KERNFORSCHUNGSZENTRUM

KARLSRUHE

Mai 1972

KFK 1586

Institut für Reaktorentwicklung

Erfahrungen mit dem Programmsystem ICES bei ingenieurtechnischen Anwendungen

zusammengestellt von
E. G. Schlechtendahl, U. Schumann

GESELLSCHAFT FÜR KERNFORSCHUNG M. B. H.

KARLSRUHE

GESELLSCHAFT FÜR KERNFORSCHUNG M. B. H.
KARLSRUHE

KERNFORSCHUNGSZENTRUM KARLSRUHE

Mai 1972 KFK 1586

Institut für Reaktorentwicklung

Erfahrungen mit dem Programmsystem ICES bei ingenieurtechnischen Anwendungen

zusammengestellt von

E.G. Schlechtendahl

U. Schumann

mit Beiträgen von

G. Enderle

W. Katz

E.G. Schlechtendahl

H. Schnauder

U. Schumann

R. Schuster

Gesellschaft für Kernforschung m.b.H., Karlsruhe

Zusammenfassung

ICES (Integrated Civil Engineering System) ist ein am Massachusetts Institute of Technology (MIT) entwickeltes Programmsystem für den rechnergestützten Entwurf im Bauwesen. In seinen wesentlichen Bestandteilen ist es jedoch gleichermaßen in vielen anderen Bereichen des Ingenieurwesens einsetzbar, insbesondere auch im Maschinenbau, Reaktorbau und im Chemieingenieurwesen. Der vorliegende Bericht schildert Erfahrungen, die im Laufe von zwei Jahren von Mitarbeitern des Institutes für Reaktorentwicklung gesammelt wurden. Hierbei wurde ICES sowohl zur Lösung konkreter Probleme der Reaktortechnik angewandt als auch mit neuen Teilen ergänzt. Die dabei gesammelten Erfahrungen wurden zu der vorliegenden kritischen Beurteilung genutzt, die insbesondere potentiellen Benutzergruppen die Entscheidung für oder gegen die Einarbeitung in ICES und seine Verwendung erleichtern soll.

Abstract

ICES (Integrated Civil Engineering System) is a program system for computer aided design in civil engineering. It was developed at the Massachusetts Institute of Technology (MIT). The essential components of ICES can also be applied in other engineering fields, such as mechanical, nuclear and chemical engineering. This report is a summary of a two year experience with ICES. At the Institut für Reaktorentwicklung, ICES has been used for the solution of special nuclear engineering problems and was extended with new developments. This critical summary, based upon practical experience, shall help other potential users to decide, whether or not ICES might be the appropriate tool for their problems.

Inhaltsverzeichnis

1. Kurze Einführung in ICES

ICES (Integrated Civil Engineering System) ist ein am Massachusetts Institute of Technology (MIT) in den Jahren 1964 bis 1968 entwickeltes Programmsystem für den rechnergestützten Entwurf im Bauwesen [1,2,3,4]. In seinen wesentlichen Bestandteilen ist es jedoch gleichermaßen in vielen anderen Bereichen des Ingenieurwesens einsetzbar, insbesondere auch im Maschinenbau, Reaktorbau und im Chemieingenieurwesen. ICES wird Ende 1971 an über 400 Stellen in 24 Ländern eingesetzt. Der vorliegende Bericht schildert Erfahrungen, die im Laufe von zwei Jahren von Mitarbeitern des Institutes für Reaktorentwicklung gesammelt wurden, die sich ausschließlich anhand von Dokumentationen in die Benutzung von ICES eingearbeitet haben. Hierbei wurde ICES sowohl zur Lösung konkreter Probleme der Reaktortechnik angewandt als auch mit neuen Teilen ergänzt. Die dabei gesammelten Erfahrungen wurden zu der vorliegenden kritischen Beurteilung genutzt, die insbesondere potentiellen Benutzergruppen die Entscheidung für oder gegen die Einarbeitung in ICES und seine Verwendung erleichtern soll.

Die hier behandelten Fragen lauten:

Was ist ICES (als Programmsystem und Organisation)?
Was enthält ICES und wie gut ist der Inhalt?
Was sind die Besonderheiten von ICES als Programmsystem?
Wer sollte ICES benutzen, um damit Anwendungsprobleme zu lösen?
Wann sollte man in ICES neue Programme programmieren?

Nicht explizit untersucht wird hier die Frage:
Was kann man aus ICES für neue Programmsysteme zum rechnergestützten Entwurf lernen?
Ein Bericht über dieses Thema ist in Vorbereitung. Auch soll dieser Bericht nicht die große Anzahl der genauen Programmbeschreibungen ersetzen; vielmehr wird hier auf die relevanten Berichte hingewiesen, um so insbesondere dem deutschsprachigen Interessenten das Kennenlernen von ICES zu erleichtern.

Zum Druck eingereicht am 2.5.1972

1.1 Das Prinzip von ICES

Hier sollen einige Grundbegriffe und die wesentlichsten Komponenten von ICES erläutert werden, auf die in den folgenden Kapiteln immer wieder Bezug genommen wird. Je nach der Art des Umganges mit ICES erscheint dem Benutzer ICES unter einem anderen Aspekt. Für den Betreiber einer Rechenanlage beispielsweise (es handelt sich dabei stets um eine Anlage der Systeme IBM/360 oder 370) ist das ganze ICES nichts anderes als eine Anzahl von Programm= und Datenbibliotheken. Jede Benutzung von ICES bedeutet die Abgabe eines Jobs im Stapelbetrieb unter der Krontrolle des Betriebssystems.*)

Für den Anwender jedoch besteht ICES aus einer Menge von Programm = und Datenbibliotheken für verschiedene Problembereiche wie z.B.für Projektplanung oder für Festigkeitsanalysen. Die Gesamtheit aller koordinierten Hilfsmittel, welche ICES zur Bearbeitung von Aufgaben eines solchen fest umrissenen Problembereiches bereitstellt, wird als Subsystem bezeichnet. Diese Subsysteme sind für den Anwender völlig unabhängig voneinander. Jedes dieser Subsysteme hat eine eigene problemorientierte Sprache (POL) zu seiner Bedienung. Dennoch sind die ICES-Subsysteme nicht so verschieden voneinander wie völlig unabhängig entwickelte Programmpakete. Das liegt daran, daß sich alle Subsysteme eines einzigen Systemkerns bedienen, der für alle Subsysteme gleichermaßen bestimmte Leistungen bereitstellt. Die wichtigsten dieser Leistungen des Systemkerns sind: die Verwaltung von Programm= und Datenbibliotheken, die Steuerung des Programmablaufes und die Bereitstellung der von den Programmen benötigten Daten, die Interpretation der Eingabe des Subsystembenutzers, die Vermittlung von Leistungen des Betriebssystems (z.B.Lesen von Eingabe, Schreiben von Ausgabe) für die Programme des Subsystems und viele weitere Hilfsdienste. Ferner stellt der Systemkern alle diejenigen Leistungen bereit, die notwendig sind, bestehende Subsysteme zu verändern oder neue Subsysteme zu erstellen. Die Erkennntnis, daß alle problemorientierten Programmsysteme (hier also "Subsysteme") stets eine große Anzahl völlig identischer Anforderungen an das EDV-System stellen, in dem sie realisiert sind, und die daraus folgende Konzentration der entsprechenden Fähigkeiten in einem Systemkern ist eine der besonders beachtenswerten Leistungen der ICES-Entwicklung.

---

*) Bezüglich interaktiven Betriebs siehe Kap. 3.1.3, Seite 24

1.2 Geschichtliche Entwicklung

Das Programmsystem ICES (Integrated Civil Engineering System) geht in seiner Grundkonzeption auf das Jahr 1964 zurück. Es war gedacht als ein Mittel zur Unterstützung von Lehre und Praxis im Bauwesen. Die Forschungs- und Entwicklungsarbeiten wurden durchgeführt vom Civil Engineering Systems Laboratory (CESL) des Massachusetts Institute of Technology (MIT). Im CESL sind die verschiedensten Branchen des Civil Engineering Department des MIT zur Durchführung interdisziplinärer Vorhaben zusammengeschlossen. Folgende Organisationen unterstützten die Arbeiten:

Ford Foundation
IBM Corporation
MIT Lincoln Laboratories
Massachusetts Bay Transportation Authority
McDonnell Automation Company
Massachusetts Department of Public Works
National Science Foundation
Portland Cement Association
U.S. Bureau of Public Roads
U.S. Department of Transportation

Nach drei Jahren Entwicklungsarbeit und mit einem Aufwand von einigen hundert Mannjahren wurde ICES in den Jahren 1967/1968 fertiggestellt. Es umfaßte damals neben dem Systemkern bereits die meisten der heute gängigen Subsysteme. Der größte Anteil des Arbeitsaufwandes war nicht in den Systemkern, sondern in die Subsysteme, und hier wiederum vor allem in STRUDL, eingeflossen. Zum damaligen Zeitpunkt wurden Softwareprodukte noch nicht, wie es heute üblich ist, als Handelsware angesehen. ICES war jedem Interessenten kostenlos zugänglich. Die Programme und ihre Implementierungsvorschriften wurden von IBM als sogenannte "Class 4"-Programme vertrieben, was im wesentlichen bedeutete, daß die Programme von IBM zwar vertrieben, aber keine Garantien für Richtigkeit der Programme und ihre Verträglichkeit mit dem Betriebssystem übernommen wurde.

In den folgenden Jahren wurde ICES durch drei Entwicklungen beeinflußt: 1) durch das rasche Anwachsen seiner Benutzerzahl, 2) durch den Trend zur kommerziellen Ausnutzung von Software, 3) durch die Tatsache, daß das MIT als Organisation sich nur noch auf die Weiterentwicklung von STRUDL konzentriert. Einige Mitarbeiter, die an der ICES-Entwicklung erheblichen Anteil hatten, gründeten private Firmen, in denen sie einzelne Subsysteme oder auch den Systemkern weiter verbesserten, um sowohl diese verbesserten Versionen wie auch ihre persönliche Erfahrung im Umgang mit ICES wirtschaftlich zu nutzen.
Andere schlossen sich bestehenden Firmen an, die ICES und seine Fähigkeiten als Softwaredienstleistung anbieten. In diesem Zusammenhang wurden ICES und einige Subsysteme beispielsweise von RCA auf ihre Anlagen für Batchbetrieb und interaktiven Betrieb umgeschrieben [5]. Besonders die McDonnell Automation Company [6] bietet einen sehr umfangreichen ICES-Service gegen Entgelt an. Andererseits wird seit Einführung des "Unbundling"-Prinzips ICES nicht mehr von IBM verwaltet und vertrieben. Es ist trotzdem auch heute noch praktisch kostenlos erhältlich. Die ICES-Users Group, eine Vereinigung von ICES-Benutzern, die sich bald nach Einführung von ICES zusammenfand und die seit Mitte 71 die Form einer Corporation hat, vertreibt gegen geringes Entgelt das am MIT entwickelte ICES. Diese ICES-Users Group veranstaltet halbjährlich Informationsveranstaltungen, auf denen beispielsweise über Neuentwicklungen, Verbesserungen und Anwendungen von Subsystemen berichtet wird, und vertreibt etwa vierteljährlich die ICES-Newsletters (ab Frühjahr 1972 umbenannt in ICES-Journal). Die USA stellen (Ende 1971) mit 251 den größten Anteil der 400 Mitglieder der ICES-Users Group [7]. In Europa hat die Vereinigung mit 53 etwa ebenso viele Mitglieder wie in Kanada. (Siehe Tab.1). Diese Tabelle enthält 4 Mitglieder aus der Bundesrepublik Deutschland. Benutzer im deutschsprachigen Raum begannen Anfang 72 mit einem regelmäßigen Erfahrungsaustausch.
In neuerer Zeit wurde von der Pennsylvania State University ebenfalls für die Systeme IBM 360 und 370 ein ICES-Systemkern entwickelt, der mit den bisherigen Subsystemen kompatibel ist [8,9,10]. Dieses PSU-ICES unterscheidet sich von dem MIT-ICES durch größere Effektivität und Flexibilität und größere Unabhängigkeit von der Betriebssystemversion. Das PSU-ICES wird wahrscheinlich auch auf kommerzieller Basis vertrieben werden; die Modalitäten lagen jedoch zum Zeitpunkt dieses Berichtes noch nicht fest.

Darüber hinaus ist ICES Vorbild und Studienobjekt für ähnliche Vorhaben zur Bereitstellung von integrierten EDV-Systemen für die Konstruktionsautomatisierung geworden. Ein solches System wurde in England unter dem Namen GENESYS [11] entwickelt und zeigt sehr enge Verwandtschaft mit ICES. Nicht zuletzt wohl wegen der mit ICES gesammelten Erfahrungen wurde für GENESYS von Anfang an eine Pflege- und Vertriebsorganisation eingerichtet. In der Bundesrepublik arbeiten insbesondere der Lehrstuhl für Theoretische Methoden der Bau= und Verkehrstechnik an der Technischen Universität Berlin (Vorhaben ISB=Informationssystem Bauwesen) und das Institut für Reaktorentwicklung im Kernforschungszentrum Karlsruhe, dem die Autoren dieses Berichtes angehören,(Vorhaben REGENT = Rechnergestützter Entwurf) auf diesem Gebiet. Die jüngste Entwicklung des ICES-Systems ist durch das Bestreben der ICES Users Group gekennzeichnet, das einschränkende Wort "Civil" zu streichen. ICES war vom System-Konzept von Anfang an nicht auf das Bauingenieurwesen eingeschränkt. In zunehmenden Maße werden die existierenden Subsysteme von ICES heute in anderen Fachbereichen (z.B.Reaktortechnik) eingesetzt. Darüber hinaus wurden auch Subsysteme erstellt, deren Anwendungsbereich das Ingenieurwesen allgemein überdeckt, wie z.B. PROJECT und das in Kap.3.4.3 beschriebene GRAPHIC.

2. Erfahrungen mit den ICES-Subsystemen

2.1 Die Subsysteme

Die von der ICES-Users Group-Organisation gegenwärtig verwalteten und vertriebenen Subsysteme sind in Tab.2 zusammengefaßt. Die in der Tabelle geführte Angabe VmMn (z.B.V1M2 bedeutet z.B. Version 1 Modifikation 2) läßt erkennen, die wievielte Fassung des Subsystems vorliegt. Darüberhinaus gibt es die bereits erwähnten Subsysteme, die auf kommerzieller Basis entwickelt und von den Entwicklungsfirmen selbst zeitweise oder auf Dauer vermietet werden. Das bekannteste dieser kommerziellen Subsysteme ist das von Daniels (dem Autor von PROJECT I)erstellte Subsystem PROJECT II [12], auf das weiter unten näher eingegangen wird.

Ein großer Teil der Subsysteme ist nur für einen relativ eng begrenzten und scharf umrissenen Kreis von Anwendern aus der Bau- und Verkehrstechnik von Interesse. Hierzu gehören die Subsysteme

SEPOL (Settlement Problem Oriented Language)
ROADS (Roadway Analysis and Design System)
BRIDGE (Bridge Design System)
TRANSET (Transportation Network Analysis)
LEASE (Limiting Equilibrium Analysis of Slopes and Embankments)
TRAVOL (Traffic Volume Data Subsystem)

Diese Subsysteme wurden bei der GfK nicht implementiert, und es soll über sie daher auch nicht berichtet werden.

Die Subsysteme

TABLE II (Tabellenverwaltung)
OPTECH (Optimierungsverfahren)
COGO (2 bis 2 1/2 dimensionale Geometrieaufgaben)
PROJECT I (Projektplanung und Verfolgung)
STRUDL II (Festigkeitsberechnungen)

sind in ihrem Anwendungsspektrum nicht auf das Bauwesen beschränkt. Probleme, für die diese Subsysteme das geeignete Arbeitsmittel sind, kommen gleichermaßen auch in Maschinenbau, Chemieingenieurwesen, Flugzeugbau und Reaktortechnik vor.

Das Subsystem

CDL (Command Definition Language, zur Definition neuer problemorientierter Sprachen)

ist gegenüber derartigen Fachrichtungsgrenzen indifferent. Dieses Subsystem ist in allen Bereichen anwendbar, in denen die Probleme in Wortsprachen definiert werden können. Hierüber wird in Kap.3.1.6 gesondert berichtet.

2.2 Tabellenverwaltung mit TABLE II

Das Subsystem TABLE II [13] dient zum Verwalten von Tabellen. Die Tabellen haben die Struktur einer zweidimensionalen Matrix, wobei

jedoch ein Matrixelement selbst ein eindimensionaler Vektor beliebiger und unterschiedlicher Länge sein kann, so daß praktisch auch dreidimensionale Tabellen verwaltet werden können. Die Zeilen der Tabelle entsprechen den von der Tabelle verwalteten Objekten, die Spalten bestimmten Eigenschaften. Die Eigenschaften können folgende Darstellung haben:

ganze Zahlen (INTEGER)
reelle Zahl (REAL)
doppelt genaue Zahl (DOUBLE)
Characterstring (ALPHA)

Ein wesentliches Merkmal von TABLE II ist die Möglichkeit, auf Tabellen und ihren Inhalt sowohl mit Hilfe einer besonderen problemorientierten Sprache,wie auch von den Moduln eines Subsystems aus zuzugreifen. Die Art des Zugriffs umfaßt von beiden Seiten her alle Manipulationsmöglichkeiten:

Erzeugung und Erweiterung einer Tabelle in Zeilen und Spalten
Eintragen von Werten
Lesen von Werten
Umsortieren nach verschiedenen Kriterien
Ändern und Löschen

Von Seiten der Problemorientierten Sprache besteht außerdem die Möglichkeit, bei Ein- und Ausgabe der Tabellenwerte die Zahlenangaben mit verschiedenen physikalischen Einheiten (für Länge, Kraft, Winkel, Temperatur und Zeit) zu versehen, während die Tabellen intern jeweils in einer Standardeinheit gespeichert werden (inch, pound, Bogenmaß, Fahrenheit und Sekunde).

Der Nutzen einer derartigen von den einzelnen Subsystemen getrennten Tabellenverwaltung ist offensichtlich. Sie gestattet eine langfristige Verfügbarkeit von routinemäßig häufig benötigten Daten (z.B.Werkstoffdaten, Gestaltsangaben für Standardbauteile) und erspart die wiederholte Eingabe dieser Daten mit dem damit verbundenen Aufwand und Fehlerrisiko. Ferner werden durch die Einheitlichkeit der Eingabe der Tabellenwerte, die Änderungsmöglichkeiten und die verschiedenen Ausgabemöglichkeiten weitere Fehlerquellen eliminiert.

Um von der zentralen Tabellenverwaltung Gebrauch machen zu können, muß ein Subsystem sich der Konvention unterwerfen, in seinem Common-

bereich 15 Positionen für TABLE II freizuhalten. Dies erschwert den nachträglichen Einbau der Tabellenverwaltung in ein fertiges oder in der Entwicklung befindliches Subsystem. Bei einer Neuplanung bedeutet dies keine Belastung.
Das Subsystem STRUDL II benutzt heute noch TABLE I [14], den Vorläufer von TABLE II, für seine Tabellenverwaltung. Da beide Entwicklungsstufen Programme gleichen Namens aber unterschiedlicher Funktion enthalten, müssen bei einer Installation, die sowohl STRUDL II mit TABLE I wie auch TABLE II bereitstellen soll, diese Subsysteme in verschiedenen Dateien gespeichert werden.

2.3 Lösung von Optimierungsproblemen mit OPTECH

Das Subsystem OPTECH I [15] enthält Lösungsmethoden für Optimierungsprobleme folgender Klassen:

a) Probleme der linearen Programmierung:
Methode von R.Clasen [16]

b) Netzwerkproblem: Methode von R.Clasen [17]

c) Probleme der nichtlinearen Optimierung:
Methode von Flood und Leon [18]

d) Probleme der ganzzahligen Optimierung:
Methode von Gonzalez-Young [19]

Während Probleme der Klasse a, b und d vollständig durch die problemorientierte OPTECH-Sprache zu beschreiben sind, wurde für die Klasse c ein Weg gewählt, der sich auch bei anderen Programmsystemen, die Aufgaben großer Komplexität bewältigen sollen, bewährt hat [20]. In diesem Fall ist die Zielfunktion selbst als eine Subroutine in der Basissprache (ICETRAN) zu formulieren.

Praktische Erfahrungen mit OPTECH liegen bei der GfK nicht vor. Die in OPTECH realisierten Methoden stammen aus den Jahren 1961 bis 1965. Neuere Verfahren, wie z.B. die Evolutionsstrategie nach Schwefel [21] sind darin leider nicht enthalten.

2.4 Geometrie- und Anordnungsaufgaben für COGO

Das Subsystem COGO I [22] dient zur Lösung von Geometrie- und Anordnungsproblemen in zwei und drei Dimensionen. Es ist besonders geeignet für Aufgaben aus dem Vermessungswesen. Auch im Bereich der Grundrißplanung oder beispielsweise für Verlegungspläne von Rohrleitungen und anderen Netzwerken kann COGO eingesetzt werden. Für Probleme in zwei Dimensionen stellt die COGO-Sprache eine große Vielfalt von Formulierungsmöglichkeiten zur Verfügung. Die dritte Dimension kann jedoch nur durch die Angabe eines Neigungswinkels (von Linien oder Flächen) gegenüber der Grundfläche angesprochen werden, wodurch die Anwendbarkeit auf echt dreidimensionale Aufgaben stark eingeschränkt ist.

COGO erlaubt zwar die Definition von geometrischen Gegebenheiten und ihre rechnerische Analyse (z.B.Flächenberechnung), jedoch in seiner ursprünglichen Form keine Ausgabe von Zeichnungen. Diesbezügliche Erweiterungen werden erst seit kurzem bearbeitet [23].

COGO ist nach STRUDL das am häufigsten benutzte Subsystem. Es wird in den USA insbesondere bei der Planung von Straßen verwendet.
Bei der GfK wurde es bisher jedoch nicht eingesetzt.

2.5 Planung und Verfolgung von Projekten mit PROJECT

Das Subsystem PROJECT I [24, 25] erlaubt die terminliche und kostenmäßige Planung und Verfolgung von Projekten mit den Methoden der Netzplantechnik. Ferner erlaubt PROJECT I die Zuweisung von Kapazitäten zu einzelnen Aktivitäten und die Abschätzung von Kostenanteilen aufgrund der spezifischen Kapazitätskosten (mit einer Einschränkung auf die unten eingegangen wird).

Für die Darstellung des Netzplanes werden in üblicher Weise die Knotenpunkte des Graphen numeriert. PROJECT I gestattet es, die Einzelaktivitäten entweder den Verbindungslinien der Knoten oder den Knoten selbst zuzuordnen, wobei jedoch die letztere Möglichkeit zu bevorzugen ist.

Die Aktivitäten können außer mit Zeit- und Kostenschätzungen zusätzlich mit einer 8-stelligen Codenummer versehen werden, deren Stellen vom Benutzer mit bestimmten Bedeutungen belegt (z.B.ausführende Firma) und für Sortierzwecke benutzt werden können. Für die Eingabe, Erweiterung und Änderung der von PROJECT I langfristig gespeicherten Netzpläne steht eine einprägsame und vielseitige Kommandosprache zur Verfügung.
Für besonders häufig vorkommende Aktivitätsfolgen (z.B.Angebotseinholung, Angebotsauswertung, Lieferantenauswahl, Auftragserteilung, Lieferung, Prüfung, Annahme) sind besondere Kurzformen der Eingabe möglich. Selbstverständlich ist die Angabe von Feiertagen oder der Zahl der Arbeitstage pro Woche sowohl für alle Projekte, wie auch für jedes Projekt individuell möglich.

Die Projektverfolgung geschieht durch Angabe der tatsächlichen Start- und Endtermine und der echten Kosten der einzelnen Aktivitäten. Diese Meldungen werden für die weitere Planung von PROJECT I berücksichtigt. Die Ausgabe der Projektinformationen in tabellarischer oder grafischer Form ist mit Hilfe der sehr gut durchdachten Befehlsform in außerordentlich einfacher und flexibler Weise zu sortieren und auszuwählen.

Die Kapazitätszuweisung an Projekte enthält einen Mangel, der ihre Anwendung nahezu verbietet: Kapazitätsbeschreibungen können nicht verändert werden (z.B.Kosten pro Einheit) und bei einem Projekt, dem Kapazitäten zugewiesen wurden, können weder Aktivitäten entfernt, noch neue hinzugefügt werden. Speziell bei der terminlichen Planung und Verfolgung von Projekten mit über 100 Aktivitäten wurden bei der GfK im routinemäßigen Einsatz von PROJECT I sehr gute Erfahrungen gemacht.

Auf kommerzieller Basis wurde aus PROJECT I heraus das System PROJECT II entwickelt [12]. PROJECT II enthält die genannten Einschränkungen von PROJECT I nicht mehr. Es bietet darüber hinaus viele wünschenswerte Möglichkeiten wie beispielsweise die automatische Optimierung der Planung mehrerer Projekte unter Berücksichtigung von Kapazitätsrestriktionen. PROJECT II dürfte von den Fähigkeiten und vom Bedienungs-

komfort her eines der besten Netzplantechnikprogramme sein, die heute existieren. Seiner allgemeinen Verbreitung steht sicher aber der sehr beachtliche Preis entgegen.

2.6 Festigkeitsanalyse mit STRUDL II

STRUDL II ist mit weitem Vorsprung dasjenige Subsystem, das am häufigsten eingesetzt wird und in das vom MIT der größte Entwicklungsaufwand eingeflossen ist. Dies drückt sich auch darin aus, daß die STRUDL II-Handbücher als bisher einzige im Juni 1971 in einer völlig neu überarbeiteten zweiten Auflage erschienen sind [26,27,28]
Ein Vorläufer von STRUDL II war das nur für Stabwerke geeignete Subsystem STRUDL I [29], in welches viele Elemente des von ICES unabhängigen Systems STRESS [30] eingeflossen sind.

Es ist unmöglich, hier auch nur annähernd das Spektrum der Möglichkeiten von STRUDL II zu umreißen. Die Strukturen, die in STRUDL II darstellbar sind, können sich zusammensetzen aus statisch bestimmten oder statisch unbestimmten ebenen oder räumlichen Stabwerken, aus Platten, Schalen oder räumlichen Gebilden, die sich aus 19 ebenen und 3 räumlichen Typen finiter Elemente zusammensetzen lassen, sowie aus beliebigen Kombinationen dieser Möglichkeiten (soweit sie physikalisch sinnvoll sind). Die meisten Typen der finiten Elemente können aus anisotropem Material bestehen. Ferner sind Elemente mit nichtlinearem Verhalten spezifizierbar, und es ist dem Anwender die Möglichkeit gegeben, Elemente eigener Wahl zu definieren und einzubeziehen. Eine derart zusammengesetzte Struktur kann folgenden Untersuchungen unterzogen werden:

- Berechnung der Spannungen, Dehnungen, Verschiebungen und Kräfte aufgrund einfacher oder kombinierter Belastungen durch äußere oder innere Kräfte und Temperaturdehnung.

- Kostenoptimierung von Stabwerkkonstruktionen durch Auswahl geeigneter Profile unter Einhaltung einer Reihe von Restriktionen (z.B. zulässige Spannung, möglichst große Zahl gleicher Profile)

- Berechnung der Eigenfrequenzen und Eigenschwingungsformen der Struktur.

- Berechnung des zeitlichen Reaktionsverhaltens der Struktur auf eine zeitabhängige Belastung im Zeit= oder Frequenzbereich.

- Berechnung des Kriech- oder Beulfaktors für die einzelnen Belastungsformen (d.h.desjenigen Faktors, mit dem einer der vorgegebenen Lastfälle multipliziert werden muß, bis die Struktur instabil wird).

Neben diesen Möglichkeiten können mit STRUDL-II auch Spannbetonkonstruktionen nach amerikanischen Normen berechnet werden.
STRUDL-II ist zu den besten Programmen seines Bereiches zu rechnen. Auch für umfangreiche Probleme wie Hochhäuser mit über 50 Stockwerken, für Raumkapselberechnungen und Berechnungen komplizierter Behälter wird es routinemäßig eingesetzt [31].

2.7 Gesichtspunkte zur Beurteilung der Subsysteme

Für einen potentiellen Anwender eines Programms, eines Programmsystems gibt es eine Reihe unterschiedlicher Gesichtspunkte, nach denen er entscheiden muß, ob er dieses Programm bzw. Programmsystem einsetzen soll. Die wesentlichsten Gesichtspunkte sind folgende:

Dokumentation der Programmfähigkeiten
Qualität der im Programm enthaltenen Lösungsmethoden
Effektivität des Programmcodes
Flexibilität der Problemformulierung
Erlernbarkeit der Handhabung
Korrektheit des Programmes
Modifizierbarkeit
Verständlichkeit der Programmnachrichten
Programmpflege
Popularität
Transportierbarkeit
Kosten

Die Reihenfolge soll hier keine Wertung darstellen.

ICES und seine Subsysteme sollen nun unter diesen Gesichtspunkten anhand der in der GfK gesammelten Erfahrungen beurteilt werden. Der Gesichtspunkt

Erweiterbarkeit

interessiert nur diejenigen, die eigene Programmentwicklung betreiben und wird daher gesondert in Kapitel 3.1 behandelt.

Die Dokumentation der Programmfähigkeiten ist sehr gut. Die Subsystem-Handbücher enthalten meist eine allgemein gehaltene Einführung, die Erläuterung der für alle Subsystemhandbücher fast gleichen Darstellung der Syntax der Bedienungssprache, eine durch übersichtliche Kapitel gegliederte Beschreibung der Syntax der Befehle und ihrer Wirkung mit Angabe von Beispielen und schließlich eine Zusammenfassung der Fehlermeldungen. Erfahrungsgemäß bedeutet dies jedoch nicht, daß man nach eingehendem Studium der Handbücher umfangreiche Probleme bereits lösen kann. Wie bei jeder anderen Programmiersprache ist zunächst eine Übungsphase mit Primitivbeispielen erforderlich, bevor der Benutzer die Subsystemfähigkeiten richtig kennt und effektiv einsetzen kann.

Über die Qualität der in den Subsystemen realisierten Lösungsmethoden muß sich der Spezialist des jeweiligen Fachgebietes jeweils selbst anhand der Dokumentation und anhand seines konkreten Problems ein Urteil verschaffen. Es ist leicht verständlich, daß in einem Subsystem, das zur Lösung von Aufgaben aus einem breiten Spektrum herangezogen werden kann, die Lösungsmethoden mehr unter dem Gesichtspunkt der allgemeinen Verwendbarkeit und der Sicherheit auch in Sonderfällen, als unter dem Aspekt der Effizienz zu sehen sind. Dennoch sind die in den wichtigsten Subsystemen (STRUDL, PROJECT, COGO, CDL) realisierten Methoden nach dem Urteil der Autoren dieses Berichtes auch bezüglich der Effizienz als sehr gut zu bezeichnen.

Die Effektivität des Programmcodes wird durch drei Merkmale von ICES negativ beeinflußt. Das erste Merkmal stammt aus der historischen Entwicklung von ICES. Der Systemkern wurde zu einem Zeitpunkt geplant und erstellt, als es für das Rechnersystem IBM/360 lediglich den Fortran-E-Compiler gab. Man muß es als einen schweren Planungsfehler

von ICES ansehen, daß der Systemkern von bestimmten Eigenschaften des von diesem Compiler erzeugten Objektcodes abhängt, so daß es nicht möglich ist, anstelle dieses Compilers die heute üblichen IBM-Fortran-Compiler (G oder H) einzusetzen, die einen schnelleren Objektcode erzeugen. Dieser Konzeptfehler ist in einer von der Pennsylvania State University entwickelten Version des ICES-Kerns inzwischen behoben.(Die Bedingungen, unter denen dieses sogenannte PSU-ICES erhältlich sein wird, sind heute noch ungeklärt.). Die zweite Effektivitätseinbuße ergibt sich aus der Tatsache, daß für die dynamischen Felder (DYNAMIC ARRAY, s.Kapitel 3.1.3) vom ICES-Systemkern ein virtueller Kernspeicher simuliert wird und daß diese dynamischen Felder auch dann, wenn sie die Form einer mehrdimensionalen Matrix haben, als über Zeiger (POINTER) verknüpfte Vektoren dargestellt werden. Dies führt dazu, daß jeder lesende oder schreibende Zugriff auf ein Element eines dynamischen Feldes eine recht langwierige Prozedur des Auffindens der momentanen Position dieses Elementes auslöst, die nur vom Programmierer durch sorgfältig geplante Maßnahmen abgekürzt werden kann. Hier könnte für aufeinanderfolgende Zugriffe auf benachbarte Elemente eine effektivere Strategie eingesetzt werden. Die software-mäßige Simulation eines virtuellen Kernspeichers muß jedoch stets einen erhöhten Aufwand erfordern gegenüber einer Anlage, die von der Hardware her diese Möglichkeit unterstützt. Der dritte nachteilige Effekt rührt daher, daß der Code eines in den Kernspeicher geladenen Programmes nicht mehr verschoben werden darf. (Siehe hierzu auch Kap.3.1.4). Dies führt dazu, daß nach intensiver Untersuchung zur Kernspeicherverwaltung in ICES [33] schließlich eine Strategie gewählt werden mußte, die oft dazu führt, daß Programmoduln erneut von externen Einheiten in den Kernspeicher geladen werden müssen. Eine Reihe von Arbeiten, die zu diesem Problem der Code-Effektivität durchgeführt wurden, hat gezeigt, daß es außerordentlich schwer ist, einen Vergleich zwischen einem ICES-Subsystem und einem "gleichwertigen FORTRAN-Programm" zu ziehen. Nach Erfahrungen der Autoren dieses Berichtes ist ein ICETRAN-Programm, das die in ICES über FORTRAN hinaus gebotenen Möglichkeiten nicht ausnutzt, etwa 3 bis 10 mal langsamer als ein entsprechendes Fortranprogramm, das mit einem optimierenden Compiler erzeugt wurde. Bei einer Anlage mit größerem Kernspeicher

ist selbstverständlich die Effektivität besser als bei einem kleinen Kernspeicher, bei dem die Daten häufiger auf externe Einheiten (Platte oder Trommel) ausgelagert werden müssen. Dieser Effektivitätsgewinn muß jedoch bei der Implementierung eines Subsystems auf einer größeren Anlage durch einen (einmaligen) Benutzereingriff vorbereitet werden. Es muß an dieser Stelle betont werden, daß ein großer Teil der Effektivitätseinbußen gegenüber einem "entsprechenden" Fortranprogramm als angemessener Preis dafür angesehen werden muß, daß ICES ein Datenmanagement bereitstellt, das es überhaupt erst ermöglicht, mit den Subsystemen auch solche Probleme zu bearbeiten, die sonst an der begrenzten Kernspeicherkapazität scheitern. Zudem wird dieses Datenmanagement vom Systemkern übernommen, braucht also nicht für jedes Subsystem neu programmiert zu werden.

Bezüglich der Flexibilität der Problemformulierung haben sich die Ersteller aller Subsysteme große Mühe gegeben. Es ist in fast allen Fällen möglich, das zu untersuchende Modell schrittweise aufzubauen, zu korrigieren, die Modellformulierung langfristig zu unterbrechen und später wieder aufzunehmen, wobei ein Protokoll der bisher eingegebenen Daten angefordert werden kann. Besonders hilfreich ist dabei, daß gewöhnlich die Befehle zur Problembeschreibung, zur Durchführung der Berechnungen und zur tabellarischen oder grafischen Darstellung der Ergebnisse sauber voneinander getrennt sind. Für den geübten Subsystembenutzer gibt es darüber hinaus eine Vielzahl von Möglichkeiten, die Befehlseingabe bis auf einen letzten signifikanten Rest abzukürzen. Als Beispiel für die Flexibilität seien hier einige der Möglichkeiten einer Terminangabe, wie sie in PROJECT verwendet werden kann, wiedergegeben.(Man beachte die amerikanische Konvention, den Monat vor den Tag zu schreiben):

9 15 67
9 15 1967
9, 15, 67
SEP 15 1967
15 SEPTEMBER 1967

Alle Formulierungen sind gleichwertig. Dagegen ist 15.9.1967 unzulässig.

Trotz der großen Flexibilität der problemorientierten ICES-Subsystemsprachen, die sich aus den besonderen Fähigkeiten des Subsystems CDL ergeben (und auf die noch gesondert eingegangen wird), lassen alle diese Sprachen einige Fähigkeiten vermissen, die die Formulierung umfangreicher Probleme erleichtern und Handrechnungen erübrigen würden. Eine derart mächtige problemorientierte Sprache wie z.B. die für STRUDL sollte wenigstens

einfache Zahlenarithmetik (+,-,*,/,**),
Verzweigungen, Schleifen (DO....WHILE...) sowie
die Möglichkeit der Definition von Makros bieten.

Besonders die ersten der genannten Mängel vermißt der mit höheren Programmiersprachen vertraute Benutzer.

Die Erlernbarkeit der Sprachen wird durch ihre Flexibilität und ihre gute Dokumentation gefördert. Diese Faktoren werden stark dadurch unterstützt, daß ein Neuling, der ja zunächst immer Beispiele studieren und nachvollziehen wird, die Eingabe für andere Probleme wegen der Problemorientiertheit der Sprache intuitiv richtig versteht. Hierzu dienen natürlich die Beispiele in den Handbüchern, besser jedoch ist es, wenn in der Umgebung des Lernenden das Subsystem bereits eingesetzt wird. Hierauf wird unter dem Stichwort Popularität noch einzugehen sein. Nach eigenen Erfahrungen kann selbst ein im Programmieren Ungeübter nach wenigen Tagen mit einem Subsystem selbständig umgehen. Um die Fähigkeiten von STRUDL-II effektiv einsetzen zu können, braucht man allerdings einige Monate.

Unter Korrektheit des Programms wollen wir verstehen, daß das Verhalten des Programmes genau seiner Beschreibung entspricht. Bei derart vielseitigen Programmen, wie den ICES-Subsystemen ist es nicht verwunderlich, aber leider doch oft störend, daß sie in diesem Sinne nicht absolut korrekt sind. Besonders nachteilig ist dies für Gruppen, die noch keine Erfahrung im Umgang mit ICES haben. Es ist häufig so, daß bestimmte Fehler von einzelnen Anwendern festgestellt und Mittel zu ihrer Umgehung oder Behebung gefunden wurden, ohne daß andere Anwender davon erfahren. Die ICES-Newsletters, das vierteljährlich erscheinende Kommunikationsorgan der ICES-Users-Group, enthalten regelmäßig Meldungen

über neu entdeckte Fehler und meist auch Anweisungen zu ihrer Behebung. Für einen völligen ICES-Neuling ist es jedoch oft schwer bis unmöglich, genau den richtigen Hinweis zu finden; normalerweise wird er - oft zu Recht -den Fehler lange bei sich suchen, bevor er andere ICES-Benutzer um Rat angeht. Es macht sich hier nachteilig bemerkbar, daß die ICES-Users-Group nur eine recht zwanglose Interessengemeinschaft ist und keine straff organisierte Pflege- und Vertriebsstelle von ICES.

Eine Erfahrung, die man stets bei der Übernahme fertiger Programmsysteme machen wird, hat sich auch bei den ICES-Subsystemen bestätigt. Übernommene Programmsysteme erfüllen die Wünsche des neuen Anwenders nie vollkommen. Oft würden schon geringe Änderungen oder Erweiterungen das Programm für den neuen Einsatzbereich entscheidend verbessern. Die Notwendigkeit für die Modifizierbarkeit der Subsysteme wurde jedoch erst spät erkannt. Lediglich die Neuauflagen der STRUDL-II-Handbücher enthalten genaue Vorschriften darüber, wie ein Benutzer in dieses Subsystem neue Typen finiter Elemente implementieren kann. In allen anderen Fällen müßte sich der an einer Modifikation interessierte Benutzer die nötige Kenntnis über Datenstruktur und Programmablauf des Subsystems mühsam aus Quellprogrammlisten erarbeiten. Eine Quellprogrammbeschreibung, wie sie für den Systemkern existiert [34], liegt für die Subsysteme im allgemeinen nicht vor.
Gemessen an dem, was Programme normalerweise bei Bedienungsfehlern demjenigen mitteilen, der sie nicht selbst geschrieben hat, ja selbst im Vergleich zu den Diagnosemeldungen von Compilern ist die Verständlichkeit der Subsystemnachrichten bemerkenswert gut. Dies beginnt damit, daß der Befehlsinterpretierer jeden Befehl zunächst protokolliert bevor er eine Ausführung veranlaßt und reicht bis zu einer sehr vollständigen Liste der (nummerierten) Fehlernachrichten der Subsysteme in dem jeweiligen Handbuch. Dennoch gibt es Meldungen wie etwa STATICS FAILED in STRUDL II, die ein Anfänger unmöglich verstehen kann. Er muß dann seine Eingabe kritisch überprüfen (das kann er meist nicht), probieren oder andere Subsystem-Kenner fragen. Darüber hinaus gibt es eine Klasse von unangenehmen, da meist unverständlichen Nachrichten, die nicht aus dem Subsystem, sondern aus dem Systemkern stammen. Bei der Anwendung der fertigen Subsysteme treten diese Nachrichten relativ selten auf und dann oft nur als Folge von Fehlern, die die Subsysteme

schon in verständlicher Form gemeldet haben. Bei der Erstellung neuer Subsysteme jedoch bereiten diese Nachrichten nicht selten wochenlanges Kopfzerbrechen, bis ihre Ursache verstanden ist.

Wie jedes andere Programm so bedarf auch ICES bzw. ein ICES-Subsystem der Programmpflege. Es ist auf den vorangegangenen Seiten schon einige Male darauf hingewiesen worden, daß unerwartete Programmreaktionen nicht immer auf Benutzungsfehler, sondern gelegentlich auch auf Programmfehler oder auch auf Unterschiede im Betriebssystem zurückzuführen sind. Im Gegensatz zu einem Programm, das von einer Softwarefirma gekauft wurde und für das dann der Lieferant meist auch die Verpflichtung zur Behebung später aufgedeckter Mängel übernimmt, ist bei ICES keine derartige Unterstützung sichergestellt. Die ICES-Newsletters enthalten in jeder Ausgabe eine Reihe von Mitteilungen von ICES-Benutzern über entdeckte Fehler und meist (nicht immer) auch einen Hinweis zur Behebung. Doch dies ist ein Notbehelf. Jede Stelle, die ICES benutzt, ist darauf angewiesen, sich auch mit den Interna des ICES-Kerns und seinen Beziehungen zum Betriebssystem in beschränktem Maße vertraut zu machen, - es sei denn, es gibt in der Nachbarschaft eine andere Stelle, bei der diese Kenntnis bereits vorliegt und die um Rat befragt werden kann. Letzteres ist natürlich in den USA - wenigstens gegenwärtig noch - leichter als in Europa.

Die Popularität eines Programms oder einer Programmiersprache ist ein schwer faßbarer Faktor, der aber für den praktischen Einsatz von außerordentlicher Bedeutung ist. So ist beispielsweise die Popularität von FORTRAN sicher der größte Vorteil dieser Programmiersprache, denn sie bewirkt, daß jeder, der damit Schwierigkeiten hat, in seiner unmittelbaren Umgebung Leute findet, die ihm helfen können. Gerade dies ist aber bei ICES nicht der Fall (oder wenigstens bisher in Europa nicht der Fall). Bei den oben vermerkten Punkten der Kritik wurde immer wieder betont, daß sie ganz besonders hart den treffen, der sich neu in ICES und seine Subsysteme einarbeitet. Angesichts dieser Erschwernis ist es eigentlich verwunderlich, daß ICES eine derart weite Verbreitung gefunden hat und muß als ein Indiz für die hervorragende Qualität der Subsysteme gewertet werden. In den USA wurde die Verbreitung von ICES dadurch gefördert, daß oft Angehörige des MIT, die mit ICES gearbeitet hatten, wenn sie in eine andere Firma eintraten, das System

dort einführten. Aber auch in der Bundesrepublik steigt die Zahl der ICES-Benutzer stetig an: Ende 1971 waren vier deutsche Firmen Mitglieder der ICES-Users-Group; die Zahl der Firmen und Behörden, die ICES benutzen oder in unmittelbarer Zukunft benutzen wollen, ist jedoch größer. Ein regelmäßiger Erfahrungsaustausch zwischen diesen Stellen ist eingeleitet.

Die Transportierbarkeit von ICES ist begrenzt auf IBM Anlagen der Familien/360 und /370 vom System IBM/360-40 mit 128 K Bytes Kernspeicher und zwei 2311-Platten an aufwärts. Dies liegt daran, daß der Systemkern zum Teil in Assembler geschrieben ist und auf bestimmte Fähigkeiten und Konventionen des Betriebssystems Bezug nimmt. Der Systemkern und einige Subsysteme (z.B.COGO I und STRUDL I, der Vorläufer von STRUDL II) wurde auch für RCA-Maschinen umgeschrieben. Den Autoren dieses Berichtes ist jedoch über den Einsatz dieser Version nichts bekannt. Aber auch innerhalb der IBM/360 und IBM/370-Familien treten mit erstaunlicher Regelmäßigkeit bei Betriebssystemänderungen Schwierigkeiten auf, die jedoch bisher stets überwunden werden konnten. Die Autoren dieses Berichtes haben das ursprünglich unter dem Betriebssystem OS/360-Release 14 entwickelte ICES bei Release 16 implementiert und betreiben es jetzt unter der ASP-Version von Release 20. Ferner ist bekannt, daß ICES unter dem Betriebssystem TSS auf IBM/360-67 (nach kleinen Änderungen) läuft [35] wie auch unter der OS-Version TSO [36]. Bisher waren die Release-bedingten Umstellungen an ICES im allgemeinen nicht schwerwiegender, als sie auch gelegentlich bei ganz normalen FORTRAN-Programmen sind. Es besteht natürlich die Möglichkeit, daß kommende Betriebssystemänderungen einen drastischen Zusammenbruch bewirken können. Die Gemeinde der ICES-Benutzer ist inzwischen aber derart angewachsen, daß man zuverlässig damit rechnen kann, daß auch in diesem Fall wieder Abhilfe geschaffen wird.

Vom Gesichtspunkt der Kosten her sind die von der ICES-Users-Group vertriebenen ICES-Subsysteme außergewöhnlich günstig, da ICES praktisch kostenlos verfügbar ist. Die geringen Kosten für Kopieren und Transport der Magnetbänder und für die Handbücher sind vernachlässigbar.

Auch der Jahresbeitrag für die ICES-Users-Group, der die Kosten für die ICES-Newsletters beinhaltet und damit die Information über Korrekturen, Änderungen oder Fehlerquellen gewährleistet, fällt angesichts der Wichtigkeit dieser Information nicht ins Gewicht. Der größte Kostenfaktor für einen europäischen Benutzer dürfte wohl alle zwei Jahre eine Reise zu einer der halbjährlich in den USA stattfindenden Tagungen der ICES-Users-Group sein. Dieser günstige Kostenaspekt dürfte die weitere Verbreitung von ICES in den nächsten Jahren stark fördern. Einer besonderen Prüfung bedürfen im Einzelfall die Angebote der verschiedenen Firmen, die ICES auf kommerzieller Basis vertreiben. Den hierfür einzusetzenden oft beachtlichen Kosten, steht als Gewinn eine verbesserte Qualität der Subsysteme (z.B.Plot-Ausgabe von STRUDL-II oder die Fähigkeiten von PROJECT-II) und die garantierte Pflege und Unterstützung bei Inbetriebnahme gegenüber.

3. ICES als Programmsystem

3.1 Besonderheiten von ICES

3.1.1 System

ICES ist nicht nur ein nur für einen bestimmten Anwendungsbereich geeignetes Programm, sondern ein echtes Programmsystem:

- ICES stellt alle Komponenten bereit, die für die Lösung eines Problems auf einer Rechenanlage erforderlich sind: Programmiersprachen mit Übersetzern und Interpretierern, Programm- und Datenmanagement, Fehlerüberwachung (Interrupthandling) sowie für bestimmte Bereiche auch fertige Problemlösepakete (Subsysteme, wie in Kap.2 beschrieben).

- Es besteht eine klare Trennung zwischen (vergl.Abb.1):
  a) Definition der ausführenden Programme mit der Programmiersprache "ICETRAN" (siehe Kap.3.1.3)
  b) Zusammenstellung von Programmen zu den vom Betriebssystem der Rechenanlage für die Ausführung in den Kernspeicher ladbaren "Lade-Modulen" (siehe Anhang 1); die Form der Eingabe dieser

Definition ist in Kap.3.1.4 beschrieben.

c) Definition der "problemorientierten Sprache" (POL) mit einem speziellen Subsystem CDL (siehe Kap.3.1.2).

d) Ausführung eines Subsystems durch Interpretation der vom Benutzer in der POL definierten Kommandos.

- ICES ist stark modular aufgebaut und daher leicht erweiterbar. Hierzu dient die Aufteilung in einen ICES-Kern ("Basic-System"), der die für jede Programmausführung benötigten Funktionen bereitstellt, und "Subsysteme" mit Problemlösepaketen für spezielle Anwendungsbereiche. Jedes Subsystem kann unabhängig von anderen erstellt werden, wenngleich einige Subsysteme (wie CDL und TABLE) als Dienstsysteme für andere Subsysteme eingesetzt werden können. Da die meisten Subsysteme voneinander unabhängig sind, ist im allgemeinen kein"Interface" [46] erforderlich, über das die für zwei Subsysteme benötigten Daten übermittelt werden. (Das Interface beliebiger Subsysteme zu dem Subsystem CDL bildet eine spezielle Datei. (ICESDICT, siehe Kap.3.2). Das Interface zu TABLE wird durch den Einbau spezieller TABLE-Routinen in das jeweilige Subsystem erzeugt).

- Auch die ICES-Subsysteme selbst lassen sich bei entsprechender Planung erweitern, ohne daß das Bestehende geändert werden muß. Dies beruht darauf, daß auch die Subsysteme modular aufgebaut sind. Ein Subsystem besteht aus Programm-Moduln sowie aus Sprachblöcken, die alle einzeln definiert, verändert und in das Subsystem eingebracht werden können. Die Programm-Moduln sind Lade-Moduln, wie in Kap.3.1.4 näher erläutert wird. Die Sprachblöcke sind Definitionen einzelner Kommandos der POL, die mit dem Subsystem CDL erzeugt werden (siehe Kap.3.1.2). Das Interface zwischen den Programm-Moduln und den Sprachblöcken bildet ein spezieller Datenbereich "COMMON" im Kernspeicher, der wie der FORTRAN Common programmiert wird, in seiner Funktion sich hiervon jedoch unterscheidet (siehe Kap.3.1.4 und 3.1.5).

3.1.2 Problemorientierte Sprache

Zu jedem Subsystem gehört in ICES eine spezielle Eingabesprache, in der der Benutzer dem Subsystem sein Problem schildert und den Rechenauftrag

definiert. Man nennt derartige Eingabesprachen "Problemorientierte Sprachen" (Problem oriented language, POL), da sie einem bestimmten Problembereich bezüglich Wortschatz und Befehlsstruktur angepaßt sind. Eine andere Bezeichnung wäre "Benutzerorientierte Sprache", wobei hier zum Ausdruck kommt, daß die Sprache es dem mit dem Problem vertrauten Benutzer besonders leicht macht, seinen Auftrag an das Subsystem zu formulieren. Eine POL muß daher zwei Eigenschaften erfüllen:

- die Sprache soll möglichst weitgehend der Struktur der natürlichen Sprache des Benutzers (Deutsche, Englische oder andere) entsprechen.

- die verwendeten Terme (Schlüsselwörter, Dimension der Daten usw.) müssen dem Fachbereich entstammen, der dem Benutzer vertraut ist, dagegen aber frei sein von Begriffen und Anweisungen, die sich auf die EDV-Anlage oder die Programmiertechnik beziehen.

Die Verwirklichung dieser Forderungen schafft folgende Vorteile:

- Die Eingabe wird einerseits lesbar und dadurch selbstdokumentierend und andererseits leicht erlernbar.

- Die zahlreichen Fehlerquellen, die bei der Verwendung starrer Formate auftreten, werden vermieden und die Programmanwendung somit zuverlässiger.

- Die Sprache wird flexibel; es ist häufig nicht erforderlich die Eingabedaten in einer starren Reihenfolge anzugeben; für manche Eingabedaten können Standardwerte oder für fehlende Anweisungen Standardreaktionen vorgesehen werden.

- Auch solche Ingenieure können das Subsystem anwenden, die von Datenverarbeitung nur wenig verstehen, da sie sich nur mit den in ihrem Arbeitsbereich üblichen Begriffen ausdrücken müssen.

- Der Anwender wird von der Beschäftigung mit EDV-Problemen befreit und kann sich mehr auf die Lösung des eigentlichen Problems konzentrieren.

- Beim Auftreten von Fehlern oder unerwarteter Ergebnisse können Fachleute des betreffenden Fachbereiches zu Fehlersuche auch dann

herangezogen werden, wenn sie weder von dem speziellen Programm noch von der EDV überhaupt besondere Kenntnisse besitzen. Der Fachmann kann allein aus der Auflistung der in der problemorientierten Sprache formulierten Eingabe erkennen, ob die dem Programm übergebene Problemstellung logische oder sachliche Fehler enthält.

Es ist ein besonderer Vorteil des ICES-Systems, daß es ein Subsystem CDL (Command Definition Language) enthält, mit dem es möglich ist, für jedes neue Subsystem eine POL zu definieren. Die spezielle POL für das Subsystem CDL wird ebenfalls CDL genannt.

Die POL-Eingabe aller Subsysteme wird von einem Kommando-Interpretierer (Command Interpreter, CI) interpretiert, der vom ICES-Kern allen Subsystemen bereitgestellt wird. Dieser Interpretierer bestimmt die Möglichkeiten und Einschränkungen der POL. So ist es nicht möglich, völlig beliebig konstruierte Sprachen als zulässige POL zu definieren, sondern die Sätze müssen ein bestimmtes Schema einhalten (vgl. als Beispiel Abb. 7):

Die ICES-POL besteht aus Sätzen, die einzeln und nacheinander gelesen und interpretiert werden. Jeweils das erste Wort eines POL-Satzes definiert das Kommando ("Command"). Die nachfolgenden Wörter werden entweder als Daten interpretiert oder als "Modifier". Modifier bestimmen eine von mehreren Variationsmöglichkeiten des Kommandos. Die Daten werden in ihrer Bedeutung entweder durch ihre Position oder durch Schlüsselwörter ("Identifier") identifiziert. Andere Wörter können als reine Füllwörter ("Ignorable Words") vorgesehen werden, um flüssiger lesbare Sätze zu ermöglichen. Die meisten Wörter können entweder vollständig oder abgekürzt bis auf eine jeweils festgelegte Minimalanzahl der ersten Buchstaben angegeben werden. Als Trennzeichen zwischen einzelnen Wörtern eines Kommandos gelten das Komma und eine oder mehrere Leerstellen ("Blanks"); andere Sonderzeichen (z.B. "+", "*" usw.) gelten hier jedoch im Gegensatz zu anderen Programmiersprachen nicht als Trennzeichen. Für den Aufbau flexibler Anweisungen ist es besonders vorteilhaft, daß der CI einen rekursiven Aufbau der Kommandos erlaubt.

Jedes einzelne Kommando wird vom CI anhand einer zugehörigen Tabelle, dem sogenannten Command Definition Block (CDB), interpretiert. Diese CDBs werden mit dem Subsystem CDL erstellt und auf einer speziellen externen Datei, die dem CI für alle Systeme zugänglich ist, abgelegt.

Die Kommando-Struktur ist insbesondere einer interaktiven Kommunikation zwischen Mensch und Maschine angepaßt. Zwar wurde ICES in der Vergangenheit nur im Stapel-Betrieb benutzt, die inzwischen entwickelten Timesharing-Betriebssysteme erlauben jedoch seit kurzem ICES auch interaktiv zu benutzen, und zwar aufgrund der Kommando-Struktur ohne [35] oder mit nur wenigen Änderungen [36].

Die einem interaktiven Betrieb gemäße Struktur kann jedoch im Stapel-Betrieb von Nachteil sein, da Schleifen und bedingte Verzweigungen nicht und eine Blockstruktur nur schwer realisiert werden können.

3.1.3 Die Programmiersprache ICETRAN

ICETRAN [37] ist eine Erweiterung von FORTRAN-E [38] (auch "Basic-Fortran" genannt). Die Erweiterung wird durch einen ICETRAN-Precompiler verifiziert, der die ICETRAN-Befehle in FORTRAN-E Anweisungen durch Unterprogramm-Aufrufe ersetzt. Diese Unterprogramme sind Assemblerroutinen des ICES-Kerns, die die zusätzlichen Fähigkeiten bereitstellen. Die vom Precompiler erzeugten FORTRAN-E-Programme werden von dem IBM-FORTRAN-E-Compiler in Maschinencode umgesetzt.

Die Wahl von FORTRAN-E als Basissprache ist zum einen dadurch begründet, daß FORTRAN die am meisten gebräuchliche technisch-wissenschaftliche Programmiersprache ist und zum anderen dadurch, daß zur Zeit der ICES-Entstehung die neueren FORTRAN-Versionen (G- und H-Compiler der IBM) noch nicht existierten. Leider enthält FORTRAN-E eine Reihe der Möglichkeiten von FORTRAN-G/H nicht (siehe Anhang 2), die demzufolge auch in ICETRAN nicht enthalten sind. Diese Einschränkungen gelten nicht für PSU-ICES [8, 9, 10].

ICETRAN bietet gegenüber FORTRAN im wesentlichen vier Gruppen von Neuerungen:

a) Dynamische Datenfelder

b) Dynamische Kontrollübergabe zwischen Unterprogrammen

c) Anweisungen zur Verwaltung sekundärer Direktzugriffsspeicher (zusätzlich zu den in FORTRAN-IV existierenden)

d) Anweisungen für Fehlerbehandlungen

Diese Erweiterungen sind in vielen Fällen äußerst wertvoll und daher oft der entscheidende Grund für die Wahl von ICETRAN als Programmiersprache. Eine Zusammenstellung der zugehörigen Syntax-Elemente zeigt Anhang 5 [37, S. 38/39]. Hier wird zunächst nur die Erweiterung durch dynamische Datenfelder erläutert. Auf die dynamische Programmkontrolle wird im folgenden Abschnitt genauer eingegangen.

In ICETRAN gibt es zusätzlich zu den von FORTRAN bekannten, durch DIMENSION-Anweisungen definierten Datenfeldern konstanterLänge, dynamische Datenfelder, die als DYNAMIC ARRAY(DA) deklariert werden. Die genauen Eigenschaften dieser DAs sind in [2,4,37] beschrieben. Hier werden nur die wesentlichen herausgestellt:

Mit DAs ist es möglich

- Kernspeicherplatz dynamisch zu belegen. Felder können allociert (d.h. vom Betriebssystem angefordert) und freigegeben werden.

- mehrdimensionale und nicht rechtwinklige Feldstrukturen (z.B. Dreiecksmatrizen) zu definieren. Die genaue Form der Struktur kann hierbei zur Ausführungszeit bestimmt werden.

- sicherzustellen, daß nicht auf Bereiche außerhalb der definierten Datenfelder zugegriffen wird.

- Felder mit dynamischer Größe so zu spezifizieren, daß sie zunächst mit einer relativ kleinen Anfangslänge angelegt werden, aber vom ICES-Kern automatisch mit vorgebbaren Inkrementen vergrößert werden, wenn der zunächst reservierte Platz nicht ausreicht.

- externe Direktzugriffsspeicher als virtuellen Kernspeicher bereitzustellen: der ICES-Kern verlagert automatisch bei Überlauf des direkt zugreifbaren Kernspeichers momentan nicht benötigte Felder auf die

externen Dateien und holt diese bei Bedarf wieder in den Kernspeicher zurück; dies geschieht völlig automatisch; der Programmierer kann jedoch diese Kernspeicherverwaltung in ihrer Effektivität durch Angabe von Prioritäten und Kennzeichnung aktuell nicht benötigte Felder beeinflussen.

- zusammengehörige Daten unterschiedlicher Form (Festkomma, Gleitkomma mit unterschiedlicher Genauigkeit, Zeichenketten) in einem Datenfeld abzulegen (ähnlich wie in einer PL/1-STRUCTURE).

- ein und dasselbe Datenfeld mit mehreren Variablennamen zu bezeichnen.

Diese dynamischen Datenfelder werden mit indizierten Variablennamen bezeichnet, sind also bezüglich der Syntax mit den in FORTRAN üblichen dimensionierten Feldern verwandt; insbesondere können auf dynamische Datenfelder im wesentlichen alle Operationen angewandt werden, die für festdimensionierte Felder in FORTRAN erlaubt sind, jedoch darüberhinaus auch noch einige weitere.

Gegenüber den "structures" in PL/1 [39] bietet ICETRAN nicht die Möglichkeit, einzelne Elemente in derartigen Feldern über "qualified names" anzusprechen; jedoch liegt dieser Unterschied nur in der Syntax und kann durch entsprechende Erweiterung des Precompilers ohne weitere Änderung der Semantik erzielt werden. Darüber hinaus enthält ICETRAN von der Syntax her keine Möglichkeiten (außer weniger Sonderfälle), DAs als Ganzes zu verarbeiten. Dafür bieten die DAs folgende nicht in PL/1 enthaltene Möglichkeiten:

- die Simulation eines virtuellen Speichers

- die Datenfeldüberwachung

- die automatische Verlängerung von Feldern, deren Größe nicht ausreicht und deren Verlängerung vorgesehen ist.

- die Definition der Array-Struktur zur Ausführungszeit und der Abfrage der aktuellen Struktur im Programm;
  DAs sind in diesem Sinne selbstbeschreibend, was die Vorausetzung ist für die in [40,41] beschriebenen Algorithmen zum Drucken, Schreiben, Lesen und Kopieren beliebiger DYNAMIC ARRAYs als Ganzes.

Die Implementierung der dynamischen Datenfelder wird im folgenden andeutet:

Die dynamische Datenfelder bestehen aus einzelnen zusammenhängend gespeicherten Feldern (sog. Subarrays), die über "Pointer" (Adreßzeiger) verknüpft sind.Zu einem dynamischen Datenfeld existiert zunächst ein sog. "Basepointer", dieser zeigt auf den Beginn eines Subarrays, der seinerseits entweder Daten oder Pointer enthält. Die Pointer zeigen ihrerseits wieder auf entsprechende Subarrays. Die so definierte Struktur ist eine Baumstruktur, bei der die Knoten von den Subarrays gebildet werden.
Je nachdem der Basepointer nur in einem Unterprogramm oder (als Subroutinen-Argument oder COMMON-Variable) darüber hinaus bekannt ist, nennt man den zugehörigen DYNAMIC ARRAY "lokal" oder "global".

3.1.4 Lade-Modul-Struktur

Zu einem ICES-Subsystem gehören ein oder mehrere Lade-Module (Definition in Anhang 1). Der Inhalt(nicht die Struktur) eines Lade-Moduls wird in der Regel mit der Programmiersprache ICETRAN programmiert. In ICES ist es mit Hilfe von ICETRAN möglich, nicht nur innerhalb eines Lade-Moduls die Kontrolle von einem Unterprogramm (das ist im OS eine Kontroll-Sektion, siehe Anhang 1) an ein anderes zu geben, auch die Kontrolle aus einem Lade-Modul an ein Unterprogramm in einem anderen Lade-Modul (genauer: an solche Kontroll-Sektionen deren Entry-Points gleichzeitig Entry Point des Lade-Moduls sind). Im Gegensatz dazu kennt man in den sonst üblichen höheren Programmiersprachen (FORTRAN, PL/1 usw.) nur einen Lade-Modul, und die Kontrolle kann dort nur innerhalb dieses Moduls von einer Kontroll-Sektion an eine andere übergeben werden.

Innerhalb eines Lade-Moduls wird die Kontrollübergabe in ICETRAN, wie auch sonst üblich, mit der Anweisung

```
CALL  name
```

an den Entry Point "name" eines Unterprogramms veranlaßt.

Der Aufruf eines Unterprogramms "name" in einem anderen Lade-Modul erfolgt mit einem der folgenden Anweisungen oder Anweisungsgruppen

```
          LINK name

oder:       LOAD  name (entry)
              :
            BRANCH  name (entry)
              :
          DELETE  name

oder:     TRANSFER  name

oder:       ADD  TO  STACK  (count,name)
                :
            LINK        TO  STACK
                :
          DELETE  FROM  STACK  (count)

oder:       ADD  TO  STACK  (name,count)
              :
            TRANSFER  TO  STACK
```

Diese Anweisungen sind im Detail in [2,4,37] beschrieben. Sie alle bewirken den Aufruf des Unterprogramms mit dem Namen "name" in einem anderen Lade-Modul, unterscheiden sich jedoch bezüglich

- des Zeitpunktes zu dem der aufgerufene Lade-Modul geladen und der belegte Platz später wieder freigegeben wird
- der Kontrollrückgabe vom aufgerufenen Unterprogramm
- der Anzahl der durch diesen Befehl zur Ausführung kommenden Unterprogramme
- der Argumentübergabe (diese wurde bei der Beschreibung der Befehlsformen nicht berücksichtigt; sie ist nur bei LINK bzw. TRANSFER TO STACK nicht möglich.)

Was sind die Vorteile der Verwendung von Lade-Modulen mit diesen ICETRAN-Anweisungen? Durch die Verwendung von Lade-Modulen mit den zugehörigen Anweisungen ist es möglich:

- die Datei der Subsystem-Programme modular aufzubauen und ohne jede Änderung des alten Bestandes neue Programme hinzuzufügen

- die Programme so zu segmentieren, daß auch große Programmpakete in einem relativ kleinen Kernspeicher gerechnet werden können

- die Segmentierung und Kernspeicherbelegung zu beeinflussen und aufgrund der Kenntnis des Programmablaufs und der durch die aktuelle Eingabe bestimmten Situation möglichst optimal zu gestalten

- die Kontrolle an einen Lade- Modul zu übergeben, dessen Name erst zur Ausführungszeit (z.B. durch Eingabe) dem Subsystem bekannt wird.

Der zweite Punkt läßt sich in der Regel auch durch geeignete Overlay-Strukturen (siehe Anhang 1) [42,43] erreichen; die drei anderen Vorteile sind dagegen mit Overlay nicht verwirklichbar. Zudem ist es in ICES ebenfalls möglich, einen Lade-Modul mit einer Overlay-Struktur zu versehen.

Ein Lade-Modul bildet die Vereinigung verschiedener Objekt-Modulen und vorhandener Lade-Modulen. Die Bildung dieser Lade-Module geschieht in ICES in einem speziellen Arbeitsgang unter der Kontrolle eines Programms, das folgende Eingaben erwartet [37, Chapter 4]:

- den Namen des Subsystems, zu dem die Objekt-Module gehören

- den Namen des Lade-Moduls, der erzeugt werden soll

- die Anzahl und Namen der Programme, deren Entry-Points die Entry-Points des Lade-Moduls bilden sollen

- die Namen der sonstigen Programme

- welche dieser Programme auf den COMMON zugreifen

- die Struktur des Lade-Moduls (ohne oder mit Overlay und im zweiten Fall die exakte Definition der Overlay-Struktur)

- eine Angabe darüber, ob das hier zusätzlich erstellte Assembler-Vorprogramm ("SETUP", siehe unten) ausgestanzt werden soll oder nicht

Bezüglich der Implementierung sei erwähnt, daß die LINK- und LOAD/BRANCH/DELETE-Anweisungen in ihrer Wirkung mit den Assembler-Macros [44] gleichen Namens verwandt sind. Die LINK-Anweisung unterscheidet sich jedoch in ihrer Funktion dadurch von dem entsprechenden Assembler-Macro, als der Lade-Modul, dem durch dieses LINK die Kontrolle übergeben wird, in ICES nach der Kontrollrückgabe im Kernspeicher für eine gewisse Dauer verbleibt und bei einem sofortigen zweiten LINK nicht neu geladen werden braucht, während er bei dem Assembler-Macro dann neu geladen wird [33].

Als zweiter Aspekt der Implementierung muß der Lade-Modul-Aufbau etwas genauer betrachtet werden, weil nur so der Programmablauf und Datenfluß der Lade-Modul-Erstellung verstanden werden kann.

In ICES enthält jeder Lade-Modul eine spezielle Kontroll-Sektion mit dem Namen SETUP, die als Interface zwischen dem ICES-Kern und den einzelnen Unterprogrammen dient. SETUP hat vor allem die Aufgabe, die Adresse des COMMON dem Lade-Modul bereitzustellen. Der COMMON ist in ICES ein Datenbereich, der direkt vom ICES-Kern verwaltet wird und auf den alle Programme in beliebigen Lade-Modulen desselben Subsystems zugreifen können. Dies steht im Gegensatz zu dem sonst in FORTRAN und dem IBM-360/OS üblichen COMMON-Bereich, der ein Teil eines Lade-Moduls ist und nur dort zugreifbar. In ICES wird daher bei jedem Aufruf eines Programms in einem anderen Lade-Modul die Kontrolle vom ICES-Kern zunächst der Kontroll-Sektion SETUP des Lade-Moduls übergeben, dem als Haupt- oder Alias-Namen der Name des gewünschten Unterprogramms (verlängert um den Prefix QQ) zugeordnet wurde. Hierbei wird SETUP zusätzlich zu den vom Programmierer spezifizierten Argumenten die Adresse des COMMON-Bereichs sowie der Name des aufzurufenden Unterprogramms als Argumente übergeben. Anhand dieser Angaben stellt SETUP die COMMON-Adresse allen Unterprogrammen in diesem Lade-Modul bereit und übergibt dann, zusammen mit den eigentlichen Argumenten, die Kontrolle an das durch den übergebenen Namen identifizierte Unterprogramm.

Da alle Moduln in nur einer Bibliothek (ICESMODS, siehe unten) verwaltet werden, kann im Prinzip innerhalb eines Subsystems auch ein Modul eines anderen Subsystems aufgerufen werden. Da jedoch die COMMON-Struktur verschiedener Subsysteme verschieden ist, ist dies nur in Ausnahmefällen und bei besonderer Planung sinnvoll. Ein solcher Ausnahmefall

ist der Aufruf der zum Subsystem TABLE (I oder II) gehörigen Programme zur Tabellenverwaltung (z.B. in STRUDL II).

3.1.5 Common-Verwaltung

Wie bereits erwähnt, stellt der COMMON im Gegensatz zu Fortranprogrammen einen Kernspeicherbereich dar, der allen Lade-Modulen eines Subsystems zugänglich ist (selbstverständlich auch allen Kontrollsektionen innerhalb eines Lade-Moduls). Maßgeblich für die Identifikation der Werte im COMMON ist wie in Fortran die relative Position bezogen auf den Anfang des COMMON. Vom ICETRAN-Precompiler wird in jedes ICETRAN-Programm vor den dort vom Programmierer angegebenen COMMON ein Standard-Common-Bereich von 80 Worten (320 Bytes) Länge eingefügt, ohne daß der Programmierer der ICETRAN-Programme in irgendeiner Weise darauf Rücksicht nehmen müßte. Der Common hat aber noch eine zweite Funktion. Er dient zur Kommunikation zwischen dem Kommando-Interpretierer und den Programmen. Der Kommando-Interpretierer legt in diesen COMMON nach den in in den CDB's enthaltenen (d.h. mittels der CDL für dieses Subsystem spezifizierten) Vorschriften die vom Subsystem-Benutzer in den Befehlen enthaltenen Daten ab. Die Identifikation der COMMON-Plätze innerhalb der CDL geschieht ebenfalls nach der Position, die jedoch hier in Bytes bezüglich des Anfangs des gesamten COMMON gerechnet wird (ein kleiner Schönheitsfehler). Das bedeutet, daß die Position 320 in der CDL dem ersten COMMON-Platz im ICETRAN-Programm zuzuordnen ist, 324 dem zweiten, usw. Dies sei an folgendem Beispiel erläutert:

CDL-Angabe

```
COMMON
ADD  'I'  320
ADD  'J'  324
END  COMMON
ADD  'DRU'
 ID  'ERST' INTEGER 'I'
 ID  'DANN' INTEGER 'J'
 EXECUTE LINK 'PRINT'
FILE
```

ICETRAN-Programm

```
      SUBROUTINE PRINT
      COMMON N,M
      WRITE (6,1) N,M
    1 FORMAT (1X,2I10)
      RETURN
      END
```

Wenn ein Subsystem obige Befehlsdefinitionen und das obige Programm PRINT enthält, so führt der Befehl

```
DRUCK  ERST  5  DANN  7
```

dazu, daß die Zahlen 5 und 7 im Format(1X,2I10) gebracht werden. Die Variablen 'I' und 'J' bzw. N und M bedeuten dieselben Speicherplätze im COMMON.

3.1.6 POL-Definition mittels CDL

Das Subsystem CDL nimmt insofern eine Sonderstellung unter allen Subsystemen ein, als es für die Sprachdefinition aller Subsysteme benötigt wird. (Auch die CDL-Sprache selbst wurde bis auf ganz wenige einprogrammierte Grundbefehle mittels der CDL definiert.) Die CDL, ihre Möglichkeiten und ihre Wirkung sind eingehend in [2,37] geschildert. Hier sollen lediglich die wichtigsten Funktionen genannt werden:

- Eröffnung eines neuen Subsystems
- Definition der Größe der Kernspeicherbereiche für den COMMON und den Arbeitsspeicher zur Verwaltung der dynamischen Datenfelder
- Definition der Befehlssyntax
- Definition der Interpretationsvorschrift (REAL, INTEGER, ALPHA) für Daten und Schlüsselworte, die im Befehl enthalten sind, und Angabe ihres Ablageortes im COMMON
- Angabe von Standardwerten für diese Daten
- Aufruf von Verarbeitungsprogrammen. (Hierbei können keine Argumente übergeben werden.)

3.2 Dateien, Programme und Informationsfluß der Subsystemerzeugung und Ausführung

3.2.1 Übersicht

Die Abb. 2 und 3 zeigen die in ICES verwendeten Dateien, Programme sowie den Datenfluß, und zwar Abb. 2 für die Subsysteme Lade-Modul-Erstellung und Abb. 3 für die Subsystem-Ausführung. Sie zeigen in feinerer

Auflösung das, was im Prinzip bereits in Abb. 1 dargestellt wurde, nämlich die Trennung der Eingaben zur Definition der ausführenden Programme, der Lade-Modul-Struktur, der POL und der aktuellen Programmausführung.

Die Abb. 2 und 3 zeigen jedoch nicht so sehr das Prinzip, sondern mehr die Implementierungsform. Es ist nicht zuletzt diese auf den ersten Blick verwirrende Zahl von Dateien und Programmen, die den Umgang mit dem ICES-System in der Praxis erschwert. Allerdings werden die einzelnen Programme von einigen wenigen Kontrollprogrammen überwacht, wie Abb. 4 zeigt, und die Ausführung dieser Programme mit der Definition der zahlreichen Dateien kann durch Job Control Language (JCL)-Prozeduren stark vereinfacht werden. Abb. 5 zeigt eine der für diesen Zweck in der GfK implementierten Prozeduren. Bei Verwendung dieser Prozeduren erscheinen tatsächlich nur die drei in Abb. 1 skizzierten Eingabedateien, wie sie auch in den Abb. 2 und 3 hervorgehoben sind.

Im folgenden werden einige der bezeichneten Programme und Dateien näher erläutert. Die Funktionen von Precompiler, Compiler, Assembler und Linkage Editor werden als bekannt vorausgesetzt.

3.2.2 Der Objekt-Modul-Lader QQNIX3

Das Programm QQNIX3 hat zwei Aufgaben:

a) Der Standard-FORTRAN-Compiler hat einen Objekt-Modul erzeugt, der voraussetzt, daß der COMMON eine Kontroll-Sektion in dem Lade-Modul ist, zu dem der übrige Objekt-Modul gehören soll. Dazu wurde im Objekt-Modul die Länge des im FORTRAN-Programm spezifizierten COMMON abgelegt. In ICES ist jedoch der COMMON ein Datenbereich, der unabhängig von den Lade-Modulen im Kernspeicher angelegt wird und mit dem FORTRAN-COMMON nicht zusammenfällt. QQNIX3 reduziert daher die COMMON-Länge auf den minimalmöglichen Wert von 4 Bytes.

b) Die einzelnen Objekt-Modulen, die vom Compiler hintereinander auf eine sequentiell organisierte Datei geschrieben wurden, werden von QQNIX3 einzeln in eine partitioned organisierte Bibliothek geladen. Dies ist erforderlich, da der Linkage Editor auf Objekt-Modulen nur in einer partitioned organisierten Bibliothek einzeln zugreifen kann. Der gesonderte Zugriff auf einzelne Objekt-Modulen ist andererseits notwendig, da aus mehreren, gleichzeitig übersetzten Programmen meh-

rere unabhängige Lade-Modulen erzeugt werden sollen.

3.2.3 Das Programm QQSETGEN zur Generierung von "SETUP" und Linkage Editor Kontrollkarten-Eingabe

Das Programm QQSETGEN hat zwei Aufgaben:

a) Es generiert aufgrund der Eingabe die in jedem ICES-Lade-Modul erforderliche Entry-Kontroll-Sektion "SETUP", deren Aufgabe in Kapt. 3.1.4 erläutert wurde. Der hier erzeugte SETUP ist ein Assembler Quell-Modul, der eine Tabelle der Namen der im Lade-Modul aufzunehmenden Unterprogramme enthält, diese Tabelle nimmt zu jedem Namen die Entry-Adresse der Unterprogramme auf.

b) Aufgrund der Eingabe erzeugt SETUP die Anweisungen an den Linkage Editor, die zur Definition der Lade-Modul-Struktur erforderlich sind. Hierzu gehören

INCLUDE-Anweisungen	- zur Integration der erforderlichen Objekt-Module aus der partitioned organisierten Objektbibliothek
NAME und ALIAS-Anweisungen	- zur Spezifikation der Haupt- und Aliasnamen des Lade-Moduls (die Namen der Unterprogramme, die Entry zu diesem Lade-Modul im ICES-Sinne sind)
eine ENTRY-Anweisung	- zur Spezifikation von SETUP als Entry des Lade-Moduls im OS/360-Sinne
OVERLAY-Anweisungen	- zur Spezifikation der Overlay-Struktur

3.2.4 Die permanenten Dateien

Von den angedeuteten Dateien werden im folgenden nur die überlicherweise permanent verwalteten Dateien erläutert; die hier verwendeten Namen dienen lediglich zur Identifikation der Dateien in den Abbildungen, sie sind nicht überall üblich, entsprechen aber den Namen der Dateien in den JCL-Prozeduren, wie sie in der GfK verwendet werden (vgl. Abb. 5).

MODS (Abkürzung für ICES-Module, auch ICES.LINK.LIB genannt)

Diese Datei ist eine partitioned organisierte Lade-Modul-Bibliothek und enthält alle zur Ausführung der Subsysteme sowie zu ihrer Erzeugung erforderlichen Programme in ausführbarer Form.

FUNCLIB (Abkürzung für Function Library)

Diese Datei ist wie MODS organisiert. Sie enthält zwei Gruppen von nichtausführbaren Programmen in Lade-Modul-Form:

- sie enthält die von FORTRAN bereitgestellten Unterprogramme (z.B. SIN, MAXO, usw.)

- außerdem enthält sie die zahlreichen Unterprogramme, die für die Ausführung der in ICETRAN - aber nicht in FORTRAN - möglichen Anweisungen von ICES bereitgestellt werden (z.B. zur Ausführung von LINK-Anweisungen oder zum Zugriff auf die dynamischen Datenfelder). Diese Unterprogramme werden in alle Lade-Module vom Linkage Editor automatisch (sog. "automatic library call") eingebaut. Um die Duplizierung von Programmen nicht zuviel Platz kosten zu lassen, führen die hier bereitgestellten Unterprogramme die verlangten Operationen in der Regel nicht selbst aus, sondern verzweigen anhand der im COMMON abgelegten Adressen zu den entsprechenden Programmteilen im ICES-Kern.

  Hier sei erwähnt, daß diese Implementierung einige Nachteile enthält: die Verzweigung in den ICES-Kern kostet Rechenzeit, die Interface-Routinen kosten Platz. Von der Pennsylvania State University (PSU) wurde in jüngster Zeit eine neue ICES-Version erstellt, die diese Nachteile vermeidet [8,9,10].

MACLIB (Abkürzung von Macro Library)

MACLIB enthält Daten im Kartenformat in einer partitioned organisierten Form. Die Daten beschreiben die für die Assemblierung von SETUP erforderlichen Assembler-Macros.

DICT (ICES-Dictionary)

Diese Datei ist eine vom ICES-Kern verwaltete Direkt-Zugriffs-Datei und enthält die CDBs(Command Definition Blocks). Anhand der CDBs werden

vom Kommando-Interpretierer die eingegebenen POL-Anweisungen interpretiert. Die CDBs werden mit dem Subsystem CDL erzeugt.

SUBS (ICES-Subsystem-Data Sets)

und

USER (ICES-User-Data Sets)

Diese beiden Dateien werden vom ICETRAN-Programmierer mit den speziell hierfür vorgesehenen Direkt-Zugriffs-Anweisungen verwaltet. Hier können die in einem Subsystem oder von einem Benutzer langfristig benötigten Daten abgelegt werden. Beispielsweise werden im Subsystem PROJECT die Daten der Feiertage und die Daten eines bestimmten Netzwerkes auf diesen Dateien abgelegt. Für die Trennung der beiden Dateien gibt es keinen zwingenden Grund und die Entscheidung über die Aufteilung der Daten auf die beiden Dateien ist dem Ermessen des Subsystemplaners überlassen. Als Regel gilt jedoch (so z.B. in TABLE /13/ realisiert), daß USER normalerweise allein für einen Benutzer relevante Daten aufnehmen soll, während die Daten in SUBS allen Benutzern eines Subsystems gleichermaßen zur Verfügung stehen. Die einzelnen Datensätze können vom Benutzer durch Paßwörter vor unerlaubten Zugriffen geschützt werden.

3.2.5 Die ICES-Jobstep-Kontrollprogramme

Um die Vielzahl der Programme nicht in einzelnen Jobsteps ansprechen zu müssen und um gewisse Loops (z.B. über QQSETGEN, Assembler und Linkage Editor je Lade-Modul) zu erlauben, werden einige der in Abb. 2 gezeigten Programme in der Regel über gewisse Kontrollprogramme (QQFUBAR und QQFUBAR2), wie Abb. 4 zeigt, verwaltet. Mit Hilfe dieser Programme kann die Subsystem-Erzeugung und -Ausführung in nur drei Jobsteps durchgeführt werden. In Abb. 4 ist zudem die Möglichkeit angedeutet, in ICES Programme unabhängig vom Command-Interpretierer laufen zu lassen (unter der Kontrolle von NONICES).

3.2.6 Kernspeicheraufteilung

Abb. 6 zeigt die während der Subsystem-Ausführung in ICES im Kernspeicher verwalteten Bereiche. Der Kernspeicher enthält zunächst einen Teil der

ICES-Kernprogramme resident. Es gibt verschiedene Versionen dieses Programms - mit und ohne Overlay-Struktur -, so daß der hierdurch belegte Platz dem insgesamt verfügbaren Kernspeicher angepaßt werden kann. Ein zweiter dient zur Aufnahme der einzelnen auszuführenden Lade-Module, die der Bibliothek MODS entnommen werden. Permanent im Kernspeicher verweilt der COMMON-Bereich, dessen Funktion bereits im Kap. 3.1.5 erläutert wurde. Schließlich enthält der Kernspeicher den Daten-Pool, der virtuell - durch geeignete Routinen des ICES-Kerns - auf eine Direktzugriffsdatei ausgedehnt werden kann.

Es gibt verschiedene Strategien zur Verwaltung dieser Kernspeicherbereiche, die sich in verschiedenen "Modifications" des ICES-Kerns wiederspiegeln. Gegenwärtig existieren vier Versionen (Mod 1-4), von denen Mod 1 jedoch nicht standardmäßig verwendet wird. Eine genaue Beschreibung der verschiedenen Strategien enthält die Dissertation von Sussman [33].

3.3 Entstehungsweg eines neuen Subsystems

3.3.1 Die Arbeitsschritte bei der Erstellung eines Subsystems

In diesem Abschnitt wird ähnlich wie in [2,3] beschrieben, welche einzelnen Arbeitsschritte zu durchlaufen sind, wenn ein neues Sybsystem in ICES erstellt werden soll. Die Arbeitsschritte sind die folgenden:

- Studiere ICES und erarbeite die notwendigen Kenntnisse durch die praktische Erstellung kleiner Beispiele
- Plane und definiere die Fähigkeiten
- Stelle die grundsätzlichen Methoden und Algorithmen bereit
- Entwerfe die Struktur und Semantik der Eingabe-Sprache (POL)
- Definiere die interne Datenstruktur
  (im Kernspeicher, COMMON, auf Platten oder Bändern)
- Definiere die Modul-Struktur
- Programmiere die auszuführenden ICETRAN-Programme

- Stelle den Direkt-Zugriff-Platz für permanente Dateien bereit
- Beschreibe die POL mit Hilfe des Subsystems CDL
- Lade die Lade-Modulen und CDBs auf die entsprechenden Dateien, üblicherweise auf permanente Dateien
- Erstelle ausführliche Testeingabe und und teste das neue Subsystem
- Erstelle Dokumentation
- Anwendung, Verbreitung und Pflege des Subsystems

Die hier angegebene Reihenfolge muß nicht überall der zeitlichen Sequenz entsprechen, wenngleich sie in etwa eingehalten werden sollte. Sicherlich werden neue Erfahrungen und Zwischenergebnisse zu einem interativen Vorgehen zwingen, das gerade durch die Unabhängigkeit der Moduln und der POL besonders unterstützt wird. Zu einigen Punkten werden praktische Hinweise gegeben, die erfahrungsgemäß wichtig sind.

3.3.2 Erlernen der notwendigen ICES-Kenntnisse

Als empfehlenswerte Literatur sind hier die Einführungen in [1,2,4] zu nennen. Notwendiges Handwerkszeug ist die Kenntnis von Fortran E [38] sowie der ICES-Programmiersprache ICETRAN, der Eingabe zu QQSETGEN zur Definition der Load-Module-Struktur und der Command-Definition Language CDL [37]. Der "Subsystem Primer" [3] bietet einige einfache Beispiele mit Anleitung zur Übung.

3.3.3 Planung der Subsystem-Fähigkeiten

Hier sind die wesentlichen Solleigenschaften des Subsystems festzulegen.

3.3.4 Methoden und Algorithmen-Grundlagen

Die Subsystemerstellung ist zu großen Teilen die Aufgabe von Programmierern. Es kann nicht ihre Aufgabe sein, die Algorithmen, wie z.B. zur Auflösung von Gleichungssystemen oder zur Optimierung, zu entwickeln. Die hier notwendigen Verfahren sind bereitzustellen oder zu entwickeln.

Oft kann auf vorhandene Unterprogramme hierbei zurückgegriffen werden. Es gibt zwar nur wenige allgemein verwendbare ICETRAN-Unterprogramme [20,21,23], jedoch können viele FORTRAN-Unterprogramme mit wenigen Änderungen verwendet werden; die Änderungen beziehen sich zumeist auf die Einschränkungen von FORTRAN E gegenüber Fortran G/H sowie der Wandlung festdimensionierter Felder in DYNAMIC ARRAYs [37] (siehe auch Anhang 3).

3.3.5 Sprachentwurf

Hier ist festzulegen, welche Operationen von der Sprache im einzelnen ansprechbar sein sollten, welche Sprachschlüsselwörter ("Commands", "Modifier" und "Identifier") gewählt werden sollen, welche Eingabedaten erwartet werden, welche Standardwerte vorgesehen sein sollen und welche Wirkung die Kommandos im einzelnen haben sollen.

Die Syntaxdefinition geschieht am besten zunächst in einer formalen Schreibweise mit der in ICES üblichen Klammernnotation [37, Erweiterungen in 13]. Es ist hier sehr leicht möglich, eine Sprache sehr komplex zu gestalten, so daß später dicke Handbücher zu ihrer genauen Beschreibung erforderlich sind. Eine gewisse Systematik der Befehlsstruktur ist daher anzustreben. Bezüglich der möglichen Abkürzungen sollte man eine bestimmte Anzahl der ersten Buchstaben für das ganze Subsystem festlegen (beispielsweise 3 oder sicherer 4 Buchstaben).

3.3.6 Definition der inneren Datenstruktur

Besonderer Beachtung bedarf hier der COMMON, da dieser Datenbereich im Kernspeicher das Interface zwischen POL und den ausführenden Programmen, sowie den einzelnen Programm-Modulen untereinander darstellt. Eine Veränderung des COMMON erzwingt eine Neuerstellung der POL und aller Lade-Moduln, die auf COMMON-Variablen zugreifen. Man sollte hier daher von vornherein Reserve-Plätze für Variablen verschiedenen Typs (Integer, Real, Double, Alpha) sowie für die Basepointer globaler DYNAMIC ARRAYs (siehe Kap. 3.1.3) vorsehen.

Bei Verwendung von lokalen DYNAMIC ARRAYs ist unbedingt darauf zu achten, daß diese vor Verlassen des zugehörigen Lade-Moduls zum aufrufen-

den Programm zerstört werden, da sonst schwer lokalisierbare Fehlermeldungen auftreten.

3.3.7 Definition der Modul-Struktur

Bevor mit der Programmierung der ICETRAN-Programme begonnen wird, muß die Modul-Struktur definiert sein, da der Aufruf eines Unterprogramms in einem fremden Lade-Modul andere Anweisungen als der Aufruf eines Unterprogramms im selben Lade-Modul erfordert (siehe Kap. 3.1.4). Innerhalb eines Lade-Moduls wird hierzu die "CALL"-Anweisung, zwischen Lade-Modulen am häufigsten die "LINK"-Anweisung benutzt. Da die endgültige Modul-Struktur oft von der ersten Planung abweicht, ist es empfehlenswert, die Kontrollübergabe zunächst stets mit der "CALL"-Anweisung zu programmieren; ICETRAN erlaubt mit der DYNAMIC PROGRAM-Anweisung die nachträgliche Umwandlung aller "CALL"-Anweisungen für ein Unterprogramm in einem Quell-Modul in "LINK"-Anweisungen [37]. Die hier genannten Überlegungen sind bei Verwendung von PSU-ICES überflüssig [8,9].

3.3.8 Direkt-Zugriff-Platz für permanente Dateien

Für folgende Dateien (vgl. Kap. 3.2) sollte permanent Platz auf Platten (2311,2314,3330) oder Trommeln bereitstehen:

MODS (erforderlich, da diese Bibliothek die ICES-Kern-Programme aufnehmen muß),

FUNCLIB (enthält die notwendigen ICES-Unterprogramme, die in jedem Lade-Modul verwendet werden),

DICT (enthält die CDBs; ohne diese kann kein Subsystem benutzt werden).

MACLIB (relativ klein; wichtig für Erstellung der Lade-Module). Empfehlenswert sind zudem permanente Dateien für USER und SUBS. Alle anderen Dateien können ohne Nachteile temporär angelegt werden. Bei häufiger Benutzung sind Programme vorzubereiten, die das Komprimieren häufig veränderter permanenter Dateien durchführen (insbesondere DICT und MODS).

3.3.9 Dokumentation

Bei umfangreichen Subsystemen empfiehlt sich die Vorbereitung von Formularen, auf denen die Programme dokumentiert werden. Die ICES-Source-Beschreibung [34] ist hier ein empfehlenswertes Vorbild.

3.4 Erfahrungen bei der Erstellung von Subsystemen

Im folgenden wird von drei Subsystemen berichtet, die in der GfK bearbeitet werden. Es wurden hierbei sowohl positive als auch negative Erfahrungen gesammelt, über die hier berichtet werden soll. Da diese Subsysteme nicht anderenorts veröffentlicht sind, ist jeweils zuerst eine Beschreibung der Subsysteme notwendig.

3.4.1 FRED, Fast Reactor Design Code *)

In der GfK existieren zahlreiche Programme zur nuklearphysikalischen, thermodynamischen und technischen Berechnung von Schnellen Reaktoren. Jedes dieser Programme ist für ein bestimmtes Teilproblem der Reaktorauslegung geeignet. Zudem liegen den meisten Programmen unterschiedliche Modelle eines Reaktors zugrunde (ein-, zwei- und dreidimensional, Mischungszoneneinteilung oder Brennelementeinteilung usw.).Ein großer Teil der nuklearphysikalischen Programme ist in dem Programmsystem NUSYS [45] integriert, viele andere, vor allem die Mehrzahl der thermodynamischen und technischen Programme sind sogenannte "stand alone codes"[46], d.h. ihre Ausführung ist unabhängig von der anderer Programme. Für eine praktische Auslegung eines Reaktors ist es jedoch erforderlich, die verschiedenen Einzelprogramme kombiniert anzuwenden: die Programme benötigen Eingabedaten, die von anderen erzeugt werden. Hierbei sind die Programme iterativ anzuwenden, d.h. einige Programme sind zunächst mit Schätzdaten auszuführen, die Ergebnisse bilden die Eingabe zu anderen Programmen, die wiederum genauere Eingabedaten für die ersten liefern. Es entstand daher der Wunsch, die zahlreichen Einzelprogramme in einem Programmsystem zu vereinigen.

Dieses Programmsystem muß insbesondere die folgenden Bedingungen erfüllen:

---

*) Die Programmierarbeiten zu FRED wurden überwiegend von Herrn W.Zimmerer ausgeführt. Die Autoren möchten ihm hierfür ihren besonderen Dank aussprechen.

- Es muß ein Programmengagement existieren, das die Ausführung der Einzelprogramme überwacht und dabei flexible Programmfolgen, einschließlich zyklischer erlaubt

- Es muß ein Dateninterface definiert sein. Ein derartiges "Interface" /-46-/ ist eine Sammlung von Daten die von den Programmen als Eingabe erwartet werden oder als Ausgabe produziert werden. Hierbei hat das System Routinen bereitzustellen, die das Lesen und Speichern der Daten den einzelnen Programmen erleichtern. Zugleich aber muß das Interface eine Umwandlung der Daten von einer Darstellungsform in eine andere (z.B. Darstellung eines Flußfeldes auf Maschennetzen unterschiedlicher Struktur oder als Reihenentwicklungen mit entsprechenden Koeffizienten) sowie die Umrechnung von einem Modell (s.o.) in ein anderes durchführen können (z.B. Kondensation einer Mehrgruppendarstellung).

- Es muß eine Bedienungssprache bereitgestellt werden, die die Anwendung der Programme in einer flexiblen und leicht erlernbaren Form erlaubt.

- Insbesondere für die nuklearphysikalischen Programme ist ein hohes Maß an Effektivität zu fordern. Es handelt sich hier um Programme, die bereits als "stand alone codes"moderne Computer bis an die Grenze ihrer Leistungsfähigkeit auslasten; dies gilt sowohl bezüglich Rechenzeit als auch der Verwaltung der Daten.

Die Erfüllung der zweiten Bedingung ist insbesondere eine Frage der logischen Datenstruktur. Hierzu gibt es nur wenige Lösungsansätze. Üblicherweise (z.B. in NUSYS /-45-/) beschränkt man sich auf die Verwaltung von linearen Feldern in diesem Interface, wobei das Interface nur den Namen dieser Felder, deren Länge und Ort kennt. Es ist den einzelnen Programmen überlassen, die Bedeutung, Struktur und Datendarstellung in diesen Feldern zu kennen und ggf. umzurechnen. Ein Interface, das in Form von Bäumen strukturiert ist, wurde in /-47-/ beschrieben. Ein Beispiel für die erforderlichen Algorithmen zur Umrechnung von Daten, die ein physikalisches Feld, definiert an den Knotenpunkten eines Maschennetzes, beschreiben, von der hexagonalen oder tridiagonalen Darstellung in rechtwinklige Maschennetze oder umgekehrt, ist in /-48-/ enthalten.

Bezüglich der anderen drei Bedingungen war zu untersuchen, inwieweit ICES hier zufriedenstellende Lösungen bereitstellt. Die Untersuchung wurde anhand einer prototypartigen Implementierung eines der in dem System zu integrierenden Programme durchgeführt. Es wurde also ein ICES-Subsystem FRED erstellt [49], das zunächst allein das bereits als Fortran-stand alone code existierende Programm THESYS [50] aufnahm.

Das von Doetschmann erstellte Programm THESYS dient [50] zur Ermittlung der 3-dimensionalen, stationären, nominellen Temperatur- und Durchsatzverteilung im Kernverband natrium- und gasgekühlter Schneller Reaktoren. Die in FRED integrierte Version stellte eine erste Ausbaustufe dieses Programms dar, die die Berechnung eines adiabaten Brennelements (kein Wärmetransport durch die Kastenwand) unter Schieflast erlaubt. Unabhängig von FRED wurde das Programm weiterentwickelt und ist nun auch u.a. in der Lage, die thermische Wechselwirkung eines Brennelements über die Kastenwände mit den Nachbarelementen zu berücksichtigen. Im einzelnen werden die Temperaturfelder für Kühlmittel, Hüllrohr, Brennstoff und die Kastenwand berechnet.

Bei der Integration von THESYS in FRED waren folgende Arbeiten durchzuführen:

a) die in FORTRAN-IV geschriebenen Programme waren so zu ändern, daß sie als ICETRAN-Routinen übersetzt werden konnten. Dies bedeutet,

   1) Elimination aller der Statements, die Sprachelemente von FORTRAN-IV enthielten, die in FORTRAN E nicht enthalten sind (siehe Anhang 2). Dies betraf vor allem logische Ausdrücke und das logische IF-Statement sowie DATA- Anweisungen; alle logischen Ausdrücke werden in arithmetische vom Typ Integer mit einem der Werte 1 oder 2 umgewandelt; die logischen IF-Anweisungen werden in arithmetische umgewandelt; DATA-Anweisungen wurden durch entsprechende Zuweisungen ersetzt.

   2) Die zweite Gruppe von Änderungen entstanden aus dem Wunsch, die dynamischen Felder in ICES, die DYNAMIC ARRAYs auszunutzen. Alle festdimensionierten Felder, deren logische Größe von der Eingabe abhängt, wurden durch entsprechend definierte DYNAMIC ARRAYs ersetzt. Hierbei ergaben sich an den Stellen Schwierigkeiten, an

denen im FORTRAN-H-Code einige der hier oft angewendeten und funktionsfähigen "Tricks" benutzt wurden, wie z.B. die Technik, einem Unterprogramm nur einen Teil eines Feldes zu übergeben:

```
DIMENSION  A (100)
 :
CALL  SUB  (A (51), 50)
 .
END
SUBROUTINE SUB (X,N)
DIMENSION  X(N)
 :
```

Hier mußten die Programme entsprechend umprogrammiert werden.

b) Die READ-Statements in THESYS mußten eliminiert werden. Die Eingabe wurde anstelle dessen in COMMON-Variablen und Feldern erwartet, die vom Command-Interpretierer entsprechend der POL mit den Eingabedaten gefüllt wurden.

c) Die Bedienungs- und Eingabesprache (POL) von FRED für die Benutzung des Programms THESYS mußte mit dem Subsystem CDL definiert werden. Zusätzlich mußten einige kleinere Unterprogramme geschrieben werden, die die Daten aus dem COMMON an die richtigen Feldplätze speicherten sowie einige Kontrollvariablen setzten, um die Vollständigkeit der Eingabe vor dem Beginn der eigentlichen Ausführung überwachen zu können.

Durch geeignete Definition der POL mit dem Subsystem CDL wurden folgende Spracheigenschaften erzielt:

- die Daten können dimensionsbehaftet eingegeben werden (Temperaturangaben als 573.16 K oder 200. C)

- die Abhängigkeit der Nusselt-Zahl NU von der Reynold- und Prandtl-Zahl sowie dem Verhältnis der mittleren zur Wand-Temperatur kann im ihrer üblichen Formelschreibweise eingegeben werden

- die einzelnen Daten können in beliebige Reihenfolge eingegeben werden; es wird hierbei vor dem Start der ausführenden Programme

die Vollständigkeit der Eingabe überprüft. Nachdem einmal alle Daten eingegeben wurden, brauchen für Parameterstudien nur die sich ändernden Daten neu eingegeben zu werden

- durch geeigneten Aufbau der Commands mit entsprechenden Füllwörtern (IGNORE-Wörter) ist die Eingabe gut lesbar.

Abb. 7 zeigt Beispiele der POL-Anweisungen und ihre Definition in CDL.

Mit FRED wurden folgende Erfahrungen gewonnen:

Eine POL mit den o.g. Eigenschaften ist in ICES mit der CDL machbar, wenngleich keine "leicht" und "schnell" lösbare Aufgabe. Insbesondere störte die mangelhafte Übersichtlichkeit der CDL-Blockstruktur und die Notwendigkeit von vielen kleinen ICETRAN-Hilfsprogrammen zum Umspeichern von Werten aus dem COMMON in DYNAMIC ARRAYs und zum Setzen von Kontrollvariablen.

Die Verwendung von DYNAMIC ARRAYs hat gezeigt, daß in FRED (im Gegensatz zu dem FORTRAN-Programm THESYS) die verarbeitbare Problemgröße nicht durch den verfügbaren Kernspeicher begrenzt ist und dieser Vorteil in ICES nahezu ohne zusätzlichen Programmier-Aufwand erreicht werden kann.

Die Umstellung von FORTRAN-IV auf ICETRAN hat (vorwiegend in der Testphase) mehrere Wochen gedauert, also relativ lang. Es war ein umfangreicher Lernprozeß erforderlich. Hierbei fehlte uns die Unterstützung durch einen erfahrenen ICES-Programmierer. An manchen kleinen Fehlern wurde viel Zeit verloren. Es muß allerdings betont werden, daß die dabei gesammelten Erfahrungen bei späteren Systementwicklungen voll genutzt werden konnten, so daß eine Wiederholung derselben Arbeit an FRED heute nur einen Bruchteil des Zeit- und Arbeitsaufwandes erfordern würde.

Zwar ist das Programm THESYS bei der Umstellung flexibler bezüglich der Problemgröße und einfacher in der Anwendung (durch POL) geworden, hat aber an Effektivität verloren. Die Rechenzeit ist um den Faktor 2 bis 5 größer geworden. Die Verweilzeit des Programms ist infolge der intensiven Ein- und Ausgabe zwischen Kernspeicher und Hintergrundspeicher ca. 5 mal so groß wie die Rechenzeit; bei durchschnittlichen FORTRAN-Programmen ist hierzu ein Faktor 2 üblich.

Aufgrund dieser - angesichts der extremen Anforderungen der Nuklearprogramme - nicht überzeugenden Ergebnisse konnte die weitere Verfolgung dieser Arbeitsrichtung nicht empfohlen werden. In der GfK wurde ein anderes Vorhaben mit ähnlichem Ziel gestartet, das in einem entsprechenden Programmsystem KAPROS [51] münden soll. KAPROS enthält einen eigenen in FORTRAN und Assembler programmierten Systemkern, der die gewünschten Funktionen bereitstellt. Eine Beschreibung dieses Systems wird 1972 vorliegen. FRED ist zudem bereits jetzt veraltet, weil das hier integrierte Programm THESYS inzwischen in seiner reinen FORTRAN-Version weiterentwickelt wurde [50].

3.4.2 TREE, Subsystem zur Definiton und Verarbeitung von Baumstrukturen

In zahlreichen Zusammenhängen treten Baumstrukturen auf [52]. Insbesondere sind Baumstrukturen bei der Verarbeitung von graphischen Informationen zu beschreiben, wie dies mit dem, im folgenden Kapitel beschriebenen Subsystem GRAPHIC möglich sein soll. In der GfK wurden daher Algorithmen zur Verarbeitung von Baumstrukturen in ICETRAN programmiert [40]. Die Routinen wurden zudem in anderen Routinen angewendet, um beliebig komplizierte DYNAMIC ARRAYs in ICETRAN als Ganzes auszudrucken, zu schreiben, zu lesen und zu kopieren [40, 41]. Im wesentlichen für das Testen dieser Routinen wurde ein ICES-Subsystem TREE erstellt, in dem es über eine entsprechend definierte POL möglich ist, beliebige Baumstrukturen zu definieren und abzuarbeiten. Es entspricht in seiner Aufgabenstellung daher dem Programm TREEPAK [53]. TREEPAK ist jedoch auf interaktive Eingabe der Baumstrukturen ausgerichtet, was die Existenz eines Displaygerätes mit Lichtgriffel voraussetzt, während in TREE eine Wortsprache zu diesem Zweck verwendet wird. Die einzelnen Knoten werden in TREE mit Namen angesprochen, die durch Zeichenketten dargestellt werden; die Zuordnung dieser Namen zu den Knotenindices, die intern verwendet werden, geschieht über HASH-Tabellen, für deren Verwaltung ebenfalls geeignete ICETRAN-Routinen erstellt wurden [54]. Außerdem enthält TREE einen Satz von Unterprogrammen, die auf Wunsch einen Unterprogramm-Trace liefern. Diese Einrichtung ist bei der Fehlersuche sehr hilfreich. An einer Erweiterung dieser Unterprogramme zu einer universellen Programmnachrichtenverwaltung wird gearbeitet.

Der Aufwand zur Definition der Testeingabe für diese Unterprogramme wurde durch die Verwendung einer POL sehr gering.

3.4.3 GRAPHIC, Ein Subsystem zur Manipulation von Abbildungen

GRAPHIC ist eine neues ICES-Subsystem, an dessen Erstellung gegenwärtig im IRE/GfK gearbeitet wird. Hier sollen kurz die Ziele und die angewandten Methoden skizziert werden, bei deren Realisierung sich ICES als gut geeignet erwiesen hat. Die folgenden Angaben definieren zunächst die Ziele:

Mit GRAPHIC ist es möglich, "Abbildungen" zu "manipulieren" und zu "verwalten". Unter Abbildungen werden hier graphische Informationen verstanden, die sich als zwei-dimensionale, schwarz-weiße Strichzeichnung darstellen lassen, wobei die Zeichnung beliebige Arten von Kurven und Symbolen, also auch Textzeichen enthalten kann. Als Manipulationsmöglichkeiten sind vorgesehen: Vereinigen von mehreren Abbildungen zu einer, Löschen von Abbildungen oder Abbildungsteilen, lineare Transformationen wie Vergrößern, Verdrehen, Verschieben von Abbildungen oder ihren Teilen, Erzeugung von Interpolations- und Approximationskurven, Beschriftung mit austauschbaren Texten (z.B. in unterschiedlichen Sprachen) usw. Mit Verwaltung ist gemeint: Abbildungen können in das System eingegeben werden und dann entweder temporär (für die Dauer der Manipulationen) oder permanent aufbewahrt und zu einem späteren Zeitpunkt ausgegeben werden. Hierbei werden die Abbildungen und ihre Einzelteile namentlich identifiziert. Das System ist in der Lage, über den Inhalt von Abbildungen Listen auszudrucken.

Als Eingabemöglichkeiten sind eine direkte Eingabe über eine POL (Wortsprache!) sowie die Übernahme der Ausgabe von anderen Programmen vorgesehen. Die Wortsprache berücksichtigt insbesondere die Forderung nach leichter Änderbarkeit der eingegebenen Informationen. So können graphische Objekte (die in ihrer Gesamtheit eine Abbildung bilden) namentlich identifiziert werden und mehrere Objekte als "Kollektion" unter einem Namen zusammengefaßt werden. Die Abbildungen können auf zwei wesentliche Arten definiert sein: a) als elementare Werte und b) als Erzeugungsvorschriften. Der Unterschied sei an einem Beispiel erläutert. Der Punkt 'A' sei Schnittpunkt der Linien 'B' und 'C'.

Wenn 'A' hier ein elementarer Wert sein soll, so verändert sich 'A' nicht, wenn später beispielsweise eine der Linien 'B' oder 'C' verschoben wird, dagegen bleibt 'A' stets der Schnittpunkt, wenn 'A' als Erzeugungsvorschrift definiert ist, unabhängig davon wie 'B' und 'C' verändert werden. In dieser Möglichkeit liegt ein wesentlicher Unterschied zu dem Subsystem COGO I.

In GRAPHIC wurde die problemorientierte Bedienungssprache in bestimmten Befehlen über die von der CDL gegebenen Möglichkeiten hinaus erweitert. Für die Beschreibung von Polygonzügen beispielsweise enthält die GRAPHIC-Sprache alle Möglichkeiten, die in ICETRAN gegeben sind, so z.B. Schleifen, Verzweigungen, arithmetische Ausdrücke usw. (siehe Kap. 2.7). Dies wurde dadurch realisiert, daß die ICES-Sprachprozessoren (Precompiler, Compiler) und die anderen Hilfsprogramme, die zur Erstellung eines Subsystems erforderlich sind (siehe Kap. 3.2) in das Subsystem GRAPHIC integriert wurden. Aufgrund entsprechender POL-Befehle wird ein Teil der POL-Eingabe nicht vom Kommando-Interpretierer verarbeitet, sondern zu einem vollständigen ICETRAN-Programm ergänzt, in einen Lade-Modul transformiert und sofort zur Ausführung gebracht.

Weiterhin ist vorgesehen, daß man die einzelnen Objekte in verschiedenen Bezugssystemen (beispielsweise eine Pumpe im natürlichen Raum, ein Diagramm in dem Raum, der durch die Koordinaten, wie z.B. Druck und Temperatur, definiert ist) definieren kann und daß lineare Transformationen, die auf Objekte angewandt werden, diese unterschiedlichen Bezugssysteme berücksichtigen [55].

Die zweite Eingabemöglichkeit ist für die Vielzahl von Benutzern gedacht, die in ihren Programmen unter Verwendung entsprechender Unterprogramme Zeichnungen auf Plottern ausgeben (z.B. via CALCOMP). Hier ist in der Regel eine Veränderung der Zeichnung nur möglich, wenn das Programm entweder sehr flexibel programmiert ist oder aber das Programm den jeweiligen Anforderungen entsprechend verändert wird. Diesen Benutzern wird ein Satz von Unterprogrammen bereitgestellt, die genau wie die sonst zum direkten Plotten üblicherweise verwendeten Unterprogramme aufgerufen werden, so daß im Benutzerprogramm keine Veränderung erforderlich ist, lediglich beim Erzeugen des Lade-Modul ist anstelle der Standard-Calcomp-Software die Bibliothek mit Graphic-Eingabe-Software

zu verwenden. Diese Unterprogramme übergeben dann die erzeugten graphischen Informationen an GRAPHIC weiter und der Benutzer kann dann dort seine Abbildung in der gewünschten Weise modifizieren und ausgeben. Auf diese Weise können Abbildungen, die von GRAPHIC-fremden Rechenprogrammen erzeugt werden, beispielsweise zum Zwecke der redaktionellen Gestaltung für eine Veröffentlichung, nachträglich bearbeitet werden.

Als Ausgabeinheit ist zunächst nur ein Plotter vorgesehen. Ein Interface zu Display-Geräten ist geplant.

GRAPHIC soll zunächst im Stapelbetrieb verwendet werden, später unter TSO auch interaktiv; letztere Möglichkeit wurde für andere ICES-Subsysteme bereits demonstriert [36].

Einen wesentlichen Teil der Programme bilden Routinen zur Verarbeitung von Baumstrukturen und Ringstrukturen. Diese sind insbesondere zur Darstellung der Erzeugungsvorschriften erforderlich. Die Verknüpfung der Objekte geschieht einerseits direkt durch entsprechende Tabellenindizes, andererseits über HASH-Tabellen mit den in [54] beschriebenen Algorithmen, soweit die Objekte namentlich verknüpft sind.

Die Verarbeitung der Eingabe geschieht in GRAPHIC ähnlich wie bei dem AED-System [56, 57]. Vom Interpretierer der Wortsprache wird für jeden Befehl zunächst eine Datenstruktur aufgebaut, die den Inhalt des Befehls in standardisierter Form präsentiert (im AED-System "first pass structure" genannt). Diese Befehlsstruktur wird von einem Parserprogramm (im AED-System "mouse" genannt [56]) durchlaufen und anhand von Tabellen interpretiert: Hier unterscheidet sich GRAPHIC vom AED-System. In GRAPHIC wird die Reihenfolge der Verarbeitung der Knoten der Struktur anhand von Tabellen bei diesem zweiten Durchgang bestimmt, während im AED-System diese Abfolge bereits beim Aufbau der "first pass structure" durch besondere Zeiger in die Struktur eingetragen wird. Das Ergebnis dieser Verarbeitung ist entweder eine Änderung der ursprünglich vorliegenden Datenstruktur, welche ein bestimmtes graphisches Objekt beschreibt, oder aber eine Darstellung der vorliegenden Information auf dem Drucker oder Plotter. In GRAPHIC wurden wesentliche Merkmale des AED-Systems übernommen, wobei jedoch die AED-Kernspeicher-

verwaltung [57] durch die Fähigkeiten des ICES-Systemkerns ersetzt wurde.

Die in GRAPHIC benötigten Programme werden in einem erweiterten ICETRAN programmiert. Es wurde ein Precompiler (in PL/1) erstellt, der diese Programme in reines ICETRAN umsetzt. Hierbei können u.a. logische Ausdrücke und IF-Anweisungen verwendet werden. Die Abbildungen dieses Berichts wurden mit GRAPHIC erzeugt.

4. Empfehlungen für den Einsatz von ICES

Aufgabe dieses Kapitels soll es sein, Interessenten und potentiellen Benutzern von ICES Hinweise dafür zu geben, unter welchen Voraussetzungen es für sie vorteilhaft und erfolgversprechend sein dürfte, ICES in ihrem Bereich einzusetzen, oder aber auch, unter welchen Bedingungen die Beschäftigung mit ICES und der Versuch es einzusetzen, wahrscheinlich ein Mißerfolg werden dürfte. Dabei ist eine Einschränkung zu machen. Die Verfasser dieses Berichtes haben keine eigenen Erfahrungen mit den spezifisch bautechnisch oder verkehrstechnik orientierten Subsystemen gesammelt. Interessenten auf diesem Gebiet, welches ja das eigentliche und ursprüngliche Einsatzgebiet von ICES ist, werden mit hoher Wahrscheinlichkeit unter der großen Zahl spezieller Subsysteme auch solche finden, die für ihre Zwecke geeignet sind. An dieser Stelle sollen besonders die universell einsetzbaren Subsysteme PROJECT und STRUDL, sowie die Möglichkeit der Programmentwicklung mittels CDL und ICETRAN behandelt werden.

4.1 Allgemeines

Eine Reihe von Bedingungen sollte in jedem Fall erfüllt sein, wenn ICES zum Einsatz kommen soll. Dies sind zunächst einmal Anforderungen an die Hardware und Software der EDV-Anlage: Der Benutzer muß über eine IBM-Anlage vom System 360 oder 370 vom Typ 360/40 aufwärts mit 128K-Bytes Kernspeicher oder mehr verfügen. Für langfristige Haltung von Programmen und Platten sind wenigstens zwei 2311-Platten erforderlich. Der Platzbedarf auf Platte (oder Trommel) richtet sich nach der Zahl der implementierten Subsysteme und den zu behandelnden Problemen. Ein Speicherbereich von etwa 10 Millionen Bytes (etwa 1/4 Platte 2314) dürfte in den meisten Fällen genügend Reserve lassen. Die Anlage muß

unter dem Betriebssystem OS (in einer seiner Versionen) betrieben werden.

Die Erfüllung allein dieser Vorbedingung wäre im Prinzip auch ausreichend, wenn nie Fehler auftreten würden. Es sei hier auf das verwiesen (in Kap. 2.7), was unter dem Beurteilungsgesichtspunkt Popularität gesagt wurde. Es genügt nicht, mit ICES nur nach Anweisung der Handbücher richtig umgehen zu können. Man muß auch in der Lage sein, die von ICES produzierten Fehlermeldungen richtig zu interpretieren, um einen in der Eingabe enthaltenen Fehler zu finden. Soweit Eingabefehler durch geplante Abfragen des Subsystems festgestellt wurden, ist die Fehlernachricht meist leicht verständlich. Häufig jedoch verursachen Eingabefehler erst im Laufe der weiteren Verarbeitung ein Versagen innerhalb des ICES-Systemkerns und eine Nachricht, die dem Subsystemanwender zunächst nicht verständlich ist. Ferner sind leider nach unserer Erfahrung in allen Subsystemen noch echte Programmfehler enthalten. Eine dritte Ursache für unerwartetes Verhalten kann Inkompatibilität der vom Magnetband übernommenen Lade-Module mit der auf einer Rechenanlage implementierten Version des Betriebssystems sein. (Solcher Art war z.B. das völlige Ausbleiben eines ICES-Kernspeicherauszuges auf der Anlage der GfK.) Um mit diesen Problemen des Fehlersuchens fertig zu werden, ist in einer Gruppe von ICES-Benutzern ein Mitarbeiter erforderlich, der gute Kenntnisse des Betriebssystems OS und der Assembler-Sprache besitzt und sich intensiv mit dem ICES-Systemkern befaßt. Darüber hinaus ist es erforderlich, das oft frustrierende Suchen nach oft sehr primitiven Fehlern dadurch abzukürzen, daß man mit anderen, möglichst nicht zu weit entfernten ICES-Benutzern Kontakt aufnimmt, die entweder mit dem Systemkern oder mit dem speziellen Subsystem bereits praktische Erfahrung gesammelt haben. Die Mitgliedschaft in der ICES Users Group ist zwar nicht notwendig, aber doch ratsam.

Es hat keinen Sinn, ICES als Ganzes oder eines seiner Subsysteme nur gelegentlich zu benutzen. In dieser Hinsicht verhält sich ICES wie jede andere Programmiersprache. Wer nicht regelmäßig damit umgeht, benötigt bei jedem Neubeginn eine neue Eingewöhnungsphase und wird nie den vollen Leistungsumfang der Subsysteme und die Erleichterungen der Programmhandhabung durch die problemorientierte Sprache voll ausschöpfen können.

4.2 Subsystemanwendung

An der Spitze der Leistungsfähigkeit steht ohne Zweifel STRUDL-II. Wer regelmäßig statische oder dynamische Festigkeitsanalysen im linear-elastischen Bereich an einfachen oder kompliziert zusammengesetzten ein- bis dreidimensionalen Bauwerken, Maschinen oder Apparaten durchzuführen hat, sollte sich unbedingt mit den Fähigkeiten von STRUDL-II vertraut machen. Der Kostenvorteil dieses Subsystems gegenüber anderen, kommerziell erhältlichen Programmen ist beträchtlich.

Das Subsystem PROJECT I sollte nur dort eingesetzt werden, wo die Projektplanung fast ausschließlich unter den Gesichtspunkten der Planung und Verfolgung von Terminen und Kosten steht. Wer darüber hinaus auch Zuweisung von Kapazitäten und Optimierungen von einem Programm durchführen lassen will, muß andere Programme heranziehen. Unter den kommerziell erhältlichen Programmen, die derart hohe Anforderungen erfüllen, ist das Subsystem PROJECT II , für das obige Einschränkungen nicht gelten, trotz der hohen Kosten sicher beachtenswert.

4.3 Programm- und Subsystementwicklung

Die Sprachen ICETRAN (zur Beschreibung der Funktion der ICES-Programme) und CDL (zur Definition der Bedienungssprache) können zusammen als ein System von höheren Programmiersprachen angesehen werden, das in Konkurrenz zu Fortran tritt. Es stellt sich nun die Frage: Wann sollte man ICETRAN und CDL benutzen, um ein neues Programm oder Programmsystem zu erstellen? Die Gesichtspunkte zur Beurteilung dieser Frage sind folgende:

- Anforderungen an Rechengeschwindigkeit
- Anforderungen an die Schnelligkeit der Programmerstellung
- Erwarteter Platzbedarf für Daten
- Erwarteter Kernspeicherbedarf der Programme (ohne Daten)
- Erwartetes Anwendungsspektrum
- Kompatibilität mit anderen Programmen
- Vertrautheit mit den Programmiersprachen.

Bei kleinen und einfachen Programmen, die nur gelegentlich benutzt werden sollen, spielt fast nur der letzte dieser Gesichtspunkte eine Rolle. Wir wollen hier Programme betrachten, die eine der folgenden Bedingungen erfüllen:

- der Kernspeicherplatz ist knapp oder es ist noch nicht sicher, ob nicht vielleicht in späteren Ausbaustufen des Programms der Kernspeicherplatz knapp wird

- an die Sicherheit und Überprüfbarkeit der Programmeingabe werden hohe Anforderungen gestellt.

Unter den genannten Gesichtspunkten soll ICETRAN/CDL mit FORTRAN und PL/1 hier nur kurz verglichen werden. Eine eingehende vergleichende Studie, die auch andere Sprachen und Systeme einbezieht, ist in Vorbereitung.

Programme mit extremen Anforderungen an Rechengeschwindigkeit, bei denen ein Faktor 5 signifikant ist, sollten nicht in ICES geschrieben werden.

Programme, die möglichst schnell erstellt werden sollen, schreibt man in der Sprache mit der man am besten vertraut ist. Dies spricht zunächst gegen ICETRAN/CDL. Sobald jedoch die Problemdaten so umfangreich werden, daß ein Teil der Daten auf externen Einheiten verwaltet werden muß, bietet ICETRAN über seine dynamischen Datenfelder (DYNAMIC ARRAY) ein einfach zu behandelndes Datenmanagement an, das es dem Programmierer erlaubt, mit einem virtuellen (d.h. scheinbar beliebig großen) Kernspeicher zu arbeiten. Im Gegensatz zu anderen Programmiersprachen braucht er sich über die Verwaltung der Daten also in diesem Fall (fast) ebensowenig Gedanken zu machen, wie wenn er externe Speicher gar nicht benötigte. Neben der Schnelligkeit der Programmerstellung wird dadurch ein erheblicher Gewinn an Sicherheit gegenüber Fehlern und eine deutliche Ersparnis an Testaufwand gewonnen.

Seltener, aber gerade bei großen Programmsystemen immer noch häufig genug, ist der Kernspeicher schon allein durch die Programme stark belegt, selbst wenn man Overlaytechnik anwendet. Die in ICES gegebenen Möglichkeiten der dynamischen Programmverknüpfung bieten für diese Situation eine sichere und leicht zu handhabende Lösung.

Für ein Programm, das fast nur von seinem Ersteller benutzt werden wird, spielt die Möglichkeit, eine problemorientierte Sprache dafür zu definieren, keine Rolle. Sie ist aber auch kein Nachteil, denn auch ICETRAN-Programme können wie FORTRAN-Programme ihre Eingabe mit Format lesen, wenn man keine spezielle Sprache dafür entwickeln will. Eine problemorientierte Sprache ist dann nützlich (man könnte fest sagen: notwendig), wenn an einen größeren Benutzerkreis gedacht ist. In diesem Fall ist ICES deutlich vorzuziehen, auch wenn die Erstellung der problemorientierten Sprache natürlich einen zusätzlichen Aufwand erfordert.

An dieser Stelle muß hervorgehoben werden, daß die Bequemlichkeit und Einprägsamkeit einer problemorientierten Sprache nicht nur unter dem Aspekt der Zeitersparnis beim Lernen und beim Erstellen einer Programmeingabe gesehen werden darf. Wichtiger ist vielleicht sogar der Aspekt der Sicherheit. Wenn ein Programm die Angabe einer Temperaturangabe nicht nur in Celsius, sondern auch in Kelvin oder Fahrenheit "versteht", so entfallen in vielen Fällen bei der Vorbereitung eines Programmlaufes risikobehaftete Umrechnungen. Auch die Tatsache, daß eine Überprüfung der sachlichen Richtigkeit einer Eingabe erheblich einfacher ist, wenn diese Eingabe in problemnahen, sprachlich gebundenen Sätzen vorliegt (und nicht nur als Zahlentabelle), trägt zur Sicherheit bei. Programme, an die hohe Sicherheitsanforderungen gestellt werden, sollten daher das in ICES gegebene Mittel der POL einsetzen.

Die Verwandtschaft mit der Programmiersprache FORTRAN kann in ICETRAN nicht immer genutzt werden. Zwar ist es möglich, FORTRAN-Unterprogramme auch von ICETRAN-Routinen aufzurufen, jedoch nur bei Einhaltung gewisser einschränkender Regeln (siehe Anhang 3). Zudem können FORTRAN-Unterprogramme nicht mit DYNAMIC ARRAYs als Argument aufgerufen werden, d.h. einige der Vorteile von ICETRAN gehen bei der Verwendung von FORTRAN-Unterprogrammen verloren. Die genannten Nachteile von ICES als Basis einer Programmentwicklung gegenüber FORTRAN (geringere Rechengeschwindigkeit und Kompatibilität) dürften beim Einsatz von PSU-ICES [8,9] fast vollständig entfallen.

Referenzen

[1] B.Schumacker (ed.):
An Introduction to ICES, MIT, Department of Civil Engineering, R 67-47, Sept., 1967

[2] D.Roos (ed.):
ICES System - General Description, MIT, Department of Civil Engineering, R 67-49, Sept., 1967

[3] J.Sussman (ed.):
ICES Subsystem Development Primer, MIT, Department of Civil Engineering, R 68-56, May, 1968

[4] D.Roos:
ICES System Design, MIT, Department of Civil Engineering, The M.I.T. Press, 2nd. ed., 1967

[5] RCA Computer Systems:
ICES Integrated Civil Engineering System
Information Manual, 70-03-039 (October 1970)

[6] McDonnell Douglas Automation Dept. K 660-I
P.O. Box 516, St.Louis, Missouri
(Persönliche Information)

[7] F.E.Hajjar (ed.):
ICES Users Group Newsletters Vol.3, No.3
Worcester Area College Computation Center
Worcester Polytechnic Institute, Worcester, Mass., (Nov.1971)

[8] M.A.Prichard:
PSU-ICES 1, Implementation Features and Philosophy of the Penn State University Integrated Civil Engineering System-Version 1
College of Engineering, PSU, University Park, Pa. 16802, CAD LAB Report Nr. 71-3 (June 1971)

[9] R.E.Rien:
Overview of PSU ICES-1
College of Engineering, PSU, University Park, Pa. 16802, CAD LAB Report No. 71-2 (June 1971)

/10/ M.A.Prichard:
ICES Support for G/H Level FORTRAN,
PSU ICES 1 Progress Report
Vortrag auf Eighth Semi Annual ICES Users Group Conference,
San Francisco, 20.-21.1.1972

/11/ D.G.Alcock, B.H.Shearing et.al.:
GENESYS Reference Manual
The GENESYS Centre, University of Technology, Loughborough,
Leicestershire (March 1970, mit "Improvements" October 1970)

/12/ R.L.Daniels:
Project/2 - Sample Run Book 1 to 4
Project Software and Development, Inc.
8A Elliot Street/Cambridge, Massachusetts 02138

/13/ R.D.Logcher, J.N.Jackson:
ICES TABLE II - Engineering User's Manual
First Edition, R69-34, MIT (June 1969)

/14/ R.D.Logcher, C.M.Power, H.L.Kwok:
ICES TABLE I - Engineering User's Manual
First Edition, R67-58, MIT (Sept.1967)

/15/ F.Ochoa-Rosso, D.K.Bivins, H.Thiriez:
ICES OPTECH I - Engineering User's Manual
First Edition, R68-65, MIT (Aug. 1968)

/16/ R.J.Clasen:
The MFOR Program
The RAND Corporation, Santa Monica (Nov. 1962)

/17/ R.J.Clasen:
RS OKF1, Out of Kilter Network Flow Roatine One
The RAND Corporation, Santa Monica (March 1971)

/18/ M.M.Flood, A.Leon:
A Generalized Direct Search Code for Optimization
Mental Health Research Institute, Univ. of Michigan

/19/ R.Young:
A Primal (all-integer) Integer Programming Algorithm
Journal of Research of the National Bureau of Standards B, Mathematics and Mathematical Physics, Vol. 69B, No.3 (July-Sept. 1965), pp.213-250

/20/ E.G.Schlechtendahl:
DYSYS - A Dynamic System Simulator for Continuous and Discrete Changes of State
KFK 1209 (Juli 1970)

/21/ H.-P.Schwefel:
Evolutionsstrategie für die numerische Optimierung
Bericht des Instituts für Meß- und Regeltechnik der TU Berlin (1971)

/22/ Engineers Guide to ICES COGO I
First Edition, R67-46, MIT (1967)

/23/ J.W.Gunther:
COGO Input Using CDL Subroutines
Vortrag auf Eighth Semi Annual ICES Users Group Conference, San Francisco, 20.-21.1.72

/24/ R.L.Daniels, E.J.Hall:
ICES PROJECT I - General Description
First Edition, R68-19, MIT (March 1968)

/25/ R.L.Daniels, B.Bayer, E.J.Hall, M.H.Roy, Jr.:
ICES PROJECT I
Engineering User's Manual
Second Edition, R68-11, MIT (August 1968)

/26/ R.D.Logcher, et.al.:
ICES STRUDL II
Engineering User's Manual, Vol. 1
Frame Analysis
First Edition, R68-91 (November 1968)

[27] R.D.Logcher, J.J.Connor, M.F.Nelson:
ICES STRUDL II
Engineering User's Manual, Vol. 2
Additional Design and Analysis Facilities
Second Edition, R70-77, MIT (June 1971)

[28] ICES STRUDL II
Engineering User's Manual, Vol. 3
Reinforced Concrete Structures
Second Edition, R70-35, MIT (June 1970)

[29] ICES STRUDL I
Engineering User's Manual
R67-56, MIT (1967)

[30] S.J.Ferves, R.D.Logcher, S.P.Mauch:
STRESS: A Reference Manual
MIT Press, 1965

[31] R.D.Logcher:
Analysis of Nuclear Reactors
S.Britten, M.Pope:
Deflection Analysis of Large
Mirrors Using STRUDL
Vorträge auf der 8th Semi-Annual ICES Users Group Conference,
San Francisco, 20.-21.1.1972

[32] U.Bozzo:
A Comparison of Different Approaches to Finite Element
Calculations [STRUDL, SAFE, GC 21]
Newsletters of the ENEA Computer Program Library, No. 12,
(Oct.1971),pp.151-155

[33] J.M.Sussman:
Primary Memory Management in ICES
MIT, Department of Civil Engineering, CESL Report R67-68(1967)

[34] IBM-Contributed Program Library
ICES/360 Source Basic Sytem and Laguage Processors V1M1
(Correction 7/69); Progr.Ord.No. 360D- .03.0.005 (1968)

/35/ R.Logcher: Time-Sharing STRUDL
Vortrag auf der Eighth Semi-Annual ICES Users Group Conference, San Francisco, USA, 20.-21.1.72

/36/ A.Herbert:
ICES Under TSO (Time-Sharing Option) at L.S.U.
Vortrag auf der Eighth Semi-Annual ICES Users Group Conference, San Francisco, USA, 20.-21.1.72
(LSU College of Engineering, Deans Office, Baton Rouge, Louisiana 70803)

/37/ J.C.Jordan (ed.):
ICES-Programmers Reference Manual, MIT, Department of Civil Engineering, R67-50, Oct.1967

/38/ IBM Operating System / 360:
FORTRAN IV (E Level Subset)
Form C28-6513
sowie
IBM System / 360 Operating System
Basic FORTRAN IV (E) Programmer's Guide
Form C28-6603

/39/ F.Bates, M.L.Douglas, R.Gritsch:
PL/1 Programming Language / One, Einführung in die Programmiersprache für den Selbstunterricht
2.Auflage, Carl Hanser-Verlag, München, 1970

/40/ U.Schumann, E.G.Schlechtendahl:
Algorithmen zur Verarbeitung von Baumstrukturen und ihre Anwendung in ICES. - KFK 1536, Januar 1972

/41/ U.Schumann, E.G.Schlechtendahl:
ICETRAN-Treestructure Routines to Save Direct Access Space and Debugging Time
Vortrag gehalten auf der Eighth Semi-Annual ICES Users Group Conference, San Francisco, USA, 20.-21.1.72,(Manuskript bei den Autoren erhältlich)

/42/ IBM System / 360 Operating System: Linkage Editor and Loader
GC 28-6538

/43/ H.Wettstein:
Die Organisation von Programmüberlagerungen als Teilaufgabe des Binders
Elektron. Rechenanlage. 13 (1971), 170-177

/44/ IBM System / 360 Operating System
Supervisor and Data Management Macro Instructions
C28-6647

/45/ **Institut für Neutronenphysik und Reaktortechnik**
Nukleares Programmsystem NUSYS.
Unveröffentlichte INR-Arbeitsberichte der Gesellschaft für Kernforschung, Karlsruhe

/46/ ENEA Computer Program Library, ISPRA, Italien (Veranstalter):
Symposium über modulare Programme zur Reaktorberechnung, 1.-3.12.1970, Varese/Italien, Proposed Definition of Terms (unveröffentlicht)

/47/ J.E.Suich, J.C.Jensen, H.C.Honeck:
Data Management for **Multiaccess Computational Systems**
aus: B F.Maskewitz u.a. (ed.): Proc. Conf. Effective Use of Computer in Nucl. Industry, Knoxville, Tennessee, April 21.-23.,1969, TID 4500, Conf.-690401, S.471-481

/48/ E.G.Schlechtendahl:
HEXAGON - Eine Systematik zur Behandlung von Problemen in Sechseckanordnungen. - KFK 1432, Juli 1971

/49/ U.Schumann:
Programmsystem für technischen Reaktorentwurf (FRED)
Projekt Schneller Brüter (Hrsg.): 2.Vierteljahresbericht 1970, KFK 1270/2, Oktober 1970, S.128-3
sowie: 4.Vierteljahresbericht 1970
KFK 1270/4, Dezember 1970, S.128-9

/50/ A. Amendola, K.Doetschmann, F.Hofmann:
Theoretische Arbeiten zum Reaktorkern-Entwurf
Projekt Schneller Brüter (Hrsg.) 2.Vierteljahresbericht 1971
KFK 1271/2
sowie 3.Vierteljahresbericht 1971, KFK-1271/3
und 4.Vierteljahresbericht 1971, KFK-1271/4, S.129-1

/51/ Gesellschaft für Kernforschung mbH., Karlsruhe:
Bericht über Forschungs- und Entwicklungsarbeiten im Jahre 1970,
Kap. 4: INR, PSB 1222.3, 5.68, Oktober 1971

/52/ D.E.Knuth:
The Art of Computer Programming Vol.1: Fundamental
Algorithms, Chapter 2, Addison-Wesely Publ., Reading Mass.
(1969)

/53/ C.M.Holifield:
Graphic display and manipulation of tree-type data structures
Naval Postgraduate School AD-709091, Dec.1969

/54/ U.Schumann:
Verwaltung von dynamischen Symbol-Tabellen nach der HASH-
Methode mit ICETRAN-Unterprogrammen
Kerforschungszentrum Karlsruhe, Externer Bericht 8/71-7

/55/ U.Schumann:
Systematische Unterteilung von grafischen Objekten nach
ihrem Bezugssystem und deren gemeinsame Darstellung
Gesellschaft für Informatik e.V., W.Giloi (Hrsg.):
Bericht Nr. 2, Symposium über Computer Graphics,
Berlin 19.-21.Oktober 1971, S.51-60

/56/ D.T.Ross:
A Generalized Technique for Symbol Manipulation and
Numerical Calculation
Comm. ACM 4(1961), pp.147-150

/57/ D.T.Ross:
The AED approach to generalized computer-aided design
Proc. ACM 2nd Annual Conf. (1967), pp.367-385

/58/ IBM System / 360 Operating System:
Concepts and Facilities - GC 28-6535

Anhang 1

Definition einiger Begriffe aus der Terminologie des IBM-Systems/360-OS [58]

Für das Verständnis der ICES-Erläuterungen ist die Kenntnis einiger Eigenheiten des Betriebssystems OS/360 wichtig. Aus diesem Grunde werden hier diejenigen Begriffe aus dem OS, die immer wieder benutzt werden, kurz erläutert.

Ein Modul (module) ist eine selbständige Programmeinheit, die durch einen Namen identifiziert ist und die Ein- oder Ausgabe einer einzelnen vollständigen Ausführung eines Compilers, Assemblers oder Linkage Editors ist.

Ein Quell-Modul (source module) ist die Eingabe zu einer Compiler- oder Assembler-Ausführung; er besteht aus einer Folge von Anweisungen in einer symbolischen Sprache.

Ein Objekt-Modul (object module) ist die Ausgabe einer einzelnen Ausführung des Compilers oder Assemblers und kann u.a. Eingabe für den Linkage Editor (im deutschen Sprachraum auch Binder genannt) sein. Ein Objekt Modul enthält eine oder mehrere Kontroll-Sektionen (control-sections) in einer nicht ausführbaren Form.

Ein Lade-Modul ist die Ausgabe des Linkage Editors und ist ein Programm, das in dieser Form in den Kernspeicher geladen und zur Ausführung gebracht werden kann. Ein Lade-Modul unterscheidet sich vom Objekt-Modul einerseits im Format, andererseits dadurch, daß solche Adreßkonstanten, die Beziehungen zwischen den Kontroll-Sektionen innerhalb des Lade-Moduls herstellen, nur im Lade-Modul mit den richtigen Werten gefüllt sind.

Eine Kontroll-Sektion (control section) ist die kleinste Einheit eines Programms; sie enthält die vom Programmierer definierten Befehle und Daten und muß als zusammenhängender Block gespeichert sein.

Ein Lade-Modul kann in mehrere Segmente aufgeteilt sein. Ein Segment ist die kleinste funktionelle Einheit (bestehend aus einer oder mehreren Kontroll-Sektionen), die als eine logische Einheit während der Ausführung eines Overlay-Programms geladen werden kann.

Ein Overlay-Programm ist ein Lade-Modul mit mehreren Segmenten, die einzeln in den Kernspeicher geladen werden können und dabei an Orte im Kernspeicher gelangen können, an denen zuvor ein anderes Segment des gleichen Lade-Moduls sich befunden hat.

Ein Lade-Modul besitzt mindestens einen, durch einen symbolischen Namen identifizierten Entry Point. Ein Entry Point eines Lade-Moduls ist die Stelle des Lade-Moduls, der die Kontrolle von einem anderen Lade-Modul übergeben werden kann.

Anhang 2

Eigenschaften von FORTRAN IV (G/H), die nicht in FORTRAN E enthalten sind [38]

The following statements in FORTRAN IV are not in FORTRAN (E Level Subset):

ASSIGN
BLOCK DATA
Labeled COMMON
COMPLEX
DATA
More than three dimensions
Adjustable dimensions
Assigned GO TO
Logical IF
LOGICAL
PRINT
PUNCH
READ b, list
END and ERR parameters in a READ
Generalized Type statement (But note that DOUBLE PRECISION is provided as an explicit type)
IMPLICIT
Call by name
Literal as argument of CALL
ENTRY
RETURNi (i not a blank)
NAMELIST
PAUSE with literal
G and L format codes

The following in-line subprograms in FORTRAN IV are not in FORTRAN (E Level Subset):

REAL
AIMAG
DCMPLX
CMPLX
DCONJG
CONJG
HFIX
CABS

The following out-of-line subprograms in FORTRAN IV are not in FORTRAN (E Level Subset):

CEXP
CDEXP
CLOG
CDLOG
CLOGIO
CDLGIO
CSIN
CDSIN
CCOS
CDCOS
CSQRT
CDSORT
DATAN2

Anhang 3

Regeln für die Verwendung von FORTRAN G/H-Programmen in ICES

FORTRAN G/H-Programme (SUBROUTINE oder FUNCTION) können in ICES in zwei Formen verwendet werden

a) als Teil eines Lade-Moduls, zusammen mit ICETRAN-Unterprogrammen

b) als selbständige Lade-Module, die von ICETRAN-Lade-Modulen über die Anweisungen gemäß Kap. 3.1.4 aufgerufen werden.

Hierbei sind folgende Regeln zu beachten:

Form a):

Die FORTRAN G/H-Unterprogramme können dem Linkage Editor auf zwei Wegen eingegeben werden:

- als Lade-Module über die DD-Karte SYSLIB per "automatic library call" /58/; hierbei müssen die Lade-Module, die den FORTRAN-G/H-Code enthalten mit den Linkage Editor-Parametern ''NCAL,REUS' erzeugt worden sein. Eine besondere Eingabe zu QQSETGEN, die sich auf die Namen dieser Unterprogramme bezieht,ist nicht erforderlich

- als Objekt-Modul über die DD-Karte INCLIB; hierbei sind die Namen der Unterprogramme in der Eingabe zu QQSETGEN aufzuführen, und zwar in der Gruppe der Programme ohne Common, die nicht Entries des Lademoduls sind.

Derart in ICETRAN-Lade-Module integrierte FORTRAN G/H-Routinen müssen folgenden Einschränkungen genügen: sie dürfen

- keine Icetran-Befehle enthalten (auch nicht in der Form des Aufrufs entsprechender ICES-Kernroutinen)

- keinen COMMON verwenden

- nicht IBCOM aufrufen, d.h. vor allem kein STOP und keine I/O-Anweisungen enthalten

(unter den gleichen Voraussetzungen können auch Assembler-Routinen in ICETRAN-Lade-Modulen verwendet werden.)

Form b):

Von ICETRAN können Lade-Module aufgerufen werden, die in beliebiger Programmiersprache programmiert wurden, also auch FORTRAN G/H. In diesen Lade-Modulen müssen alle Referenzen auf externe Symbole vom Linkage Editor gelöst worden sein.

Weiterhin sind folgende Regeln zu beachten:

- Die Argumentenliste der G/H-Routinen muß, falls überhaupt erforderlich, mit zwei Dummy-Argumenten beginnen. Diese werden beim Aufruf automatisch mit dem Unterprogrammnamen und der Common-Andresse gefüllt, deren Verwendung in der Regel nicht sinnvoll sein dürfte.

- Das Fortran G/H-Unterprogramm muß mit allen seinen Unterprogrammen zu einem Lade-Modul gelinkt werden und in eine Bibliothek geladen werden, die im G-Step als JOBLIB oder STEPLIB zur Verfügung steht. Hierbei ist dem Lade-Modul der Name (z.B. XXXXX) als ENTRY und der mit dem Prefix QQ versehene Name (im Beispiel also QQXXXXX) als Lade-Modul NAME zuzuordnen, wozu im Beispiel folgende Anweisungen für den Linkage Editor erforderlich sind:
  bENTRYbXXXXX
  bNAMEbQQXXXXX

- Von Icetran-Programmen kann der Fortran G/H-Lade-Modul nur über LINK bzw. LØAD/BRANCH erreicht werden. Hierbei dürfen für die beiden Dummy-Argumente keine aktuellen Werte eingesetzt werden.

- Es muß sichergestellt werden, daß solche Dateien, die sowohl von den ICES-Routinen als auch von dem Fortran G/H-Lade-Modul benötigt werden, vor dem LINK bzw. BRANCH und vor dem RETURN im Fortran G/H-Lade-Modul geschlossen (CLOSE) werden. Andernfalls endet der Job mit dem System-Completion-Code 513 (doppeltes Open). Dieses Schließen kann in Fortran-Programmen mit dem Befehl "REWIND n" erreicht werden; n ist hierbei die Fortran-File-Nummer.

  Es ist zu beachten, daß im Falle eines Fehlers, der vom System entdeckt wird, von den Systemroutinen, wie z.B. IBCOM, Nachrichten auf den Fortran-File 6 geschrieben werden. Dieser ist daher tunlichst auch dann zu schließen, wenn kein explizites WRITE auf File 6 in dem Fortran G/H-Programm vorgesehen ist.

Anhang 4

Liste häufig benutzter Abkürzungen

AED	Automated Engineering Design (siehe [57])
CDL	Command Definition Language
CESL	Civil Engineering Systems Laboratory of MIT
CI	Command Interpreter
DA	DYNAMIC ARRAY
EDV	Elektronische Datenverarbeitung
FRED	Fast Reactor Evaluation and Design (siehe 3.4.1)
GENESYS	General Engineering System
GRAPHIC	System zum Manipulieren von Abbildungen (siehe 3.4.3)
ICES	Integrated Civil Engineering System
ICESSUBSYSTEME	
BRIDGE	Bridge Design System
CDL	Command Definition Language
COGO	Coordinate Geometry Subsystem
LEASE	Limiting Equilibrium Analysis of Slopes and Embankments
OPTECH	Optimization Techniques System
PROJECT	Project Engineering Control
ROADS	Roadway Analysis and Design System
SEPOL	Settlement Problem Oriented Language
STRUDL	Structural Design Language
TABLE	Tabular Data Creation and Manipulation
TRANSET	Transportation Network Analysis
TRAVOL	Traffic Volume Data Subsystem
ICESDATEIEN	
DICT	Dictionary
MODS	Modules
SUBS	Subsystem-Data Sets

USER	User-Data Sets
FUNCLIB	Function Library
MACLIB	Macro Library
ICETRAN	ICES-FORTRAN
ISB	Informationssystem Bauwesen
JCL	Job Control Language
KAPROS	Karlsruher Programm-System
LSU	Louisiana State University
MIT	Massachusetts Institute of Technologie
NUSYS	Nukleares Programmsystem
OS / 360	Operating System of IBM/360
POL	Problem Oriented Language
PSU	Pennsylvania State University
REGENT	System für den rechnergestützten Entwurf
THESYS	Thermal Design of Nuclear Reactors (siehe 3.4.1)
TREE	Baumstruktursubsystem in ICES (siehe 3.4.2)
TSO	Time Sharing Option des Betriebsystems OS/360

Anhang 5

Übersicht über die in ICETRAN zusätzlich zu den FORTRAN-Anweisungen möglichen Befehle /-2-/

Dynamic Data Statements

DYNAMIC ARRAY $a_1(k_1)$, $a_2(k_2)$,..., $a_r(k_r)$,..., $a_n(k_n)$

DEFINE $a(i_1,i_2,...,i_n),s_1,...,s_m$,type, priority, growth

RELEASE $a(i_1, i_2,..., i_n)$, priority

DESTROY $a(i_1, i_2,..., i_n)$, size

SWITCH ($a(i_1,..., i_n)$, $b(j_1,..., j_n)$)

COMMON $a(k_1)$, $a_2(k_2)$,..., $a_n(k_n)$

Literal Constant Definition

v='e'

Program Structure Statements

SUBROUTINE name (a_1, a_2,..., a_n)

FUNCTION name (a_1, a_2,..., a_n)

mode FUNCTION name (a_1, a_2,..., a_n)

CALL name (a_1,..,a_n)

LINK name (a_1,..,a_n)

TRANSFER name

LOAD name (ENTRY)

DELETE name

BRANCH name (ENTRY, a_1,..., a_n)

DYNAMIC PROGRAM $name_1$, $name_2$,..., $name_n$

OVERLAY PROGRAM $name_1$, $name_2$,..., $name_n$

Program Stack Statements

ADD TO STACK (count, name)

DELETE FROM STACK (count)

COPY FROM STACK (count, name)

TRANSFER TO STACK

LINK TO STACK

Error Processing Statements

INHIBIT

ENABLE

ERROR RETURN

Disk Control Statements

DISK LENGTH (NDS, recid, i, next, last)

DISK GET (NDS, recid, area, L_1, L_2, i, next, last)

DISK OPEN (NDS, 'filnam', n, b)

DISK PUT (NDS, 'filnam', recid, area, L_1, L_2, next, last)

DISK CLOSE (NDS, 'filnam')

DISK DELETE (NDS, 'filnam', recid)

DISK FILE INFO (NDS, 'filnam', i, first, last)

DISK SET INFO (NDS, j, k, m)

DISK FILE RENAME (NDS, 'oldnam', 'newnam')

DISK FILE PROTECT (NDS, 'filnam')

DISK FILE ALLOW (NDS, 'filnam')

Tabelle 1

Mitglieder der ICES-User's Group Ende 1971 [7]

Gesamtzahl der aktiven Mitglieder: 400

UNITED STATES	251
Alabama	6
Arizona	1
Arkansas	1
California	29
Colorado	1
Connecticut	4
Delaware	2
District of Columbia	6
Florida	2
Georgia	3
Hawaii	3
Illinois	15
Indiana	2
Iowa	2
Kansas	3
Kentucky	3
Louisiana	4
Maine	2
Maryland	3
Massachusetts	25
Michigan	8
Minnesota	1
Missouri	6
Nebraska	4
Nevada	1
New Hampshire	1
New Jersey	7
New Mexico	1
New York	22
North Carolina	3
Ohio	9
Oklahoma	3
Oregon	4
Pennsylvania	21
South Carolina	2
Tennessee	5
Texas	17
Vermont	1
Virginia	6
Washington	5
West Virginia	4
Wisconsin	3

OTHER COUNTRIES	149
Australia	16
Austria	2
Brazil	3
Canada	55
Canal Zone	1
Costa Rica	1
England	9
France	9
Germany	4
Ireland	1
Italy	8
Japan	5
Mexico	1
Netherlands	7
New Zealand	4
Norway	2
Puerto Rico	1
Saudi Arabia	2
South Africa	4
Spain	3
Sweden	6
Switzerland	4
Venezuela	1

Tabelle 2

MIT-ICES-Subsysteme (Stand Juni 1971)

Subsystem	Version	Ausgabedatum
Systemkern (einschl. CDL)	V 1 M 4	15.3.71
COGO I	V 1	15.2.71
OPTECH	V 1 M 1	1.10.70
PROJECT I	V 1 M 1	1.10.70
ROADS	V 1 M 1	30.10.70
STRUDL II	V 2 M 0	30.11.70
TABLE I	V 1 M 0	1.10.70
TABLE II	V 1 M 0	1.10.70
TRANSET I	V 1 M 3	31.7.70
STRUDL II Update	V 2 M 0	23.4.71

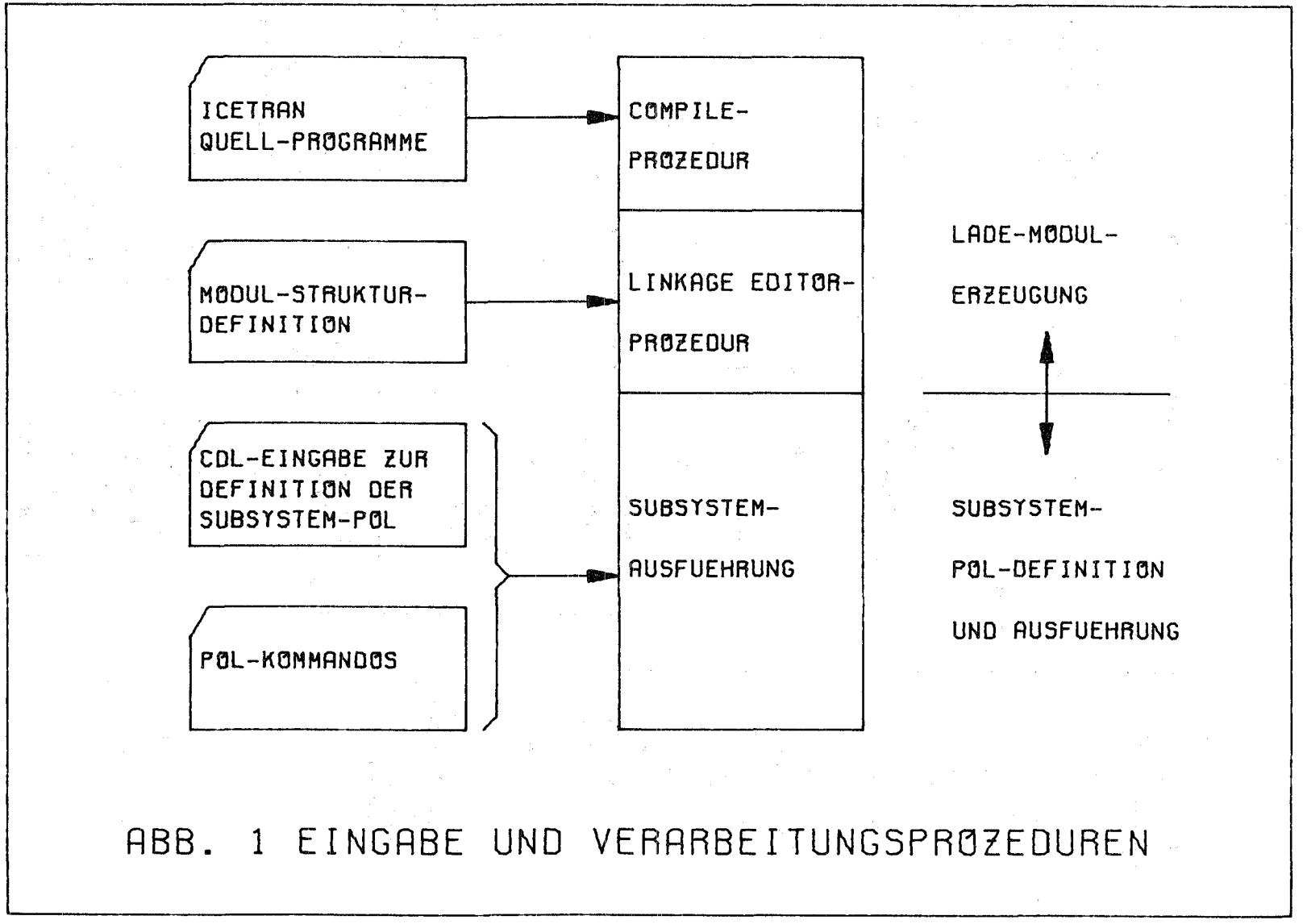

ABB. 1 EINGABE UND VERARBEITUNGSPROZEDUREN

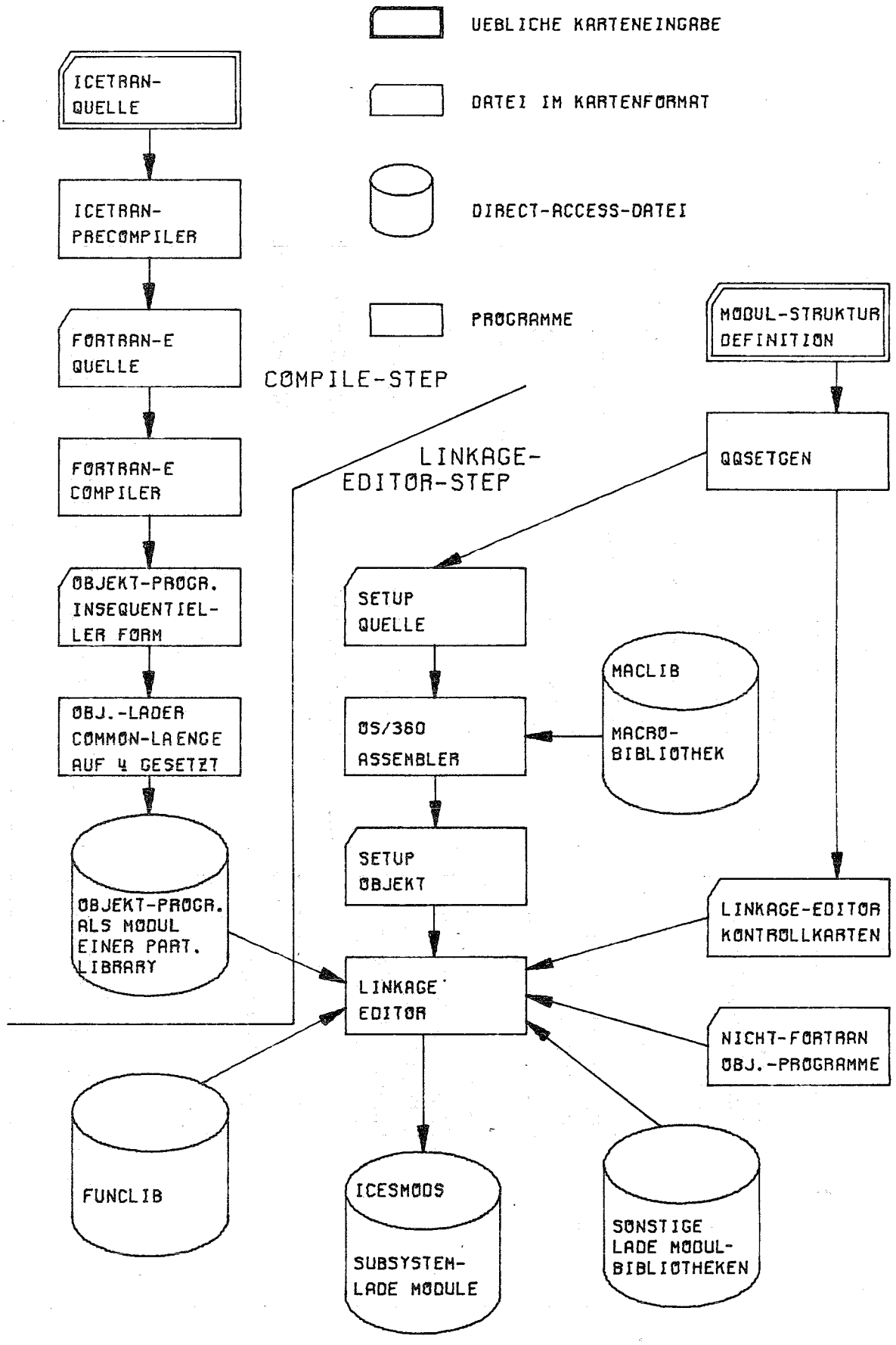

ABB.2 DATEIEN, PROGRAMME UND INFORMATIONSFLUSS DER GENERIERUNG VON SUBSYSTEM LADE-MODULEN

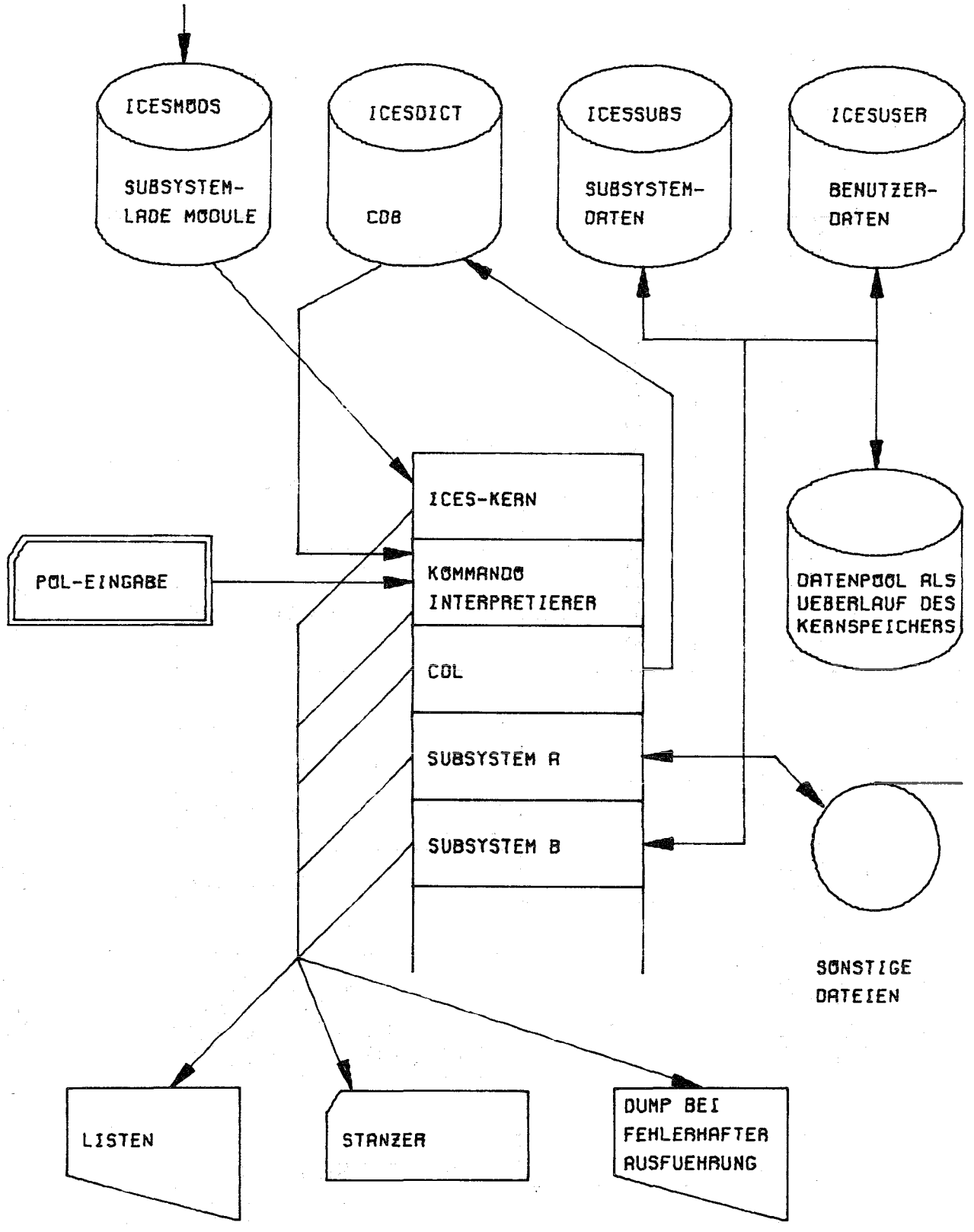

ABB.3 DATEIEN, PROGRAMME UND INFORMATIONSFLUSS DER SUBSYSTEM-AUSFUEHRUNG

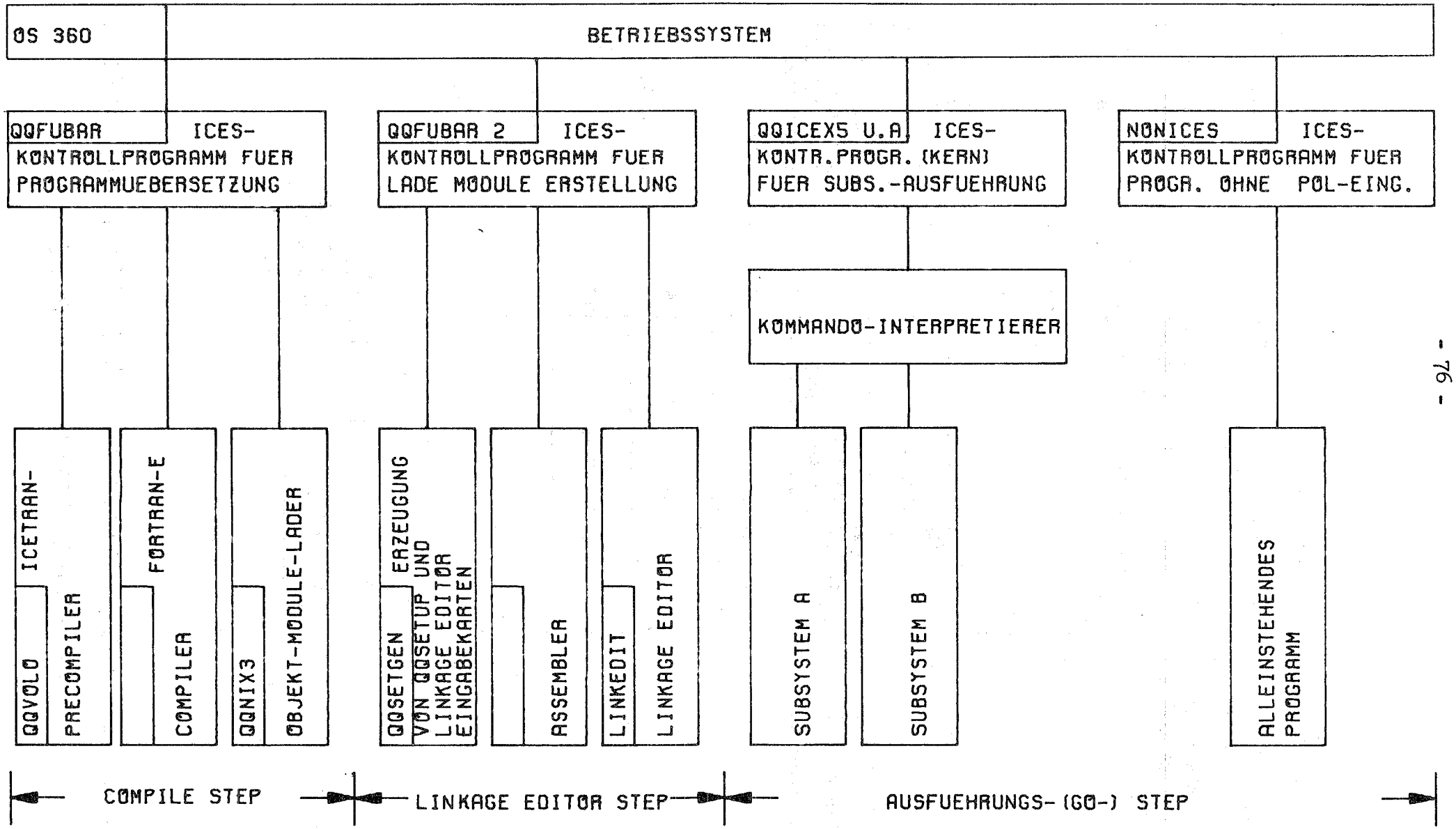

ABB.4 ICES-PROGRAMMHIERACHIE

```
//ICCLG  PROC PREFIX=GR,BLK=931,QQQ=QQQICEX3,DUMFIL=NULLFILE
//C EXEC PGM=ICESCOMP,PARM=&PREFIX,REGION=240K,TIME=2
//STEPLIB DD DSN=ICES.MODS,UNIT=3330,VOL=SER=NUSICE,DISP=SHR
//FILE2 DD UNIT=SYSDA,SPACE=(7260,1600),DSN=&DUMFIL
//FILE3 DD UNIT=(CTC,,DEFER)
//FILE4 DD UNIT=SYSDA,SPACE=(80,(8000,4000))
//FILE5 DD UNIT=SYSDA,SPACE=(1600,(100,50))
//PRINT DD UNIT=(CTC,,DEFER)
//SYSPRINT DD UNIT=(CTC,,DEFER),DCB=(BLKSIZE=968,RECFM=FBM,LRECL=121)
//SYSUT1 DD UNIT=SYSDA,SPACE=(3600,(40,10))
//SYSUT2 DD UNIT=SYSDA,SPACE=(3600,(40,10))
//SYSLIN DD UNIT=SYSDA,SPACE=(3200,(100,40)),DCB=*.FILE2
//OUTPUT DD DSN=&&OBJ,DCB=(BLKSIZE=1680,LRECL=80,RECFM=FB),UNIT=SYSDA, *
//              DISP=(NEW,PASS),SPACE=(1600,(100,50,10))
//FILE1 DD DDNAME=SYSIN
//L EXEC PGM=ICESLINK,COND=(3,LT,C),REGION=240K
//STEPLIB DD DSN=ICES.MODS,UNIT=3330,VOL=SER=NUSICE,DISP=SHR
//FT02F001 DD UNIT=SYSDA,SPACE=(3200,(5,5)),                          *
//              DCB=(BLKSIZE=3200,RECFM=FB,LRECL=80)
//FT04F001 DD  UNIT=SYSDA,SPACE=(3200,(2,2)),DCB=*.FT02F001
//FT05F001 DD UNIT=SYSDA,SPACE=(3200,(3,2)),DCB=*.FT02F001
//FT06F001 DD UNIT=(CTC,,DEFER),DCB=(BLKSIZE=931,LRECL=133,RECFM=FBA)
//SYSPRINT DD UNIT=(CTC,,DEFER),DCB=(BLKSIZE=968,LRECL=121,RECFM=FBM)
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(60,30))
//SYSUT2 DD UNIT=SYSDA,SPACE=(1024,(40,10))
//SYSUT3 DD UNIT=SYSDA,SPACE=(1024,(30,10))
//MACLIB DD DSN=ICES.MACLIB,DISP=SHR,UNIT=3330,VOL=SER=NUSICE
//SYSPUNCH DD UNIT=SYSDA,SPACE=(3200,(10,5)),DCB=*.FT02F001
//SYSLIB DD DSN=ICES.FUNCLIB,UNIT=3330,VOL=SER=NUSICE,DISP=SHR
// DD DSN=GFK.FORTLIB,DISP=SHR
//INCLLIB DD DSN=&&OBJ,DISP=(OLD,DELETE)
//SYSLMOD DD DSN=&&NEWLD,DISP=(NEW,PASS),UNIT=SYSDA,                  *
//              SPACE=(TRK,(90,,9))
//SYSLIN DD DSN=*.SYSPUNCH,DISP=(OLD,DELETE),VOL=REF=*.SYSPUNCH,      *
//              DCB=*.FT02F001
// DD DSN=*.FT04F001,DISP=(OLD,DELETE),VOL=REF=*.FT04F001,            *
//              DCB=*.FT02F001
//G EXEC PGM=&QQQ,REGION=240K,TIME=60
//STEPLIB DD DSN=&&NEWLD,DISP=(OLD,DELETE)
// DD DSN=ICES.MODS,UNIT=3330,VOL=SER=NUSICE,DISP=SHR
//FT06F001 DD UNIT=(CTC,,DEFER),DCB=(BLKSIZE=&BLK,LRECL=133,RECFM=FBA)
//FT06F002 DD UNIT=(CTC,,DEFER)
//DD1 DD DSN=ICES.USER,UNIT=3330,VOL=SER=NUSICE,DISP=OLD
//DD2 DD DSN=ICES.SUBS,UNIT=3330,VOL=SER=NUSICE,DISP=OLD
//DD3 DD DSN=ICES.DICT,UNIT=3330,VOL=SER=NUSICE,DISP=OLD
//DD4 DD UNIT=SYSDA,SPACE=(TRK,200),DCB=DSORG=DA
//FT05F001 DD DDNAME=SYSIN
// PEND
```

Abb. 5: Prozedur zum Kompilieren, Linken und Ausführen von ICETRAN-Programmen

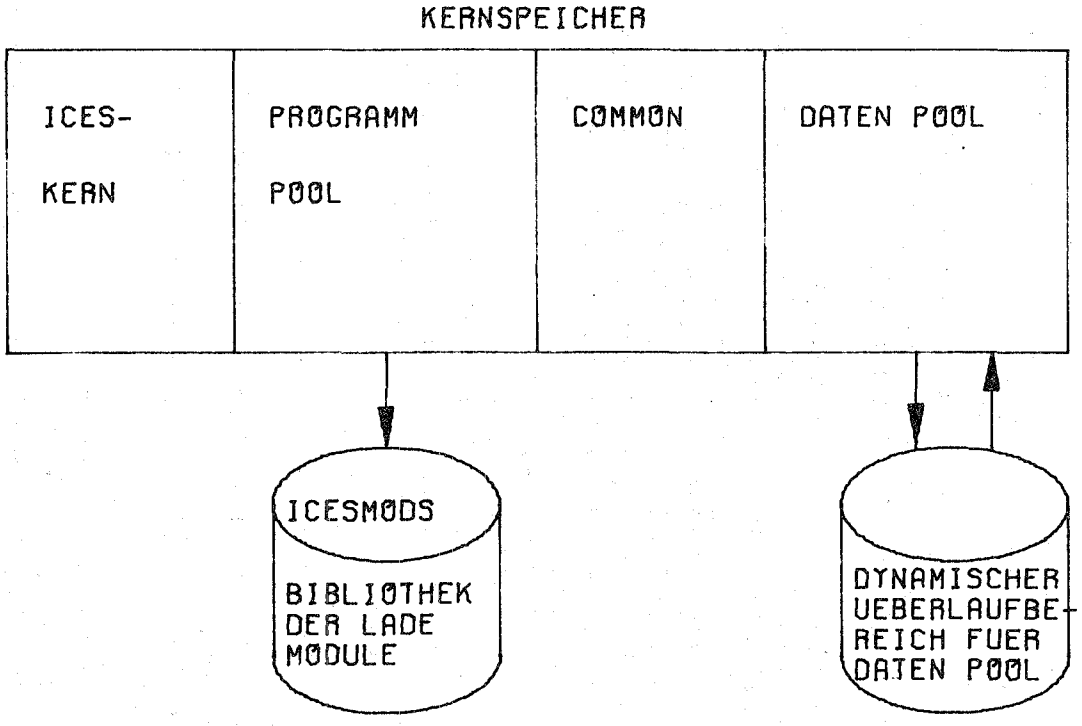

ABB.6 INTERNE VERWALTUNGSBEREICHE

BEISPIELE DER FRED SPRACHE

MOEGLICHE EINGABE:

```
NU = 0. + 0.625 * RE** 0.40 * PR  ** 0.40

TEMPERATUR AM EINTRITT    653. K

DURCHMESSER DES PINS 0.25 IN
```

ZUGEHOERIGE CDL - EINGABE:

```
ADD 'NU'
   IGNORE '*' '**' 'TB+(TW-TB)*'
       ID '=' REAL 'CNU1' STANDARD 0.
       ID '+' REAL 'CNU2' REQ
       ID 'RE' REAL 'CN1' REQ
       ID 'PR' REAL 'CN2' REQ
       ID '(TB/TW)' REAL 'CN3' STANDARD 0.
   CONDITION REAL 'CN3' EQ 0.
       ID 'REF' REAL 'CDT' STANDARD 0.
   END CONDITION OPTIONAL
FILE

ADD 'DU'  $ DURCHMESSER
   MODIFIER 'HYD' $ HYDRAULISCH
       CALL 'HYD'
   OTHERWISE
       IGNORE 'DES' 'BRENNSTABES' 'DER' 'STAEBE' 'PINS'
       NO ID REAL 'SWL' REQ
       EXISTENCE 'CM' 'MM' 'M ' 'I' 'MIL' SET 'NDIML' STANDARD 1
       EXECUTE LINK 'THLANG'
       MOVE REAL 'SWL' TO 'D'
   END MODIFIER
FILE

ADD 'TE' $ TEMPERATUR
   IGNORE 'AM' 'EINTRITT' 'IM' 'KUEHLMITTEL' 'DES' 'KUEHLMITTELS'
   NO ID REAL 'TEMP' REQ
    EXISTENCE 'C' 'K' 'F' SET 'NDIMT' STANDARD 1
    EXECUTE LINK 'R
    EXECUTE LINK 'FRTEMP'
   MOVE REAL 'TEMP' TO 'TE'
FILE
```

Abb. 7: Definition und Benutzung der FRED-Sprache