KfK-PFT-E 8
August 1982

Förderungsprogramm Fertigungstechnik
des Bundesministers
für Forschung und Technologie

Ein portables Prozeßprogrammiersystem auf der Basis von PEARL

PFT-Entwicklungsnotiz

P. Holleczek
Physikalisches Institut
Universität Erlangen

Kernforschungszentrum Karlsruhe

KfK-PFT-E 8

PROJEKTTRAEGERSCHAFT FERTIGUNGSTECHNIK
ENTWICKLUNGSNOTIZ KfK-PFT-E 8

EIN PORTABLES PROZESSPROGRAMMIERSYSTEM AUF DER BASIS VON PEARL

VON

P.HOLLECZEK

PHYSIKALISCHES INSTITUT
UNIVERSITAET ERLANGEN

DIE DIESEM BERICHT ZUGRUNDE LIEGENDEN ARBEITEN WURDEN MIT MITTELN DES BUNDESMINISTERIUMS FUER FORSCHUNG UND TECHNOLOGIE (BMFT) GEFOERDERT

PROJEKTTRAEGER: KERNFORSCHUNGSZENTRUM KARLSRUHE GMBH, KARLSRUHE
IDENTIFIKATION: P4.2/3A;ER-FIE/1

55 SEITEN
14 ABBILDUNGEN
2 TABELLEN
40 LITERATURSTELLEN

AUGUST 1982

Ein portables Prozeßprogrammiersystem auf der Basis von PEARL

Peter Holleczek

November 1981

Vorwort

Dieser Bericht behandelt abschließend das vom 1.1.74 bis 31.12.80 durchgeführte Vorhaben P4.2/3 des Physikalischen Instituts der Universität Erlangen-Nürnberg mit dem Thema: Ein portables Prozeßprogrammiersystem auf der Basis von PEARL. Da eine große Zahl von Einzelberichten zu diesem Vorhaben vorliegt, wird in diesem Bericht nur eine grobe Übersicht über die durchgeführten Entwicklungen gegeben. Der Anhang enthält eine Zusammenstellung aller angefertigten Einzelberichte und Veröffentlichungen.

Als wissenschaftliche Mitarbeiter gehörten diesem Vorhaben an:

Peter Elzer	1.1.74 - 06.04.78
Peter Holleczek	1.1.74 - 31.05.79
Werner Lindstedt	1.1.74 - 31.12.80
Kuno Pelz	1.1.74 - 30.06.79
Franz-Josef Prester	1.1.74 - 31.12.80
Roland Rössler	1.1.74 - 31.05.78
Friedrich Siller	15.10.75- 31.10.77
Martin Trautner	1.6.78 - 31.12.80

Außerdem wurden im Rahmen dieses Vorhabens fünf Diplomarbeiten und eine Studienarbeit angefertigt.

Zusammenfassung

Das Ziel des Vorhabens war, ein weitestgehend portables Programmiersystem für die Programmiersprache PEARL zu entwickeln. Dieses Ziel wurde in einem mehrstufigen Projekt erreicht, in dem sich Entwicklungs- und Erprobungsphasen miteinander abwechselten. Die einzelnen Phasen waren:

- PEARL-Pilotimplementation für eine SIEMENS 306
- Erfahrungsauswertung
- Entwicklung und Anwendung portabler maschinennaher Komponenten
- PEARL-Implementation für eine SIEMENS 310
- Überarbeitung einiger portabler Komponenten.

Die erste Projektstufe wurde im Rahmen einer Arbeitsgemeinschaft abgewickelt, die folgenden Projektphasen im wesentlichen am Physikalischen Institut. In der ersten Projektstufe wurde ein Programmiersystem entwickelt, in dem sich der Erstellungsaufwand von portablen und nicht-portablen Bestandteilen mit ca. 10 MJ etwa die Waage halten. Der Aufwand zur Portierung des Programmiersystems betrug also etwa die Hälfte des gesamten Erstellungsaufwandes.
In den weiteren Arbeiten wurden Methoden angewandt und Komponenten entwickelt, mit denen der Portierungsaufwand abermals mehr als halbiert werden konnte. Alle entwickelten Methoden und Komponenten wurden in zwei Programmiersystemen (für eine SIEMENS 306 und eine SIEMENS 310) erfolgreich eingesetzt. Sie wurden schließlich so überarbeitet, daß ein Einsatz für Mikroprozessoren möglich war.

1. Einleitung

1.1 Vorgeschichte

Seit 1969 gab es am Physik. Inst. der Univ. Erlangen-Nürnberg, (kurz "PIE"), eine Gruppe, die sich mit der Definition von PEARL und mit Möglichkeiten einer Implementierung beschäftigte. Als Implementationsrechner stand eine SIEMENS 305/306 zur Verfügung. Die Finanzierung erfolgte im Rahmen von Vorhaben 411-5999-DVN13 (Überlegungen zur Definition einer experimentorientieren Programmiersprache, 1.7.69-30.6.71) und Vorhaben 411-5999-DVN15 (Entwicklung von PEARL, ein prozeß- und experimentorientierten Programmiersprache, 1.7.71-31.12.73) des Bundesministers für Bildung und Wissenschaft und des Bundesministers für Forschung und und Technologie. Das allgemeine Interesse an der PEARL-Entwicklung legte eine Konzentration der Kräfte nahe, was 1973 zur Gründung der ASME*), einer Arbeitsgemeinschaft aus Firmen und Hochschulinstituten führte. Die Finanzierung des PIE erfolgte ab 1974 durch den Projektträger PDV des Bundesministers für Forschung und Technologie.

Hauptarbeitsgebiete der PEARL-Gruppe am PIE vor der PDV-Finanzierung waren die Mitarbeit bei der Entwicklung und der Definition der Programmiersprache PEARL, Implementationsvorarbeiten und die Mit-Erarbeitung des ASME-Konzepts.

- Mit-Entwicklung und -Definition der Programmiersprache PEARL

  Hierbei wurde hauptsächlich an der Entwicklung und Definition von Systemteil und Ein-/Ausgabe mitgearbeitet. Dies waren dann auch die Schwerpunkte des PIE bei der Erstellung des ersten PEARL-Dokuments /PL73/, dessen Redaktion am PIE vorgenommen wurde und das 1973 erschien.

*) Mitglieder der ASME waren:
-ESG: Fa. Elektronik System Gesellschaft mbH, München
-IRP: Inst. für Regelungstechnik und Prozeßautomatisierung der Universität Stuttgart
-IVD: Inst. für Verfahrenstechnik und Dampfkesselwesen der Universität Stuttgart
-PIE: Physikalisches Institut der Universität Erlangen-Nürnberg

- Implementationsvorarbeiten

  Während der Definitionsarbeiten an der Sprache entstand ein Konzept für ein PEARL-Compiliersystem, das bereits eine Erstellung eines syntaxgesteuerten Compilers in einer höheren Programmiersprache vorsah und das den PEARL-Systemteil zur Betriebssystem-Generierung heranziehen sollte. Die Realisierbarkeit eines Compiliersystems auf den Implementationsrechnern SIEMENS 305/306 wurde geprüft. Hierzu wurde insbesondere das FORTRAN-EA-Laufzeitpaket und das Realzeitbetriebssystem ORG I untersucht und in das Konzept eingeordnet.

  Mit Gründung der ASME (im Jahr 1973) brachte das PIE seine Erfahrungen in die Arbeitsgemeinschaft ein und arbeitete innerhalb der vorgegebenen Schnittstellen an der PEARL-Pilot-Implementation mit.

- Das ASME-Konzept wird im nächsten Abschnitt näher erläutert.

1.2 Randbedingungen

Die Aufgabenstellung des PIE in seinem PDV-Forschungsvorhaben ist nur aus der Gesamt-Aufgabenstellung der ASME zu verstehen. Das Kernthema war die "Erstellung eines portablen Prozeßprogrammiersystems auf der Grundlage von PEARL". Aufgrund des Interesses, das PEARL entgegengebracht wurde, galt es, möglichst schnell für eine Implementation zu sorgen und auch eine weitere Verbreitung der Sprache über ihre Compiler zu ermöglichen. Es wurde deshalb ein mehrstufiger Projektplan entworfen. In der ersten Stufe der Pilotimplementation sollten schnellstmöglich Compiliersysteme erstellt werden, die eine leichtere Übertragbarkeit des Compiliersystems für andere Rechner erlaubten. Die Entwicklung eines weitestgehend portablen Compiliersystems war das Endziel.

Das Konzept der Pilotimplementation ("ASME-Stufe 1") sah vor, den Übersetzungsablauf so zu strukturieren, daß verschiedene Zielmaschinen leicht bedient werden konnten. Der Compiler sollte auf den Zielmaschinen ablauffähig sein. Die Übersetzung der Quellsprache sollte nicht direkt in den Code der Zielmaschine erfolgen, sondern in den Code einer zielmaschinenunabhängigen Zwischensprache ("CIMIC" = Compiler Internal Machine Independent Code).

Die Erstellung des Compilers bis zur CIMIC-Erzeugung war Aufgabe der Fa. ESG, München. Die Erstellung aller zielmaschinennahen Komponenten war Aufgabe der beteiligten Institute, die auch die Zielmaschinen betreuten. Für die erste Projektphase war ein Zeitraum von zwei Jahren veranschlagt. Die Grobstruktur des ASME-Implementationsverfahrens ist in Fig. 1 dargestellt.

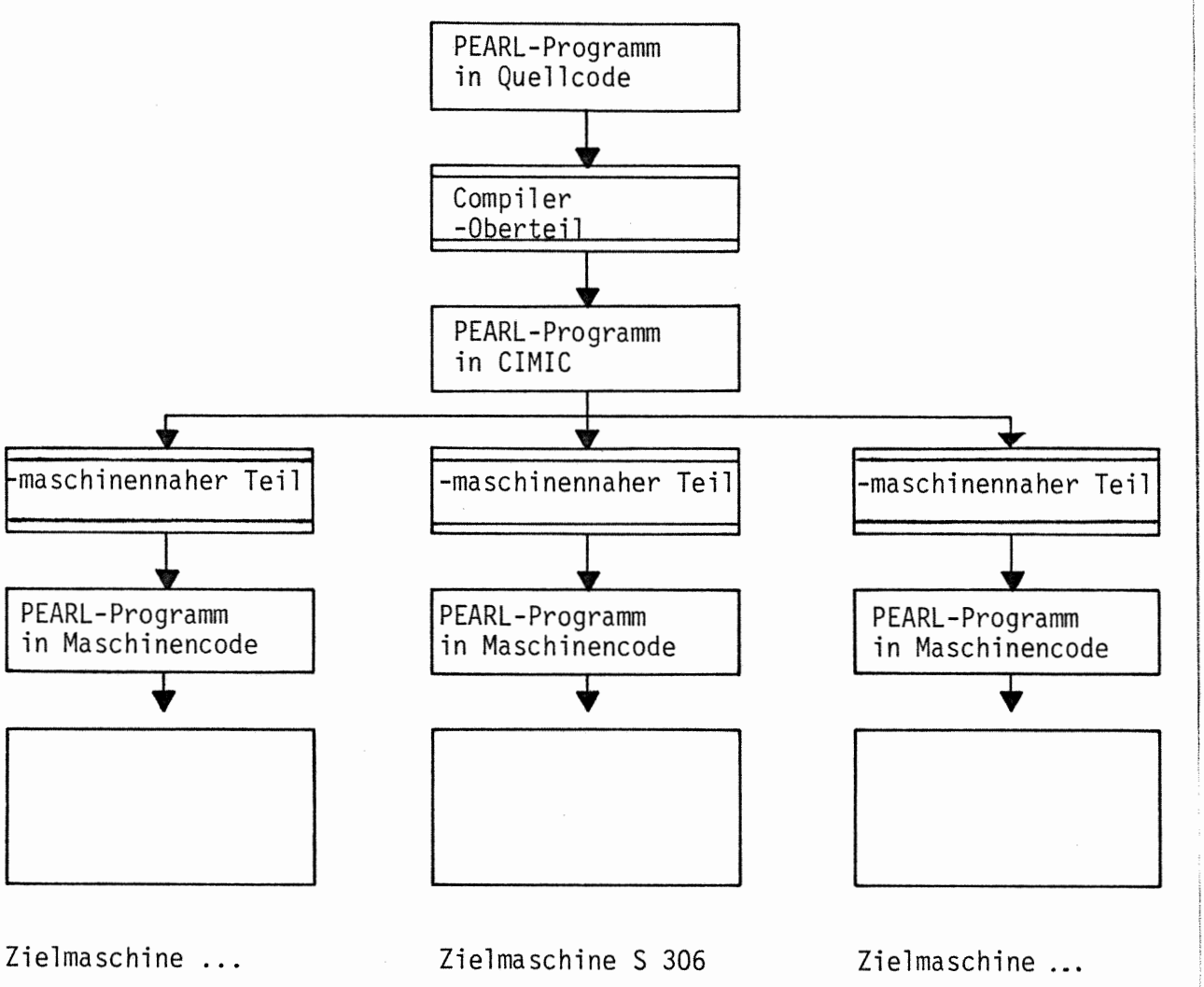

Fig. 1 Grobstruktur der ASME-PEARL-Implementation

Die Aufgabenstellung der anderen Projektphasen war natürlich weniger stark detailliert, da zum Startzeitpunkt vollkommen offen war, ob PEARL mit vertretbarem Aufwand implementierbar war, oder ob nicht unüberwindbare Schwierigkeiten auftreten würden.

Für das PIE bestand in der ersten Stufe die Aufgabe hauptsächlich darin (außer z.B. am Konzept mitzuarbeiten), die zielmaschinenspezifischen Anteile des Compiliersystems für die SIEMENS 306 zu erstellen. In den anschließenden Projektstufen lag abhängig vom Ergebnis der Stufe I das Schwergewicht in Erstellung und Test portabler Systemkomponenten.

Die Projektstufen der Arbeit des PIE lassen sich damit etwa wie folgt gruppieren und zeitlich anordnen:

- (ASME-) PEARL-Pilotimplementation ("Stufe") für eine SIEMENS 306 (bis 1975)
- Erfahrungsauswertung (1976)
- Entwicklung und Anwendung portabler maschinennaher Komponenten (ab 1977)
- PEARL-Implementation für eine SIEMENS 310 (1978-1979)
- Überarbeitung einiger portabler Komponenten (1980).

Den einzelnen Projektstufen ist infolgedessen jeweils ein Kapitel dieses Berichts gewidmet. Aufgrund des ausführlichen vorliegenden Materials sind die Kapitel sehr knapp gehalten.

2. Die ASME-Pilotimplementation für eine SIEMENS 306

2.1 Aufgabenstellung

Die Aufgabe des PIE in der Pilotimplementation war die Entwicklung der maschinenabhängigen Komponenten des Compiliersystems. Maschinenunabhängig wurde von der Fa. ESG der Compileroberteil erstellt, der als "obere" Schnittstelle die Verarbeitung von Quellsprache im "ASME-PEARL-Subset I" / PL 75 / und als "untere" Schnittstelle die Zwischensprache CIMIC-I / CI 76 / aufwies. (Sprachsubset wie Zwischensprache wurden gemeinsam unter den ASME-Partnern erarbeitet.)

Maschinenabhängig waren also zu bewältigen (s. auch Fig. 3 und Fig. 4)

- die Erzeugung ablauffähiger Programme aus ihrer CIMIC-Darstellung
- die Erstellung aller Laufzeitkomponenten, die zur Realisierung der von Sprache und Rechnerkonfiguration vorgegebenen Aufgaben dienten
- die Erstellung eines Rahmenprogramms zur Kontrolle sowohl des Übersetzungsablaufs als auch zum Steuern ablauffähiger Programme zu Testzwecken ("Test- und Bedien-System")

Die zu versorgende Rechnerkonfiguration des PIE zeigt Fig. 2.

Die SIEMENS 306 ist ein "klassischer" Prozeßrechner mit 2 Akkumulatoren, 24-Bit-Wortstruktur und Basis-Adressierung. Alle Geräte sind entweder über 24-bit- oder 6-bit-breite Kanalwerke angeschlossen. Jedes Kanalwerk verfügt über sein eigenes Basis-Adreßregister; für ein Programm stehen zwei Basis-Adreß-Register zur Verfügung. Relativ zu einem Basisadreßregister kann ein Adreßbereich von 16 k Worten erreicht werden. Der vorhandene Speicherbereich von 32 k Worten kann also durch ein PEARL-Programm vollständig ausgenutzt werden.

Das zugehörige Betriebssystem ORG 306 unterstützt "Multitasking" und belegt einen Speicherplatz von ca. 10 k Worten.

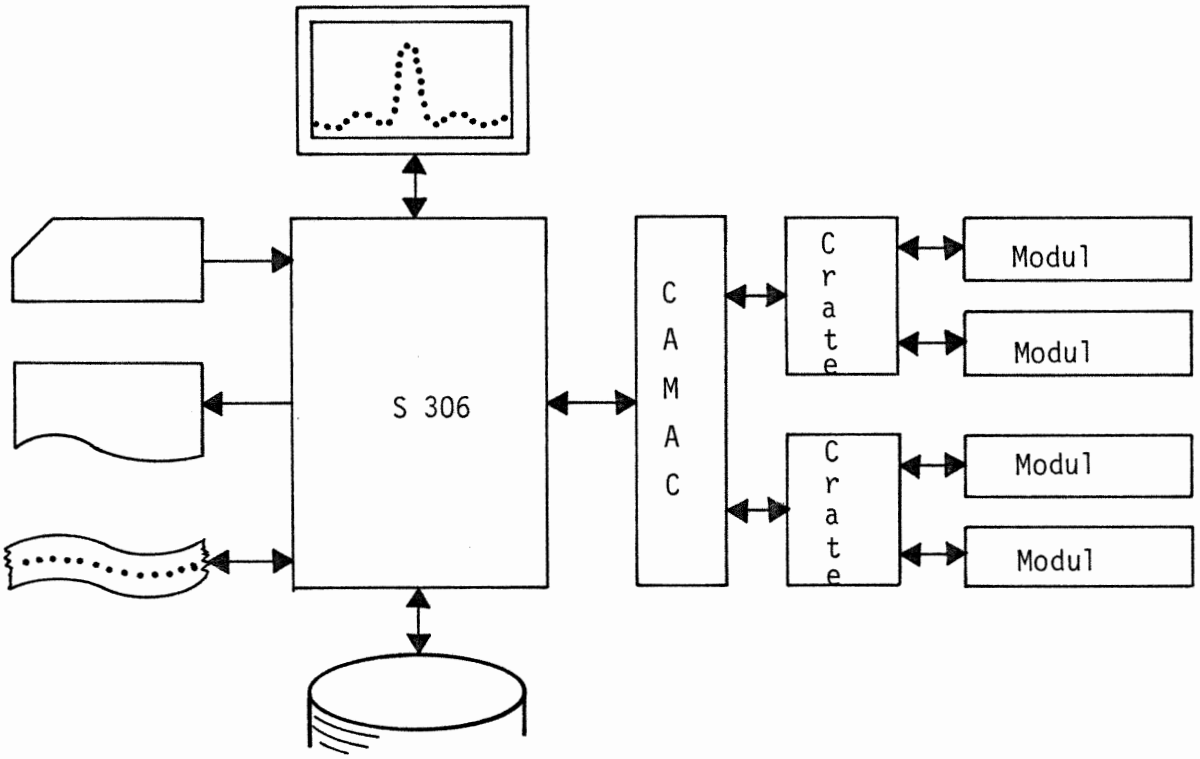

Fig. 2 Grundkonfiguration der SIEMENS 306 des PIE

Als Besonderheit an Peripherie sind zu erwähnen

- ein graphischer Bildschirm (Siemens GSG1)
  (mit Punkt-/Vektor-/Inkrement-Modus, verschiedenen Punktstärken und Vektortypen, sowie Charakter-Generatoren für verschiedene Schrifttypen)
- ein CAMAC-Interface
  (ein herstellerunabhängiges Prozeßperipheriesystem mit hierarchischer Struktur zum Anschluß von bis zu 7 "Crates", die jeweils auf bis zu 23 "Stations" mit rechnerseitig genormten Einschüben versehen werden können).

Da die 306 ursprünglich über keine Anschlußmöglichkeiten eines CAMAC-Interfaces verfügte, waren an dieser Stelle eigene Hardware-Entwicklungen nötig.

2.2 Lösungsüberblick

- allgemeines

Da die erstellten Komponenten im einzelnen beschrieben werden, soll kurz die Gesamtlösung der Stufe I im Zusammenhand skizziert werden. Das Übersetzungssystem bestand aus einer Aufeinanderfolge von Compileroberteil, Codegenerator, Assembler, Binder und Lader (s. Fig. 3), gesteuert durch ein Bediensystem.

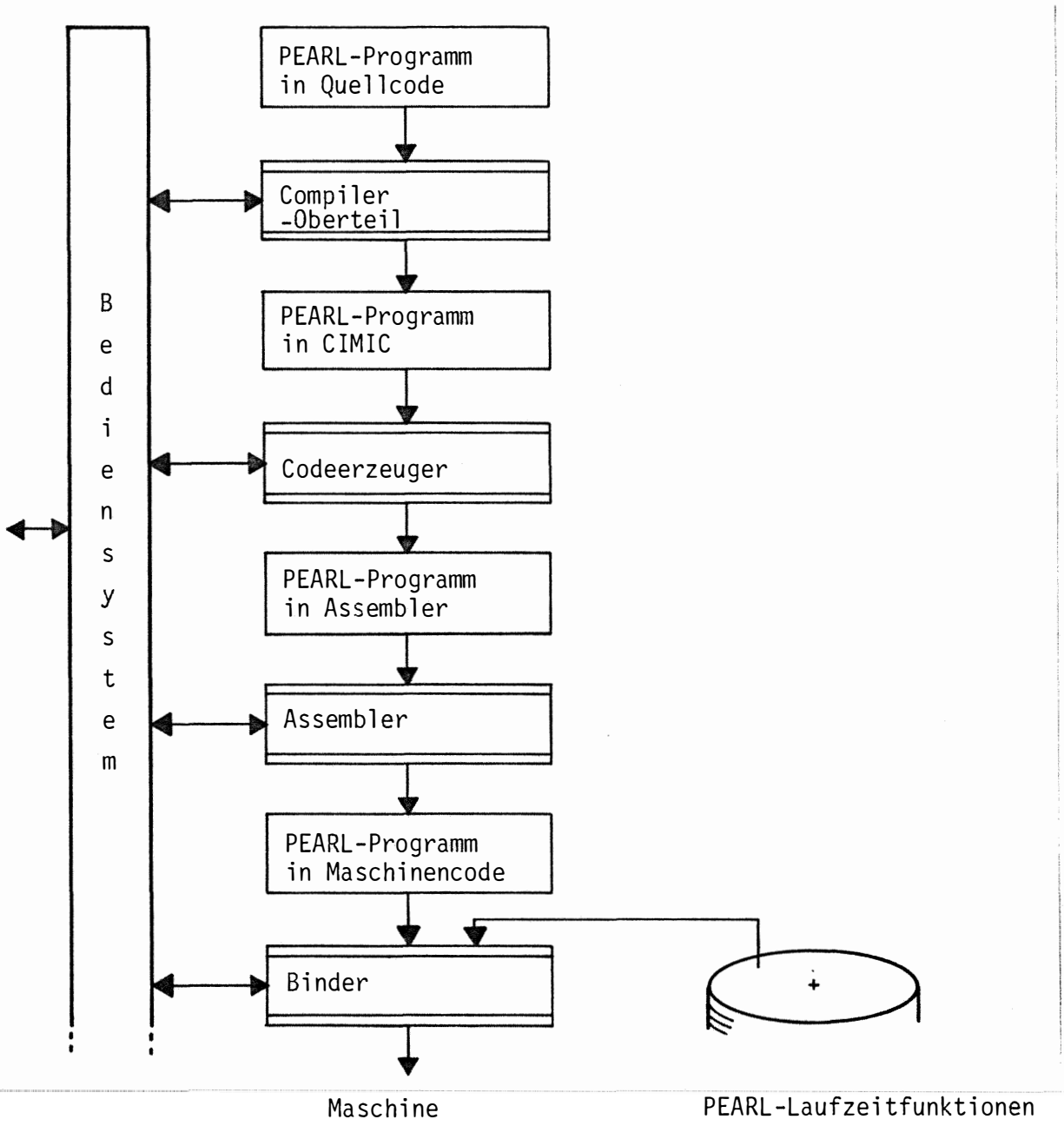

Fig. 3 Komponenten des Übersetzungssystems

Die zur Laufzeit aktiven Komponenten, wie übersetztes Anwenderprogramm, Laufzeitfunktionen und Betriebssystem, lassen sich mit ihrer Aufrufhierarchie in einem Schichtenmodell darstellen (s. Fig. 4)

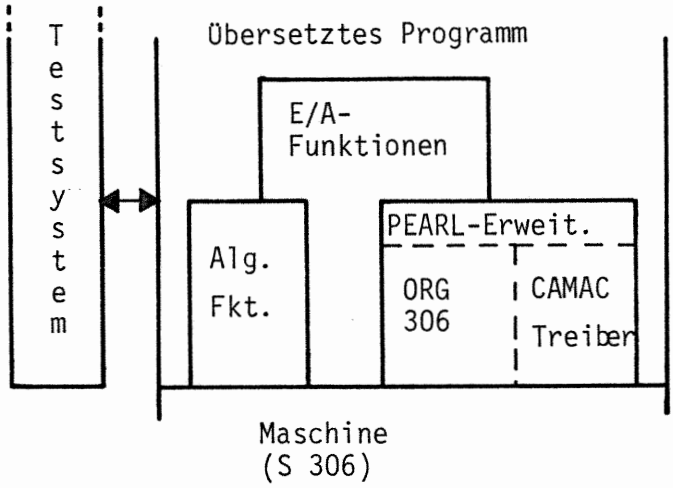

Fig. 4 Komponenten des Laufzeitsystems

Zur Laufzeit ist außerdem noch ein Testsystem aktiv, mit dessen Hilfe lauffähige Programme untersucht werden können.

- Installation des Compileroberteils
  Der von der Fa. ESG entwickelte Compileroberteil verarbeitet den ASME Subset-I. Dieser Sprachsubset hat etwa die Mächtikeit der heutigen DIN-Vornorm BASIC PEARL, enthält aber keine Strukturen, dafür aber die inzwischen aus der Sprache verbannte graphische Ein-/Ausgabe und geht im Tasking über die Norm hinaus. Der Übersetzer des Systemteils erlaubt für die verschiedenen Zielrechner jeweils verschiedene Konfigurations-"Welten". Der Compiler war in FORTRAN geschrieben und hatte zur leichteren Portierung eine eigene EA-Schnittstelle (für Einlesen und Ausdruck der Quellprogramme, Ablegen und Holen der Zwischencodes, Erzeugen von CIMIC). Am PIE wurde auf der S306 der Compiler über seine 14 Phasen hinaus soweit segmentiert, damit er in einem Laufbereich

von ca. 20 k Worten Platz hatte. Ein- und Ausgabemedien waren per Bedienung wählbar.
Zu Segmentierung konnte der Overlay-Binder des Herstellers herangezogen werden.
Übersetzt wurde Compileroberteil mit dem vorhandenen FORTRAN IV-H-Compiler. Die Ein-/Ausgabe-Schnittstelle wurde aus Geschwindigkeitsgründen in Assembler programmiert.

- CIMIC I
  CIMIC kann als rechnerunabhängige Assemblersprache einer virtuellen Maschine betrachtet werden. Sie kennt alle PEARL-Typen. Den in Stufe I zu versorgenden Zielmaschinen entsprechend verfügt die virtuelle Maschine über einen Akkumulator als Verarbeitungsregister. Arithmetische Ausdrücke und Schleifen sind aufgelöst. Durchgereicht werden auch alle Betriebssystemaufträge. Ein-/Ausgabe-Aufrufe werden zum Teil (in Anweisungsbeginn, Aufruf einzelner Elemente einer Liste, Anweisungsende) aufgelöst.

  Globale Bezeichner werden unverändert aus dem Quellprogramm übernommen, für alle anderen Größen werden Bezeichner generiert. Die Notation ist "menschen-" lesbar und gleichzeitig schwach formatiert (je CIMIC-Anweisung eine "Zeile"), um die Verarbeitung zu erleichtern.

- Codeerzeuger
  Der Codeerzeuger hatte die Aufgabe, PEARL-Programme aus ihrer CIMIC-I-Darstellung in die Assembler-Darstellung der SIEMENS 306 umzusetzen. Die Assembler-Schnittstelle enthielt u.a. Aufrufe an das Laufzeitsystem. Der Umsetzer wurde mit Hilfe des Makrogenerators Stage II erstellt. Der Stage II lag in einer Assemblerversion vor. Die Codeer-

zeugung erfolgte aus Gründen der Speicherplatzersparnis in zwei Phasen. Der Umsetzer belegte einen Speicherbereich von 20 k Worten. Da ein Mehr-Modul-Konzept realisiert wurde, gab es eine (zweite) Version des Codeerzeugers, der globale Spezifikationen verarbeiten konnte.

- Assembler und Binder
  Zur Überführung aus symbolischem Code in Maschinencode wurde der Hersteller-Assembler E300 verwendet. Da ein eigenständiger Binder nicht vorlag, wurde der FORTRAN-Binder aus dem FORTRAN-Compiliersystem (FORTRAN IV H-Level) extrahiert.

- Laufzeitsystem
  Laufzeitfunktionen müssen für solche Aufgaben vorhanden sein, für die der Codeerzeuger keine vollständigen Befehlsfolgen oder Betriebssystemaufrufe erzeugen kann. Dazu gehören z.B. die meisten Ein-/Ausgabe-Aufrufe und komplexere arithmetische Operationen. Vielfach stellen Laufzeitfunktionen auch nur formale Anpassungen (z.B. Parameter-Umformungen) an vorhandenen Systemkomponenten dar.
  Standard-Ein-/Ausgabe/PR 76/:
  Aufgabe der Laufzeitfunktionen für die Standard-Ein-Ausgabe ist die Umwandlung der in den Datenlisten angegebenen Größen zwischen interner und externer Darstellung gemäß der in GET/PUT-Anweisungen angegebenen Formatelemente, sowie der Transfer der Daten von/zum physikalischen Gerät. Um Zeit zu sparen, wurden für die "üblichen Datentypen" (wie FIXED = Ganzzahl, FLOAT = Gleitpunktzahl) die Formatierungsfunktionen aus der Fortranlaufzeitbibliothek extrahiert und ggf. leicht verändert. Da diese Bibliothek nur in Maschinencode vorlag, wurde ein Rückübersetzungs-

programm erstellt, das aus dem Maschinencode mit Hilfe der Externlisten wieder symbolischen, assemblierbaren Code erzeugte. Für die PEARL-spezifischen Datentypen (Charakter- und Bit-Strings, Clock- und Duration-Strings) wurden neue Formatierungsfunktionen in Assembler erstellt.

Filehandling /HO 76A/:
Aufgabe der Laufzeitfunktionen für das Filehandling ist die Realisierung der PEARL-Aufrufe CREATE, OPEN, CLOSE, DELETE für "organisierte" Datenträger (wie Platte, Magnetband). Sie enthalten Konsistenzprüfungen zur gegenseitigen Verriegelung der Aufrufe (auch aus anderen Tasks), eine Buchführung über den Zustand der (physikalischen) Datensätze und Aufrufe des Betriebssystems. Diese Funktionen mußten vollständig neu in Assembler erstellt werden.

Graphische Ein-/Ausgabe / PR 77 /:
Aufgabe der Laufzeitfunktionen der Graphischen Ein-Ausgabe ist die Umwandlung der in der Datenliste angegebenen Größen zwischen interner Darstellung und graphischer Steuerinformation gemäß der angegebenen Formatelemente bei SEE/DRAW-Operationen. Hierbei handelt es sich um Funktionen z.B. zur Darstellung von Punkten, Vektoren und zur Skalierung des Bildes.

Prozeß-Ein/Ausgabe / LT 76 /:
Die Aufgabe der Laufzeitfunktionen zur Prozeß-Ein-/Ausgabe ist die Herstellung von Adreß-Beziehungen zwischen Programmdaten und externer Endstelle bei MOVE-Operationen. Bei der hier verwendeten Hardware-Konfiguration (Maschine/CAMAC-Peripherie) ist keine Adressierung der externen Endstellen über

Arbeitsspeicher-Adressen möglich. Die baumartige Struktur der CAMAC-Peripherie muß vielmehr mit Hilfe der Verzweigungs-Angaben aus dem SYSTEM-Teil aufgelöst werden.
Die Verzahnung der Laufzeitfunktionen mit den Betriebssystemaufrufen ist hier besonders eng, da das Betriebssystem die CAMAC-Peripherie nicht "kannte" und hierfür eigens erweitert werden mußte.
Diese Laufzeitfunktionen mußten neu in Assembler geschrieben werden.

Sonstige Laufzeitfunktionen:
An weiteren Laufzeitfunktionen sind zu nennen
- arithmetische Laufzeitfunktionen
- "Standard"funktionen (z.B. EXP, SIN, COS, ...)
- Funktionen zur Behandlung PEARL-typischer Datentypen (z.B. Operationen auf Character-/Bit-Strings, Clock-/Dur-Größen)
- Funktionen zur Parameteranpassung an Betriebssystemaufrufe für Tasking and Timing.

Die Forderung der Reentrant-Fähigkeit der Laufzeitfunktionen wurde dadurch erfüllt, daß zu jeder Task die benötigten Funktionen hinzugebunden wurden.

- Betriebssystem-Anpassungen / RO 76 /
Das vorhandene Hersteller-Betriebssystem ORG 306 war prinzipiell für PEARL-Programme geeignet, von seinen Leistungen her aber nicht ausreichend.

Für alle angeschlossenen Standard-Geräte (Platte, Kartenleser, Schnelldrucker, Lochstreifen-Ein-/Ausgabe, Blattschreiber und prinzipiell auch graphisches Sichtgerät) waren die im Betriebssystem enthaltenen Treiber verwendbar. Der An-

schluß einer CAMAC-Prozeßperipherie war nicht vorgesehen.

Das Betriebssystem erlaubte zwar Multi-Tasking, verfügte aber nur über wenig Funktionen zum Task-Management (z.B. nur Starten und Beenden von Tasks).

Das Betriebssystem wurde also erweitert um

- Funktionen des Task-Managements (SUSPEND, CONTINUE)
- Synchronisierung (REQUEST, RELEASE)
- Bearbeiten von Time-Interrupt-Schedules
- Steuerung der Prozeß-Peripherie und Verarbeiten von Prozeß-Interrupts

Die Erweiterungen waren nicht trivial, da das Betriebssystem für Assembler-Programme konzipiert war und entsprechend über Schutzmechanismen gegen fehllaufende Programme verfügte, die erst "systemkonform" außer Kraft gesetzt werden mußten. Die Betriebssystem-Erweiterungen belegten einen Speicherplatz von ca. 2 k Worten.

- Test und Bedien-System / RO 77 /
  Das Test- und Bedien-System ist das Interface zwischen Benutzer und Programmiersystem.
  Zur Übersetzungszeit übernimmt es anhand der eingegebenen Steuerkommandos die Steuerung des Übersetzungsablaufs. Zur Laufzeit erlaubt es den Zugriff auf bestimmte Größen des PEARL-Programms.

  Der Übersetzungsablauf (einschließlich Binden) kann entweder vollständig mit einem Kommando angestoßen werden (wobei alle weiteren nötigen Informationen, z.B. Modulname, aus dem Quellprogramm entnommen werden), oder komponentenweise (z.B. für den Testbetrieb) vorgenommen werden. Im Mehr-Modul-Betrieb sind (beim Binden) zusätzlich noch

die Namen der Module anzugeben, auf die sich die globalen Spezifikationen beziehen. Das Laden erfolgt lediglich über Angabe der Modul-Namen: alle Tasks und Unterprogramme dieser Module werden dann nacheinander in den freien Arbeitsspeicherbereich geladen.

Zur Laufzeit ist ein Zugriff auf alle globalen Größen (über ihren Namen aus dem Quellprogramm) möglich:

- Testen bzw. Setzen von globalen Variablen
- Testen (bzw. Setzen) von Semaphoren - was aber nur bedingt sinnvoll ist, etwa zum Verifizieren von Verklemmungen
- Testen aller Task-Zustände
- Durchführen aller Task-Operationen.

Außerdem ist ein quellsprachbezogener Werte- und Zeilentrace möglich.
Das Test- und Bediensystem wurde vollständig in Assembler erstellt. Zur Entschlüsselung der Kommandos (insbesondere im Laufzeit-Test) verfügt es über eine listengesteuerte Syntaxanalyse, um auf etwaige Änderungen flexibel reagieren zu können, wovon aber nie Gebrauch gemacht wurde.

- CAMAC-Hardware-Entwicklung / TL 76 /
  Da es für die SIEMENS 306 keine Anschlußmöglichkeit für CAMAC-Prozeßperipherie gab, waren spezielle Hardware-Entwicklungen nötig. Mit Herrn Dr. Trebst vom Physikalischen Institut wurde das Konzept des CAMAC-Controllers und dessen Anschluß an die 306 erarbeitet. Die Realisierung erfolgte durch die Fa. EDS, Aachen. Der Test wurde durch das PIE vorgenommen.

2.3 Ergebnisse

- Implementationsaufwand

Trotz der Verwendung mancher Komponenten der Hersteller-Software belief sich die Implementationszeit auf ca. 2,5 Jahre. Der für die einzelnen Komponenten nötige Aufwand ist aus Tabelle 1 ersichtlich. Nicht enthalten sind dabei Inbetriebnahme, Arbeiten an der Hardware etc.

Systemkomponenten	Aufwand in Mann-Jahren
Codeerzeuger	1,5
Laufzeitfunktionen	
Standard EA	1,5
Filehandling	1
Graphische EA	1
Prozeß-EA	1,5
Algorithmik	0,5
Betriebssystem (-Anpassung)	1,5
Test- und Bediensystem	0,5
Gesamtsystem	9

Tab. 1: SIEMENS 306-Implementationsaufwand (Spezialanfertigungen)

Der Aufwand für die maschinenabhängige Erstellung der Komponenten des 306-PEARL-Compiliersystems liegt in der gleichen Größenordnung wie die maschinenunabhängige Erstellung des Compileroberteils (mit ca. 10 MJ). Dieser Aufwand müßte bei jeder Implementation für einen neuen Rechner neu erbracht werden.

- Einige Leistungsangaben

Die Übersetzungsgeschwindigkeit des Compiliersystems kann anhand des folgenden Beispiels erläutert werden. Für ein Programm aus 500 Quellzeilen brauchte der

Compileroberteil ca. 10 Minuten, die gesamte Übersetzung dauerte knapp 30 Minuten. Die langsamste Komponente war allerdings hierbei der Assembler mit einem Anteil von ca. 13 Minuten (was hauptsächlich in der Konzeption des Assemblers begründet liegt, der auf einen Speicherbereich von 4 k Worten zugeschnitten ist).
Aus demselben Quellprogramm erzeugt der Compiler ein Maschinengrogramm, das nach dem Binden einen Speicherbereich von 1,9 k Worten belegt. Die Zeit zum Auflösen von Prozeß-Interrupts, begrenzt durch Hersteller- bzw. PEARL-Betriebssystem, betrug ca. 2,5 ms.

- Anwendungen
  Da das ASME-I-Compiliersystem (abgesehen vom PAS-2-Compiliersystem der Fa. BBC) nach seiner Fertigstellung im Sommer 1975 / EH 75 / das erste PEARL-Compiliersystem war, waren die Anforderungen recht vielschichtig. Zum Beweis der Leistungsfähigkeit der PEARL-Sprachelemente wurde eine Reihe von Demonstrationsprozessen aufgebaut, die mit PEARL programmiert wurden. Das Compiliersystem mit den Demonstrationsprozessen wurde über Jahre hinweg einer Vielzahl von Interessengruppen vorgestellt. Für interessierte Firmen diente das Compiliersystem als Studienobjekt. Im Hause selbst wurde ein graphisch-interaktives Auswertesystem für kernphysikalische Experimente erstellt / SC 75, SC 79 /.
  Im Rahmen der Universität wurde das Compiliersystem seit 1975 in der PEARL-Vorlesung und im Prozeßrechnerpraktikum eingesetzt / HP 81 /.

3. Erfahrungsauswertung

Die Auswertung erfolgte in größerem Kontext später im Rahmen der Berichte /HO76B/ und /BE77/. Hier seien deshalb unmittelbar die Detail-Erfahrungen wiedergegeben. Sie gliedern sich nach den Komponenten und Schnittstellen des Compiliersystems.

3.1 Bemerkungen zu den Komponenten

- Compileroberteil

  Positiv zum Compileroberteil war zu vermerken, daß seine Installation aufgrund der FORTRAN-Technologie völlig problemlos war. Selbst die Segmentierung für den geringen zur Verfügung stehenden Laufbereich ließ sich leicht bewerkstelligen. Aus der nämlichen FORTRAN-Technologie resultierte aber auch sein wesentlicher Nachteil, die langen Übersetzungszeiten. Dies mußte jedoch kein prinzipieller Effekt sein, da der PEARL-Compiler, auf einer der anderen Zielmaschinen übersetzt, aufgrund des effizienteren Codes des dortigen FORTRAN-Compilers wesentlich schneller arbeitete.

  Strukturelle Schwächen des Compilers waren an zwei Stellen zu vermerken.

  - Die Erzeugung eines menschenlesbaren CIMIC-Codes ist zwar für Testzwecke gut geeignet, ist aber im normalen Produktionsbetrieb ein Umweg, der vom Codegenerator wieder rückgängig gemacht werden muß.
  - Der PEARL-Systemteil wurde zielmaschinenspezifisch fest im Compileroberteil abgehandelt, Umstellungen (Erweiterungen) waren durch Ändern von Syntaxlisten möglich.

- CIMIC

  CIMIC-I war für die in Stufe I gegebenen Zielmaschinen geeignet. Es war klar, daß für die aufkommenden modernen Registermaschinen eine Überarbeitung zu erfolgen hätte.

  Im Interesse einer schnelleren Übersetzbarkeit hätte man sich, wie unter "Compileroberteil" erwähnt, eine kompaktere, nichtmnemonische Darstellungsweise gewünscht. Im Detail gab es

eine Reihe von Kleinigkeiten zu bemängeln, wie z.B. die aufgrund der Kartenstruktur der CIMIC-I-Anweisungen nötige Beschränkung der CHAR-Strings auf 40 Zeichen.

- Codegenerator
  Da die Schnittstellen des Codegenerators (CIMIC und Assembler) vorgegeben waren, bleibt lediglich die Erstellungstechnik zu diskutieren. Beim verwendeten Makrogenerator Stage II überwiegen wohl die Nachteile.
  Zum Vorteil gereicht dieser Technik, daß mit wenig Aufwand schnell Erfolge erzielt werden können, da die Erkennung und Verarbeitung von Zeichenstrings, wie sie bei CIMIC-I zuerst nötig ist, vom Stage II à priori erledigt wird.
  Nachteilig wirkt sich aus, daß dieser Mechanismus sehr langsam ist und überdies sehr unübersichtlich, änderungsunfreundlich und schlecht zu dokumentieren ist. Nichtsdestoweniger wurde gerade auf Dokumentation viel Mühe verwandt; es wurden dazu eigene Dokumentations-Makros erstellt.

- Laufzeitfunktionen
  Der Aufwand zur Erstellung der Laufzeitfunktionen war immens, ihre Realisierung vollständig maschinenabhängig.

  Bei manchen Klassen von Laufzeitfunktionen (z.B. bei der Standard- und graphischen Ein/Ausgabe) lag der Aufwand in den umfangreichen Algorithmen begründet, die sich aber als durchaus rechnerunabhängig erwiesen.

  Bei manchen Klassen von Laufzeitfunktionen (z.B. Prozeß-Ein/Ausgabe, Algorithmik) ist der Aufwand durch den Zwang zu einer zeit-effizienten Programmierung begründet.

- Betriebssystemanpassungen
  Das Erweitern des vorhanden (Assembler-) Betriebssystems zu einem PEARL-Betriebssystem führte an verschiedenen Stellen zu Problemen:

  - Das systemkonforme Außerkraftsetzen der Schutzmechanismen:
    Es galt, das Betriebssystem, von dem keine Quellfassung vorhanden war, auf eine solche Weise zu erweitern, daß

seine eingebauten Schutzmechanismen im nachhinein immer noch wirksam sind. (z.B. Eindringen in den Adreßbereich des Betriebssystems, ohne daß das Betriebssystem völlig "schutzlos" ist).

- Erweiterungen:
  Ist eine Funktion im vorhandenen Betriebssystem nicht vorgesehen, reichen unter Umständen "lineare Extrapolationen" der Grundfunktionen nicht aus. In diesen Fällen mußte oft zu unerlaubten Tricks gegriffen werden, wie z.B. beim Einführen der SUSPEND- und CONTINUE-Operationen,(da das 306-Betriebssystem nur ACTIVATE- und TERMINATE konnte).

- Zeitverhalten:
  Das Zeitverhalten eines komfortablen Betriebssystems, das auch über viele Sicherheitsmechanismen gegen fehlerhafte Assemblerprogramme verfügt, ist ohnehin nicht besonders schnell. Durch Overlays für solche PEARL-Funktionen, die sich aus einfacheren Funktionen zusammensetzen lassen, vergrößern sich die Laufzeiten noch.

- Test- und Bediensystem
  Das quellsprachbezogene Test-System stellt eine nahezu ideale Ergänzung des Compiliersystems dar, das den "Testern" keine Assemblerkenntnisse abverlangt. Größere Implementationsprobleme traten nicht auf. Bemängelt werden könnte, daß
  - nur GLOBALE Größen getestet werden konnten,(andernfalls hätte der Compiler so geändert werden müssen, daß auch Quell-Namen für nicht-GLOBALE weitergereicht werden können);
  - sicherlich noch komfortablere Testoperationen denkbar wären (z.B. Simulation von Eingaben);
  - die Implementation maschinenabhängig und damit nicht übertragbar war (sollte späteren Entwicklungen vorbehalten bleiben).

  Die Mängel sind jedoch insofern fast gegenstandslos, als es einige Jahre dauern sollte, bis andernorts ähnliche Testsysteme zur Verfügung standen und ein Vergleich möglich war.

- Sprache

Da die ASME-I-Pilotimplementation - last but not least - den Zweck hatte, Funktionen und Implementierbarkeit der PEARL-Sprachelemente zu prüfen, wurden auch hierzu detaillierte Untersuchungen angestellt. Hauptsächliche Kritikpunkte waren

- die unorganische SIGNAL- (d.h. exception-) Behandlung;
- die aufwendigen Laufzeitprüfungen bei der Ein-/Ausgabe (nahezu der gesamte Aufgabenbereich der Filehandling-Laufzeitfunktionen besteht aus Zulässigkeitsprüfungen).

Die Verbesserungsvorschläge des PIE, nämlich

- Ersetzen der SIGNAL-Behandlung durch Aufrufe mit Fehlerausgang;
- weitestgehende statische Prüfmöglichkeit bei Ein-/Ausgabe und Filehandling-Aufrufen;
- eine organischere Gestaltung des PEARL-Systemteils unter Berücksichtigung eines moderneren EA-Konzepts /HO77/

wurden zusammen mit Vorschlägen anderer Implementatoren dem PEARL-Arbeitskreis des VDI/E vorgelegt und in einer Klausurtagung behandelt.
Während es bei der historischen SIGNAL-Behandlung blieb, wurden mit dem DATION-Konzept Grundlagen für weitgehende statische Prüfungen geschaffen.

3.2 Konsequenzen

Die aus den Stufe-I-Arbeiten vorliegenden Erfahrungen wurden, soweit sie das Compiliersystem des PIE berührten, gesammelt und in detaillierte Verbesserungsvorschläge umgewandelt /HO76B/. Sie hatten z.B. eine Verringerung des Erstellungsaufwandes bei den Laufzeitfunktionen, insbesondere der Ein-/Ausgabe, zum Ziel. Der Weg dorthin sollte über die Definition geeigneter Schnittstellen und über die Verwendung portabler Algorithmen führen.

Da die hier verwendete Methode der Erfahrungsauswertung vielversprechend erschien, wurde unter Federführung des PIE von der ASME eine Studie /BE77/ über Verfahren zur Erstellung portabler Automatisierungssoftware angefertigt. Ihr Kernthema war der Problemkreis üblicher Realzeit-Compiliersysteme einschließlich aller Komponenten (z.B. Betriebssysteme). Sie ging aber auch auf Methoden der Programmerstellung ein. Für jede einzelne Komponente wurde eine Situationsanalyse geboten, die nicht nur die in der ASME, sondern auch anderswo verwendeten Methoden berücksichtigte. Außerdem wurde für jede Komponente ein "wünschbares" Ziel angegeben und untersucht, unter welchen Randbedingungen und innerhalb welcher Frist welche (portable) (Teil-)Lösung erreicht werden könnte.

Ein Ergebnis der Studie war, daß am Grundkonzept des Compiliersystems, der Verwendung einer Zwischensprache (hier CIMIC), nicht gerüttelt werden sollte.

Leider stellte sich heraus, daß es eine geschlossene Überarbeitung des gesamten Compiliersystems nicht geben sollte, auch nicht wenigstens der zentralen Komponenten, wie CIMIC bzw. Compileroberteil. Eine Überarbeitung gerade dieser Komponenten wäre aber nötig gewesen, um etwa alle Verbesserungen an den Laufzeitkomponenten zu erleichtern und ihnen einen endgültigen Charakter zu verleihen.

So wurden Compileroberteil und CIMIC bei den ehemaligen Erstellern gewissen Änderungen unterzogen, während es dem PIE oblag, folgende maschinennahen Komponenten zu überarbeiten:

- Codegenerator
- Laufzeitfunktionen für Standard-Ein-/Ausgabe
  Graphische Ein-/Ausgabe
  Algorithmik/Arithmetik
- Betriebssystem.

Als weiteres Ergebnis der Studie zeichnete sich ab, daß eine erfolgversprechende Überarbeitung des ASME-Compiliersystems möglichst nur als ganzes erfolgen sollte, da alle Komponenten stark von einander abhingen.

4. Entwicklung portabler maschinennaher Komponenten

Bei der Entwicklung maschinennaher portabler Komponenten ging es darum, den in diesem Bereich im Vergleich zum Compileroberteil sehr großen Adaptionsaufwand zu verringern: während beim Compileroberteil z.B. lediglich eine Adaption nötig war, die einen Bruchteil des Erstellungsaufwandes ausmachte, lief bei den Assemblerlösungen für die maschinennahen Komponenten jede Portierung immer auf eine Neuerstellung hinaus. Wo möglich, sollte also versucht werden, maschinenunabhängige Lösungen zu finden.
Der Weg zur Entwicklung weitestgehend portabler Komponenten war in allen Fällen ähnlich. Zuerst ging es darum, Schnittstellen zu definieren, innerhalb derer maschinenunabhängige Lösungen möglich waren. (Der Einfachheit halber soll im weiteren die näher an der Quellsprache bzw. CIMIC angesiedelte Schnittstelle als "obere", die näher an der Maschine angesiedelte als "untere" Schnittstelle identifiziert werden.) Außerhalb dieser Schnittstelle sind i.a. maschinen- oder systemspezifische Anpassungen vonnöten. In welcher Sprache die maschinenunabhängigen Lösungen formuliert werden, kann nur im Einzelfall entschieden werden. Kommen weitere Abhängigkeiten ins Spiel, z.B. die starke Geräteabhängigkeit bei der graphischen Ein-/Ausgabe, mußten Möglichkeiten gefunden werden, ihren Einfluß auf die maschinenunabhängigen Algorithmen zu begrenzen.

4.1 Standard-Ein-/Ausgabe

Die Laufzeitfunktionen für die Standard-Ein-/Ausgabe dienen dazu, die in der Datenliste einer GET-/Put-Anweisung vorkommenden Größen entsprechend den Elementen einer Formatliste zwischen interner und externer Darstellung zu wandeln. Diese Aufgabe läßt sich weiter unterteilen in:

- Formatweiterschaltung (Verwalten der Zeiger auf Daten- und Formatliste) und
- Formatierung (Wandeln eines Datenelements entsprechend seines Formats).

Da die Formatweiterschaltung von der Systemarchitektur abhängt und insofern nicht allgemeingültig ist, wurde allein die Formatierung, die ein typisches Problem darstellt, in die weiteren Überlegungen miteinbezogen. Die "obere" Schnittstelle der portablen Standard-E/A-Laufzeitfunktionen ist also der Aufruf eines Daten-/Formatelemente-Paares, die "untere" Schnittstelle der Aufruf eines Zeichenstrings im Transferpuffer zum E/A-Gerät über das Betriebssystem (vgl. Fig. 5).

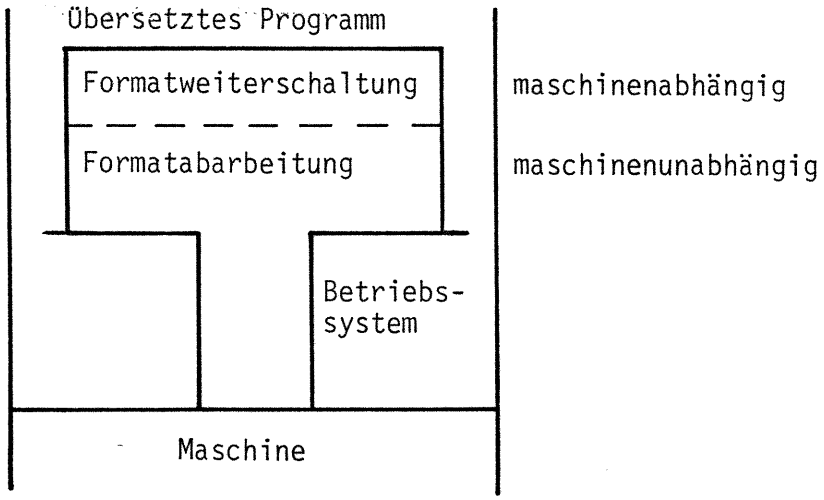

Fig. 5 Aufrufhierarchie der Ein-/Ausgabe-Laufzeitfunktionen

An die zur Formulierung der maschinenunabhängigen Algorithmen zu verwendende Programmiersprache müssen folgende Anforderungen gestellt werden: sie muß auf dem Zielrechner laufen bzw. leicht installierbar sein, muß über geeignete Sprachkonstrukte (z.B. Zeichenstring-Behandlung) verfügen und muß hinreichend (laufzeit- bzw. speicher-)effizient sein. Aus den beiden ersten Gründen kam nur PEARL selbst in Frage. Die hinreichende Effi-

zienz ließ sich wohl erst nach Abschluß der Arbeit verifizieren, doch stellen formatgesteuerte zeichenweise arbeitende Geräte keine so hohen zeitlichen Anforderungen wie etwa Prozeßperipheriegeräte.
Erstellt wurden also Formatierungsroutinen für alle in PEARL vorkommenden Daten- bzw. Standard-Format-Typen.
Die Formulierung erfolgte in einem algorithmischen Subset von PEARL, so daß schon nach Vorliegen eines Übersetzers für gerade diesen Subset die Laufzeitfunktionen für die Zielmaschine erzeugt werden können (vgl. Fig.6)

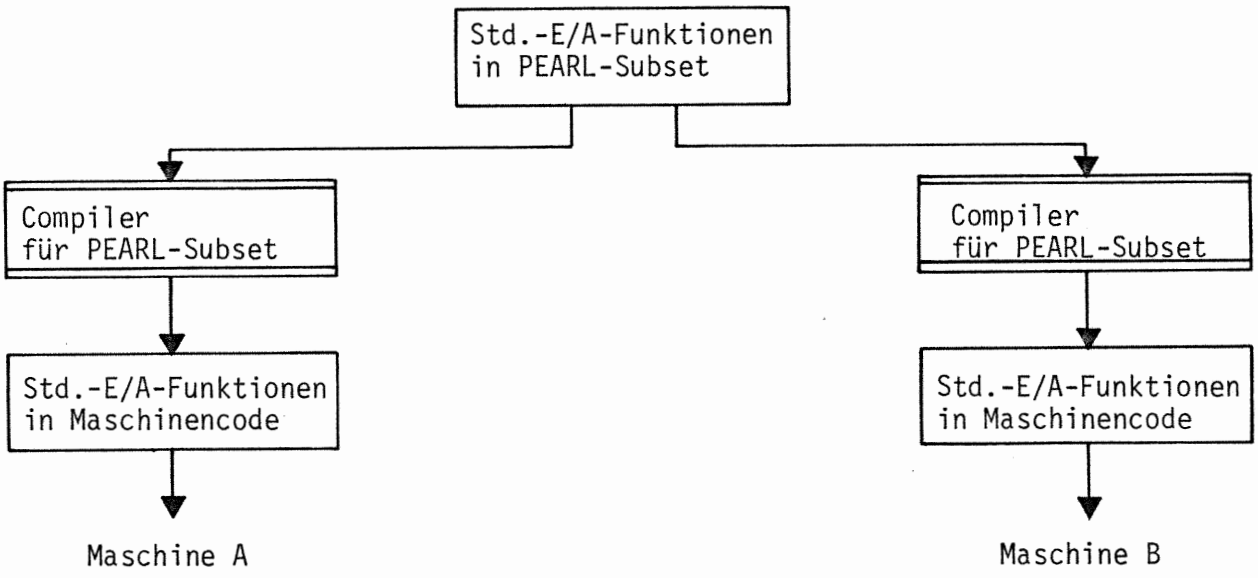

Fig. 6 Erzeugung der Ein-/Ausgabe-Laufzeitfunktionen für verschiedene Maschinen.

Mit Hilfe der vorhandenen PEARL-Compiler wurden für die Rechner SIEMENS 306 und SIEMENS 310 (siehe auch Kapitel 5) Standard-E/A-Laufzeitfunktionen erzeugt und eingesetzt.
Da an der 306 auch Laufzeitfunktionen in Assembler vorlagen, war ein Vergleich zwischen diesen und den in einer höheren Programmiersprache formulierten Funktionen möglich. Es zeigte sich eine Speicherplatzverlängerung um der Faktor 4, aber eine Laufzeitverlängerung um den

Faktor 20. Während sich die Speicherplatzverlängerung im erwarteten Rahmen hielt, war die Laufzeitverlängerung unerwartet hoch. Nähere Untersuchungen zeigten, daß hierfür neben dem sicher nicht zu effizienten Code des PEARL-Compilers hauptsächlich ein Umstand verantwortlich war: Die Maschinenunabhängigkeit war an manchen Stellen zu weit getrieben worden; so erfolgte die sehr häufig vorkommende Umwandlung zwischen CHAR- und FIXED-Größen über den Aufruf einer Prozedur, während sie maschinenabhängig lediglich eine andere statische Interpretation ohne Laufzeitaufwand bedingt. Bei Beachten solcher Zusammenhänge, die sinnvollerweise durch Separation bestimmter maschinenabhängiger Teilfunktionen berücksichtigt werden, stellt eine in PEARL geschriebene Standard-E/A ein brauchbares Hilfsmittel dar.

4.2 Graphische Ein-/Ausgabe / PR 80 /

Die Laufzeitroutinen für die Graphische-E/A dienen, ähnlich der Standard-E/A, dazu, die in der Datenliste einer SEE-/DRAW-Anweisung vorkommenden Größen entsprechend den Elementen einer Formatliste zwischen interner und externer Darstellung zu wandeln. Wie bei der Standard-E/A wird auch hier nur die Formatierung, nämlich die Wandlung eines Datenelements, portabel in Angriff genommen. Die "obere" und "untere" Schnittstelle entspricht derjenigen der Standard-E/A.
Zwei wesentliche Unterschiede gibt es jedoch zur Standard-E/A:

- Es müssen einige numerische Berechnungen angestellt werden (z.B. Vergrößerungen, Bildausschnittbegrenzungen, Interpolationen).
- Da es keinen genormten graphischen Code gibt und die "untere" Schnittstelle der Aufruf eines Datums im Transferpuffer zum E/A-Gerät ist, müssen gerätespezifische Besonderheiten in den Laufzeitfunktionen berücksichtigt werden.

Die erste Bedingung stellt eine weitere Anforderung an die Erstellungssprache, die auch über eine ausreichende Arithmetik verfügen muß, was PEARL als Werkzeug verwendbar machte. Die zweite Bedingung legte nahe, die gerätespezifischen Stellen der Routinen zu lokalisieren und zu parametrisieren, damit sie von einem Generator erzeugt werden können.
Es wurden also Formatierungsroutinen für alle in PEARL vorkommenden Daten bzw. graphischen Format-Typen erstellt. Die Routinen enthalten markierte geräteabhängige Stellen, die von einem Generator ausgefüllt werden. Der Generator wiederum wird durch Angabe von Geräte-Parametern gesteuert. Anhand eines Satzes von Geräte-Parametern kann ein gerätespezifisches Laufzeitsystem erzeugt werden (vgl. Fig. 7).

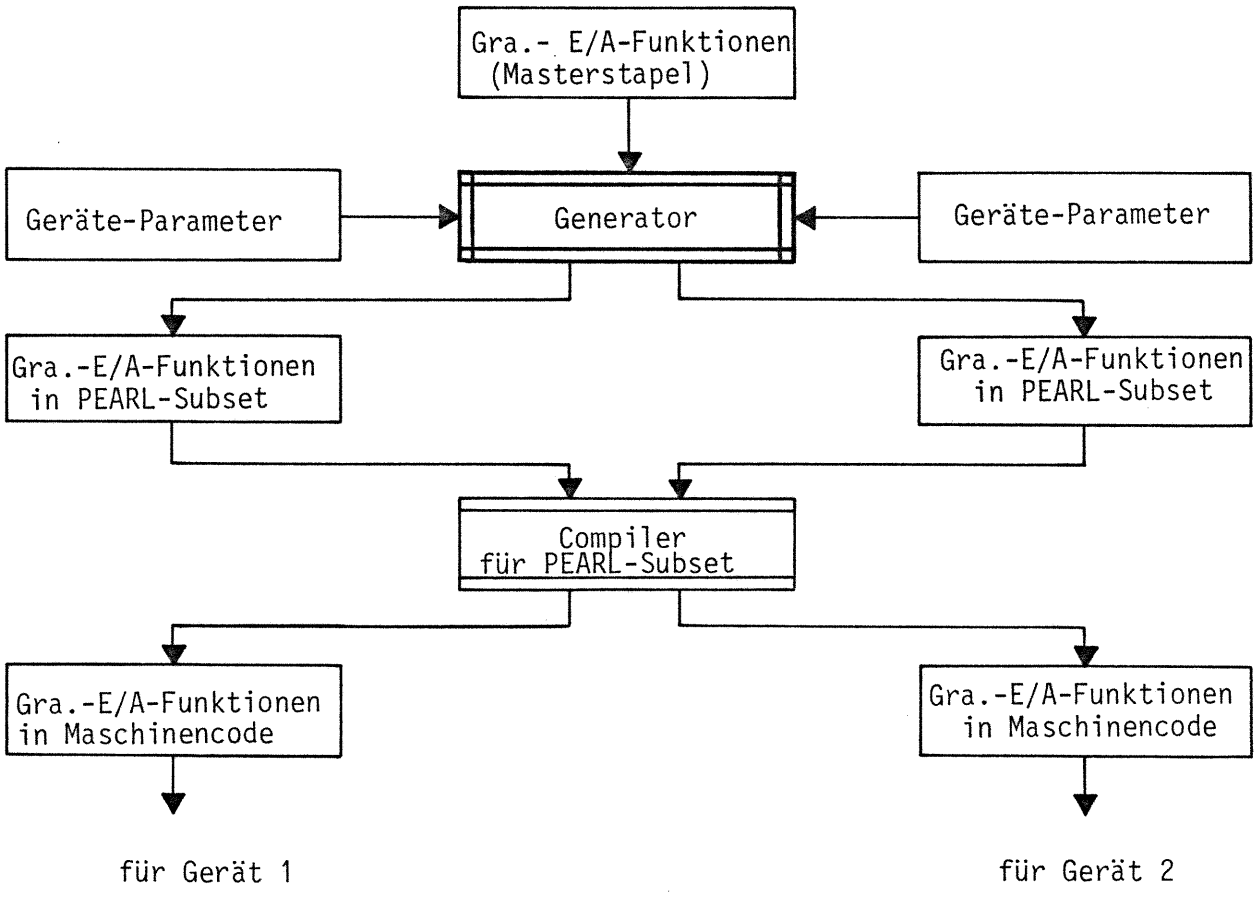

Fig. 7 Erzeugung der graphischen Laufzeitfunktionen für verschiedene Geräte

Mit Hilfe des vorhandenen PEARL-Compilers wurden für die SIEMENS 306 und eine Reihe von graphischen Geräten (Bildschirme, Punkt-, Linienplotter) Laufzeitfunktionen erzeugt und getestet.
Da wie bei der Standard-Ein-/Ausgabe auch hier in Assembler geschriebene Laufzeitfunktionen vorlagen, war ein Vergleich zwischen diesen und den generierten, über eine höhere Programmiersprache erzeugten, Laufzeitfunktionen möglich. Es zeigte sich eine Codeverlängerung von 700 Worten auf 1700 Worte, eine Laufzeitverlängerung bei Ausgabe eines vollständigen Bildes von 2.4 auf 3.7 Sekunden. Bei der Ausgabe einer starken Detailvergrößerung ergab sich dagegen eine Laufzeitverringerung von 4.7 auf 2.8 Sekunden. Die Erklärung für diesen Effekt ist, daß sich bei Erstellung von Laufzeitfunktionen in einer höheren Programmiersprache komfortablere Bild-Begrenzungsalgorithmen formulieren lassen, die zur Laufzeit von Vorteil sind, aber auch eine Verlängerung des Codes bewirken. Im Vergleich zur Standard-E/A sind hier eine Reihe von Rechenaufgaben enthalten, die sich in Assembler nur sehr unangemessen formulieren lassen.

4.3 Algorithmische Laufzeitfunktionen / LI 80, LI 81 /

Die algorithmischen Laufzeitfunktionen sind eine Sammelbezeichnung für alle Laufzeitfunktionen, die weder das Betriebssystem noch die Ein-/Ausgabe betreffen. Beispiele sind:

- mathematische Standardfunktionen und arithmetische Verknüpfungen,
- Funktionen zur Behandlung PEARL-typischer Daten,
- Funktionen auf zusammengesetzte Datentypen.

Es handelt sich dabei also um Funktionen, die vom Codegenerator aus Komplexitätsgründen nicht aufgelöst werden. Sie stellen gleichsam eine lineare Funktionserweiterung der Maschine dar und sind ggf. auch bei Ein-/Ausgabe-Laufzeitfunktionen beteiligt. An sie müssen also besondere Effizienzbedingungen, wie z.B. an ein Betriebs-

system auch, gestellt werden.
Ihre "obere" Schnittstelle sind die vom Codegenerator aus CIMIC erzeugten Funktionsaufrufe, ihre "untere" Schnittstelle ist der Zielmaschinen-Code. Die Effizienz dieser Laufzeitfunktionen wird letztlich dadurch beeinflußt, in welcher Sprache sie formuliert und wie aus dieser Sprache ablauffähiger Code erzeugt wird. An diese Formulierungs-Sprache werden also ganz spezielle Anforderungen gestellt.
Die Wahl fiel auf eine virtuelle Drei-Adress-Maschine, in deren Befehlsvorrat die Laufzeit-Funktionen formuliert werden. Eine solche Maschine hat den Vorteil, daß sie auf einem höheren Niveau steht als die üblichen Zielmaschinen, und daß deshalb bei der Erzeugung von ablauffähigem Code keine "Rückübersetzung" nötig ist. Die Umsetzung von virtuellem Adreß-Code in den Zielmaschinen-Code erfolgt in verschiedenen Schritten (z.B. über den Code einer virtuellen Ein-Adreß-Maschine), um auf jeder Stufe Eigenschaften der Zielmaschine berücksichtigen zu können. Man kann von einer "asymptotischen" Annäherung der Zielmaschine sprechen. Das Umsetzungsprinzip ist in Fig. 8 skizziert.

Der erwähnte Umsetzer wurde in ERLAN/RT78/ erstellt. Außerdem wurde eine Reihe von typischen algorithmischen Laufzeitfunktionen formuliert und in den Code der drei Zielmaschinen SIEMENS 306 und 310 (s. auch Kapitel 5) und Zilog Z80 umgesetzt.
Der auf diese Weise (teilweise) maschinell erzeugte Code wurde mit vorhandenen Routinen auf der SIEMENS 306 verglichen. Es zeigte sich, daß der mit der hier entwickelten Methode erzeugte Code kürzer war als der vorhandene. Dies liegt wohl daran, daß bei einer portablen (einmaligen) Erstellung von Laufzeitfunktionen mehr Sorgfalt investiert wird, als bei häufigen Speziallösungen.

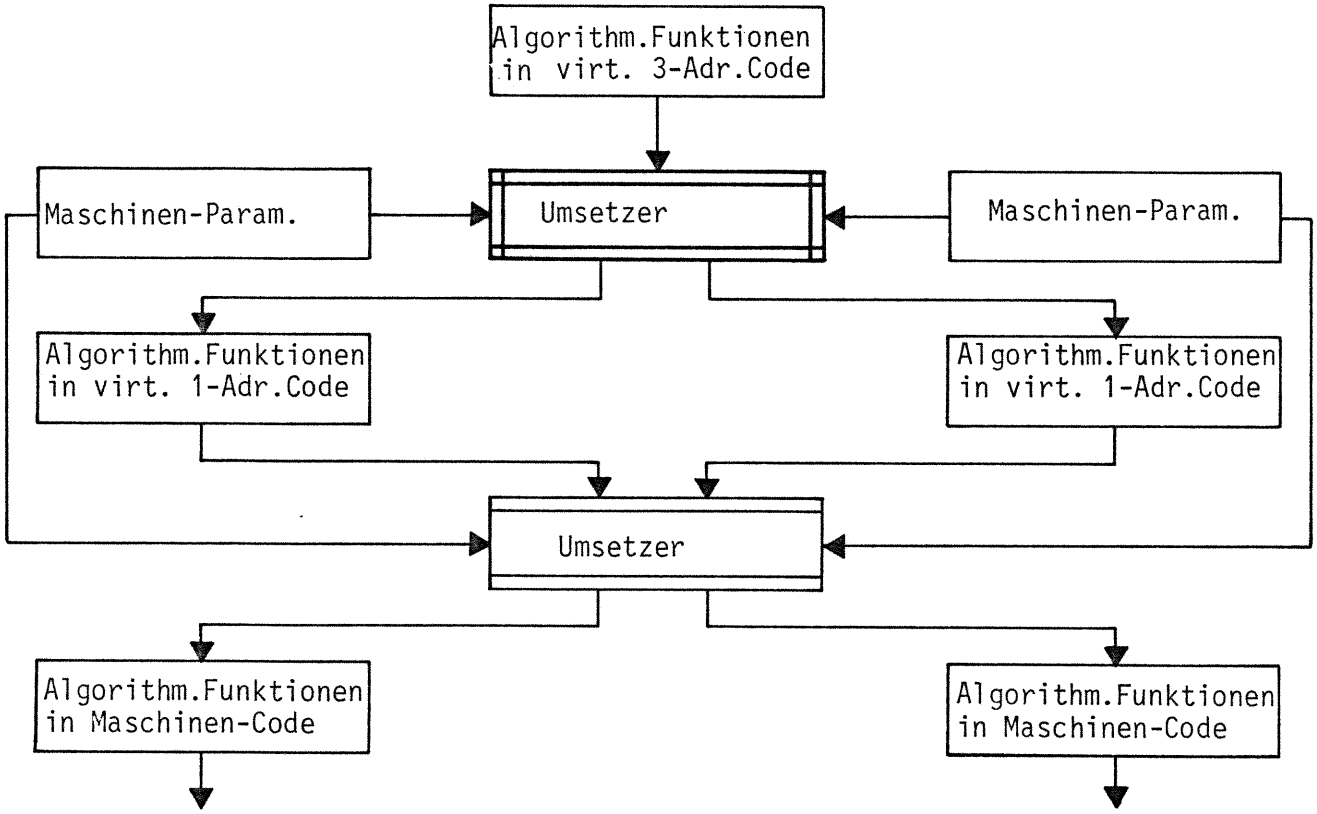

Fig. 8 Schema der stufenweise Umsetzung der algorithmischen Laufzeitfunktionen. (Der Übersichtlichkeit halber sind nur zwei Umsetzungsstufen dargestellt)

4.4 Codeerzeuger* /PE 80/

Die Entwicklung eines "portablen" zielmaschinenunabhängigen Codeerzeugers nach dem Muster der Laufzeitfunktionen (durch Lokalisieren einer "oberen" und "unteren" Schnittstelle) ist natürlich ein Unding, da dies lediglich das Einschieben eines weiteren Übersetzungsschrittes bedeuten würde. Die Frage ist eher, ob es gelingt, aus dem Codeerzeugungsvorgang Teile zu extrahieren, die unabhängig von der Zielmaschine sind, und/oder die Zielmaschinen so zu parametrisieren, daß über einen Generator Zielmaschinen-Charakteristika

*) Diese Arbeit wird seit 1.7.79 vom Lehrstuhl für Programmiersprachen der Universität Erlangen-Nürnberg verfolgt.

in den Codeerzeuger eingesetzt werden können.
In diesem Fall wurde eine Mischung aus beiden Wegen gewählt. Der generierbare Codeerzeuger besteht im wesentlichen aus einem Masterstapel und einem Generator. Der Masterstapel stellt einen Codeerzeuger in Quellsprache dar, der an den zielmaschinenspezifischen Stellen unvollständig ist (z.B. Länge der Adreßeinheit, Wortlänge, u.v.a.m.). Der Generator setzt anhand eines zielmaschinenspezifischen Parametersatzes die fehlenden Größen in den Codeerzeuger ein. Der so vervollständigte Codeerzeuger kann dann übersetzt und ablauffähig gemacht werden. (Das Generierungsverfahren ist in Fig. 9 skizziert).

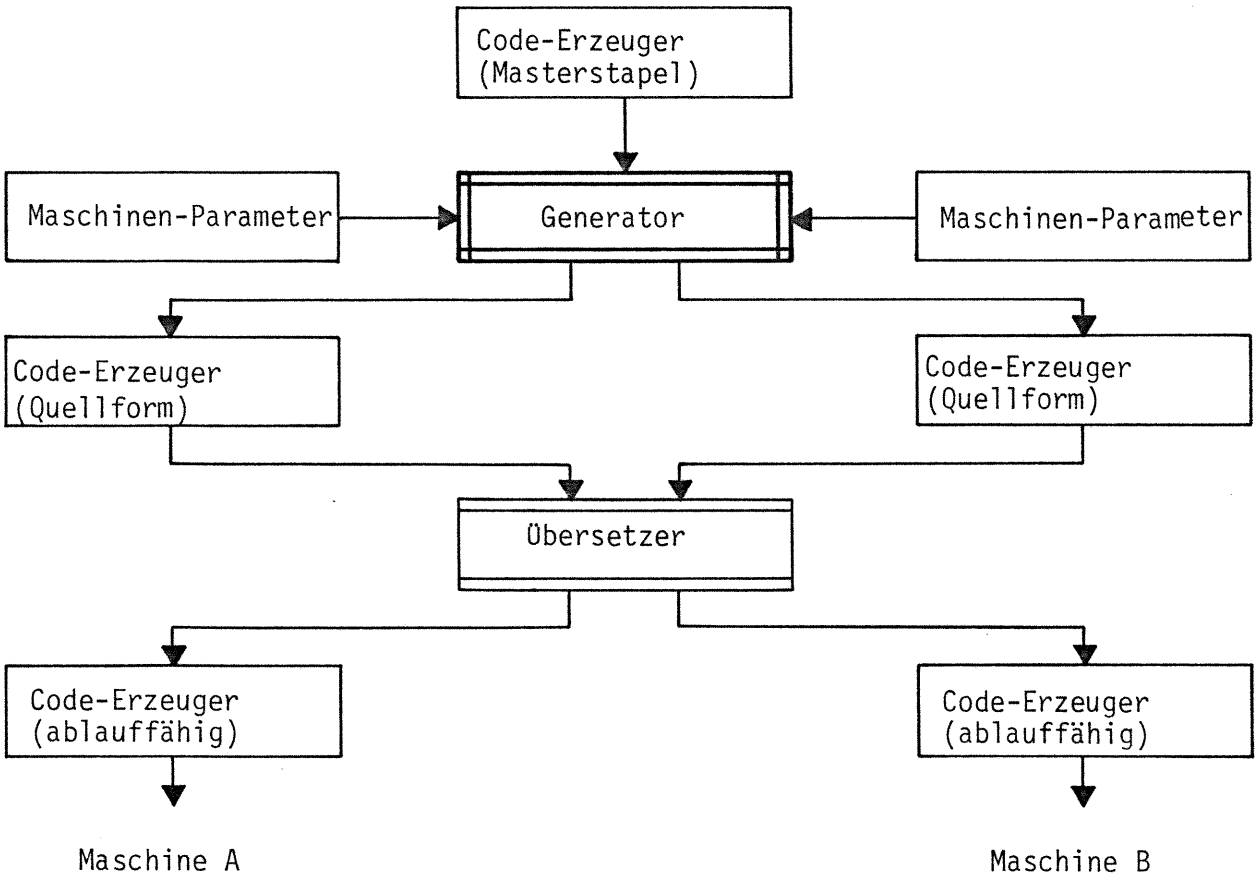

Fig. 9 Skizze des Generierungsverfahrens für Codeerzeuger

Diese Technik stellt gewisse Anforderungen an die "Umgebung". So sind die Zwischensprachen der CIMIC-Familie aufgrund ihres stark historischen Erscheinungsbildes für automatische Generiervorgänge schlecht geeignet. Es wurde deshalb eine speziell für diese Belange besser geeignete Zwischensprache ("CODE") entworfen. Sie unterscheidet sich z.B. von CIMIC I äußerlich durch einen einfacheren Anweisungsaufbau, konzeptionell durch eine wesentlich tiefere Ansiedelung der zugrundeliegenden virtuellen Maschine. Auf diese Weise wird der Erkennungsteil des Codeerzeugers (und auch dessen Generierung) einfacher. Weitere Vorkehrungen sorgen dafür, daß die Codeerzeuger den Charakter von Ein-Pass-Assemblern bekommen.
Die anderen Anforderungen richten sich an die "untere" zielmaschinenspezifische Schnittstelle des Codeerzeugers. Um den Übersetzungsvorgang zu straffen, wird nicht mehr Assemblercode erzeugt, sondern gleich Bindecode. Diese Schnittstelle berücksichtigt die üblichen Anschlüsse für System-, Anwender- und Programm-Bibliotheken. Codeerzeuger (-Masterstapel) und Generator wurden, wie der Umsetzer für die algorithmischen Laufzeitfunktionen, in ERLAN /RT78/ geschrieben. Zur Verifikation des generierbaren Codegenerators wurde ein Umsetzer von CIMIC I nach nach CODE erstellt, der eine Zusammenarbeit zwischen vorhandenem Compileroberteil und generiertem Codeerzeuger auf der SIEMENS 306 erlaubt /EB 79/.

4.5 Betriebssystem /RO 80/

Die Aufgaben eines PEARL-Betriebssystems ("PBS") im Kontext der hier dargestellten Compiliersystem sind:

- Organisation der Parallelarbeit von Tasks (d.h. Abwickeln der Taskingaufrufe und Synchronisierung),
- Vornehmen von Einplanungen für Task-Operationen (d.h. Verwalten von Zeit- bzw. Interrupt-Schedules),

- Verwalten von Geräten,
- Verarbeitung von Geräte- bzw. Prozeß-Interrupts.

Die Algorithmen zur Lösung dieser Aufgaben sind prinzipiell maschinenunabhängig. Lediglich der Zwang zu effizienten Lösungen ließ viele Implementatoren bisher zum Assembler als Erstellungswerkzeug greifen. Hier wurde der Versuch unternommen, ein PEARL-Betriebssystem rechnerunabhängig zu formulieren, wobei berücksichtigt wurde, daß bestimmte Teil-Aufgaben nur rechnerabhängig zu realisieren sind.

Die "obere" Schnittstelle des Betriebssystem (zu den Laufzeitfunktionen hin) ist etwa durch Aufrufe der oben genannten Funktionen skizziert. Als "untere" Schnittstelle (zur Maschine hin) wurde ein Satz von Funktionen definiert (z.B. Retten und Restaurieren der Register einer Task), der anlagenabhängig zu erstellen ist. Diese Aufrufhierarchie ist in Fig. 10 dargestellt.

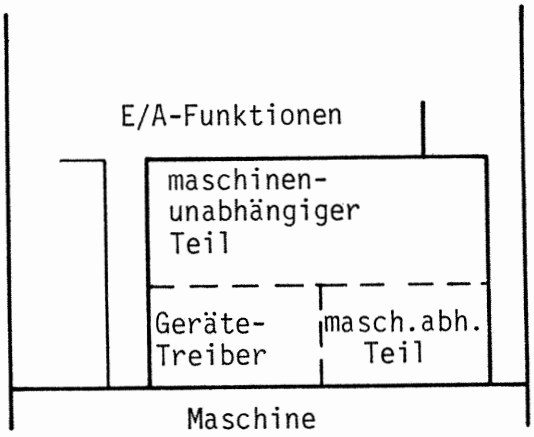

Fig. 10 Aufrufhierarchie des PEARL-Betriebssystems

Die Formulierung der PBS-Aufgaben erfolgte in einem abstrakten Assembler SPASS (Systemprogrammiersprache auf Assemblerebene). Das Betriebssystem ist modular aufgebaut: für jede PEARL-Funktion gibt es eine eigene Prozedur.

Die Erzeugung eines ablauffähigen Betriebssystem erfolgt auf die Weise, daß (vergl. Fig. 11)

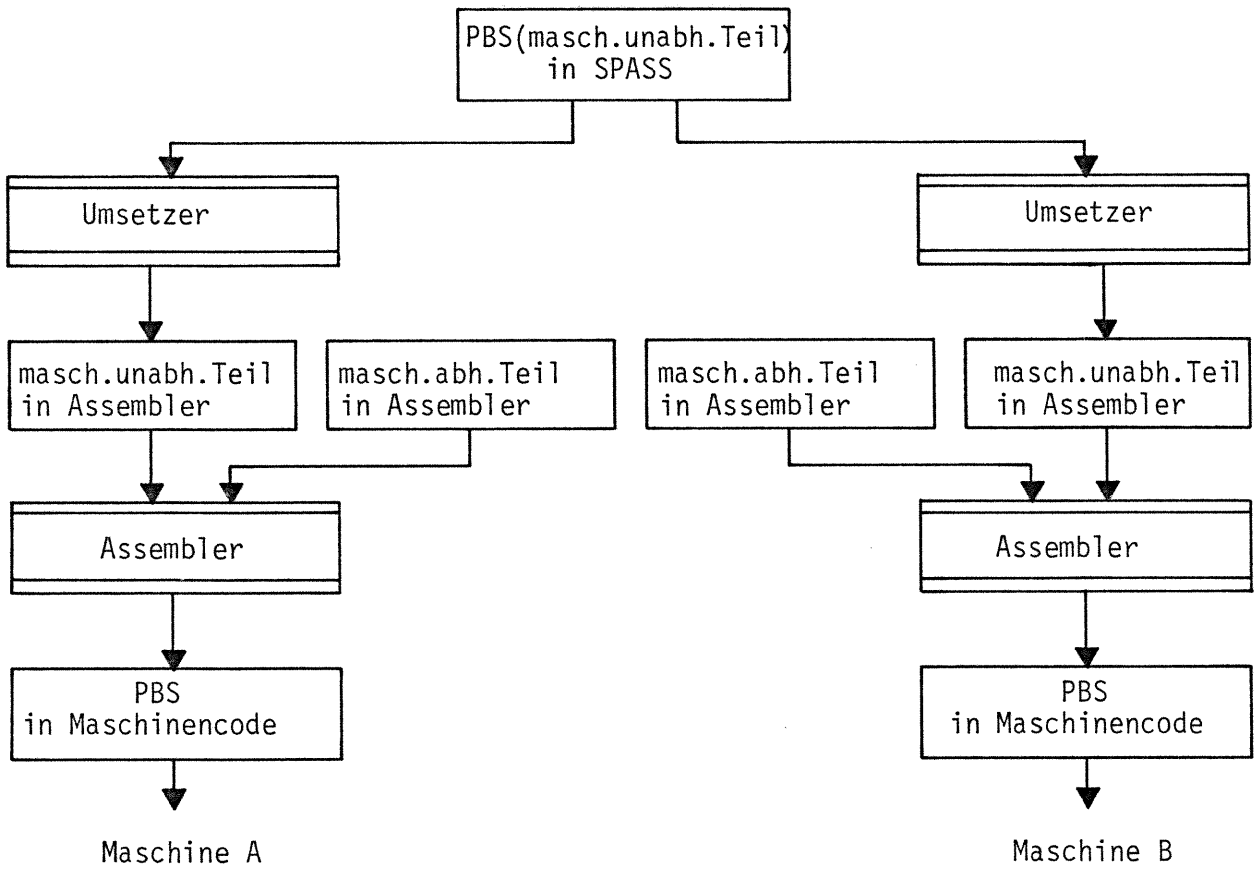

Fig. 11 Erzeugung des PEARL-BEtriebssystems (PBS) für verschiedene Maschinen

- ein Umsetzer von SPASS in den Code der Zielmaschine erstellt wird, soweit er noch nicht vorliegt,
- der in SPASS formulierte maschinenunabhängige Teil des PEARL-Betriebssystems hiermit übersetzt wird und
- die (ziel-) maschinenabhängigen Funktionen des PEARL-Betriebssystems erstellt werden.

Der Aufwand für diese drei Schritte ist wesentlich geringer als eine Neuerstellung des Betriebssystems. Der aufwendigste Schritt, die Erstellung des SPASS/Assembler-Umsetzers kann auf Grund der Struktur von SPASS leicht in einem Makrogenerator (z.B. Stage II) oder durch Programmierung in FORTRAN, PEARL etc. vorgenommen werden.

Die Erstellung der maschinenabhängigen Funktionen fällt kaum in Gewicht, da ihr typischer Umfang einige 100 Bytes beträgt.

Das PBS wurde mit dieser Technik auf drei Rechnern installiert, und zwar auf einer SIEMENS 306 /RO 79/, einer SIEMENS 310 /FP 80/ (siehe auch Kapitel 4) und einem Z80 Mikroprozessor /RO 78/ .
Ein Vergleich des portablen PEARL-Betriebssystems mit vorhandenen Betriebssystemen ist schwer zu ziehen, da herkömmliche Prozeßrechner-Betriebssysteme einen größeren Aufgabenbereich haben, als ein reines PEARL-Betriebssystem. Bei einem PEARL-Betriebssystem kann eine gewisse "Korrektheit" von Programmen vorausgesetzt werden, während bei herkömmlichen Betriebssystemen ein oft hoher Sicherheitsaufwand z.B. gegen fehllaufende Assemblerprogramme getrieben werden muß. Deswegen darf nicht verwundern, daß das PBS, wenn es alternativ zu einem herkömmlichen Betriebssystem eingesetzt wird, kürzer ist und schnellere Reaktionszeiten erlaubt (s. auch /RO 80/ und Kap. 8).

5. Die Implementation für eine SIEMENS 310 /FH79/

Hinter der 310-Implementation stand die Absicht, im Sinn einer Zwischenbilanz, möglichst viele der inzwischen entwickelten portablen Systemsoftware-Komponenten zur Erstellung eines neuen vollständigen Compiliersystems zu benutzen und Erfahrungen mit ihrem Einsatz zu gewinnen. Das Vorgehen bei der Implementation sah vor, an Stellen, wo (noch) keine portabel erstellten Komponenten vorlagen, maschinenabhängige Speziallösungen zu entwickeln. Eventuell vorhandene Hersteller-Systemsoftware war, wenn problemlos möglich, mitzuverwenden. Auf möglichst klare Schnittstellen war zu achten.

Vorteil dieses pragmatischen Vorgehens war, die portablen Komponenten in realistischer Umgebung auf eine Zielmaschine zu bringen.

5.1. Beschreibung /TR80, FP80, LP80/

Die SIEMENS 310 schien im Rahmen der Zielsetzung als Zielmaschine hinreichend typisch für die derzeit auf dem Markt befindlichen Kleinrechner zu sein. Als 16-bit-Registermaschine verfügt sie über wenig Hardware-Unterstützung bei der Arithmetik. Ihre Unterbrechungsstruktur macht sie für kleinere Prozeßanwendungen geeignet. An Geräten stand neben einer einfachen Standardperipherie ein modulares Prozeßperipheriesystem (z.B. mit Interruptregister, Digital-E/A, Analogeingabe) zur Verfügung. Sie war mit einem Standard-Betriebssystem ausgerüstet, das auch einen Realzeitbetrieb zuläßt (Interruptverarbeitung, Semaphoroperationen, etc.). Darüberhinaus war ein (Cross-) Assembler und ein Ladebinder verfügbar.

Um Vergleichsmöglichkeiten mit der 306-Implementation zu haben, wurde der schon auf der 306 laufende Compiler-Oberteil beibehalten. Zunächst nicht implementiert wurde die Dateiverwaltung und (wegen fehlender Hardware) die Graphische E/A.
Für die Anwender hatte dieses Verfahren den Vorteil, daß in der gleichen Sprache zwei verschiedene Rechner programmiert werden konnten. Der Codeerzeuger wurde wie an der 306

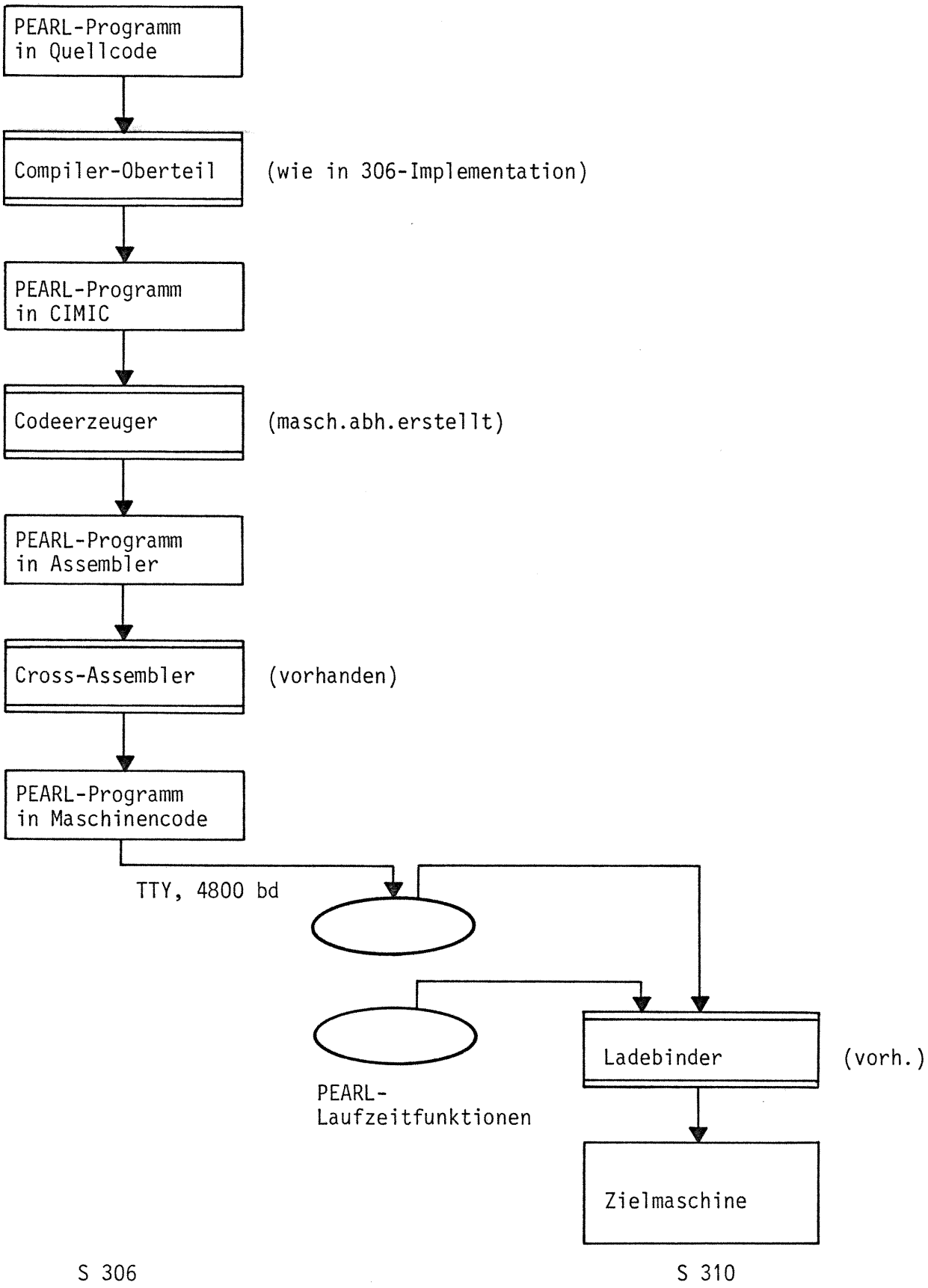

Fig. 12 Schematische Darstellung des Übersetzungsablaufs im SIEMENS 310 - Compiliersystem

maschinenabhängig erstellt. Als Werkzeug diente allerdings nicht mehr der Makrogenerator Stage II, sondern eine Systemsprache, was nach den Erfahrungen aus der Pilotimplementation geraten schien. Die verwendete Programmstruktur erlaubte reentrantfähige (Laufzeit-) Prozeduren. Zur Versorgung der 310 mit erzeugtem Code wurde eine Rechnerkopplung installiert (s. Fig. 12).

Die algorithmischen Laufzeitfunktionen wurden teils portabel, teils maschinenspezifisch erstellt. Mit diesen Laufzeitfunktionen, dem Codegenerator und dem Compiler konnte ein algorithmischer PEARL-Subset übersetzt werden. Dies wurde ausgenutzt, um die Standard-E/A-Laufzeitfunktionen zu übersetzen. Nach Erstellung der Funktionen zur Formatweiterschaltung war die Standard-E/A ablauffähig.

Als Betriebssystem stand das portable PEARL-Betriebssystem zur Verfügung. Um es in Betrieb zu nehmen, mußte es von dem abstrakten Assembler-Code in den 310-Code umgesetzt und durch maschinenabhängige Primitiv-Funktionen erweitert werden. Im Gegensatz zu früheren Testeinsätzen wurde es aber nicht auf die nackte Maschine aufgesetzt, sondern auf das vorhandene Standard-Betriebssystem. Dadurch waren zwar wiederum Effektivitätseinbußen unausweichlich, doch für den praktischen Betrieb von ausschlaggebender Bedeutung war, daß sämtliche Dienstprogramme mit benutzt werden konnten. Das erleichterte die Handhabung im Testbetrieb, das Binden, Laden und Starten einzelner Programme ungemein: Für das Binden und Laden wurde der Ladebinder des Herstellers benutzt. Fig.13 zeigt die Aufrufhierarchie der 310-Laufzeitbausteine.

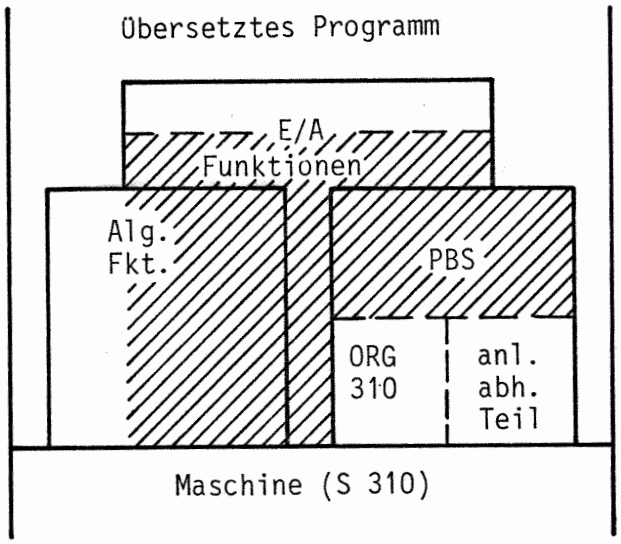

Fig. 13: Aufrufhierarchie der 310-Laufzeitbausteine
Portabel erstellte Lösungen sind gestrichelt angedeutet

Die Implementation des 310-Compiliersystems bestand also im wesentlichen aus Spezialanfertigungen für Codegenerator und Prozeß-E/A-Laufzeitfunktionen, sowie aus der Anpassung portabler Laufzeitbausteine für die Standard-E/A, die Algorithmik und das Betriebssystem.

5.2 Ergebnisse

Die 310-Implementation läßt sich am besten im Vergleich zur 306-Implementation untersuchen. Als Vergleichskriterien bieten sich der Implementationsaufwand und bestimmte Leistungsdaten an. Der Erstellungsaufwand für die 310-Implementation ist in Tabelle 2 dargestellt. Auffällig ist, daß etwa die Hälfte des Gesamtaufwandes auf die maschinenspezifische Erstellung des Codeerzeugers entfällt.

Systemkomponente	Aufwand in Mann-Jahren
Codeerzeuger	1
Laufzeitfunktionen	
Standard-E/A	0,25
Prozeß-E/A	0,25
Algorithmik	0,25
Betriebssystem(-Anpassung)	0,50
Gesamtsystem	ca 2 MJ

Tabelle2: SIEMENS 310-Implementationsaufwand
(Spezialanfertigungen und Anpassung portabler Komponenten)

Um einen Vergleich mit der 306-Implementation zu ermöglichen, müssen die 306-Aufwandsangaben um die Werte für Graphik, Filehandling sowie Test- und Bediensystem, die an der 310 nicht implementiert wurden, reduziert werden. Auch die Implementation der Prozeß-E/A an der 306 ist, da sie komfortabler und, wegen fehlender Treiber, aufwendiger war, nicht direkt vergleichbar. Berücksichtigt man dies, dann steht einem Aufwand von ca. 6 MJ für die 306-Implementation ein Aufwand von ca. 2 MJ für die 310-Implementation gegenüber.

Angesichts dieser Zahlen erhebt sich sofort die Frage, ob diese Einsparungen nicht durch mangelnde Effektivität zu teuer erkauft sind. Bevor diese Frage untersucht wird, muß daran erinnert werden, daß die Prämisse bei dieser Implementation die Aufwandsminimierung war, und deswegen keine Optimierungen berücksichtigt wurden, es sei denn, sie waren konzeptionell verankert.

Von der Bearbeitung der Standard-E/A-Laufzeitfunktionen war bekannt, daß sich der Effektivitätsverlust hinsichtlich einer Laufzeitvergrößerung kritischer auswirkt als hinsichtlich einer Speicherplatzerhöhung. Ein für Realzeitanwendungen typischer Test besteht z.B. darin, die Frequenz zu ermitteln, mit der ein Anwenderprogramm im Grenzfall durch Interrupts aus der Prozeßumwelt ("Interrupt-Reaktions-Programm") zyklisch ausgelöst werden kann. Diese Frequenz hängt natürlich von der Dauer des Interrupt-Reaktions-Programms ab. Legt man ein Programm zugrunde, das auch Reaktionsmöglichkeiten erlaubt und einen Betriebssystemaufruf sowie einige weniger zeitkritische Opterationen enthält und ca. 4 ms dauert, dann erreicht man damit eine Frequenz von ca. 100 Hz. Die Zeit vom Eintreffen eines Interrupts bis zum Ablauf des ersten PEARL-Statements im Anwenderprogramm beträgt also ca. 6 ms. Der entsprechende Wert bei der 306-Implementation liegt bei ca. 2.5 ms.

Die niedrigen Geschwindigkeiten kommen durch das "mitlaufende" Hersteller-Betriebssystem sowie die (verglichen mit der 306) relativ hohen Operationszeiten zustande und liegen bei einem Betriebssystem, das auf einen "nackten" Mikroprozessor aufgesetzt ist /RO80/ viel niedriger. Gezeigt wurde aber, daß mit

Hilfe portabler Systemsoftwarekomponenten mit verhältnismäßig geringem Aufwand ein brauchbares und komfortables Realzeit-Compiliersystem installiert werden kann.

Interessant ist noch der Vergleich der Übersetzungszeiten des 306- und des 310-Compiliersystems: Anstelle des Makrogenerators Stage II wurde diesmal eine Systemsprache (ERLAN) zur Erstellung des Codeerzeugers verwendet. Anstelle des residenten Assemblers wurde ein Cross-Assembler des Herstellers verwendet. Die Übersetzungszeit eines 500-Zeilen-Programms, das wieder zum Test herangezogen wurde, betrug nunmehr nur noch 20 min, wovon wiederum der (Cross-) Assembler den Löwenanteil beanspruchte, sank also um 1/3. Bei weiteren Arbeiten dieser Art wurde deshalb der Stage-II-Makrogenerator nicht mehr in Betracht gezogen.

5.3 Anwendungen

Die SIEMENS 310 mit ihem PEARL-Compiliersystem war für den Einsatz in der Laborautomatisierung gedacht. In zwei Anwendungsfällen wurde untersucht, ob eine derart kleine Maschine auch für nicht-triviale Probleme geeignet ist.
Es handelte sich hierbei um

- die Entwicklung einer automatischen Ablaufsteuerung zur Messung optischer Spektren
- die Messung der Zugfestigkeit von Werkstoffen.

Die erste Entwicklung erfolgte in Zusammenarbeit mit der Abteilung Molekülphysik des PIE, die zweite in Zusammenarbeit mit einem Labor der Fa. Siemens AG.

Es zeigte sich hierbei, daß einerseits sowohl die verwendete Art der Cross-Compilierung praktikabel war und daß andererseits PEARL gegenüber BASIC deutliche technische Vorteile brachte. Die beiden Anwendungsfälle werden im folgenden kurz beschrieben.

- Entwicklung einer automatischen Ablaufsteuerung zur Messung optischer Spektren /BE81A/

  Es wurde eine Testapparatur aufgebaut, die es ermöglichte, rechnergesteuert optische Spektren zu messen. Zur Steuerung des Experimentes wurden zwei elektronische Schaltungen, eine Schrittmotorsteuerung und ein Impulszähler entwickelt, die sowohl "off-line" als auch in Verbindung mit dem Rechner betrieben werden können. Am Rechner waren dazu zwei Prozeß-schnittstellen, eine Digitalausgabe und eine kombinierte Digitaleinausgabe vorhanden (vgl. Fig. 14).

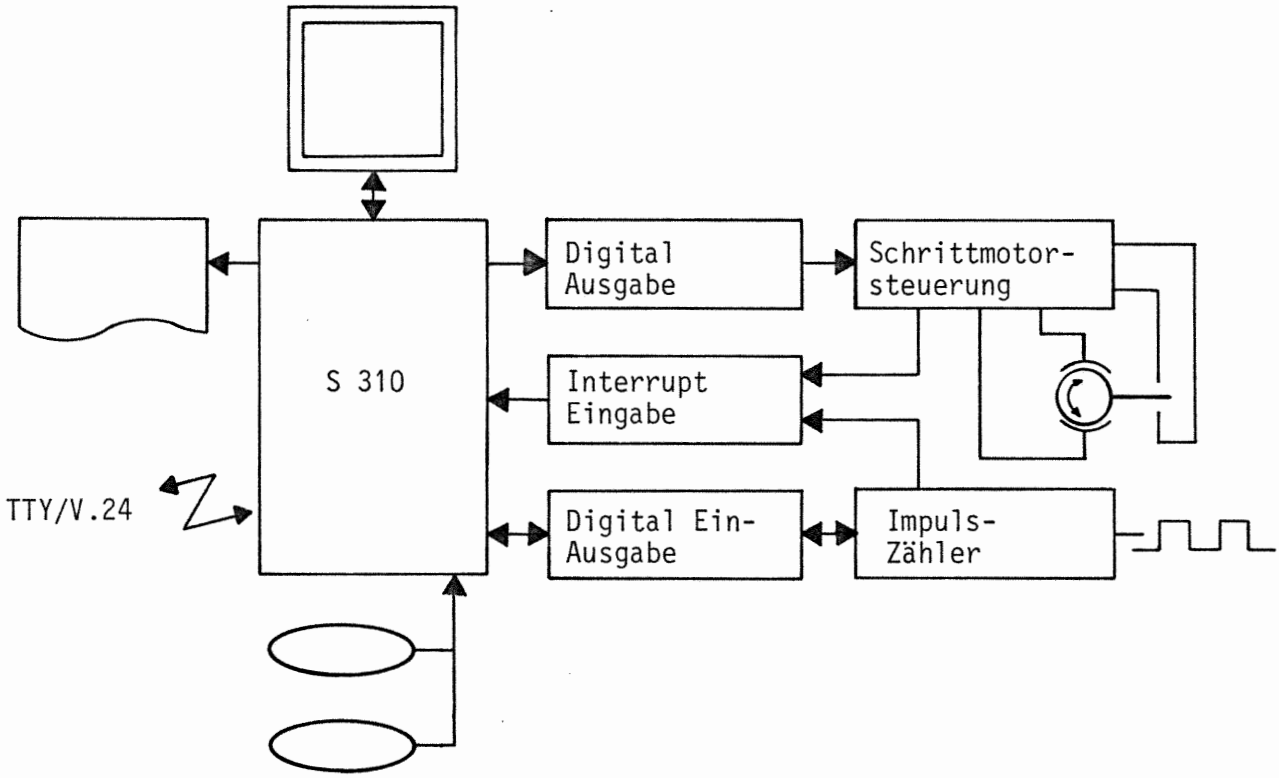

Fig. 14: Schematischer Aufbau des automatischen Meßsystems

Die Schrittmotorsteuerung hat die Aufgabe, die von der 16-Bit-Digitalausgabe gelieferten Daten zu interpretieren und in entsprechende Schrittimpulse für den Motor umzuwandeln. Sie ermöglicht in Verbindung mit der SIEMENS 310, den Motor in beiden Drehrichtungen um ein definiertes Intervall aus n Schritten zu bewegen, bzw. den Motor bis an ein Ende

eines durch Endschalter eingegrenzten Bereichs zu fahren. Die ordnungsgemäße Erledigung eines Fahrauftrages oder die Auslösung eines Endschalters werden durch Interrupt an den Rechner gemeldet.

Der steuerbare Impulszähler kann in zwei Arten eingesetzt werden. Er kann die Zeit messen, die für die Aufsummation einer gewissen Impulszahl vergeht, oder innerhalb eines Zeitintervalls eingehende Impulse zählen. Die Betriebsartvorwahl bzw. die Angabe der betreffenden Zählgrenze wird vom Rechner mit der Digitalausgabe eingestellt. Nach erfolgter Messung liefert das Gerät das Zählergebnis an die Digitaleingabe der SIEMENS 310.

Auf der 310 läuft ein in PEARL geschriebenes Meßprogramm. Es hat die Aufgabe, an die Schrittmotorsteuerung die benötigten Daten in binärer Form zu liefern, sowie den Impulszähler vorzuwählen und das Zählergebnis auszulesen. Die Meßdaten werden in Feldern organisiert und auf einer Datei auf Floppy-Disk gespeichert. Parallel zur Messung werden die Zählraten auf dem Drucker in Form einer Wertetabelle ausgegeben. Zur Anfertigung von Zeichnungen können die Daten zur CYBER des Regionalen Rechenzentrums Erlangen transferiert und dort mit einem Plotter dargestellt werden. Der Aufbau und Einsatz des beschriebenen Meßsystems war problemlos möglich. Die Zeit zur Erstellung des Meßprogramms betrug ca. zwei Monate.

- Messung der Zugfestigkeit von Werkstoffen /TR80/

  Die 310 war mit einer Apparatur zur Messung der Zugfestigkeit von Werkstoffen gekoppelt. Zum Betrieb dieser Konfiguration gab es bereits ein fertiges, in BASIC geschriebenes Programm, das eine gute Vergleichsmöglichkeit bot.

  Die Meß-Apparatur besteht aus einer "Zugprüfmaschine", die ihre Daten, nämlich "Kraft" und "Dehnung", über eine Analogeingabe an die 310 weitergibt. Die bei der Kraft- bzw. Dehnungsmessung eingestellten Meßbereiche sowie die Betriebsbereitschaft der Zugprüfmaschine können über eine Digitaleingabe

der 310 gelesen werden. Zum Dialog mit dem Bediener steht ein Bildschirmterminal zur Verfügung. Die Ergebnisse werden auf einem Drucker ausgegeben.

Je nach Dehnungsaufnehmer sind mit dieser Apparatur folgende Messungen möglich:

- reine E-Modul-Messung
- Zugfestigkeits- und Bruchdehnungsmessung (mit Grobbestimmung des E-Moduls)
- Messung der Zugfestigkeit ohne Dehnungsmessung.

Angesichts der Erfahrungen mit dem BASIC-Programm (lange Meßzeiten, Auswertung in einem separaten Lauf nach der Aufnahme) bestand für das PEARL-Programm die Forderung, die Auswertung bereits parallel zur Datenaufnahme vorzunehmen, um z.B. die Erfüllung eines Abrißkriteriums sicher feststellen zu können.

Der Hauptteil des erstellten PEARL-Programms bestand aus zwei Tasks. Die Meßtask bediente die Analog- und die Digitaleingabe und diente neben der Aufnahme der Meßdaten auch zur Erkennung eines Meßwertüberlaufs, des Meßbereichs und des STOP-Zustands der Zugprüfmaschine. In der Auswertetask erfolgte die Bestimmung von E-Modul, Zugfestigkeit und Bruchdehnung. Außerdem diente sie zum Erkennen des Abriß-Kriteriums.

Während das BASIC-Programmm 0.6 sec. für einen Meßpunkt (Wertepaar Kraft/Dehnung) benötigte und demzufolge auch ziemlich ungenaue Analysen lieferte, wurde mit dem PEARL-Programm eine Zeit von 75 msec pro Meßpunkt erreicht, wovon allein 70 msec durch die Wandlungszeit der Analogeingabe bedingt waren. Das für diesen Zweck erstellte PEARL-Programm bestand aus ca. 1260 Quellzeilen. Die Zeit für Erstellung und Test des PEARL-Programms betrug ca. 2 Mann-Monate.

6. Überarbeitung einiger portabler Komponenten

Der Einsatz der entwickelten portablen Komponenten hatte gezeigt, daß der eingeschlagene Weg richtig war. Einiges blieb zwar grundsätzlich zu wünschen übrig: So gab es keinen zeitgemäßen Compileroberteil und nur die inzwischen "historische" Zwischensprache CIMIC-I .
Hiermit mußte man jedoch leben.
Nichtsdestoweniger waren mindestens einige Detailkorrekturen sinnvoll und notwendig, um die vorliegenden Komponenten in größerem Umfang in der Praxis einzusetzen.

- Es mußte ein modernes (PEARL-) Sprachgewand zur Verfügung stehen,
- Vorhandene Pakete mußten auf den neuen Sprachsubset umgestellt werden,
- Die bisher angewandte Technik des PEARL-Betriebssystems (Aufsetzen auf das Hersteller-Betriebssystem) mußte geändert werden.

Alle diese Aktionen standen in Zusammenhang mit der geplanten PEARL-Implementation für einen Z80-Mikrorechner, für die ein DFG-Forschungsvorhaben in Aussicht stand (siehe Kap. 8).
Für diese Implementation war es angebracht, die bisher erfolgreich eingesetzten Komponenten und Techniken "abzurunden".
Es wurde deshalb

- Der AVIONIK-PEARL-Compiler der Fa. ESG, der mit seinem Sprachstand von 1977 /PL 77/ weitgehend BASIC-PEARL entspricht, installiert. Compiler anderer Technologie hatten sich als nicht handhabbar gezeigt /LO 77/.
- Das portable Standard-E/A-Paket wurde auf AVIONIK- bzw. BASIC-PEARL umgestellt.
- Das portable PEARL-Betriebssystem wurde für einen Einsatz auf einem "nackten" Mikroprozessor vorbereitet und funktionell erweitert.

- Installation AVIONIK-PEARL-Compileroberteil /BE 81B/

  Der Compileroberteil wurde uns freundlicherweise von Fa. ESG, München, einschließlich internem und externem Dokumentationsmaterial unentgeltlich zur Verfügung gestellt. Er ist nach der gleichen Technik in FORTRAN erstellt wie der ASME-Stufe-I-Compiler. Als Zwischensprache erzeugt er CIMIC-AV /CV 80/. Probleme bei der Installation waren also nicht zu erwarten.

  Die Inbetriebnahme erfolgte zunächst auf der Cyber 173 des Regionalen Rechenzentrums, da diese Maschine für die geplanten Cross-Compilationen hinreichend geeignet schien. Da bei dieser Maschine der verfügbare Speicherplatz nicht so engen Grenzen unterworfen ist wie bei den ASME-I-Übersetzungsmaschinen, wäre eine lineare (unsegmentierte) Fassung möglich gewesen. Da dies aber wegen hinreichender Priorität im Timesharing-System nicht sinnvoll erschien, wurde eine Segmentierung entsprechend der Hauptsegmente vorgenommen. Auf diese Weise belegt der Compiler ca. 40 k Worte. Seine Übersetzungsgeschwindigkeit beträgt ca. 10 Statements pro CPU-Sekunde. Seine Arbeitsweise (z.B. auf welchem Drucker innerhalb des Netzes die Listen erscheinen sollen) kann über Kommandoprozeduren beliebig gesteuert werden.

- Überarbeitung des Standard-E/A-Laufzeitpakets /KL 81/

  Das Ein-/Ausgabe-Paket lag in ASME-I-PEARL vor. Zur Erhaltung der Kompatibilität mit den vor Ort laufenden Compilern (AVIONIK auf der Cyber des RRZE, BASIC-PEARL auf der Siemens R30) wurde es auf AVIONIK- bzw. BASIC-PEARL umgestellt.
  Dabei wurden bekannte Fehler und strukturelle Schwächen verbessert. Außerdem wurde das Paket funktionell dahingehend erweitert, daß es inzwischen nicht nur die Formatabarbeitung, sondern auch die Format-Weiterschaltung umfaßt. Diese Erweiterung wurde in CIMIC-AV

formuliert. Sie ist damit CIMIC-abhängig, aber zielmaschinenunabhängig.

- Überarbeitung des PEARL-Betriebssystems /KU 81/

  Das portable PEARL-Betriebssystem wurde in größerem Stil erstmals im Rahmen der SIEMENS 310-Implementation eingesetzt. Es wurde dabei auf das vorhandene Betriebssystem ORG 310 aufgesetzt, von dem aber nur wenige Funktionen ausgenutzt wurden (Gerätetreiber). Genaue Tests zeigten wegen dieser "Aufsetztechnik" ein eher durchschnittliches Zeitverhalten bei zeitkritischen Reaktionen.
  Es wurde deshalb das PEARL-Betriebssystem so umstrukturiert, daß es auch auf einer "nackten" Maschine über alle wesentlichen Funktionen verfügt. Insbesondere wurde es um einige Funktionen erweitert (wie z.B. das für die Praxis nötige Fremd-Terminieren von Tasks).

7. Schlußbemerkungen

- Was wurde erreicht?

Ziel des Vorhabens war die Entwicklung eines weitestgehend portablen Programmiersystems auf der Grundlage von PEARL. Dieses Ziel wurde erreicht. In mehreren Stufen konnte demonstriert werden, daß

- PEARL überhaupt mit begrenztem Aufwand für kleine Rechner implementierbar ist (ASME-Stufe-I), was à priori durchaus offen war;
- es möglich ist, Komponenten im traditionell maschinennahen Bereich maschinenunabhängig zu formulieren, was nicht nur die Voraussetzung für Portierbarkeit darstellt, sondern auch die Probleme durchschaubar macht;
- es möglich ist, portable maschinennahe Komponenten mit Erfolg in PEARL-Implementationen einzusetzen, wobei ein deutlich verringerter Erstellungs-/Adaptionsaufwand zu verzeichnen ist;
- es gelingt, mit portablen maschinennahen Komponenten Leistungen zu erreichen, die sich im Rahmen maschinenabhängiger Lösungen halten;
- es bei Beachten bestimmter Randbedingungen auch möglich ist, mit portablen Lösungen sogar die kleinsten (8bit-) Mikroprozessortypen zu versorgen und Leistungen zu erreichen, die Minirechnern entsprechen (siehe Nachtrag).

- Welche Erfahrungen wurden gemacht?

Rückwirkend betrachtet kann eine Zusammenarbeit zwischen Instituten und der Industrie wie in Stufe I der ASME als fruchtbar betrachtet werden. Einerseits war die Arbeitsaufteilung gut gewählt (der industrielle Partner übernahm die Realisierung bekannter Aufgaben, die Institute arbeiteten mehr im Neuland), andererseits ging der Austausch von Impulsen und Erfahrungen leicht vonstatten. Voraussetzung für diese Zusammenarbeit war natürlich der gute Wille, der Kompromisse leicht finden ließ.

Trotz "Einschlafens" der ASME gelang am PIE dann doch noch, nahezu wie geplant, die Entwicklung der meisten vorgeschlagenen Komponenten des Compiliersystems. Gerade deswegen ist zu bedauern, daß eine Überarbeitung wesentlicher, nicht am PIE entwickelter Teile, wie Compileroberteil bzw. CIMIC nicht erfolgen konnte, obwohl es weder an entsprechenden Erfahrungen noch an detaillierten Vorschlägen mangelte. So mußte an manchen Stellen der Nachweis der Effizienz der am PIE entwickelten Komponenten unterbleiben. Außerdem wären gerade im Hochschulbereich herstellerunabhängige Compiler in leicht beherrschbarer Technologie, die BASIC-PEARL übersetzen und eine vernünftige Zwischensprache absetzen, für die Verbreitung von PEARL von großem Nutzen gewesen.

Ungeachtet dessen sei abschließend dem Projektträger gedankt, der stets um eine Kontinuität der Arbeiten bemüht war, sowie den Herren Prof. Dr. N. Fiebiger und Prof. Dr. A. Hofmann vom Physikalischen Institut, die für gute Arbeitsbedingungen sorgten.

8. Nachtrag /HO 81/

Obwohl die erfolgreiche Verwendbarkeit des entwickelten Baukastensystems mit der Siemens 310 PEARL-Implementation demonstriert werden konnte, blieb der Nachweis der Brauchbarkeit solcher Komponenten für Mikroprozessoren einem nachfolgenden DFG-Forschungsvorhaben des PIE und des Regionalen Rechenzentrums Erlangen vorbehalten. Unter dem Thema "Programmiertechnik für Mikroprozessoren"(im Rahmen einer weitergespannten Aufgabenstellung) wurde hierbei inzwischen ein PEARL-Cross-Compiliersystem für einen Z80 entwickelt, das folgende portable Komponenten aus dem PDV-Vorhaben weiterverwendet:

- den AVIONIK-PEARL-Compiler auf der Cyber des Regionalen Rechenzentrums
- das portable PEARL-Betriebssystem und
- die portable Standard-EA,

ergänzt um einen weitgehend modular aufgebauten Codegenerator, verschiedene Laufzeitfunktionen und Treiber.

Im Gegensatz zur 310-Implementation wurde hier das PEARL-Betriebssystem auf den "nackten" Z80 aufgesetzt, so daß z.B. Interrupt-Reaktionszeiten und Betriebssystem-Durchlaufzeiten wesentlich kürzer als bei der 310-Implementation sind und in der Größenordnung von speziellen PEARL-Implementationen von Mini-Rechnern liegen.
Mit dieser Leistungsfähigkeit scheint es ein brauchbares Werkzeug zum Betreiben von 8 bit-Rechnern der "Billigst"-Klasse darzustellen.

9. Literatur

BE77 H.B. Berner, J. Biewald, P. Elzer, P. Holleczek, R. Lauber, W. Lindstedt, K. Pelz, F.J. Prester, J. Reh, R. Rössler, F. Siller:
VEPAS, Verfahren zur Erstellung von Prozeßautomatisierungssoftware
Gesellschaft für Kernforschung mbH, Karlsruhe, PDV-Entwicklungsnotiz E101, 1977

BE81A R. Besold:
Entwicklung einer automatischen Ablaufsteuerung zur Messung optischer Spektren und Anwendung auf Gasentladungsspektren einfacher Kohlenwasserstoffe
Diplomarbeit, Phys. Inst. Universität Erlangen-Nürnberg, 1981

BE81B H. Beck:
Die Installation des AVIONIK-PEARL-Compileroberteils auf der CYBER des Regionalen Rechenzentrums
Studienarbeit, Inst. f. Informatik II, Univ. Erlangen-Nürnberg, 1981

CI76 Mühlhahn:
Spezifikation CIMIC/1
Gesellschaft für Kernforschung mbH, Karlsruhe, Bericht KFK-PDV 75, 1976

CV80 Beschreibung der Zwischensprache CIMIC/AV
Elektronik System Gesellschaft, München, Internes Arbeitspapier, 1980

EB79 W. Eberlein:
Implementation eines Zwischensprache-Transformators
Diplomarbeit, Inst. f. Informatik II, Univ. Erlangen-Nürnberg, 1979

EH75 P. Elzer, P. Holleczek:
ASME-Compiler wird der Öffentlichkeit vorgestellt
Regelungstechnik 12 (1975) 433-436

FH79 A. Fleischmann, P. Holleczek, F.J.Prester, W.Lindstedt, M. Trautner:
Ein PEARL-Cross-Compiliersystem für eine Siemens 310
Regelungstechnische Praxis, 21. Jahrgang (1979) Heft 12

FP80 A. Fleischmann, F.J. Prester:
Ein PEARL-Betriebssystem für die Siemens 310
Gesellschaft für Kernforschung mbH, Karlsruhe, PDV-Entwicklungsnotiz E143, 1980

HO76A P. Holleczek:
Das Filehandling für den Erlanger ASME-PEARL-Subset
Gesellschaft für Kernforschung mbH, Karlsruhe, PDV-Entwicklungsnotiz E83, 1976

HO76B P. Holleczek:
Stufe-2-ASEME-PEARL-Compiliersystem-Grundideen
Gesellschaft für Kernforschung mbH, Karlsruhe, PDV-Entwicklungsnotiz E89, 1976

HO77 P. Holleczek:
Der PEARL-Systemteil, seine Darstellung nach ersten Erfahrungsauswertungen
Gesellschaft für Kernforschung mbH, Karlsruhe, PDV-Entwicklungsnotiz E96, 1977

HO81 P. Holleczek:
Ein portables PEARL-Kompiliersystem für den Z80-Mikrorechner.
PEARL-Rundschau, Band 2, Nr. 2, 1981

HP81 P. Holleczek, K. Pelz:
PEARL-Kurs und Praktikum an der Universität Erlangen-Nürnberg
PEARL-Rundschau, Band 2, Nr. 4, 1981

KL81 G. Klebes:
Anpassung portabler E A-Funktionen für ein PEARL-Compiliersystem und Vergleich der verschiedenen Erstellungssprachen
Studienarbeit, Inst. f. Informatik II, Univ. Erlangen-Nürnberg, 1981

KU81 R. Kummer:
Anpassung eines rechnerunabhängigen PEARL-Betriebssystems an einen Z80-Mikrorechner
Studienarbeit, Inst. f. Informatik IV, Univ. Erlangen-Nürnberg, 1981

LA78 K. Lampe:
Erstellung der Standard-EA-Laufzeitroutinen des PEARL-ASME-Stufe 1-Compiliersystems in PEARL
Diplomarbeit, Inst. f. Informatik II, Univ. Erlangen-Nürnberg, 1978

LI76 W. Lindstedt:
Bedienung von CAMAC-Peripherie mit PEARL-Stufe 1 an der SIEMENS 306
Gesellschaft für Kernforschung mbH, Karlsruhe, PDV-Entwicklungsnotiz E78, 1976

LI80 W. Lindstedt:
Portabilität algorithmischer Laufzeitprogramme in:
Portable Software
Tagung 1/1980 des German Chapter of ACM, H.J. Schneider (Hrsg.), B.G. Teubner, Stuttgart, 1980

LI81 W. Lindstedt:
Ein Verfahren zur Erstellung portabler algorithmischer Laufzeitprogramme
Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung der Univ. Erlangen-Nürnberg, Band 14, Nr. 5, Juni 1981

LO77 Armgard von Löhneysen:
Implementation eines Code-Generators für CIMIC/C
Diplomarbeit, Inst. f. Informatik II, Univ. Erlangen-Nürnberg, 1977

LP80 K. Lampe, F.J. Prester:
Die rechnerunabhängigen Laufzeitroutinen der PEARL-Standard-E/A und ihre Anpassung für die Siemens 310 und 306
Gesellschaft für Kernforschung mbH, Karlsruhe, PDV-Entwicklungsnotiz E146, 1980

PE80 K. Pelz:
Konzept eines einfachen Codeerzeuger-Generators / erste Erfahrungen
Tagung 1/1980 des German Chapter of ACM, H.J. Schneider (Hrsg.)
B.G. Teubner, Stuttgart, 1980

PL73 K.H. Timmesfeld, B. Schürlein, P. Rieder, K. Pfeiffer, G. Müller, K. Kreuter, P. Holleczek, V. Haase, P. Elzer, S. Eichentopf, B. Eichenauer, J. Brandes:
PEARL - Vorschlag für eine Prozeß- und Experimentautomatisierungssprache
Gesellschaft für Kernforschung mbH, Karlsruhe, Bericht KFK-PDV1, 1973

PL76 Gruber, Inderst, Piche:
Programmieranleitung für das ASME-1-PEARL-Subset
Gesellschaft für Kernforschung mbH, Karlsruhe, Bericht KFK-PDV 100, 1976

PL77 PEARL-Subset for AVIONIC Applications, Language Description
Elektronik-System-Gesellschaft mbH, München, 1977

PR76 F.J. Prester:
Die Standard EA für den Erlanger ASME-PEARL-Stufe-1-Subset
Gesellschaft für Kernforschung mbH, Karlsruhe, PDV-Entwicklungsnotiz E76, 1976

PR77 F.J. Prester:
Die graphische Ein-/Ausgabe des Erlanger ASME-PEARL-Subsets
Gesellschaft für Kernforschung mbH, Karlsruhe, PDV-Entwicklungsnotiz E104, 1977

PR80 F.J. Prester:
Generierbare Treiber für die graphische Ein-/Ausgabe
Tagung 1/1980 des German Chapter of ACM, H.J.Schneider (Hrsg.)
B.G. Teubner, Stuttgart, 1980

RO77 R. Rössler:
Laufzeithilfen für PEARL; in: Testen und Verifizieren von Prozeßrechnersoftware
Gesellschaft für Kernforschung mbH, PDV-Entwicklungsnotiz E97, 1977

RO76 R. Rössler:
Programmstruktur und Laufzeitverwaltung für den PEARL-Subset (Stufe 1) der ASME in der Zielmaschine S306.
Gesellschaft für Kernforschung mbH, Karlsruhe, PDV-Entwicklungsnotiz E75, 1976

Ro78 R. Rössler:
PEARL-Betriebssystem für den Z80
Gesellschaft für Kernforschung mbH, Karlsruhe, PDV-Entwicklungsnotiz E119, 1978

RO79 R. Rössler:
Betriebssystemstrategien zur Bewältigung von Zeitproblemen in der Prozeßautomatisierung
Dissertation, Universität Stuttgart, 1979

RO30 R. Rössler:
Ein portables Betriebssystem auf der Basis einer Makrosprache
Tagung 1/1980 des German Chapter of ACM, H.J. Schneider (Hrsg.)
B.G. Teubner, Stuttgart, 1980

RT78 R. Rössler, M. Trautner, K. Pelz, W. Lindstedt:
ERLAN - Sprachbeschreibung und Compiliersystem
Forschungsbericht, Phys. Inst. Univ. Erlangen-Nürnberg, 1978

SC75 K. Schenk:
Sprachvergleich von Echtzeitprogrammiersprachen anhand zweier Probleme der kernphysikalischen Experimentdatenverarbeitung
Diplomarbeit, Inst. f. Informatik IV, Univ. Erlangen-Nürnberg, 1975

SC79 R. Schüler:
Entwicklung eines Auswerteprogramms für mehrdimensionale Impulshöhenspektren und dessen Anwendung bei dem Korrelationsexperiment ^{208}Pb $(\alpha,\alpha'\gamma)$
Zulassungsarbeit, Phys. Inst., Universiät Erlangen-Nürnberg, 1979

RL76 H.J. Trebst, W. Lindstedt:
Ein CAMAC-Controller für die S306
Forschungsbericht, Phys.Inst. Univ. Erlangen-Nürnberg, Mai 1976

TR80 M. Trautner:
Codegenerator und Systembeschreibung der Siemens 310-PEARL-Implementation
Gesellschaft für Kernforschung mbH, Karlsruhe, PDV-Entwicklungsnotiz E142, 1980

TR80 M. Trautner:
ZUG - Ein PEARL-Programm zur Messung der Zugfestigkeit von Werkstoffen
Physikalisches Institut der Univ. Erlangen-Nürnberg, interner Bericht, Januar 1980