RUHR-UNIVERSITÄT BOCHUM

Arbeitsberichte
des
Rechenzentrums

Direktor: o.Prof. Dr. Hartmut Ehlich

Nr. 7808 ISSN 0341-0358

Nachdruck des Arbeitsberichtes Nr. 2 des AK 4
der STARG 440
"Anwendungssoftware künftiger
Groß-Rechnersysteme"

Bochum im Oktober 1978

1. Auflage 1978.

ISSN 0341-0358

Rechenzentrum der Ruhr-Universität Bochum
Universitätsstr. 150, Gebäude NA
Postfach 102148

4630 Bochum 1

E I N L E I T U N G

Im Laufe des Jahres 1977 haben Vertreter folgender Institutionen

AEG-Telefunken, FB Weitverkehr und Kabeltechnik, Backnang
Großrechenzentrum für die Wissenschaft in Berlin
AEG-Telefunken, Großrechenzentrum Berlin
Ruhr-Universität Bochum
Finanzverwaltung des Landes NRW, Düsseldorf
Universität Hamburg
Regionales Hochschulrechenzentrum Kaiserslautern
Gesamthochschule Kassel (Vorsitzender)
Leibniz-Rechenzentrum der Bayer. Akademie d. Wiss., München
Universität Stuttgart (TH)
Eberhard-Karls-Universität Tübingen

im Arbeitskreis 4 der ständigen Arbeitsgruppe STARG440 das vorliegende Papier erarbeitet. Auf ihrer 39. Sitzung am 29.9.1978 beschloß die STARG, die derzeit 36 Mitglieder von TR440/445 Rechner betreibenden Institutionen (Anlagenwert etwa 500 Mio DM) vertritt, den Bericht zu veröffentlichen. Insbesondere wurde jedes Mitglied der STARG ermächtigt, von dem Bericht Gebrauch zu machen im Rahmen der Beschaffung des eigenen Rechnernachfolgesystems.

Der vorliegende Nachdruck gibt die Originalfassung wieder.

H. Ehlich, H. Zoller

Arbeitsbericht Nr. 2 des AK4 der STARG

Stand vom 29.9.78

Anwendungssoftware künftiger Groß-Rechnersysteme

Inhalt

0. Vorbemerkungen

Das vorliegende Papier wurde von einem Arbeitskreis erstellt, der - von der STARG440 im Frühjahr 1977 eingesetzt - den Auftrag hatte, sich mit Anwendungssoftware künftiger Großrechnersysteme zu befassen. Das Ziel ist, Anforderungen für Entwicklung und Einsatz von Anwendungssoftware aufzuzeigen und inzwischen bekannt gewordene oder schon heute absehbare Fehler vermeiden zu helfen.

Die 1970 gegründete STARG (Ständige Arbeitsgruppe der TR440-Rechenzentren) trifft sich regelmäßig, um Erfahrungen beim Betrieb der Großrechner TR440 auszutauschen und Anforderungen für die Weiterentwicklung des Rechnersystems zu formulieren. In neuerer Zeit treten dabei Probleme beim Einsatz von Anwendungssoftware immer mehr in den Vordergrund. Dies beruht zu einem gewissen Teil darauf, daß Rechner verstärkt im Verbund betrieben werden. Im übrigen führen die anstehenden Beschaffungen von Nachfolgerechnersystemen zu den im vorliegenden Papier niedergelegten Forderungen.

Die STARG wendet sich mit diesem Bericht an alle Hersteller von Anwendungssoftware. Es sind also neben Rechnerherstellern auch Softwarehäuser und die Programmierabteilungen der Rechenzentren selbst angesprochen.

Standardisierungsforderungen im Bericht sind zunächst als Standardisierungsforderungen innerhalb eines bestimmten Rechnersystems zu verstehen. Darüberhinaus sollten alle Hersteller von Rechnersystemen und/oder Anwendungssoftware von geeigneter Instanz zu einer globalen Standardisierung bzw. Normierung gedrängt werden.

Wir wollen unter "Rechnersystem" Großrechneranlagen, Verbunde größerer und/oder kleinerer Rechenanlagen oder auch größere Verbundnetze verstehen.

Als "Anwendungssoftware" werden Programmsysteme verstanden, die auf verschiedenen solchen Rechnersystemen benutzbar sein sollen. Dabei treten die Rechenanlage(n) und ihr spezielles Betriebssystem aus Benutzersicht in den Hintergrund. Der Benutzer hat meist (nur noch) Bedienerkenntnisse für ein bestimmtes Programmsystem; auf Kenntnisse über Rechenanlage und Betriebssystem kann er zunehmend verzichten. Dieser in immer stärkerem Maße feststellbare Trend impliziert eine Struktur innerhalb der insgesamt mit Anwendungssoftware Befaßten, in der sich eine gewisse Hierarchie festellen läßt. Letztere ergibt sich u.a. aus der

Anzahl der von einer bestimmten Tätigkeit Betroffenen.

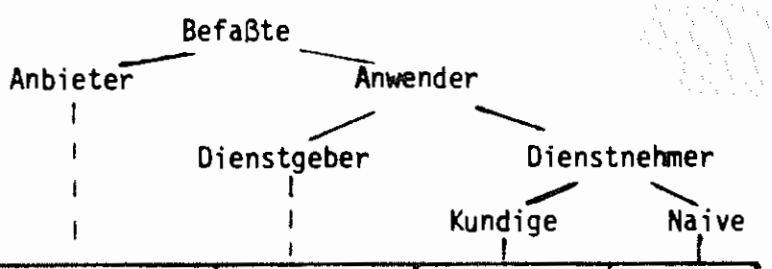
Die Struktur läßt sich wie folgt beschreiben:

Die Grobstruktur besteht aus der Einteilung in Hersteller (hier "Anbieter" genannt) und "Anwender" von Anwendungssoftware. Neben der Durchführung von Adaption, Verifikation und Pflege haben beide Gruppen die Aufgabe, Ausbildung, Werbung und Beratung für unterschiedliche Zielgruppen wahrzunehmen.

Bei den Anwendern ist noch weiter zu untergliedern und eine Unterscheidung zwischen Rechenzentrum (hier "Dienstgeber" genannt) und Benutzern ("Dienstnehmer") zu treffen. Bei letzteren sind "Kundige" (Berater, Administratoren etc.) und "Naive" zu unterscheiden.

Tätigkeiten und Beteiligte lassen sich in ein Schema bringen, aus dem eine Hierarchie in folgendem Sinn ersichtlich wird: Mit steigender Gruppennummer (Stufe) wächst die Anzahl der Beteiligten (und damit auch die Menge eingesetzter Manpower); Nachlässigkeiten auf einer Stufe führen zu vervielfältigten Effizienzverlusten auf allen darunter liegenden Stufen hinsichtlich Personaleinsatz, Anlagennutzung, Vermarktung usw.. Im nachfolgend aufgezeichneten Schema ist durch Kreuz gekennzeichnet, welche Gruppen welche Tätigkeiten ausführen; falls eine Tätigkeit ein Dienst für eine oder mehrere andere Gruppen ist, dann ist dies mit "für" angegeben.

Strukturbaum:



Tätigkeit \ Gruppe (Stufe)	1	2	3	4
Entwurf	X			
Implementierung	X			
Adaption	X	X		
Verifikation	X	X		
Pflege	für 2	X		
Ausbildung	für 2	für 3,4	für 4	
Beratung	für 2	für 3,4	für 4	
Werbung	für 2,3,4	für 3,4	für 4	
Nutzung			X	X

Struktur und Hierarchie innerhalb der mit Anwendungssoftware Befaßten

Die ständig steigende Zahl der Benutzer von Anwendungssoftware sowie die große Menge offenkundiger Mängel und Fehler in den Tätigkeiten auf den höheren Stufen, welche sich als zunehmende Problematik beim dienstgebenden Rechenzentrum bemerkbar machen, wirkten als besondere Motivation auf die Verfasser zur Erstellung des vorliegenden Berichtes.

1. Forderungen an Anwendungssoftware

1.1 Anpassung an das Rechnersystem

Um größtmögliche Portabilität zu erreichen, wird dringend empfohlen, als Implementierungssprache eine international genormte Programmiersprache zu benutzen. Als Alternative kann allenfalls eine abstrakte Maschinensprache, welche "bootsfähig" sein sollte, hingenommen werden. Der Einsatz anlagenspezifischer Spracherweiterungen (wie Stringhandling, gewisse Datenbankzugriffe etc.) sollte unterbleiben. Besonders schlechte Erfahrungen im Bereich sprachspezifischer Ein- und Ausgabe lassen nur den Einsatz elementarer problemlosübertragbarer Routinen empfehlbar erscheinen. Ebenso ist es selbstverständlich, daß als Abbildung auf Datenträger nur international genormte Codes und Strukturen annehmbar sind.

Anlagenabhängige Teile eines Anwendungssystems müssen in getrennten Bausteinen - mit standardisierten Schnittstellen gegenüber dem restlichen Anwendungssystem - realisiert werden. Die oft fehlenden eindeutigen Definitionen der Leistung von Assemblerbausteinen (Bitmanipulation etc.) sind schwere Belastungen bei der Anpassung. Die Dokumentation muß, als Anpassungshilfen, die Beschreibung der Schnittstellenstandards und Hinweise, wie anlagenabhängige Bausteine etwa implementierbar sind, enthalten.

Der Betriebsmittelbedarf (Hauptspeicher, Hintergrundspeicher, Peripheriegeräte) sollte durch den Anwender leicht über Parameter zu verändern sein. Es sind Möglichkeiten zu einer einfachen Segmentierung vorzusehen. Die Modularisierung eines Anwendungssystems darf nicht nur von der Datenverarbeitungsseite bestimmt sein. Der Benutzer muß vielmehr aus seiner Sicht nach Programmleistung modularisieren, d.h. bausteinweise zusammensetzen können. Nur so wird das Mitschleppen nicht benötigter Programmteile vermeidbar.

Als wichtigste Anpassungshilfe wird die Quellverfügbarkeit angesehen. Der Anbieter muß durch entsprechende Vertraulichkeitszusagen in die Lage versetzt werden, dem Anwender neben der Systemdetaildokumentation auch die gesamten Programmquellen in wiedereinlesbarer Form zu überlassen. Ein Testsatz von Daten zur Prüfung der Richtigkeit und Effizienz der Implementierung auf dem speziellen Rechnersystem ist mitzuliefern. Zur Oberprüfung eines Anpassungserfolges sollte ein Testrahmen, der alle wesentliche System-

funktionen einschlägig anspricht, mit zum Lieferumfang gehören. Nicht fehlen darf ein ausgefeiltes Fehlerbehebungsverfahren, Ausbesserungsarbeiten an Programmquellen, Dokumentation und Testrahmen dürfen nur unter Programmüberwachung durchführbar sein, sodaß jede Änderung am Anwendungssystem nachvollziehbar ist (Geschichtsdokumentation).

1.2 Benutzung

Der Wert und damit die Anschaffungswürdigkeit eines Anwendungssystems werden in steigendem Maße an der Benutzung gemessen, Entsprechend drohen alle Aufwendungen für die Anpassung bei ausbleibender Benutzung wertlos zu sein. Eine gut ausgearbeitete Benutzerdokumentation ist Grundvoraussetzung für breite Benutzbarkeit. Nicht nur auf Systemebene sondern auch auf Benutzerebene müssen standardisierte, gut beschriebene Schnittstellen vorhanden sein.

Die Erfahrung hat gezeigt, daß im Gegensatz zu den Vorstellungen vieler Anbieter von Anwendungssoftware, die meisten Benutzer Eingabedaten, Zwischenergebnisse usw. mit schon vorhandenen "Programm-Umgebungen" in Wechselwirkung bringen müssen. Entsprechend ungern werden "verschlossene" Systeme von Benutzern angenommen.

Auch auf Benutzerebene muß ein Testrahmen zur Verfügung stehen. Dieser muß einerseits die Funktion gestufter Einführungsbeispiele erfüllen, andererseits den erfahreneren Benutzer anhand des Testrahmens in die Lage versetzen, etwa nach Systemmodifikationen die Funktionsverifikation selbst durchzuführen.

Die Dokumentation für Benutzer muß ganz wesentlich auf die sich immer mehr herausbildende Stufung Vorbereiter - Benutzer auf der Dienstnehmerseite Rücksicht nehmen. Entsprechend müssen die Schnittstellen die Zufügbarkeit von Benutzerbausteinen und die Zufügbarkeit von Benutzer-Dokumentationsteilen unterschiedlich gestuft beschrieben werden.

Die Anwendungssoftware muß wirtschaftlich d.h. mit geringem Rechenzeit- und Speicherbedarf arbeiten. Z.B. kann Modularisierung durch phasenweise sequentielle Verarbeitung mit Zwischenspeicherung der Daten bei großen Datenmengen zu erheblichen Laufzeitproblemen führen, was u.U. unwirtschaftlich ist. Die Forderung nach kostengünstigerer (d.h. wirtschaftlich vertrebarer) Einsetzbarkeit wird ggf. zum Verzicht auf Konstruktionsphilosophien wie "abstrakte Maschine" oder " Virtualisierung" führen (trotz der hohen damit erreichbaren Portabilität). Unter diesen Gesichtspunkten sollten solche, z.Zt. verbreitete Techniken im Einzelfall überprüft werden um Ineffizienz zu vermeiden.

Die Anwendungssoftware muß eine problemorientierte und dem Benutzer verständliche fachspezifische, formatfreie Eingabesprache anbieten. Änderungen (auch sogenannte Weiterentwicklungen) dürfen nur auf-

wärtskompatibel durchgeführt werden. Derartige Außenschnittstellen sollten an schon verbreitete Außenschnittstellen anderer Anwendungssysteme angepaßt sein.

1.3 Wartung

Vom Anbieter eines Anwendungssystem wird erwartet, daß er Fehlerbehebung und Verbesserungen vertraglich zusagt. Er muß die Maintenierung und Aktualisierung über einen definierten Zeitraum (z.B. n Jahre) garantieren. Diese Garantie alleine reicht jedoch erfahrungsgemäß nicht aus. Ohne Nachweis eines funktionstüchtigen Fehlerbehebungs-Verfahrens, sowohl für Programm- als auch für Dokumentationsteile, kann eine verwantwortungsbewußte dienstgebende Stelle kein Anwendungssystem zur Benutzung freigeben.

Folgende Voraussetzungen müssen vom Anbieter erfüllt werden, um Maintenierbarkeit zu gewährleisten:

(1) Verfügbarkeit in maschinenlesbarer Form sowohl der Programmquellen (zur Neugenerierung) als auch der Dokumentation;

(2) Ausbesserungsmechanismus, der Änderungen nur per Programm und unter Anwendung einer Maintenance-Versions- Generationsnummern-Technik durchgeführt ("Updatebarkeit");

(3) Geschichtsdokumentation (= Dokumentation der Änderungsgeschichte) automatisch per Programm;

(4) Programm zum Versionsvergleich;

(5) Standardtestrahmen zum Funktionstest.

Ein Anwendungssystem muß so konstruiert sein, daß Fehlerverfolgung und -behebung leicht möglich sind. Die Quellen müssen vom Anbieter bei allen Dienstgebern auf einheitlichem, neuestem und fortgeschriebenen Stand gehalten werden und dies muß durch einen ggf. neutralen sachkundigen Dritten mit vernünftigem Aufwand nachprüfbar sein. Dies beinhaltet auch eine Forderung nach sauberer, klarer und in vernünftiger Zeit durchschaubarer Programmierung und Dokumentation.

1.4 Dokumentation

Neben einer ordnungsgemäßen Programmdokumentation müssen eine Anwendungsbeschreibung und eine Leistungsbeschreibung für den Benutzer geliefert werden. Außerdem sollte eine Implementierungs-Beschrei-

bung für den installierenden Dienstgeber mitgeliefert werden. Spätestens bei Auslieferung einer neuen Programmversion muß die dazugehörige Dokumentationsänderung mit ausgeliefert werden. Die gesamte Dokumentation soll, in dem für die Quellversion gültigen Stand, mit auf dem Datenträger der Quelle sein ("Inline-Dokumentation"). Die Beschreibungen sollen ausführlich angeben, worin sich die vorliegende Version von der letzten, vorletzten, ... unterscheidet. Die Dokumentation soll in Anlehnung in die in Abschnitt 0. besprochene Hierarchie in verschiedenen (gestuften) Versionen vorliegen, etwa

- für den Allgemeinbenutzer;
- für dialogorientierte Benutzer;
- für (rechenzentrumsseitige) Betreuer aller Benutzer;
- für Unterrichtende, die Benutzer und/oder Betreuer unterrichten;
- für Implementierungsprogrammierer;
- für Programmierer von Ergänzen/Erweiterungen;

usw..

Die benötigten Schulungsunterlagen sollten Teile enthalten, die den Selbstunterricht unterstützen (z.B. Texte, Bilder, programmierter Unterricht etc.). Eine interaktive Bedienungsanleigung, die vom Terminal aus angesprochen werden kann, ist nötig.

Entsprechend ungünstige Erfahrungen mit vorhandenen Anwendungssystemen führen zu der Forderung nach preiswerter Dokumentation (Überlassung von Copyright, Mikrokopien etc.) und Schulungsunterlagen (z.B. Skripten, Overhead-Folien, Video-Bänder, ausgearbeitete Anwendungsbeispiele).

2. Anforderungen an Schnittstellen zwischen Rechnersystem und Anwendungssoftware

Vorbemerkung

Die Einbettung von Anwendersystemen verschiedenster Art in das (aus Hardware und Systemsoftware bestehende) Rechnersystem muß auf bequeme Art und Weise möglich sein; Anwendersysteme müssen wirtschaftlich sein im Hinblick auf Anpassung, Implementierung, Nutzung und Wartung; Anwendersysteme müssen dem Endbenutzer einen Zugang zur Rechnerbenutzung ermöglichen, der ohne besondere Kenntnisse der Systemsoftware erfolgen kann.

Dazu ist eine breitgefächerte Menge standardisierter Schnittstellen zur Systemsoftware bzw. zur Hardware nötig. Die Forderungen bezüglich solcher Schnittstellen werden im vorliegenden Abschnitt 2 besprochen; einige zusätzliche Bemerkungen finden sich in anderen Abschnitten, nämlich wenn aus der Sicht des betreffenden Abschnitts gewisse Schnittstellenprobleme tangiert werden.

Bevor auf die Schnittstellen eingegangen wird, seien noch folgende Bemerkungen gemacht:

(1) Bei den Betrachtungen über Schnittstellen wird klar, daß Kommunikation und Interaktion zwischen Moduln und Bausteinen heutiger Anwendersysteme immer stärkere Bedeutung gewinnen und deshalb diesbezügliche Möglichkeiten in der Systemsoftware und Hardware vom Architekten des Rechensystems beim Entwurf voll eingeplant werden müssen.

(2) Dem Betreiber eines Anwendungssystems müssen ausreichende Hilfsmittel zur Organisation des Betriebs seines Anwendungssystems zur Verfügung gestellt werden. Insbesondere muß es möglich sein, den Organisationsaufwand differenziert auf verschiedene Stufen der Anwenderhierarchie (siehe Abschnitt 0.) zu verteilen. Dazu soll es u.a. verschiedene Ebenen der Protokollierung von Ablaufmeldungen und Rechen-Ergebnissen geben, die der Einsicht und dem Verständnis der auf den jeweiligen Hierarchiestufen befindlichen Benutzer entsprechen (z.B. Fehlermeldungen für Systemprogrammierer, Mitteilungen und Hinweise für den Implementierer, Hinweise für den reinen Benutzer; Protokollierung des Ablaufs für den Systemprogrammierer, einfachere Protokollform für den Benutzer, u.ä.). Dieser Sachverhalt ist mit Schnittstellenproblemen verwandt und sei deshalb hier erwähnt.

2.1 Ein/Ausgabe

Anwendungssoftware sollte weitgehend unabhängig vom jeweils benutzten Rechnersystem implementiert werden können. Hieraus ergeben sich Forderungen bezüglich der Ein/Ausgabe:

2.1.1 Einheitlichkeit der Zugriffe zu Daten von Programmen aus

Sämtliche verfügbaren Datenarten, Dateitypen und Ähnliches, sollen von allen Programmiersprachen aus verwendbar sein und auch einheitlich gehandhabt werden können. Im Hinblick auf Interaktion und Kommunikation zwischen Programmbausteinen wäre es wohl die eleganteste Lösung, einen zentralen Modul oder Satz von Teilmoduln einzurichten, der diese Leistungen einheitlich für alle Compiler bzw. Sprachen erbringt. Auch die Anschlüsse zu Datenbanksystemen sollten hierin mit einbezogen sein.

Im Hinblick auf effiziente Produktionsläufe soll die E/A eine besonders schnelle Version haben, bei der gewisse Funktionen und/oder Kontrollmechanismen eingeschränkt und/oder abgeschaltet sein dürfen. Bezüglich des Aufbaus aller Dateien, Datenstrukturen und dergleichen sollen innerhalb eines Systems einheitliche Normen gelten. Damit wäre das wichtige Ziel zu erreichen, daß der Output jedes Programmbausteins als Input für jeden anderen verwendet werden kann, ohne daß dem Inkompatibilitäten entgegen stehen (siehe dazu u.a. Abschnitt 2.2).

2.1.2 Verkehr der Anwendungsprogramme mit E/A-Geräten oder im Verbund angeschlossenen Rechnersystemen

Es soll möglich sein, daß von der Anwendungssoftware aus Aufträge, Eingabe, Ausgabe oder sonstige Information an beliebige Geräte gesandt oder aber von dort empfangen werden. Dabei wird hier unter "Gerät" das ganze Spektrum der an einem Groß-Rechnersystem angeschlossen Ein- oder Ausgabemedien verstanden. Darin sind eingeschlossen: Fremdrechner, Prozessrechner, Konsolen, intelligente Terminals, aber auch E/A-Geräte wie Drucker, Stanzer, Leser, Plotter u.a.. Dies sollte so geschehen, daß nur ganz wenige logische Schnittstellen ("virtuelle Geräte") ausreichen und somit vom Anwenderprogramm aus nicht die vielen Eigenarten der angeschlossenen Geräte berücksichtigt werden müssen. Es würde damit für ganze Klassen von Geräten eine einheit-

liche Handhabung möglich (z.B. Konzept des virtuellen Terminals).

Auch soll der Dialog bezüglich aller Dialoggeräte einheitlich und standardisiert sein. Bei graphischen Dialoggeräten soll zudem eine geräteunabhängige Graphikschnittstelle existieren.

Das Ansprechen von Geräten (Geräte im oben beschriebenen Sinn) muß gezielt möglich sein, d.h. sie müssen "adressiert" werden können. Dazu ist eine eindeutige Adreßzuordnung zu allen an den Rechner angeschlossenen bzw. im Rechnerverbund vorhandenen Geräten nötig (z.B. nach dem am TR440 verwendeten Geräte/Stadions-Prinzip). Dieses Konzept muß beinhalten, daß im Falle solcher Terminals, an die ihrerseits weitere EA-Geräte angeschlossen sind, die Möglichkeit besteht, diese sekundären EA-Geräte auch einzeln und gezielt anzusprechen (z.B. "Terminal-Cluster", "Master-Slave-Konfiguration").

Im Falle von Rechnerverbundsystemen ist eine weitere Einrichtung im System nötig: Sie soll Transportwege vom Anwendungssystem zum Betriebssystem eines anderen Rechners oder zu einem dort befindlichen Anwendungssystem herstellen (Routing-Funktion). Die dazu im Rechnerverbundsystem zur Verfügung gestellten Schnittstellen sollten bei allen Rechnern einheitlich sein. Damit wird das Durchschleusen von Informationen allgemeinster Art über mehrere Knoten des Netzes hinweg möglich.

Beispiele:

- Kommunikation eines Anwendungssystems mit einem Objekt in einem anderen Rechner des Verbunds etwa im Sinne mehrerer Teildatenbanken;
- Kreation eines Auftrags für einen anderen Rechner durch ein Anwendungssystem im eigenen Rechner.

2.2 Verknüpfung von Programmbausteinen

Ein oft in der Anwendungssoftware auftretendes Problem ist, mehrere kleinere Programmbausteine zu einem größeren System zu verknüpfen. Hierbei ist mit Programmbaustein entweder ein Unterprogramm, eine Prozedur einer höheren Programmiersprache, ein Hauptprogramm oder auch eine Menge von bereits verknüpften Hauptprogrammen gemeint. Die heute hierfür zur Verknüpfung stehenden Mechanismen bewegen sich einerseits auf der Ebene der Programmiersprachen und andererseits auf der Ebene der Job-Control-Languages (kurz JCL).

2.2.1 Verknüpfung auf JCL-Ebene

Üblicherweise besteht ein Job nicht nur aus einem einzigen Lauf eines Programms, sondern aus mehreren oder ganzen Ketten von Programmläufen. Damit auch komplexere Jobs bequem und sicher zu erzeugen und gut zu dokumentieren sind, muß die JCL eine logisch durchsichtige, komfortable Programmiersprache sein, die syntaktisch engen Regeln folgt und über gute Testhilfe-Möglichkeiten und u.a. einen Syntaxchecker verfügt.

Zur Formulierung des Auftragsablaufes werden in der JCL die allgemein anerkannten Bausteine der strukturierten Programmierung benötigt, nämlich WHILE-DO, REPEAT-UNTIL, IF-THEN-ELSE-FI, CASE-IN-OUT-ESAC und parametisierbare Prozeduren. Bezüglich Datenstrukturen sind nötig: Dateien (Files), Programme und Programmbausteine (einschließlich JCL-Code), Datenträger (Volumes), E/A-Medien (z.B. Karten, Papier für Druck, Papier für Plott, Markierungsbelege, Bildschirme, Tastatur), Texte (Strings) sowie aus solchen Grundstrukturen zusammengesetzte Strukturen (z.B. lineare Listen, powerset-Typen im Sinne von PASCAL, Strukturen im Sinne von ALGOL 68). Variable sollen für alle Arten von Objekten vorhanden sein. Ferner braucht die JCL Sprachmittel zur Synchronisation einzelner Teile eines Jobs innerhalb des Gesamtauftrags sowie zur Kommunikation von Aufträgen oder Teilaufträgen untereinander oder mit dem Betriebssystem (siehe auch Abschnitt 2.4). Einsatzgebiet solcher Sprachmittel ist die zeitlich parallele Steuerung von Teilen eines Jobs etwa zur Durchsatzbeschleunigung in Mehrprozessoranlagen, zur Vermeidung von Zugriffskonflikten bei Zugriff auf gemeinsame Datenbestände oder die Koordination von Teilen eines Jobs oder eines ganzen Jobs aufgrund bestimmter Ereignisse.

Weiter ist zu fordern, daß die im Abschnitt 2.1.1 beschriebene Standardisierung der Dateiformate und Datenstrukturen gewährleistet ist.

Damit ist eine wichtige Voraussetzung zur Verwendung sogenannter Methodenbanken geschaffen. Für Methodenbanken wird darüberhinaus in der JCL ein Steuerungsmechanismus zur Entnahme von Standardprogrammbausteinen aus der Bank benötigt, der es auch erlaubt in bequemer Weise die Jobs zusammenzubauen und zum Ablauf zu bringen (vgl. auch Abschnitt 3.3.2).

2.2.2 Verknüpfung auf der Ebene der Programmiersprachen

Bei der heutigen Rechneranwendung wird üblicherweise nicht nur eine Programmiersprache, sondern es werden (entsprechend der Intention der höheren Programmiersprachen, eine bequeme problemorientierte Formulierung zu ermöglichen) mehrere Sprachen verwendet.

Es ist nicht nur wünschenswert, innerhalb eines Jobs mehrere Sprachen verwenden zu können, sondern auch in einem in einer bestimmten Sprache geschriebenen Hauptprogramm Unterprogramme verschiedener auch anderer Sprachen verwenden zu können. Voraussetzung für die Verwendbarkeit von Unterprogrammen aus verschiedenen Programmiersprachen sind:

- Einheitlichkeit der Linkmechanismen für Compilate aus den verschiedenen Sprachen;
- einheitliche interne Anordnung von mehrdimensionalen Feldern und Strukturen;
- eine bezüglich aller Sprachen kompatible E/A (siehe auch Abschnitt 2.1.1);
- einheitliche Techniken für Unterprogrammaufruf, Rücksprung, Parameterübergabe (oder zumindest Stadard-Schnittstellen-Routinen zur Übergabe von Parametern zwischen Moduln, die in verschiedenen Quellsprachen geschrieben sind).

Neben der Schaffung dieser Voraussetzungen ist es nötig, daß die Leistungen der auf symbolischer Ebene verfügbaren Testhilfen für alle Standard-Programmiersprachen in vollem Umfang und mit einheitlicher Benutzungsform verfügbar sind (Beispiele: diverse statische Testhilfen, Rückverfolgung, Variablendump auf symbolischer Ebene, Trace, Backtrace, Kontrollpunkte usw. vgl. Abschnitt 3.2).

Bei heutigen Rechnersystemen ist ein Hauptprogramm mit größeren Mengen von Unterprogrammen nicht ohne weiteres lauffähig. Gründe sind u.a. die Beschränktheit des Adressraums und die Beschränktheit der physikalisch zur Verfügung stehenden Speicher. Um dennoch große Jobs lauffähig zu machen, werden heute so umständliche Techniken wie Overlay, Segmentierung und ähnliches verwendet. Für zukünftige Rechnersysteme ist es jedoch dringend erforderlich, daß Anwendungssysteme ohne Rücksichtnahme auf Größenbeschränkungen der Programme erstellt werden können und die den Segmentierungs- oder Overlay-Techniken entsprechende Leistungen des Programmierers automatisch

von der Systemsoftware erbracht werden. Insbesondere bei großen und sehr großen Programmsystemen würde dies die Unabhängigkeit des Anwendungssystems vom jeweils verwendeten Rechnersystem (d.h. von dessen Hardware bzw. Systemsoftware) verwirklichen.

Schließlich ist noch wünschenswert, daß die Leistungen, die JCL-Statements erbringen, auch von den Programmiersprachen aus verfügbar sind (solche Leistungen sind üblicherweise nicht in Form von Statements der Programmiersprachen vorhanden). Eine Lösung wäre, daß die Leistung von JCL-Statements in Form von Prozeduraufrufen angefordert werden kann. Dies sollte von allen Sprachen aus möglich sein. Beispiele für wichtige Funktionen, die hiermit realisiert werden könnten, sind: Kreation einer Datei vom Programm aus; Herstellung der Zuordnung dieser Datei zur logischen Datei, die vom Programm aus gesehen wird; Bearbeitung dieser Datei; Aufheben der Zuordnung zwischen Datei und logischer Datei; u.s.w.. Dabei sollte der Versuch einer Normung für die Prozeduren gemacht werden um sicherzustellen, daß der Austausch von Software zwischen unterschiedlichen Rechnersystemen möglich ist.

2.3 Datenhaltung

Die bisher gemachten Erfahrungen mit Anwendungssoftware und der Wunsch nach einer Reihe bestimmter Eigenschaften, die man von heutiger Anwendersoftware erwartet, ziehen gewisse Forderungen an die Datenhaltung nach sich. Darauf wird nachfolgend eingegangen.

2.3.1 Programmbibliotheken

Eine Programmbibliothek soll nicht nur Quelltexte von Programmen, sondern auch deren Compilate, lauffähige Programme, JCL-Programme und komplette Jobs gemeinsam speichern, verwalten und dokumentieren. Dies bedeutet, daß in einer Programmbibliothek auch Bezugslisten zwischen Maschinencode und symbolischem Programmtext (für Testhilfesysteme), Dokumentationen von Benutzern über ihre Programme, Informationen wie Statistik über Benutzung von Software (d.h. Informationen zur Kostenabrechnung der Benutzung von Bibliothekssoftware etc.) in der Bibliothek abgelegt werden müssen (siehe auch Abschnitt 3.1, in dem zu Problemen der Quellverwaltung und Programmdokumentation Stellung genommen wird).

2.3.2 Verwaltung externer Datenträger

Das Betriebssystem muß in der Lage sein, die Verwaltung beliebiger Untermengen der Gesamtheit aller Datenträger an ein Anwendungssystem oder mehrere (u.U. kommunizierende) Anwendungssysteme zu delegieren. Das heißt, daß das Anwendersystem (bzw. die Anwendersysteme) Prüfungen von Zugriffsberechtigungen, Abrechnungen der Benutzung, die Vergabe von Speicherplatz an Benutzer u.ä. selbst durchführt und zwar in enger Zusammenarbeit mit der Datenträgerverwaltung des Rechnersystems (und des Rechenzentrums). Es muß möglich sein, daß die Datenträgerverwaltung des Anwendungssystems die Kontrolle bzw. den Schutz vor unerwünschter Benutzung der (ihm unterstellten) Datenträger voll selbst ausführt. Damit wären z.B. folgende sehr oft benötigte Leistungen realisierbar:

- Für jeden externen Datenträger ist ein individueller Benutzerkreis festsetzbar;
- eine Abstufung der Benutzerberechtigung (Lesen, Schreiben etc.) ist möglich;
- Informationen über den Gebrauch der externen Datenträger durch die Benutzer des Anwendungssystems (wann, wer, wo,was) ist für Abrechnungs- oder Fehlerverfolgungszwecke möglich;

  usw..

2.3.3 Einheitlichkeit der Datenhaltung

Anwendungssysteme würden künftig davon profitieren, wenn die Rechner eine einheitliche Datenhaltung mit in sich konsistenter Nomenklatur hätten. So ist z.B. das Vorhandensein von so verschiedenen Formen wie "Gebiet", "Datei", "Phys-Datei", "Bibliothek" im Betriebssystem des TR440 kein gutes Beispiel. Besser wäre es von der Datei als alleinigem "Universal-Daten-Behälter" wegzukommen und auch allgemeinere Datenstrukturen (z.B. Text, Makro, Programm, JCL-Code, zu bindendes Compilat, ...) zuzulassen.

Schließlich wäre noch wünschenswert, daß die einheitliche Nomenklatur auch für alle Hersteller, für alle Anlagentypen usw. gilt. Innerhalb dieses Einheitssystems sollten u.a. alle heute geläufigen (allerdings mehr zufällig entstandenen) Dateitypen verfügbar sein.

2.3.4 Datensicherung

Die Möglichkeiten der Datensicherung bei Datenhaltung und Verarbeitung der Daten, welche vom Betriebssystem angeboten werden (z.B. Logbuch, Journalschreibung, doppelte Dateilhaltung, Veränderungskonserve, Stützpunktschreibung, Parallel-Schreiben auf unterschiedliche Träger, Überlesen fehlerhafter Information, gestückeltes Lesen und Schreiben, ...) müssen vom Anwendungssystem gezielt in eigener Verantwortung genutzt werden können, damit der Aufwand für Datensicherung dosierbar wird. Dabei ist besonderer Wert darauf zu legen, daß das Rechnersystem es (in Form geeigneter Hilfen und Funktionen) möglichst weitgehend erlaubt, fehlerhafte Informationen zu rekonstruieren, Dateien mit Fehlern möglichst weitgehend zu retten und auf beschädigten Datenträgern wenigstens diejenigen Informationsbereiche noch zu lesen, die nicht gestört sind.

2.3.5 Datenschutz

Im Gegensatz zu vielen derzeitigen Betriebssystemen, die die Datenschutzaspekte noch kaum berücksichtigen, müssen zukünftige Betriebssysteme Hilfen bereitstellen, die es erlauben, die Datenschutzvorschriften wirksam zu realisieren. Die gesamte Menge dieser Hilfen muß in bequemer Weise von Anwendungssystem verwendet werden können.
Beispiele solcher Hilfen sind:

(1) Zugriffsschutz, z.B.:

- Schreib/Lese-Sperre
- nur Leseerlaubnis
- nur Schreiberlaubnis
- nur Hinzufügen neuer Information (ohne Überschreiben)
- nur Exekution (dabei wird vorausgesetzt, daß es sich um im Sinne eines Programmes ausführbaren Code handelt; es soll darüberhinaus weder Lesen noch Schreiben möglich sein).
- Zugriffsschutz auf unterschiedlichen logischen Ebenen der Dateistruktur (ganze Datei, Block, Satz, Teilsatz, ...) .

(2) Physikalisches Löschen des Inhalts von Speichern, wenn dieser nicht mehr gebracht wird, z.B. wenn eine Datei freigegeben wird.

(3) Protokollierung aller Zugriffe (wirklich erfolgte und auch solche, die nur versucht wurden, aber wegen Unzulässigkeit abgewehrt wurden).

(4) Verstärkter Schutz des Inhalts externer Datenträger dadurch, daß nicht nur Paßwortschutz vorhanden ist, sondern daß die Information kryptographisch verschlüsselt abgelegt wird.

(5) Die vom System bereitgestellten Schutzmöglichkeiten müssen mit unterschiedlicher hierarchischer Stufung durch das Anwendungssystem bzw. den Benutzer verwendet werden können, z.B.:

- Der Normalbenutzer soll einen als ("executable" erklärten Dateiinhalt weder lesen noch verändern können, jedoch im Sinne eines Programmes starten können;
- das Rechenzentrum soll möglicherweise eben diesen Dateiinhalt lesen können;
- der Systemberater soll u.U. der Einzige sein, der auch den Inahlt verändern darf.

Im Zusammenhang mit der hierarchischen Stufung soll es auch möglich sein, daß Information von Benutzern grundsätzlich gegen den Zugriff durch das Rechenzentrum geschützt werden kann.

2.4 Kommunikations- und Steuerungsmechanismen

An verschiedenen Stellen des Abschnitts 2 wurde bereits auf die wachsende Bedeutung der Steuerungs- und Kommunikations-Mechanismen zwischen Aufträgen und Programmbausteinen hingewiesen. Es soll nun auf die erforderlichen Leistungen genauer eingegangen werden. Selbstverständlich müssen diese Funktionen von Anwendungssystemen aus voll verfügbar sein.

2.4.1 Synchronisation und Parallellauf

Es ist ein Kennzeichen heutiger Rechensysteme, daß mehrere Prozesse quasi-parallel zueinander ablaufen können, in gewissen Fällen sogar recht parallel zueinander ablaufen. Damit sind Mechanismen zur Synchronisation solcher parallel ablaufender Prozesse nötig. Anwendungssysteme machen heute mehr und mehr Gebrauch von den Möglichkeiten des Parallellaufs, und es ist damit nötig, daß vom Anwendungssystem aus voller Gebrauch von diesem Steuerungsmechanismen gemacht werden kann. Solche Mechanismen sind z.B. nötig für:

- Koordination von Dateizugriffen;
- es soll nicht nur eine Zugriffskoordination für die ganze Datei, sondern wahlweise auch für einzelne Datensätze gefordert werden können;
- Koordination des Ablaufs von Aufträgen und Teilaufträgen;
- Koordination von nicht nur logisch parallel, sondern echt zeitlich parallel ablaufenden Aufträgen, etwa in Mehrprozessoranlagen. (Dies ist z.B. durch Verwendung gemeinsamer Datenstrukturen mit entsprechenden Koordinatiosmechanismen erreichbar.)

2.4.2 Zeit- und ereignisabhängige Steuerung

Bezüglich der Zeitabhängigkeit zu fordernde Leistungen sind:

- Wecken und Passivieren von Auftragsteilen oder ganzen Aufträgen;
- zeitlich gesteuertes Initiieren von Benutzereingaben;
- zeitlich gesteuerter Start von Teilhabersystemen oder gewisser Aufgaben solcher Systeme.

Weiter soll Steuerung nicht in Abhängigkeit von der Uhrzeit, sondern von gewissen Ereignissen (wie z.B. "eine Stunde keine Daten eingelaufen", "Zahl der eingelaufenen Daten hat eine Sollmenge erreicht", u.ä.) möglich sein.

2.4.3 Wechselwirkung zwischen Aufträgen

Dem Anwendersystem müssen Mechanismen verfügbar sein, die es erlauben, direkt zwischen Aufträgen Informationen auszutauschen (nicht auf dem Umweg über Dateien). Weiterhin muß es möglich sein, daß ein Auftrag einen anderen kreiert.

3. Benötigte Software-Erstellungshilfen

Der Trend, daß die Entwicklung von Software immer größeren Kapitaleinsatz erfordert wird sich fortsetzen. Es ist daher notwendig, daß ein künftiges Groß-Rechnersystem weitestgehende Unterstützung bei der Entwicklung von Software bietet. Diese Unterstützung muß also erheblich über den Rahmen dessen hinausgehen, was heute üblicherweise zur Verfügung steht, nämlich nur Compiler der verbreiteten Programmiersprachen.

Betrachtet man den Prozess der Programmentwicklung, so kommt man zu den im Folgenden aufgeführten Forderungen.

3.1 Quellenverwaltung und Programmdokumentation

Es sind Hilfsmittel zur Verwaltung, Generierung und beliebigem Rekonstruktion von Programmversionen erforderlich. Jede Programmversion muß bei der Generierung eine eindeutige Versionsnummer erhalten und über diese Versionsnummer jederzeit identifizierbar sein. Beim Erzeugen des Programms muß jeweils sichergesetllt sein, daß Änderungen gegenüber einer Basisversion sofort erkennbar sind. Es muß möglich sein, gekennzeichnete Erweiterungen und Korrekturen des Programms gezielt einzubeziehen oder auch wegzulassen. Die einzelnen modifizierten Versionen eines Programmes müssen klar erkennbar sein. Zur Unterstützung der Programmdokumentation sollten Hilfsmittel zur Textaufbereitung und Texthaltung verfügbar sein. Beim Ändern eines Programmes muß automatisch die Dokumentation geändert werden. Aus diesem Grund sind "Softwareprozessoren" bereitzustellen, die eine Dokumentation eines Programmes in Form von Skripten, Büchern u.ä. automatisch aus der im Quellcode enthaltenen internen Dokumentation herausziehen. Dafür sind geeignete Textverwaltungs- und Textgenerierungs-Systeme nötig. Der Programmierer sollte gezwungen sein, ein Minimum an interner Dokumentation mitzuliefern. So könnte z.B. bei einer Unterprogrammvereinbarung zwingend vorgeschrieben sein, die Ein- und Ausgabevariablen des Programms zu beschreiben und den Zweck des Programmteiles zu erläutern. Im Hinblick auf automatische Verifikation von Programmen ist vorzusehen, daß die Semantik des Programms in einer Dokumentationssprache zu beschreiben ist, so die Übereinstimmung mit dem aktuellen Code überprüfbar wird und daß die zur automatischen Verifikation nötigen Zusätze (z.B. "Schleifen-Invariante") auf genormte Art in den Quelltext eingeführt werden.

3.2 Testhilfen und Protokolldienste

Gute Testhilfen sind an fast keinem heutigen Rechnersystem zu finden, und es ist eigenartig, daß sich die meisten Computerbenutzer sehr leicht damit zufrieden geben, daß Fehlersuche beim Programmieren eine "schwierige und trickreiche" Angelegenheit ist. Schließlich sollte doch die Maschine die Arbeit erleichtern und nicht zur Arbeitsbeschaffung für Programmierer dienen. Testhilfen, wie nachfolgend beschrieben, sollten heute bei allen Rechnern selbstverständlich sein, insbesondere bei Groß-Rechnersystemen.

Die nachfolgende Beschreibung der Testhilfen stellt kein (vollständiges) Pflichtenheft dar, sondern soll nur als Beispielkatalog aufgefaßt werden.

3.2.1 Allgemeines

(1) Die Leistungen der Testhilfen sollten ausschließlich mit quellbezogenen Variablennamen und -Werten erbracht werden und im Klartext (keinerlei Binär- oder Hexadezimalzahlen oder interne Adressen o.ä.) erfolgen.

(2) Die Leistungen sollten völlig einheitlich für Stapel- und Interaktivbetrieb (soweit logisch sinnvoll) und weiterhin einheitlich bezüglich aller Remote-Batch-Stationen, Terminals und der direkten Rechenzentrumsperipherie sein.

(3) Alle Testhilfen sollen nicht nur auf einzelne Zeilennummern des Quelltextes, sondern auch auf Namen (Variablen, Marken, Prozeduren, Dateinamen u.ä.)ansetzbar sein.

(4) Die Testhilfen müssen (soweit logisch sinnvoll und möglich) einheitlich für alle Sprachen sein und einheitliche Meldungen liefern.

(5) Für alle Sprachen soll ein Prompter (Editoperator) zum Eintrag von Quellzellen in eine Datei, mit sofortigem Syntaxtest (soweit möglich) existieren, der mit den Compilern völlig identischen Sprachumfang hat. Statt dessen könnte man auch verlangen, daß alle Compiler einen Prompter-Modus haben.

(6) Die Testhilfen sollen auf Wunsch zwecks schnellerer Laufzeit gestuft abgeschaltet werden können, wobei der zum Produktionscompiler völlig identische Sprachumfang in jeder Stufe gewährleistet sein muß.

(7) Referenzlisten, Adressbücher, Identifier-Listen, Attribut-Listen, Externbezug-Listen, Pseudo-Assemblat (bzw. direkter Objektcode) sollen auf Wunsch nach der Übersetzung ausgegeben werden, wobei einheitliche Bezeichnungen von allen Compilern zu verwenden sind.

(8) Es soll auf Wunsch sowohl beim Übersetzen, wie auch beim Programmlauf eine Programm-Profilerstellung erfolgen (z.B. mit Häufigkeitszählung für die Durchläufe durch die einzelnen Programmzeilen oder für Verweilzeiten in bestimmten Programmbereichen, in Unterprogrammen bzw. Prozeduren, u.ä.).

(9) Alle Protokolle (reports) sollten in "maschinell weiterverarbeitbarer" Form zur Verfügung gestellt werden.

(10) Alle Testhilfen sollen ohne Veränderung des Quelltextes aktivierbar sein; lediglich Neu-Übersetzung (und eventueller Link-Lauf) wird in Kauf genommen.

3.2.2 Beispiele wichtiger dynamischer Testhilfen

(1) Dynamische Kontrollen

Durch entsprechende Angaben im Compilier- und Executekommando soll während des Laufs entweder im ganzen Programm oder nur für eine oder mehrere Anweisungen kontrolliert werden, ob Indexvariable nicht Feldgrenzen überschreiten, ob sich Unterprogrammaufrufe bezüglich Anzahl und Art der aktuellen Parameter mit den formalen Parametern des Unterprogramms vertragen und ob Schleifenparameter zulässige Werte haben. Alle Fehlerausdrucke und sonstigen Angaben zu Verletzungen der jeweiligen Sprach-Semantik (z.B. inkompatible Modes, rechnen mit undefinierten Größen, Bereichsüberschreitungen, Verwendung anlagenspezifischer Spracherweiterungen) sollten ausschließlich auf Quellsprachebene erfolgen.

(2) Dump

Im "Alarmfall" (z.B. Division durch Null, Verarbeiten einer Variablen mit undefiniertem Wert) oder auf explizite Anweisung (siehe die später behandelten "Kontrollpunkte") sollen die momentanen Werte von Variablen auf Quellsprachenebene ausgegeben werden. Dabei soll die Hilfe eines sogenannten Rückverfolgers bereitstehen, welcher die aktuelle Aufrufverschachtelung von der Unterbrechungsstelle bis zurück ins Hauptprogramm in Quell-

sprachenform rekonstruiert. Es soll eine Reihe von Varianten des Dumps möglich sein: Dumpen von bestimmten oder allen Variablen von Hauptprogrammen und allen Unterprogrammen; Dumpen von bestimmten oder allen Variablen der an der aktuellen Aufrufverschachtelungen beteiligten Programme und Unterprogramme; Dumpen von bestimmten oder allen Variablen von ganz bestimmten Unterprogrammen und/oder dem Hauptprogramm (etwa: "nur V1, V7 in Subroutine GEX" oder "alle Variablen außer Feld ARRAY in Subroutine NAM"); Dumpen in verschiedenen Varianten, aber mit Setzen der Werte bestimmter Variablen auf angegebene Zahlen- (oder String-) Werte an der Unterbrechungsstelle (z.B. mit "I3=1I, V(7)=47.12").

(3) Trace (Ablaufprotokollierung)

Er ermöglicht eine Verfolgung des Programmlaufes auf Quellsprachebene. Dabei werden entweder einzelne Programmbereiche oder das ganze Programm im Ablauf protokolliert. Angegeben werden z.B.: GOTO-Anweisungen (Sprünge), Wertzuweisungen an Variablen oder Feldelemente, Unterprogrammaufrufe, IF-Anweisungen (Bedingungen), oder Kombinationen solcher Ereignisklassen.

(4) Backtrace

Er spart gegenüber dem Trace Rechenzeit und Druckseiten. Dies geschieht dadurch, daß wie beim Trace der Ablauf mitverfolgt wird, wobei aber jeweils die letzten N Meldungen (etwa N=50) gepuffert bleiben und nicht ausgegeben werden. Nur im Alarmfall oder auf explizite Anweisung des Benutzers wird der Pufferinhalt zum Druck aufbereitet und ausgegeben.

(5) Kontrollpunkte und Anhaltebedingungen

Durch beliebige Angaben im Compiliere-Kommando oder dynamisch nach Anhalten des Programmlaufs können beliebige Programmzeilen mit zusätzlichen Marken versehen werden, ohne daß der Quelltext verändert wird. Diese Marken ("Kontrollpunkte") sind der beiden Zustände "passiv" und "aktiv" fähig. Aktivsetzen und Passivsetzen kann im Execute-Kommando oder durch besondere Kommandos erfolgen. Wird im Lauf des Programms eine Quellzeile mit einem aktivgesetzten Kontrollpunkt erreicht, dann wird der Lauf unterbrochen und es besteht Gelegenheit, (per Kommando) in den Lauf einzugreifen, während alle passivgesetzten Kontrollpunkte ohne Programmunterbrechung übergangen werden.

Während so Anhaltepunkte an bestimmten Stellen im Programm festgelegt werden können ist eine andere Möglichkeit auf oft wünschenswert, nämlich, daß bei Erfüllung bestimmter Bedingungen angehalten wird (etwa "beim Siebten Durchlauf der Schleife in Quellzeile 17" oder "bei

jedem Aufruf der Prozedur P" oder "falls Variable F den Wertbereich -1.0 bis +1.0 durch Zuweisung oder READ verläßt"). Nach Anhalten aufgrund einer solchen "Anhaltebedingung" soll ebenfalls Gelegenheit bestehen (per Kommando) ins Programm einzugreifen.

Nach dem Eingriff soll das Programm auf Wunsch an der Unterbrechungsstelle (oder an sonstiger vorgebbarer Stelle) fortgesetzt werden. Schließlich ist es nützlich Kontrollpunkte und Anhaltebedingungen zu kombinieren, womit dann Anhaltebedingungen "passiviert" werden können, sodaß sie auf Wunsch kommentarlos übergangen werden.

Als Eingriffe kommen in Frage und sind bei Stapelbetrieb (in Form von Kommandos) vorzudefinieren:

a) Es soll (auf Quellebene) eine Variable, ein Feldelement, ein Feld gedumpt werden;

b) es soll die momentane Aufrufverschachtelung oder eine Teilangabe dazu ausgegeben werden;

c) alle Variablen eines Unterprogramms sollen gedumpt werden;

d) ein Backtrace mit N Schritten soll ausgegeben werden;

e) eine oder mehrere Trace-Versionen sollen eingeschaltet werden (oder "aus", falls vormals eingeschaltet);

f) gewisse aktive Kontrollpunkte sollen passivgesetzt werden und/oder gewisse passive Kontrollpunkte sollen aktivgesetzt werden wenn gewisse Bedingungen (die auf JCL-Ebene formulierbar sind) erfüllt sind. Beispiele: Wenn der Kontrollpunkt soundsoviele Male durchlaufen ist; wenn eine bestimmte Variable einen gewissen Wert überschritten hat;

g) sonstige Testhilfen sollen zu Hilfe genommen, aktiviert und passiviert werden;

h) logische Geräte- und Dateinummern sollen umbenannt werden, so daß z.B. mit einer Testdatei (statt Realdatendatei) weiter gerechnet wird.

Die Job-Control-Language muß ausreichende Sprachmittel haben, um solche Eingriffe einerseits für Stapelaufträge effizient und bequem formulieren zu können; andererseits müssen diese Leistungen im interaktiven Betrieb voll zugänglich sein.

3.2.3 Beispiele wichtiger statischer Testhilfen

(1) Aussagekräftige Protokollierung bei der Compilierung

Alle Fehlermeldungen (Syntaxfehler, ...) sollen Zeichengenau lokalisiert und in quellsprachebezogenem Klartext verfaßt sein (also z.B. nicht "Fehler 3761 in Zeile 110"). Fehlermeldungen und sonstige Bemerkungen zum Quelltext sollen in unmittelbarer Nähe der betreffenden Stelle im Quelltext angebracht werden und nicht etwa am Ende des Protokolls gesammelt sein. Die Texte sollen zutreffend und aussagekräftig sein (z.B. nicht "Lesepuffer zu klein", sondern "Quellzeile zu lang"; bei ALGOL-Programmen z.B. nicht den Begriff "Feld" verwenden, sondern "Array" wie in der ursprünglichen ALGOL-Beschreibung; bei FORTRAN-Programmen z.B. nicht den Begriff "String" verwenden, denn man spricht hier von "Hollerith-Konstante").

(2) Kennzeichnung besonderer Quelltextbereiche

Bereiche im Quelltext, deren Inhalt nicht den Syntaxregeln genügt oder nicht genügen braucht (wie z.B. Kommentare, Strings, Raum zwischen Syntaxfehlerstelle und Wiederaufsetzstelle des Compilers) sollen vom Compiler gekennzeichnet werden. Dabei genügt für Kommentare, Strings und ähnliches eine zeilenweise Kennzeichnung (etwa am linken Rand), während Syntaxfehlerstellen und Wiederaufsetzpunkte des Compilers zeichengenau angegeben sein müssen.

(3) Meldungsklassen

Die Meldungen des Compilers sollen in Klassen eingeteilt sein, z.B. "Fehler", "Warnung", "Hinweis" und der Benutzer soll gestuft die Ausgabe verlangen können, z.B.: "keine Meldungen"; "nur Fehlermeldungen"; "Fehler und Warnungen, aber keine Hinweise"; "alle Meldungen". Die Meldungsklassen sollen für alle Compiler einheitlich sein.

(4) Anzeige toter Stellen

Nicht initializierte oder tote Variable, toter Code (nie zu durchlaufende Statements etc.) sollen auf Wunsch vom Compiler gekennzeichnet werden.

(5) Hinweis auf vermutete Fehler

Programme oder Teile davon, die zwar syntaktisch zulässig aber semantisch unsinnig sind sollten vom Compiler mit Warnungen versehen werden (Beispiel: Vergessene Kommentarabschlüsse bei Sprachen, die formatfreie Quellformulierung haben, wie z.B. ALGOL 60, wo Kommentarabschluß = Statementtrennzeichen ist).

(6) Anzeige von Optimierungen

Optimierende Compiler sollten (u.a. zur besseren Interpretierbarkeit der Aussagen von dynamischen Testhilfen) ihre Maßnahmen anzeigen. Dies könnte z.B. durch ein zweites Quellprotokoll geschehen, in dem die Optimierungen in Form von Quelländerungen wiedergegeben sind. Beispiele anzuzeigender Optimierungen sind:

- Entfernung toten Codes
- Vorziehen gemeinsamer Teile aus arithmetischen Ausdrücken
- Herausziehen von schleifeninvarianten Ausdrücken etc. vor Schleifen und sonstige Umstellung der Abarbeitungsreihenfolge der Statements
- Verzicht auf Wertzuweisung an tote Variable
- Verzicht auf Wertzuweisung an Laufvariable nach Schleifenoptimierung
- Einsubstituieren von (offenen) Unterprogrammen in ihre Aufrufstelle (z.B. wenn das Unterprogramm nur einmal aufgerufen wird)
- Einsubstituieren der Parameter-Beschaffung in Unterprogramme (wenn z.B. der aktuelle Parameter bei allen Aufrufen derselbe ist).

(7) Anzeige von Blockverschachtelungen und ähnlichen Klammerstrukturen

Zweck ist die leichtere Richtigkeitskontrolle durch den Programmierer. Der Compiler kann die Struktur z.B. anzeigen durch Blocknummern an jeder Zeile, Markieren von Blockanfang und Blockende, typographische Aufbereitung des Quelltextes (Einrücken, Seiteneinteilung etc.).

(8) Erstellung dokumentarischer Information

Eine Reihe dokumentarischer Informationen ist von großer Bedeutung (sie können leicht während der Kompilation oder später beim Bindelauf erstellt werden) und sollten auf Wunsch ausgegeben werden z.B.:

- Statische Aufrufverschachtelung (welches Programm ruft welche auf?)
- "Adressraumüberlappungen" ("Welche Variablen des rufenden Programmes verwendet das Unterprogramm außer den aktuellen Parametern"? z.B. Common-Blöcke, globale Variable in ALGOL)

- Erstellen der Datei- und Datensatzstrukturen (Dateibeschreibung) der vom Programm erzeugten bzw. verwendbaren Dateien, wenigstens soweit statisch definiert.

3.3 Software-Technik

Es sollten die neueren Erkenntnisse aus der Softwareentwicklung angewendet werden. Gedacht ist hierbei an die Unterstützung solcher Techniken wie "chief programmer concept", "Methodenbanken" und "strukturierte Programmierung". Das angestrebte Software-Erstellungssystem sollte so flexibel sein, daß für einzelne Projekte die jeweils gesetzten Normen für die Programmierung automatisch überwacht werden (etwa "goto-freie Programmierung"). Außerdem sollte ein integraler Bestandteil des Systems die Buchführung über den Projektfortschritt übernehmen. Es sollte leicht feststellbar sein, in welchem Zustand sich ein Projekt befindet, wie weit die einzelnen Module fertiggestellt sind und ob die jeweils vorgeschriebenen Dokumentationsunterlagen vorliegen (Projektkontrolle).

3.3.1 Techniken für "besseres Programmieren"

Es wird von verschiedensten Autoren immer wieder gefordert, daß die Entwicklung der Software genauso ingenieurmäßig betrieben werden muß wie die Entwicklung der Hardware. Diese Forderung besteht zu Recht. Die Realisierung setzt aber voraus, daß das erforderliche "Werkzeug" zur Verfügung steht, z.B. in Form von Makrogenerator-Systematiken (z.B. STAGEII) oder Generatoren zur Unterstützung der "Strukturierten Programmierung".

3.3.2 Programmentwicklungs- und Programmdokumentationssystem

Ein solches System stellt all die Hilfen zur Verfügung, die die Erzeugung von Programmen bzw. Programmpaketen aus Modulbibliotheken mit gleichzeitiger Dokumentationserzeugung ganz oder weitgehend ermöglichen, wie "Methodenbanken", normierte Programmbibliotheken u.ä..

Nachfolgend wird stichwortartig (nicht erschöpfend !) der mögliche Leistungsumfang eines "Programmentwicklungs- und Programmdokumentationssystems" dargelegt.

(1) Grundsätzliche Zielsetzung

Alle Phasen der Systementwicklung von der Studie bis zum fertigen Programm sollen systematisiert und durch maschinelle Hilfsmittel rationalisiert werden. Dazu wird die Systembeschreibung, die sukzessive während des Projektlaufes aufgebaut wird, in einer Datei gespeichert. Gesteuert durch den Dateiinhalt werden durch Programm-Generatoren die Programme maschinell erzeugt. Die Dokumentation wird durch Auswertungsprogramme ebenfalls aus dem Dateiinhalt abgeleitet. Dadurch ist die Übereinstimmung von Dokumentation und Programmen gewährleistet.

(2) Systembeschreibungssprache

Mit Hilfe einer Systembeschreibungssprache (Logikbeschreibungssprache) - etwa an COBOL angelehnt und leicht erlernbar - hat der Benutzer die Möglichkeit, Informationssysteme mittels z.B.

- Integrationsschaubilder
- Aufgabenbeschreibungen
- Kanalbeschreibungen
- Algorithmen
- Entscheidungstabellen
- Tabellen
- Satzbeschreibungen
- Listbilder

zu beschreiben.

Die Aufgaben werden durch Programme, die Kanäle durch Dateien realisiert. Dementsprechend sind als weitere Sprachelemente (zur Definition der technischen Einzelheiten) erforderlich:

- Programm-Montageanweisungen und
- Dateibeschreibungen

Die Programm-Montageanweisung dient dabei in Anlehnung an den Arbeitsplan im Maschinenbau der Definition des Programms, das aus einzelnen Bausteinen der Systembeschreibung zu montieren ist.

(3) Methodenbank

Die den Sprachelementen der Systembeschreibungssprache entsprechenden Bausteine werden in einer Datei gespeichert. Diese Datei, die "Methodenbank" kann daneben auch eine Rei-

he anderer Bausteinarten, wie Anweisungen der JCL, Testdaten und dergl. aufnehmen. Die Bausteine der Methodenbank können mit entsprechenden Programmen gewartet werden. Sie können gegebenenfalls am Bildschirm angezeigt werden, um Auskunft über Inhalt und Stand einzelner Bausteine zu geben.

(4) Generator-System und Programm-Generatoren

Die Programm-Generatoren selektieren aufgrund der Programm-Montageanweisungen die für ein Programm benötigten Bausteine aus der Methodenbank, übersetzen sie in eine Programmiersprache und fügen die Programmstücke zu einem lauffähigen Programm zusammen.

Kern des Generator-Systems ist ein Generator, der es gestattet, auch komplizierte Mehrphasenprogramme zu erstellen; alle bekannten Speicherungs- und Zugriffsarten, einschließlich der Datenbanksysteme und der Bildschirme, werden unterstützt. Als weitere Generatoren sind mindestens erforderlich: Ein Entscheidungstabellengenerator, ein Listbildgenerator und - falls eine der üblichen (also veralteten!) Job-Control-Sprachen verwendet werden muß - ein Generator zur Erzeugung der JCL-Anweisungsfolge ("Steuerkartengenerator").

(5) Auswertungsprogramme

Für verschiedenste Auswertungen sollten Dienstprogramme zur Verfügung stehen, so z.B. für:

- Selektion von Systembeschreibungen aus der Methodenbank,
- Inhaltsverzeichnis und Referenzliste für die Methodenbank,
- Programmstücklisten und Teileverwendungsnachweis für die Bausteine.

(6) Eine kritische Bemerkung zu Programm-Generatoren

Was unterscheidet eigentlich einen Programm-Generator (oder ein Generatoren-System) von einem Compiler? Dazu betrachten wir seine Eingabe-Sprache: Sie ist irgendeine problemorientierte Sprache, z.B. zur Beschreibung von Entscheidungstabellen. Betrachten wir seine Zielsprache: Sie ist eine gängige Programmiersprache, z.B. COBOL. Der Unterschied zwischen Programmgenerator und Compiler ist also nur die Zielsprache, die bei letzteren eine Maschinensprache ist.

Also ist ein Generatorsystem ein Compiler. Beim Einsatz von Compilern für höhere Sprachen ist es aber - wenn eine Programmänderung nötig ist - selbstverständlich, daß der Programmierer nicht im erzeugten Code herumbastelt, sondern in der Quellsprache ändert. Genauso sollte man sich beim Einsatz von Programmgeneratoren darüber im Klaren sein, daß man nicht in den von solchen Generatoren erzeugten Codes herumbastelt, egal ob nur eine temporäre oder versuchsweise sonder sonst wie bedingte Programmänderung gewünscht wird. Damit würde die eigentliche Intension, nämlich eine einfache, problemnahe und einfache Formuliermöglichkeit (eben die Systembeschreibungssprache) einzusetzen durchbrochen.

Aus der Sicht des Informatikers ist sogar zu bedenken, ob denn Programm-Generatorsysteme nicht bloß eine momentane Erscheinung im Bereich der Automatisierung von Programmentwicklung und -Dokumentation sind. Die konsequente Verfolgung der Grundidee führt nämlich vom Programmgenerator zu einem Compiler mit entsprechender Quellsprache (z.B. Entscheidungstabellen), bei dem zusätzlich die Möglichkeit zum "Sprachtransfer" zu geeigneten höheren Programmiersprachen (z.B. COBOL, ALGOL, PL1, ...) besteht, damit ggf. Teile in solchen Sprachen formulierbar sind. Bei guter Integration ins Betriebssystem wäre auch ein "Steuerkartengenerator" hinfällig.

3.4 Software-Revision

Mit den in Abschnitt 3.1 beschriebenen Maßnahmen ist sichergestellt, daß eine Version eines Programmes eindeutig identifizierbar ist. Um der in Zukunft voraussichtlich verlangten noch schärferen Kontrolle laufender Programme zu genügen, muß sichergestellt werden, daß Programmodule auch in Bibliotheken eindeutig gekennzeichnet sind und darüberhinaus ihre Erstellungsweise sofort rekunstruierbar ist. Die eingesetzten Programme sind jeweils mit den externen Programmnamen und Versionsnummern zu protokollieren. Gleiches gilt für die verwendeten Dateien. Des weiteren müssen die in Abschnitt 2.3.5 (Datenschutz) beschriebenen Hilfen zur Verfügung stehen und auch von den Programmen voll angewendet werden. Laufzeit- und Arbeitsspeicherbedarfskontrolle sollten auch auf einfache Weise laufend möglich sein.

Neben diesen aus der Software-Revision direkt resultierenden Forderungen müssen Hilfen zur Nachprüfung der Richtigkeit ("Programmverifikationshilfen")verlangt werden. Dabei ist an einfache Hilfen (z.B. automatische Oberprüfung gewisser vom Programmierer in den Quellcode einzubringenden "typographischen Redundanzen"; Beispiel dazu: Klammerstrukturen, Blockschachtelungen, Kommentare werden vom Compiler daraufhin überprüft,

ob sie mit entsprechenden Einrückungen des Programmtextes korrespondieren und/oder in sinnvollem Zusammenhang stehen) bis hin zum "automatischen Richtigkeitsbeweis" von Programmen zu denken. Ansätze zu letzterem nehmen z.B. "Schleifen-Invariante" die der Programmierer zu liefern hat, zu Hilfe. Automatische Programmhinweise*) würden neben der Erfüllung wichtiger Forderungen der Rivision insbesondere beim Programmtest erhebliche Einsparungen bringen.

Weiterhin sind Hilfen zur vereinfachten Erprobung (Erstellung von Ablaufplänen und Daten zum Test u.v.a.m.), sowie die Automatisierte Dokumentation der durchgeführten Tests samt Ergebnissen erforderlich.

3.5 Interaktive Programmerstellung

Interaktive Programmentwicklung muß für alle bedeutenden Programmiersprachen möglich sein, d.h. das Erstellen und das Testen von Programmen im echten Dialog. Wenn nun nachfolgend Forderungen genannt werden, die u.U. an anderer Stelle dieses Berichtes auch erscheinen, dann geschieht das, um die große Bedeutung zu unterstreichen, die die interaktive Programmerstellung inzwischen hat.

Es ist ein leistungsstarkes Texthaltungssystem erforderlich, das vielseitige und leicht handhabbare Funktionen zur Speicherung und zur Manipulation beliebiger Texte, insbesondere für Programme und deren Daten bietet.

Die meisten der heute gängigen Programmiersprachen sind für einen intensiven Dialog je eingegebenen Befehl aus verschiedenen Gründen wenig geeignet. Hilfen können jedoch z.B. Prompter (vergl. (5) in Abschnitt 3.2.1) geben. Ansonsten wird bei der Programmerstellung der Dialog so durchgeführt, daß kleinere Kompilations- und Testläufe direkt im interaktiven Betrieb geschehen, größere Arbeiten aber auch in eine Stapel-Eingabewarteschlange abgegeben werden.

---

*) Literaturhinweise: C.A.R. Hoare "An axiomatic Basis for Computer Programming" in Comm. ACM 12, 10 (1969), P. 576-580; N.Wirth "Systematisches Programmieren", Teubner-Verlag, Stuttgart, 1972; F.W. von Henke und D.C. Luckham "Automatic Program Verification", Stanford University of California, Dept. of Computer Science (1974), STAN-CS-74-474, AIM-256.

Es ist also erforderlich, daß alle Leistungen der Testhilfen und Protokolldienste (soweit logisch sinnvoll) auch für die interaktive Programmentwicklung zur Verfügung stehen (vgl. auch Abschnitt 3.2).

Weiterhin muß es möglich sein, bereits vorhandene Methoden bzw. Bausteine (z.B. aus einer Methodenbank; siehe auch Abschnitt 3.3.2) in den Dialog einzubeziehen.

Beispiele zur interaktiven Programmerstellung:

(1) Programmablauf unter interaktiver Kontrolle des Benutzers. Die Echtläufe finden wie die Compilations- und Testläufe im interaktiven und im Stapel-Betrieb statt.

Die Unterbrechungen der Ausführungsphase können sein:

a) vorprogrammierte Unterbrechungen:
Der Dialog mit dem Benutzer erfolgt an vorher im Programm festgelegten Stellen. Das System wartet an diesen Stellen auf Benutzereingabe und der Benutzer erwartet die Ausgabe des Systems an der Datenendstation;

b) Eingriffe von außen:
Hier kann der Benutzer an beliebigen Stellen den Programmablauf unterbrechen (s. z.B. (5) in Abschnitt 3.2.2) und einen Dialog beginnen. Ein solcher Dialog gestattet die Bearbeitung wenig strukturierter Probleme, in denen der Benutzer aufgrund von Teilergebnissen die Problemformulierung vervollständigt;

c) Fehlerfälle.

Die Fortsetzung an der Unterbrechungsstelle kann geschehen durch:

a) Eingabe von Daten, die eventuell mit Hilfe von Tischrechnerfunktionen des Computers ermittelt werden;

b) Eingabe von Anweisungen;

c) Starten anderer Programme, d.h. neuer oder bereits im Unterbrechungszustand wartender Programme.

(2) Interaktive Ergebnisauswertung:

Selektive Ausgabe der Ergebnisse der ausgeführten Programme u.v.a.m. ist mit Hilfen, wie sie z.B. in Abschnitt 3.2 beschrieben sind möglich. Es sollte selbstverständlich sein,

daß diese Dinge an jedem Groß-Rechnersystem und für jede Programmiersprache voll verfügbar wird. Wie unbefriedigend die diesbezügliche Situation heute noch ist, mag man daraus ersehen, daß es bereits als großer Erfolg empfunden wird, wenn heute wenigstens von einigen Computerherstellern für gewisse Anlagen wenigstens für eine Sprache (nämlich COBOL) ein "interaktiver DEBUG" angeboten wird.

4. Migrationshilfen

4.1 Allgemeines zu Migration

Unter Migration sei hier nicht nur der allmähliche Übergang der Anwendungen von einem vorhandenen Rechnersystem zu dessen Nachfolger verstanden. Vielmehr sei auch der Fall eingeschlossen, daß zu einem vorhandenem Rechnersystem ein weiteres hinzukommt und die bisherigen Aufgaben sowohl auf der einen wie der anderen Anlage erledigbar sein sollen.

Zur allgemeinen Erleichterung von Migration gibt es zwei wichtige Grundideen:

(1) Portable Software:

Die System-Software wird möglichst rechnerunabhängig erstellt. dafür ist jedoch notwendig, daß das Rechnersystem die Werkzeuge für die Erstellung portabler Programme bereitstellt. Zu nennen sind normierte Systemerstellungs-Sprachen wie BCPL, PASCAL u.ä. sowie Hilfsmittel zur Implementation von abstrakten Maschinen wie Makro-Generatoren (etwa STAGE2, MLI etc.). Es ist z.Zt. zu beobachten, daß diese Lösung zunehmend größeren Anklang findet. Man muß also fordern, daß ein zukünftiges Groß-Rechnersystem solche Software-Erstellungshilfen unterstützt. Weiterhin müssen Hilfen im Sinne von Abschnitt 3.3 untersützt werden. Schließlich muß die Anwendungssoftware selbst möglichst portabel sein (siehe dazu u.a. Abschnitt 1.1).

(2) Kompatible Systeme:

Das (zukünftige) Groß-Rechnersystem besitzt bezüglich Software möglichst viele Gemeinsamkeiten mit bereits existierenden Systemen und auch deren Nachfolgern. Dies führt u.a. zu dem Versuch, Programmiersprachen, Datenträger, JCL, Kommunikations-Schnittstellen etc. zu normieren. Desweiteren wäre das der Versuch, Rechnersysteme generell zu vereinheitlichen, wobei es fraglich ist, ob eine diesbezügliche Entwicklung in Zukunft zu erwarten ist; es steckt in einer solchen Unternehmung sogar eine Gefahr, nämlich die, daß sich alle Hersteller dem Marktführer anpassen, womit der technische Fortschritt erheblich verlangsamt wäre.

Es ist nicht zu erwarten, daß in naher Zukunft eine dieser Grundideen voll zum tragen kommt. Man wird sich in der Praxis mit teilweisen Lösungen zufrieden geben müssen. Es sind deshalb nachfolgend einige weniger globale Gedanken zu der Migrationhilfen aufgeführt. Sie richten sich sowohl an Hersteller von Rechnersystemen wie auch an Ersteller von Anwendungssoftware und der nötige Aufwand erscheint, bezogen auf den erreichbaren Nutzen, durchaus vertretbar.

4.2 Migrationshandbücher

Vom Hersteller des zukünftigen Groß-Rechnersystems ist zu erwarten, daß er alle Möglichkeiten in Form eines Handbuches beschreibt, die in seinem System bei der Erstellung von portablen Programmen genutzt werden können, und daß er ferner eine Liste aller derjenigen Systemeigenschaften bereitstellt, bei deren Verwendung die Migration erschwert wird. Insbesondere ist in diesem Zusammenhang eine Aufstellung aller Sprachmittel erforderlich, welche in einer normierten Programmiersprache anlagenspezifisch sind oder an der Anlage von einem vielverwendeten Sprachdialekt ("Quasi-Norm") abweichen. Weiter sollten Abweichungen von normierten oder üblicherweise gebrauchten Informations-Verschlüsselungs-Systemen (z.B. ISO-7-Bit-Code) aufgeführt werden.

Um Anwendungssoftware effizient zu gestalten, kann es notwendig sein, spezielle Betriebssystemeigenschaften auszunutzen. Von dem Ersteller der Anwendungssoftware ist dann zu erwarten, daß er die Schnittstellen zum Betriebssystem klar dokumentiert, und daß sie über einen separaten Modul aufgerufen werden. Dieser Modul muß bei einem Systemwechsel leicht austauschbar sein. Ferner muß der Ersteller von Anwendungssoftware darauf hinweisen, wo er implizit oder explizit die Rechengenauigkeit und andere spezielle Eigenschaften eines bestimmten Rechnersystems ausnutzt.

Insgesamt betrachtet muß man also verlangen, daß sowohl der Hersteller des Groß-Rechnersystems als auch der Ersteller der Anwendungssoftware je ein Migrationshandbuch mitliefern.

4.3 Anpassung des Sprachumfanges

Wo anlagenspezifische Abweichungen vom normierten Sprachumfang (bzw. "Quasi-Norm", vgl. Abschnitt 4.2) gängiger Programmiersprachen einem Hersteller unumgänglich oder dringend wünschenswert erscheinen ist folgendes erforderlich:

(1) Es sollte nie Einschränkungen im Sprachumfang gegenüber der Norm bzw. Quasi-Norm geben, sondern nur Erweiterungen.

(2) Die Compiler sollen die Stellen anzeigen, an denen solche Erweiterungen im Quellprogramm auftauchen und auch zur Laufzeit des Programms sollte angezeigt werden, wenn solche Erweiterungen benutzt werden.

(3) Der Hersteller hat Wandelprogramme zu liefern, die Quelltexte mit normierter Sprache in anlagenspezifische Sprachdialekte übersetzen und umgekehrt.

4.4 Bemerkungen zu einigen Benutzerschnittstellen

(1) Obwohl die Forderung nach einer leistungsfähigen Job-Control-Language an anderen Stellen schon erhoben wurde, sei sie hier im Hinblick auf Migration nochmals erwähnt, denn die Umstellung der JCL-Anteile von Jobs stellt einen erheblichen Anteil der bei der Migration zu leistenden Gesamtarbeit dar. Also muß JCL eines neuen Rechnersystems (im Sinne einer höheren Sprache) leistungsfähig sein, zwecks geringen Aufwandes bei der Neu-Formulierung des JCL-Anteils der zu übernehmenden Jobs.

(2) Mindestens die häufig benutzten Programmiersprachen ALGOL 60, BCPL, COBOL, ALGOL 68, FORTRAN, LISP, PASCAL, PL/1, SIMULA, sollen mit einem adäquaten Sprachumfang an einem neuen System verfügbar sein.

(3) In Ergänzung zu Abschnitt 2.3 sei im Hinblick auf Migration betont, daß die Datenhaltung das Spektrum der üblichen Dateizugriffsmethoden und -organisationen vollständig überdecken muß. Es ist nicht zumutbar, wenn die Logik eines zu übernehmenden Programms geändert werden muß, weil die verwendete Dateiorganisation keine Entsprechung im neuen System hat.

(4) Bezüglich aller externer Datenträger ist eine Normierung der Informationsverschlüsselung, Kennsätze u.s.w. dringend nötig. D.h., daß z.B. auch für Magnetplatten mindestens die Vereinheitlichung nötig ist, wie sie bei Magnetbändern allmählich erreichbar zu werden scheint.

4.5 Adaptierung zusätzlicher Compiler

Schon heute gilt für einen Großteil der Anwendungsgebiete, daß Entwicklung und Einsatz von Software nur dann finanziell und Kommerziell vertretbar sind, wenn die Software auf verschiedenen Groß-Rechnersystemen lauffähig ist. Zukünftige Groß-Rechnersysteme müssen deshalb so konstruiert sein, daß eine Übernahme von auf anderen Rechnern laufender Anwendungssoftware leicht möglich ist.

Ein Punkt, dem dabei besondere Beachtung zukommt ist, daß Anwendungssoftware (oder Teile davon) immer in der geeignetsten Programmiersprache formuliert werden sollen (ohne Rücksicht

auf die Verbreitung der Sprache) und in dieser Sprache auf ein neues Rechnersystem übernommen werden können. Also muß das neue System ggf. Hilfen zur leichten Implementation weiterer Sprachen bieten. Dazu gehören:

(1) Standardisierte Schnittstellen(d.h. einheitlich für alle Sprachen des neuen Rechnersystems) für Compiler, Dump-Operator, Rückverfolger usw.. Ziel ist, daß Testhilfen, Protokollierung, die JCL-Statements für Übersetzung, die Anschließbarkeit von Unterprogrammen in anderen Sprachen ("Sprachtransfer") etc. auch für nachträglich am Rechnersystem implementierte Sprachen,vom Benutzer gesehen,im gewohnten Schema liegen (vgl. u.a. Abschnitt 3.2).

(2) Standardisierte Schnittstellen (d.h. einheitlich für alle Sprachen des neuen Rechnersystems) für die (bei Kompilation) erzeugten Objekte, z.B. Schnittstellen zur Datenverwaltung , sonstige E/A-Schnittstellen, Schnittstellen zur JCL-Sprache und zu interaktiven Testhilfen, Unterprogrammanschlüsse, Anschlüsse von Unterprogrammen für fremde Sprachen.

(3) Schemata, Normen, Routinen etc. für die interne Organisation (wie Stackverwaltung) Pufferverwaltung, Unterprogramm-Aufrufe, Parameterübergabe, Lauf-Anfangs- und Lauf-Ende-Organisation, Ablage von Variablen, Feldern, ...) sollten möglichst einheitlich für alle Objekte sein und (soweit logisch möglich) nicht davon abhängen, aus welcher Sprache sie (per Übersetzung) erzeugt wurden.

5. Messungen beim Betrieb und Kostenabrechung der Benutzung von Anwendungssoftware

Die steigenden Kosten für die Produktion, Übernahme und Pflege von Software erzwingen wirksame Mechanismen zur Erfassung und Weiterverrechnung aller dieser Kosten. Der Hersteller von Anwendungssoftware (der im allgemeinen nicht Hersteller des Groß-Rechnersystems sein wird) sollte im Auge haben, daß nicht immer der Betreiber einer DV-Anlage Endabnehmer des Softwareproduktes ist, sondern daß für die gesamte Anwenderhierarchie (vgl. Abschnitt 0., "Vorbemerkungen") eine Feststellung und Weitergabe der Kosten möglich sein muß.

Besonders im Bereich der Dienstnehmer sind vielfältige organisatorische Konstruktionen verbreitet. So sind z.B. an großen Rechenzentren oft Service-Unternehmen als Benutzer tätig, die ihre Dienstleistungen ihrerseits an einzelne "Kunden" weitergeben. Solche Service-Unternehmen sind z.B. die EDV-Abteilungen großer Institute, teils mit eigenem Rechner, teils auch ohne eigene Hardware. Teilweise nehmen aber auch die Produzenten von Anwendungssoftware Rechenzentren als Benutzer in Anspruch und geben die Leistungen ihrer Anwendungssysteme selbst an andere Benutzer weiter, wobei eine Vielzahl von vertraglichen Regelungen denkbar ist und auch praktiziert wird. In jedem Fall sollte es möglich sein, die Softwarekosten auf die Endbenutzer in gerechter Weise zu verteilen. Hierzu sind spezielle Leistungen sowohl des Betriebssystems (Abrechnungsdaten bedürfen eines besonderen Schutzes), als auch des jeweiligen Anwendungssystems erforderlich.

5.1 Monitore

Monitore sind Programmodule oder selbständige Spezialrechner zur Ermittlung von Benutzungsprofilen, Benutzerprofilen, Ablaufprofilen etc., sowohl von Programmen oder Programmteilen in höheren Sprachen, als auch von JCL-Programmen. Für Wirtschaftlichkeits-Betrachtungen bei Anwendungssystemen sind derartige Werkzeuge unentbehrlich.

(1) Benutzungsprofile stellen die Belastung der DV-Anlage, aufgeschlüsselt auf die einzelnen Programme dar. Dies ist für Rechenzentren wichtig, um den Pflegeaufwand für die Anwendungssysteme entsprechend ihrer wirtschaftlichen Bedeutung für das Rechenzentrum bemessen zu können.

(2) Benutzerprofile schlüsseln die Benutzung eines Anwendungssystems nach Benutzern (wer, wann, wie, oft, welche Betriebsmittel ...) auf (wichtig für Lizenzgeber und Rechenzentrum).

(3) Ablaufprofile stellen die Häufigkeit und die Betriebsmittelnutzung der einzelnen Teile des Anwendungssystems beim dynamischen Ablauf fest. Der erforderliche Detaillierungsgrad derartiger Statistiken ist abhängig vom jeweiligen Interessenten innerhalb der Anwenderhierarchie: Der Benutzer wird durch Auskünfte über den Aufwand einzelner Algorithmen u.U. zu wirtschaftlicherer Arbeitsweise geführt. So könnte für den Anwender eines Formel-Manipulationssystems die Aufschlüsselung der Rechenzeit nach Bedarf für Ausgabeformatierung und echte Auswertung wichtig sein - jedenfalls dann, wenn er Einfluß auf Leistung und Umfang bei der Ausgabeformatierung innerhalb des Anwendersystem hat. Ebenso sollten z.B. Statistiksysteme dem Benutzer den Betriebsmittelbedarf für die einzelnen Verfahren mitteilen. Für derartige Ablaufprofile müssen die erforderlichen Meßpunkte verfahrensorientiert im Anwendungssystem selbst implementiert sein. Das Betriebssystem muß lediglich die nötigen Meßdaten zur Verfügung stellen.

Das Rechenzentrum als Implementierer und Betreiber braucht neben verfahrensorientierten Übersichten wesentlich detailliertere Auskünfte über den Programmlauf, um einen effizienten Rechenbetrieb zu gewährleisten. Programme zur Erstellung von Programmprofilen (siehe auch Abschnitt 3.2.1), die Durchlaufstatistiken auf der Ebene der Implementierungssprache (oder sogar detaillierter) über Betriebsmittelbedarf (CPU, virtueller Speicher, ...) liefern, sollten vorhanden sein. Bei der Auswahl der Implementierungssprache sollte man diesen Gesichtspunkt nicht vernachlässigen.

Ferner ist zu fordern, daß Systeme zur automatischen Software-Erstellung (siehe auch Abschnitt 3.3.2) die Anschlußstellen an Monitore mitliefern. Der Produzent von Anwendungssoftware wird auch im eigenen Interesse versuchen, eine Vielfalt derartiger Werkzeuge einzusetzen, um konkurrenzfähige Produkte anbieten zu können. Für die hier erwähnten Monitore gilt, daß sie in einfacher Weise an- und abschaltbar sein müssen, um den eigentlichen Produktionslauf so schnell und sparsam wie möglich zu halten und Messungen auf das jeweils notwendige Mindestmaß im jeweils notwendigen Bereich zu beschränken.

5.2 Kostenabrechnung

Die Benutzungskosten für Anwendungssysteme setzen sich zusammen aus:

(1) den üblichen Kosten für die Benutzung der DV-Anlage (Betriebsmittelkosten und Verbrauchsmaterialkosten), bezüglich derer sich das Anwendungssystem nicht von anderen Programmen unterscheidet und

(2) den spezifischen Kosten für die Benutzung des Anwendungssystems.

Wir beschäftigen uns im Folgenden vornehmlich mit den letztgenannten, weil die Betriebsmittel-Abrechnung bereits Gegenstand einer früheren Publikation der STARG war.

Die anwendungssystem-spezifischen Kosten entstehen als Folge von Lizenzgebühren oder Entwicklungsaufwand, Implementierungs- und Wartungsaufwendungen, Speichergebühren, Beratungsaufwand u.ä. zwischen allen Stufen der Anwenderhierarchie.

5.2.1 Messung der Benutzung

Die Messung der Benutzung von Anwendungssystemen sollte nach anlagenunabhängigen aber anwendungssystem-bezogenen Maßzahlen (Leistungsmengen) erfolgen (z.B. Zahl der geschriebenen Ausgabesätze, Zahl der Eingabesätze mit einem verfahrensspezifischen Faktor, o.ä.).

Das Anwendungssystem erfaßt die Leistungsmengen möglichst unter Vermeidung nicht allgemein üblicher Systemdienste und gibt sie zur Sicherung und weiteren Verarbeitung an das Betriebssystem weiter. Ein entsprechender Systemdienst "Privatstatistik" beim TR440) ist unverzichtbar, weil Abrechungsdaten besonders gegen Verlust und Änderung geschützt werden müssen und die Koordinierungsmechanismen für das gleichzeitige Schreiben mehrerer Programme in den gleichen Pool von Statistikdaten nicht bis auf die Ebene des Anwendungssystems hochgezogen werden sollten.

Eine Erfassung der Leistungsmengen und anderer Benutzungsdaten an zentraler Stelle ist auch deshalb geboten, weil alle für Kostenabrechnung und Statistik erforderlichen Daten zusammenhängend verarbeitet und besonders sorgfältig verwaltet werden müssen.

5.2.2 Kostenberechnung

Aus den (gem. Abschnitt 5.2.1 zentral erfaßten) Leistungsmengen werden aufgrund einer Preisliste, die sowohl vom Lizenzgeber als auch vom Rechenzentrum vorgegeben sein kann, die spezifischen Kosten für die Benutzung des Anwendungssystems berechnet. Während die Leistungsmengen anlagenunabhängig und nicht manipulierbar sein sollten, können für ein Anwendungssystem mehrere Preislisten (z.B. extern, intern oder gemäß Benutzerhierarchie) existieren. Am Ende des Programmlaufes sollten die ermittelten Leistungsmengen aus Gründen der Kostentransparenz ebenso wie die kostenrelevanten Betriebsmittel im Benutzerprotokoll ausgewiesen werden.

Bei einigen Anwendungssystemen hat es sich im Rechenzentrumsbetrieb bewährt, für die diversen Kundenklassen die Abrechnung ihrer Benutzung durch Multiplikation der "normalen" Abrechnung mit einem Faktor vorzunehmen. Voraussetzung für diese in der Praxis verwendete Methode ist, daß die Betriebsmittel-Nutzungsmengen nicht nur für den Job als Ganzes, sondern auch programmspezifisch erfaßt werden. Dabei muß die Möglichkeit gegeben sein, daß das Anwendungssystem zusätzliche Information, etwa über Abrechnungs- und Benutzerklassen diesen Daten hinzufügen kann.

Unabhängig von der Art der Abrechnung sollte der Hersteller des Groß-Rechnersystems Schnittstellen zur Aufnahme und Weiterverarbeitung der Abrechnungsdaten aus den Anwendungssystemen im Standard-Abrechnungsprogramm zur Verfügung stellen.

5.2.3 Prüfung auf Berechtigung zur Benutzung von Anwendungssoftware

Neben der Möglichkeit einer detaillierten Kostenabrechnung für die Benutzung der Anwendungssoftware wird es wichtig sein, bei deren Aufruf zu prüfen, ob der aufrufende Benutzer auf diese Software überhaupt zugreifen darf.

Um eine Abstufung der Benutzungserlaubnis (z.B. nur Teile, Ergänzen des Anwendungssystems durch private Moduln, Maintenance, Veränderung der Benutzungsberechtigung, Benutzung nur bis zu m Leistungseinheiten usw.) implementieren zu können, muß auf spezielle Systemdienste zurückgegriffen werden. In diesem Zusammenhang ist die Möglichkeit, Code nur zur Ausführung, nicht aber zum Lesen oder Schreiben freizugeben, besonders wichtig. Ferner muß es möglich sein, besonders geschützte Routinen zum Prüfen von Paßworten oder Paßwort-Algorithmen in die Anwendungssoftware zu integrieren. Auch hat sich die Kontrolle, ob das Anwendungssystem von einem speziell hierfür zugelassenen Eingabegerät aus gestartet wurde, bewährt. Für solche Kontrollen kann auf die Möglichkeit, den Code gegen unbefugtes Lesen zu schützen, nicht verzichtet werden. Bezüglich der erreichbaren Sicherheit kann davon ausgegangen werden, daß solche Betriebssysteme, die allen Erfordernissen des Datenschutzgesetzes im Vielfachzugriffsbetrieb genügen, auch allen aus der Sicht der Anwendungssoftware zu stellenden Schutzforderungen gerecht werden.

BISHER ERSCHIENENE ARBEITSBERICHTE DES RECHENZENTRUMS
DER RUHR-UNIVERSITÄT BOCHUM

Nr. 7101: K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA; eine Dialogsprache für den TR440 (vergriffen)

Nr. 7102: K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, ein Dialogsystem und seine Implementierung in ALGOL (vergriffen)

Nr. 7103: K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, Manual für den Benutzer (vergriffen)

Nr. 7104: 4. Jahresbericht des Rechenzentrums (Juni 1970 bis Juni 1971) (vergriffen)

Nr. 7105: H. Wupper
WR MB02 – Ein einfaches Band-Betriebssystem für einen mittleren Rechner

Nr. 7201: H. Windauer
Existenzsätze nur (0,1,...,R-2,R)-Interpolation

Nr. 7202: W. Schelongowski
DIATRACE-Ein System zur interaktiven Assemblerprogrammierung

Nr. 7203: M. Jäger, M. Rosendahl, R. Staake
Einführung in die Listenverarbeitung anhand der Dialogsprache AIDA

NNr. 7204: R. Mannshardt, P. Pottinger
Einführung in die Benutzung des Teilnehmer-Rechensystems TR440 in der RUB (vergriffen)

Nr. 7205: 5. Jahresbericht des Rechenzentrums (1.7.1971 bis 30.6.1972)

Nr. 7206: M. Rosendahl
BOGOL-TAS, ein Weg zur systemnahen Programmierung in ALGOL am TR440

Nr. 7207: W. Stark
ILW, Programmsystem zur Berechnung des Instationären Ladungswechsels von Verbrennungskraftmaschinen (Modulbeschreibung und Eingabekonventionen) (vergriffen)

Nr. 7208: W. Stark
ILW, Programmsystem zur Berechnung des Instationären Ladungswechsels von Verbrennungskraftmaschinen (Regelmechanismus und Berechnung der Rohrströmung) (vergriffen)

Nr. 7209: H. Ehlich
Anregung und Kritik zum Betriebs- und Programmiersystem der TR440

Nr. 7210: M. Rosendahl
BOGOL-STRING, eine flexible Zeichenkettenverarbeitung in ALGOL 60

Nr. 7211: H. Camici, H. Claus, H. Ehlich, D. Kipp
Arbeitsbericht über ein Programm zur Haushaltsführung (vergriffen)

Nr. 7301: R. Mannshardt, K.-H. Mohn, H. Münch, P. Pottinger
Einführung in die Benutzung des Teilnehmer Rechensystems TR440
2. geänderte Auflage (vergriffen)

b.w.

Nr. 7302: K.-H. Mohn
Über einige Anwendungen des Computers in der Medizin

Nr. 7303: R. Buchmann
BODAT, ein schnelles und platzsparendes System zur Datenmanipulation und -speicherung in ALGOL 60 und FORTRAN (vergriffen)

Nr. 7304: M. Hauenschild
Ansätze zur komplexen Kreisarithmetik

Nr. 7305: R. Buchmannn
RB&QUELLHALT, ein TR440-Datenbanksystem zur platzsparenden Quellhaltung auf Datenträgern mit direktrem Zugriff (LFD, WSP)

Nr. 7306: 6. Jahresbericht des Rechenzentrums (1.7.1972 bis 31.12.1973)

Nr. 7401: R. Buchmann
Der Systemoperator BO&BS3OP
Messungen und Steuerungen des Betriebssystems auf Operatorebene

Nr. 7402: R. Mannshardt
Herleitung und Prüfung spezieller Runge-Kutta-Verfahren mit einem impliziten Rechenschritt

Nr. 7403: R. Buchmann, H. Wupper
Unzulänglichkeiten des TR 440 Programmiersystems und ihre Umgehung (vergriffen)

Nr. 7404: R. Green, K.-H. Mohn
Quellbezogene FORTRAN Optimierungen für den Compiler des TR 440

Nr. 7405: R. Buchmann
BODAT, ein schnelles und platzsparendes System zur Datenmanipulation und -speicherung in ALGOL 60 und FORTRAN (2..ergänzte Auflage)

Nr. 7501: R. Buchmann
Zur Theorie der Montage von Programmoduln

Nr. 7502: 7. Jahresbericht des Rechenzentrums (1.1. bis 31.12.1974) (vergriffen)

Nr. 7503: H.-D. Sander
BOTRAN, eine Fortran Spracherweiterung durch Code-Prozeduren

Nr. 7504: W. Schelongowski
DIATRACE - Ein System zur interaktiven Assemblerprogrammierung

Nr. 7505: Camici, Prof. Dr. Ehlich, Schürmann
Über ein Programm zur Material- und Vervielfältigungs-Abrechnung

Nr. 7506: Camici, Prof. Dr. Ehlich, Herrmannies
Über ein Programm für die Telefonabrechnung einer Nebenstellenanlage

Nr. 7507: Camici, Prof. Dr. Ehlich, Kipp
Über ein Programm zur Haushaltsführung

Nr. 7508: Camici, Cipa, Prof. Dr. Ehlich
Bericht über ein Programm zu Verwaltung der Studentendaten

Nr. 7509: J. Riege
Zur mehrdimensionalen Spline-Interpolation bezüglich beliebiger linearer Funktionale

Nr. 7601: H. Ehlich, J. Riege u. K-H. Schloßer
Ein Programmsystem zur Ausleihverbuchung und interaktiven Rechnerunterstützung in der allgemeinen Buchbestandsverwaltung
Teil 1: Offline-System

Nr. 7602: M. Rosendahl
BOGOL-STRING, eine flexible Zeichenkettenverarbeitung in ALGOL60
2. erweiterte und geänderte Version

Nr. 7603: R. Buchmann, M. Rosendahl
BOGOL-TAS, eine Spracherweiterung von ALGOL60 durch Codeprozeduren zur Systemprogrammierung

Nr. 7604: R. Buchmann
AUFBEREITE, ein universell einsetzbarer Dateiänderungsoperator für verschiedene Datei- und Dialoggerättypen und/oder Betriebsarten

Nr. 7605: 8. Jahresbericht des Rechenzentrums (1. Januar bis 31. Dezember 1975)

Nr. 7606: R. Buchmann, H. Wupper
BO&ZEICHNE, Vorschlag zur geräteneutralen Graphik (vergriffen)

Nr. 7701: Camici, Ehlich, Kipp, Wiedemann
Inventarisierungsprogramm

Nr. 7702: K.A. Görg, W. Stark, R. Wojcieszynski
Verfahren zur Berechnung der Strömung durch eine Drosselstelle ohne Speicherwirkung (vergriffen)

Nr. 7703: H. Wupper
ALGOL68 Eine Einführung in die Programmierung für Anfänger (vergriffen)

Nr. 7704: Camici, Prof. Dr. Ehlich, Volmer, Wiedemann
Aufbau und Verwendung der Personaldatei der Ruhr-Universität Bochum

Nr. 7705: M. Rosendahl
AIDA, Handbuch für den Benutzer

Nr. 7706: V. Riedel, K.-H. Schloßer
Ein Programmsystem zur Ausleihverbuchung und interaktiven Rechnerunterstützung in der allgemeinen Buchbestandsverwaltung
Teil 2: On-line-System

Nr. 7707: M. Peuser, H. Wupper
Ein geräteneutrales und rechnerneutrales System zur graphischen Ausgabe

Nr. 7708: H. Wupper
ERZEUGE, eine Erweiterung der Kommandosprache

Nr. 7709: A. Heidt, G. May
ADWAND ein System zur Bearbeitung von Analogdaten an TR86 und TR440

Nr. 7710: R. Buchmann
BO&PAGING Der virtuelle Kernspeicher für den TR440

Nr. 7711: R. Buchmann, H. Wupper
Unzulänglichkeiten des TR 440 Programmiersystems und ihre Umgehung
2.geänderte Auflage

b.w.

Nr. 7712: 9. Jahresbericht des Rechenzentrums (1. Januar 1976 – 31. Dezember 1976)

Nr. 7713: H. Ehlich, H. Zoller
Nachdruck des Arbeitsberichts Nr. 1 des AK 4 der STARG 440
"Anforderungen an ein zukünftiges Gross-Rechnersystem"

Nr. 7714: F.J. Delvos, H. Posdorf
Zur Optimalität von zweidimensionaler Serendipity-Interpolation

Nr. 7715: H. Ehlich, W. Stark, R. Wojcieszynski
Verfahren zur Berechnung der Strömung durch eine Drosselstelle ohne Speicherwirkung
2. überarbeitete Auflage

Nr. 7801: H. Posdorf
Boolesche Methoden bei zweidimensionaler Interpolation

Nr. 7802: H. Ehlich, F. Kruse, W. Stark
Verfahren zur Berechnung der instationären Rohrströmung

Nr. 7803: K.-P. Mankwald, M. Rosendahl
Über die Strukturanalyse von Programmen durch schrittweise Reduktion ihrer Strukturgraphen

Nr. 7804: F.J. Delvos, H. Posdorf
Zur Booleschen zweidimensionalen Lagrange Interpolation

Nr. 7805: G. Baszenski, F.J. Delvos, H. Posdorf
Darstellungsformeln zur zweidimensionalen Reduzierten Hermite-Interpoltion

Nr. 7806: 10. Jahresbericht des Rechenzentrums (1. Januar 1977 – 31. Dezember 1977)

Nr. 7807: H. Wupper
Experiences with Algol 68 Transput (vergriffen)

Nr. 7808: H. Ehlich, H. Zoller

Nachdruck des Arbeitsberichtes Nr. 2 des AK 4 der STARG 440
"Anwendungssoftware künftiger Groß-Rechnersysteme"