherausbau und Hintergrundspeicher mit wahlfreiem Zugriff (Kassettenplatte, Wechselplatte, Festkopfplatte).

Die Läufe des Compilers und die Compilertabellen belegen auf Hintergrundspeicher statisch ca. 350 KBytes. Dynamisch beim Compilieren werden zusätzlich, abhängig von den zu übersetzenden Programmen, zwischen 300 KBytes und 500 KBytes Hintergrundspeicher benötigt.

Der Compiler ist selbst außer ein paar Ein-/Ausgabe-Routinen in PEARL geschrieben. Er wurde einmal mit einem sogenannten bootstrap-Verfahren auf den Rechner gebracht und kann sich dort nun selbst übersetzen.

Der Compiler wird mit einem einfachen Kommando gestartet. Die zu übersetzenden Quellprogramm-Moduln können in Quelldateien auf Hintergrundspeicher vorgegeben werden, wo sie auch mit einem Texteditor bearbeitet werden können, oder sie können über Papierperipheriegeräte eingegeben werden. Die größten an einem Stück übersetzbaren Quellmoduln sind je nach Quellzeilenstruktur und anderen Programmeigenschaften 2000 bis 3000 Quellzeilen lang.

Der aus mehreren Läufen bestehende Compiler-"Oberteil" erzeugt als Zwischenprodukt eine Reihe von Listen sowie ein Abbild des Quellprogramms in umgekehrter Polnischer Nation, in der Arrayindizierungen, Strukturelementselektionen, Prozeduraufrufe usw. wie spezielle Operationen behandelt werden. Diese Schnittstelle zwischen Compiler-Oberteil und Codegenerator sowie die Formulierung des Compilers in PEARL machen den Compiler weitgehend rechnerunabhängig und damit übertragbar.

Bei den Läufen des Compileroberteils werden zur Analyse der Eingabeprodukte Bottom-up-Parser, genauer Bounded-Context-Parser verwendet. Wesentliche Bestandteile dieser Parser sind Parsertabellen, an Hand deren analysiert wird und die mit Hilfe von Parsergenerator-Programmen aus vorgegebenen (kontextfreien) Grammatiken automatisch hergestellt wurden.

Der Codegenerator des Compilers erzeugt direkt Bindemoduln - nicht Assembler -, die auf Hintergrundspeicher abgelegt werden oder über Papierperipherie ausgegeben werden können.

Der erzeugte Objektprogrammcode kann durch Parameter im Compilerstartkommando z.T. beeinflußt werden (mit/ohne Indexgrenzenprüfung, Prozeduren generell reentrant oder nicht).

Die Zeit für die Übersetzung von 1000 Quellzeilen eines übersichtlich geschriebenen, durchschnittlichen Programms (im wesentlichen eine elementare Anweisung pro Zeile) aus einer Hintergrund-Quelldatei in Bindemoduldateien auf Hintergrundspeicher liegt - ohne Protokollierungszeiten - bei AEG 80-20/4 unter 5 Minuten.

Der Compiler erzeugt beim Übersetzen ein Protokoll, dessen Umfang durch Parameter im Compilerstartkommando gesteuert werden kann. Zum maximalen Protokollumfang gehören folgende Teile:

- formatgetreuen Quell-Listing mit Zeilennummern
- Listing aller PEARL-Datenobjekte im Programm mit Angabe der Definitionsstelle und sämtlicher Aufrufstellen im Quellprogramm (Zeile, Spalte) sowie der Lage im Speicher des erzeugten Objektprogramms
- Code-Listing in assemblernaher Form mit Rückverweisen auf entsprechende Quellzeilen und -spalten
- Verzeichnis der erzeugten Bindemoduln mit statischen Längen
- Füllungsgrad bzw. Längen wichtiger Compilerlisten

Dazu werden ggf. genaue Fehlermeldungen gegeben, die - von wenigen gravierenden Fällen abgesehen - nicht zum Abbruch des Übersetzungsvorgangs führen, so daß mit einer einzigen Übersetzung eine ganze Reihe unabhängiger Fehler im Quellprogramm festgestellt werden kann.

4. Objektprogramme

Die Sprachelemente von PEARL, die nicht einfach durch inline-Maschinenbefehlsfolgen und nicht direkt durch entsprechende Betriebssystemdienste realisiert werden können, werden mit Hilfe der Moduln eines Laufzeitpakets realisiert, die die erforderliche Leistung entweder vollständig erbringen oder ggf. eine Abbildung auf geeignete Betriebssystemdienste vornehmen. Insbesondere werden auf diese Weise die Tasks des PEARL-Programms 1:1 auf vom Realzeit-Betriebssystem MARTOS-K verwaltete "Programme" bzw. Aktivitäten der Tasks auf Prozesse über diesen Programmen abgebildet.

Das Laufzeitpaket ist stark modularisiert. Beim Laden und Binden eines speziellen PEARL-Objektprogramms werden nur die hierfür erforderlichen Moduln des Laufzeitpakets hinzugeladen und gebunden, und zwar automatisch aus entsprechenden Bibliotheken.

Die vom Compiler erzeugten Bindemoduln des Objektprogramms können mit dem normalen im Dialog zu bedienenden Binder-Lader des AEG 80-20-Programmiersystems geladen und gleichzeitig gebunden werden, wobei bezüglich der Ablage im Speicher einige wenige Konventionen zu beachten sind. Bequemer ist das Laden und Binden mit Hilfe einer Kommandofolge, die auf Lochkarten oder in einer Datei vorgegeben wird und die sich an die sogenannte automatische Programmsystemgenerierung wendet, mit der auch Betriebssysteme generiert werden. Es können auch Moduln hinzugeladen werden, die nicht aus PEARL-Quellen entstanden sind und die vom PEARL-Programm aus als globale Prozeduren aufgerufen werden können.

5. Weiterentwicklung

In Arbeit ist eine dritte Möglichkeit für das Laden und Binden:
Der erforderliche Dialog mit dem Lader-Binder des Programmiersystems wird von einem Programm geführt, das gewisse vom Compiler über die übersetzten Programme angelegte Beschreibungsdateien direkt verwertet und die für PEARL-Programme geltenden Lade-Binde-Konventionen ausnutzt. Dadurch wird das Laden von PEARL-Programmen in vielen Fällen so einfach wie das Starten des Compilers.

Außerdem werden die quellbezogenen Testhilfen ausgebaut, wofür bereits die erforderlichen Voraussetzungen im Compiler und z.T. im Laufzeitpaket geschaffen sind.

Daneben werden weitere Objektprogramm-Optiemierungen in den Compiler eingebaut.

6. Anwendungen

Die AEG 80-20-PEARL-Implementation wurde inzwischen mehrfach ausgeliefert und hat nach dem Urteil der Benutzer einen stabilen Stand erreicht.

Sie wird seit Anfang 1978 auch in der Schulungsabteilung für PEARL-Kurse mit Übungen eingesetzt.

Eine Anwendung der Implementation ist der in PEARL geschriebene Compiler selbst. Hier wird nicht nur der gute und leistungsfähige "algorithmische Sprachkern" von PEARL in Anspruch genommen, sondern auch das Tasking. So laufen die besonders "EA-intensive" lexikalische Analyse und der erste Syntaxanalyselauf nicht nacheinander sondern als Tasks zeitlich parallel mit entsprechender Synchronisierung. Auch die verschiedenen Ein-/Ausgabe-Vorgänge des Compilers sind als Tasks organisiert, die zeitlich parallel zu Verarbeitsungs-Tasks laufen, jeweils geeignet synchronisiert.

Erstmals mit der damals verfügbaren Entwicklungsversion wurde von Mitarbeitern von Prof. Jünemann (Universität Dortmund) für die INTERKAMA 77 das Modell eines flexiblen Fertigungssystems, gesteuert mit PEARL-Programmen, implementiert. Es handelt sich zwar nur um ein Modell, aber dennoch um ein nicht triviales (z.B. je 80 Digital-Ein- und -Ausgaben, 40 Tasks). (Ein Aufsatz über dieses Modell-Projekt ist u.a. in /2/ sowie in PDV-E112 und KfK-PDV 171 enthalten).

Die Durchführung des Projektes in der Kürze der verfügbaren Zeit wäre ohne Verwendung von PEARL (oder einer mindestens gleichwertigen höheren Realzeit-Programmiersprache) nicht zu schaffen gewesen.

Bei diesem Projekt wie auch bei anderen Anwendungen erwies sich der über Basic PEARL hinausgehende Sprachumfang als hilfreich.

Literaturhinweise

/1/ AEG 80-20 PEARL Sprachbeschreibung
AEG-TELEFUNKEN Konstanz 1979
(Diese Beschreibung ist in erster Linie als Handbuch zur AEG 80-20-PEARL-Implementation gedacht und orientiert sich deshalb in ihrer Gliederung mehr an der Systematik der Sprache PEARL als an didaktischen Gesichtspunkten).

/2/ Zeitschrift "Datenverarbeitung AEG-TELEFUNKEN"
Nr. 2/3, 1977