KfK 3217
Oktober 1981

Ein System zum funktionellen Modellieren unter Verwendung von Datenbanktechniken und interaktiven graphischen Arbeitsmethoden

K. Leinemann
Institut für Reaktorentwicklung

Kernforschungszentrum Karlsruhe

Kernforschungszentrum Karlsruhe
Institut für Reaktorentwicklung

KfK 3217

Ein System zum funktionellen Modellieren unter Verwendung von Datenbanktechniken und interaktiven graphischen Arbeitsmethoden

K. Leinemann

Als Dissertation genehmigt von der Fakultät für
Maschinenbau der Universität Karlsruhe

Kernforschungszentrum Karlsruhe G.m.b.H., Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
ISSN 0303-4003

Ein System zum funktionellen Modellieren unter Verwendung von Datenbanktechniken und interaktiven graphischen Arbeitsmethoden

Zusammenfassung

Dieses CAD - System dient der interaktiven Aufbereitung der Daten, die ein zu analysierendes System in funktioneller Hinsicht beschreiben. Es unterstützt die Manipulation von Informationen, wie sie durch Blockdiagramme und Tabellen erfaßt werden und stellt damit ein Werkzeug bereit, das bereits in einem frühen Konstruktions-/Planungsstadium nicht nur zeichnerische Unterstützung bietet, sondern eine problemangepaßte Informationsverarbeitung realisiert, die auf Datenbanktechniken fußt und diese durch interaktive graphische Methoden ergänzt.

GRIMBI verfügt über eine Datendefinitionssprache (DDL) zur logischen und graphischen Beschreibung von Modellklassen, die die problemtypischen Modellierungsrestriktionen erfassen und GRIMBI somit an spezielle Problembereiche anpassen. Dazu wurde das im Bereich des funktionellen Modellierens gebräuchliche Blockdiagramm - Datenmodell als problemorientiertes Datenmodell adaptiert. Auf Grund der Modellklassen - Informationen ist GRIMBI in der Lage, Entwurfsfehler zu entdecken, so daß nur konsistente Modelle aufgebaut werden können.

Für das Modellieren gemäß einer Modellklasse stellt GRIMBI Standardoperationen, systemtechnische Arbeitsmethoden wie schrittweises Verfeinern und Abstrahieren und das Verwalten von Modellalternativen bereit. Modelliert wird interaktiv mit einer selbständigen Kommando/Menü-Sprache oder im Stapelbetrieb mit der GRIMBI-DML, einer Datenmanipulationssprache als Erweiterung der Gastsprache PL/1. Für die Auswertung eines Modells (Analyse) stehen DML - Befehle zur Unterstützung des Navigierens in einem Modell bereit, d.h. für den relativen Zugriff auf Daten des Modells.

GRIMBI ist als Subsystem des integrierten CAD-Systems REGENT implementiert. Es ergänzt REGENT durch eine problemspezifische Datenbankkomponente und eine graphisch orientierte Dialogkomponente. Diese Kopplung an REGENT ergibt eine Integration von Stapelbetriebsaktivitäten (Modellklassendefinition, umfangreiche Modellanalysen) mit interaktiven Arbeiten (Modellsynthese).

Der Einsatz eines intelligenten graphischen Terminals in Verbindung mit einem Großrechner und einer geeigneten Aufgaben- und Datenverteilung ergibt problemgerechte Antwortzeiten und die Möglichkeit, das Terminal allein auch als Blockdiagramm-Zeichensystem zu nutzen.

GRIMBI wird vorerst bei der Sicherheitsanalyse kerntechnischer Anlagen (Fehlerbaumanalyse) eingesetzt.

A System for Functional Modelling using Data Base Techniques and Interactive Graphic Working Methods

---

Summary

This CAD-system serves for preparing data of systems to be analysed under functional aspects. It supports manipulation of informations included in blockdiagrams and tables and therefore is an aid for the early design phases not only for drawing but especially for a problem oriented information handling, based on data base techniques completed by interactive graphic working methods.

GRIMBI contains a data definition language (DDL) to describe model classes logically and graphically. Modell classes represent problem oriented modelling restrictions. Model class description is based on a problem oriented data model, obtained by adapting the known blockdiagram data model. The model class information enables GRIMBI to detect wrong modelling and to get consistent models.

Modelling according to a model class is done by standard operations, working methods like stepwise refinement and abstraction and management of design alternatives. Modelling is done using a self-contained command/menue-language or the GRIMBI-DML for batch environment, a data manipulation language, which is an expansion of the host language PL/1. Model analysis is supported by special DML-statements for navigation in the data structure.

GRIMBI is implemented as subsystem of the integrated CAD kernel system REGENT. It completes REGENT by a problem oriented data base component and a graphics oriented dialog component. This coupling to REGENT provides an integration of batch activities (model class definition, analysis of models) with interactive tasks (model synthesis).

Usage of an intelligent graphic terminal connected to a large host yields, together with a suitable task and data distribution, a convenient response time behaviour, and the possibility to use the terminal for drawing of blockdiagrams without host support, that means without considering problem restrictions and without GRIMBI-DML access to the data for analysis.

GRIMBI is presently used in safety analysis of nuclear plants (fault tree analysis).

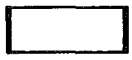
Inhaltsverzeichnis

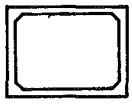
Abbildungsverzeichnis

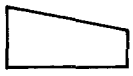
Liste haüfig verwendeter Abkürzungen

$\in$	ist Element aus
$\subseteq$	ist Teilmenge von
$\subset$	ist echte Teilmenge von
$\rightarrow$	ist funktional abhängig von
$\times$	kartesisches Produkt


Datenmenge, Datei, Zustand

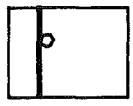

Prozess, Programmpaket


graphischer Bildschirm

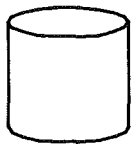

Tastatur


Karteneingabe
Eingabe für Stapelbetrieb


Tablett


Zeichenmaschine


Steuerknüppel


Plattenspeicher

1. Einleitung

Eine wesentliche Aufgabe des Ingenieurs besteht darin, Anlagen, Bauwerke, Geräte oder Maschinen, also dingliche, technische Objekte, die eine geforderte Funktion erfüllen, zu konstruieren und zu produzieren oder zu planen und zu montieren.

Daraus ergeben sich bezüglich der Arbeitstechniken zwei Bereiche /1/,in denen man mit ganz unterschiedlichen, der jeweiligen Aufgabe angepaßten Modellen arbeitet (Abb.1).

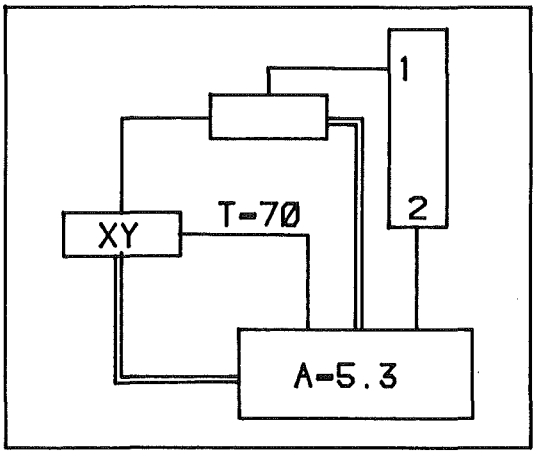
(1) Der funktionellen Phase liegt eine systemtechnische Betrachtung der Objekte zugrunde. Dabei wird das zu entwickelnde Produkt einerseits als Element eines Systems angesehen, andererseits aber selbst als System aus bekannten Grundbausteinen aufgebaut und durch seine Topologie und kennzeichnende Eigenschaften beschrieben. Gebräuchliches Hilfsmittel für die Informationsdarstellung in diesem Bereich ist das Blockdiagramm , das die Systemtopologie und die Bauelementparameter symbolisch repräsentiert.

(2) In der gestalterischen Phase wird das zu bearbeitende technische Objekt als physisches Gebilde betrachtet, dessen Geometrie und dessen technologische Daten festzulegen sind. In diesem Bereich werden technische Zeichnungen zur Informationsdarstellung eingesetzt. Sie repräsentieren ein nach festen Regeln angefertigtes Bild des Objektes, in dem geometrische Größen auch wieder durch geometrische Größen dargestellt werden.

Diese allgemeine Klassifizierung nach Modelltypen und daraus sich ergebenden Arbeitstechniken findet sich auch in dem speziellen Bereich sicherheitstechnischer Untersuchungen kerntechnischer Anlagen.

Hierbei werden für fluid-struktur-dynamische Analysen Anlagenmodelle zur Erfassung physikalisch-technischer Zusammenhänge und relativ einfacher Geometrien bearbeitet. Für ihre Beschreibung haben sich problemorientierte Sprachen, wie sie beispielsweise das integrierte CAD-System REGENT /2/ unterstützt, gut bewährt. Bei komplexen geometrischen Modellen, wie sie im allgemeinen für strukturmechanische Analysen mit Finit-Element-Methoden auftreten, ist dagegen die Verwendung eines interaktiven graphischen Geometrieprozessors zweckmäßig, um fehlerfreie Modelle zu erhalten.

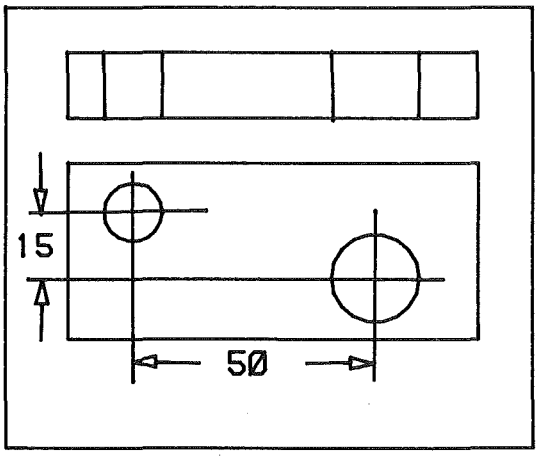
FUNKTIONAL



BLOCKDIAGRAMM

* TOPOLOGIE
* WERTE

GESTALTERISCH



TECHNISCHE ZEICHNUNG

* GEOMETRIE
* WERTE

TYPISCHE OPERATIONEN/OPERANDEN

- EINFUEGEN/LOESCHEN VON SYMBOLEN/VERBINDUNGEN
- EINFUEGEN/AENDERN VON WERTEN

- ZEICHNE LINIE/KREIS...
- ZEICHNE TANGENTE AN...
- SCHRAFFIERE...
- BILDE FLAECHE AUS...
- BILDE KOERPER AUS...
- TRANSFORMIERE GEOMETR. OBJ.
- ZEICHNE ANSICHT, SCHNITT...
- EINFUEGEN/AENDERN VON TECHNOLOGISCHEN DATEN
- BEMASSEN

A1

Abb.1: Modellierungs-Klassen

Diesem geometrischen Modellieren von Anlagen und Anlagenteilen für die mechanische Analyse steht das funktionelle Modellieren gegenüber. Es dient der Modellerstellung für Analysen unter funktionellen Aspekten wie beispielsweise der Ermittlung der Ausfallwahrscheinlichkeit einer Anlage. In diesem Bereich wird die Anlage durch ein geeignetes Blockdiagramm modelliert, das manuell in Programmeingaben umgesetzt wird. Für einige spezielle Analyseprogramme sind auch hier interaktive Systeme entwickelt worden, um das Modellieren komplexer topologischer Zusammenhänge zu erleichtern.

Wie die im zweiten Kapitel folgende ausführliche Diskussion des Standes der Technik von EDV-Systemen für derartige Aufgaben aber zeigt, gibt es kein selbständiges CAD-System, das das funktionelle Modellieren als allgemeine ingenieurstechnische Methode umfassend unterstützt.

Gegenstand der vorliegenden Arbeit ist daher ein CAD-System zur Unterstützung der funktionellen Bearbeitung technischer Objekte, die darin besteht

- ein Datenmodell des technischen Objektes unter problemabhängigen Restriktionen aufzubauen, zu verändern und zu verwalten,
- dieses Modell in geeigneter Form darzustellen und
- seine Auswertung durch Analysealgorithmen zu erleichtern.

Die Grundidee dabei ist, die Blockdiagrammtechnik als bewährte Methode in diesem Bereich /3/ beizubehalten, sie aber für den Rechnereinsatz aufzubereiten. Das bedeutet vor allem, die Zeichnung (das Blockdiagramm) nicht mehr als Datenträger zu verwenden, sondern lediglich als Darstellungsmethode für Daten, die in einem DV-System problemgerecht verwaltet werden.

Jeder Arbeitsbereich besitzt ein eigenes spezifisches Datenmodell des zu bearbeitenden Objektes, so wie jeder Bereich seine eigene Fachsprache kennt. Alle diese Modelle repräsentieren natürlich das gleiche Objekt, daher sollte ein umfassendes zentrales Datenmodell den Kern eines Gesamt-CAD-Systems bilden. Die einzelnen Teilmodelle könnten entweder als logische Teildatenbanken (Sichten) dieses zentralen Modelles realisiert werden (Subschema-Konzept) oder als temporäre physische Teildatenbanken realisiert werden. Letztere Möglichkeit ist Grundlage der der vorliegenden Arbeit (Abb.2). Diese Vorgehensweise erfordert für jedes Teilmodell einen Modelltransformator, der das Teilmodell nach seiner Fertigstellung in die zentrale Datenbank überträgt bzw. ein spezielles Teilmodell aus dem Gesamtmodell für einen Arbeitsschritt extrahiert.

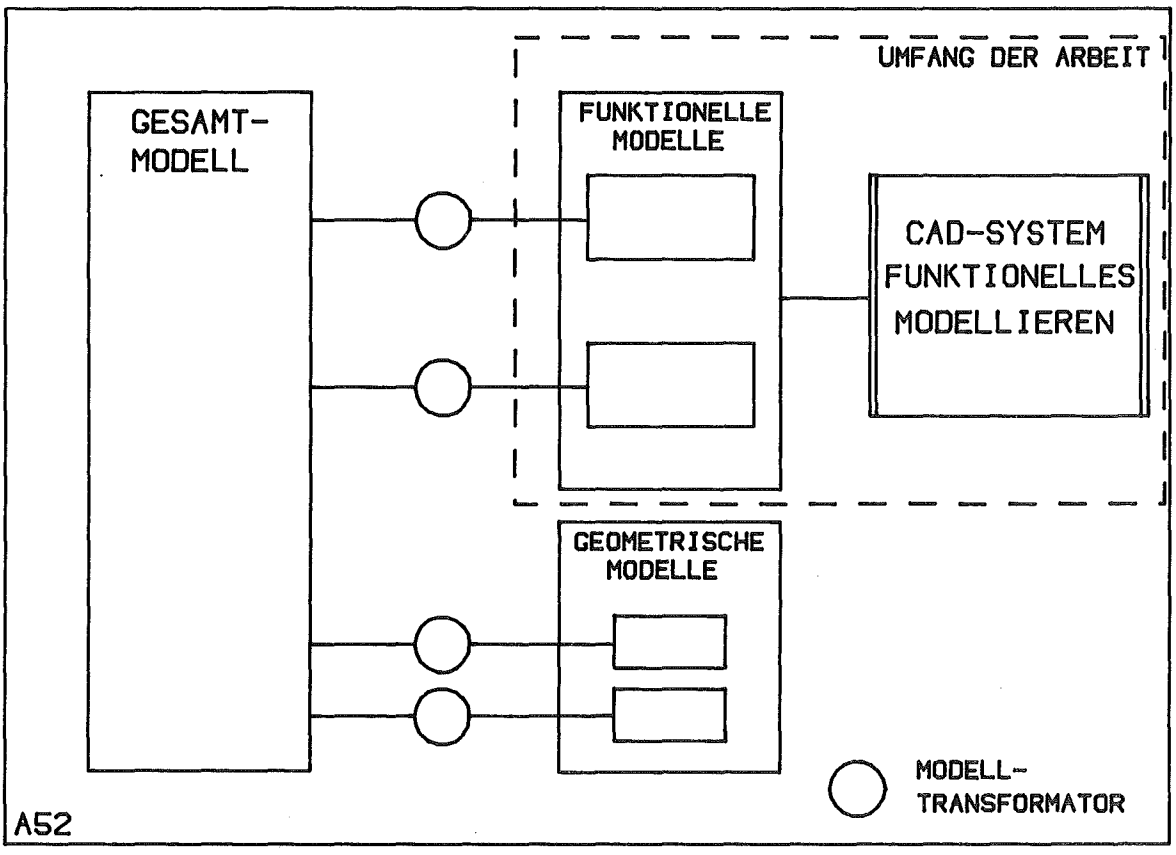


Abb.2: Mögliche Struktur eines CAD-Gesamtsystems

Ein Arbeitsbereich, in dem neben- und nacheinander die unterschiedlichsten funktionellen Modelle technischer Objekte behandelt werden, ist die Anlagenplanung im Bereich der chemischen Industrie und der Kerntechnik. Eine Analyse ihrer Arbeitsabläufe und Arbeitsergebnisse verdeutlicht einige charakteristische Merkmale und gestattet, Forderungen an ein CAD-System für das funktionelle Modellieren abzuleiten.

1.1. Anlagenplanung als Einsatzgebiet funktionellen Modellierens

Unter einer Anlage im Sinne dieser Arbeit versteht man eine Menge von Aggregaten zur Stoff-, Energie- und Informationsumwandlung, die durch Leitungen zum Stoff-, Energie- und Informationstransport verbunden sind. Die Anlagenplanung befaßt sich mit der Ermittlung einer geeigneten funktionellen und räumlichen Struktur einer solchen Anlage, die einer vorgegebenen Gesamtfunktion unter vorgegebenen Randbedingungen entspricht. Die dazugehörigen kaufmännischen und organisatorischen Aspekte werden erst in zweiter Hinsicht berücksichtigt. Der Anlagenplanungsprozeß ermittelt aus der verfahrenstechnischen Spezifikation einer Anlage alle Daten, die für den Bau und Betrieb erforderlich sind. Abb.3 unterscheidet fünf Abschnitte der

Anlagenplanung. Die Abgrenzungen und Bezeichnungen der einzelnen Phasen orientieren sich an den Modellvorstellungen in den jeweiligen Bereichen: Nach Phasen mit funktionellen Modellen erfolgt ein Übergang auf geometrisch bestimmte Modelle. In den Schlußphasen werden dann beide Modelltypen nebeneinander eingesetzt.

Ausgehend von einer Anlagenspezifikation durch die hinein- und herausflie$enden Ströme und technische, kaufmännische, organisatorische und umweltbezogene Randbedingungen, ergibt der Prozeßentwurf ein Netz verfahrenstechnischer Funktionen, das aus Funktionselementen des Stoff-, Energie- und Informationsumsatzes und Strömen von Stoffen, Energie und Informationen besteht. Angaben zur Grösse der Ströme und die zur Durchführung erforderlichen Betriebsbedingungen (Temperaturen, Drücke etc.) ergänzen die Prozeßtopologie.

Der darauffolgende Entwurfsschritt, die funktionelle Anlagenplanung befaßt sich mit der technischen Ausrüstung, die für die Durchführung des vorher festgelegten verfahrenstechnischen Prozesses erforderlich ist (DIN 28004 /4/). Es geht also darum, den einzelnen Prozeßfunktionen Apparate und Maschinen und den Prozeßströmen Leitungen zuzuordnen und eine geeignete MSR-Technik (Meß-, Steuer- und Regeltechnik) zu planen, die dafür sorgt, daß die festgelegten Betriebsbedingungen eingehalten werden.

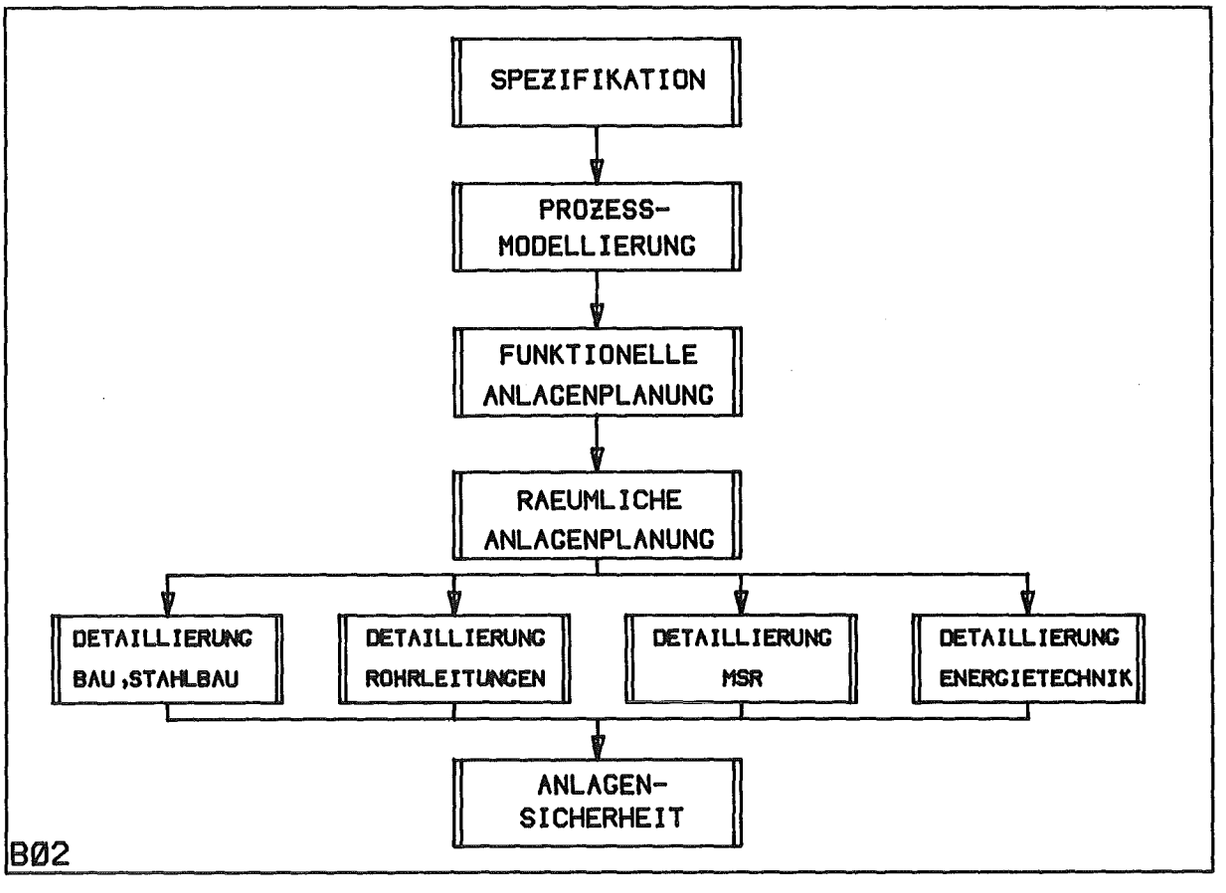


Abb. 3: Teilprozesse der Anlagenplanung

Diese funktionelle Struktur der Anlage bildet die Basis für den räumlichen Anlagenentwurf, bei dem man die Anlagenkomponenten im vorgegebenen Raum positioniert, ihnen also geometrische Daten zuordnet.

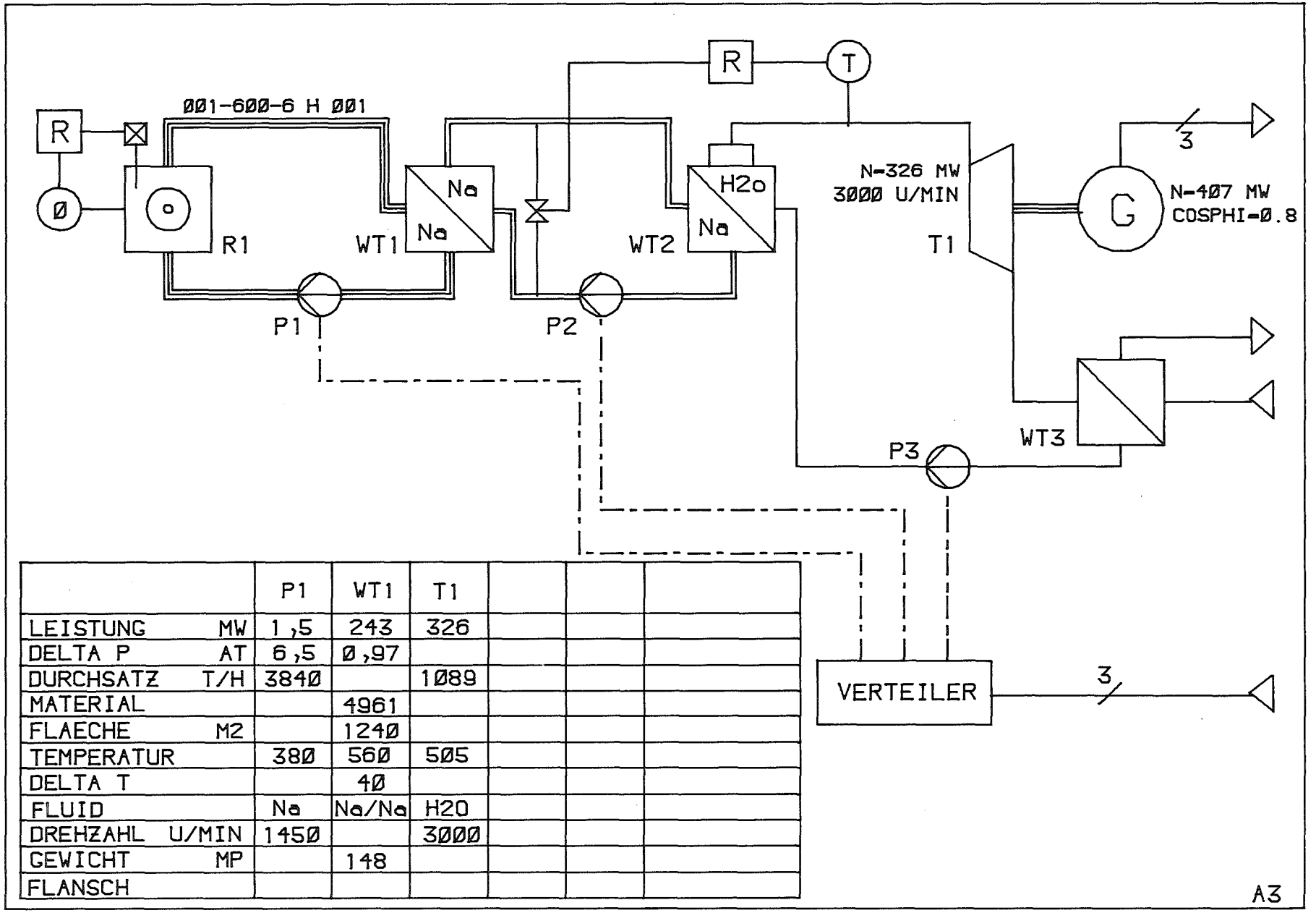
Der Bau einer Anlage erfordert eine Konkretisierung der funktionellen und räumlich-abstrakten Beschreibung. Sie erfolgt in der Phase der Anlagendetaillierung in fachspezifischen Teilbereichen wie Bau/Stahlbau, Rohrleitungsplanung, MSR-Technik, elektrische Energietechnik.

Insbesondere in der Kerntechnik folgt abschließend die Untersuchung der Anlagensicherheit, die Detaildaten der Anlage erfordert. In dieser Phase wird ein Modell der Anlage erstellt, das es gestattet, Aussagen über die Ausfallwahrscheinlichkeit abzuleiten und Hinweise für geeignete Konstruktionsänderungen zu gewinnen.

Diese Gliederung des Planungsprozesses weicht geringfügig von derjenigen in /5/ ab. Dies ist auf die stärkere Orientierung an den verschiedenen Modelltypen zurückzuführen und auf eine damit zusammenhängende stärkere Abstraktion, die Teilprozesse wie Berechnung und Auslegung oder die Apparatespezifikation nicht explizit in Erscheinung treten läßt.

Der in Abb.3 angegebene Planungsablauf bedeutet nicht, daß eine Planungsphase vollständig abgeschlossen werden muß, bevor die folgende begonnen werden kann. Es ist im allgemeinen so, daß in einer Phase erst einmal vorläufige Daten ermittelt werden, mit denen dann in den Folgephasen vorerst gearbeitet wird. Ein Endergebnis entsteht erst durch eine mehrfache Iteration Das iterative Vorgehen bedingt, daß die Anlagendaten mit einer Eigenschaft gekennzeichnet werden müssen, die eine Aussage über ihren Wert macht. Die Daten sind beispielsweise als vorläufig oder als endgültig zu betrachten, was Auswirkungen auf Konsistenzprobleme bei Datenbanken in diesem Bereich hat (Kapitel 2.2.3.).

In allen genannten Bereichen mit Ausnahme des räumlichen Entwurfs und der Bau/Stahlbau-Planung betrachtet man die Anlage oder Teile der Anlage unter verschiedenen Aspekten funktionell. Um wesentliche Probleme dieser Betrachtungsweise hinsichtlich einer EDV-Unterstützung zu verdeutlichen, werden in den beiden folgenden Teilabschnitten die funktionelle Anlagenplanung und die Sicherheitsanalyse von Anlagen exemplarisch etwas genauer dargestellt.



	P1	WT1	T1			
LEISTUNG MW	1,5	243	326			
DELTA P AT	6,5	Ø,97				
DURCHSATZ T/H	384Ø		1Ø89			
MATERIAL		4961				
FLAECHE M2		124Ø				
TEMPERATUR	38Ø	56Ø	5Ø5			
DELTA T		4Ø				
FLUID	Na	Na/Na	H2O			
DREHZAHL U/MIN	145Ø		3ØØØ			
GEWICHT MP		148				
FLANSCH						

Abb.4: R & I - Fließbild eines Kernkraftwerkes

1.1.1. Funktionelle Anlagenplanung

Diese Planungsphase umfaßt die Ermittlung der erforderlichen gerätetechnischen Ausrüstung für einen vorgegebenen Prozeß. Das Ergebnis wird im allgemeinen in Form eines R & i - Fließbildes (Rohrleitungs- und Instrumentierungs-Fließbildes) dokumentiert. Ein vereinfachtes Beispiel aus dem Bereich der Reaktortechnik zeigt Abb.4. Das Fließbild stellt die Kühlkreisläufe eines Natrium-gekühlten Kernreaktors dar. Ohne genauer auf Details einzugehen, wird hier deutlich, daß die Blockdiagramme und damit die Informationsverarbeitung in diesem Bereich vor allem durch folgende Eigenschaften charakterisiert werden:

- viele unterschiedliche Typen von Objekten und Verbindungen,
- viele Objekteigenschaften, die durch Variation der Symbolgraphik, durch Texte oder ergänzende Tabellen dargestellt werden,
- problemorientierte Restriktionen für den Aufbau einer Struktur. Sie betreffen die Zulässigkeit von Objekteigenschaften, die beispielsweise durch Normen eingeschränkt sind, sie betreffen aber vor allem die Verbindungen zwischen den verschiedenen Anlagenkomponenten. Diese Art von Restriktionen verbietet beispielsweise, einen Fluidausgang einer Pumpe mit dem Ausgang eines Elektro-Verteilers oder der Stromzuführung einer Pumpe zu verbinden (Abb.5).

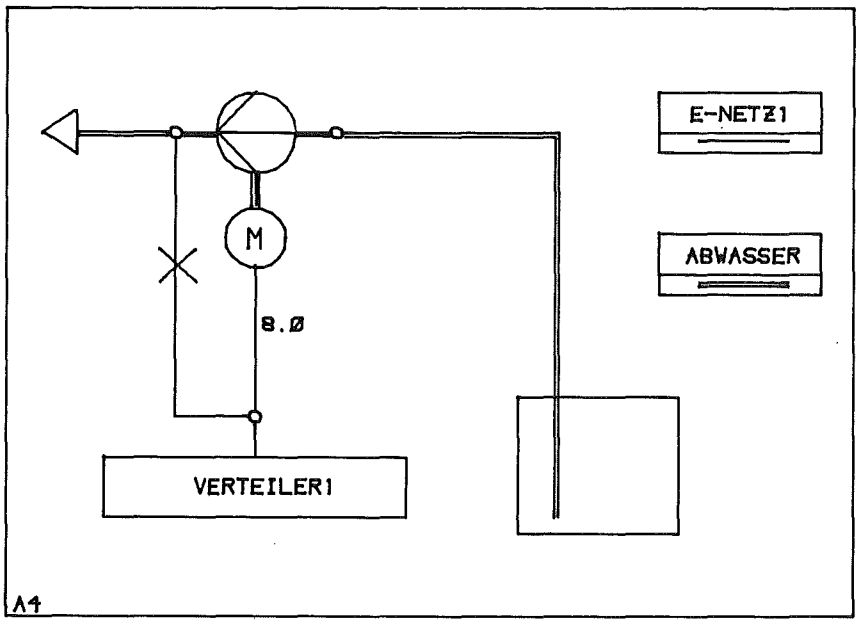


Abb.5: Entwurfsfehler

Die Ausführung von Fließbildern verfahrenstechnischer Anlagen ist in DIN-28004 /4/ und DIN-19227 /6/ genormt. Diese Fließbilder werden aus Symbolen aufgebaut, die einen Apparat oder eine Maschine durch ein Grundsymbol darstellen, das entsprechend der spezifischen Eigenschaften des jeweiligen Falles durch Zusatzsymbole ergänzt wird. Informationen, die durch diese Symbolik nicht erfaßt werden, ergänzt man in Form von Texten oder Tabellen. Die Abb.6 zeigt die Variation eines Grundsymbols in Abhängigkeit von einer Objekteigenschaft, dem Pumpentyp.

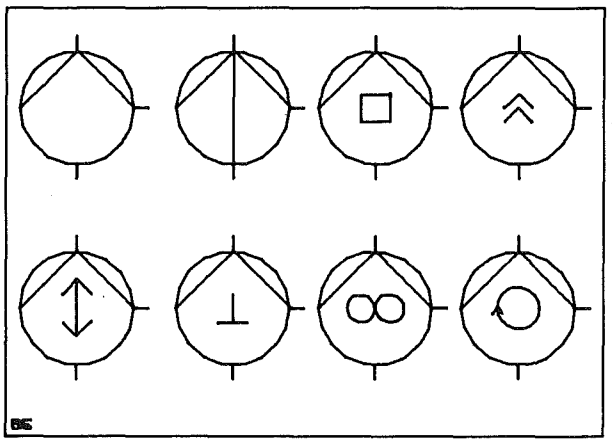


Abb.6: Symbolvariationen

1.1.2. Sicherheitsanalyse

Die Sicherheitsanalyse von Anlagen verfolgt zwei Ziele:

- die Ermittlung quantitativer Zuverlässigkeitsangaben für die Risikoermittlung und den Vergleich von Teilsystemen oder Alternativen Entwürfen und

- die Aufdeckung von Schwachstellen, um gezielt Entwurfsänderungen zur Zuverlässigkeitserhöhung durchführen zu können.

Für die Sicherheitsanalyse großer Anlagen, wie beispielsweise Kernkraftwerke, haben sich zwei Methoden bewährt: die Fehlerbaumanalyse und die Störfallablaufanalyse. Sie unterscheiden sich in der Methodik des Modellierens und bei der graphischen Darstellung des Modells, basieren aber beide auf dem sogenannten Booleschen Modell /7/.

Das Boolesche Modell kann Systeme mit endlich vielen Komponenten darstellen, die die beiden Zustände "intakt" und "defekt" annehmen können. Ordnet man den Komponenten boolesche Variable (Primärvariable) zu, so läßt sich das Gesamtsystem durch einen booleschen Ausdruck dieser Variablen beschreiben. Bezeichnet die Variable TOP das Gesamtsystem, so ist TOP=f(Vi) ein Modell des Systems mit i Komponenten. Die boolesche Funktion f modelliert das sicherheitstechnische Verhalten des Systems, indem sie eine Aussage darüber

macht, welche Komponenten defekt sein müssen, damit das Gesamtsystem defekt ist. Daraus läßt sich dann ableiten, wie groß die Wahrscheinlichkeit des Systemausfalls bei vorgegebenen Wahrscheinlichkeiten für Komponentenausfälle ist.

Modelliert wird mit Elementen, wie sie Abb.7 zeigt /8/. Diese Elemente bilden drei Klassen: Variable, Operationen und Teilsystemgrenzen. Ein Fehlerbaum ist eine graphische Repräsentation einer booleschen Funktion vom Typ TOP=f(Vi), bei der Zwischenvariable eingeführt worden sind, um die schrittweise Bearbeitung zu erleichtern. Darüberhinaus werden alle Variablen verbal beschrieben, damit ihre Bedeutung für jeden leicht verständlich ist und die Konstruktion des Baumes nachvollzogen werden kann.

Den Variablen sind außerdem verschiedene Attribute zugeordnet, die für die Analyse von Bedeutung sind. Primärvariable repräsentieren Zustände von Komponenten, deren Verhalten bekannt ist und die nicht weiter analysiert werden. Die Zwischenvariablen stellen Teisysteme dar, die im Verlauf der Synthese durch boolesche Ausdrücke (Baum von Variablen und Operationen) beschrieben werden. Die Variable TOP dient als Ergebnisvariable.

Diese Variablen werden im allgemeinen durch Konjunktionen, Disjunktionen und Negationen verknüpft. In speziellen Anlagen wie den Kernkraftwerken ergänzt man diese Funktionen noch durch Komplexfunktionen wie beispielsweise die 2aus3-Auswahl.

Das Zusatzsymbol Punkt am Eingang einer Funktion bewirkt eine Negation der Information.

Die Schnittstellenelemente Transfer-in und Transfer-out gestatten es, eine große, unübersichtliche Anlage in Teilanlagen aufzugliedern.

Die Standardoperationen mit denen ein System aus diesen Elementen aufgebaut wird sind: Einfügen, Löschen, Ändern von Elementen und deren Parametern, Verschieben von Elementen und Elementgruppen, Suchen von Systembereichen oder Elementen.

Eine wichtige Aufgabe der Sicherheits-Analyse besteht darin, verschiedene Anlagenvarianten zu untersuchen, um ihre Zuverlässigkeit zu vergleichen und ein Konstruktionsoptimum zu finden. Dazu müssen zu den verschiedenen Teilsystemmodellen Entwurfsalternativen bearbeitet werden.

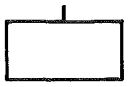
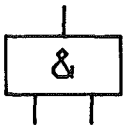
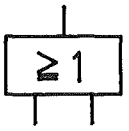
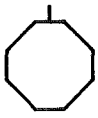
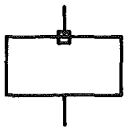
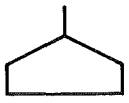
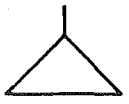
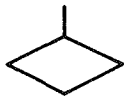
SYMBOL	BEDEUTUNG	SYMBOL	BEDEUTUNG
	VARIABLE KASTEN ENTHAELT BESCHREIBUNG		KONJUNKTION
	PRIMAERVARIABLE		DISJUNKTION
	PRIMAERVARIABLE, DIE ZU EINER ABHAENGIGEN KOMPONENTE GEHOERT		NEGATION
	VARIABLE, DER EIN KONSTANTER WERT ZUGEWIESEN WIRD		FEHLERBAUMEINGANG
	VARIABLE, DIE WEITER ENTWICKELT WERDEN SOLLTE		FEHLERBAUMAUSGANG
A6			

Abb.7: Elemente eines Fehlerbaumes (Auswahl aus /8/)

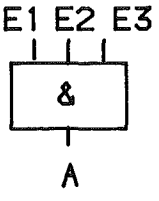
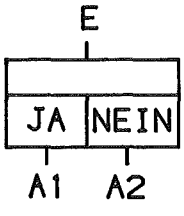
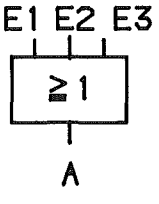
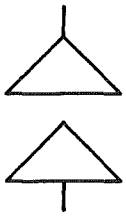
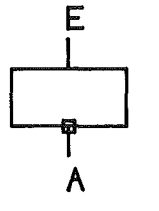
SYMBOL	BEDEUTUNG	SYMBOL	BEDEUTUNG
	ANFANGSEREIGNIS ZWISCHENEREIGNIS		KONJUNKTION A = E1 & E2 & E3
	EINFACHE VERZWEIGUNG EREIGNIS E FUEHRT ZUR FUNKTIONSANFORDERUNG AN EINE KOMPONENTE MIT 2 DISJUNKTEN ZUSTAENDEN		DISJUNKTION A = E1 V E2 V E3
	GRAPH-EINGANG GRAPH-AUSGANG		NEGATION A = $\overline{E}$
A8			

Abb.8: Elemente der Ereignisablaufanalyse (Auswahl aus/9/)

Während man den Aufbau von Fehlerbäumen bei einem wesentlichen, unerwünschten Ereignis beginnt (z.B. Bruch der Hauptkühlmittelleitung in einem Kernkraftwerk) und dazu schrittweise die Ursachen ermittelt, startet die Störfallablaufanalyse (Abb.8, /9/) mit einem bestimmten Ereignis, dessen mögliche Folgeereignisse aufgesucht werden (z.B. Ermitteln der Folgen eines Pumpenausfalls im Primärkreislauf). In der Praxis werden beide Methoden kombiniert eingesetzt, da bei der Untersuchung sehr großer Anlagen die Fehlerbaummethode allein auch nicht praktikabel ist. Man gliedert daher mit Hilfe der Störfallablaufmethode die Gesamtanlage in Teilsysteme, die dann mit der Fehlerbaummethode detailliert bearbeitet werden. Die Abbildungen 9,10,11 zeigen, wie drei verschiedenartige Modelltypen bei Sicherheitsanalysen nebeneinander eingesetzt werden. Ausgehend von einem R&I-Modell wird der Reaktor durch Ereignisablaufdiagramme und Fehlerbäume dargestellt, die in Teilbereichen eventuell noch durch einen weiteren Modelltyp, das Markovsche Zustandsübergangsmodell, ergänzt werden.

Unerwünschten Ereignisse, die mit der Fehlerbaummethode bearbeitet werden sollen, ermittelt man systematisch durch die Ereignisablaufanalyse (/10/,Abb.9). Dabei geht man von von einem Anfangsereignis aus, beispielsweise einem Rohrbruch, der den Kühlmittelverlust hervorruft, und verfolgt dann die Weiterentwicklung, je nachdem, ob die verschiedenen hierfür vorgesehenen Sicherheitssysteme funktionieren und mit welcher Wahrscheinlichkeit (P in Abb.9) dies der Fall ist. Das auslösende Ereignis in Abb.9 ist der Rohrbruch (A), seine Wahrscheinlichkeit P(A). Für die Sicherheitssysteme wird in jedem Fall elektrische Leistung benötigt, P(B) ist die Wahrscheinlichkeit, daß sie nicht verfügbar ist, 1-P(B) 1 (wegen P(B)<<1), daß sie vorhanden ist. Entsprechende Wahrscheinlichkeiten werden für das Notkühlsystem und den Sicherheitsbehälterabschluß definiert. Die angegebenen Produkte geben die unterschiedlichen Wahrscheinlichkeiten an, daß das Anfangsereignis nicht beherrscht wird. Erst die Wahrscheinlichkeiten der Teilsysteme P(X) werden mit der Fehlerbaummethode ermittelt.

Da die komplexen Algorithmen der Fehlerbaumanalyse relativ viel Speicherplatz und CPU-Zeit benötigen, handelt es sich bei ihnen um Programmsysteme für den Stapelbetrieb /11/. Ein weiteres Merkmal bezüglich des EDV-Einsatzes ist die geringe Modellklassenstabilität in diesem Bereich, da die Analysetechniken noch relativ neu sind und laufend verbessert werden, z.B. durch den Einbau neuer Elementtypen (und zugehöriger Algorithmen) /11/.

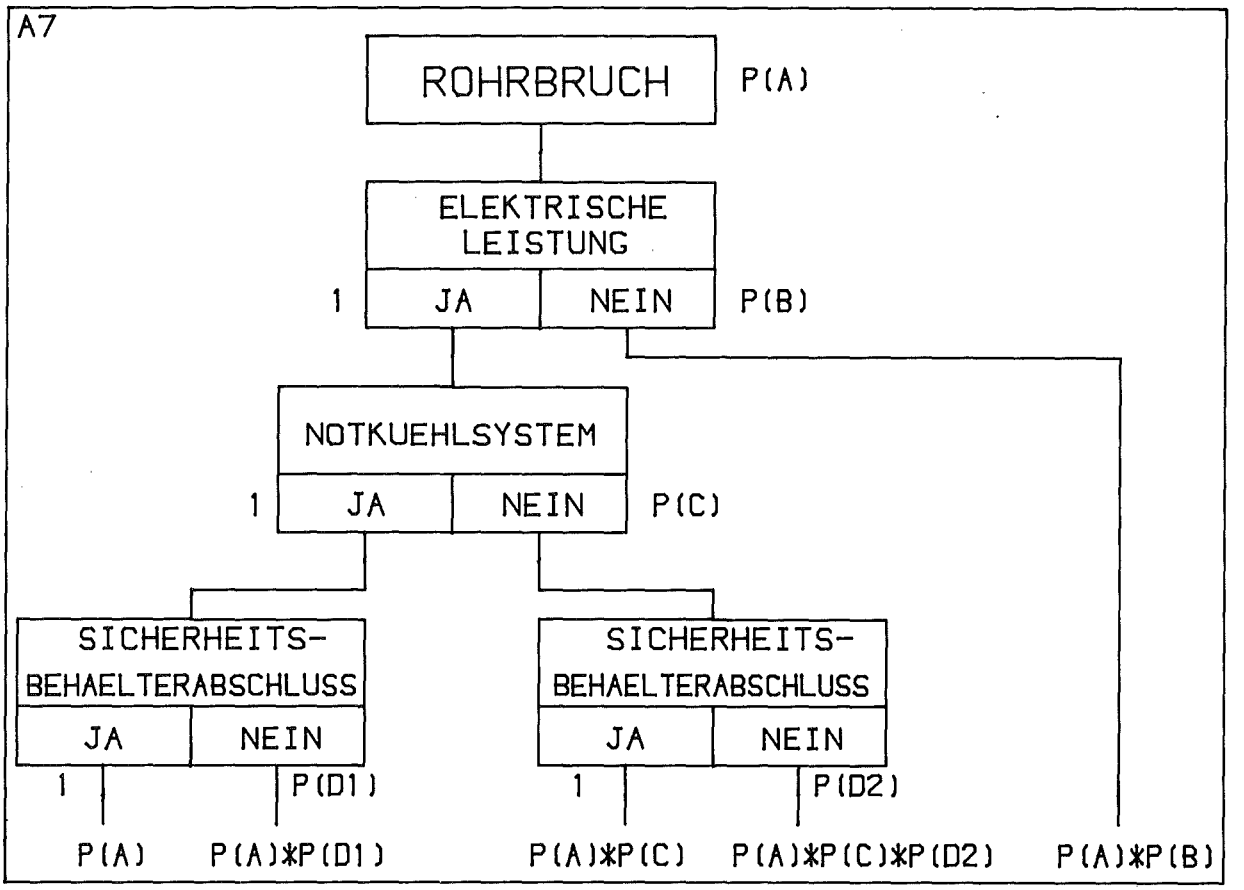


Abb.9: Ereignisablaufbaum eines Kernreaktors

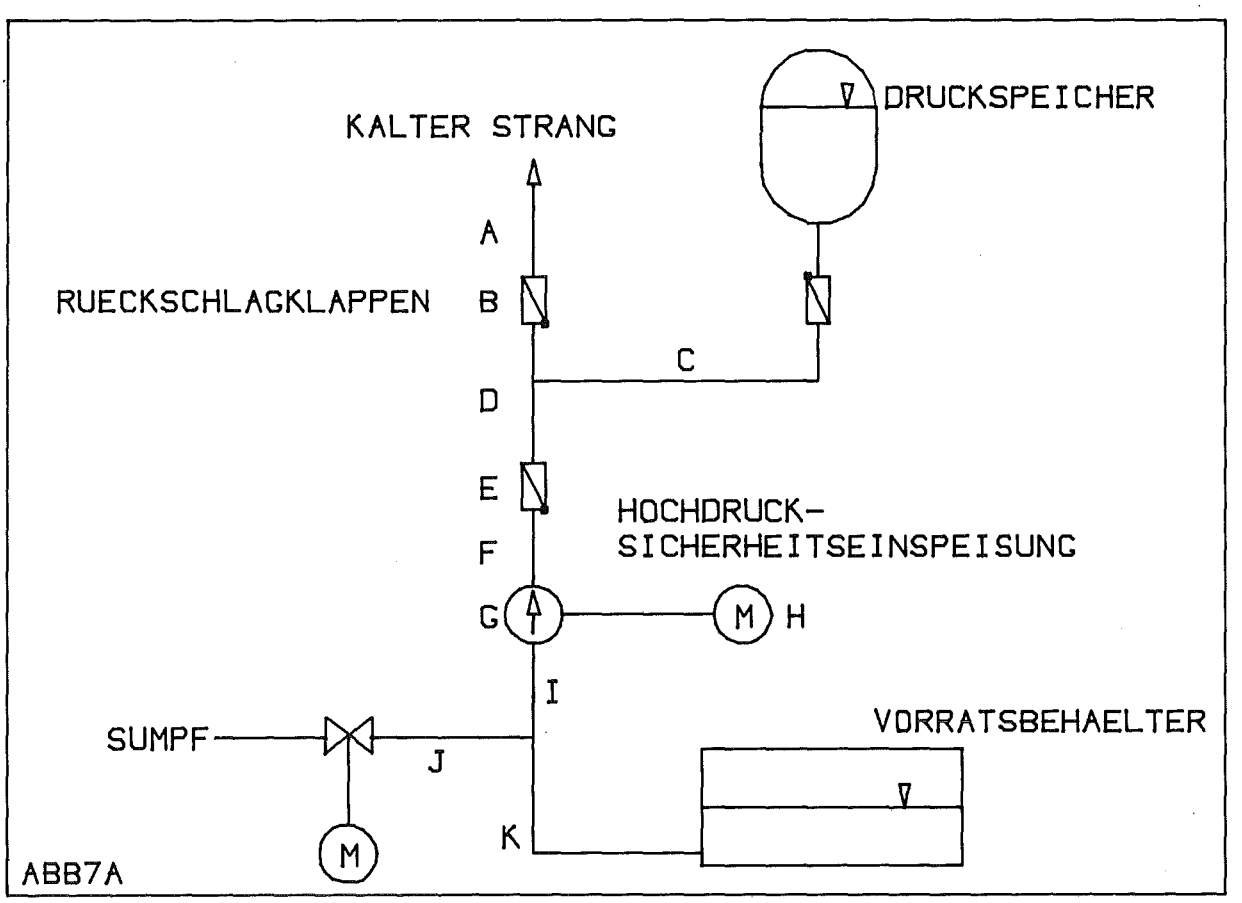


Abb.10: R&I-Fließbild eines Reaktorkühlsystems

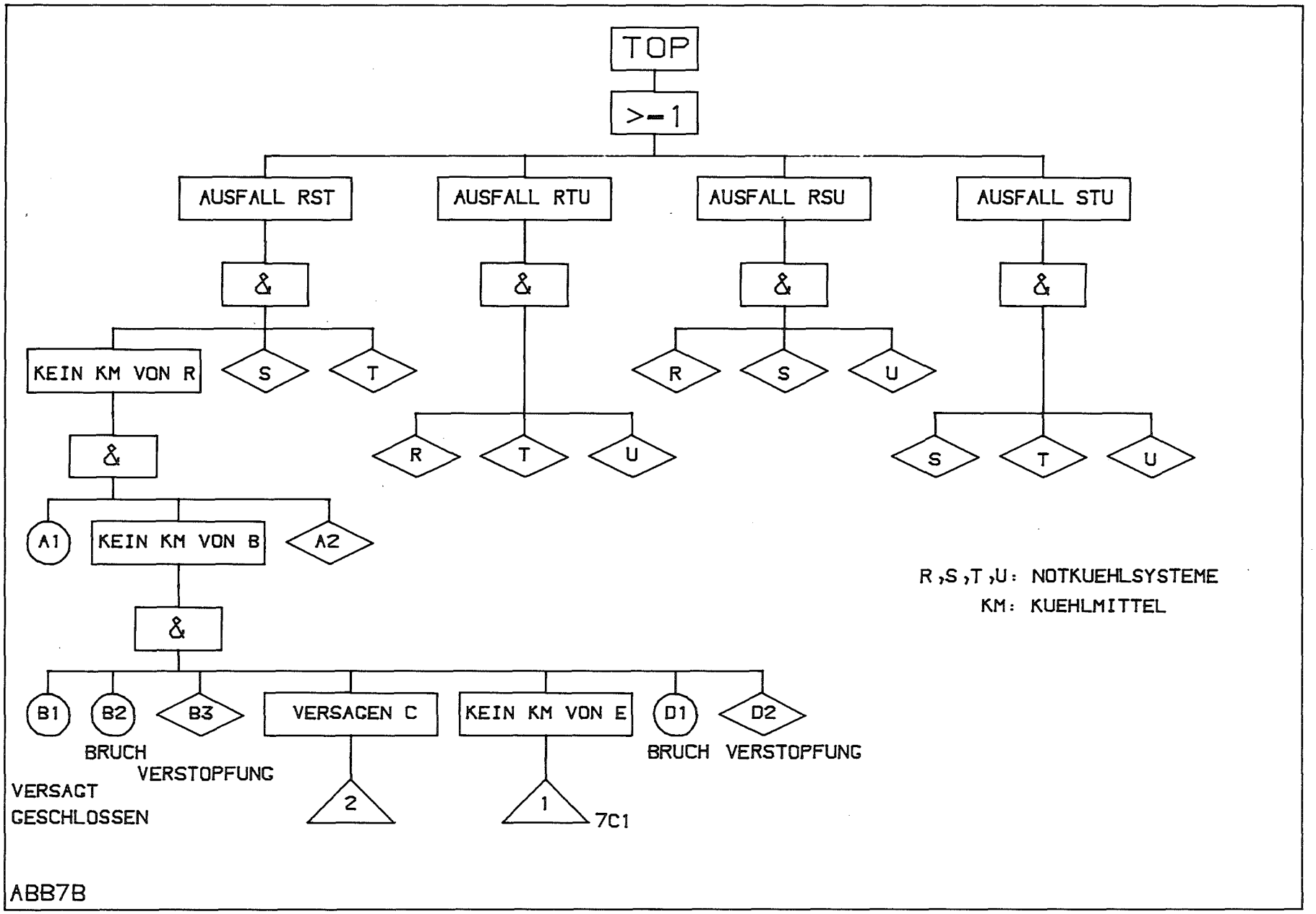


Abb.11a: Fehlerbaum eines Kühlsystems

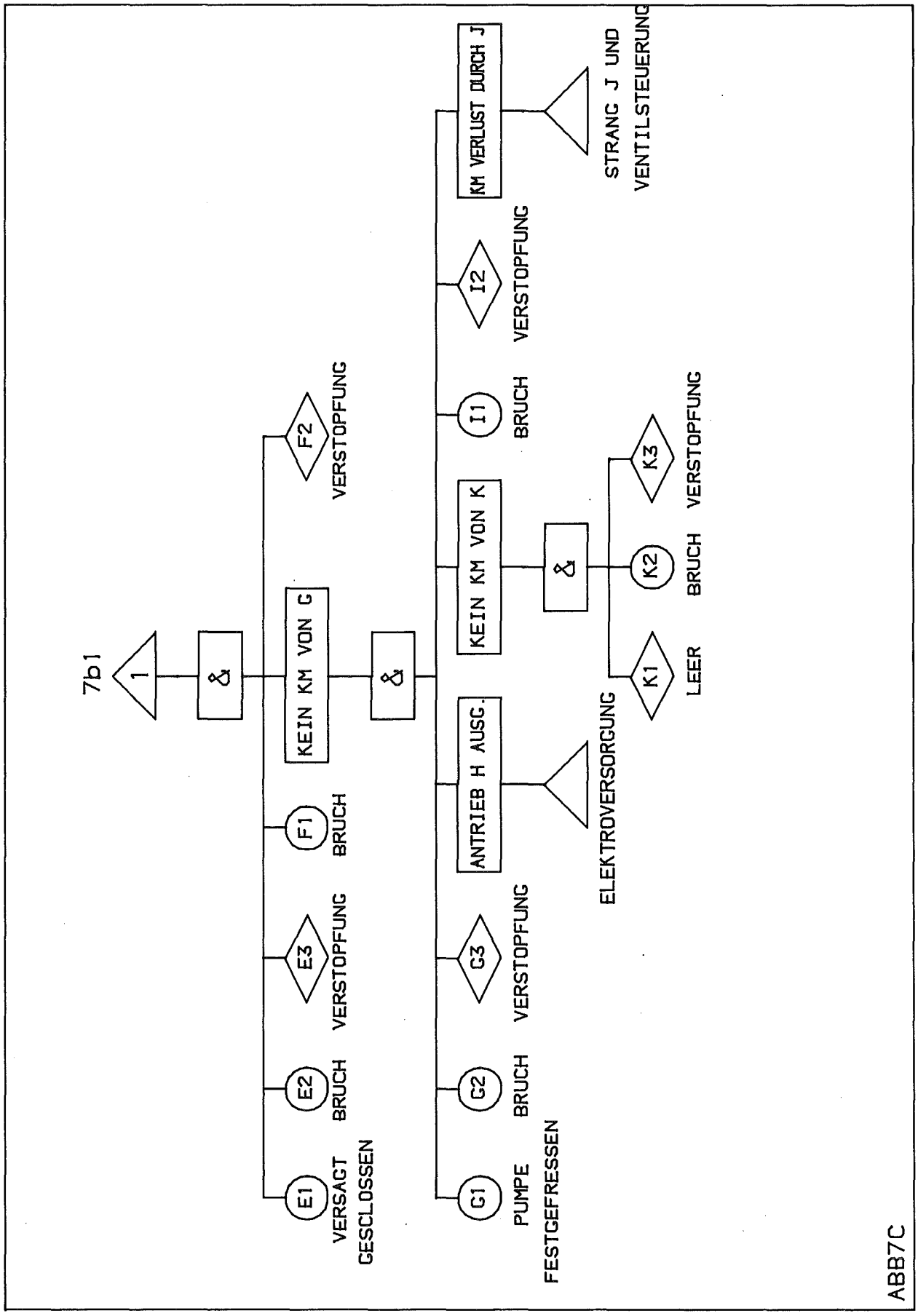


Abb.11b: Fehlerbaum eines Kühlsystems

1.2. Merkmale funktionellen Modellierens und Folgerungen für ein CAD-System

Wie aus den vorangehenden Kapiteln zu entnehmen ist, wird in vielen vom Problem her sehr unterschiedlichen Bereichen die gleiche Methodik eingesetzt. Dabei handelt es sich um Vorgehensweisen, die für alle Entwurfsbereiche ihre Gültigkeit haben und andere, deren Anwendbarkeit sich auf den funktionellen Entwurf beschränkt. Die folgenden Überlegungen gehen von diesem einführenden Überblick aus und präzisieren durch Abstraktion einige Merkmale, um Forderungen - graphisch durch einen vorangestellten Pfeil (==>) hervorgehoben - an ein CAD-System zur Unterstützung des Modellierens zu belegen. Diese Forderungen sind, isoliert betrachtet, zum Teil Stand der Technik, sie erfordern erst in der Kombination neue Systemlösungen.

Die Merkmale des funktionellen Modellierens und ihre Folgen für ein CAD-System werden unter drei Aspekten zusammengefaßt: der Struktur des Modellierens, dem Datenmodell des funktionellen Modellierens und der graphischen Darstellung der Entwurfsdaten.

Struktur des Modellierens

Konstruieren oder Planen sind iterative Prozesse, d.h. die einzelnen Teilprozesse (wie beispielsweise die Anlagenplanungsprozesse in Abb.3) werden mehrmals durchlaufen, um vorläufige Ergebnisse schrittweise zu verbessern. Jeder dieser Teilprozesse des Konstruieren/Planens bei denen es sich jeweils um ein Modellieren handelt, zerfällt im wesentlichen in zwei Schritte: die Synthese und die Analyse /12/. Abb.12 zeigt die grundsätzliche Struktur des Modellierens, ohne Bezug auf eine etwaige Unterstützung durch die EDV.

Bei der Synthese wird ausgehend von Spezifikationen und Daten aus Vorphasen unter Berücksichtigung von Entwurfshilfsdaten (Kataloge, Regelwerke etc.) ein Modell erstellt, das entsprechend der jeweiligen Arbeitsphase und den eingesetzten Arbeitshilfen eine Zeichnung, ein EDV-Modell oder eine mathematische Gleichung ist. An die Synthese schließt sich eine Analyse des Modells an. Dabei sind drei Bereiche zu unterscheiden. Bei der manuellen Analyse wird das Modell (z.B. eine Zeichnung) vom Ingenieur beurteilt, mit Kollegen diskutiert (engineering judgement). Das Ergebnis sind etwaige Modellierungsfehler und Urteile über das Modell. In der algorithmischen Analyse wird das Modellverhalten simuliert, es werden Berechnungen durchgeführt, die möglicherweise ebenfalls Modellierungsfehler erkennen lassen und Kennwerte ergeben. Die Ergebnisse dieser beiden Teilphasen werden manuell bewertet (im

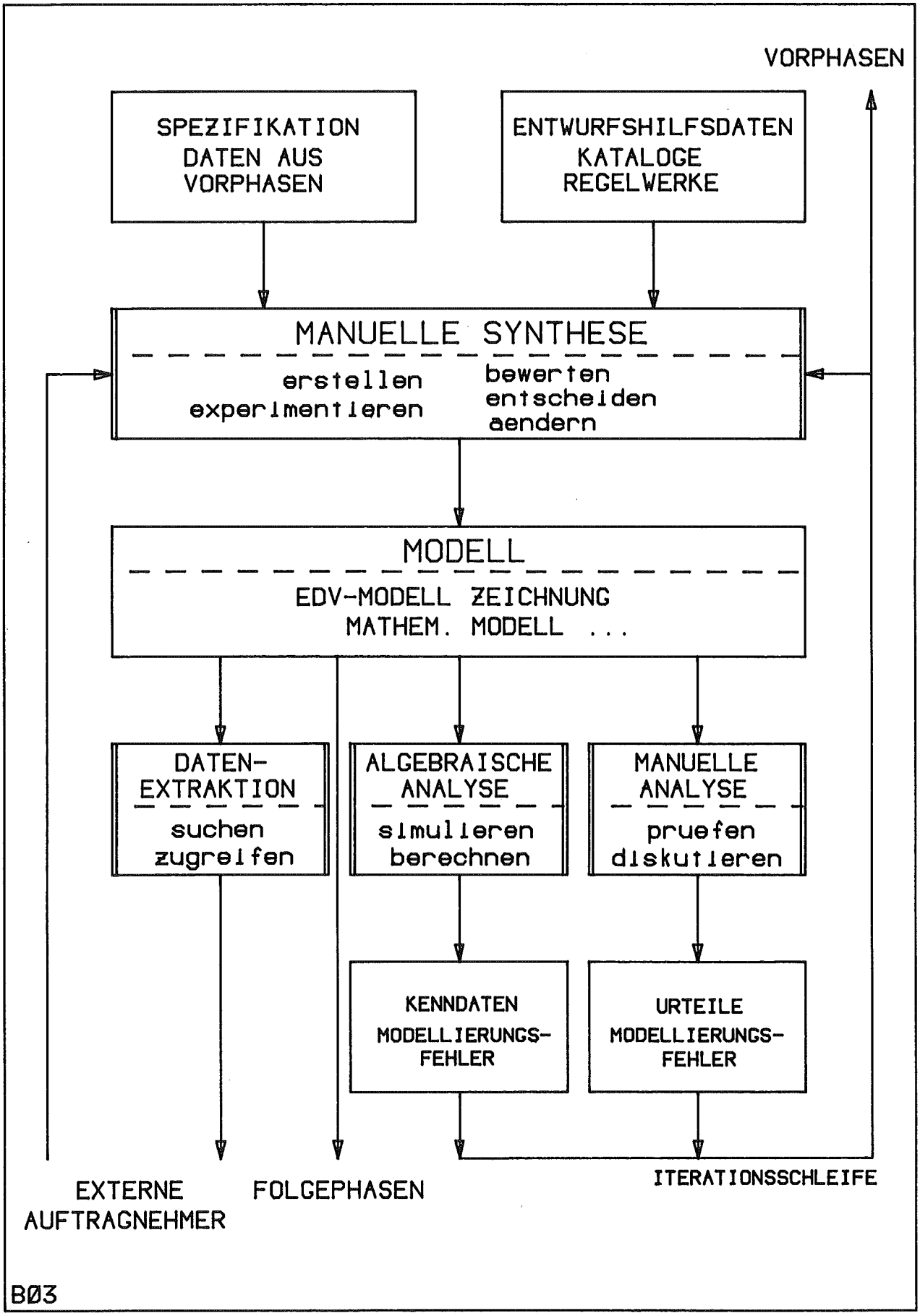


Abb.12: Allgemeine Struktur des Modellierens

allgemeinen) und gemäß ingenieurtechnischer Entscheidungen wird daraufhin das Modell geändert. Diese Iterationsschleife kann sich auch wie Abb.12 zeigt über mehrere Phasen erstrecken. In der Anlagenplanung beispielsweise könnte sich beim räumlichen Modellieren ergeben, daß gewisse Behälter nicht unterzubringen sind und daher der Prozeß umstrukturiert werden muß.

Bei der Datenextraktion werden dem Modell lediglich Daten entnommen, die in externen Prozessen benötigt werden (z.B. Stücklistenerstellung für den kaufmännischen Bereich).

Eine mögliche Unterstützung des Modellierens durch die EDV zeigt Abb.13. Im Zentrum des Systems steht ein interaktiver CAD-Prozessor über den der Benutzer Zugriff auf Spezifikationen, Entwurfshilfsdaten und auf eine Datenbasis hat, in der die Modelle verwaltet werden. Der interaktive Prozessor kann Teilprozesse für die interaktive, algebraische Analyse anstoßen.

Daneben spielen vor allem beispielsweise in der Anlagensicherheitstechnik (z.B bei der Fehlerbaumanalyse) Analysealgorithmen eine zentrale Rolle, die im Stapelbetrieb der EDV-Anlage laufen müssen, da die Rechenzeiten und die Speicherplatzanforderungen zu groß sind. Sie erzeugen vor allem Listen und Diagramme für die Modellbewertung. Falls das Modell entsprechend aufgebaut ist, können diese Algorithmen aber eventuell auch Modellwerte ändern.

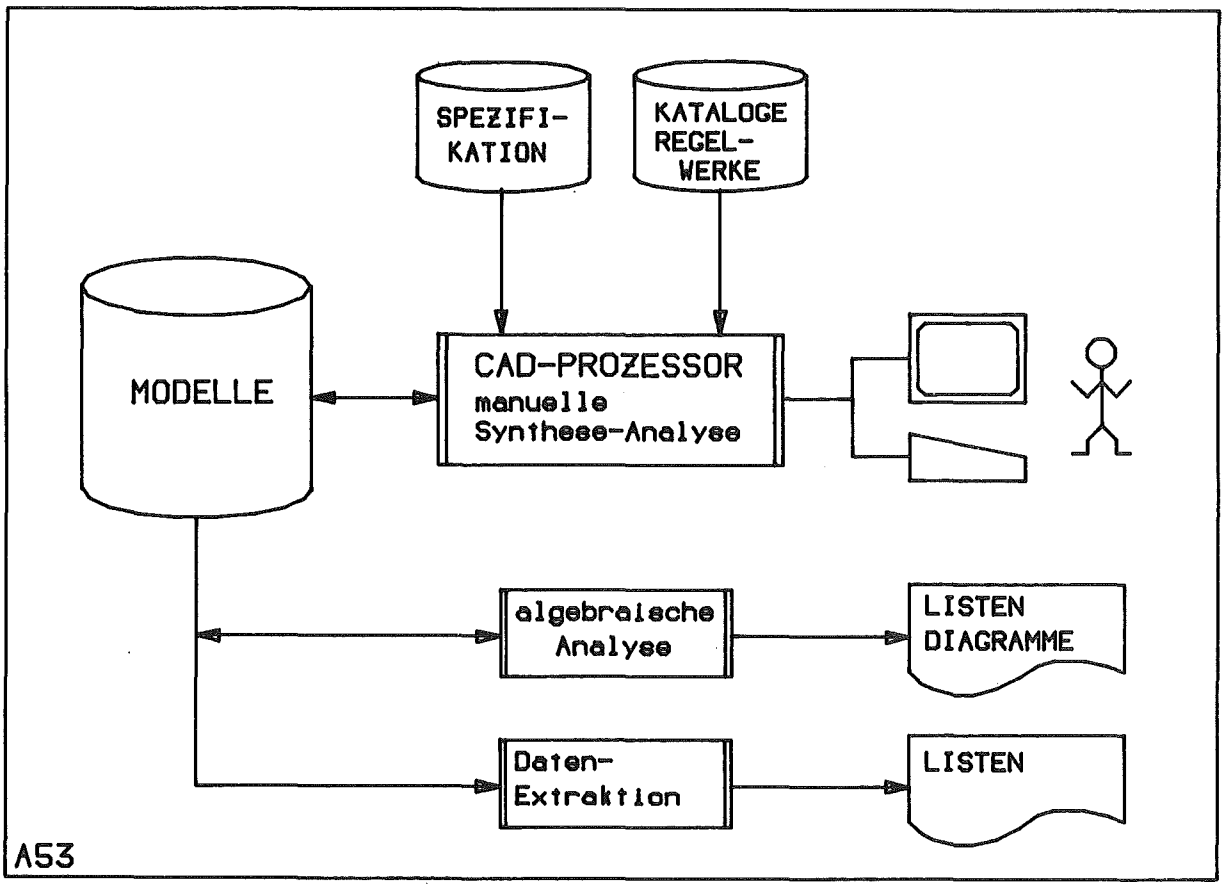


Abb.13: EDV-Unterstützung des Modellierens

Die Ergebnisse der Analysen und der externen Datenbearbeitung veranlassen den Entwurfsingenieur im allgemeinen zu Entwurfsmodifikationen. Diese Rückkopplung muß sich nicht auf eine Phase beschränken, sondern kann mehrere Phasen umfassen. Es lassen sich also im wesentlichen drei Aktivitäten unterscheiden, die für das Modellieren wichtig sind:

- Ideen formulieren, Aufbau des Modells, Experimentieren mit dem Modell
- Verwendung des Modells zur Kommunikation mit Kollegen
- Auswertung des Modells mit Hilfe von Programmen

Daher sollte ein CAD-System die

==> interaktive Manipulation mit Standardoperationen wie Einfügen, Löschen von Objekten, Verbinden, Trennen von Objekten, Eingeben und Ändern von Attributwerten, Verwalten von Modellen und Teilmodellen unterstützen, eine

==> lesbare graphische Darstellung des Modells und eine

==> problemorientierte Sprache für die Formulierung der Datenauswertung bieten.

Durch die Zweistufigkeit (Synthese-Analyse) des Entwurfsprozesses ergibt sich die Frage, wo in einem CAD-System zweckmäßigerweise die Prüfung der Modellkonsistenz, die Prüfung der Zulässigkeit, der Richtigkeit des Modells erfolgen sollte. Es liegt nahe, diese Prüfung mit der Analyse zu koppeln, da diese Algorithmen die Modellrestriktionen kennen. Das hat aber den Nachteil, daß die Prüfung zu einem Zeitpunkt erfolgt, zu dem Entwurfsfehler möglicherweise bereits eine Reihe weiterer Fehlentscheidungen nach sich gezogen haben und dann umfangreiche Korrekturen erforderlich werden (Fehlerlisten, Abb.14).

Eine programmtechnisch flexible Lösung besteht darin, die Modellprüfung als selbständigen Prozeß zu implementieren, der der Analyse vorgeschaltet wird. Diese Lösung läßt eine schnellere Reaktion erwarten, da die Anforderungen an die Betriebsmittel im allgemeinen geringer sind, als bei der eigentlichen Analyse. Die wesentlichen Nachteile der ersten Lösung bleiben aber erhalten. In der Realität würde man diesem Zwischenprozeß noch die Aufgabe der Datentransformation zuordnen, um so den Syntheseprozeß stärker von der Analyse zu entkoppeln. Verwendet man zur Synthese beispielsweise ein Zeichensystem, so könnte diesem Zwischenprozeß neben der Datenprüfung die Interpretation der graphischen Daten zugeordnet werden.

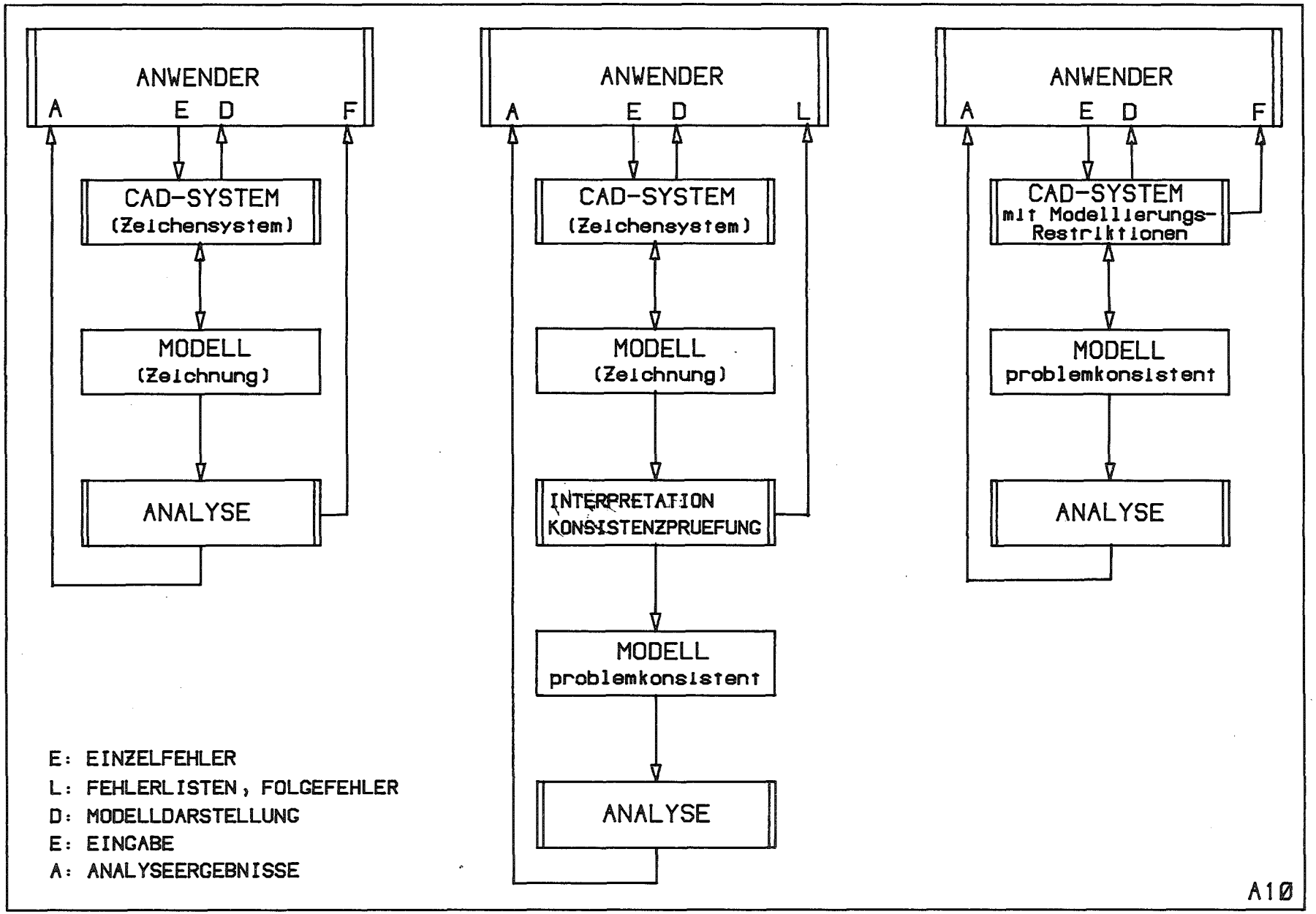


Abb.14: Grundstrukturen von CAD-Systemen für das Modellieren

Stattet man dagegen bereits das Teilsystem zur Syntheseunterstützung mit dem nötigen Problemwissen aus, teilt man ihm die jeweiligen Modellrestriktionen mit, so erkennt es falsche Entwurfsentscheidungen sofort und verhindert damit Folgefehler. Das ist besonders dann sehr wertvoll, wenn die Analysealgorithmen im Stapelbetrieb laufen müssen (Fehlerbaumanalyse). Es sollten daher

===> problembezogene Systemhilfen beim Modellieren (Synthese)
zur Verfügung stehen.

Charakteristisch für den Entwurf ist in vielen Bereichen die Kurzlebigkeit der Methoden und ihre Instabilität. Beides ist auf eine dauernde Fortentwicklung zurückzuführen. (z.B. Fehlerbaum-Modelltyp der Sicherheitsmodellierung).
Die Betrachtung der Anlagenplanung insgesamt und der sicherheitstechnischen Modellierung hat gezeigt, daß verschiedene Modellklassen nebeneinander einander ergänzend eingesetzt werden (Fehlerbaummethode, Ereignisablaufanalyse, R&I-Schemata). Ein CAD-System, das die Bearbeitung verschiedenster Modelltypen erlaubt, um dem Anwender eine einzige Schnittstelle anzubieten, muß sich daher durch

===> leichte Anpassung an neue Problembereiche (Modellklassen) und
===> dynamischer Modellklassenwechsel auszeichnen.

Beim funktionellen Modellieren sind Tätigkeiten, die hohe Flexibilität und direkte Reaktion erfordern (Synthese und manuelle Analyse, Abb.12), eng mit Arbeiten verzahnt, die hohe Anforderungen an Betriebsmittel der Rechner stellen (Analyse) und daher im Stapel - Betrieb bearbeitet werden sollten. Daher ist eine

===> Integration von interaktivem Betrieb und Stapelverarbeitung
wüschenswert, die die Anwendung des Systems erleichtert.

Datenmodell des funktionellen Modellierens

Um ein CAD-System zu befähigen, verschiedene Modellklassen entsprechend problemspezifischer Modellklassenrestriktionen zu bearbeiten, sollte eine geeignete Sprache zur Beschreibung der Modellklassen (Daten- oder Modellklassendefinitionssprache) zur Verfügung stehen. Eine solche Sprache basiert zweckmäßigerweise auf einer problemgerechten abstrakten Vorstellung von den zu bearbeitenden Modellklassen, d.h. auf einem Datenmodell, das dem Problembereich des funktionellen Modellierens angepaßt ist. Nur so lassen sich Modellklassen-Restriktionen einfach und benutzerfreundlich formulieren.

Die folgende Betrachtung verfolgt daher das Ziel, eine Gruppe von Begriffen abzugrenzen, die die Basis für eine Sprache zur Beschreibung von Modellklassen bilden könnten. Dabei wird zwischen der Beschreibung der funktionellen Modellstruktur, der Beschreibung der problemorientierten Datenorganisation und der graphischen Darstellung der Daten unterschieden. Ein Datenmodell in diesem Sinne ist eine abstrakte Vorschrift, zur Einschränkung der Entwurfsfreiheit, um ein gültiges Modell zu erhalten, das den Restriktionen der Analyse (oder anderer Folgeprozesse) entspricht.

Problemmodellierung

Funktionelles Modellieren besteht darin, Bauelemente (Objekte) einer spezifischen Menge zu einem System zu verknüpfen, das eine vorgegebene Funktion erfüllt. Die bereitstehenden Elemente besitzen Eigenschaften und die Fähigkeit, mit ihrer Umgebung zu kommunizieren. Diese Fähigkeit wird durch Anschlüsse charakterisiert. Die Verknüpfung der Objekte erfolgt im Rahmen fest vorgegebener Relationen, d.h. durch Bildung von Objektpaaren. Die Entwurfsrestriktionen beziehen sich auf die Attributwerte der Objekte und auf Verbindungen zwischen Objekten (Tupel der Relationen).

Dieses Datenmodell basiert auf bewerteten gerichteten Graphen /13/, es läßt sich vereinfacht formal als Graph G darstellen:

$$G = (V, R , v, r)$$

mit der Knotenmenge V, der Kantenmenge $R \subset V \times V$ und den beiden Abbildungen v: V->W(V) und r: R->W(R), die den Knoten und Kanten Werte der Wertemenge W zuordnen .

Der Entwurfsingenieur unterscheidet in seiner Vorstellung, wie es auch durch die Blockdiagramme zum Ausdruck kommt, zwischen Objekten und expliziten Objektassoziationen /14/ (Verbindungen zwischen Objekten). Sowohl den Objekten als auch den Relationen ordnet er Eigenschaften zu; den Objekten darüberhinaus Anschlüsse, die die Fähigkeit eines Objektes beschreiben, sich in eine Umgebung einzufügen.

Die Problemabgrenzung wird durch eine spezifische Menge von Datenelementen für den Modellaufbau und Entwurfsrestriktionen beschrieben, die sich auf Objektwerte oder auf die Zulässigkeit von Objektverbindungen in einem bestimmten Kontext beziehen. Die

===> Verwendung eines graphentheoretischen Datenmodells als Basis einer Modelltypbeschreibung

ist daher naheliegend.

Eine Modellklasse bzw. Restriktionen für die Operationen zur Modellbearbeitung sind Daten, also etwas Statisches. Aus diesem Grunde ist eine

===> deskriptive Modelltypbeschreibung

problemgerechter als die Formulierung durch Algorithmen.

Problembezogene Datenorganisation

Um große Systeme erfolgreich bearbeiten zu können, gliedert man sie unter Wahrung der richtigen Beziehungen in überschaubare Teilsysteme (Interessenbereiche) und behandelt sie auf einer höheren Abstraktionsstufe als einfache Objekte mit Eigenschaften und Anschlüssen (Abb.15). Sie verlieren also ihre innere Struktur. Umgekehrt beschreibt man aber auch vorerst unstrukturierte Objekte nachträglich auf einer Ebene geringerer Abstraktion durch elementare Bausteine und deren Verbindungen. Diese Techniken des schrittweisen Verfeinerns oder Abstrahierens sind aus der Systemtechnik bekannt. In der Blockdiagrammtechnik bedeutet dies die wechselweise Darstellung eines Teilsystems durch einen Block oder ein Blockdiagramm.

Entscheidend dabei ist die Idee der System/Teilsystem-Grenze , die durch die Anschlüsse erfaßt wird. Diese Schnittstellen haben die Aufgabe, funktionelle, geometrische oder kaufmännisch-organisatorische Übergangsbedingungen zu erfassen. Schnittstellen werden so gewählt, daß möglichst wenige und einfache Ein/Ausgänge entstehen. Die Gliederung eines Systems in Teilsysteme und deren Referenz aus Teilsystemen höherer Abstraktionsstufe stellt das Gesamtsystem als Hierarchie von Teilsystemen dar. Es handelt sich um die aus der Systemtechnik bekannten Aufgaben des Konfigurations- und Schnittstellenmanagements /15/.

Typisch für den Entwurf ist dabei, daß zu einem Teilsystem oft alternative Entwürfe erstellt werden, die erst im Verlauf der weiteren Arbeit wieder entfallen, wenn man sich auf Grund von Analysen für eine Lösung entschieden hat (Abb.16).

Diese Gesichtpunkte führen zur Forderung

===> problembezogener Systemhilfen für die Verwaltung von Teilsystemen, Schnittstellen, Abstraktionsebenen und Alternativentwürfen.

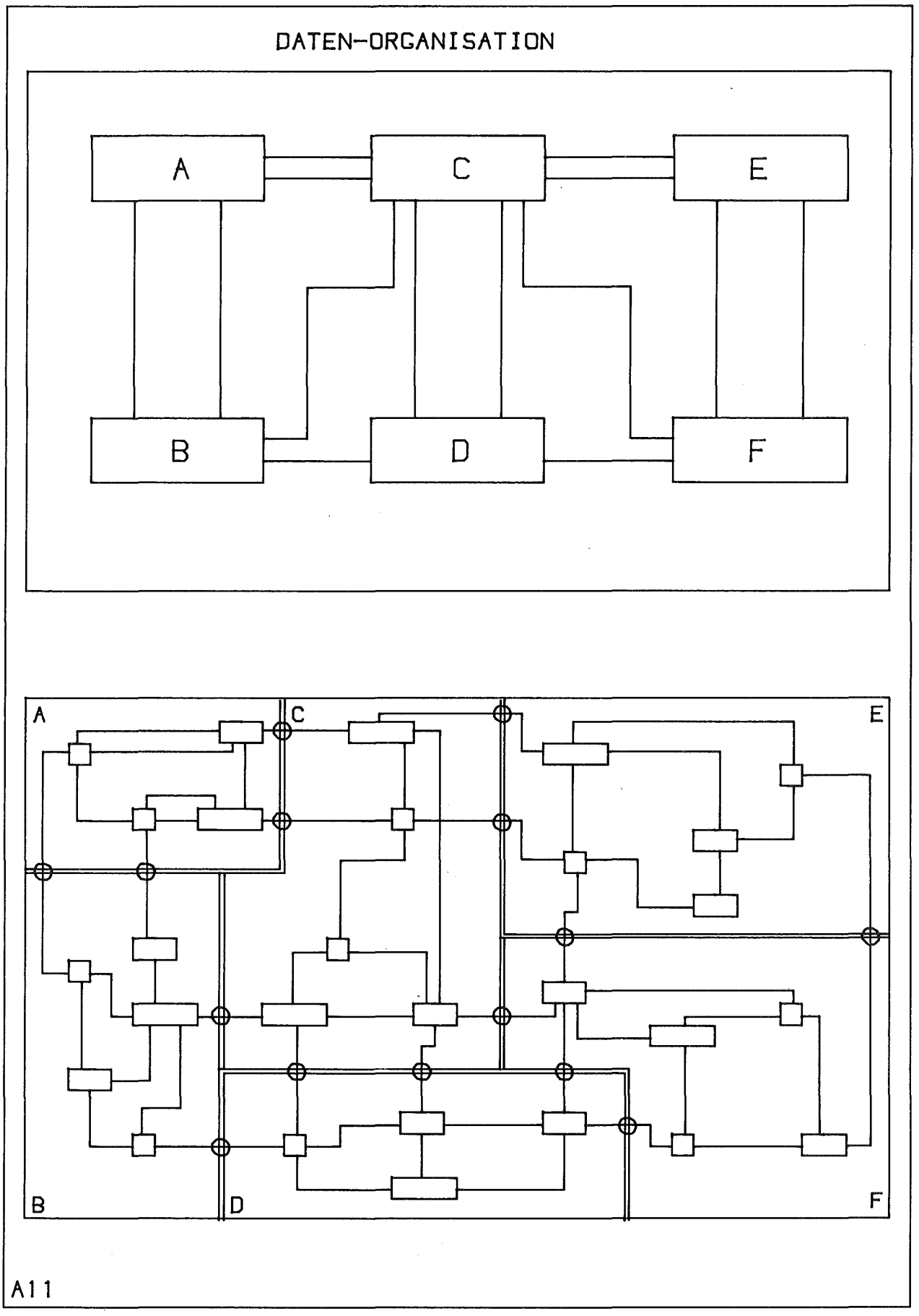


Abb.15: Datenorganisation zur Erhöhung der Überschaubarkeit

Datendarstellung und Datenidentifikation

Da eine lesbare graphische Darstellung des Modells anzustreben ist, läßt sich keine Standarddarstellung verwenden, die vom jeweiligen Problembereich unabhängig ist. Daher muß eine problemspezifische graphische Beschreibung der Modellkomponenten die logische Modelltypbeschreibung ergänzen. Allerdings beschränkt sich dabei im Interesse einer wirklich übersichtlichen graphischen Darstellung (Blockdiagramm) die Beschreibung auf die Symbolik. Die Symbolanordnung und die Leitungsführung sollte dem Anwender überlassen bleiben, da sich kein allgemeines Kriterium für die Lesbarkeit einer graphischen Darstellung angeben läßt.

Die Beschreibung der symbolischen Darstellung der Modelldaten sollte die starke Strukturierung dieser Informationen (Objekt, Anschluß, Attribut, Relation, Verbindung) berücksichtigen, vor allem auch, um ihre Identifikation im interaktiven Betrieb durch Zeigen (PICK) zu ermöglichen. Entsprechend der üblichen Blockdiagrammtechnik sollte auch die Freiheit gegeben sein, Attributwerte entweder durch graphische Zusatzsymbole des Grundsymbols (parametrisiertes Symbol) oder durch ergänzende Texte darzustellen. Als Beispiel zeigt Abb.17 diese verschiedenen Möglichkeiten der graphischen Darstellung von Attributwerten durch Texte (Typ-Attribut) und durch graphische Symbole (Negationssymbol, Wasserstand im Behälter).

Ein CAD-System für die interaktive Bearbeitung komplexer technischer Objekte erfordert daher eine

==> strukturierte und parametrisierte Symbolik.

Diese graphische Beschreibung der logischen Datenobjekte entspricht in ihrer Bedeutung, von der Identifikation abgesehen, den Ausgabeformaten für Variable in Programmiersprachen wie FORTRAN oder PL/1. In diesen Sprachen legt man mit einem sogenannten FORMAT-Ausdruck fest, wie der Wert einer Variablen bei der Datenausgabe graphisch durch Elementarzeichen abzubilden ist.

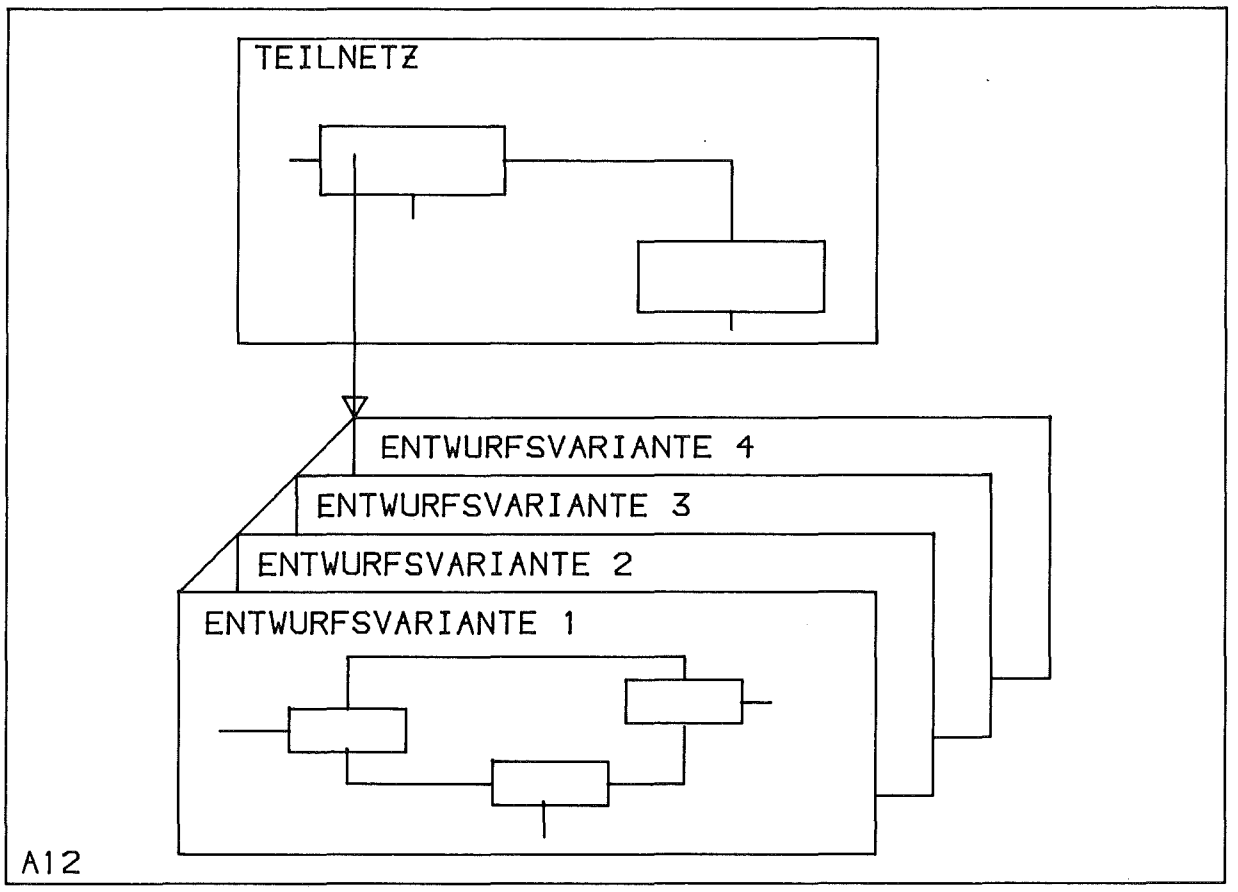


Abb.16: Entwurfsalternativen

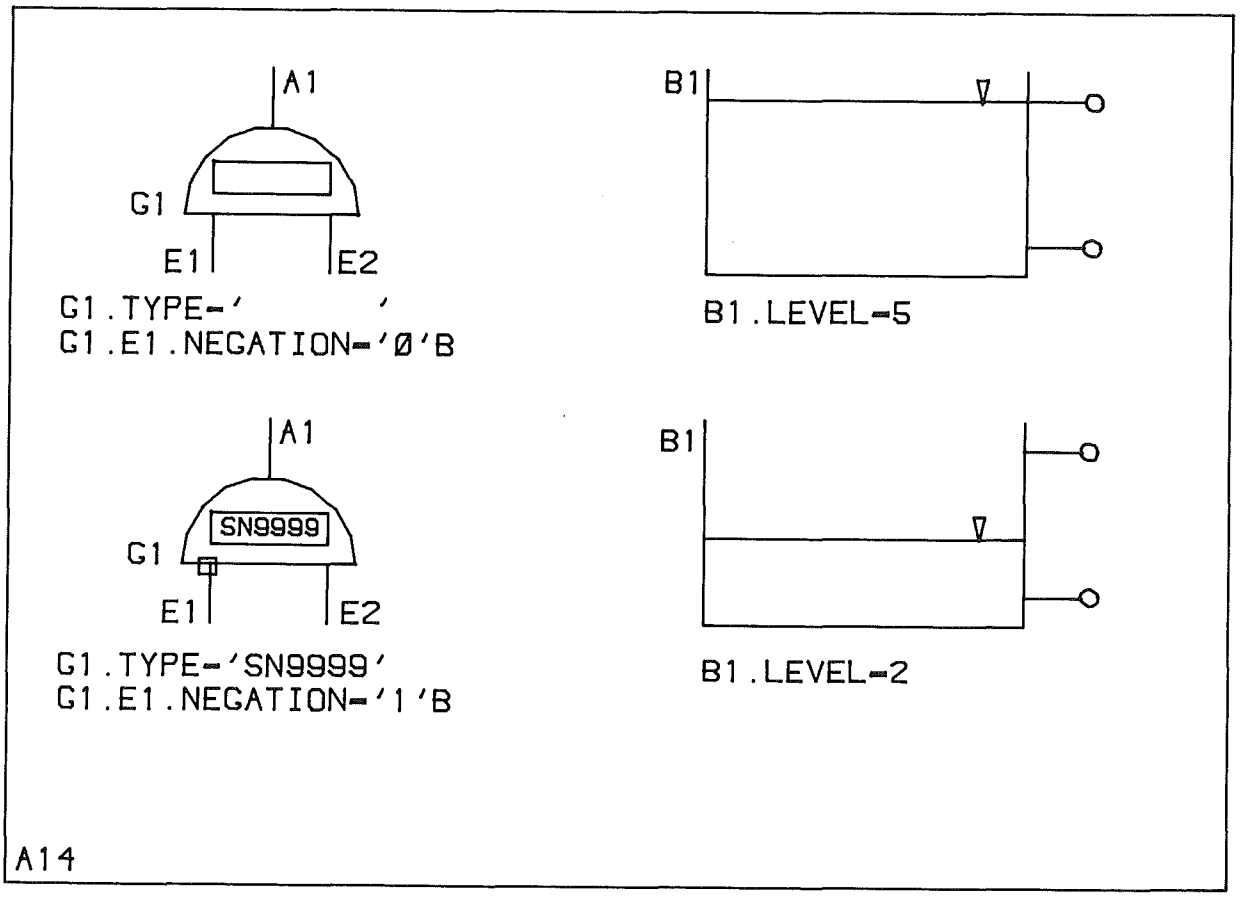


Abb.17: Strukturierte und parametrisierte Symbolik

1.3. Zielsetzung der Arbeit

Wie in diesen einleitenden Darstellungen gezeigt wurde, läßt sich im Bereich der Konstruktion/Planung das funktionelle Modellieren als typischer Arbeitsablauf mit spezifischen Anforderungen an eine EDV-Unterstützung abgrenzen, so daß es gerechtfertigt ist, für diese Arbeitsmethodik ein CAD-System zu entwickeln. Die Untersuchung des Standes der Technik zeigt, daß die zur Unterstützung dieses Bereiches in Frage kommenden EDV-Systeme die aufgestellten Forderungen nicht erfüllen.

Grundlage der Entwicklung ist dabei die Methodik des Arbeitens mit Blockdiagrammen, die für die EDV-Unterstützung adaptiert wird. Als Informationsspeicher wird eine Datenverwaltung eingesetzt, die die Modelle unter Berücksichtigung problemtypischer Restriktionen verwaltet. Blockdiagramme (Zeichnungen) dienen lediglich als eine Möglichkeit der externen Darstellung der Modelle.

Das System gestattet sowohl interaktives Modellieren als auch die Modellbearbeitung (insbesondere die Modellanalyse) im Stapelbetrieb.

Da dieses CAD-System als "Methode" zum Erstellen, Ändern und Verwalten von funktionellen Modellen betrachtet und als solche in das CAD-Methodenbanksystem REGENT eingefügt wird, ist eine leichte Kopplung mit den verschiedensten Modellanalyse-Methoden gewährleistet.

2. Stand der Technik bei EDV-Systemen für das funktionelle Modellieren

Wie im einführenden Kapitel gezeigt wurde, ist das Blockdiagramm in der Phase des funktionellen Modellierens ein wichtiges Hilfsmittel des Ingenieurs. Es repräsentiert Systemkomponenten durch Symbole und Verbindungen zwischen den Komponenten durch verschiedenartige Linien. Sowohl die Symbole als auch die Verbindungen werden durch Texte oder auch graphische Hilfsmittel (Zusatzsymbole, Linienarten etc.) ergänzt, um neben der Systemtopologie weitere Informationen über das System zu erfassen. Überschreitet die Menge dieser Informationen ein gewisses Maß, so daß die Übersichtlichkeit gefährdet wird, ergänzt man sie meistens durch Tabellen (Apparatelisten etc.).

Die EDV-Unterstützung dieses Aufgabenbereiches begann mit der Erstellung von Blockdiagrammen durch allgemeine oder auch spezielle Zeichensysteme. Für die Bearbeitung größerer Informationsmengen insbesondere auch unter dem Gesichtspunkt der Datenqualität werden in den letzten Zeit immer stärker Datenbanken eingesetzt. Ein drittes Hilfsmittel, das vor allem die Implementierung und die Handhabung von Analysealgorithmen unterstützt, bilden die integrierten CAD-Systemkerne. Die Charakteristika und spezifischen Fähigkeiten dieser Systemetypen im Hinblick auf das funktionelle Modellieren werden im folgenden untersucht.

2.1. Graphische CAD-Systeme

Bei dieser Gruppe von CAD-Systemen, die für das funktionelle Modellieren geeignet sind, steht der Aspekt des Zeichnens von Blockdiagrammen im Vordergrund. Ausgehend von allgemeinen Zeichensystemen verlief die Entwicklung schnell hin zu Systemen, die an die spezifischen Belange der Blockdiagrammtechnik angepaßt waren, um die Eingabe, aber auch die Auswertung der Daten zu erleichtern. Dabei ging es darum, die Topologie des modellierten Systems und problemspezifische Eigenschaften der Systemkomponenten für die Weiterverarbeitung bereitzustellen. Es lassen sich folgende Gruppen von Zeichensystemen unterscheiden:

- Allgemeine Zeichensysteme mit Symboldefinition
- Blockdiagramm-Zeichensysteme
- Systeme mit problembezogener Datenstruktur

Im folgenden werden die verfügbaren Datenstrukturen und operativen Fähigkeiten dieser Systemklassen kurz dargestellt und ihre Brauchbarkeit für das funktionelle Modellieren beurteilt.

2.1.1. Allgemeine Zeichensysteme mit Bildsegmenten

Wird ein derartiges System eingesetzt, so definiert man die benötigten Symbole als Bildsegmente und faßt sie in einer Bibliothek zusammen, die als problemspezifisch vom Anwender nicht verändert werden darf. Beim Einfügen von Symbolen in ein Blockdiagramm wird eine Bilddatei aufgebaut, die je nach Typ des Zeichensystems entweder Kopien der Symbole oder Referenzen auf die Symbole, Verbindungslinien und ergänzende Texte enthält. Der Anwender hat dafür zu sorgen, daß die Bildelemente ein gültiges Blockdiagramm darstellen. Das System schränkt den Anwender nicht ein. Aus der Datenstruktur ist weder die Topologie direkt zu entnehmen, noch die Bedeutung der Texte. Eine Auswertung der Bildinformationen ist nur durch die Interpretation geometrischer Daten (Mustererkennung) möglich, eine nicht triviale Aufgabe. Ein System dieser Klasse ist der Graphic Function Manager (GFM) für die TEKTRONIX-4081 /16/. Diese Systeme können lediglich das Zeichnen erleichtern, sie bilden keine geeignete Basis für ein CAD-System, das neben der Synthese auch die Weiterverarbeitung der Daten umfassen soll.

Ein anderes System dieser Klasse, das System IGDS /17/, geht einen Schritt weiter und unterstützt die Bearbeitung von Daten, die die Graphik ergänzen. Dazu wird mit Hilfe einer Datendefinitionssprache (DDL) eine Menge von Satztypen definiert. Jedem graphischen Element kann dann ein solcher Satz zugeordnet werden. Die Satzelemente werden durch einen Namen, einen Datentyp und den zulässigen Wertebereich beschrieben. Die so aufgebaute Datenbasis enthält nicht die Topologie des erstellten Blockdiagramms, die Daten der Datenbasis werden unabhängig von den zugehörigen Symbolen als Texte dargestellt. Auch bei diesem System muß die Topologie des Blockdiagramms durch Interpretation der graphischen Daten abgeleitet werden.

2.1.2. Zeichensysteme für Blockdiagramme

Beschränkt man die Fähigkeiten eines Zeichensystems auf die Bearbeitung von Blockdiagrammen, so kann man ihm einige Aufgaben übertragen, die die Arbeit des Anwenders erleichtern. Derartige Systeme überwachen vor allem die Einhaltung von Restriktionen, die der Blockdiagrammtechnik entsprechen:

(1) Ein Blockdiagramm besteht aus einer beschränkten Zahl von Symbolen, die durch Linien verbunden werden.
(2) Ein Symbol wird durch seine Graphik, durch Anschlußpunkte und Zuordnungspunkte für Texte charakterisiert.
(3) Verbindungslinien verlaufen nur zwischen Anschlußpunkten, Texte sind immer Textpunkten zugeordnet.

Die Kenntnis dieser Regeln gestattet es, eine Datenstruktur aufzubauen, der die Topologie des Modells und Objekteigenschaften in Form der Texte leicht entnommen werden können. Außerdem kann ein derartiges CAD-System überprüfen, ob Anschlußpunkte oder Attribute unberücksichtigt blieben. Diese oder ähnliche Fähigkeiten besitzen Systeme wie DESIGNPAD /18/, CABLOS /19/, CASS /20/, LEADS /21/, AGS877 /22/ und /23/, /24/, /25/, /26/, /27/, /28/, /29/, /30/.

Die Systemhilfen zur Fehlervermeidung beschränken sich darauf, daß nicht beliebige Graphik, sondern nur Blockdiagramme erzeugt werden können. Dem Anwender bleibt es überlassen, zu überprüfen, ob die Blockdiagramme in dem aktuellen Entwurfskontext gültige Modelle darstellen.

2.1.3. Systeme mit problembezogener Datenstruktur

Neben den mehr von der Graphik bestimmten Systemen wurden Systeme vorgeschlagen, die die Bearbeitung der Netzwerktopologie zur Systemgrundlage machen, eine Problemanpassung der Datenstruktur erlauben, damit die Systemintelligenz erhöhen und die Auswertung der Modellinformationen erleichtern.

AEDNET

Diese Systeme basieren zum Teil auf Ring-Datenstrukturen wie beispielsweise AEDNET /31/, das zwar einige allgemeine Implementierungswerkzeuge bereitstellt, aber letztlich speziell auf die Technik elektrischer Schaltkreisentwicklung zugeschnitten wurde. Die AEDNET-Algorithmen zur Datenstruktur-Bearbeitung unterscheiden folgende Elementtypen zum Aufbau von Netzen:

- Knoten: repräsentieren die Verzweigungen in einem elektrischen Netzwerk,
- Element: durch Namen, Zahl der Anschlüsse, Werte und ein graphisches Symbol charakterisiertes Netzwerkobjekt,
- Teilnetz: ein Element, dem eine Struktur zugeordnet ist, bestehend aus Knoten, Elementen und anderen Teilnetzen (aquivalent circuit).

Die typspezifischen Eigenschaften der Netzwerkelemente werden innerhalb der Datenstruktur in sogenannten "generic elements" erfaßt, die der Menge der Elemente dieses Typs zugeordnet sind.

Das System kennt drei Relationen. Die erste faßt alle Anschlüsse zusammen, die zu einem Knoten gehören, die zweite ordnet alle Elemente eines Typs dem entsprechenden "generic element" zu, und die dritte verknüpft Teilnetze mit der zugehörigen Struktur.

Es handelt sich hier um ein Softwarepaket zur Datenstrukturmanipulation, das vor allem Netze behandeln kann, wie sie in der Schaltkreistechnik gebräuchlich sind. Alle problemorientierten Prüfungen der Eingabe bleiben den zu programmierenden Anwendungsprogrammen überlassen. Nur sie kennen die Bedeutung der Werte, die den Netzwerkelementen zugeordnet werden können. Die Datenstruktur-Routinen verwalten nur deren Speicherplatz.

GRAPHPROZESSOR

Ein anderes System, gedacht als Implementierungsbasis für Anwendungsprogramme, die auf allgemeinen Graphen und Graphenoperationen beruhen, ist der GRAPHPROZESSOR /32/. Die Datenstruktur besteht aus Knoten (Kreise, Punkte) und Kanten (Linien), denen Wertefelder zugeordnet werden können. Die Systemmoduln umfassen die bekannten Graphenoperationen wie: Hinzufügen und Löschen von Knoten und Kanten, Hinzufügen von Werten, Identifizieren von Knoten und Kanten durch die Graphik oder durch Werte, Erzeugen von Teilgraphen in Abhängigkeit von Knotenwerten, Suchen der kürzesten Wege, Erstellen der Adjazenzmatrix etc.

Auch hier sind das Problemwissen und die anwendungsspezifischen Operationen in Algorithmen zu formulieren. Zur Implementierung von Anwendungssystemen steht ein Systemkern bereit.

NEDLAN

Während bei AEDNET und GRAPHPROZESSOR die Anpassung an Anwenderprobleme durch Problemalgorithmen formuliert werden muß, wird in einem weiteren System /33/, /34/, /35/ die Anpassung in einem Prozess der Arbeitsvorbereitung mit Hilfe einer Netzdefinitionssprache NEDLAN vorgenommen.

Mit NEDLAN werden in einer modifizierten Backus-Naur-Form (nicht sehr anwendungsfreundlich) die für ein Problem jeweils zulässigen Objekttypen durch ihre Graphik, ihre Problemattribute (Zeichenketten) und ihre Strukturierungsfähigkeit beschrieben. Das Datenverwaltungssystem basiert ähnlich wie AEDNET auf Ringstrukturen. Es kennt zwei Systemrelationen: die Bildung von Elementmengen und Elementverbindungen. In NEDLAN wird daher für jedes Objekt spezifiziert, in wie viele Mengen es eingefügt und mit wievielen anderen Objekten es verbunden werden darf. Jedem Objekttyp sind zwei Werte zugeordnet, die die Objektbedeutung für die Auswerteprogramme charakterisieren (Objektklassen). Verbindungen sind keine speziellen Objekte, für das System sind alle Objekte gleichwertig. Die verschiedenen problemspezifischen Attribute und Objektklassen werden erst durch die bereitzustellenden Anwendungsprogramme ausgewertet. Das Basissystem erkennt lediglich, ob einem Objekt die richtige Zahl von Werten zugeordnet und ob die vorgegebene Zahl von Objektverknüpfungen eingehalten wurde.

Die Anwendungsprogramme müssen dafür sorgen, daß eine bezüglich des jeweiligen Problems richtige Datenstruktur aufgebaut wird. Auch bei diesem System handelt es sich im Grunde um ein Paket zur Datenstrukturbearbeitung, bei dem allerdings die strukturelle Anpassung der Datenelemente und ihre graphische Darstellung durch eine Sprache beschrieben und in Tabellen abgelegt wird.

OUTIL-GRAPHIC-ADAPTIF

Ein komplettes Werkzeug zur Bearbeitung von Graphen wird in /36/ vorgestellt. Es umfaßt einen festen Satz von Manipulationsfunktionen zur Bearbeitung von Graphen und einen Prozessor zur Beschreibung der interessierenden Graphenklassen. Sowohl die Klassenbeschreibung als auch die Manipulation von Graphen erfolgen interaktiv. Eine Graphenklasse wird durch eine Menge von Knotentypen und die Art der Verbindungen definiert.

Ein Knotentyp wird charakterisiert durch

- einen Namen,
- ein graphisches Symbol (sehr einfach, durch wenige Parameter erfaßt),
- einen Typ (einfacher Knoten oder Repräsentant eines Teilgraphen),
- zugeordnete Information (Text, dessen Bedeutung dem System unbekannt ist, dessen Eingabe als erforderlich deklariert werden kann),
- einen Eingang und einen Ausgang.

Verbindungen einer Graphenklasse sind entweder einfache Verbindungen oder Verbindungen mit Verzweigungen. Ebenso wie den Knoten, können auch den Verbindungen textliche Informationen zugeordnet werden, wobei ebenfalls festgelegt werden kann, ob diese Information beim Verbindungsaufbau vom System angefordert werden soll.

Folgende Operationen zur Manipulation von Graphen sind vorgesehen:

- Einfügen/Löschen von Knoten/Verbindungen
- Eingabe von ergänzender Information, falls für den Typ vorgesehen
- Ändern des Darstellungsmaßstabs

Das System umfaßt darüberhinaus Hilfsmittel zur Implementierung und zur Steuerung der Ausführung von Anwendungsprogrammen, die die Graphen bearbeiten. Diese Hilfsmittel steuern die Zusammenarbeit zweier Rechner (Intelligentes graphisches Terminal, Großrechner) und überprüfen, ob alle erforderlichen Eingabedateien für die Datenanalyse bereitstehen.

Eine Problemanpassung des Synthesesystems liegt auch hier nur im Ansatz vor: Es bleibt den Anwendungsprogrammen überlassen, zu prüfen, ob Entwurfsentscheidungen in dem aktuellen Kontext richtig waren.

GRADAS

GRADAS /37/ ist ein interaktives System zur Bearbeitung von Funktionsplänen, dessen Datenmodell auch auf die Blockdiagrammtechnik abgestimmt ist. Es kennt Objekte, die durch Graphik, Texte und Verweise auf andere Objekte charakterisiert werden. Die Beschreibung der Objekttypen und die Abarbeitung der Anwenderkommandos wird in algorithmischer Form in zwei Moduln (Moduln "Syntax" und "Kontext") erfaßt, so daß das System ansonsten anwendungsunabhängig ist.

Die verwendete Datenbank basiert auf DATAS /38/, das assoziative Datenstrukturen realisiert. Die Datenverwaltung arbeitet auf Objektebene: die innere Struktur der Objekte ist nur den Anwendungsmoduln bekannt, die ebenfalls vom System verwaltet werden. Eine problemgerechte Verarbeitung der Daten durch das Basissystem ist daher nicht möglich, sondern ist nur durch Anwendungsmoduln realisierbar.

2.1.4. Andere Systeme

Die bisher betrachteten Systeme können alle mehr oder weniger ausgeprägt an eine bestimmte Aufgabenstellung (Blockdiagramm-Klasse) angepaßt werden, indem eine Symbolbibliothek aufgebaut oder mit Hilfe einer Datendefinition oder algorithmisch in einem Prozeß der Arbeitsvorbereitung eine Problemklasse beschrieben wird, innerhalb der man dann modelliert.

PIXIN /39/ dagegen ist eine Wortsprache (Programmiersprache) zur Beschreibung von allgemeinen Netzwerken, die sowohl Blockdiagramme umfassen können als auch geometrische Netzwerke (z.B. Stabwerke). PIXIN kennt keine vorgegebenen Einschränkungen der Modellierungsfreiheit und überläßt damit dem Anwender bzw. den Anwendungsprogrammen die gesamte Eingabeprüfung.

Die PIXIN-Datenelemente sind:

- POINT, LINE: Grundelemente
- SET: Menge von Elementen
- QUALIFIER: Text oder Zahlenwert mit einer Positionsangabe für die Zuordnung zu einem Element
- MODIFIER: zur Angabe graphischer Parameter oder beliebiger Werte für Analyseprogramme
- ATTACHEMENT-POINT: Punkte, die referiert werden können, um Zuordnungen zu beschreiben
- PICTURE: geordnete Menge von beliebigen Elementen
- INSTANCE: Referenz eines PICTURE

Darüberhinaus gibt es eine Reihe von produkt- oder problemorientierten Systemen: z.B. zur Schaltkreisentwicklung /22/,/40/, /41/, /42/, /43/ oder zur Fehlerbaumanalyse /44/. Diese Systeme werden nicht näher vorgestellt, da sie der Zielsetzung nicht entsprechen, nämlich ein System bereitzustellen, das sich an die verschiedensten Aufgabenstellungen leicht anpassen läßt.

2.1.5. Kritik

Die in Kapitel 1 erläuterten Forderungen nach einem Werkzeug, das es gestattet, mit einem festen Satz von Operationen ein Modell problembezogen zu bearbeiten, wird von keinem System erfüllt, lediglich das OUTIL-GRAPHIC-ADAPTIF zeigt einen Ansatz in dieser Richtung. Die Problemanpassung beschränkt sich auf relativ einfache topologische Restriktionen (Zahl der zulässigen Anschlüsse, Anschlußtyp), wenn man sie nicht in Anwendungsmoduln formulieren will.

Alle anderen Systeme sind insbesondere auch nicht selbständig, d.h., sie erfordern die Programmierung der gewünschten Modellierungsfunktionen.

Keines der Systeme verwendet ein Datenmodell mit dem Datentyp "Relation", der in der Blockdiagrammtechnik eine Menge von Verbindungen (Objektpaaren) eines Typs charakterisiert. Das Fehlen dieses Typs aber erschwert die Beschreibung einer Reihe typischer Modellierungs-Restriktionen.

Es fehlt allgemein eine klare Trennung zwischen der Bearbeitung logischer Daten (Problemdaten) und ihrer graphischen Darstellung bzw. Identifikation. Das zeigt sich am deutlichsten daran, daß eine Manipulation der Problemdaten ohne Graphik (z.B. durch eine problemorientierte Sprache) nicht möglich ist. Der Anwender hat auch beispielsweise nicht die Freiheit, Objektwerte entweder graphisch durch eine geeignete Symbolik (Abb.17) oder durch Texte an bestimmten Positionen darzustellen bzw. auf eine Darstellung ganz zu verzichten und sie allein durch Änderungs- und Abfrage-Kommandos zu bearbeiten. Gerade das aber ist bei der Vielzahl der Parameter in vielen Bereichen sehr nützlich.

Nicht die Graphik sollte die Basis eines Systems zur Unterstützung des funktionellen Modellierens bilden, sondern ein problemorientiertes Datenverwaltungssystem, das die Graphik zur Datendarstellung und Identifikation verwendet.

2.2. Datenbanken als CAD-Hilfsmittel

Mit wachsendem Einsatz der EDV im CAD-Bereich wurde das Datenmanagement immer mehr zum zentralen Problem. Solange lediglich ein Programm oder ein Anwender mit einer bestimmten Datenmenge arbeitete, gab es keine Schwierigkeiten. Erst die Verwendung von Programmketten und der Mehrbenutzerbetrieb führten dazu,daß bei Änderung der Datenstruktur für ein Programm mindestens auch ein weiteres Programm der Kette mit geändert werden mußte. Da man in Verbindung mit den

Programmketten auch Dateiketten für die Ablage der Ergebnisse und Zwischenergebnisse einsetzte, wurden leicht Datenbestände verwendet, die aufgrund von Änderungen in anderen Entwurfsphasen bereits veraltet waren. Das gleiche Problem ergibt sich, wenn mehrere Anwender interaktiv im Mehrbenutzerbetrieb an verschiedenen Phasen des gleichen Projektes arbeiten. Bei einem Dateienpool sind die Probleme der Datensicherung und der Gewährleistung einer hohen Datenqualität (Richtigkeit der Daten) kaum zu lösen. Darüberhinaus müssen Standardaufgaben der Datenverwaltung immer wieder neu bearbeitet werden, um sie an geänderte Bedingungen anzupassen /45/, /46/, /47/, /48/, /49/, /50/, /51/.

Von diesen Problemen ausgehend wurde ein neues Instrument entwickelt, das alle Standardaufgaben der Datenverwaltung zusammenfaßt: die Datenbank, eine Bezeichnung für ein Algorithmenpaket mit einer standardisierten Anwenderschnittstelle und einer Menge spezifischer Dateien, die System- und Problemdaten enthalten. Die Einzeldateien der herkömmlichen Vorgehensweise werden beim Einsatz einer zentralen Datenbank durch logische Dateien ersetzt. Sie umfassen jeweils den interessierenden Ausschnitt aus der Gesamtmenge der in physischen Dateien vorliegenden Daten, über deren Aufbau und Inhalt das einzelne Programm (oder der Anwender) nichts weiß.

Der Einsatz einer Datenbank anstelle einer Vielzahl von Dateien dient folgenden Zielen:

- Datenunabhängigkeit der Anwenderprogramme,
- hohe Konsistenz der Daten,
- geringe Redundanz,
- hohe Verfügbarkeit der Daten zu jedem Zeitpunkt und für jeden Anwender,
- Datensicherheit,
- Standardisierung von Datenbeständen.

2.2.1. Datenbankarchitektur

Die Struktur eines Datenbanksystems geht aus Abb.18 hervor. Ein Datenbanksystem besteht für den Anwender im wesentlichen aus folgenden Komponenten /46/, /51/:

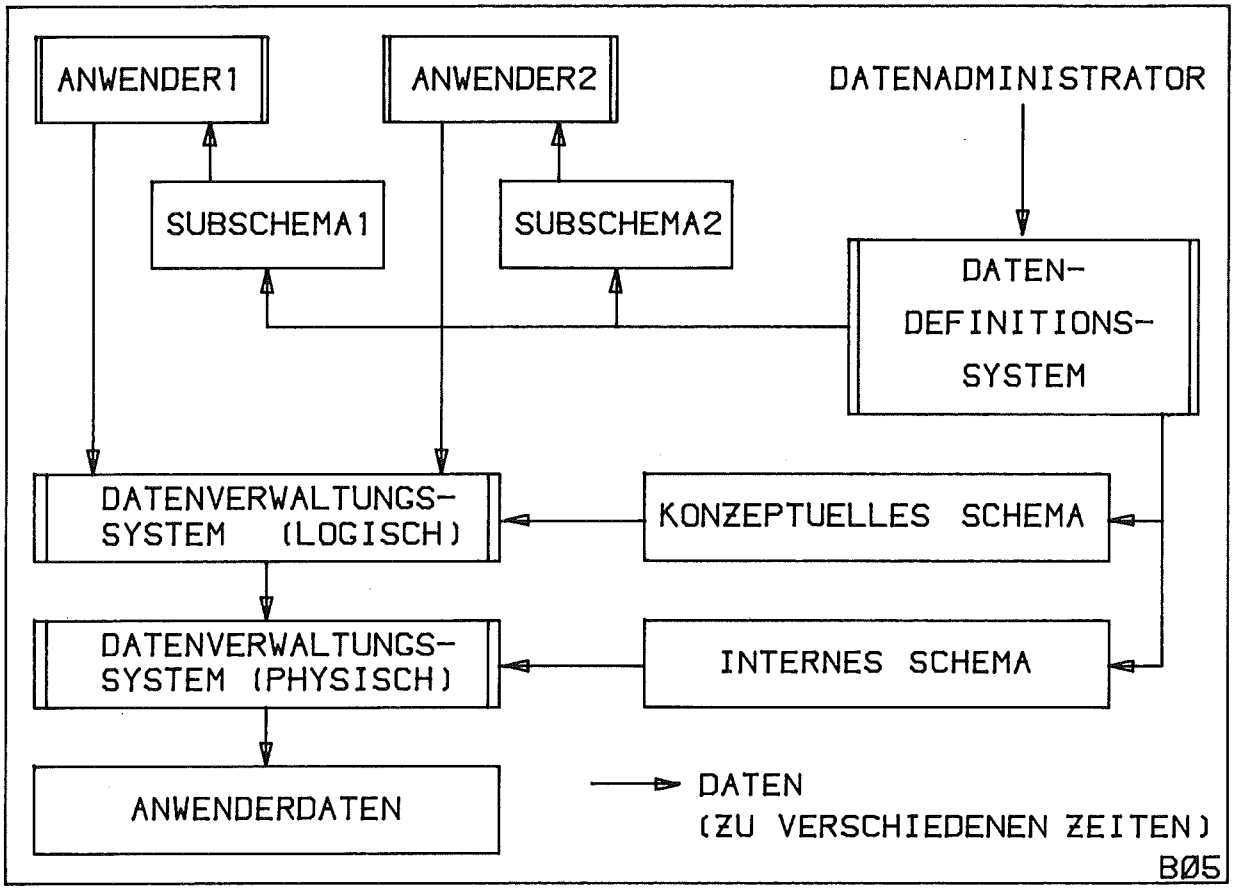


Abb.18: Struktur eines Datenbanksystems

(1) Datenmanagementsystem (DMS) : eine Menge von Operationen wie INSERT, DELETE, CHANGE, OPEN, CLOSE etc.

(2) Anwenderdaten : Dateien, in denen die Problemdaten in einer Form gespeichert sind, die dem Anwender nicht bekannt ist.

(3) Schema : eine Menge von Vorschriften zur problemspezifischen Beschränkung bzw. Modifikation der Operationen

(4) Datendefinitionssystem (DDS): System zur Festlegung der Schemata

Es werden zwei Operationsebenen unterschieden: die Datenbankadministration (Einrichten einer Datenbank und Erstellen der Schemata) und das Datenmanagement. Die Datenbankadministration dient gewissermaßen der Arbeitsvorbereitung und der Bearbeitung von Infrastrukturaufgaben. Das zentrale Schema (konzeptuelles Schema) enthält eine Beschreibung sämtlicher zu bearbeitender Daten eines Problemkreises. Diese Beschreibung umfaßt alle zulässigen Datentypen und Assoziationen, wobei je nach System ein spezifisches Datenmodell verwendet wird. Die Datenbeschreibung wird ergänzt durch die Festlegung von Zugriffsbeschränkungen und eventuelle explizite Konsistenzbedingungen.

Das interne Schema beschreibt die Art der Datenspeicherung auf den Dateien der Datenbank, es bestimmt entscheidend die Effektivität der Zugriffe, hat aber für den Anwender ansonsten keine Bedeutung. Da bei einer Anwendung im allgemeinen nur ein beschränkter Bereich der Gesamtdatenmenge unter spezifischen Aspekten bearbeitet wird, ermöglicht das Datenbanksystem mit Hilfe der Subschemata eine weitere Einschränkung bzw. Spezialisierung der Operationen. Diese Schema - Subschema - Abbildung führt zu der gewünschten Datenunabhängigkeit der Programme und dient dem Datenschutz. Die Freiheitsgrade des Anwenders (der Operationen) können also problemgerecht in zwei Ebenen eingeschränkt werden.

Das konzeptuelle Schema einer Datenbank ist ein komplettes Modell eines Ausschnitts aus der realen Welt, also beispielsweise das Modell einer Anlage in allen den Einzelheiten, die für den Planungsprozeß von Bedeutung sind. Es enthält alle Anlageninvarianten und Standard-Planungsrestriktionen, soweit sie formalisierbar und durch den Schema-Beschreibungsmechanismus darstellbar sind. Ein Subschema erfaßt die Anlage lediglich unter einem Teilaspekt, z.B. dem der Bauplanung. Ein Subschema maskiert aus der Gesamtmenge der Daten und Restriktionen einen Teil heraus, so daß der Umgang mit den Daten sicherer und leichter wird.

Eine wesentliche Entscheidung, die bei der Konzipierung eines Datenbanksystems getroffen werden muß, betrifft den Zeitpunkt, an dem Schemainformationen ausgewertet werden /51/:

(1) bei der Übersetzung der Programme,
(2) beim Laden der Programme,
(3) beim Eröffnen der Bank (OPEN),
(4) beim Zugriff auf die Daten.

Im ersten Fall müssen die Programme neu übersetzt werden, wenn sich das Schema ändert, die Zugriffseffektivität ist allerdings sehr groß, während im Fall(4) eine hohe Flexibilität erreicht wird (dynamische Datenunabhängigkeit). Schemaänderungen sind hier möglich, ohne ein Programm übersetzen und laden zu müssen, ein Vorteil, der im Entwurfsbereich mit möglicherweise wenig stabilen und vielen verschiedenen konzeptuellen Schemata (vielen Modellklassen) zu bedenken ist. In der Analysephase wird häufig auf größere Datenmengen zugegriffen, so daß dabei die Effektivität stärker im Vordergrund steht, weniger die Flexibilität. Die Compilation erschwert natürlich auch vor allem die Bearbeitung mehrerer Datenbanken (Modelltypen) nebeneinander, wie es für die Anlagenplanung typisch ist.

2.2.2. Datenmodelle

Die gültigen Informationen für eine Datenbank beschreibt der Datenbank-Administrator mit Hilfe einer Datendefinitionssprache (DDL), einer Komponente des Datendefinitionssystems (DDS). Das Ergebnis sind die Schemata (Abb.18). Die Art und Weise dieser Datenbeschreibung wird von dem spezifischen Datenmodell /52/ bestimmt, das eine ganz spezielle Vorstellung von der abzubildenden Realität ausdrückt. Die Struktur eines Datenmodells hängt von der Art der zu beschreibenden Informationen, ihrem Zweck und ihrer Verwendung ab.

Um ein Datenmodell zu charakterisieren, muß man die Systemdatentypen und die damit zulässigen Operationen darstellen. Im folgenden wollen wir aber davon ausgehen, daß die Grundoperationen der verschiedenen Systeme (Einfügen, Löschen, Ändern etc) sich ähneln, so daß nur die verschiedenen Systemdatentypen verglichen werden müssen. Wie aus Abb.19 hervorgeht, ist ein Schema gewissermaßen ein Exemplar (Instance) eines Datenmodells. Es besteht aus einer Menge von Datenobjekten, die alle den Systemdatentypen entsprechen. Der

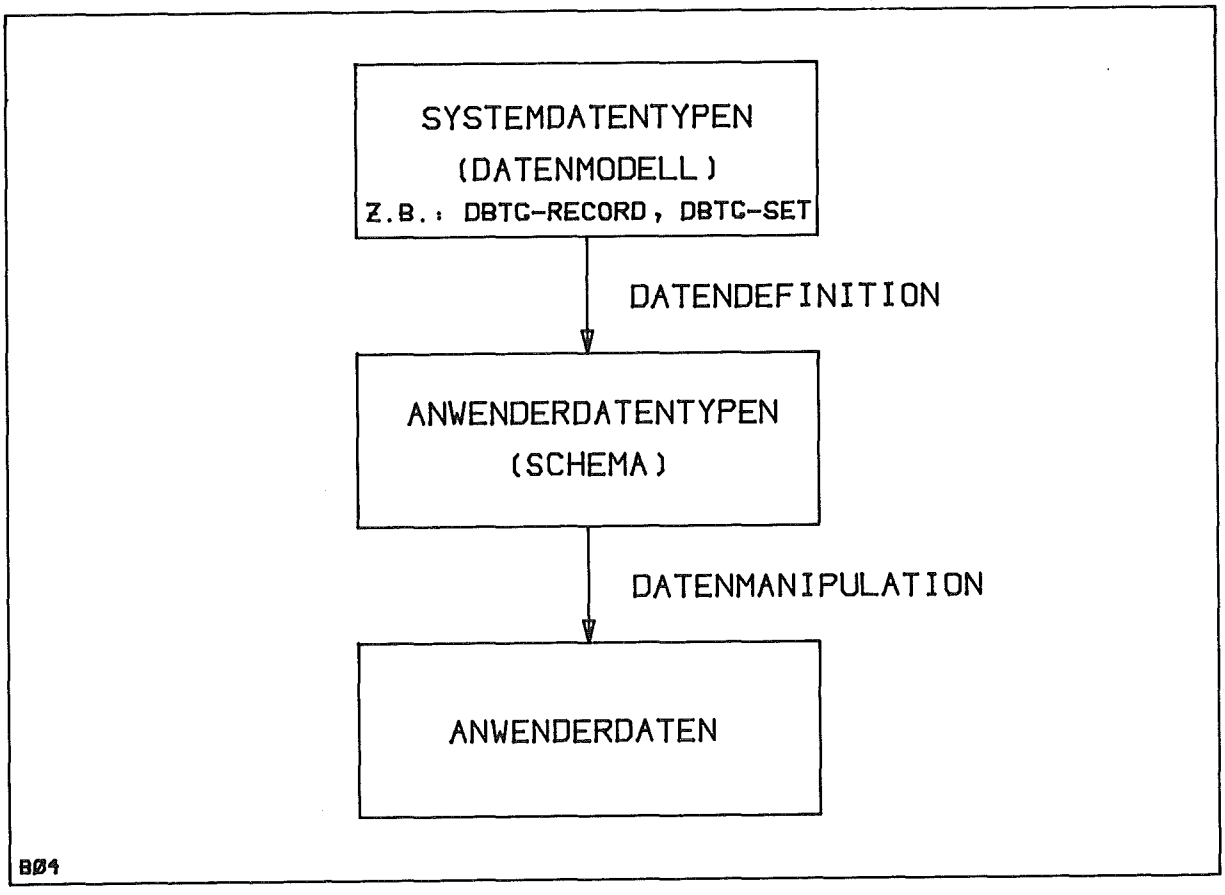


Abb.19: Datenbankarchitektur

analoge Prozeß wiederholt sich auf der Ebene der Datenmanipulation: Die Anwenderdaten (das Modell) sind Exemplare der Anwenderdatentypen (des Schemas).

Die verschiedenen Möglichkeiten der Datenbeschreibung werden exemplarisch an drei Datenmodellen vorgestellt.

(1) Das Netzwerkmodell (z.B. DBTG /49/, /53/) kennt als Systemdatentypen den RECORD und den SET,
(2) das hierarchische Datenmodell (z.B. IMS /46/) RECORDs und TREEs
(3) und das Relationenmodell (z.B. System-R /54/) die RELATION.

Man unterscheidet zwei große Klassen von Datenmodellen /52/, indem man sich die Daten als Graphen oder als Mengen vorstellt. Die Graphenmodelle kennen Objekte (Knoten) und Relationen auf den Objekten. Die Objekte selbst können strukturiert sein, sowohl den Objekten als auch den Relationen können Werte zugeordnet werden. Im Unterschied zu den Objekten sind zwischen Werten keine Relationen definiert. Die Mengenmodelle beschreiben dagegen alle Daten durch Mengen, die flach oder geschachtelt sind. Die Elemente der flachen Mengen dürfen selbst keine Mengen sein. Dieser grundsätzliche Unterschied in der Sicht der Daten bedingt neben der unterschiedlichen Datendarstellung auch verschiedenartige Auswerteoperationen. Bei Graphenmodellen wird navigierend auf die Daten zugegriffen, d.h. man geht schrittweise von Objekt zu Objekt durch die Datenstruktur, um einzelne Objekte zu finden. Beruht das Datenmodell dagegen auf Mengen, so erwartet man Mengen als Suchergebnis. Sie werden mit speziellen Mengenoperationen durch geeignete Einschränkungen des Gesamtdatenbestandes gewonnen.

Bei der Betrachtung der verschiedenen Datenmodelle soll deutlich werden, ob sie für die genannten CAD-Aufgaben brauchbar sind (/47/, /55/, /56/, /57/, /58/, /59/). Daher handelt es sich nicht um einen umfassenden Vergleich der Modelle. Es geht vielmehr darum, Grundaspekte plausibel darzustellen, um entscheiden zu können, ob eventuell ein spezielles Datenmodell für das funktionelle Modellieren zweckmäßiger ist als Datenmodelle allgemeiner Datenbanksysteme.

Netzwerk-Datenmodell

Dem Netzwerkmodell liegt eine Graphenstruktur mit strukturierten Knoten zugrunde. Die zulässigen Objekttypen (RECORDs) werden durch binäre Relationen (SETs) verknüpft. SETs beschreiben eine 1:n-Beziehung, d.h. einem Element vom Vorbereichstyp (OWNER-RECORD) können n Elemente des Nachbereichstyps

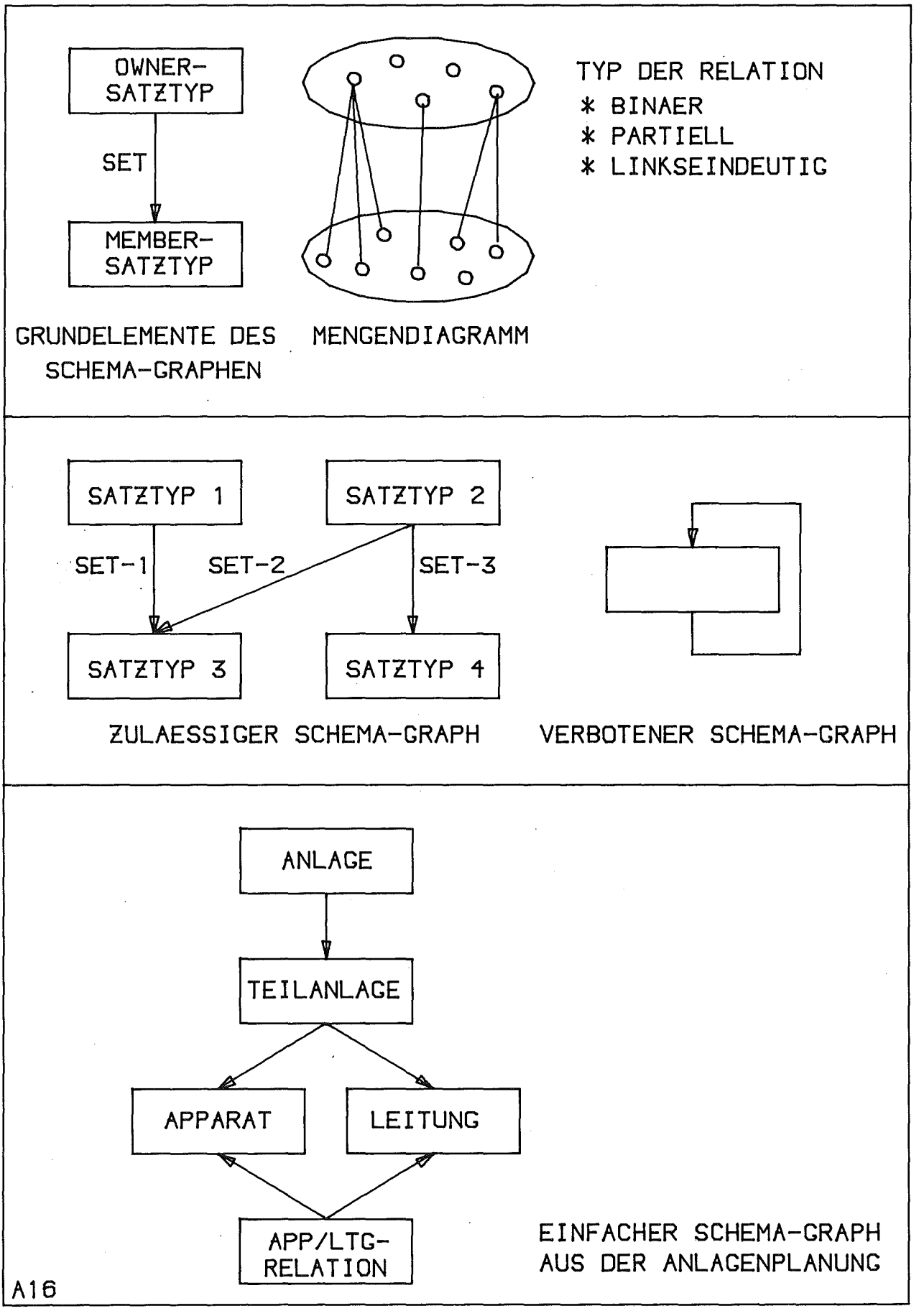


Abb.20: Netzwerk-Datenmodell mit Beispiel

(MEMBER-RECORD) zugeordnet werden. Ein Element eines Nachbereichtyps gehört zu genau einem Element des Vorbereichtyps. Es handelt sich also um Funktionen im mathematischen Sinne, und zwar beim DBTG-Vorschlag um partielle Funktionen (Abb.20). SETs haben keine Attribute.

Die Struktur der Objekttypen (RECORDS) wird durch eine Menge der bekannten Datentypen wie INTEGER, REAL und CHARACTER beschrieben.

Beim DBTG-Modell dürfen OWNER- und MEMBER-Recordtypen nicht gleich sein, so daß beispielsweise ein Stammbaum, bei dem alle Knotenebenen vom gleichen Satztyp sind (homogener Baum), nicht im Schema beschrieben werden kann. Mit den Systemdatentypen RECORD und SET läßt sich ein Netzwerk von Anwenderdatentypen (Schema) deklarieren, das die Vorstellung eines Entwurfsingenieurs von einer Anlage wiedergibt. Dieses Datenmodell entspricht der Denkweise des Anlagenplaners, der es gewöhnt ist, mit Graphen zu arbeiten. Das Beispiel in Abb.20 zeigt auch die Schwäche des DBTG-Modells, keine n:m-Abbildungen zuzulassen. Man kann sie durch einen Hilfs-Satztyp und zwei SETs, die diesen Hilfs-Satztyp als OWNER bzw. MEMBER enthalten, nachbilden.

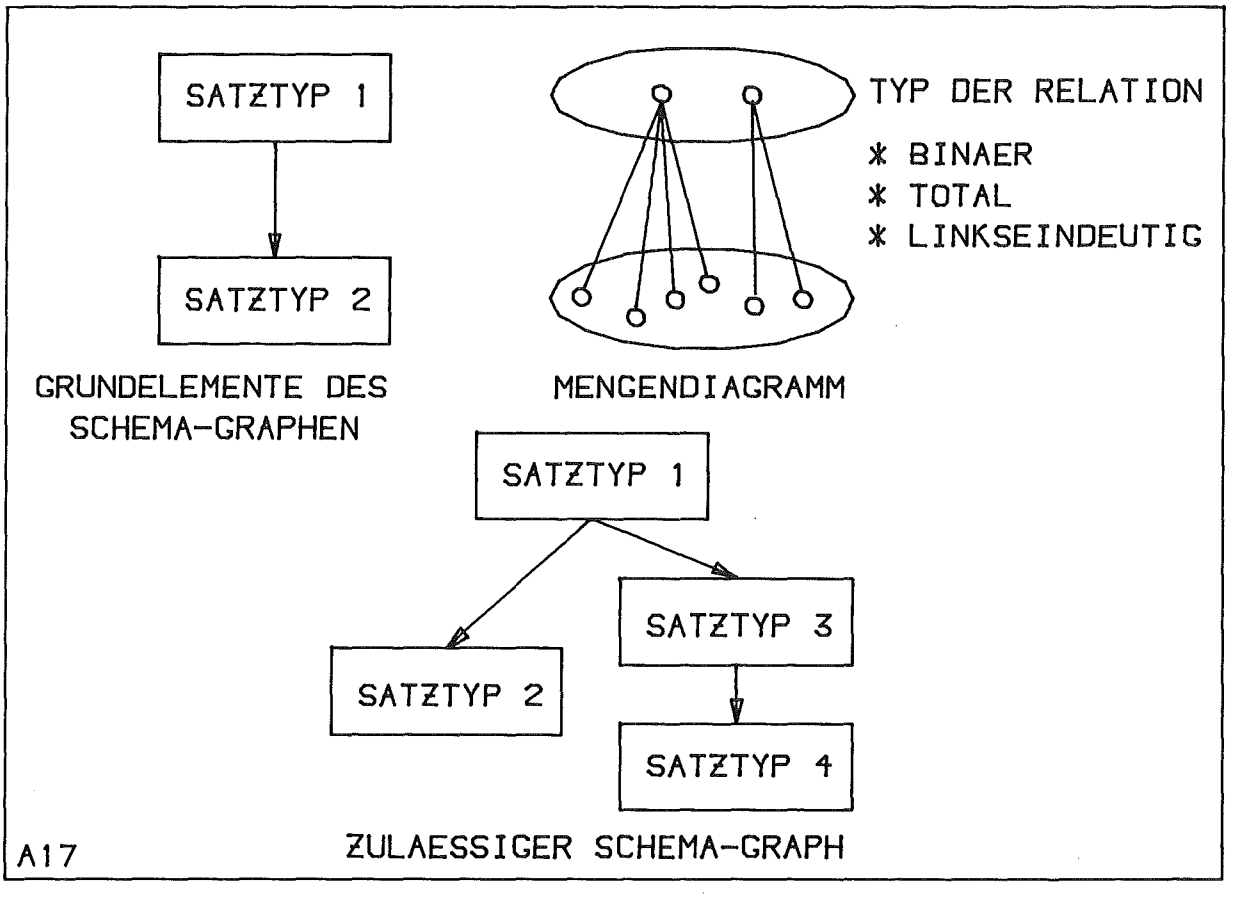


Abb.:21: Hierarchisches Datenmodell

Hierarchisches Datenmodell

Das hierarchische Datenmodell ist ein Sonderfall des Netzwerkmodells. Sein Schema-Graph ist eine Menge disjunkter Teilgraphen mit Baumstruktur. Die möglichen Relationen sind totale Funktionen (/61/,Abb.21).

Relationen-Modell

Hierbei geht man von der Vorstellung aus, daß sich alle Informationen durch Tabellen ausdrücken lassen. Einziger Systemdatentyp ist daher die Relation, die einen Tupeltyp beschreibt. Ein Tupel ist eine geordnete Menge von Daten der Basisdatentypen INTEGER, REAL etc. Die RELATION entspricht dem RECORD des Netzwerkmodells. Das System kennt keine expliziten Objekttyp-Verknüpfungen, sie werden erst beim Zugriff auf die Daten implizit formuliert (Abb.22).

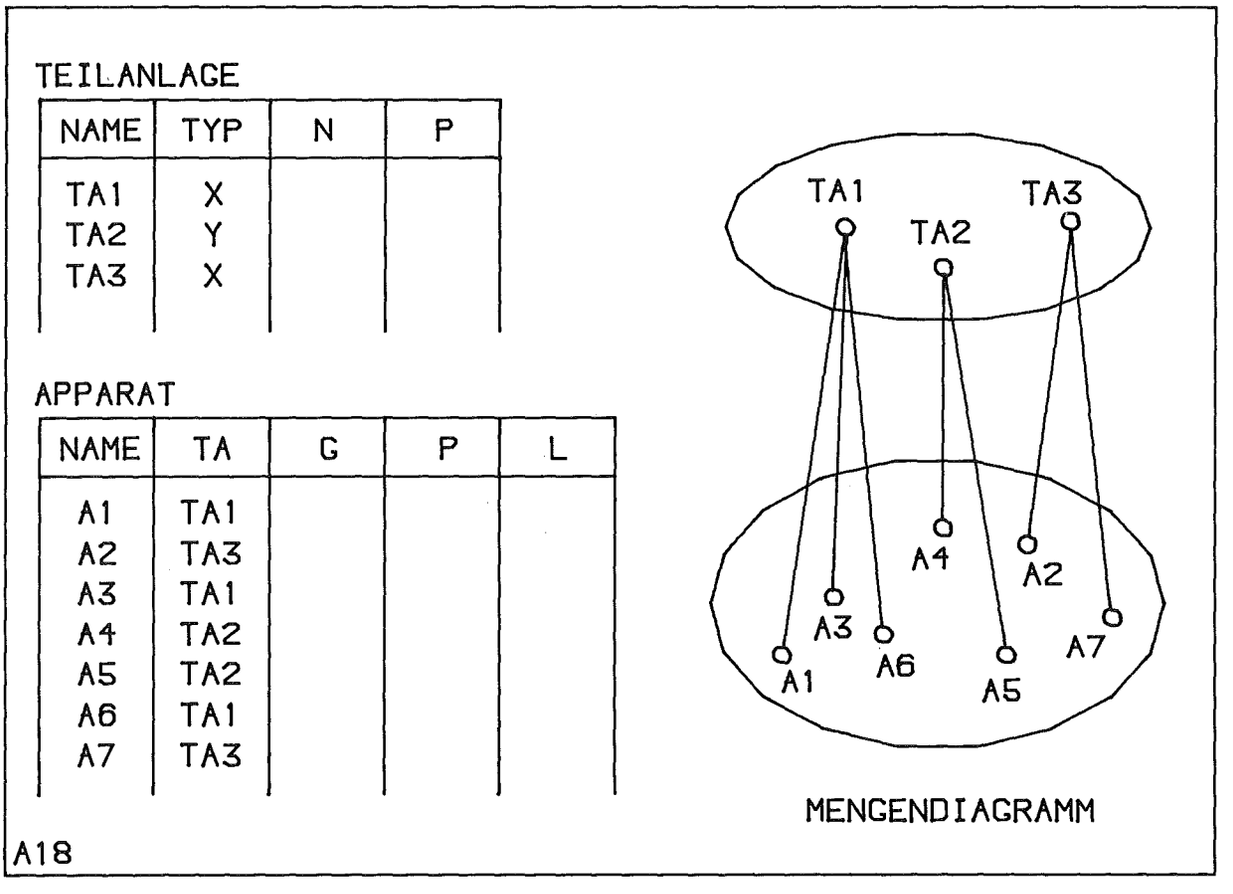


Abb.22: Relationenmodell

Das Relationenmodell wurde von Codd /60/ erstmals vorgeschlagen und ist seitdem theoretisch sehr gründlich untersucht worden, da es sich um ein einfaches, mathematisch gut erfaßbares Modell handelt.

Bei der Schemadefinition werden Relationentypen (Tabellentypen) definiert:

R <= <relationen-name>(<attribut-name_1>,...,<attribut-name_n>)

Grundelemente des Modells sind Attribute A_n, denen Wertebereiche V_i zugeordnet sind. Eine Relation ist eine Teilmenge R c V_1 X V_2 X ...X V_n über den Bereichen (Domänen) V_1,...,V_n. R heißt n-stellige Relation, ihr Grad ist n. Ein Element r = (a_1, ..., a_n) heißt Tupel der Relation mit a_i € V_i.

Eine Relation kann durch eine Tabelle dargestellt werden, deren Spaltenbezeichnungen die Attributnamen sind und deren Zeilen die Tupel entsprechen. Es gilt: Die Zeilen der Tabelle sind paarweise verschieden, die Reihenfolge der Zeilen ist bedeutungslos. Eine Relation ist in der sogenannten ersten Normalform, wenn die Wertebereiche V_i elementar sind, also vom Typ INTEGER, REAL etc. und nicht vom Relationentyp. Man spricht dann auch von einer flachen Datenstruktur.

Charakteristisch für das Relationenmodell sind die zugehörigen Auswerte- (Zugriffs-) Operationen, die es gestatten, die Gesamtmenge der Daten einzuschränken und dadurch implizit Objektrelationen zu formulieren /51/:

- Projektion: Streichen von Spalten.
- Join: Zusammensetzen von zwei Tabellen mit Spalten gleichen Wertebereichs.
- Auswahl: Streichen von Zeilen, für die eine bestimmte Bedingung nicht erfüllt ist.
- Vereinigung von Tabellen gleichen Typs.
- Durchschnitt von Tabellen gleichen Typs.
- Differenz von Mengen.

Dieses Modell entspricht nicht den Vorstellungen des Ingenieurs in der Phase des funktionellen Modellierens, in der er von einer systemtechnischen Betrachtung ausgeht und zwischen Objekten und Objektverknüpfungen explizit unterscheidet, so wie es auch in den Blockdiagrammen zum Ausdruck kommt.

Kritik

Die Datenbanktechnik bietet die eingangs geforderte Möglichkeit, Problemwissen (in Form des Schemas) zu beschreiben, um es in einem System für die Eingabeprüfung zu verwenden und einen Satz von Standardoperationen an ein Problem anzupassen. Allerdings sind die bekannten Datenmodelle der Datenbanken für allgemeine Anwendungen konzipiert. Sie sind in der Lage, theoretisch alle denkbaren Datenmengen zu modellieren, oft aber nur mit beträchtlichem Aufwand, da auf gewisse Besonderheiten einzelner Spezialfälle keine Rücksicht genommen werden konnte. Die Basisobjekte der Datendefinitionssprachen (DDL) sind elementar (Relation, Record, Set), verglichen mit denen der Blockdiagrammtechnik. Daher muß man viele Restriktionen, die durch das Basismodell nicht erfaßt werden, explizit ausdrücken, was aber nach dem Stand der Technik problematisch ist.

Die Auswahl eines bestimmten Datenmodells entspricht der Auswahl einer Programmiersprache: Die Datendefinitionssprachen der bekannten Datenbanken nehmen die Stelle der "general-purpose"-Programmiersprachen ein, erwünscht sind in vielen Fällen aber stärker problemorientierte Sprachen, um die Aufgabe schneller formulieren und effektiver lösen zu können. Es ist also ein problemorientiertes Datenmodell zu suchen, das die Formulierung problemorientierter Restriktionen erleichtert oder sie sogar weitgehend unnötig macht.

Die Modellsemantik wird einerseits systemimmanent durch die Bedeutung der Datenelemente (Systemdatentypen: Record, Set, Relation etc) implizit beschrieben und andererseits explizit durch Restriktionen formuliert. Beim Entwurf eines spezialisierten Datenbanksystems hat man nun die Möglichkeit, bereits viel Semantik in die Systemdatentypen (in die zugehörigen Verarbeitungsalgorithmen) zu stecken, wodurch die Übersichtlichkeit bei der Schemabeschreibung und die Effektivität der Bearbeitung erhöht werden.

Beispielsweise ist es für die Bearbeitung von Blockdiagrammstrukturen zweckmäßig, Datentypen vorzusehen, die bereits "Anschlüsse" enthalten. So kann das Basissystem erkennen, ob Eingänge mit anderen Eingängen verbunden wurden oder vielleicht Eingänge frei blieben. Derartige Restriktionen lassen sich mit einem allgemeinen Datenbanksystem nur explizit erfassen, wenn es überhaupt möglich ist.

Die bekannten Datenbank-Datenmodelle sind für die spezifische Anwendung des funktionellen Modellierens wenig geeignet, weil sie zu allgemein konzipiert sind und daher die Datendefinition zu umständlich wird. Das aber ist in einem Bereich, der sich durch viele wenig stabile Schemata auszeichnet und daher häufige Schemadefinitionen erfordert, nicht tragbar.

Ein wesentliches Merkmal des funktionellen Modellierens ist die große Lokalität der Datenmanipulation bei der Eingabe und beim Zugriff während des Modellierens: Die Arbeit beschränkt sich immer auf ein fest umrissenes Teilsystem, das der Ingenieur komplett vor Augen haben muß. Die einzelnen Teilsysteme sind, wie bereits erläutert, zu einer Hierarchie verknüpft. Auch dieser Teil des Datenmodells für das funktionellen Modellieren findet in den bekannten Datenmodellen keine Entsprechung, weder logisch noch im Hinblick auf die Zugriffseffektivität.

Die AREA des DBTG-Modells faßt Datensätze gewisser RECORD- und SET-Typen auf den Speichermedien physich zusammen, um die Zugriffseffektivität zu erhöhen. Diese Zuordnung kann aber nicht bei der Datenmanipulation beeinflußt werden. Sätze eines Typs liegen immer in der dem Typ zugeordneten AREA. Damit läßt sich aber beispielsweise eine Verteilung der Ventile einer Anlage auf verschiedene Teilanlagen (AREAs) nicht modellieren. Es ist eine typische Forderung von CAD-Systemen an Datenbanken, Exemplare eines Record-Typs über mehrere Cluster (Interessenbereiche) zu streuen.

Ein Datenzugriff in Form des "Durchblätterns" von Teilbereichen einer Gesamtdatenmenge entsprechend ihrer logischen Strukturierung in Teilsysteme wird nicht unterstützt. Es fehlt ein kontextabhängiges "Paging", wie es z.B. auch in /56/ gefordert wird.

Keines der genannten Datendefinitionssysteme bietet die Möglichkeit, die logische abstrakte Definition der Daten durch eine Beschreibung ihrer externen Darstellung und ihrer graphischen Identifikationsmöglichkeiten zu ergänzen, da sie als allgemein einsetzbare Standard-Datenmodelle konzipiert sind und die Datendefinition daher bewußt auf die Logik (von einigen Hinweisen für die physische Speicherung abgesehen) beschränkt wurde. Die externe Datendarstellung wurde Anwendungspaketen überlassen. Bei selbständigen Systemen, d.h. Systemen, die allein durch die Datendefinition an ein Problem angepaßt werden (keine Anwenderprogramme, sondern z.B. interaktiver Betrieb mit Standard-Befehlen), muß die Beschreibung der externen Darstellung aber in die Datendefinition integriert werden, wenn man sich nicht mit einer Standard-Darstellung der Datentypen begnügen will.

2.2.3. Datenbankintegrität

Mit der Einführung einer logischen Datenbeschreibung (Schema) liegt ein Mittel vor, mit dessen Hilfe die Qualität der Daten gewährleistet werden kann. Das Datenbank-Management sorgt dafür, daß nur Daten der im Schema angegebenen Typen in die Datenbank eingefügt und den Regeln entsprechend verknüpft werden dürfen.

Man spricht von der Integrität einer Datenbank, wenn es um die Korrektheit der Daten geht. Dabei unterscheidet man die operationale Integrität, bei der es um die Aufrechterhaltung der Datenqualität z.B. auch bei Mehrbenutzerbetrieb geht, von der logischen oder semantischen Integrität. Die semantische Integrität oder Konsistenz der Daten bedeutet, daß sie widerspruchsfrei ein Modell der abzubildenden Realität darstellen. Nur diese Problematik, die die Vermeidung falscher Benutzereingaben betrifft, wird im folgenden behandelt /51/, /61/.

Eine Reihe von Konsistenzbedingungen sind in den Datenmodellen bereits implizit durch die Typisierung enthalten, da das System dafür sorgt, daß nur Objekte vom vorgegebenen Typ aufgenommen werden. Darüberhinaus müssen beispielsweise

- beim Relationenmodell alle Tupel einer Relation verschieden sein,
- beim hierarchischen Modell muß ein Kind-Element auch einem Vater-Element zugeordnet sein und
- in einer DBTG-Datenbank sind alle Beziehungen vom Typ 1:n.

Hierbei handelt es sich um implizit durch die Art des jeweiligen Datenmodells festgelegte Konsistenzbedingungen.

Eine mögliche Klassifizierung der Konsistenzbedingungen gibt /51/:

(1) Konsistenzbedingungen für individuelle Objekte (z.B.: Farbe = rot, blau oder grün)
(2) Konsistenzbedingungen für Beziehungen zwischen individuellen Objekten (z.B.: wenn Rohrleitung X aus Cu, dann auch Leitung Y aus Cu)
(3) Konsistenzbedingungen für Mengen von Objekten. Dabei sind Operationen auf Mengen (Summen, Durchschnitt) erforderlich (z.B.: das Gewicht aller Apparate in Raum X darf Y t nicht überschreiten)
(4) Konsistenzbedingungen für Beziehungen zwischen Objektmengen (z.B.: die Menge der Pumpen ist immer eine Teilmenge der Maschinen)

In realisierten Datenbanksystemen sind allerdings nur beschränkte Möglichkeiten vorgesehen, Konsistenzbedingungen explizit anzugeben. Sie beziehen sich vor allem auf die Zulässigkeit von Duplikaten, auf die Angabe von Wertebereichen für Attribute und auf die zugehörige Verwendung von Datenformaten (CHECK-Klausel bei DBTG). Konsistenzbedingungen vom Typ (2) sind beim funktionellen Modellieren besonders wichtig und häufig, sie lassen sich

aber im DBTG-Modell nicht formulieren. Im DBTG-Entwurf ist die Möglichkeit vorgesehen, mit der ON-Klausel den Zeitpunkt anzugeben (ON STORE ...), zu dem ein Anwenderprogramm zur Konsistenzprüfung aufgerufen werden soll. Die Hauptlast einer umfangreicheren Konsistenzprüfung wird also dem Anwender überlassen. Die Technik, den Zeitpunkt der Konsistenzprüfung vom Anwender bestimmen zu lassen, ist im Entwurfsbereich sehr hilfreich. Auf diese Weise läßt sich die Forderung realisieren, für begrenzte Zeit inkonsistente Modelle zuzulassen, um damit das Konzept der nicht freigegebenen Entwürfe zu realisieren. In /56/ wird für den Entwurfsbereich eine dem entsprechende, "verzögerte" Konsistenzüberwachung gefordert.

Überträgt man diese abstrakten Überlegungen auf die Realität des funktionellen Modellierens beispielsweise von Anlagen, so stellt man fest, daß die Gesamtheit der hier vorliegenden Konsistenzbedingungen sehr groß ist. Sie umfaßt letztlich alle Komponenten-Dateien, Stoffdaten, Entwurfsregeln, Normen etc., die in umfangreichen Handbüchern enthalten sind (informative Daten /47/). Eine EDV-Unterstützung der Konsistenzüberwachung sollte daher in einem ersten Anlauf darin bestehen, diese Unterlagen in Form eines Informationssystems aufzubereiten, mit dessen Hilfe der Entwurfsingenieur die Datenkonsistenz selbst überprüft. Bei fortschreitender Entwicklung würde nach und nach die Grenze G in Abb.23 nach links verschoben und damit ein wachsender Anteil dieser Regeln in das Schema einer Entwurfsdatenbank übernommen /62/. Bei den sogenannten informativen Daten (Daten eines Informationssystems über Entwurfsmittel) handelt es sich um Daten, die den Schemainformationen entsprechen und sie ergänzen. Die Schemainformationen sind der beim augenblicklichen Stand der Entwicklung sehr kleine Teil des informativen Datenbestandes, der durch das Datenmanipulationssystem (DMS) berücksichtigt wird.

2.2.4. Datenzugriff

Zur Bearbeitung der Daten stehen verschiedene Datenmanipulationssprachen (DML) bereit. Man unterscheidet hierbei /45/, /51/ prozedurale und deskriptive Sprachen. Prozedurale Sprachen verwenden vor allem Sprachelemente der Kontrollogik (IF...THEN...ELSE...), sie gehen satz- oder tupelweise vor (one record or tupel at a time), während deskriptive Sprachen Mengen und Mengenoperationen als Sprachelemente kennen. Bei prozeduralen Sprachen steht der Suchprozess im Mittelpunkt, im Gegensatz zu den deskriptiven Sprachen, in denen das Suchergebnis beschrieben wird. Welche Sprache zweckmäßiger ist, hängt vom Anwendungsfall bzw. vom Anwender (seiner Qualifikation) ab.

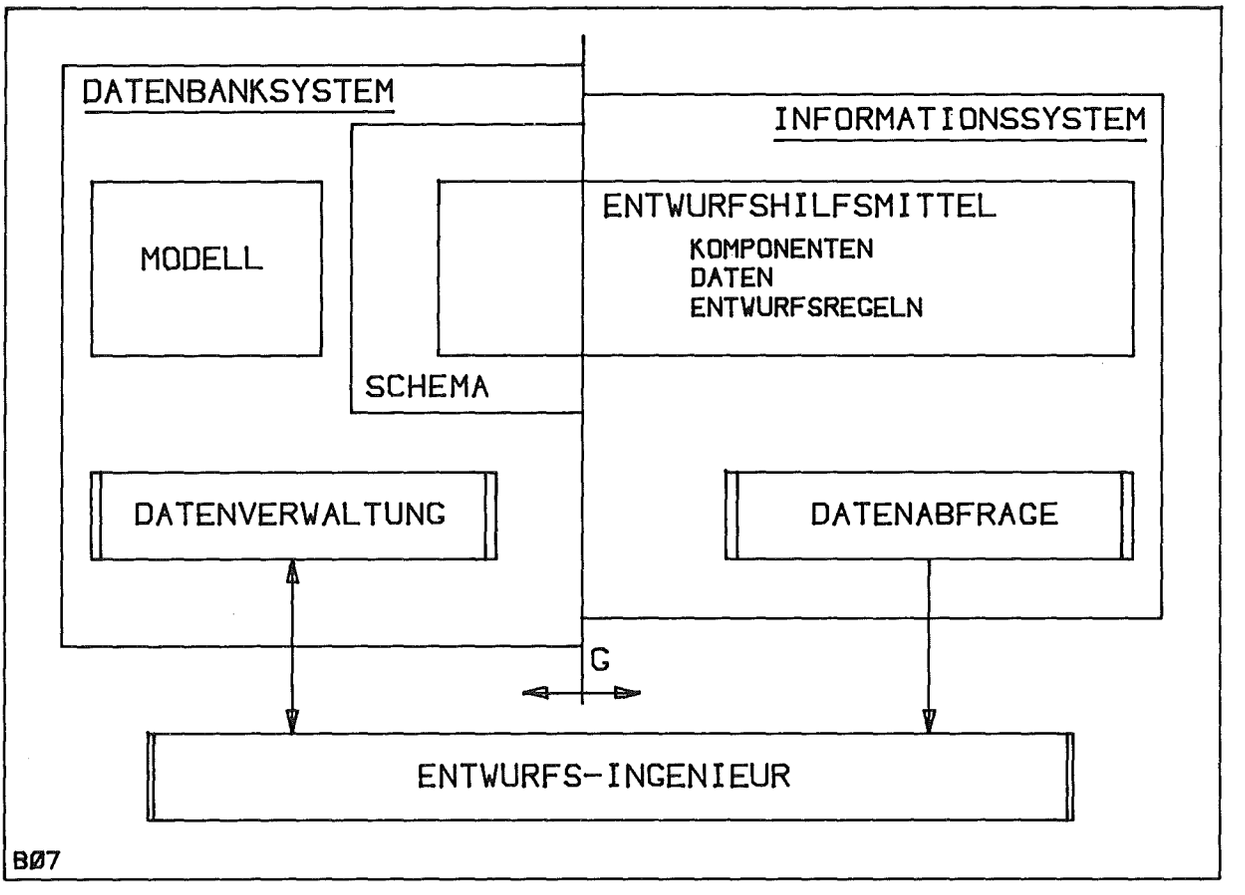


Abb.23: Datenbank- und Informationssystem

Die Sprachen werden in eine Trägersprache (z.B. PL/1) eingebettet oder als selbständige Sprache realisiert. Beispiele sind DL/1 des IMS /63/ als prozedurale Sprache mit PL/1 als Trägersprache und die deskriptive Sprache ALPHA /64/.

Neben dieser Charakterisierung der DML spielt in der interaktiven Anwendung einer Sprache die Art der Notation eine große Rolle: Bei der Arbeit am Bildschirm ist es angenehmer und anschaulicher, zweidimensional graphisch unterstützt zu arbeiten, statt mit einer linearen Notation, d.h. mit Zeichenketten. Ein Ansatz in dieser Richtung ist "QUERY BY EXAMPLE" /61/. Hier werden Relationen als Tabellen mit allen Attributbezeichnungen im Tabellenkopf dargestellt.

Eine graphische Unterstützung ist im Hinblick auf ein CAD-System für das funktionellen Modellieren sehr wichtig, da komplexe Strukturen nur so anschaulich und erfaßbar bearbeitet werden können. Eine einfache Tabellengraphik ist dazu aber nicht ausreichend, sie muß durch eine Blockdiagramm-Graphik ergänzt oder ersetzt werden.

Im Bereich des funktionellen Modellierens ist aber auch der Datenzugriff anders geartet als bei den bekannten DB-Systemen: Es geht bei der interaktiven Bearbeitung von Systemen (z.B. Anlagen) weniger um Fragestellungen der Art "Nenne mir alle Objekte mit dem Gewicht größer als 1 Tonne", sondern vielmehr um den Zugriff auf Teilsysteme, die man komplett mit allen oder einer Teilmenge ihrer Werte am Bildschirm sehen möchte. Die Auswahl der Teilsysteme erfolgt, indem man beispielsweise einzelnen Objektverbindungen folgt, die Teilsystemgrenzen überschreiten (und damit eventuell auch den Record-Typ wechselt), oder indem man Detaillierungs- oder Abstraktionsschritte nachvollzieht. Nur beim Auswerten des fertigen Modells durch Analysealgorithmen ist eine Sprachschnittstelle als Erweiterung einer Gastsprache (der Sprache, in der die Algorithmen programmiert werden) erwünscht.

2.3. Integrierte CAD-Systeme

Die Grundidee dieser Systemklasse ist, dem Anwender - Systemimplementierer, Systemanwender - Hilfsmittel bereitzustellen, die die üblichen Betriebssyteme gar nicht oder nicht in einer Form anbieten, die in dieser speziellen Aufgabenklasse erwünscht ist. Ein typischer Vertreter dieser CAD-Systeme ist REGENT /66/, /67/, /2/, auf das sich die folgenden Ausführungen beziehen. REGENT besteht aus einem Systemkern und Subsystemen (Anwendersystemen) zur Bearbeitung eines abgegrenzten Problembereiches. Dem Subsystemimplementierer stehen Hilfsmittel bereit

- zur Entwicklung problemorientierter Sprachen (POL) /68/, /69/,
- zum Modulmanagement,
- zum Datenmanagement und /70/
- zur Verwaltung von Fehlernachrichten.

Für den Subsystemanwender sind insbesondere die Fähigkeiten des Subsystem-Managements von Bedeutung, die eine wechselweise Nutzung verschiedener Subsysteme für eine Aufgabe ermöglichen ("CAD-Methodenbank"). Ein Subsystem ist gekennzeichnet durch

- eine problemorientierte Sprache zur Problemformulierung,
- eine Datenstruktur, die Probleminformationen enthält,
- eine Menge von Moduln, die die gestellte Aufgabe bearbeiten,
- eine Menge von Fehlernachrichten und eventuell
- eine Datenbasis.

Wichtiger Bestandteil des REGENT-Systems ist das Subsystem GIPSY /71/, /72/, das die Bearbeitung zwei- und dreidimensionaler Graphik erlaubt. Ein typischer Anwendungsfall von REGENT im Konstruktionsbereich ist beispielsweise die Variantenkonstruktion: Mit einer problemorientierten Sprache beschreibt man die Modifizierung eines Modells, für das verschiedene Berechnungsmoduln verwaltet werden und für das eine parametrisierte Darstellungsvorschrift vorliegt, die mit GIPSY formuliert wurde.

Typisch für die REGENT-Philosophie ist die algorithmische Beschreibung aller Probleme, insbesondere auch der Modellrestriktionen für ein Subsystem, was der eingangs genannten Forderung nach deskriptiver Beschreibung widerspricht, bei anderen Problemen aber auch von Vorteil ist.

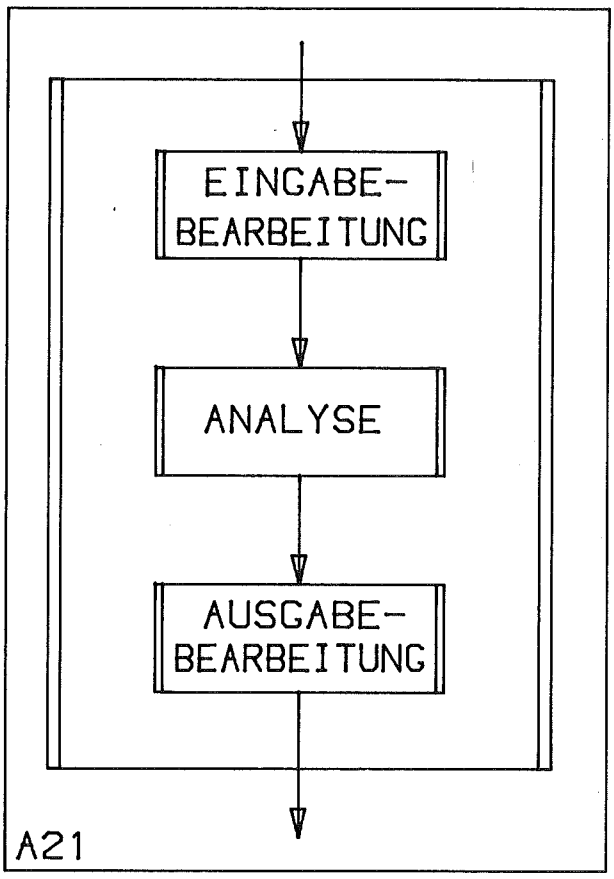


Abb.24: Mögliche Systemstruktur bei REGENT-Einsatz

Beim Einsatz integrierter CAD-Systeme entspräche die Systemstruktur beispielsweise der in Abb.24 gezeigten. Ein Eingabemodul, der die POL abarbeitet, bereitet die Daten für die nachfolgende Analyse auf und prüft sie auf Fehler. Ein Ausgabemodul stellt die Ergebnisse für die Beurteilung graphisch dar, wobei GIPSY verwendet würde. Neben der beschränkt geeigneten sprachlichen Modellierung ohne Interaktion, besteht ein weiterer Nachteil darin, daß bei Änderungen der Aufgabe (des Modelltyps) auch die Eingabemoduln und die Ausgabemoduln neu programmiert werden müssen, da sie die problemspezifischen Restriktionen der Bearbeitung enthalten.

Von diesen ganz speziellen Forderungen abgesehen, hat sich REGENT aber als hervorragendes Werkzeug zur schnellen Implementierung von Anwendungssystemen erwiesen.

2.4. Schlußfolgerungen

Diese Betrachtung der verschiedenen möglichen Hilfsmittel für den funktionalen Entwurf verdeutlicht, gemessen an den einleitend genannten Forderungen, ein Reihe von Unzulänglichkeiten und Schwächen der zur Verfügung stehenden Systeme.

Das führte zu dem Entschluß, auf der Basis eines derartigen Hilfsmittels ein neues System zu implementieren, das den Vorstellungen entspricht. Dazu werden bewährte Techniken und Konzepte der vorhandenen Systeme übernommen, erweitert und kombiniert. REGENT bietet sich als geeignete Implementierungsbasis an, da seine Subsystemtechnik einerseits die Implementierung und Verwaltung der Analysealgorithmen unterstützt, zum anderen aber auch die Implementierung von neuen CAD-System-Komponenten erleichtert. Dabei bildet vor allem die damit verbundene leichte Definition von problemorientierten Sprachen (POL) /70/ eine gute Basis, um eine Datenverwaltungskomponente mit problemorientierter Datendefinitions- und Datenmanipulationssprache (DDL,DML). zu implementieren.

Die Datenbankkomponente soll keine Datenbank im gebräuchlichen, umfassenden Sinne sein, sondern lediglich die Technik der Datendefinition und Datenmanipulation für die gestellte Aufgabe bereitstellen. Sie wird als physische temporäre Teildatenbank eines größeren, umfassenden Datenbanksystems betrachtet, die auf eine Teilaufgabe zugeschnitten ist und durch eine logische Teildatenbank (durch Subschema definiert) nicht effektiv realisiert werden kann.

Auf einer höheren, abstrakteren Ebene der Datenverwaltung, die sämtliche Daten eines technischen Objektes (einer Produktpalette, eines Betriebes) zusammenfaßt (Abb.2), wäre ein Einsatz eines allgemeinen Datenbanksystems eher denkbar, da dort das erforderliche Schema stabiler sein könnte als in Teilbereichen und die geschilderten Nachteile sich nicht mehr auswirkten. Eine derartige Datenbank würde die Modelleinzelheiten der Teilbereiche möglicherweise nicht mehr auflösen, sondern diese Teildaten in übergreifenden Blöcken mit Kennwerten verwalten.

3. Das System GRIMBI zum funktionellen Modellieren

****

GRIMBI - GRaphische Interaktive Manipulation von Blockdiagramm-Informationen - ist ein System, das die im ersten Abschnitt genannten Forderungen an ein CAD-Hilfsmittel für das funktionale Modellieren erfüllt. Es enhält als Kern eine Datenverwaltung, die mit Hilfe einiger geeigneter Datenbanktechniken die Modelle dieses Bereiches verwaltet. Als Datenmodell dient das der Blockdiagrammtechnik. Für die interaktive graphische Datenbearbeitung enthält GRIMBI eine Graphik-Komponente, die üblichen Darstellungsformen der Blockdiagrammtechnik entspricht /71/, /72/.

Implementierungsbasis ist das integrierte CAD-System REGENT (siehe Kap.2.3.), wodurch eine Integration interaktiver Techniken (als GRIMBI-Komponente) mit solchen der Stapelverarbeitung (typisch für REGENT) erreicht wird.

GRIMBI vereinigt in sich also Grundideen der Datenbanktechnik, der interaktiven graphischen Systeme und der integrierten CAD-Systeme vom Typ REGENT.

Es ergänzt das "CAD-Methodenbank-System" REGENT um die Methode des interaktiven graphisch gestützten funktionellen Modellierens.

Die grundlegende Architektur des GRIMBI-Systems zeigt Abb.25. In einem Prozeß der Arbeitsvorbereitung beschreibt der Anwender (oder der Modellklassen-Administrator) die Umgebung, in der er modellieren möchte, die Modellklasse (Schema). Dazu steht ihm eine Datendefinitionssprache (GRIMBI-DDL) zur Verfügung, die eine problemgerechte Modellklassenbeschreibung gestattet. Eine solche Modellklassenbeschreibung umfaßt die beim Modellieren innerhalb dieser Modellklasse gültigen Datentypen und Restriktionen. Beispiele für derartige Modellklassen sind: "Fehlerbaum", "R&I für Kernkraftwerke" etc.

Bei der eigentlichen Anwendung des GRIMBI-Systems, beim Modellieren, prüft der Entwurfsprozessor die Benutzerangaben auf Grund der Modellklassenrestriktionen und liefert sofort etwaige Einzelfehler.

Da die Modellklassenbeschreibung auch bei der Analyse herangezogen wird, kann eine weitgehende Datenunabhängigkeit der Analyseprogramme erzielt werden.

In eine GRIMBI-Datenbank können beliebig viele Modelle eigefügt werden, die der der Bank zugeordneten Modellklasse entsprechen.

Die Abb.26 zeigt in Ergänzung zu Abb.25 die DV-technische Realisierung der GRIMBI-Architektur.

Die Modellklassen-Definition erfolgt mit einem GRIMBI-DDL-Programm im Stapelbetrieb. Ergebnis eines Programmlaufes mit einem GRIMBI-DDL-Programm ist eine GRIMBI-Datenbank, die für die Modellklasse initialisiert ist. Eine solche Datenbank kann interaktiv bearbeitet werden, wobei die Modellelemente entsprechend der Modellklassenbeschreibung graphisch dargestellt werden.

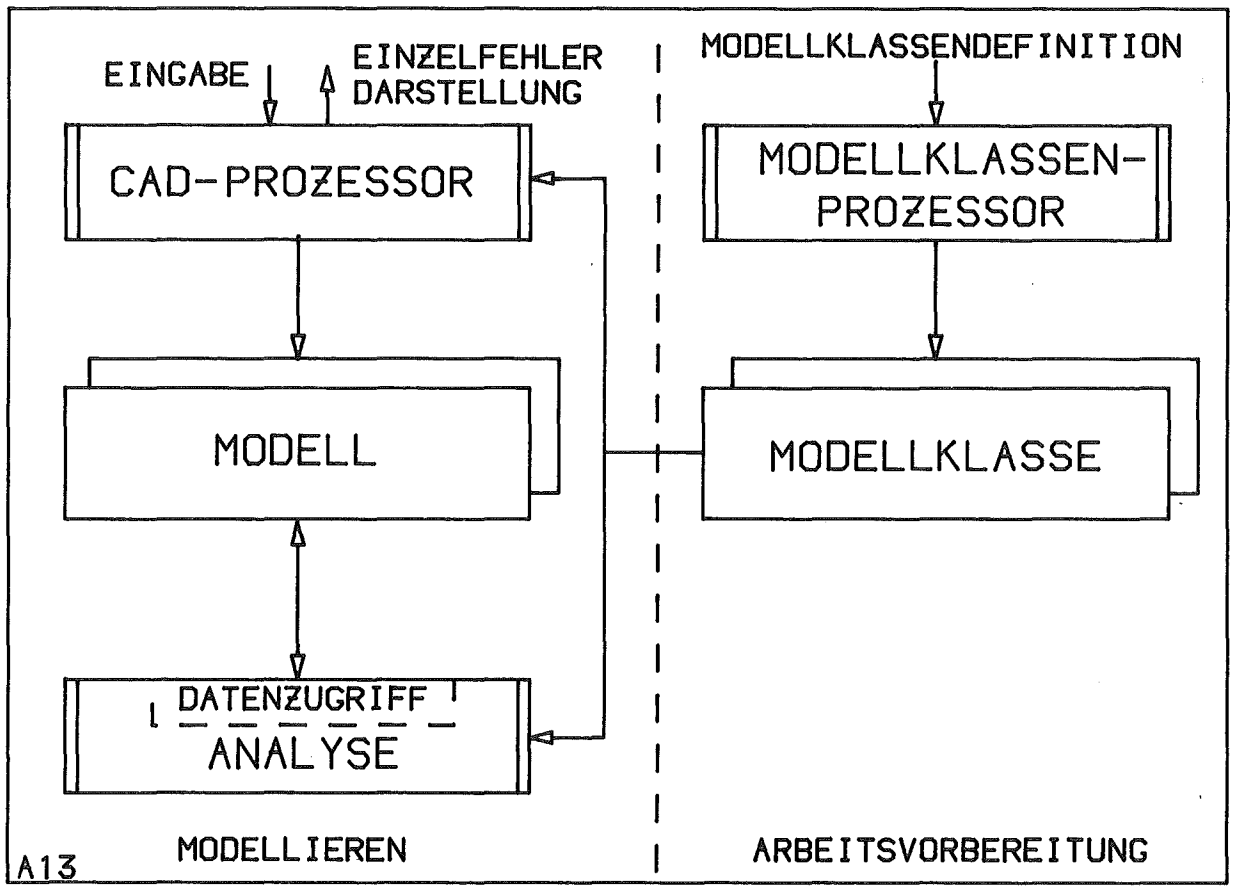


Abb.25: Logischer Aufbau des CAD-Systems GRIMBI

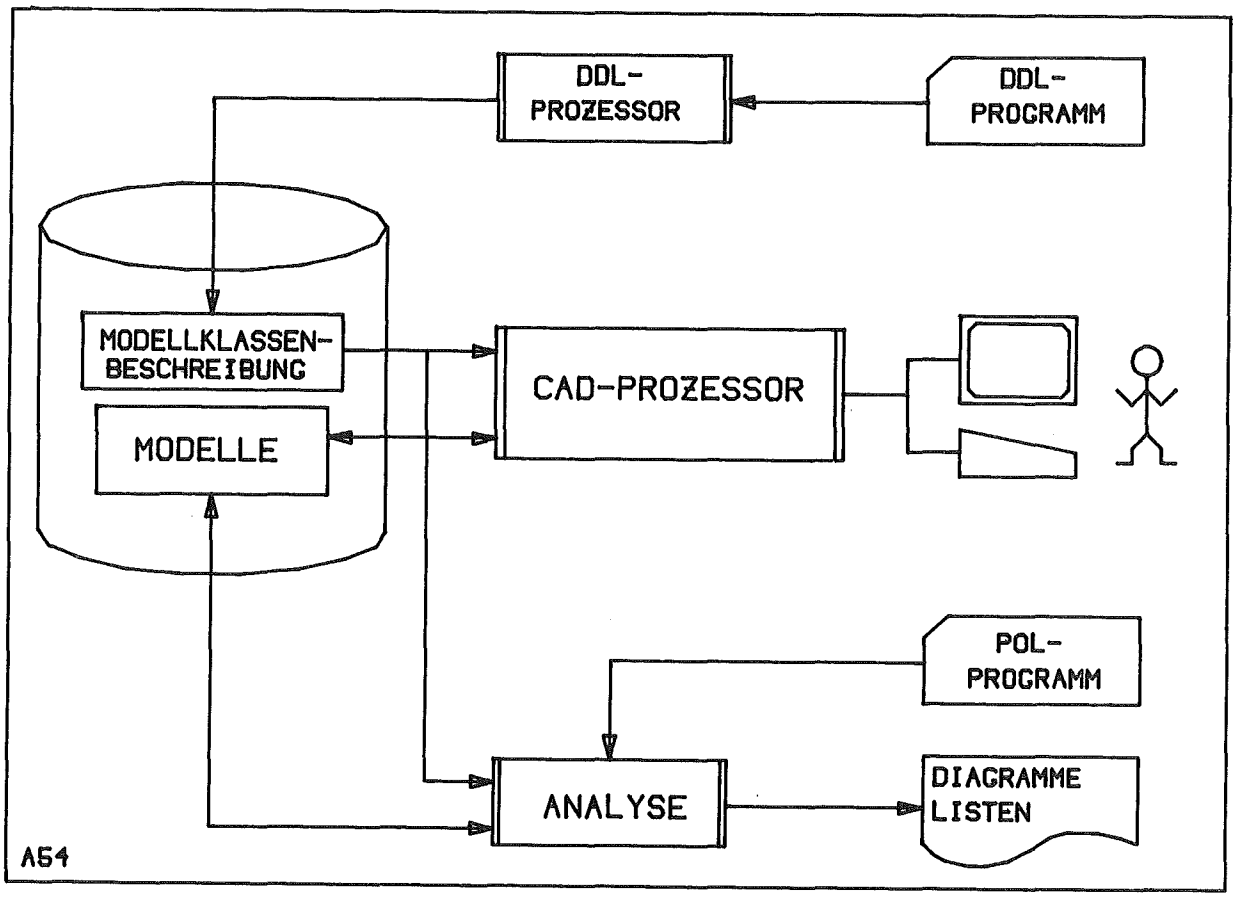


Abb.26: EDV-Systemarchitektur von GRIMBI

Daneben ist es auch möglich, mit Hilfe einer Datenmanipulationssprache (GRIMBI-DML, eine PL/1-Erweiterung wie die GRIMBI-DDL) im Stapelbetrieb zu modellieren. Derartig entstandene Modelle können später im interaktiven Betrieb durch die Darstellungsgraphik ergänzt werden, so daß dann ein weiteres interaktives Modellieren möglich ist.

Die Modellanalyse ist eine Aktivität im Stapelbetrieb, bei der mit Hilfe der GRIMBI-DML auf die Modelle zugegriffen wird, um Analysedaten zu extrahieren oder Analyseergebnisse in das Modell einzufügen.

GRIMBI unterscheidet zwischen dem eigentlichen

- Modellieren, bei dem eine logische Struktur unter Berücksichtigung der jeweiligen Modellrestriktionen erarbeitet und als EDV-Modell in einer Datenbank verwaltet wird und dem
- graphischen Editieren der externen Darstellung der Modelle im interaktiven Betrieb. Hierbei handelt es sich lediglich darum, Symbolpositionen oder Leitungsführungen graphisch zu verändern, ohne das logische Modell zu beeinflussen.

Dieser Trennung der Operationen entspricht auch eine Trennung der Daten in zwei Teildatenbasen, so daß der Editier-Prozeß vollständig auf einem intelligenten Satelliten implementiert werden konnte.

Wie Datenbanksysteme unterscheidet GRIMBI zwei Manipulationsebenen (Abb.27).

(1) Bei der Definition einer Modellklasse (Schemadefinition) werden alle Problem- (Anwender-) Datentypen mit Hilfe der Systemdatentypen deklariert, die die Datendefinitionssprache (GRIMBI-DDL) kennt. Problemdatentypen sind daher Exemplare (Instances) der Systemdatentypen. Um die interaktive, graphisch gestützte Bearbeitung zu ermöglichen, werden die Problemdatentypen nicht nur logisch beschrieben, sondern auch bezüglich ihrer externen graphischen Darstellung und Identifikation durch Zeigen.
(2) Für das Modellieren und die Analyse von Modellen steht ein fester Satz von Operationen bereit, die Exemplare von Problemdatentypen erzeugen und manipulieren. Dabei begrenzen die Problemdatentypen die Operationen, die für Systemdatentypen definiert sind, derart, daß gültige (konsistente) Modelle entstehen. In diesem Sinne stellt eine Modellklasse eine Bearbeitungsvorschrift im Rahmen bestimmter Aktionen dar.

Nach der detaillierten Beschreibung dieser Fähigkeiten wird die realisierte Systemarchitektur vorgestellt, die wesentlich von der spezifischen Hardware eines ersten Anwendungsfalles bestimmt ist, d.h. durch die Verwendung eines intelligenten graphischen Terminals unter einem Timesharing-Betriebssystem eines Großrechners. Durch eine solche Hardware-Konfiguration läßt sich die erforderliche Integration des interaktiven Timesharing-Betriebs und der Stapel-Verarbeitung gut realisieren.

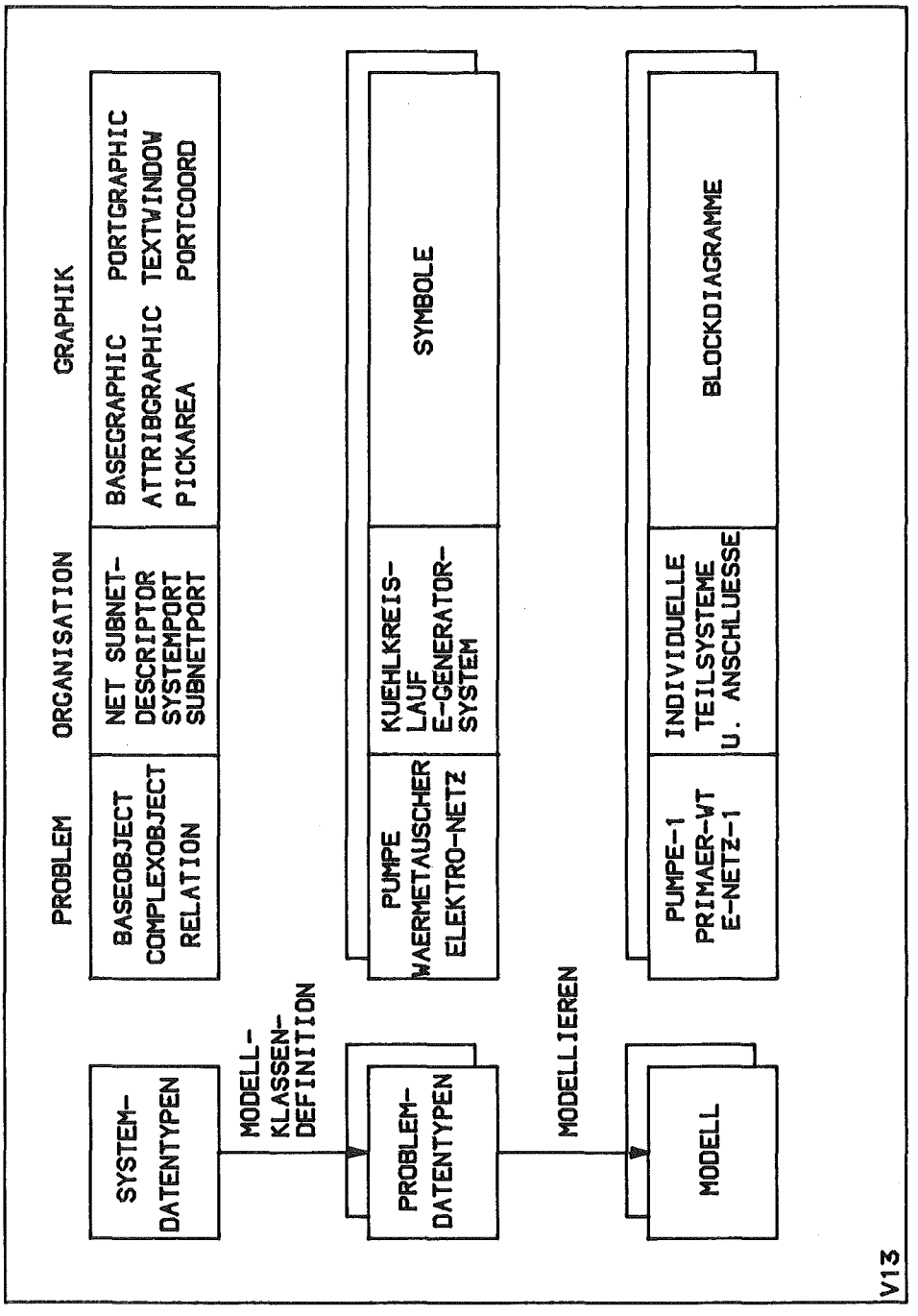


Abb.27: GRIMBI-Manipulationsebenen und Objekte

3.1. Definition von Modellklassen: Das GRIMBI-Datenmodell

Grundlage des GRIMBI-Entwurfs ist die Entscheidung, ein Datenverwaltungssystem mit interaktiv-graphisch gestütztem Datenzugriff und problemorientierter Datenmanipulationssprache als CAD-Hilfsmittel zu implementieren, das das bewährte Blockdiagramm-Datenmodell, d.h. ein Graphenmodell und kein Mengenmodell verwendet. Diese Entscheidung beruht auf der Erkenntnis der vorangegangenen Kapitel, daß nur durch ein problemangepaßtes Datenmodell die spezifischen Erfordernisse eines Aufgabenbereiches optimal erfaßt werden können. Der wesentliche Aspekt dabei war, möglichst viele typische Entwurfsrestriktionen durch das Datenmodell abzubilden, um weitgehende Konsistenz zwischen der Realität, dem Modell und den Annahmen der Analyseprogramme über die Eingabedaten zu erzielen, und den Entwurfsingenieur bei der Fehlersuche zu entlasten. Die Problemanpassung sollte dabei über die abstrakte logische Beschreibung der Daten hinausgehen und sich auch auf die zugehörige graphische Darstellung und Identifikation beziehen.

Die Zielsetzung für GRIMBI erforderte eine Analyse der Blockdiagrammtechnik, um herauszufinden, welche Arten von Informationen diese Technik erfaßt.

3.1.1. Übersicht über das Datenmodell

Das Blockdiagramm-Datenmodell ist ein Graphenmodell (Kapitel 2.2.3.), durch das Objekte (Dinge, Entities) und Relationen auf diesen Objekten explizit unterschieden und bearbeitet werden. Daneben existiert eine dritte Kategorie: die Werte. Sie unterscheiden sich von den Objekten, weil zwischen ihnen keine expliziten Relationen definiert werden /73/.

Die Einordnung von Daten in diese Kategorien ist willkürlich, sie wird allein durch den Zweck und die Gewohnheit in einem Kontext bestimmt.

Objekte werden unter zwei verschiedenen Aspekten betrachtet:

(1) als selbständiges Ding oder
(2) als Element eines Systems.

Im ersten Fall wird das Objekt durch Werte konkretisiert, im zweiten durch Anschlüsse (Valenzen). Sie beschreiben, bezogen auf den aktuellen Kontext, die Fähigkeit eines Objektes, sich in eine Struktur einzuordnen.

Unter einem System versteht man eine Menge von Objekten, die miteinander in Beziehung stehen. Diese Beziehungen werden im Blockdiagramm-Datenmodell durch zweistellige Relationen beschrieben.

$$S = (V , R) \quad \text{mit} \quad V = \{ v_1 , \ldots , v_n \} \quad \text{Objektmenge}$$
$$R = \{ R_1 , \ldots , R_m \} \quad \text{Relationen-Menge}$$
$$R_i \subset V \times V \quad \text{Relation}$$

Formal ist ein System ein Paar aus der Objektmenge und einer Menge von zweistelligen Relationen.

Sowohl Objekten als auch Relationen können Werte zugeordnet werden.

$$a : V \rightarrow W$$
$$r : R \rightarrow W \qquad \text{mit W als Wertemenge}$$

Ein Objekt wird durch ein Wertetupel

$$v = (w_1 , \ldots , w_n)$$

beschrieben, was einer Darstellung des Objektes in einem Blockdiagramm durch angefügte Texte und modifizierte Symbolgraphik entspricht. Relationen werden in dieser abstrakten Darstellung wie Objekte behandelt, deren graphische Darstellung in einem Blockdiagramm der Legende für die Verbindungslinien entspricht (Abb.28). Die 2-Tupel der Relationen repräsentieren die Objektverbindungen, auch sie können durch Werte charakterisiert werden, was in der Graphik durch Texte an den Verbindungslinien ausgedrückt wird. Die Textur der Verbindungslinien charakterisiert die Zugehörigkeit der Verbindung (des Tupels) zu einer Relation. Die Existenz der Relationen ist unabhängig von der Existenz irgendwelcher Objekte, so wie die Existenz der Objekte selbst. Tupel dagegen - als Elemente einer Relation - können nur existieren, wenn die referierten Objekte vorhanden sind.

Ein zentrales Konzept der Blockdiagrammtechnik ist das der Anschlüsse. Sie werden als Subobjekte aufgefaßt und fassen die Informationen zusammen, die für die Verbindungen von Bedeutung sind. Anschlüsse sind entweder Eingänge, Ausgänge oder ungerichtet. Sie bestimmen den Ein- bzw. Ausgangsgrad der Knoten, d.h. durch sie wird festgelegt, wie viele Verbindungen zu einem Objekt führen oder von einem Objekt wegführen können.

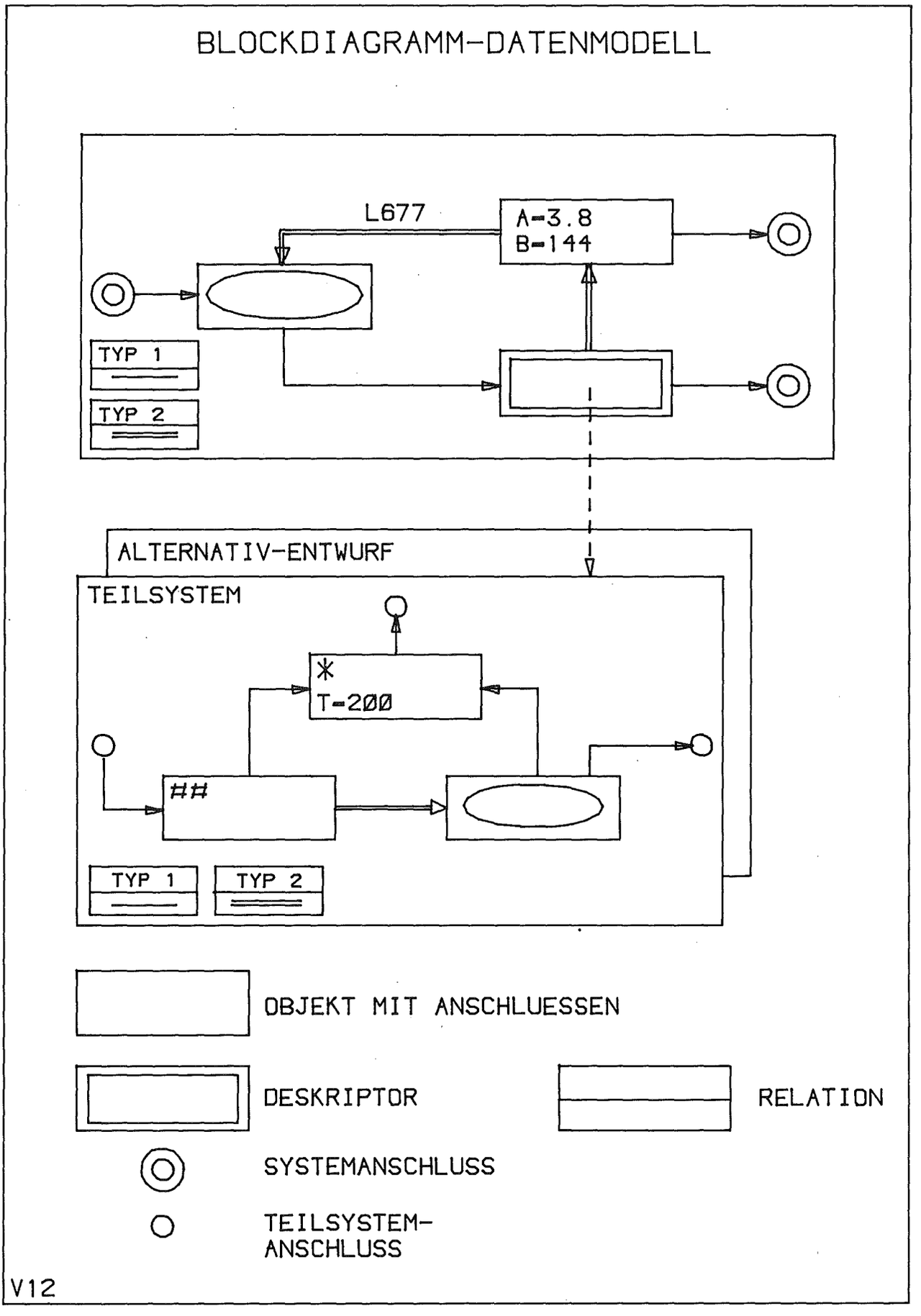


Abb.28: Das Blockdiagramm-Datenmodell

Es gibt aber noch einen dritten Aspekt, ein Objekt zu betrachten, nämlich den, ein Objekt selbst

(3) als System von elementaren Objekten

aufzufassen: $V = (V' , R')$.

Das bedeutet, ein Objekt zu detaillieren, auf eine niedere Abstraktionsstufe überzugehen. Der Block (das Objekt) wird selbst wieder durch ein Blockdiagramm dargestellt: ein Teil-(Sub-)Netz. Objekte, die so bearbeitet werden und mit einem Teilnetz korrespondieren, nennen wir Deskriptoren.

Diese Korrespondenz kann man formal so ausdrücken:

$$(w\_1,...,w\_n) \Longleftrightarrow (V' , R')$$

Ein Deskriptor repräsentiert ein Teilnetz auf einer höheren Abstraktionsstufe durch Werte.

Eine spezielle Art von Anschlüssen sind die Quellen und Senken eines Teilnetzes. Sie werden als selbständige Objekte behandelt, da sie die Schnittstellen des Teilnetzes darstellen. Sie korrespondieren mit den Anschlüssen des zugehörigen Deskriptors. Die Schnittstelle des Gesamtsystems wird ebenfalls durch derartige Anschlüsse erfaßt.

Die bisher gewählte Darstellung des Datenmodells durch den bekannten mathematischen Formalismus hatte den Zweck, die Klassifizierung des Datenmodells als Graphenmodell zu verdeutlichen. Für die Erläuterung der Einzelheiten eignet sich aber eine BNF-Form der Darstellung besser, sie wird daher im folgenden ausschließlich verwendet. Eine vollständige Syntax (mit Produktionen bis zu Konstanten) der GRIMBI-DDL befindet sich im Anhang 1, wo auch die Syntaxnotation erläutert ist.

In den Text wurde im Interesse einer übersichtlichen, kompakten Darstellung nur ein Syntaxauszug aufgenommen. Die Bedeutung der Variablen geht weitgehend aus ihrer Bezeichnung hervor.

Ergebnis dieser Betrachtung ist die Unterscheidung zweier Objektgruppen:

(1) Basisobjekte (mit Anschlüssen) und Relationen beschreiben die Problemstruktur und
(2) Netze/Teilnetze mit zugeordneten Deskriptoren und mit Netz/Teilnetzanschlüssen beschreiben die organisatorische Gliederung eines Gesamtsystems.

Diese Objektklassen haben in der Blockdiagrammtechnik ihre spezifische Bedeutung. Alle Objekte eines konkreten Modells lassen sich in diese Klassen einordnen, so daß sie von den Manipulationsoperationen entsprechend behandelt werden. Diese Klassen werden im folgenden Systemklassen genannt, die durch Systemdatentypen (Systemobjekttypen) repräsentiert werden.

Systemdatentypen in diesem Sinne beim Relationenmodell sind die Relationentypen, beim DBTG-Modell sind es RECORD und SET. Einen Vergleich des Blockdiagramm-Datenmodells und des DBTG-Modells zeigt Abb.29. Das Blockdiagramm-Datenmodell ist also differenzierter, wodurch natürlich die Systemunterstützung besser sein kann, da dem System mehr Wissen über den Modelltyp mitgeteilt wird.

3.1.2. Typisierung der Problemdaten eines Blockdiagramms

Um das CAD-System zu befähigen, Eingaben zu prüfen und den Anwender bei seinen Manipulationen in logischer und graphischer Hinsicht problemgerecht zu unterstützen, muß dem System Wissen über die Arbeit, also über Art und Darstellung der zulässigen Modelle (Blockdiagramme), mitgeteilt werden. Das geschieht bei GRIMBI wie bei Datenbanken durch eine sogenannte Datendefinition, deren Ergebnis ein Schema ist: Eine Modellklassenbeschreibung durch eine Datenmenge, nicht durch Algorithmen. Für diese Datendefinition abstrahiert man die interessierenden Anwenderobjekte durch "Generalisierung", indem man Objekte mit gleicher Bedeutung bezüglich des Problems zusammenfaßt und ihre Gemeinsamkeiten durch einen Objekttyp (Anwender-Objekttyp, Problemdatentyp) darstellt. Jeder dieser Objekttypen wird in eine der vorgenannten Systemklassen eingeordnet, um dem CAD-System seine Bedeutung im Kontext der Datenmanipulation zu beschreiben.

Objekte gehören zu einem Objekttyp, wenn sie durch die gleiche Anzahl von Werten beschrieben werden und alle korrespondierenden Werte die gleiche Bedeutung haben. Gleiche Bedeutung heißt, daß die gleichen Operationen auf die Werte angewandt werden können. Diese Abstraktion erfolgt formal dadurch, daß man nicht durch eine Wertegruppe ein Objekt, sondern durch eine Gruppe von Attributen einen Objekttyp beschreibt. Die Datenstrukturen sind durch die Systemtypen und damit durch die diesen zugeordneten Operationen definiert: Man spricht bei einer solchen Vorgehensweise auch von abstrakten Datentypen /74/, /75/. Beim Konkretisieren, beim Übergang von einem Typ zu einem Objekt, weist man jedem Attribut einen Wert zu.

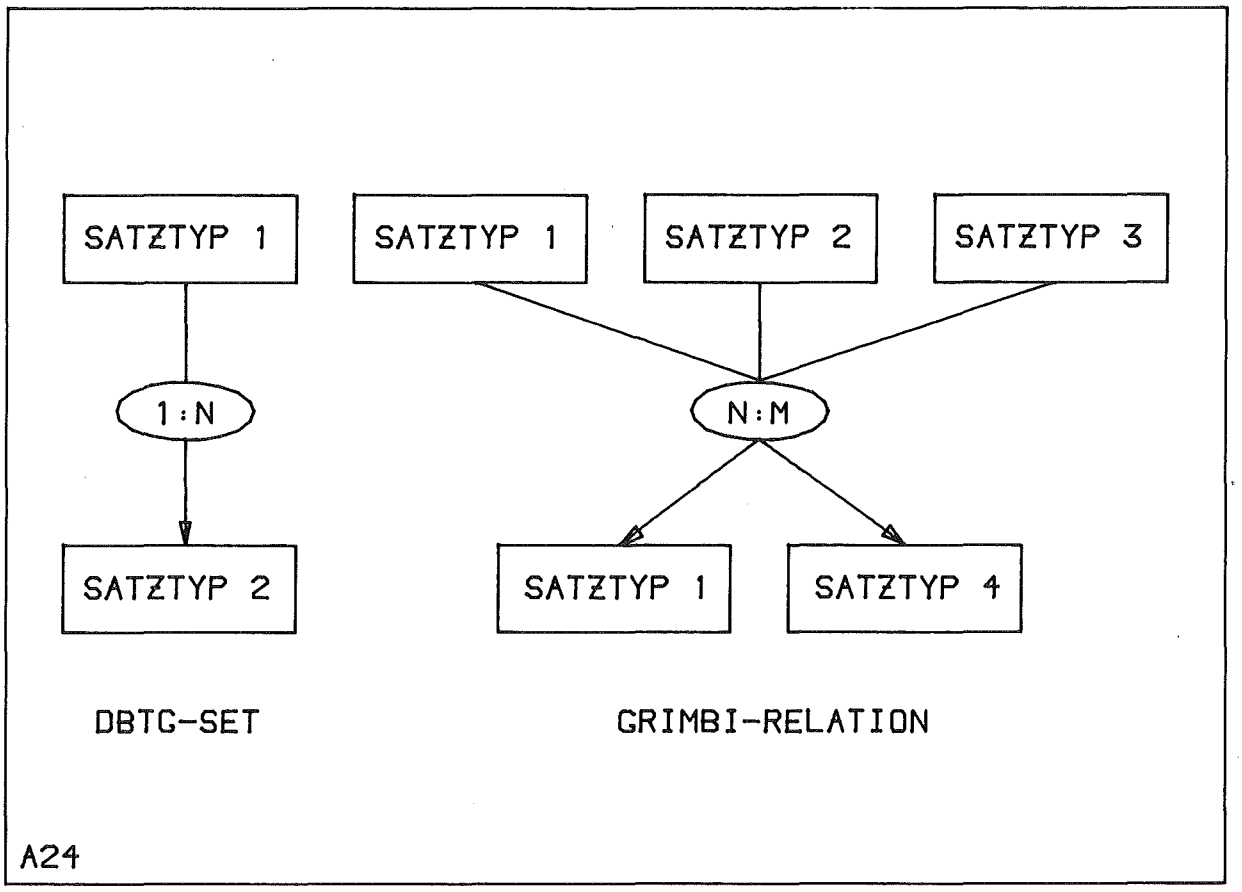


Abb.29: Blockdiagramm-Datenmodell und DBTG-Modell

Attribute

Attribute stehen stellvertretend für eine Gruppe von Werten. Sie sind bei GRIMBI kennzeichnet durch

- die Länge eines (eventuellen) Attributfeldes,
- einen Datentyp,
- eine Klassifizierung des Attributes für die Auswertung und Darstellung,
- die Elemente der Wertemenge,
- einen Standardwert und
- eine Angabe über die Notwendigkeit einer Wertzuweisung bei der Erzeugung eines Objektes, das dieses Attribut besitzt.

Ein Attribut wird einem Objekttyp durch die ATTRIBUTE-Klausel zugeordnet, die in der GRIMBI-Datendefinitions-Sprache folgendermaßen aussieht:

Syntax

```
<attrib>      ::= <attrname> ATTRIBUTE <datatype>
                                       {DIMENSION(<dim>)}
                                       {VALUESET(<valueset>)}
                                       {DEFAULT(<default>)}
                                       {CLASS(<class>)}
                                       {REQUIRED}

<datatype>    ::=  BIN FIXED(15)   | BIN FIXED(31)
                 | BIN FLOAT(21)   | BIN FLOAT(53)
                 | CHAR(<n>)       | BIT(<n>)

<valueset>    ::= (<from> TO <to>) | (<v_1>,...,<v_n>)
```

Beispiel

```
FARBE   ATTRIBUTE CHAR(8) VALUESET('ROT','BLAU','GRUEN')
                  DEFAULT('ROT') CLASS(1)

GEWICHT ATTRIBUTE BIN FLOAT(21) VALUESET(1. TO 100.) REQUIRED
```

Das Attribut erhält einen Namen, mit dem es identifiziert wird. Die zulässige Wertemenge wird entweder durch eine Werteliste oder unter Annahme einer Standardmenge durch Bereichsgrenzen angegeben, was den Möglichkeiten von PASCAL /76/ entspricht. Wenn die Option REQUIRED angegeben wird, hat der DEFAULT-Wert die Bedeutung eines unbestimmten Wertes, sonst wird er verwendet, wenn beim INSERT kein Wert für das Attribut genannt wird (siehe Kapitel 3.2.). Beim CLOSE SUBNET fordert das System Werte der REQUIRED-Attribute an, falls sie nicht beim INSERT OBJECT oder durch CHANGE eingegeben wurden. Attribute können ein Wertefeld erfassen, dessen Länge mit der DIM-Klausel angegeben wird. Die CLASS-Klausel klassifiziert die Attribute bezüglich der Auswertung und ihrer Darstellung (Kapitel 3.1.4.).

Ebenso wie die Problemdatentypen werden auch die Systemdatentypen durch Attribute beschrieben, die allerdings vom System fest vorgegeben sind. Sie heißen im folgenden Systemattribute. Die Wertzuweisung erfolgt bei der Schemadefinition.

Systemattribute

Syntax

```
<system-attribs> ::= {STEXT(<text>)}
                     {SUBTYPE(<problemtype-list>)}
                     {SUPERTYPE(<problemtype-list>)}
```

Die folgende Gruppe von Systemattributen charakterisiert alle Systemtypen.

STEXT

Werte dieses Attributes beschreiben verbal die Bedeutung des zugehörigen Objektes, beispielsweise für Auskünfte über Objekttypen beim Modellieren (HELP-Kommando). Ein Objekttyp darf durch einen Text mit der maximalen Länge 256 erläutert werden.

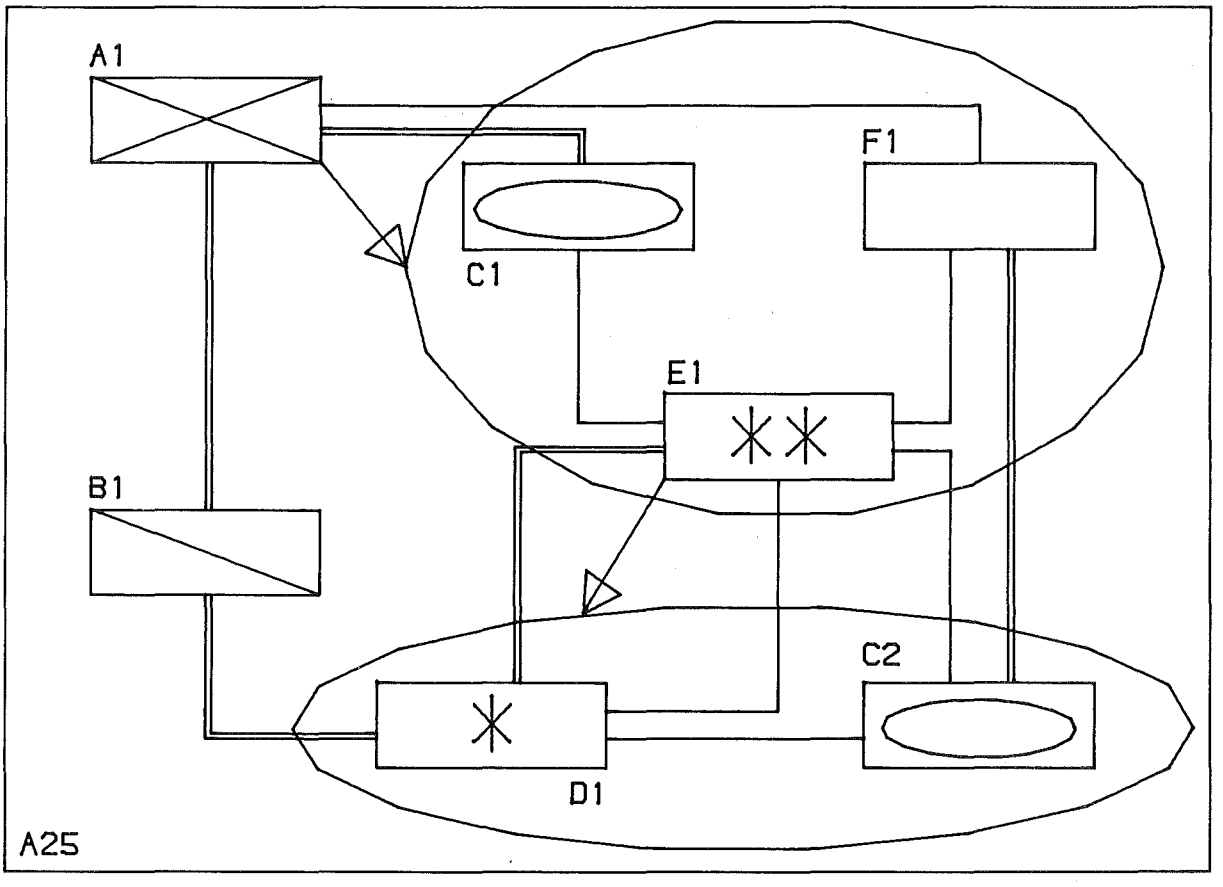


Abb.30: SUBTYPE-SUPERTYPE-Klauseln

SUBTYPE,SUPERTYPE

Außer den Anwenderrelationen zwischen den Objekten ist implizit eine Systemrelation definiert, die es gestattet, einen Objektbaum (bzw. Mengen) aufzubauen (siehe DMS-Operationen INCLUDE, EXCLUDE). Die Systemattribute SUBTYPE und SUPERTYPE geben für jeden Objekttyp an, welchen anderen Objekttypen ein Objekt dieses Typs unter- oder übergeordnet werden darf. Ein einfaches Beispiel zeigt Abb.30, in der die Objekte C1, E1, F1 dem Objekt A1 und die Objekte D1,C2 dem Objekt E1 untergeordnet sind. Das folgende Beispiel zeigt Ausschnitte einer zugehörigen Deklaration.

Beispiel

```
DCL 1 A BASEOBJECT SUBTYPE(C,E,F)....;
DCL 1 C BASEOBJECT SUPERTYPE(A,C)...;
DCL 1 E BASEOBJECT SUPERTYPE(A) SUBTYPE(D,C)...;
DCL 1 F BASEOBJECT SUPERTYPE(A) SUBTYPE(D,C)...;
DCL 1 D BASEOBJECT SUPERTYPE(E)...;
```

Diese Relation spielt beim DELETE-Befehl und der Datenstrukturanalyse eine Rolle (siehe Kapitel 3.2.).

Anschlüsse

Anschlüsse von Objekttypen beschreiben die Möglichkeiten, ein Objekt dieses Typs in eine Struktur einzuordnen, es mit anderen Objekten zu verbinden. Ein Anschluß wird als Subobjekt des Objektes betrachtet, er wird wie ein Objekt durch spezifische Systemattribute und beliebige Problemattribute beschrieben. Die Systemattribute charakterisieren den speziellen Anschlußtyp, die Anschlußdimension und den Anschlußgrad (Eingangs- bzw. Ausgangsgrad).

Syntax

```
<port>          ::= <portname> PORT <porttype>
                                   {DIMENSION(<dim>)}
                                   {FAN(<fanmin>,<fanmax>)}
                                   {<portattr-list>}
<porttype>      ::= IN | OUT | INOUT

<portattr-list> ::= 3 <attrib> {,3 <attrib>}_0^n
```

Beispiel

```
<port> ::= AND_IN  PORT  IN DIMENSION(2) FAN(1,1)
<port> ::= AND_OUT PORT OUT FAN(0,10)
```

Ein Anschluß ist entweder ein Eingang, ein Ausgang oder ungerichtet. Diese Angabe bestimmt, ob ein Objekt des beschriebenen Typs Element des Vorbereichs, des Nachbereichs oder ein beliebiges Element einer Relation sein darf. Mit der DIMENSION-Klausel wird die Länge eines Anschlußfeldes festgelegt. Die FAN-Klausel gibt an, wie viele Verbindungen gleicher Bedeutung höchstens an einen Anschluß führen dürfen, bzw. wie viele mindestens herangeführt werden müssen. Ist <fanmin>=0, so darf der Anschluß frei bleiben.

Die Beispiele beschreiben die Anschlüsse eines AND-Gatters, dessen zwei Eingänge als Feld betrachtet werden, da sie die gleichen Eigenschaften haben. Beide Eingangselemente müssen angeschlossen werden (damit am Ausgang ein definiertes Signal erscheint). Der Ausgang darf frei bleiben, von ihm dürfen maximal 10 Verbindungen ausgehen.

3.1.3. Beschreibung der Problemdatentypen

Die Systemdatentypen haben eine fest umrissene Bedeutung, die durch die zugehörigen Operationen (siehe Kapitel 3.2.) festgelegt ist und durch die Systemattribute modifiziert werden kann. Bei der Modellklassen-Definition (Schemadefinition) mit Hilfe der von GRIMBI bereitgestellten Datendefinitionssprache (GRIMBI-DDL), deklariert man die interessierenden Problemobjekttypen (Problemdatentypen) als Abbilder (Exemplare, Instances) der Systemtypen, wie sie in Abb.27 bereits aufgeführt sind. Eine Modellklasse ist also eine Menge von DESCRIBE-Anweisungen, die man durch eine Datentyp-Graphik ergänzen kann (siehe Anhang 1 und Kapitel 3.1.4.)

Syntax

```
DESCRIBE 1 <problemtype> <systemtype> ;

        <systemtype> ::= <baseobject> | <relation> | <descriptor> | <net>
                       | <systemport> | <subnetport> | <complexobject>

        <problemtype>::= CHAR8-string
```

Beispiel:

```
DESCRIBE 1 AND <baseobject>...;
```

Liegt eine derartig definierte Menge von Problemdatentypen (d.h. eine Modellklasse) vor, so kann man mit Hilfe der Datenmanipulationssprache Modelle dieser Klasse bearbeiten. Beispielsweise fügt der Befehl

```
INSERT <name> <problemtyp> ...;
```

ein Objekt eines Problemdatentyps unter dem angegebenen Namen in die Datenbasis ein. Für den Problemdatentyp AND aus dem Beispiel sähe die Manipulation so aus:

```
INSERT AND1 AND ....;
```

Soviel zur Verwendung der Problemdatentypen, die einzelnen Operationen für diese Datentypen werden in Abschnitt 3.2. erläutert.

Der Problemdatendefinition entspricht z.B. in PL/1 die Deklaration von Variablen:

```
DCL A BIN FLOAT(21) BASED;  <====> DESCRIBE .....
```

Die Allokierung solcher Variablen und die anschließende Wertzuweisung hat die gleiche Bedeutung wie das INSERT beim Modellieren:

```
ALLOC A; A=20;              <====> INSERT ......
```

Soviel vorweg zur Verwendung der Problemtypen. Im folgenden wird erst einmal die Bedeutung und Anwendung der einzelnen Systemdatentypen besprochen.

BASEOBJECT

Der Systemtyp BASEOBJECT dient der Beschreibung von Anwenderdatentypen, deren Exemplare (Instances) nicht weiter strukturiert werden und die keine Systemschnittstellen oder Relationen darstellen. Objekte vom Typ BASEOBJECT werden durch Systemattribute, Problemattribute und Anschlüsse beschrieben. Sie haben mindestens einen Anschluß, damit sie überhaupt in eine Struktur eingefügt werden können.

Syntax

```
DESCRIBE 1 <problemtype> BASEOBJECT <system-attribs>
                                    {<attriblist>}
                                    {<portlist>};
```

<attriblist> ::= 2 <attrib> {,2 <attrib>}$_0^n$

<portlist> ::= 2 <port> {,2 <port>}$_0^n$

Ein einfaches Beispiel eines BASEOBJECTs ist der im folgenden beschriebene Datentyp Pumpe mit Material-Ein/Ausgängen und Energiezufuhr. Pumpen darf man nur in eine Kolonnen-Untermenge einordnen, den Pumpen dürfen nur Motoren untergeordnet werden.

Beispiel:

```
DESCRIBE 1 PUMPE      BASEOBJECT STEXT('Anwendertyp Pumpe')
                                 SUPERTYPE(KOLONNE) SUBTYPE(MOTOR)
         2 FLSTG      ATTRIBUTE  STEXT('Förderleistung')
                                 REAL VALUESET(10. TO 100.) DEFAULT(20.),
         2 TYP        ATTRIBUTE  STEXT('Pumpentyp') CHAR(8)
                                 VALUESET('KREISEL','ZAHNRAD')
                                 DEFAULT('KREISEL')
         2 MEDIUM     ATTRIBUTE  STEXT('Eingangs-Medium')
                                 CHAR(3) VALUESET('H2O','NA') REQUIRED,
         2 EINGANG    PORT       STEXT('Material-Eingang')
                                 IN FIELD(1) FAN(1,1)
         3 FLANSCH    ATTRIBUTE  CHAR(1) STEXT('Flansch-Klasse')
                                 VALUESET('A','B','C','X') DEFAULT('A'),
         2 AUSGANG    PORT       STEXT('Material-Ausgang')
                                 OUT FIELD(1) FAN(1,4),
         3 FLANSCH    ATTRIBUTE  STEXT('Flansch-Klasse')
                                 CHAR(1) VALUESET('A','Z')
         2 ELEKTRO    PORT       IN FIELD(1) FAN(1,3)
                                 STEXT('Stromanschluß'),
         3 SPANNUNG   ATTRIBUTE  INTEGER STEXT('Anschlußspannung')
                                 VALUESET(110,220,380) DEFAULT(380);
```

RELATION

Relationen beschreiben Anwenderdatentypen, die eine Menge von Objektverbindungen repräsentieren. Im mathematischen Sinne handelt es sich um die Beschreibung einer zweistelligen Relation durch die Festlegung von Restriktionen für die zulässigen Objektpaare, durch die Einschränkung des Vor- und Nachbereichs und die Struktur der Relation.

Syntax

```
DESCRIBE <problemtype> RELATION {<system-attribs>}
                               {FROMDOMAINE(<fromlist>)}
                               {TODOMAINE(<tolist>)}
                               {TYPE(<strutype>)}
                               {RESTRICTION(<restrictionlist>)}
                               {RENTRYCO(<entrylist>)}
                               {RENTRYCL(<entrylist>)}
                               {<attriblist>}
                               {<tupelattrib>};

<restrictionlist> ::=  <operand> <operation> <operand>
                       {,<operand> <operation> <operand>}0n

<operand>         ::=   FROM.<attrname> {(<ind>)}
                      | FROM..<attrname>{(<ind>)}
                      | TO.<attrname>   {(<ind>)}
                      | TO..<attrname>  {(<ind>)}
                      | REL.<attrname>  {(<ind>)}
                      | <const>

<operation>       ::=      = | ¬= | < | > | <= | >=

<tupelattrib>     ::= 2 <tattrname> TUPELATTRIBUTE <datatype>
                                    {DIMENSION(<dim>)}
                                    {VALUESET(<valueset>)}
                                    {DEFAULT(<default>)}
                                    {CLASS(<class>)}
                                    {REQUIRED}

<strutype>::= TREE | .....
```

Beispiel

```
DESCRIBE 1 E_NETZ  RELATION STEXT('Versorgungsnetz')
                           FROMDOMAINE(VERTEILR.AUSGANG)
                           TODOMAINE(PUMPE.ELEKTRO,LAMPE)
                           RESTRICTION(TO..SPANNUNG=REL.SPANNUNG,
                                       FROM..SPANNUNG=REL.SPANNUNG)
         2 SPANNUNG ATTRIBUTE INTEGER
                              VALUESET(220,380) DEFAULT(380),
         2 DURCHM   TUPELATTRIBUTE REAL VALUESET(1. TO 10.)
                                        DEFAULT(1.5);
```

Relationen haben die gleichen Systemattribute wie Basisobjekte, werden ebenfalls durch Problemattribute charakterisiert und verhalten sich bezüglich der Operationen INSERT, DELETE, INCLUDE, EXCLUDE wie diese. Alle anderen Attribute sind spezifisch für Relationen. Sollen in einer Struktur die Verbindungen (Tupel) bewertet werden, so ordnet man der Relation ein TUPELATTRIBUTE zu, das die Möglichkeiten der Bewertung beschreibt.

Die anderen Klauseln dienen der Festlegung von Restriktionen, die beim Zuordnen von Tupeln (d.h. beim Aufbau von Verbindungen zwischen Objekten, siehe CONNECT-Befehl) zu beachten sind. Die FROMDOMAINE- und TODOMAINE-Klauseln spezifizieren den Vor- und Nachbereich der Relation durch Listen von Problemdatentypen und Problemdatentyp-Anschlüssen. Im Kontext einer solchen Relation sind nur Verbindungen zwischen den Anschlüssen der spezifizierten Typen zulässig. Beispielsweise darf eine Relation vom Typ E_NETZ nur Objekte vom Typ VERTEILER mit solchen der Typen PUMPE und LAMPE verbinden, Pumpen und Verteiler nur über den Anschluß ELEKTRO. Um eine Relation auf einen der Typen 1:1, 1:n oder n:m einzuschränken, muß man die FAN-Werte der FROM- bzw. TODOMAINE-Elemente geeignet definieren. Die RESTRICTION-Klausel erfaßt einen Restriktionstyp, der sich auf die Attributwerte der zu paarenden Objekte bezieht. Sie werden durch die angegebenen Vergleichsoperationen zueinander oder zu Konstanten in Beziehung gesetzt. Dabei qualifizieren FROM./TO. Objektattribute von VOR-/Nachbereichselementen , FROM../TO.. Anschlußattribute und REL. Relationsattribute. Konstanten werden gemäß der PL/1-Regeln in die Typen der Attribute gewandelt. Die Einzelrestriktionen der Liste müssen alle erfüllt sein, um ein Tupel in eine Relation des Typs einfügen zu können. Die E_NETZ-Relation läßt beispielsweise nur Verbindungen zwischen Objekten zu, deren Anschlußspannungen der für die Relation vorgesehenen Spannung entsprechen.

Neben diesen lokalen Restriktionen, die sich lediglich auf die zu einer Verbindung gehörenden Objekte beziehen, sind globale Restriktionen vorgesehen. Sie betreffen mögliche Strukturtypen. Bedingt durch den ersten Anwendungsfall für GRIMBI ist bisher ein Typ vorgesehen: Der Typ TREE stellt sicher, daß die Relation schleifenfrei ist. Um aber beliebige Restriktionen und Kombinationen von Restriktionen erfassen zu können, reichen diese Beschreibungsmittel möglicherweise nicht aus. Daher sieht auch GRIMBI für Spezialfälle, ähnlich wie das DBTG-Modell, ergänzend den Einbau von Prüfalgorithmen vor, die der Anwender in der Phase der Modellklassenbeschreibung mit der DML (die eine Erweiterung von PL/1 um die in Kapitel 3.2. beschriebenen Befehle ist) formuliert. Die RENTRYCO/RENTRYCL-Klauseln teilen dem System die Namen zweier solcher Prozeduren mit, die dann nach jedem CONNECT (RENTRYCO) bzw. beim CLOSE SUBNET (RENTRYCL) aufgerufen werden und die Gültigkeit der Struktur prüfen. Eine derartige Prozedur, die zum Systemumfang gehört, ist die Prozedur FREE. Sie prüft bei einem CLOSE NET, ob freie Anschlüsse vorhanden sind und markiert sie.

GRIMBI-Relationstypen

In diesem Zusammenhang wird noch einmal zusammenfassend auf die unterschiedlichen Relationstypen hingewiesen, die in GRIMBI eine Rolle spielen.

Anwenderrelationen sind die eben besprochenen Relationen zur Erfassung der Problemstruktur (m:n-Relationen mit Problemrestriktionen). Sie werden vom Anwender bei der Modellklassendefinition (im Schema) frei festgelegt. Die Befehle CONNECT/DISCONNECT stehen zu ihrer Bearbeitung beim Modellieren bereit.

Die Deskriptor-Teilnetz-Relation (1:n-Relation) hat die Bedeutung "besteht (im Detail) aus" und dient der Abgrenzung von Teilstrukturen. Sie ist dem System bekannt und dient der Modellgliederung in übersichtliche Abschnitte. Diese Relation ist ihrer Struktur nach ein Baum, ein Teilnetz gehört zu genau einem Deskriptor, einem Deskriptor können mehrere Teilnetze (Alternativen) zugeordnet sein. Für diese Relation legt der Anwender lediglich den jeweils zulässigen Objekttyp fest.

Um Mengenbildungen zu ermöglichen, kennt das System eine Mengen-Relation, deren Struktur fest liegt (1:n-Relation zur Zusammenfassung von Objekten innerhalb eines Netzes/Teilnetzes). Der Anwender gibt mit der SUBTYPE- bzw. der SUPERTYPE-Klausel lediglich den Vor- bzw. Nachbereich an (Kapitel 3.1.2. und 3.2.).

Die Objekt-Objekttyp-Relation ist eine m:1-Relation der Bedeutung "wird beschrieben durch". Alle Modellobjekte, die auf einen Objekttyp verweisen, haben die gleiche Grundbedeutung, gehören zu einer Klasse. Auch diese Relation hat eine unveränderliche Struktur.

DESCRIPTOR-SUBNET

Um größe Systeme bei Äuflösung aller Details zu bearbeiten, gliedert man das Gesamtsystem in einzelne überschaubare Teilsysteme, die auf einer Ebene höherer Abstraktion als strukturlose Elemente mit Attributen und Anschlüssen (d.h. wie Basisobjekte) behandelt werden. Es geht darum, ein System durch schrittweises Abstrahieren oder Detaillieren, wie es einleitend für ein CAD-System zum funktionalen Modellieren gefordert wurde, zu bearbeiten. Diesem Ziel dient die Möglichkeit, einen sogenannten DESCRIPTOR-(SUBNET)-Problemdatentyp zu definieren, der einen zulässigen Teilnetztyp bezüglich zweier Abstraktionsebenen charakterisiert. Im Gegensatz zu Objekten vom Typ BASEOBJECT besitzt dieser Objekttyp eine Struktur und eine variable Zahl von Anschlüssen (falls nicht die Option FIXPORTS angegeben wurde). Wie aus der Syntax ersichtlich, wird ein DESCRIPTOR/SUBNET wie BASEOBJECTs und RELATIONs auch durch Problemattribute und Standardattribute beschrieben, darüberhinaus aber durch einige implizit definierte Attribute, denen vom System automatisch Werte zugewiesen werden (Ausnahme: die Werte 4 und 5 des Attributes DEVSTAT, siehe unten). Es handelt sich um folgende implizite Deklarationen, deren Bedeutung aus der Beschreibung hervorgeht.

```
- CR_NAME ATTRIBUTE CHAR(8) STEXT('Name der Erstellers')
- CR_DATE ATTRIBUTE CHAR(6) STEXT('Erstellungsdatum')
- CR_TIME ATTRIBUTE CHAR(6) STEXT('Erstellungszeit')
- RV_NAME ATTRIBUTE CHAR(8) STEXT('Name des Ändernden ')
- RV_DATE ATTRIBUTE CHAR(6) STEXT('Änderungsdatum ')
- RV_TIME ATTRIBUTE CHAR(6) STEXT('Änderungszeit ')
- ACTIVE  ATTRIBUTE BIT(1)  STEXT('aktive Teilnetzalternative')
- DEVSTAT ATTRIBUTE BIN FIXED(15) VALUESET(0 TO 5)
        STEXT('Entwurfsstadium')
```

Dabei bedeutet:

```
DEVSTAT=0 : ungeprüfter Entwurf
       =1 : lokale Konsistenz (siehe RELATION)
       =2 : gültiger Strukturtyp (z.B. TREE)
       =3 : geprüft durch Anwenderalgorithmus
       =4 : vorläufig freigegeben durch Bearbeiter
       =5 : freigegeben durch Bearbeiter
```

Syntax

```
DESCRIBE 1 <problemtype> DESCRIPTOR {<system-attribs>}
                                    {SUBNETDOMAINE(<objtype-list>)}
                                    {FIXPORTS} {<attriblist>},

       {,2 <portname> <snporttype>}_0^n;

<objtype-list> ::= <snproblemtype> {,<snproblemtype>}_0^n

<snproblemtype> ::= <baseobject> | <relation> | <descriptor> |<subnetport>
```

Beispiel

```
DESCRIBE 1 NKSYST DESCRIPTOR STEXT('Notkühlsystem')
                             SUBNETDOMAINE(BEHÄLTER,PUMPE,VENTIL,
                                           NKS_AUS,NKS_EIN,MSR_EA,ELEKTRO,
                                           E_NETZ,KMLTG,MSRLTG)
          2 LEISTUNG ATTRIBUTE BIN FLOAT(21) STEXT('Kühlleistung')
                               VALUESET(10. TO 1000.) DEFAULT(100.),
          2 KUEHLM   ATTRIBUTE CHAR(3) STEXT('Kühlmittel')
                               VALUESET('NA',H2O') DEFAULT('NA'),
          2 KM_EIN1  NKS_EIN,/*Kühlsystemeingang vom Typ NKS_EIN*/
          2 KM_AUS1  NKS_AUS;/*Kühlsystemausgang vom Typ NKS_AUS*/
```

Bei der Strukturierung eines Teilnetzes (Modellieren) ist nur die Verwendung von Objekten derjenigen Problemtypen zulässig, die in der SUBNETDOMAINE-Klausel angegeben sind. In dem Beispiel seien BEHÄLTER, PUMPE, VENTIL BASEOBJECT-Problemdatentypen, NKS_EIN, NKS_AUS, MSR_EA und ELEKTRO SNPORT-Typen (siehe Beispiel SUBNETPORT) und die übrigen vom Typ RELATION. Mit anderen Worten: Ein Notkühlsystem dieses Typs darf nur mit Hilfe der genannten Elementtypen aufgebaut werden. Gibt man die Option FIXPORTS an, so muß mindestens ein Anschluß definiert werden, da beim Modellieren dann keine Anschlüsse ergänzt werden können. Bei einem DESCRIPTOR- Datentyp ist es erlaubt, Anschlüsse sowohl während der Modellklassendefinition als typspezifisch festzulegen, als auch Anschlüsse beim Modellieren zu ergänzen. Ein Notkühlsystem, wie es im Beispiel charakterisiert ist, hat mindestens einen Kühlmitteleingang und einen Kühlmittelausgang, die bei Bedarf während des Modellierens durch weitere Anschlüsse der genannten Typen ergänzt werden können (z.B. INSERT E_EIN ELEKTRO DESCRIPTOR(<descriptor-name>); siehe Kapitel 3.2.4.).

Deskriptor und Teilnetz repräsentieren in verschiedenen Umgebungen das gleiche Objekt, daher referiert man sie auch unabhängig vom aktuellen Kontext mit dem gleichen Namen. Zu einem DESCRIPTOR-Exemplar können mehrere SUBNET-Exemplare gleichen Typs gehören, wobei alle diese Teilnetze die gleichen Anschlüsse besitzen müssen. Das hat zur Folge, daß implizit in alle Nachbarteilnetze ein Anschluß eingefügt wird, wenn man eines der Teilnetze um einen Anschluß erweitert. Natürlich gilt das auch für den zugehörigen Deskriptor. Alle zu einem Deskriptor gehörenden Teilnetze bilden ein Feld, sie werden daher durch ihren indizierten Namen referiert. Diese Strukturierungsmöglichkeit dient der Verwaltung von Entwurfsalternativen. Um eine eindeutige Auswertung einer solchen Struktur zu ermöglichen, wird mit dem Befehl ACTIVATE (Kapitel 3.2.) eine der Teilnetz-Alternativen als aktiv gekennzeichnet (implizit definiertes Attribut ACTIVE).

GRIMBI erlaubt beim Aufbau eines Modells mit diesen Mitteln nur echte Bäume (d.h. ein Teilnetz-Exemplar hat nur einen Deskriptor) und keine Rekursion.

NET

Problemdatentypen der Art NET repräsentieren die oberste Abstraktionsebene des zu bearbeitenden Systems, im übertragenden Sinne stellen sie die Übersichtszeichnung eines Zeichnungssatzes dar. Wie Teilnetze besitzen sie die Systemattribute und die implizit definierten Bearbeitungsattribute, es können ihnen auch beliebige Problemattribute zugeordnet werden. Ein Modell darf nur ein Exemplar eines NET-Datentyps und lediglich Anschlüsse des Typs SYSTEMPORT, d.h. Anschlüsse, die auf die Außenwelt des zu modellierenden Systems verweisen. Die NETDOMAINE- Klausel enthält alle für die Strukturierung eines Netzes zulässigen Problemdatentypen.

Syntax

```
DESCRIBE 1 <problemtype> NET {<system-attribs}
                             {NETDOMAINE(<net-problemtype-list>)}
                             {<attriblist>};

<net-problemtype-list> ::= <net-problemtype>

                           {,<net-problemtype>}_0^n
<net-problemtype>      ::= <baseobject> | <descriptor> | <systemport>
                           |<relation>
```

Beispiel

```
DESCRIBE 1 KW        NET        STEXT('Kraftwerks-Übersicht')
                                (NETDOMAINE(W_GEN, KÜHLSYST, NOTKSYST, E_GEN,
                                 E_QUELLE, E_SENKE , ROHRNETZ, E_NETZ)
         2 TYPE      ATTRIBUTE STEXT('Kraftwerkstyp') CHAR(8) DEFAULT('KKW')
                                VALUESET('KKW','GASKW','KOHLEKW','H2OKW'),
         2 LEISTUNG ATTRIBUTE BIN FLOAT(21) VALUESET(1. TO 1000.)
                                DEFAULT(300.);
```

Das Beispiel zeigt die Definition eines Problemdatentyps der Art NET, dessen Exemplare stark abstrahierte Kraftwerkstrukturen aufnehmen können. Für den Aufbau einer derartigen Struktur stehen Bauelementtypen zur Verfügung: Wärmegenerator (W_GEN), Kühlsystem (KÜHLSYST), Notkühlsystem (NOTKSYST), Stromgenerator (E_GEN) seien DESCRIPTOR-Typen, da sie im Verlaufe des Entwurfsprozesses detailliert werden sollen, Energiequelle (E_QUELLE), Energiesenke (E_SENKE) bilden die Kraftwerkschnittstellen zur Umwelt und die RELATION-Typen ROHRNETZ und E_NETZ dienen zur Verbindung der Elemente.

SYSTEMPORT-SUBNETPORT

Objekte dieser Typen bilden die System- und Teilsystemgrenzen. Es sind die Quellen und Senken der Interessenbereiche. Während die SYSTEMPORT- Objekte auf die Umgebung des zu modellierenden Systems verweisen, zeigen die SUBNETPORT - Objekte (bzw. die entsprechenden DESCRIPTOR- Anschlüsse auf SUBNETPORT - Objekte anderer Teilsysteme.

Syntax

```
DESCRIBE 1 <problemtype> <netport> {<system-attribs>}
                                   {porttype>}
                                   {DIMENSION(<dim>)}
                                   {FAN(<fanmin>,<fanmax>)}
                                   {<attriblist>};
<netport> ::= SYSTEMPORT | SUBNETPORT
```

Beispiel

```
DESCRIBE 1 NKS_EIN SUBNETPORT IN FAN(1,1),
        2 MEDIUM  ATTRIBUTE CHAR(3) VALUESET('NA','H2O'),
        2 FLANSCH ATTRIBUTE BIN FIXED(15) VALUESET(1 TO 10) DEFAULT(1);
```

COMPLEXOBJECT

Komplexobjekte sind Typen, denen zusätzlich zu Anschlüssen und Attributen bei der Modellklassendefinition eine Struktur (Teilnetz) aus Basisobjekten und Relationsobjekten zugeordnet wird. Die Attribute eines solchen Problemtyps korrespondieren mit Attributen der Objekte des Komplexes, die Anschlüsse entsprechen den Subnetports. Die Werte der einzelnen Komplextyp- Exemplare ersetzen die Werte einzelner Strukturelemente. Es handelt sich daher um eine parametrisierte Struktur, wobei sich die Parametrisierung allerdings nicht auf die Topologie beziehen kann (Abb.31).

Syntax

```
DESCRIBE 1 <problemtype> COMPLEX   <system-attribs> ,

     {2 <attrname> ATTRIBUTE LIKE(<structure-object-attrib>)}0n;

 {INSERT <structure-object-name> <problemtype> <properties>;}2n

 {CONNECT <port> <port> RELATION(<relname>,<value>);}1n
END COMPLEX;
<complex-problemtype> ::= <baseobject> | <relation>

<properties> ::= {<attrname>(<valuelist>)}0n
```

Die Struktur eines Komplextyps wird mit INSERT- und CONNECT-Befehlen beschrieben (entsprechend der Modellierung, Kapitel 3.2.). Die Anschlüsse entsprechen den SUBNETPORTs der zugeordneten Struktur, sie haben auch den gleichen Namen. Die Komplexobjektattribute werden frei benannt, entsprechen aber Attributen der Strukturobjekte (LIKE-Klausel). Wird daher einem Objekt der Art COMPLEXOBJECT ein Wert zugewiesen, so wird er dem in der LIKE-Klausel angegebenen Strukturobjekt-Attribut zugeordnet.Bei den Komplexobjekt-Attributen handelt es sich gewissermaßen um formale Parameter wie bei Prozeduren. Objekttypen dieser Art haben ihren Sinn, wenn Strukturen sich häufig ändern, was jeweils auch eine Änderung der Auswertealgorithmen nach sich zöge, falls für jeden

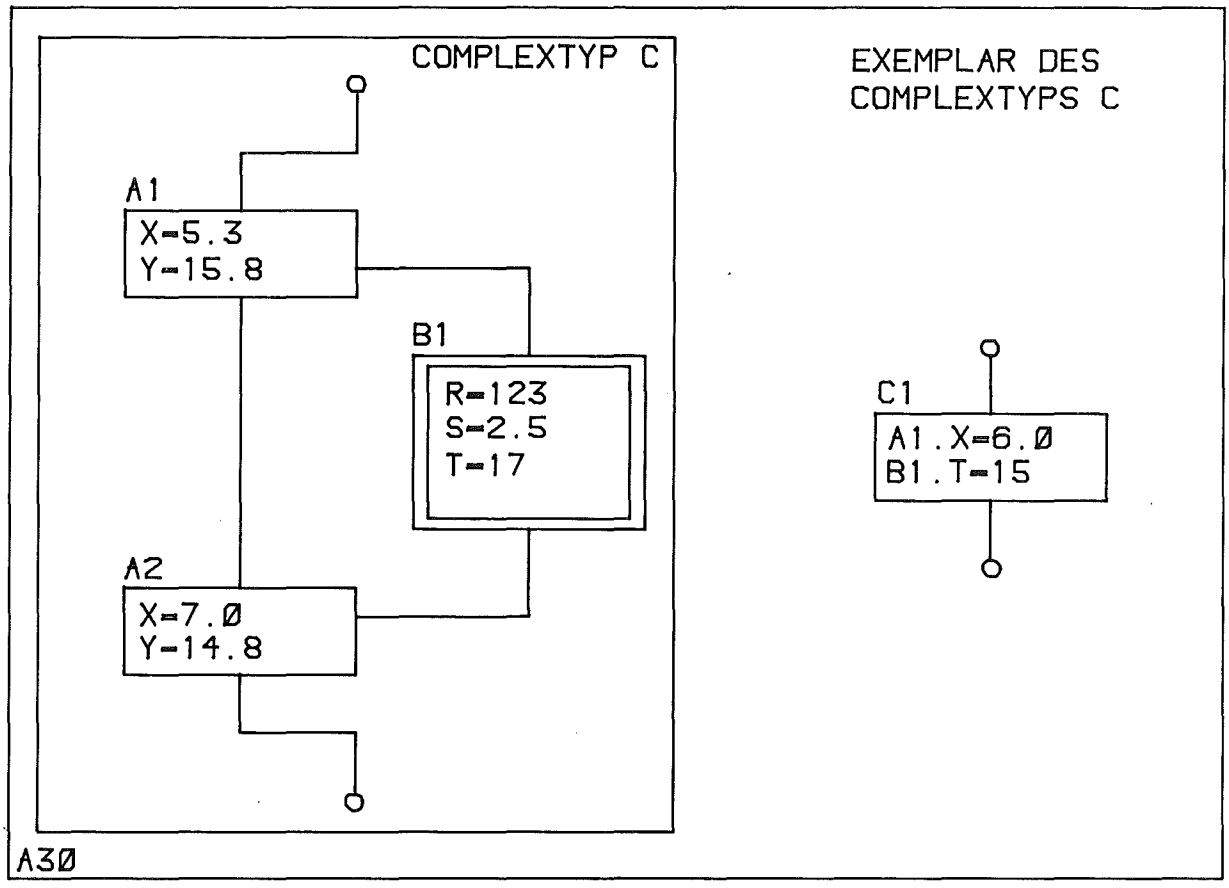


Abb.31: Komplexobjekt

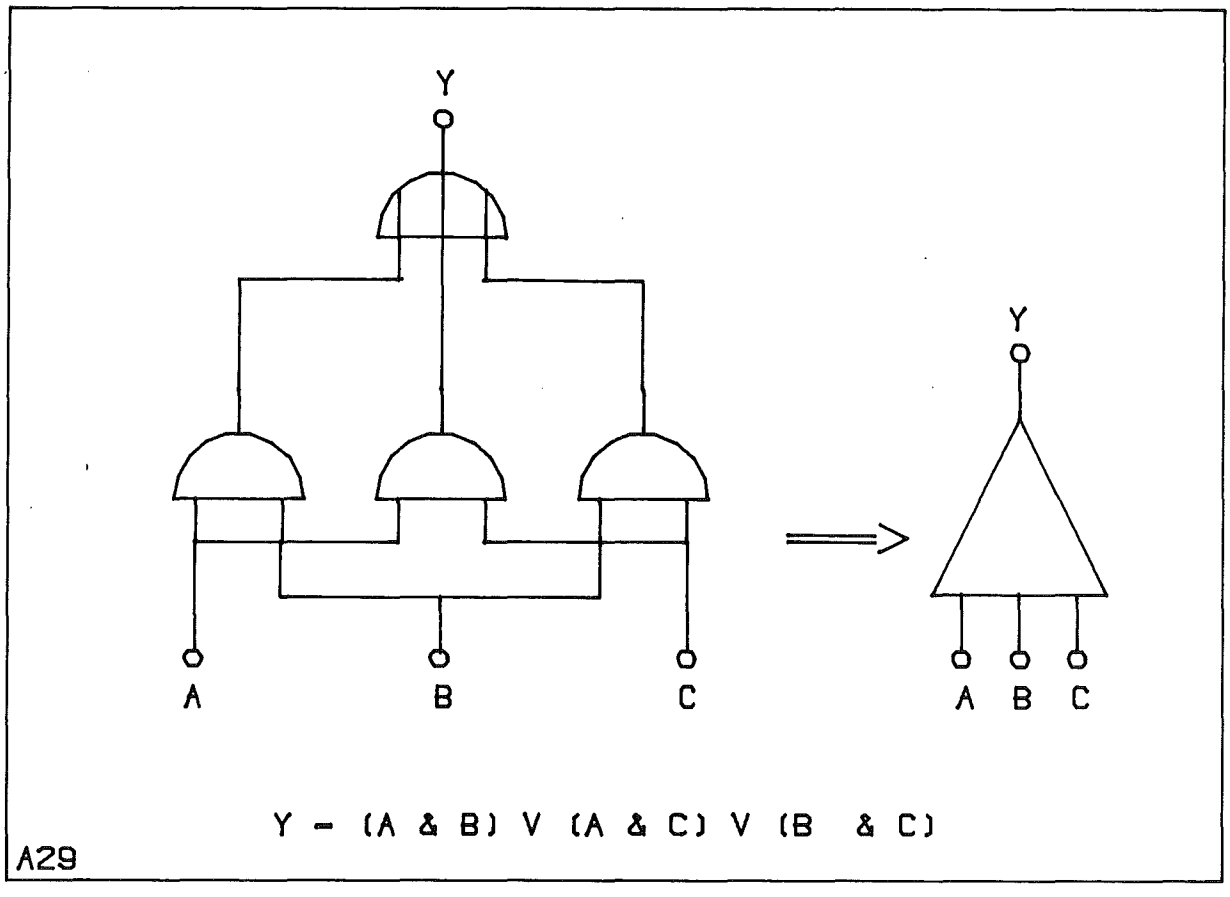


Abb.32: Beispiel eines Komplexobjektes

derartigen Strukturtyp ein Basisobjekttyp stände und damit das Wissen über die Struktur in den Auswertealgorithmen vorliegen müßte. Sind diese Strukturen aber ebenso stabil wie die anderen Problemtypen, so wird das Wissen über die Struktur besser in die Algorithmen aufgenommen.

Ein einfaches Beispiel für eine Komplexstruktur ist in der Fehlerbaumanalyse ein Logikelement mit der Funktion einer 2aus3-Auswahl (Abb.32).

Die Komplextyp-Bildung ist auch während des Modellierens möglich, indem der Anwender eine Bibliothek gewissermaßen als dynamische Schemaerweiterung mit Komplexstrukturen aufbaut (Kapitel 3.2.: DESCRIBE COMPLEX).

BANK-NETPOOL

Die bisher betrachteten Datentypen dienen der Beschreibung von Problemdatentypen, mit denen der Anwender arbeiten will. Daneben kennt das System aber zwei weitere Datentypen : BANK und NETPOOL. Ihre Bedeutung ist problemunabhängig, invariant. Sie werden vom Anwender mit Hilfe der Operationen (Kapitel 3.2) so wie die Problemdatentypen bearbeitet. Eine BANK umfaßt eine Modellklassenbeschreibung und beliebig viele Objekte vom Typ NETPOOL. NETPOOL-Objekte repräsentieren ein Modell der zugehörigen Modellklasse (Abb.33), sie bestehen aus einem Netz und beliebig vielen Teilnetzen, repräsentieren also gewissermaßen einen "Zeichnungssatz", in dem das Netz die "Übersichtszeichnung" darstellt.

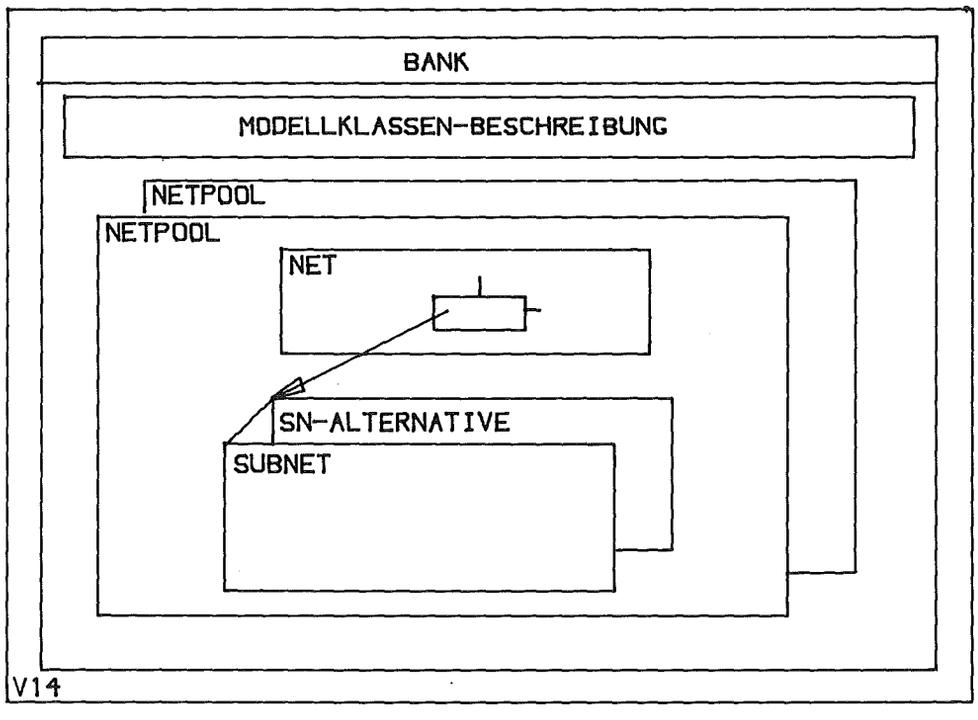


Abb.33: Datentypen der Verwaltung

3.1.4. Graphik der Problemdatentypen

Um mit GRIMBI interaktiv, graphisch unterstützt arbeiten zu können, enthält der Datendefinitionsteil Anweisungen, mit denen einer Modellklasse eine geeignete graphische Repräsentation zugeordnet werden kann (Abb.27). Die graphische Datentypbeschreibung muß nicht zusammen mit der logischen Beschreibung erfolgen, sondern kann zu einem späteren Zeitpunkt ergänzt oder geändert werden (Anhang 1). Diese Beschreibung legt die graphische Darstellung der Modelldaten fest. Sie entspricht damit in ihrer Bedeutung der FORMAT-Anweisung bzw. der FORMAT-Liste einiger Programmiersprachen (FORTRAN, PL/1), die allerdings lediglich die Datendarstellung durch Zeichenketten erfaßt.

Beispiel

```
DCL A BIN FLOAT(21), N BIN FIXED(31);
F1: FORMAT(SKIP,E10.7);
PUT EDIT(A,N)(R(F1),X(3),F(8));
```

Diese FORMAT-Angabe legt fest, daß der darzustellende Wert, in diesem Falle der Wert der Variablen A, als Zeichenkette der Form -0.500E-02 ausgegeben wird.

Während diese Beschreibung sich lediglich auf die externe Datendarstellung bezieht, wird mit der GRIMBI-DDL auch die Datenidentifizierung durch Zeigen erfaßt, die für die interaktive Arbeit unverzichtbar ist.

Wegen der logischen Strukturierung der Datentypen (in Anschlüsse, Attribute etc.) sind auch die Symbole, die den Datentypen zugeordnet werden, strukturiert.

Ein Symbol besteht aus folgenden graphischen Elementen (Abb.34):

(1) ein Grundsymbol, das die Bedeutung des zugehörigen Objektes ausdrückt,
(2) Anschlußsymbole, die jeweils einen Anschluß repräsentieren,
(3) Attributsymbole,die Attribute darstellen und
(4) Textfenster, Bereiche mit einer Zeilen/Spalten-Struktur, in denen Texte dargestellt werden.

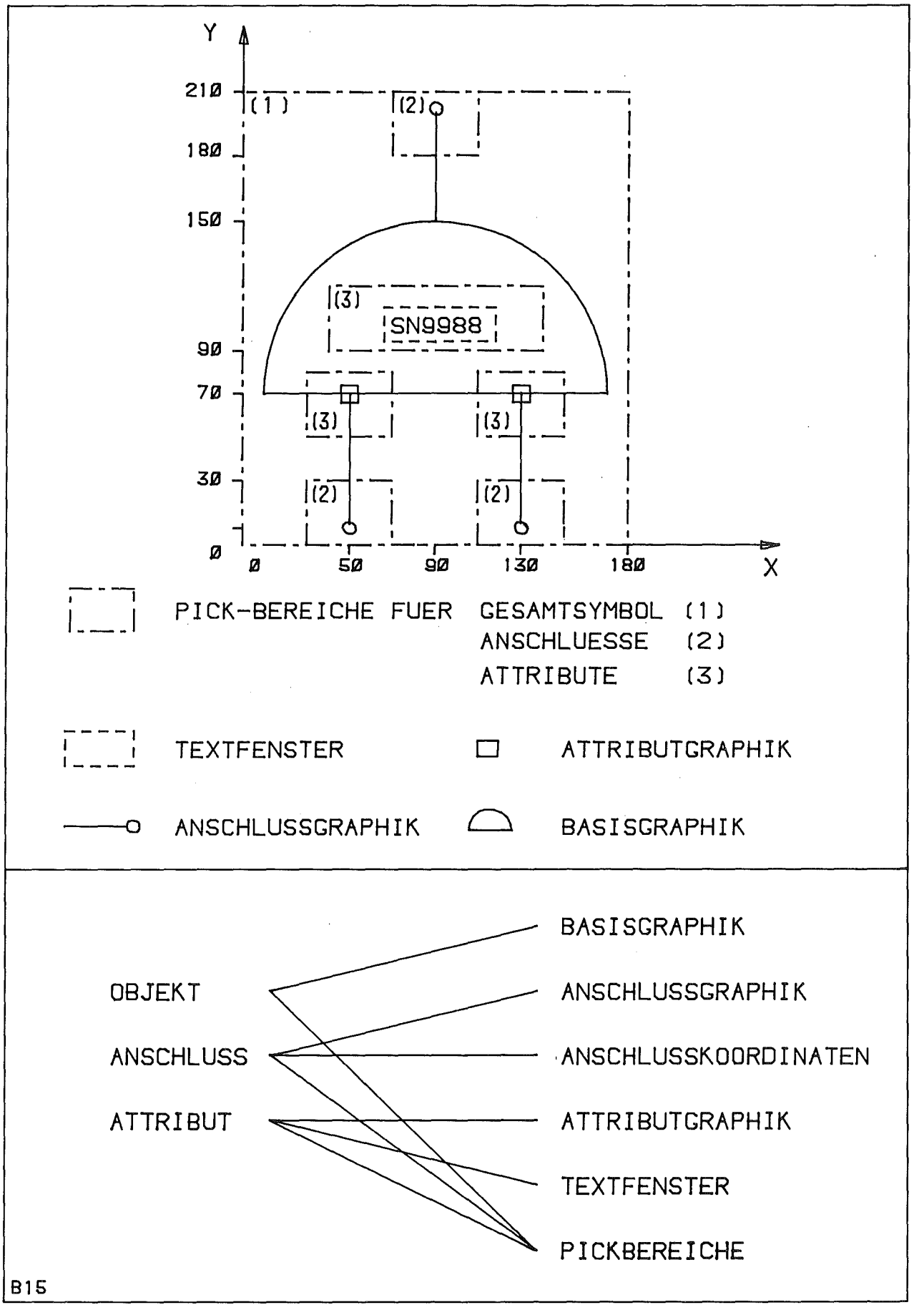


Abb.34: BASEOBJECT-Graphik

Ebenfalls graphisch beschrieben, aber nicht dargestellt werden

(5) Anschlußkoordinaten, d.h. die Punkte, von denen die Verbindungen ausgehen und
(6) Pickbereiche, die dem Gesamtobjekt, den Anschlüssen und den Attributen zugeordnet werden.

Die vollständige Syntax dieses Definitionsteiles befindet sich in Anhang 1. Sie wird im folgenden schrittweise erläutert. Der Graphik-Teil der Datendefinition sieht so aus (siehe auch Beispiel-Datendefinition zu Abb.34 am Ende diese Kapitels):

Syntax

<graphic> ::= OPEN GRAPHIC(<grid>);

{<objtype-representation>}$_1^n$

CLOSE GRAPHIC;

Der Wert <grid> legt im Symbolkoordinatensystem (siehe unten) ein Raster fest, das für die Positionierung des Symbols und die Anschlußkoordinaten verwendet wird. Mit dem Programmteil <objtype-representation> beschreibt man dann für jeden Problemdatentyp eine Symbolik, wobei die oben genannten sechs Symbol-Elemente unterschieden werden.

Syntax

<objtype-representation> ::= OPEN OBJTYPE <systemtype> <problemtype>;

<gr-repr> {<gr-repr>}$_0^n$

CLOSE OBJECTTYPE;

<gr-repr> ::= <pick> | <portcoord> | <basegraphic>
| <portgraphic> | <attribgraphic> | <textwindow>

Das Grundsymbol eines Objekttyps wird durch den BASEGRAPHIC-Teil beschrieben, dessen Syntax so aussieht:

<basegraphic> ::= BASEGRAPHIC;

$\{$<gr-object>$\}_1^n$
END BASEGRAPHIC;

Die Anschlußsymbole werden unabhängig vom Grundsymbol behandelt, um sie gegebenenfalls verschwinden zu lassen (z.B. wenn sie nicht belegt sind, um sie verschieben zu können (z.B. bei Deskriptoren) oder um sie blinken zu lassen. Ebenfalls zur Beschreibung der Anschlüsse gehört die Angabe desjenigen Punktes, an den eine Verbindung herangeführt wird. Der Anwender (oder Modellklassen-Administrator) hat dafür zu sorgen, daß dieser Punkt auch ein Punkt der Anschlußgraphik ist. Die Zuordnung einer Anschlußbeschreibung zu dem entsprechenden logischen Datenelement erfolgt durch den Anschlußnamen und bei Anschlußfeldern zusätzlich einen Index.

Syntax

<portgraphic> ::= PORTGRAPHIC <which>;

$\{$<gr-object>$\}_1^n$
END PORTGRAPHIC;

<portcoord>::= PORTCOORD <which> <ix> <iy> $\{$,<which> <ix> <iy>$\}_0^n$
<which> ::= <portname>{(<ind>)}

Für die graphische Darstellung der Attributwerte sind zwei Möglichkeiten vorgesehen: ihre Repräsentation durch Texte oder durch eine Symbolik (Attributgraphik). Sogenannte Textfenster legen die Position (<x> <y>) und Gestalt (<dx> <dy>,d.h. Breite und Höhe) der textlichen Wertdarstellung fest.

Syntax

<textwindow> ::= TEXTWINDOW {<x> <y> <dx> <dy>
RASTER(<lines>,<cols>)

$\{$FORMAT(<pl/1-format>)$\}_0^n$

Die RASTER-Klausel bestimmt die Aufteilung des Textes in Zeilen und Spalten. Die Art der Wandlung der Werte in eine Zeichenkette wird mit der FORMAT-Klausel angegeben. Die Zuordnung eines Attributes zu einem Textfenster erfolgt durch die CLASS-Klausel: CLASS(n) bedeutet, daß der entsprechende Attributwert im n-ten Fenster erscheinen soll. Sind einem Textfenster mehrere Attribute zugeordnet, so wird durch CHANGE PROPRTYLEVEL (Kapitel 3.2) angegeben, welcher der Werte graphisch darzustellen ist. Wird zu einem Attribut CLASS(0) angegeben, so soll das Attribut durch eine Attributgraphik dargestellt werden. Die Sichtbarkeit der Attributgraphik ist von den Attributwerten abhängig: Ein Attributsymbol ergänzt das Grundsymbol, wenn dem Attribut ein bestimmter Wert oder Wertebereich zugeordnet ist. Fehlt die CLASS-Angabe, so handelt es sich um ein Attribut ohne graphische Darstellung.

Syntax

<attribgraphic> ::=
ATTRGRAPHIC <attrname>{(<ind>)} (<comp><value>,{<comp><value>});

$\{$<gr-object>$\}_1^n$
END ATTRGRAPHIC;
<comp> ::= > | < | <= | >= | = | ¬=

Im Gattersymbol des Beispiels erscheint die Signalnegation, wenn dem zugehörigen Attribut IN.NEG der Wert "1"B zugewiesen ist. Ein anderer Anwendungsfall wäre die graphische Darstellung eines Behälter-Füllstandes. GRIMBI realisiert damit eine parametrisierte Symbolik, die besonders wichtig ist, wenn man Werte einer Datenbank darstellen will.

Die Pick-Bereiche werden explizit als Rechteckbereiche unabhängig von der Graphik definiert, um auch Datenbasis-Elemente eindeutig identifizieren zu können, deren graphische Repräsentationen sich berühren oder überschneiden. Diese Technik ermöglicht es, jeweils einem der Attributwerte keine Graphik zuzuordnen, ohne die Identifikation des Attributes zu verhindern. Das NEG-Attribut des Beispiels läßt sich so auch im Zustand NEG="0"B identifizieren. Diese Art der PICK-Definition hat, gegenüber der PICK-Definition durch die Symbolik, den Vorteil der optimalen individuellen Festlegung des Unschärfebereiches für das Zeigen.

Syntax

<pick> ::= PICKAREA <what> <where> $\{$<what> <where>$\}_0^n$
<what> ::= * | <portname>{(<ind1>)} | *.<attr>{(<ind2>)}
| <portname>.<attr>{(<ind1,ind2>)}
<where> ::= <ix> <iy> <idx> <idy>

Mit * wird der Objekttyp als Ganzes referiert, *.<attr> referiert ein Objektattribut.

Die bisherigen Ausführungen gelten uneingeschränkt für Problemtypen der Art BASEOBJECT, Besonderheiten der anderen Typen werden im folgenden erläutert.

RELATION-Graphik

Das Symbol für einen Problemtyp der Art RELATION stellt einen Teil der Zeichnungslegende für die Erläuterung der verschiedenen Verbindungstypen in einem Blockdiagramm dar (Abb.35). Da eine Relation keine Anschlüsse besitzt, entfallen die entsprechenden Beschreibungsteile, alle anderen gelten wie bei Basisobjekten. Spezifisch für Relationen ist die Symbolik der Verbindungslinien (Tupel).

Syntax

```
OPEN OBJTYPE RELATION <problemtype>;
 <basegraphic>;
 <pickarea>;
 <textwindow>;
 TUPELGRAPHIC LINESTYLE(<linestyle>)
              TEXTWINDOW(<dx>,<dy>,<tlen>) {FORMAT(<pl/l-format>)}
              {FONT(<font>)} {SLANT(<slant>)}
CLOSE OBJTYPE;
```

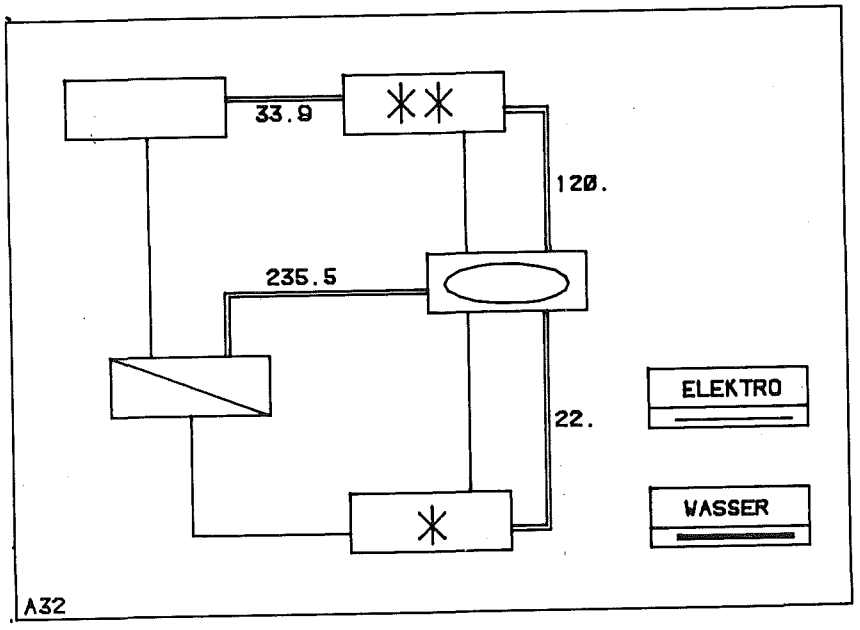


Abb.35: RELATION-Graphik

Mit dieser Syntax beschreibt man die Darstellung der Verbindungslinien (LINESTYLE) und die der Tupelattribut-Werte durch einen Text in einem Textfenster der Größe (<dx>,<dy>) mit <tlen> Zeichen, unter Verwendung des Zeichensatzes <font> und der Zeichenneigung <shearing>. Die Positionierung dieses Textfensters erfolgt beim Editieren der Verbindung (CONNECT, ROUTE). Ein Tupelattribut benötigt keinen PICK-Bereich, da einer Verbindung nur ein Attribut zugeordnet ist und daher die Linie selbst zur Identifizierung verwendet werden kann.

NET-Graphik

Ein Netz stellt die Übersichtszeichnung dar, sie wird graphisch durch das Zeichnungsschriftfeld repräsentiert, das als BASEGRAPHIC beschrieben wird. Das Netz hat keine Anschlüsse, so daß sowohl die PORTGRAPHIC als auch die PORTCOORDs-Abschnitte entfallen.

Syntax

```
OPEN OBJTYPE NET <problemtype>;
 <basegraphic>;
CLOSE OBJTYPE;
```

DESCRIPTOR/SUBNET-Graphik

Objekte dieses Typs haben zwei Erscheinungsformen: Sie sehen einerseits wie Basisobjekte aus, andererseits aber wie ein Netz (beim Detaillieren). Da sie eine variable Zahl von Anschlüssen haben und die Anschlüsse sich auf den beiden Abstraktionsebenen entsprechen, muß automatisch für eine einfache graphische Zuordnung gesorgt werden. Dies geschieht durch eine proportionale Positionierung der zusammengehörigen Anschlußsymbole in vier vorgegebenen Randbereichen (Abb.36) des Teilnetzes bzw. des Deskriptors. Fügt man beispielsweise einen Anschluß in einen der vier Deskriptorbereiche ein, so wird automatisch das zugehörige SUBNETPORT-Symbol in dem entsprechenden Subnet-Bereich an der entsprechenden Position ergänzt. Die Anschlußsymbolik wird für den Bereich 1 (oben) definiert und entsprechend um 90, 180 oder 270 Grad gedreht.Das System sorgt dafür, daß der zu jedem Deskriptoranschluß anzugebende POSITIONPOINT sich immer auf einer inneren Anschlußbereichsgrenze befindet, die mit der PORTPOSITION-Klausel spezifiziert wird.

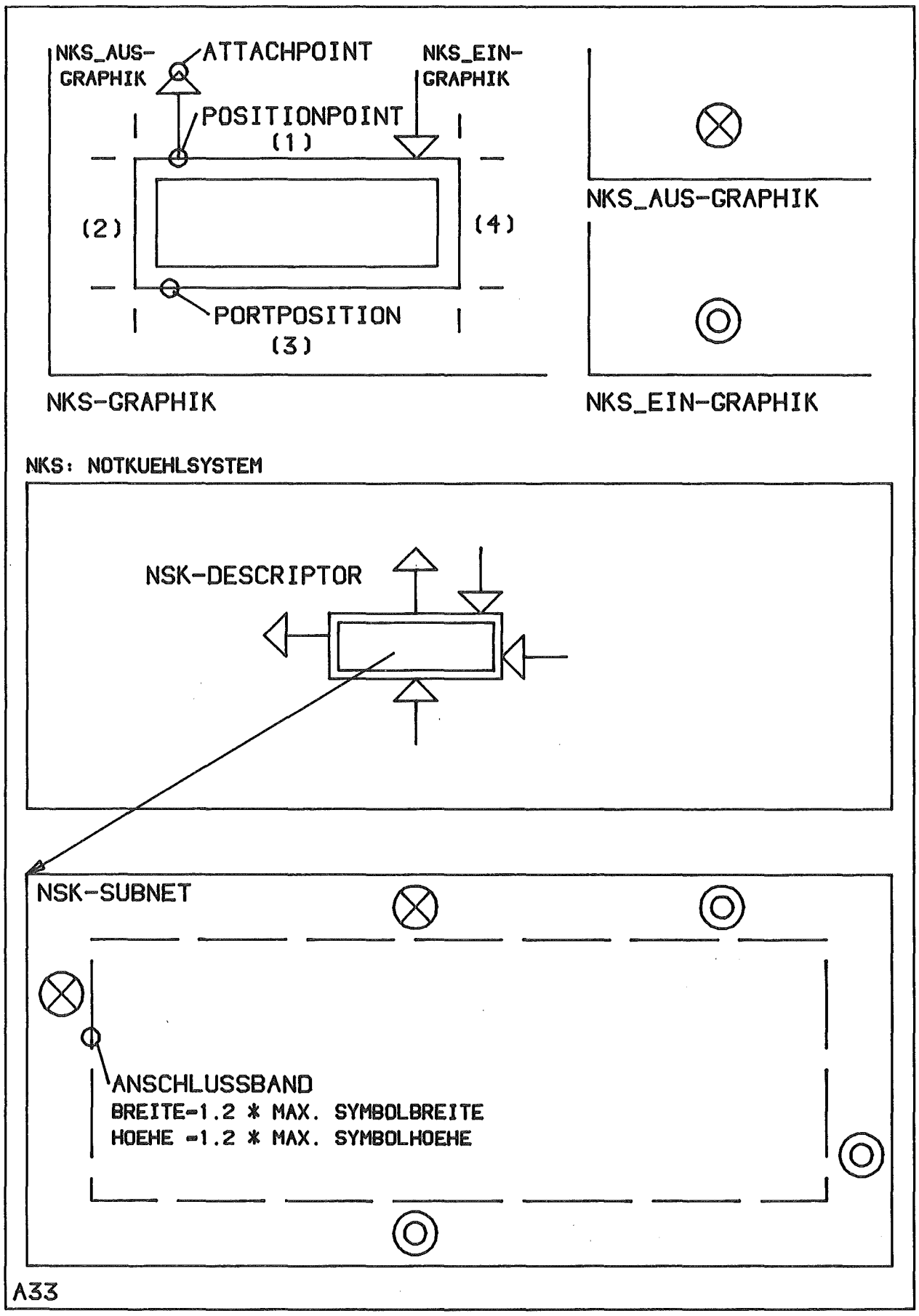


Abb. 36: DESCRIPTOR/SUBNET-Graphik

Die ATTACHPOINT-Klausel enthält die Koordinaten für den Punkt, an den die Verbindungen herangeführt werden sollen. Der Mittelpunk der SUBNETPORT-Symbole liegt immer auf der Anschlußbereichs-Mittellinie. Die Breite/Höhe des Anschlußbereiches entspricht der 1,2-fachen Breite/Höhe des größten Anschlußsymbols.

Die Beschreibung eines Problemtyps der Art DESCRIPTOR/SUBNET gliedert sich in zwei Abschnitte: im einen wird der eigentliche Deskriptor als Symbol mit Anschlüssen charakterisiert, in dem anderen das Teilnetz durch ein Schriftfeld. In beiden Abschnitten beziehen sich die Attributgraphik, die Pickbereiche und die Textfenster nicht auf Anschlüsse. Für sie ist auf der Deskriptor-Ebene die DESCPORTGRAPHIC vorgesehen, auf der Teilnetz-Ebene erfolgt die graphische Anschlußbeschreibung durch die SUBNETPORT-Graphik.

Syntax

```
OPEN OBJTYPE DESCRIPTOR <name>;
 DESCRIPTOR; /*Beschreibt Deskriptorsymbol*/
  <basegraphic>;
  <attributgraphic>;
  <pickarea>;
  <textwindow>;
  {<desc-port-graphic>}(1:np) /*np: Zahl der SUBNETPORT-Typen
                                 in SUBNETDOMAINE-Klausel*/
  PORTPOSITION <x> <y> <dx> <dy>;
 END DESCRIPTOR;
 SUBNET; /*Beschreibt Zeichnungs-Schriftfeld*/
  <basegraphic>;
  <attributgraphic>;
  <pickarea>;
  <textwindow>;
 END SUBNET;
CLOSE OBJTYPE;

<desc-port-graphic> ::= DESCPORTGRAPHIC <subnetporttype>;
                        {<gr-object>}_1^n;
                        POSITIONPOINT <x> <y>;
                        ATTACHPOINT <x> <y>;
                        PICKAREA <x> <y> <dx> <dy>;
                       END DESCPORTTYPE;
```

SYTEMPORT/SUBNETPORT-Graphik

Problemtypen dieser Art werden wie Basisobjekte ohne Anschlüsse beschrieben.

Syntax

```
OPEN OBJTYPE <port-type> <problemtype>;
 <basegraphic>;
 <textwindow>;
 <attributgraphic>;
 <pickarea>;
CLOSE OBJETTYPE;
<port-type> ::= SYSTEMPORT | SUBNETPORT
```

Graphik-Objekte

Zur Beschreibung dieser Symbolkomponenten in einem Symbolkoordinatensystem stehen folgende graphische Elemente zur Verfügung, deren Syntax anschließend dargestellt ist:

- Polygonzug (offen, geschlossen): POLY
- Punkthaufen mit Markierungssymbolen: POINTS
- Kreis, Kreisbogen: CIRCLE
- Text: TEXT

Geschlossene Polygonzüge und Kreise können durch eine Schraffur (HATCH) ergänzt werden, deren Neigung und Abstand angegeben wird. Sie werden durch eine Reihe graphischer Attribute ergänzt:

- Zeichensatz: FONT
- Zeichenbreite, Zeichenhöhe: WIDTH,HIGHT
- Zeichenneigung: SLANT
- Markierungssymboltyp: SYMBOL
- Linientyp: LINESTYLE
- Raster krummer Linien: GRAIN

Syntax

```
<gr-object> ::= <poly>  | <points> | <text> | <circle> | <gr-attrib>

<poly>      ::= POLY <poly-type> {<x> <y>}1n
<poly-type> ::= OPEN | CLOSE {HATCH(<alpha>,<dx>)}

<points>    ::= POINTS {<x> <y>}1n;
<text>      ::= TEXT <x> <y> <alpha> STRING(<string>);
<circle>    ::= CIRCLE <x> <y> <radius> <alpha1> <alpha2>
                       {HATCH(<alpha>,<dx>)}

<gr-attrib> ::= GRATTRIBUTE {<attrparm>}1n;
<attrparm>  ::= FONT(<font> | <SLANT(<slant>) | WIDTH(<width>)
              | HIGHT(<hight>) | LINESTYLE(<linestyle>)
              | GRAIN(<grain>) | SYMBOL(<symbol>)
```

Die Symbole werden in einem katesischen Koordinatensystem mit 0 <= x,y <= 256 beschrieben. Die Koordinaten der graphischen Elemente Polygonzug, Kreis etc. sind REAL-Werte in diesem Bereich, die Positionen und die Gestalt der Textfenster und der Pickbereiche werden als INTEGER- Werte erwartet. Für die Lage der Anschlußpunkte wird ein gesondertes Raster festgelegt (GRID bei OPEN GRAPHIC), das auch für die Symbolpositionierung verwendet wird. Ein GRID-Wert von 10 bedeutet, daß nur die Koordinatenwerte 0 10 20 30 ... akzeptiert, bzw. andere Werte auf diese auf- oder abgerundet werden.

Beispiel zu Abb.34: GRIMBI-DDL für ein UND-Gatter.

```
ENTER DBANET(BANK1) NEW('R','W','U');
 DESCRIBE
   1 AND   BASEOBJECT,
   2 TYPE  ATTRIBUTE CHAR(6) STEXT('Kennzeichen für Gattertyp'),
                     VALUESET('SN0000' TO 'SN0000') CLASS(1) REQUIRED
   2 IN    PORT      IN DIM(2) FAN(1) STEXT('Gattereingangsfeld'),
   3 NEG   ATTRIBUTE BIT(1) DEFAULT('0'b) CLASS(0) STEXT('Signalnegation'),
   2 OUT   PORT      OUT FAN(5),
   3 NEG   ATTRIBUTE BIT(1) DEFAULT('0'b) CLASS(0) STEXT('Signalnegation');

OPEN GRAPHIC(10);
 OPEN OBJTYPE BASEOBJECT AND;
  PICKAREA * 180 210, IN(1) 30 0 40 30, IN(2) 110 0 40 30,OUT(1) 70 180  40 30,
           IN.NEG(1)  30 50 40 30, IN.NEG(2) 110 50 40 30, *.TYPE 40 90 100 30;
  PORTCOORD IN(1) 50 10, IN(2) 130 10, OUT 90 200;
  TEXTWINDOW 60 95 60 15 RASTER(1,6);
  BASEGRAPHIC;
   GRATTRIBUTE LINESTYLE(1) GRAIN(10.);
   POLY OPEN  10 70 170 70; CIRCLE 90. 70. 80. 0. 180.;
  END BASEGRAPHIC;
  ATTRIBGRAPHIC IN.NEG(1) (='1'B);
   GRATTRIBUTE LINESTYLE(1); POLY CLOSED 45 65 55 65 55 75 45 75;
  END ATTRIBGRAPHIC;
  ATTRIBGRAPHIC IN.NEG(2) (='1'B);
   GRATTRIBUTE LINESTYLE(1); POLY CLOSED 125 65 135 65 135 75 125 75;
  END ATTRIBGRAPHIC;
  PORTGRAPHIC IN(1);
   GRATTRIBUTE LINESTYLE(1) GRAIN(10.);
   POLY OPEN 50 15 50 70; CIRCLE 50. 10. 5.0 0. 360.;
  END PORTGRAPHIC;
  PORTGRAPHIC IN(2);
   GRATTRIBUTE LINESTYLE(1) GRAIN(10.);
   POLY OPEN 130 15 130 70; CIRCLE 130. 10. 5.0 0. 360.;
  END PORTGRAPHIC;
  PORTGRAPHIC OUT;
   GRATTRIBUTE LINESTYLE(1) GRAIN(10.);
   POLY OPEN 90 150 90 195; CIRCLE 90. 200. 5.0 0. 360.;
  END PORTGRAPHIC;
 CLOSE OBJTYPE;
```

3.2. Modellieren mit GRIMBI

Beim Modellieren geht es nun darum, entsprechend den Restriktionen einer Modellklasse (Schema), ein konkretes Modell zu erstellen, indem Objekte der Problemtypen generiert, ihren Attributen Werte zugewiesen und die Objekte zu Strukturen verknüpft werden. Die Wirkungsweise der Operationen wird durch das Schema eingeschränkt: Es können beispielsweise weder beliebige Objekte in die Datenbasis eingefügt noch beliebige Objekte zu einer Struktur verbunden werden, um die wichtigsten Operationen zu nennen. Um eine hohe Flexibilität zu gewährleisten, interpretieren die Operationen das zur jeweiligen Bank gehörende Schema, so daß ein dynamischer Modellklassenwechsel bzw. ein gleichzeitiges Bearbeiten von Datenbasen mit unterschiedlichen Schemata möglich ist (Forderungen an ein CAD-System gemäß Kapitel 1). Die bereitstehenden Operationen bilden drei Gruppen:

- Zugriff auf Interessenbereiche (Netze, Teilnetze),
- strukturiertes Modellieren und
- Grundoperationen des Modellierens.

Die Bedeutung der einzelnen Operationen und ihrer Varianten wird anhand der POL für die Stapel-Programmierung erläutert. Im interaktiven, graphisch unterstützten Betrieb können diese Operationen entweder in gleicher Form (Zeichenketten) oder aber über Funktionstasten bzw. Menüs auf einem Tablett angesprochen werden, wobei dann auch ein Prompting-Mechanismus bereitsteht. Vor allem aber entfällt die Bezeichnung von Objekten: Sie wird durch Zeigen ersetzt. In der folgenden Syntaxnotation kennzeichnen spitze Doppelklammern ("<<...>>") variable Sprachelemente, die bei interaktivem Betrieb eine Zeigeoperation des Anwenders erfordern, im Stapelbetrieb aber durch eine Zeichenkette ersetzt werden müssen. Die vollständige Syntax der GRIMBI-Datenmanipulationssprache (DML) befindet sich im Anhang 2. In den folgenden Kapiteln werden zur Erläuterung immer die betreffenden Syntaxteile der GRIMBI-DML herausgegriffen. Einführend behandelt der nächste Abschnitt die GRIMBI-Systemzustände, deren Kenntnis für ein Verständnis der DML erforderlich ist.

Ergänzend zu den folgenden Erläuterungen gibt die Abb.55 eine Übersicht über die wichtigsten Befehle für das interaktive Modellieren.

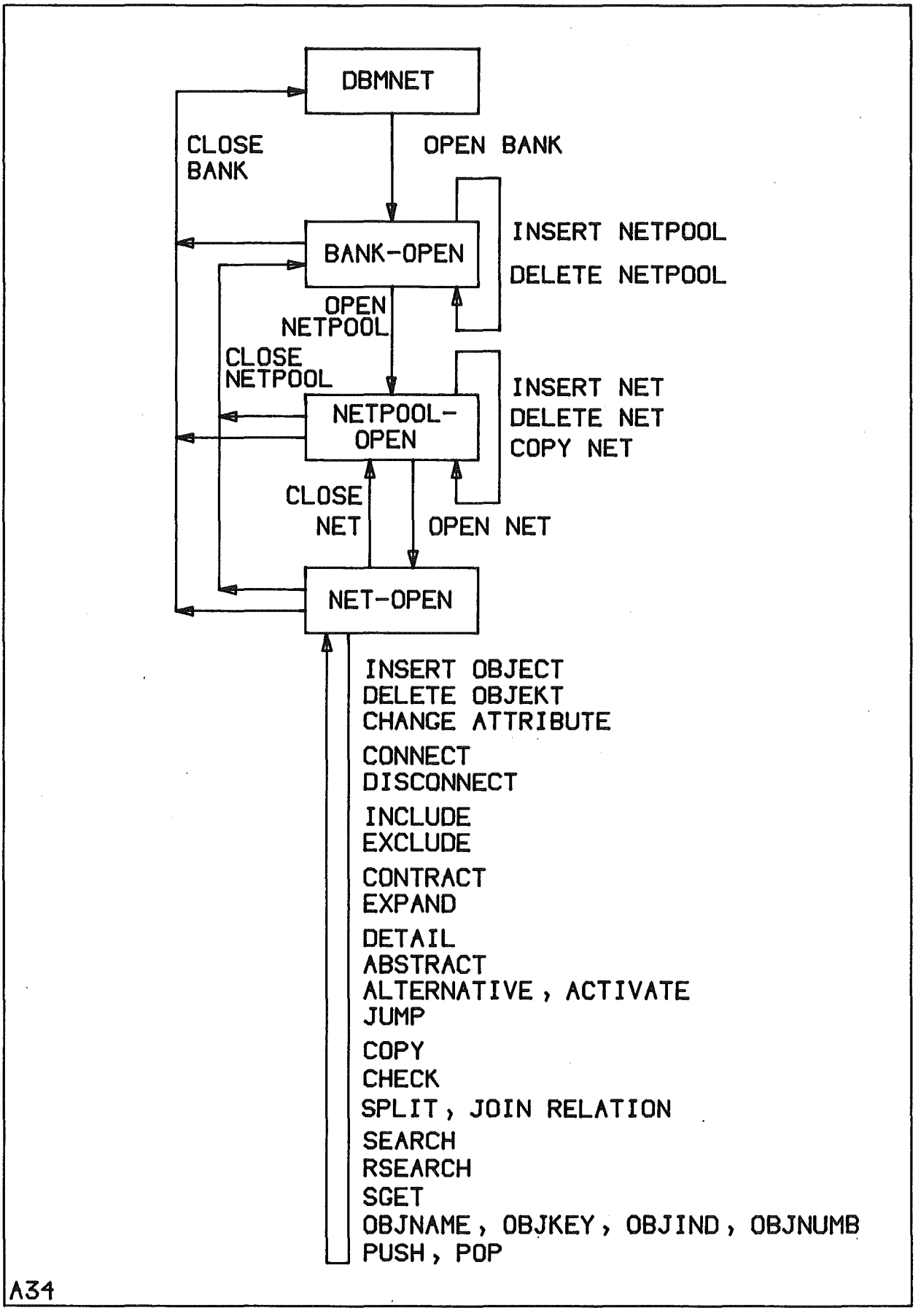


Abb. 37: Systemzustände im Stapelbetrieb

3.2.1. Systemzustände

Betrachtet man die möglichen Systemzustände, so muß man zwei grundverschiedene GRIMBI-Einsatzarten unterscheiden:

- Das Modellieren im Stapelbetrieb mit einer DML in einer Trägersprache (PL/1) ohne Graphik und

- die interaktive Betriebsweise an einem intelligenten graphischen Terminal (TEKTRONIX 4081, Kapitel 3.5.) unter Verwendung einer selbständigen DML.

Die Abb.37 zeigt die Systemzustände für den Stapelbetrieb und die Operationen für die einzelnen Zustandsübergänge. Nacheinander werden die BANK, ein NETPOOL und ein NET oder SUBNET eröffnet, um in den Zustand des eigentlichen problemorientierten Modellierens (Aufbau einer Problemdatenstruktur) zu gelangen. Die Zustände werden durch CLOSE-Befehle verlassen.

Beim interaktiven Arbeiten mit GRIMBI sind einige zusätzliche Zustände zu unterscheiden, die sich vor allem durch den Einsatz eines intelligenten graphischen Terminals (IGT) sinnvoll realisieren lassen (Abb.38). Durch eine geeignete Aufgaben- und Datenverteilung auf das IGT und den Host-Rechner (Kapitel 3.5.) lassen sich neben dem reinen Stapelbetrieb auf dem Host-Rechner ein ONLINE- und ein OFFLINE-Zustand unterscheiden.

Setzt der Anwender sich an ein IGT, so initialisiert er aus dem System-Grundzustand heraus durch IPL GRIMBI das GRIMBI-System. Er kann sich dann entscheiden, ob er

- modellieren (MODELLING-Zustand) will, unter Beachtung aller durch die jeweilige Modellklasse vorgegebenen Restriktionen, für deren Überprüfung der Host- Rechner (Kapitel 3.5.) benötigt wird (ONLINE), oder ob er

- GRIMBI lediglich als Blockdiagramm-Zeichensystem (DRAWING-Zustand) oder zum Editieren (Verändern der Graphik, ohne die Topologie oder Attributwerte zu beeinflussen) und Betrachten vorhandener Blockdiagramme verwenden will (BROWSING- oder DRAWING-Zustand), was ohne Host-Unterstützung möglich ist.

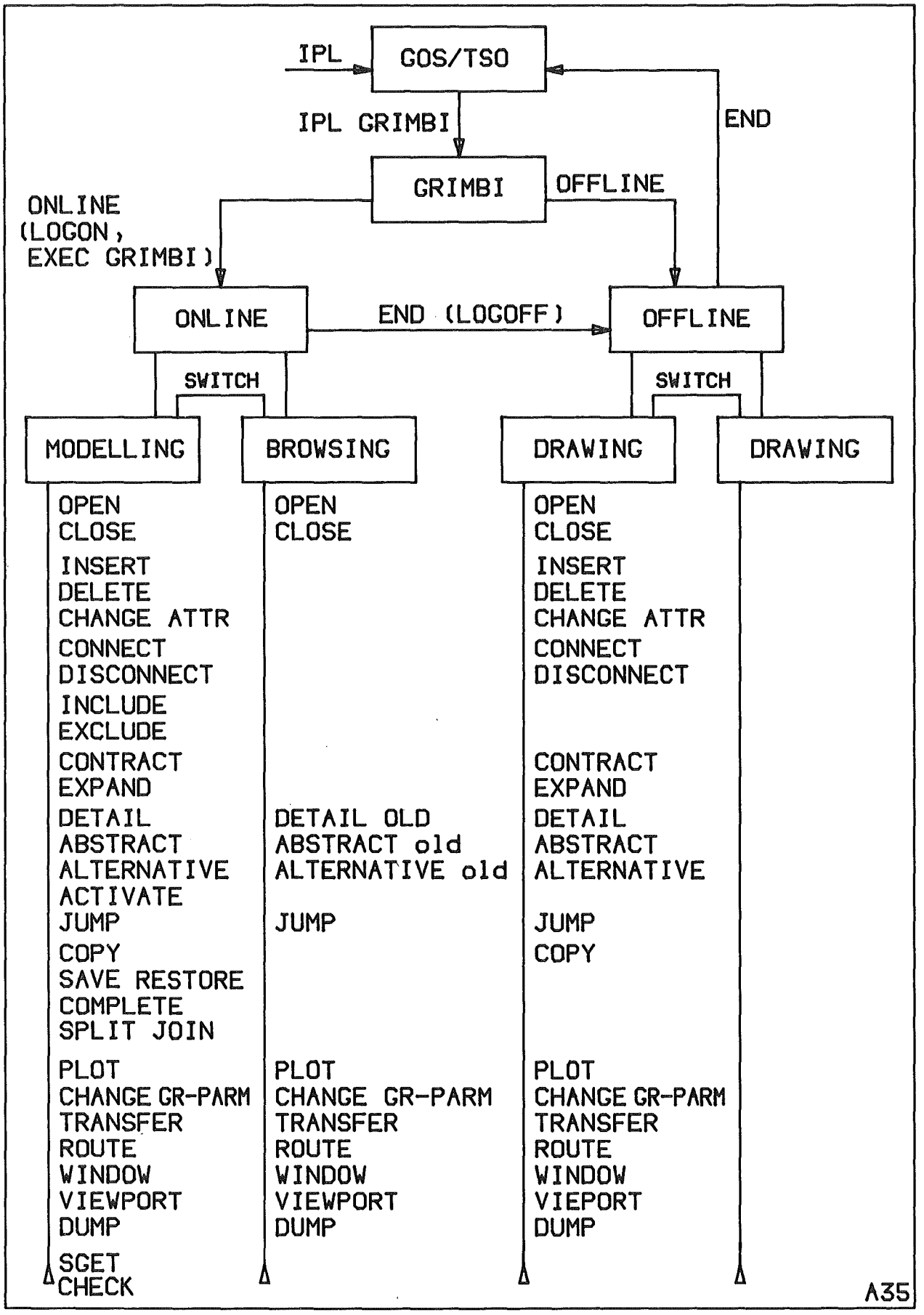


Abb.38: Systemzustände bei interaktivem Betrieb

GRIMBI unterscheidet zwei Typen von Datenbasen:

- Eine MODELLING-BANK enthält Modelle, die den jeweiligen Modellklassen-Restriktionen unterliegen und mit der GRIMBI-DML bearbeitet werden konnen (z.B. durch Analysealgorithmen). Diese GRIMBI-BANKEN enthalten neben den in geeigneter Form abgelegten Problemdaten (auf dem Host-Rechner), graphische Informationen zur Darstellung der Modelle (auf dem IGT).

- Modelle einer DRAWING-BANK unterliegen lediglich üblichen Blockdiagramm-Restriktionen (Verwenden des Symbolsatzes der jeweiligen Modellklasse, beliebiges Verbinden von Symbolanschlüssen, keine Überprüfung der Attributwerte). Sie können nicht mit Hilfe der GRIMBI-DML bearbeitet, sonder nur graphisch ausgegeben werden. Diese GRIMBI-DATENBANKEN enthalten graphische Informationen. Der DRAWING-Betrieb erleichtert nur die Zeichnungserstellung.

Die GRIMBI-BANKEN dürfen nicht beliebig bearbeitet werden. Eine MODELLING-BANK kann man nur in den Zuständen MODELLING und BROWSING, eine DRAWING-BANK in den Zuständen DRAWING und BROWSING für den jeweils vollständigen Befehlssatz eröffnen. Im DRAWING-Zustand steht nur der BROWSING-Befehlssatz zur Verfügung, wenn man eine MODELLING-BANK bearbeitet. MODELLING-BANKEN lassen sich OFFLINE also entsprechend dem BROWSING nur "betrachten" und graphisch editieren (Kapitel 3.3.).

Im ONLINE-Betrieb ist das HOST-GRIMBI (Kapitel 3.5) aktiv. In diesem Zustand werden zwei Arbeitsumgebungen verwaltet: MODELLING und BROWSING. Es handelt sich um die Realisierung einer "Split-Screen-" bzw. "Split-Environment-" Technik, wie sie vom IBM-SPF /79/ bekannt ist. Der Benutzer kann dabei parallel zwei verschiedene Banken auch mit verschiedenen Schemata bearbeiten, allerdings wird nur in einer modelliert, während die andere lediglich angesehen oder graphisch editiert werden darf. Die BROWSING- Bank dient vor allem der Informationsausgabe, zur Darstellung der Arbeitsumgebung oder des Bearbeitungszustandes beim Modellieren eines Teilsystems. Der Übergang von einem dieser Teilzustände in den anderen erfolgt durch SWITCH. Beiden Zuständen kann man für die Graphik getrennte Schirmbereiche (Viewports) zuweisen, sie können aber auch wechselseitig den Gesamtschirm benutzen.

Auch der ONLINE-Zustand kennt die Split-Environment-Technik, allerdings sind hier für beide GRIMBI-Banken bzw. für verschiedene Netze einer Bank die gleichen Operationen zugelassen. Das Umschalten dient lediglich dazu, lange CLOSE-OPEN-CLOSE-OPEN-Folgen zu vermeiden, die erforderlich wären, wenn nur jeweils ein Netz geöffnet sein könnte, der Anwender aber Informationen aus einem anderen Teilnetz der gleichen oder einer anderen Bank benötigte.

3.2.2. Zugriff auf Interessenbereiche

Nach der Initialisierung (Kapitel 3.6) enthält eine Datenbasis (eine BANK) lediglich eine Modellklassenbeschreibung (Schema). In eine BANK können beim Modellieren beliebig viele Modelle der vorgegebenen Klasse eingefügt werden. Diese Modelle werden NETPOOL genannt, weil sie selbst eine Menge verknüpfter Netze/Teilnetze enthalten (Abb.33). Die Operationen OPEN, CLOSE, ABSTRACT, DETAIL, JUMP und ALTERNATIVE ermöglichen die Auswahl eines Interessenbereiches vom Typ BANK, NETPOOL, NET oder SUBNET (Abb.39). Sie können nacheinander in dieser Reihenfolge angesprochen und für die Bearbeitung bereitgestellt werden.

OPEN BANK(<bankname>) {BANKDDN(<ddn>)} {PASS(<password>};

Diese Operation eröffnet die Bank <bankname>, die sich auf der Datei<ddn> befindet und durch Passwort geschützt ist. Entsprechend dem Passwort wird sie für Lese-, Schreib- oder Update-Operationen eröffnet. Die Bank wird zur aktuellen Bank.

OPEN NETPOOL(<netpool-name>) {PASS(<password>)};

Mit dieser Operation wird der Netpool <netpool-name> zum aktuellen Netpool, auf den sich alle folgenden Operationen beziehen. Wenn er durch ein Passwort geschützt ist, so muß es in der PASS-Klausel angegeben werden.

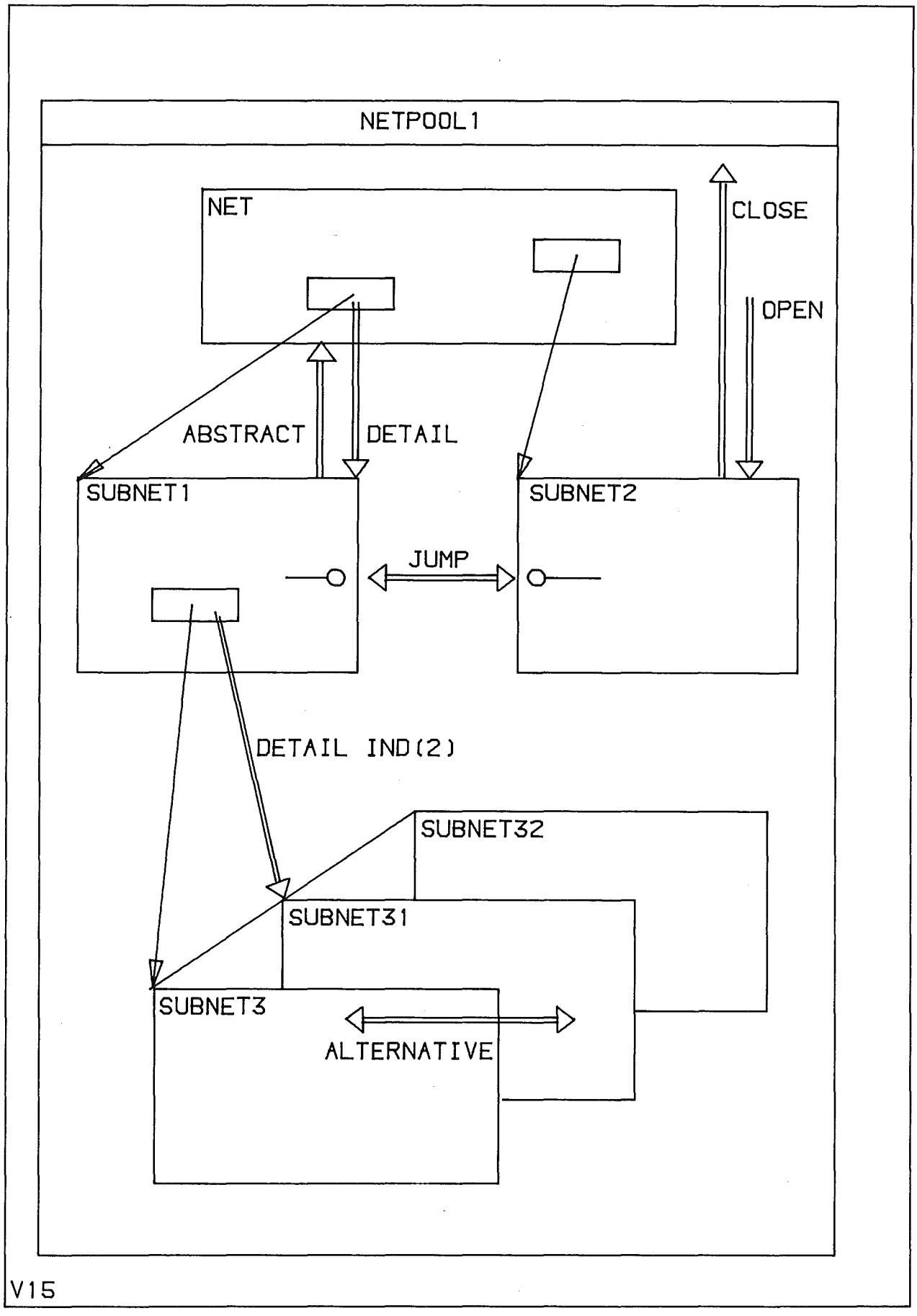


Abb.39: Zugriff auf Interessenbereiche

OPEN NET(<net-or-subnet-ref>);
CLOSE <oject> {LET}; <object> ::= BANK | NETPOOL | NET

Mit OPEN eröffnet man das bezeichnete Netz, das damit zum aktuellen Netz wird. CLOSE schließt die aktuelle Bank, den aktuellen Netpool oder das aktuelle Netz. Dabei werden implizit auch die aktuellen untergeordneten Netpools oder Netze geschlossen. Bei CLOSE NET werden die im Schema festgelegten Restriktionen überprüft, um die Modellkonsistenz sicherzustellen. Dabei besteht die Möglichkeit, durch die LET-Option das Netz/Teilnetz auch dann zu verlassen, wenn es nicht konsistent ist, da bei der Entwurfsarbeit häufig mit Vorentwürfen gearbeitet wird, die noch nicht allen Rstriktionen genügen.

JUMP <<subnetportname>>;

Mit JUMP wird das aktuelle Netz geschlossen und dasjenige Netz zum aktuellen Netz, auf das der mit <<subnetportname>> bezeichnete SUBNETPORT zeigt.

DETAIL <<desciptorname>> {OLD(<ind>) }
ABSTRACT;

DETAIL schließt das aktuelle NET/SUBNET und eröffnet das mit <<descriptorname>> bezeichnete (Abb.40). Liegen Teilnetzalternativen vor, so kann man mit der OLD-Klausel die gewünschte auswählen. ABSTRACT führt den Übergang vom aktuellen Teinetz zum übergeordneten Teilnetz durch, das den zugehörigen Deskriptor enthält.

ALTERNATIVE <which>; <which> ::= IND(<ind>) | NEXT | ACTIVE

realisiert eine CLOSE-OPEN-Sequenz zu der Teilnetzalternative, die durch <ind> bezeichnet wird, zur nächsten Alternative oder zur der Alternative, die als ACTIVE gekennzeichnet ist (Kapitel 3.1.3.). Die ACTIVE Alternative in einer Teilnetzmenge, die zu einem Deskriptor gehört, ist diejenige, die bei der Strukturauswertung bearbeitet wird.

DIRECTORY <dir>; <dir> ::= BANK | NETPOOL {<netpool>}

liefert alle Netpools der aktuellen Bank bzw. alle Netz/Teilnetze eines Netpools.

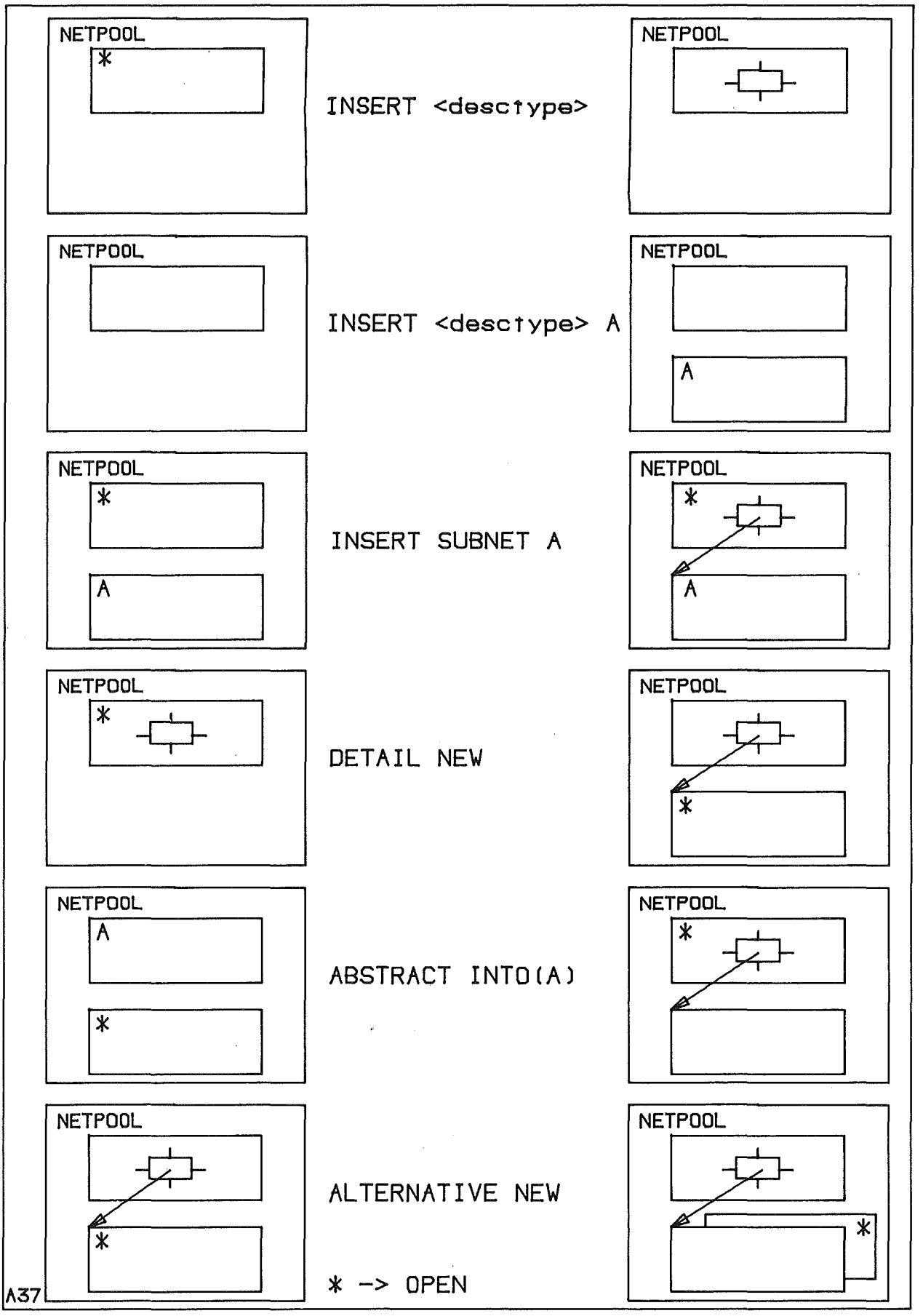


Abb.40: Strukturiertes Modellieren

3.2.3. Strukturiertes Modellieren

Diese Gruppe faßt Operationen zum Einfügen und Löschen von Interessenbereichen, d.h. von Objekten der Arten NETPOOL, NET, SUBNET, zusammen. Es handelt sich zum Teil um die gleichen Befehle, die für diese Aufgabe lediglich durch die NEW-Option ergänzt werden (Abb.41).

```
INSERT <<objname>> <problemtype&>;
 <problemtype&> ::=

      NETPOOL {PASS(<read>,<write>,<update>)} {<property>}0n

    | <net-desc-type>  {<property>}0n
    | SUBNET(<subnet-name>)
 <property>     ::= {<attrn>(<valuelist>)}
```

In die aktuelle Bank (den aktuellen Netpool) wird ein Netpool (Netz) unter dem angegebenen Namen eingefügt. Einem Netpool können drei Schlüsselwörter zugeordnet werden. <net-desc-type> spezifiziert einen Netztyp, d.h. einen als NET oder DESCRIPTOR definierten Anwenderdatentyp (Kapitel 3.1). Handelt es sich um einen DESCRIPTOR-Typ, so generiert das System im Zustand NET-OPEN einen Deskriptor des angegebenen Typs, im Zustand NETPOOL-OPEN (d.h. wenn kein Netz geöffnet ist) ein NET/SUBNET. Ein so erzeugtes Teilnetz ist noch nicht in die Netz/Teilnetz-Hierarchie eingefügt. Diese Vorgehensweise entspricht dem Bottom-up-Entwurf, bei dem zuerst Objekte mit größerer Detaillierung entworfen werden, die man später erst als Teile in eine Struktur einfügt. Dazu verwendet man dann bei geöffnetem Netz den INSERT-SUBNET-Befehl, mit dem ein bereits vorhandenes Teilnetz angegeben wird, das in das aktuelle Netz/Teilnetz in abstrakter Form (als Deskriptor) eingefügt werden soll. Mit der <properties>-Angabe ordnet man den Attributen des erzeugten Objektes Werte zu.

```
DETAIL <<descriptorname>> NEW;
ABSTRACT INTO(<netname>);
```

Detailleren eines Deskriptors mit der Option NEW heißt, das aktuelle Netz/Teilnetz zu schließen, ein neues Teilnetz zu generieren, es mit dem Deskriptor zu verknüpfen und es zu eröffnen. Ist dem Deskriptor bereits ein Teilnetz zugeordnet, so wird das neue als Alternative angesehen. Hat man ein Teilnetz bearbeitet, das noch keinem Deskriptor untergeordnet ist, so kann man durch diesen Befehl im bezeichneten Netz/Teilnetz einen Deskriptor für das aktuelle Netz generieren. Dazu wird das aktuelle Teilnetz geschlossen, das bezeichnete eröffnet und ein entsprechender Deskriptor eingefügt, der die gleichen Anschlüsse und Namen hat.

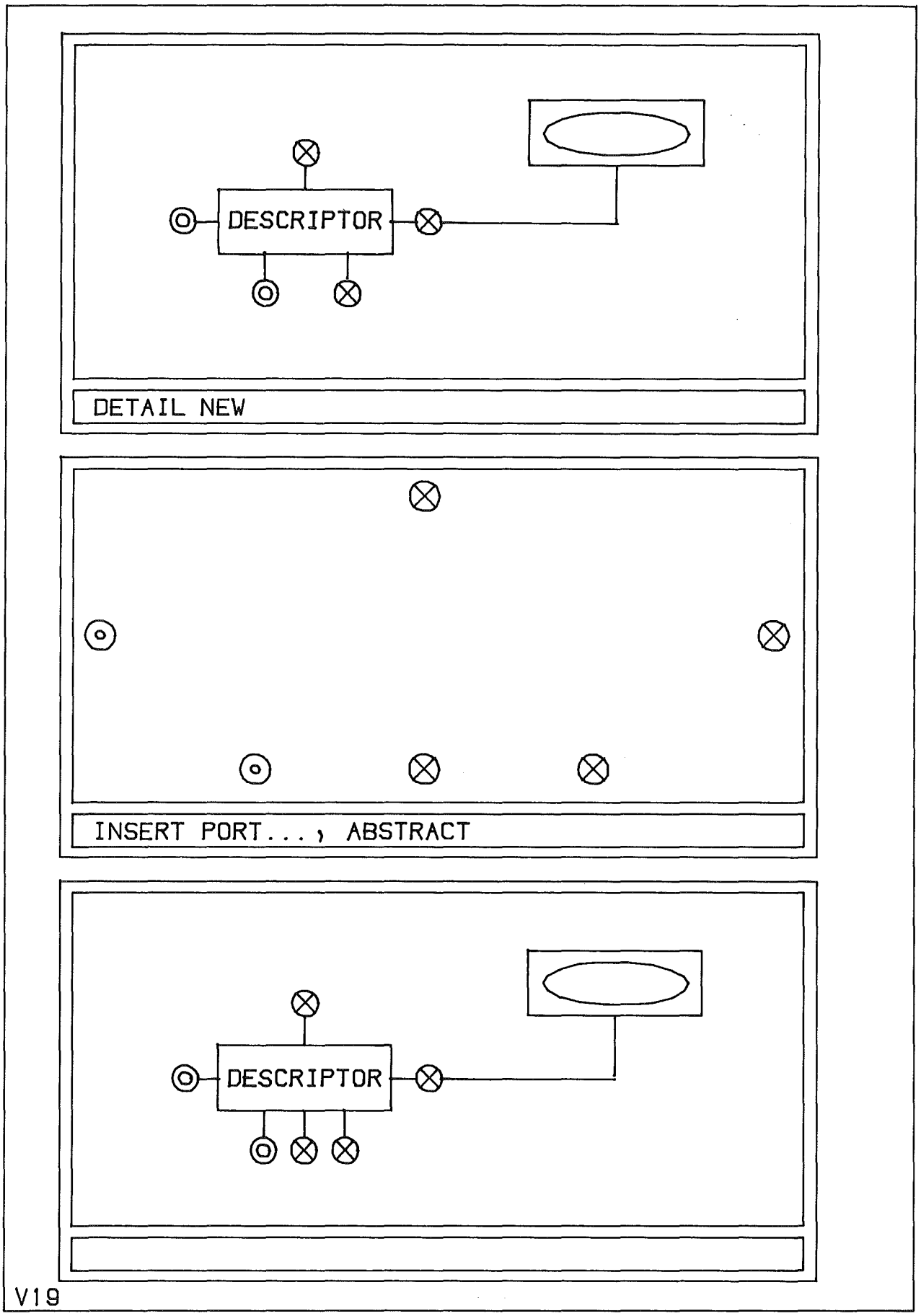


Abb.41: DETAIL-ABSTRACT

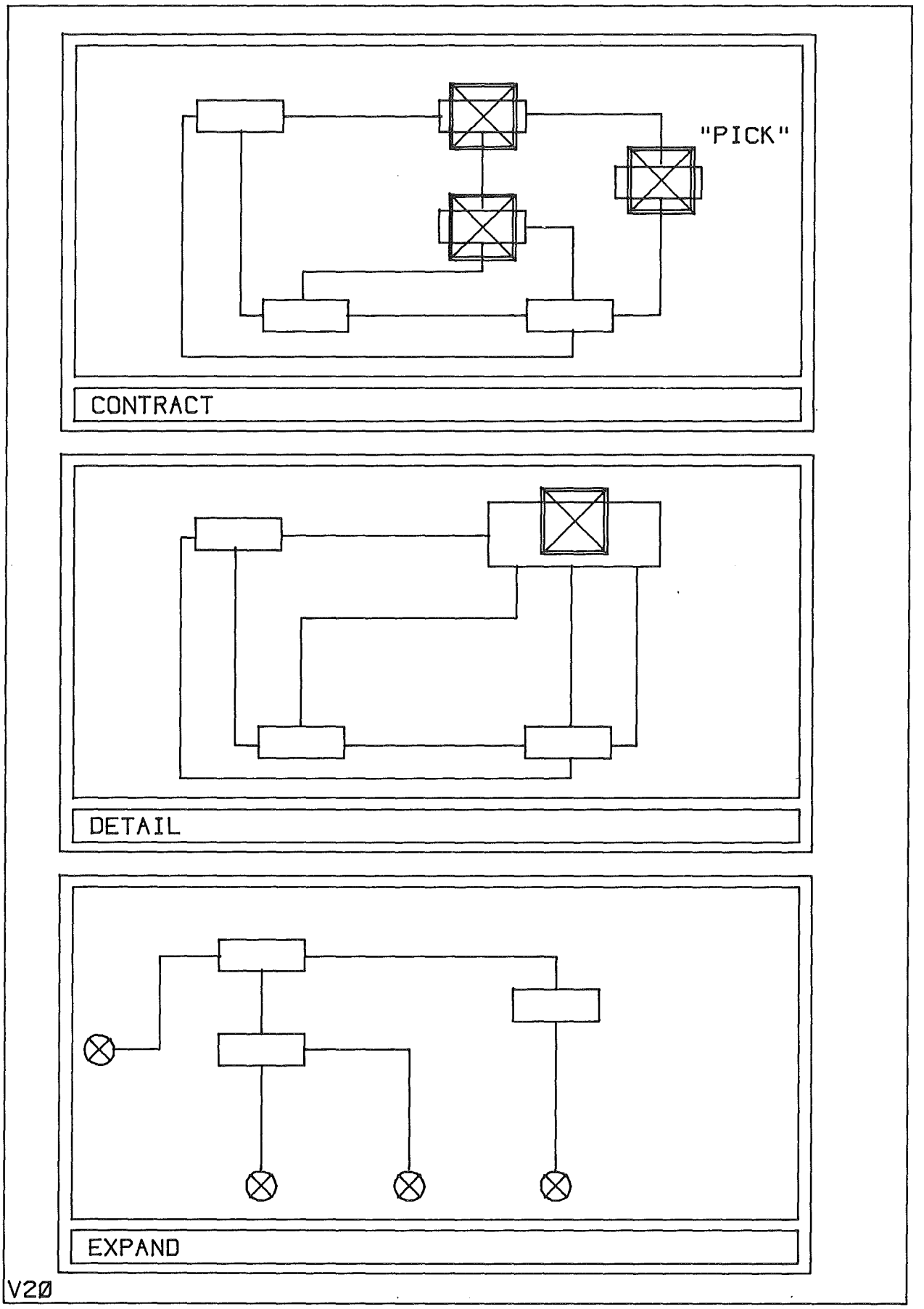


Abb.42: CONTRACT-EXPAND

ALTERNATIVE NEW;

erzeugt zum aktuellen Teilnetz eine Alternative, d.h. ein Teilnetz gleichen Typs, das dem gleichen Deskriptor zugeordnet wird. Das neue Teilnetz wird zum aktuellen.

CONTRACT <<objectlist>> TO(<descriptortype>) {NAME(<descriptorname>)}
EXPAND <<descriptorname>>;

CONTRACT ersetzt im aktuellen Netz/Teilnetz eine Objektgruppe durch einen Deskriptor des Typs <descriptortype> (Abb.42). Die verdrängte Struktur wird in ein zugehöriges, neu zu generierendes Teilnetz eingefügt. EXPAND bewirkt das Gegenteil, indem im aktuellen Teilnetz der bezeichnete Deskriptor durch die zugehörige Teilnetz-Struktur ersetzt wird. Sind Alternativen vorhanden, so wird die aktive gewählt. Die dem Deskriptor zugeordneten Teilnetze werden gelöscht.

DELETE <v-obj> {ALL}; <v-obj> ::= NETPOOL | NET | SUBNET

löscht das aktuelle Netz, Teilnetz oder den aktuellen Netpool, falls kein Netz/Teilnetz geöffnet ist. Wenn ALL angegeben ist, werden auch die untergeordneten Teilnetze gelöscht.

ACTIVATE

macht das aktuelle Teilnetz zum aktiven innerhalb einer Gruppe von Teilnetz-Alternativen.

3.2.4. Grundoperationen des Modellierens

Für die eigentliche Modellbeschreibung stehen die Problemdatentypen der Klassen BASEOBJECT, COMPLEXOBJECT und RELATION bereit. Elemente dieser Typen werden mit den Operationen INSERT/DELETE, CONNECT/DISCONNECT, INCLUDE/EXCLUDE und CHANGE PROPERTY bearbeitet. Sie erzeugen, löschen, verbinden und ändern Objekte der Problemtypen und weisen ihnen Werte zu.

INSERT <<name>> <problemtype> $\{$<attribname(<valuelist>)$\}_0^n$;
DELETE <<name>> {ALL};
INSERT <<name>> <subnetport-type> DESCRIPTOR <<desc-name>>

$\{$attribname(<valuelist>$\}_0^n$;

INSERT erzeugt ein Exemplar des Anwenderdatentyps <problemtype> mit dem Namen <name> , weist ihm Werte zu und fügt es in das aktuelle Netz/Teilnetz ein. DELETE löscht das Objekt mit Namen <name> im aktuellen Netz. Die Option ALL bewirkt, daß alle diesem Objekt untergeordneten Objekte (INCLUDE) oder untergeordnete Teilnetze ebenfalls gelöscht werden, genauso wie alle Verbindungen, die diese Objekte enthalten. Löscht man eine Relation, bedeutet dies, daß alle zu dieser Relation gehörenden Verbindungen auch gelöscht werden. Der zweite INSERT-Befehl fügt einen Anschluß des Typs <subnetport-type> in einen Deskriptor ein.

CONNECT <portname1> <portname2> REL(<relname> {<tupelvalue>})
bei interaktivem Betrieb:
CONNECT <<portname1>> <<portname2>> <<relname>> {<tupelvalue>}

Diese Operation verbindet zwei existierende Objekte des aktuellen Netzes über die beiden genannten Anschlüsse. Die Verbindung (das Tupel) wird der Relation <relname> zugeordnet, sie wird durch den Wert <tupelvalue> charakterisiert. Das Objekt mit dem Anschluß <portname> muß ein für diese Relation gültiger Vorbereichstyp (FROMDOMAINE) sein, der Typ des Objektes, zu dem der zweite Anschluß gehört, muß ein gültiger Nachbereichstyp (TODOMAINE) sein. Zwei Anschlüsse werden nur verbunden, wenn ihre Typen einander nicht widersprechen (IN-OUT, IN-INOUT, INOUT-INOUT, OUT-INOUT). Darüberhinaus wird überprüft, ob die Restriktionen eingehalten werden und ob der Relations-Typ durch die neue Verbindung erhalten bleibt (d.h. ob es beispielsweise ein Baum bleibt).

DISCONNECT <connection>; <connection> ::= <portname1> {<portname2>}
| <<line>> | <<relation-name>>

löscht eine Verbindung aus dem aktuellen Netz. Eine Verbindung wird eindeutig durch zwei Anschlüsse oder durch Zeigen auf die Verbindungs-Linie (nur im interaktiven Betrieb) identifiziert. Bei Angabe nur eines Anschlusses können Mehrdeutigkeiten auftreten (FAN>1), in diesem Falle werden alle Verbindungen, die von dem Anschluß ausgehen, gelöscht. Wird ein Relations-Objekt bezeichnet, so sind damit alle zu dieser Relation gehörigen Verbindungen gemeint.

INCLUDE <<son_name>> <<father_name>>;

Diese Operation fügt das mit <son_name> bezeichnete Objekt in den System - Objektbaum (System-Menge) unterhalb des mit <father_name> bezeichneten Objektes ein. Dabei wird überprüft, ob gültige Typen aufgrund der Angaben in den SUB- bzw. SUPERDOMAINE - Klauseln verknüpft werden.

EXCLUDE <<name>> {ALL};

löst das Objekt <name> aus dem System - Objektbaum (System-Menge). Mit der ALL - Option wird erreicht, daß ein eventuell unterhalb von diesem Objekt vorhandener Teilbaum ebenfalls aufgelöst wird, der ohne diese Option erhalten bliebe. Das Objekt <name> würde dann eine Baumwurzel bilden.

CHANGE PROPERTY <<name>> $\{$<attribname>(<valuelist>)$\}_1^n$;

Mit CHANGE PROPERTY werden Attributwerte des Objekts <name> geändert, indem dem Attribut <attribname> die Werte <valuelist> zugewiesen werden.

SGET <<name>> <attribname>;

Im interaktiven Betrieb liefert SGET zu einem identifizierten Objekt den Wert des angegebenen Attributes in einem Standardformat.

```
COPY <c-opt>; <c-opt> ::= WITHIN <<obj-list>> |
                          INTO   <newnetname> <<obj-list>> |
                          FROM   <fromnetname>
```

Der COPY-Befehl erleichtert den Aufbau von Modellen mit gleichen oder ähnlichen Teilstrukturen. COPY WITHIN kopiert innerhalb des aktuellen Netzes die durch <obj-list> bezeichnete Objekt-Gruppe einschließlich der inneren Verbindungen. Mit COPY INTO wird eine entsprechende Objekt-Gruppe in ein neues Netz/Teilnetz, das implizit mit dem Namen <newnetname> in den aktuellen Netpool eingefügt wird, kopiert. Dieses neue Netz/Teilnetz ist vom gleichen Typ wie das aktuelle. Um in das aktuelle Netz/Teilnetz ein anderes Netz/Teilnetz (gleichen Typs) einzukopieren, steht der Befehl COPY FROM zur Verfügung.

COPY generiert neben den bezeichneten Elementen auch die zwischen ihnen bestehenden Verbindungen (Tupel der zugehörigen Relationen) und Relations-Exemplare. Um zu verhindern, daß mit COPY-WITHIN- oder COPY-FROM-Befehlen im aktuellen Netz neue Relationen entstehen, kann mit dem JOIN-Befehl eine Relation mit einer anderen gleichen Typs verschmolzen werden.

```
JOIN <<relname1>> WITH <<relname2>>;
```

Dieser Befehl ordnet die Tupel der mit <<relname1>> bezeichneten Relation der zweiten bezeichneten Relation zu und löscht die erste. Beide Relationen müssen vom gleichen Typ sein.

Die gegenteilige Operation, nämlich die Aufteilung der Tupel (Verbindungen) einer Relation auf zwei Relationen gleichen Typs, bewirkt der Befehl SPLIT.

```
SPLIT  <<tupellist>>;                    /*bei interaktivem Betrieb*/
SPLIT   <tupellist> TO <newrelname>;
```

Die mit <tupellist> bezeichnete Gruppe von Verbindungen (Tupelgruppe) bildet nach SPLIT eine neue Relation (im Stapelbetrieb mit dem Namen <newrelname>). Diese neue Relation wird implizit generiert und die ihr zugeordneten Tupel werden in der alten Relation gelöscht. Alle Verbindungen der <tupellist> müssen vom gleichen Typ sein. Verbindungen, die nicht vom Typ der ersten bezeichneten Verbindung sind, werden ignoriert.

3.2.5. Komplexobjekt-Definition

Um beim Modellieren auch Komplextypen definieren zu können, läßt GRIMBI beim Modellieren eine Schema-Erweiterung zu, eine Definition von Komplexobjekten auf der Basis der vorhandenen Problemtypen. Die erforderlichen Befehle entsprechen denen der GRIMBI-DDL (Kapitel 3.1.: DESCRIBE COMPLEX ergänzt um die Komplex-Graphik). Korrekturen führt man mit DELETE COMPLEX <name> und einer Neudefinition aus.

3.2.6. Graphische Unterstützung

Arbeitet man mit dem interaktiven graphischen Terminal (IGT), so werden diese Operationen graphisch unterstützt, indem die Operanden (Objekte, Anschlüsse, Attribute) durch Zeigen mit einem graphischen Eingabegerät identifiziert und gemäß der Modellklassenbeschrei- bung dargestellt werden. Bei den Befehlen INSERT und CONNECT wird der Benutzer aufgefordert, Objektsymbole auf dem Bildschirm innerhalb einer Zeichenfläche zu positionieren und Verbindungslinien, für die das System einen Verlauf vorschlägt, eventuell (in einem speziellen Rubberbanding-Verfahren) umzugestalten.

Der Systemvorschlag für eine Verbindung enthält zwischen den beiden Anschlüssen einen Knickpunk, so daß die Verbindung aus einem waagerechten und einem senkrechten Abschnitt besteht. Dieser Knickpunkt kann nun verschoben und durch eine Folge neuer Knickpunkte ergänzt werden. Die Verbindung wird so, ausgehend vom ersten Anschluß schrittweise aus horizontalen und vertikalen Teilstücken aufgebaut. Wird dabei eine bestehende Verbindung erreicht, die in den gleichen Anschluß mündet, wie die aktuell zu bearbeitende, so kann man sie hier in die erreichte Verbindung einmünden lassen. Die Einmündung wird durch einen Punkt markiert.

Wurde ein Tupelattribut definiert, so muß der Anwender auch zu jeder Verbindung die Position des Linientextes angeben.

Als Symbolpositionen werden nur Rasterpunkte akzeptiert, deren Abstand schemaspezifisch ist und mit der Modellklassendefinition festgelegt wurde.

Soviel zu der graphischen Ünterstützung der Befehle zum Strukturaufbau. Darüberhinaus stehen graphische Operationen bereit, die dazu dienen, Blockdiagramme, d.h. Zeichnungen zu verbessern, um ihre Lesbarkeit zu erhöhen, ohne allerdings den logischen Inhalt zu verändern. Der folgende Abschnitt faßt diese Operationen zum Editieren von Blockdiagrammen zusammen.

Sind einem Textfenster durch die CLASS-Klausel mehrere Attribute zugeordnet, so wird mit dem Befehl CHANGE PROPERTYLEVEL(<p-level>) das <p-level>-te Attribut für die graphische Darstellung ausgewählt.

SAVE überträgt das aktuelle Netz auf den Host-Rechner und ordnet es den Problemdaten zu. Diese Kopie der Graphik kann mit RESTORE wieder zurückübertragen, auf dem Host-Rechner mit dem GRIMBI-DML- Befehl PLOT (DBMNET-Stapel-Programm) gezeichnet oder in ein AGF-PLOTFILE-Format (Kapitel 3.5.5.) für eine Weiterbearbeitung umgewandelt werden.

COMPLETE dient der Bearbeitung von Banken, die im Stapel-Betrieb ohne Graphik erstellt wurden. Die Operation bezieht sich auf ein Netz/Teilnetz und fordert den Benutzer auf, alle Objekte nacheinander auf dem Bildschirm zu positionieren und die Verbindungen zu verlegen, d.h. Leitungsknickpunkte anzugeben und den Leitungstext zu positionieren. Es sind vom Benutzer also lediglich graphische Editier-Funktionen zu erfüllen. Nach Abschluß von COMPLETE kann ein solches Netz/Teilnetz wie ein interaktiv erstelltes bearbeitet werden.

Entsprechend den Möglichenkeiten der verwendeten graphischen Hardware, einem Speicherbidschirm mit beschränkten Refresh-Fähigkeiten, werden die jeweils zu positionierenden Symbole, Symbolgruppen und Verbindungen im Refresh-Modus dargestellt, so daß sie verschoben werden können und ihre aktuelle Position oder Gestalt immer sichtbar ist.

3.3. Editieren

Diese Gruppe von Operationen bezieht sich allein auf die graphische Darstellung der Modelle, sie läßt keine Veränderungen der Netzwerktopologie oder anderer logischer Inhalte zu. Bedingt durch die GRIMBI-Systemarchitektur (Kapitel 3.5.), erfordern diese Operationen auch im ONLINE-Zustand keine HOST-Interaktionen.

Grundsätzlich ist jedem Interessenbereich (Netz,Teilnetz) eine Zeichnung (Blockdiagramm) zugeordnet, die beim OPEN vollständig auf dem Bildschirm abgebildet wird. Die Zeichenflächen besitzen ein DIN-Seitenverhältnis und können im Hoch- oder Querformat verwendet werden. Das System kennt eine Standard-Blattgröße und verwendet als Standardlage das Querformat, falls für ein Netz/Teilnetz keine expliziten Änderungen spezifiziert werden. Blattgröße und -lage können jederzeit verändert werden, allerdings nur in DIN-Schritten.

Für ein neues Netz/Teilnetz spezifiziert man mit dem Befehl

CHANGE SYMBOLSIZE(<symsize>)

wieviel Symbolbreiten des breitesten Symbols einer Standardblatt-Breite entsprechen sollen (Standardwert <symsize>=10).

CHANGE SHEETFORMAT

ändert ein Hoch- in ein Querformat und umgekehrt.

CHANGE SHEETSIZE<size> <size> ::= UP | DOWN | EXTENT

wählt das nächst größere, kleinere oder notwendige DIN-Format als Blattgröße (virtuelle Zeichnung) wobei der Blattmittelpunkt erhalten bleibt. Mit

CHANGE SHEETPOSITION <<newpos>>

wird ein neuer Blattmittelpunkt angegeben.

Für die Spezifikation der Abbildung einer internen (virtuellen) Zeichnung auf eine Zeichenfläche eines Ausgabegerätes verwendet man die Befehle WINDOW und VIEWPORT. Dabei wird immer eine verzerrungsfreie Darstellung gewährleistet.

WINDOW <wopt>

<wopt> ::= FACTOR <factor> | SHIFT <<w-pos>> | ALL | <<w-ll>> <<w-ur>>

Verwendet man die FACTOR-Option so bleibt die Lage des Fensters auf der virtuellen Zeichenfläche erhalten, lediglich seine Größe wird um den angegebenen Faktor verändert. Die Fensterlage wird durch WINDOW SHIFT variiert, wobei der Benutzer die neue Fenstermitte angeben muß. Bei Verwendung der ALL-Option erhält das Fenster die Größe des virtuellen Blattes. Bei dem Befehl WINDOW ohne Optionen muß der Anwender für ein neues Fenster die linke untere und die rechte obere Ecke eingeben.

VIEWPORT <vopt>

<vopt> ::= RESET | <<v-ll>> <<v-ur>>

Dieser Befehl spezifiziert den Bildschirmbereich, auf den das Fenster abgebildet werden soll. Dazu gibt man entweder die linke untere und die rechte obere Ecke des Bereiches an oder man bewirkt mit RESET, die Verwendung der maximal zur Verfügung stehenden Bildschirmfläche. Das System sorgt dafür, daß sowohl das WINDOW als auch der VIEWPORT gleiches DIN-Seitenverhältnis haben.

PLOT <unit> <unit> ::= DISPLAY | PLOTTER <dx> | PDBFILE <pdbfile>

Mit dem PLOT-Befehl wird das aktuelle Netz/Teilnetz gezeichnet, und zwar im Standardfall auf dem Bildschirm (VIEWPORT), ansonsten auf einem anderen Zeichengerät (unter Angabe der gewünschten Bildbreite in Metern). Wird PDBFILE spezifiziert, so das Bild im TEKTRONIX-PDB-Format /80/ auf die Datei <pdbfile> geschrieben, so daß sie anschließend mit dem GFM (Graphic Function Manager /16/) weiter bearbeitet werden kann.

Der PLOT-Befehl kennt eine Reihe von Parametern, die mit CHANGE PLOTTYPE verändert werden.

CHANGE PLOTTYPE $\{<\text{pparms}>\}_1^n$;

<pparms> ::= CONNECTIONS {(<relationtype-list>)} | NOCONNECTIONS
| SYMBOLS {(<objtype-list)>} | NOSYMBOLS
| ATTRIBS | NOATTRIBS
| TEXTS {(<text-window-number-list>)} | NOTEXTS
| NORELATIONS | NONETS | ALL

CONNECTIONS bewirkt, daß entweder alle Verbindungen oder nur die angegebenen Typen gezeichnet werden, ensprechendes gilt für SYMBOLS. Es können aber sowohl die Symbole als auch die Verbindungen insgesamt unterdrückt werden. ATTRIBS/NOATTRIBS steuert die Darstellung gewisser Objektwerte durch graphische Zusatzsymbole (Kapitel 3.1.4.). Will man alle zu einem Symbol gehörenden Texte darstellen oder lediglich eine Auswahl, so verwendet man die TEXTS-Option, NOTEXTS unterdrückt alle Texte. Darüberhinaus können die Relationssymbole und die Netz/Teilnetz-Symbole (Schriftfeld) unterdrückt werden.

Um alle Symbole mit ihren Typbezeichnungen darzustellen, steht der

DUMP SYMLIB $\{$<symparm>$\}_0^4$
<symparm> ::= PICKAREAS | TEXTWINDOWS | ATTRGRAPHIC | PORTCOORDS

Befehl zur Verfügung. Mit den Optionen spezifiziert man, um welche Teilinformation die Grunddarstellung ergänzt werden soll.

TRANSFER $\{$<<symbol-list>>$\}_1^n$
ROUTE <<connection>>

Um die graphische Darstellung eines Netzes/Teilnetzes zu verbessern, ohne die Topologie oder andere logische Informationen zu ändern, verwendet man diese beiden Befehle. TRANSFER verschiebt ein Symbol oder eine Symbolgruppe und zwingt den Anwender, nacheinander alle Verbindungen, die mit dem Symbol oder der Symbolgruppe verknüpft sind (und nicht nur innerhalb einer Gruppe verlaufen), zu ändern oder den Symstemvorschlag für ihren Verlauf zu akzeptieren. Mit ROUTE ändert man den Verlauf einer einzelnen Verbindung, indem man nach dem Identifizieren den Systemvorschlag (Verbindung mit einem Knickpunkt) annimmt oder durch beliebige Knickpunkte ergänzt. Dabei ist es auch möglich, die Verbindungsgraphik (den "Strich") bei Erreichen einer bereits bestehenden Verbindung enden zu lassen, wenn sie zum gleichen Anschluß führt. Die Einmündung wird durch einen Punkt kenntlich gemacht, um Kreuzungen von zweiseitigen Einmündungen unterscheiden zu können.

3.4. Modellanalyse

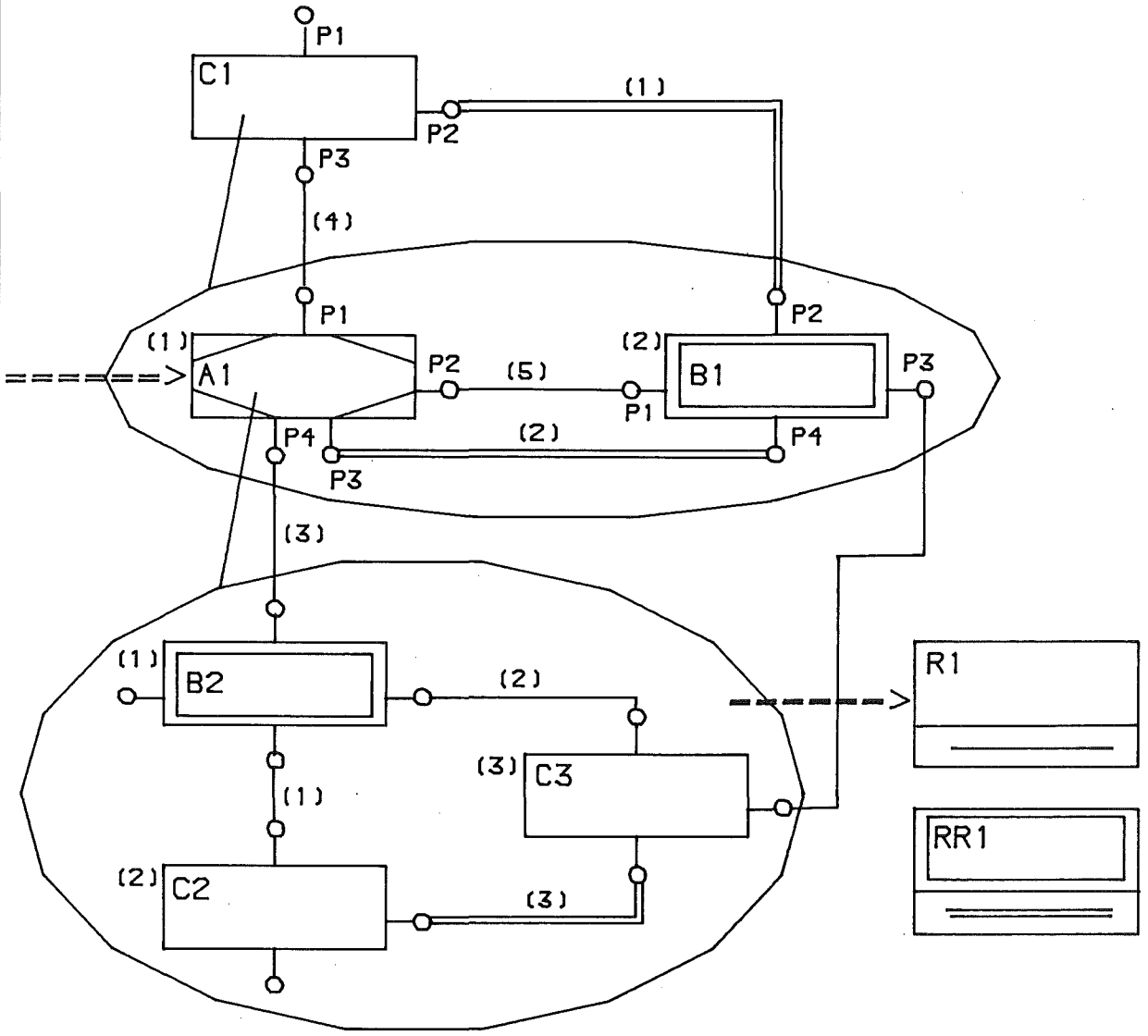
Zur Unterstützung der Auswertung eines Modells enthält die GRIMBI-DML Operationen, die den Zugriff auf die Daten erleichtern und eine weitgehende Datenunabhängigkeit der Programme garantieren. Auf die Daten wird navigierend, d.h. objektweis zugegriffen, eine Methode die bei der Bearbeitung von Datennetzen angemessen erscheint und und für den Entwurfsingenieur, der es gewohnt ist, mit Graphen umzugehen, naheliegt. Für die eigentliche Datenauswahl verwendet man die Sprach-Konstrukte der Gast-Sprache PL/1, beispielsweise den IF...THEN...ELSE...-Befehl. Zur Unterstützung des Navigierens durch die Datenstruktur führt das System eine aktuelle Stelle (Abb.43), eine Gruppe von Daten, die den aktuellen Standpunkt im Netz beschreibt, auf den sich Folgeoperationen beziehen können. Eine aktuelle Stelle gliedert sich, angepaßt an das verwendete Datenmodell, in die Beschreibung eines aktuellen Objektes (beliebigen Typs) und einer aktuellen Relation, die unabhängig voneinander sind. Stellen können gekellert werden.

SEARCH

Diese Operation unterscheidet zwei Klassen von Optionen: den direkten Zugriff mit Hilfe des Objektnamens oder des Systemschlüssels (der mit einem der im folgenden beschriebenen OBJ....-Befehle ermittelt werden kann und schnelleren Zugriff erlaubt) und den relativen Zugriff auf Objekte, ausgehend vom aktuellen Objekt oder der aktuellen Relation.

```
SEARCH <s-opt> {PUSH} {FOUND(<found>)};

      <s-opt>  ::= <direct> | <relative>
      <direct> ::= NAME(<name>)              |
                   KEY(<key>)                |
                   TYPE(<problemtype>,<n>)
    <relative> ::= SON(<ind>)                |
                   FATHER                    |
                   BROTHER(<port>) {DETAIL}|
                   RELATION(<port>)          |
                   FROMOBJ TUPEL(<ind>)      |
                   TOOBJ   TUPEL(<ind>)
```



BEFEHL	NEUE AKTUELLE STELLE
SEARCH NAME(C1)	(C1,R1)
SEARCH FATHER	(C1,R1)
SEARCH BROTHER PORT(P2)	(B1,R1)
SEARCH RELATION PORT(P3)	(C1,RR1)
SEARCH FROMOBJ TUPEL(2)	(B2,R1)
SEARCH TOOBJ TUPEL(1)	(C2,R1)
SEARCH SON INDEX(3)	(C3,R1)
RSEARCH NAME(RR1)	(A1,RR1)

A4Ø

Abb.43: Aktuelle Stelle

Bei der direkten Suche wird das Objekt mit dem Namen <name>, mit dem Systemschlüssel <key> oder das <n>-te Objekt des angegebenen Typs zum aktuellen Objekt. Nach der Aktualisierung eines Netzes muß eine SEARCH - DIRECT - Operation durchgeführt worden sein, bevor SEARCH - RELATIVE - Operationen zulässig sind. SEARCH - RELATIVE geht von dem aktuellen Objekt aus, sucht dessen Sohn, Vater, Bruder oder eine Relation, deren Vor- oder Nachbereich das aktuelle Objekt angehört. Die Auswahl des Sohnes erfolgt durch seinen Index, die des Bruders oder der Relation durch Angabe des entsprechenden Objektanschlusses. Die Optionen FROMOBJ bzw. TOOBJ aktualisieren das Objekt, das dem Vor- oder Nachbereich der aktuellen Relation angehört und dem durch <ind> bezeichneten Tupel der Relation.

Gibt man bei SEARCH-BROTHER die DETAIL-Option an, so wird, falls man über den angegebenen Anschluß ein DESCRIPTOR- oder COMPLEX-Objekt erreicht, nicht dieses selbst, sondern das entsprechende Strukturelement zum aktuellen Objekt. Handelt es sich dabei um ein COMPLEX-Objekt-Element, so werden ihm die aktuellen Werte zugewiesen, die das COMPLEX-Objekt möglicherweise enthält.

RSEARCH

Die Operation R(elation)-Search verändert den Relationen-Teil der aktuellen Stelle. Diese Zweiteilung der Stellenbeschreibung ermöglicht eine einfachere, bequemere Programmierung von Suchalgorithmen. Die beiden Stellen - Komponenten sind unabhängig voneinander, sie werden jeweils nur von "eigenen" Operationen verändert.

```
RSEARCH <r-opt> {PUSH} {FOUND(<found>)};

        <r-opt> ::= NAME(<name>)          |
                    KEY (<key>)           |
                    TYPE(<problemtype>)   |
                    PORT(<port>)
        <found> ::= BIT(1)-variable
```

Es wird die Relation mit Namen <name> bzw. Schlüssel <key> oder die nächste vom Typ <problemtype> zur aktuellen Relation gemacht (direkter Zugriff). Mit der Option PORT wird, ausgehend vom aktuellen Objekt, diejenige Relation zur aktuellen erhoben, auf die der Anschluß <port> des aktuellen Objektes zeigt. PUSH bewirkt , daß die alte aktuelle Stelle nicht überschrieben, sondern gekellert wird. FOUND setzt die Variable <found> je nach Suchergebnis auf '1'B oder '0'B.

PUSH POP DUMP-POSSTACK

Diese Operationen kellern bzw. entkellern die aktuelle Stelle. DUMP POSSTACK druckt den Stellenkeller.

Zugriff auf Objekt-Verwaltungsinformation

Zur Unterstützung der Navigation und der Auswertung stehen einige Operationen bereit, die Verwaltungsinformationen zu einzelnen Objekten bzw. Objekttypen liefern (ändern die aktuelle Stelle nicht).

```
OBJNAME(<name>) {KEY (<key-var>)}
                {SYSTYPE(<systype-var>)}
                {PROBLEMTYPE(<problemtype-var>)};
OBJKEY (<key>)  {NAME(<name-var>)}
                {SYSTYPE(<systype-var>)}
                {PROBLEMTYPE(<problemtype-var>)};
OBJIND (<ind>)  {NAME(<name-var>)}
                {KEY(<key-var>)}
                {SYSTYPE(<systype-var>)}
                {PROBLEMTYPE(<problemtype-var>)};
OBJNUMB <of>    TO(<var>);

        <of> ::= ALL | SYSTYPE(<systype> | PROBLEMTYPE(<problemtype>)
```

Wie die Syntax zeigt, handelt es sich hier um die Objektinformationen Schlüssel, Name, Systemtype (BASEOBJ, RELATION, SYSPORT, SNPORT, COMPLEX, NET, SUBNET) und Anwendertyp (Problemtyp). Sie werden aufgrund des Namens, des Schlüssels und des Indexes für das aktuelle Netz/Teilnetz ermittelt. OBJNUMB liefert die Gesamtzahl der Objekte in einem Netz oder die Zahl der Objekte eines Typs.

Ergänzend zu diesen Funktionen kennt die GRIMBI-DML eine Datenstruktur, die alle Daten zur Kennzeichnung einer Stelle umfaßt. Bei der Programmierung eines Analysealgorithmus kann auf diese Werte zugegriffen werden. Ihre detaillierte Beschreibung bleibt einem GRIMBI-Anwenderhandbuch vorbehalten.

Zugriff auf Problemdaten

Die SGET-Operation liefert einen bestimmten Attributwert des aktuellen Objektes.

SGET({<portname>.}<attrname>{(<ind1>{,<ind2>})} TO(<var>);

Der Wert des bezeichneten Objektattributes wird der Variablen <var> zugewiesen.

Architektur von Auswertemoduln

Bei der Auswertung einer Datenbasis sind zwei Fälle zu unterscheiden (Abb.44):

- Sind die Analysealgorithmen nicht als REGENT-Subsystem implementiert, so wird ein DBMNET-Programm in der GRIMBI-DML erstellt, das die Modelle auswertet und die Ergebnisse in einer Form auf einem Zwischenspeicher ablegt, die für die Analyse vorgeschrieben ist. Ein zweites DBMNET-Programm kann die Analyseergebnisse, falls gewüscht, in die Datenbasis einfügen.

- Liegt dagegen das Analysesystem als REGENT-Subsystem vor, so wird man dieses Subsystem durch Moduln ergänzen, die mit Hilfe der GRIMBI-DML direkt auf die Datenbasis zugreifen. Das ist möglich, da in REGENT Moduln eines Subsystems die POL anderer Subsysteme enthalten dürfen.

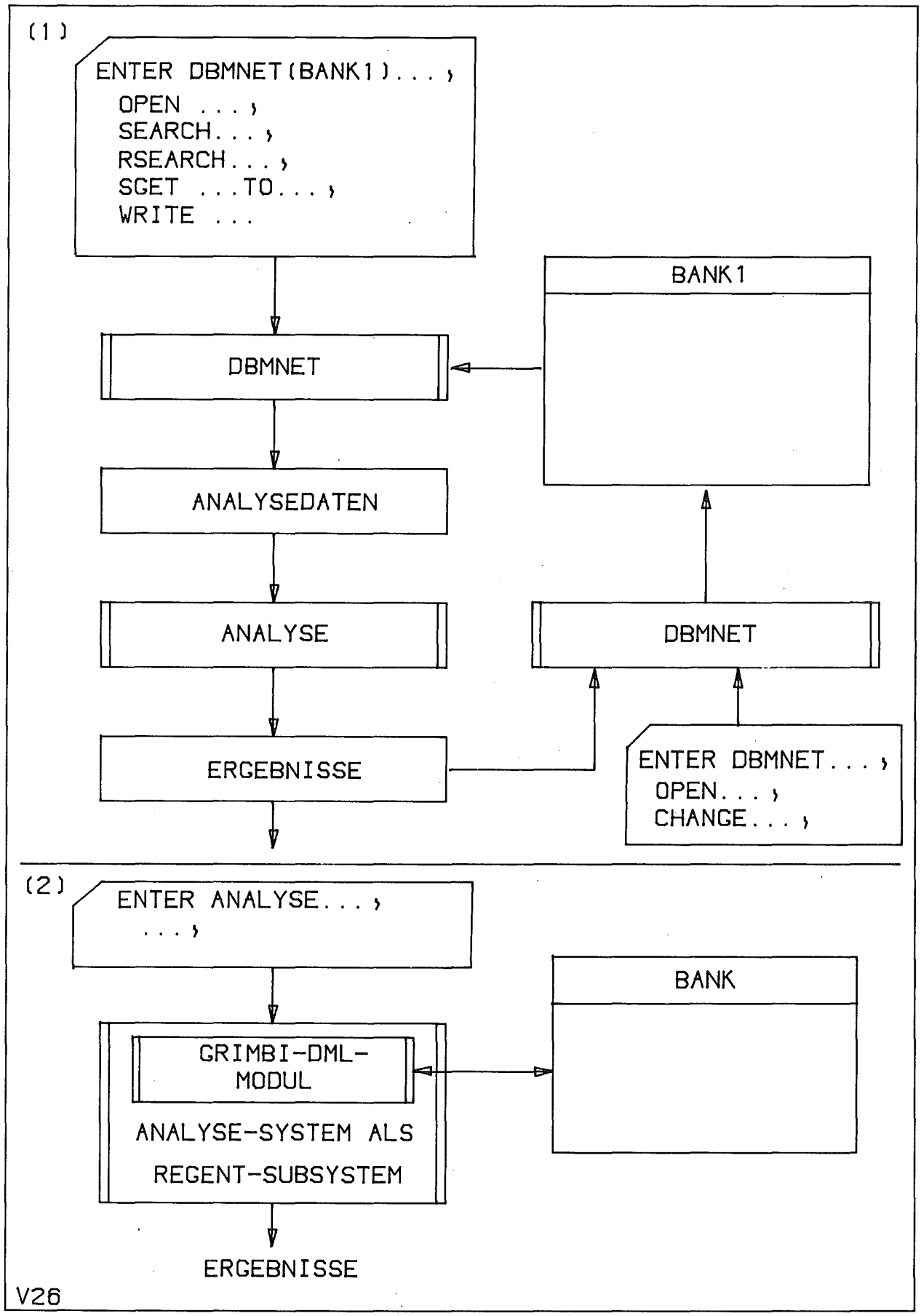


Abb.44: Vorgehensweise bei der Modellauswertung

3.5. Systemarchitektur

Die Systemarchitektur von GRIMBI wird einerseits durch die Grundphilosophie (Unterscheidung von Modellklassen-Definition und Modellieren) beeinflußt und andererseits durch die Entscheidung, den interaktiven Betrieb mit einem intelligenten graphischen Terminal (IGT) durchzuführen, um auch im Timesharing-Betrieb trotz hoher Datenraten (Graphik) tragbare Antwortzeiten zu erzielen.

Die zu lösenden Probleme betreffen neben der Art und Weise der Realisierung im Detail,

- die Arbeitsteilung zwischen dem Gastrechner und dem Terminal (IGT),
- die Arbeitsteilung zwischen dem Stapel- und Timesharing-Betrieb auf dem Gastrechner und
- die Datenverteilung zwischen Gastrechner und Terminal (IGT).

Abb.45 zeigt die Randbedingungen für die Implementierungsentscheidungen bezüglich der Hardware und der zur Verfügung stehenden Software.

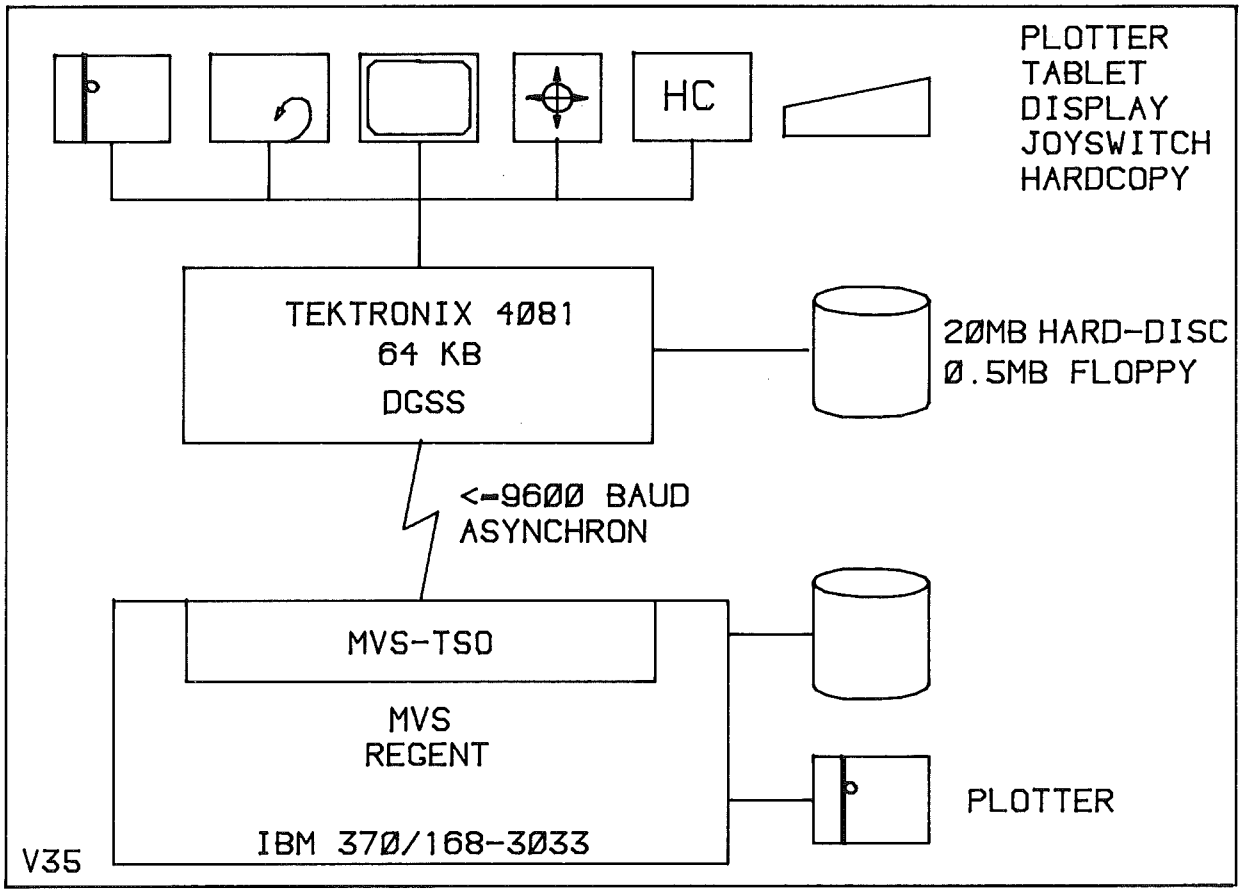


Abb.45: Hardware und Systemsoftware der GRIMBI-Implementierung

3.4.1. Übersicht

Die Basis des Systems bilden zwei REGENT-Subsysteme: DBANET (mit der GRIMBI-DDL) zur Definition von Modellklassen und DBMNET (mit der GRIMBI-DML) zum Modellieren (Abb.46). DBMNET kommuniziert im interaktiven Betrieb mit dem Teilsystem GRIMBI-IGTNET, ein Modul, der auf dem IGT läuft. IGTNET bearbeitet die gesamte Graphik und den Dialog mit dem Anwender (Abb.47). Die IGTNET-Software erweitert das DBMNET-Subsystem um eine interaktive graphische Komponente, die im interaktiven Betrieb auch den REGENT-Sprachübersetzer ersetzt.

Die Verteilung der Daten zeigt Abb.48. Die Modellklassenbeschreibung und die Problemdaten befinden sich auf dem Gast-Rechner. INIT-GRBANK initialisiert auf dem Terminal eine BANK und überträgt die graphische Modellklassenbeschreibung auf das Terminal , d.h. es wird innerhalb der BANK auf dem IGT eine Symbolbibliothek angelegt. Beim Modellieren werden die graphischen Daten ausschließlich auf dem Terminal verwaltet, nur um Sicherheitskopien anzulegen, kann der Anwender Netze/Teilnetze auf den Gast-Rechner übertragen (SAVE). Sie können dort lediglich gezeichnet werden.

Die REGENT-Subsysteme nutzen das REGENT- Subsystem- und Modulmanagement, die Definition und Verarbeitung problemorientierter Sprachen mit Hilfe des Teilsystems PLS /68/ und GIPSY /72/ zur Ausgabe "geretteter" Blockdiagramme. Für das Datenmanagement wurde eine neue Komponente implementiert, die an die Erfordernisse von GRIMBI angepaßt ist: Diese Datenbankkomponente steht natürlich auch außerhalb von GRIMBI allen anderen REGENT-Subsystemen zur Verfügung.

GRIMBI ergänzt somit REGENT (Abb.46) durch

- eine Datenbankkomponente (DBANET, DBMNET) mit problemorientiertem Datenmodell und

- eine graphisch orientierte Dialogkomponenente (DBMNET, IGTNET) für die Bearbeitung dieser Datenbank.

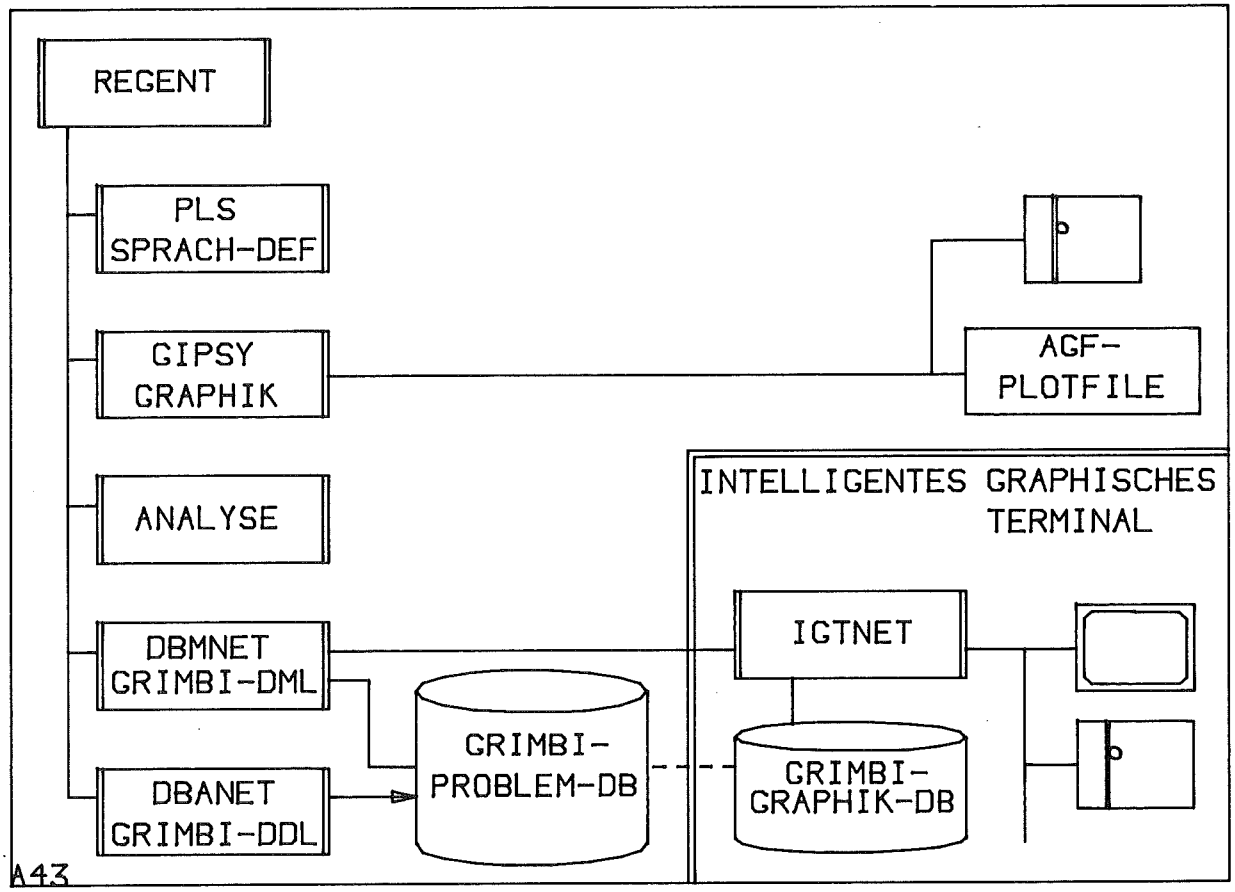


Abb.46: GRIMBI und REGENT

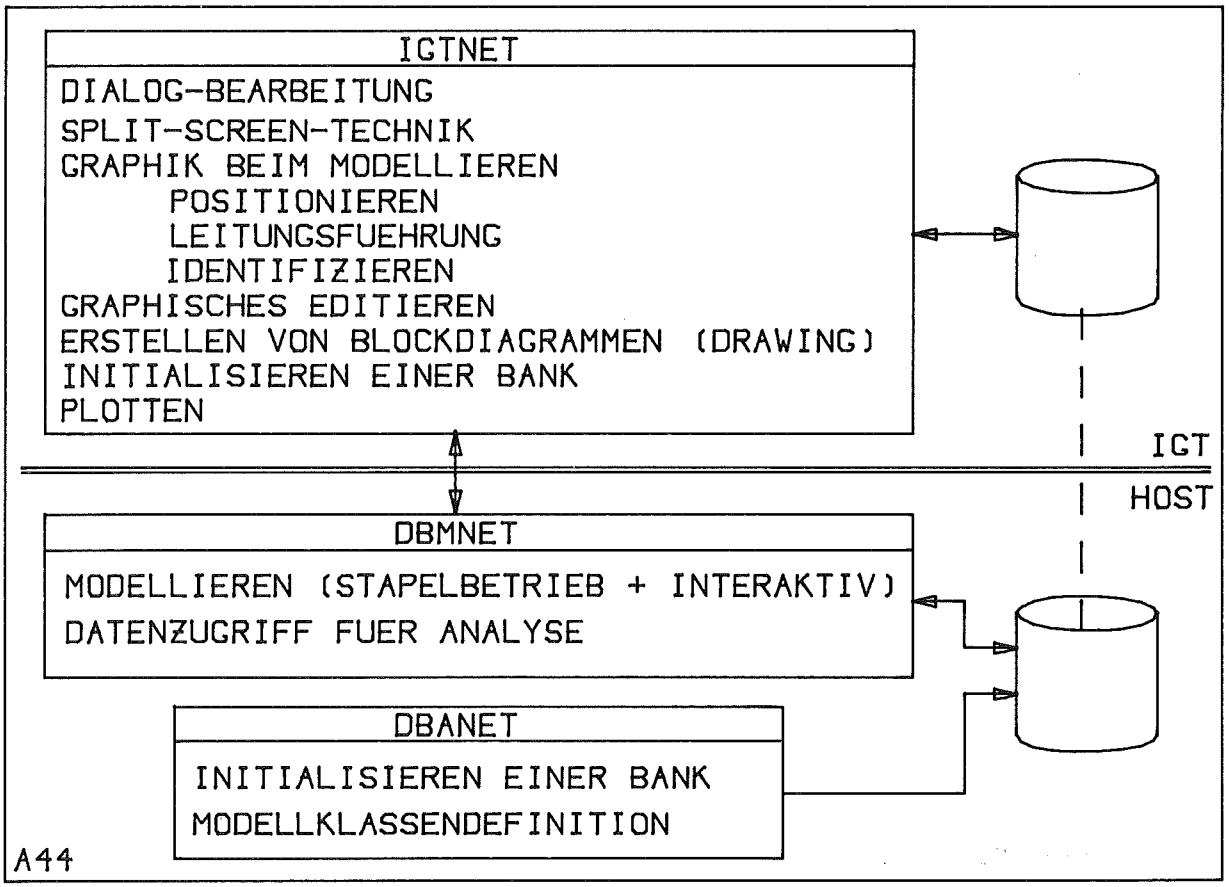


Abb.47: Aufgabenverteilung Host-IGT

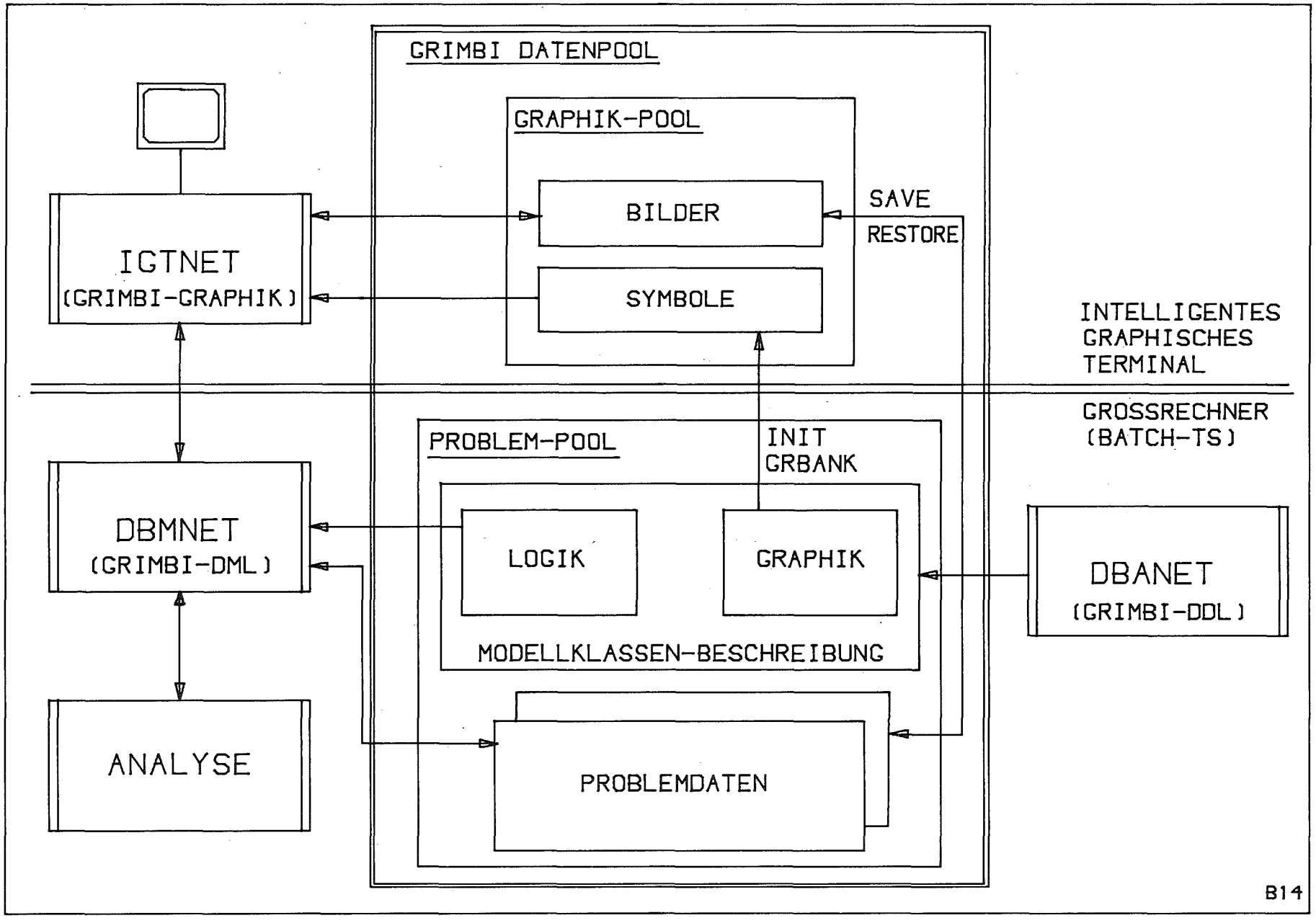


Abb. 48: GRIMBI-Architektur

3.5.2 Das REGENT-Subsystem DBANET

Das Subsystem DBANET (Data Base Administration for NETs) bearbeitet

- das Initialisieren einer BANK und
- die Definition von Modellklassen.

Diese Aufgaben erfordern vor allem gründliche Planung. Sie werden deshalb zweckmäßigerweise im Stapelbetrieb bearbeitet. Das System DBANET enthält die GRIMBI-DDL (Kap.3.1. und Anhang 1).

Den schematischen Ablauf der Arbeit mit DBANET zeigt Abb.49 (siehe auch Kap.3.6). Das Ergebnis eines DBANET-Laufes ist eine BANK mit einer Modellklassenbeschreibung, die je nach Zielsetzung auch den Graphik-Bereich umfaßt.

3.5.3 Das REGENT-Subsystem DBMNET

Das Subsystem DBMNET (Data Base Management for NETs) bearbeitet drei Bereiche:

- Modellieren mit Hilfe der GRIMBI-DML im Stapelbetrieb (Abb.50).
- Zugriff auf die Daten einer BANK für die Analyse, wofür ebenfalls die DBMNET-POL verwendet wird (Abb.46).
- Zusammenarbeit mit dem System IGTNET beim interaktiven Modellieren.

Voraussetzung für die Arbeit mit DBMNET ist natürlich eine vorausgegangene Initialisierung einer BANK mit DBANET.

Die REGENT-Anweisungstreiber der DBMNET-POL enthalten keine Modellsemantik. Sämtliche Konsistenzprüfungen werden von den beiden DBMNET-Moduln DBMS1 und DBMS2 durchgeführt, die die Semantik der Modellbeschreibung entnehmen (Abb.51). Die Anweisungstreiber setzen lediglich die POL in ein internes Befehlsformat um, das dann von DBMS1 aufgrund der Schemainformationen und gegebenenfalls zusätzlich von DBMS2 auch aufgrund des Kontextes geprüft wird. Das interne Befehlsformat gestattet die Ankopplung von IGTNET und eine einfache Protokollierung sämtlicher Aktionen auf einer Logging-Datei.

Bei interaktivem Betrieb werden die Befehls/Antwort-Blöcke an das IGTNET-Programm übertragen. Die Rechner-IGT-Kommunikation erfolgt mit einer für die REGENT-PL/1-Umgebung modifizierten TEKTRONIX-4081-DGSS-Software /81/.

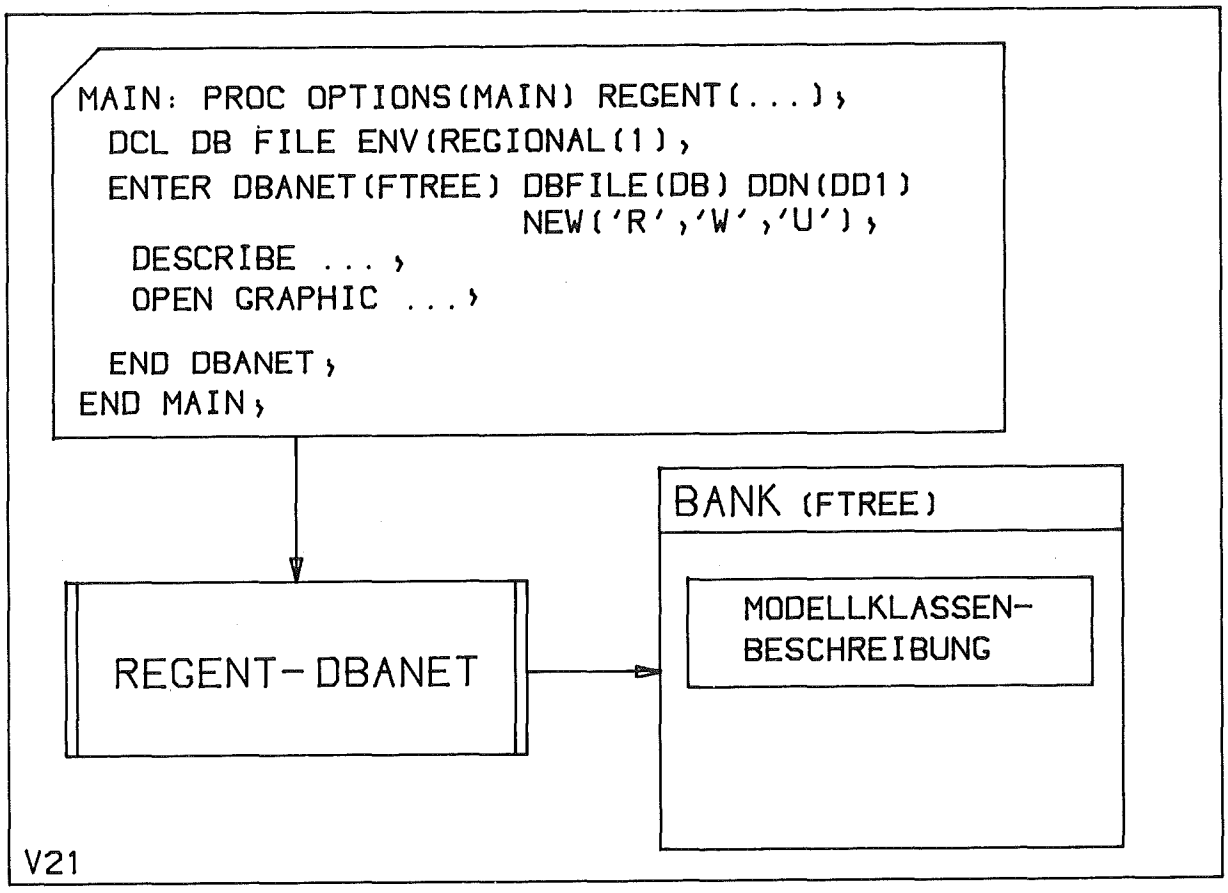


Abb.49: Initialisieren einer Bank

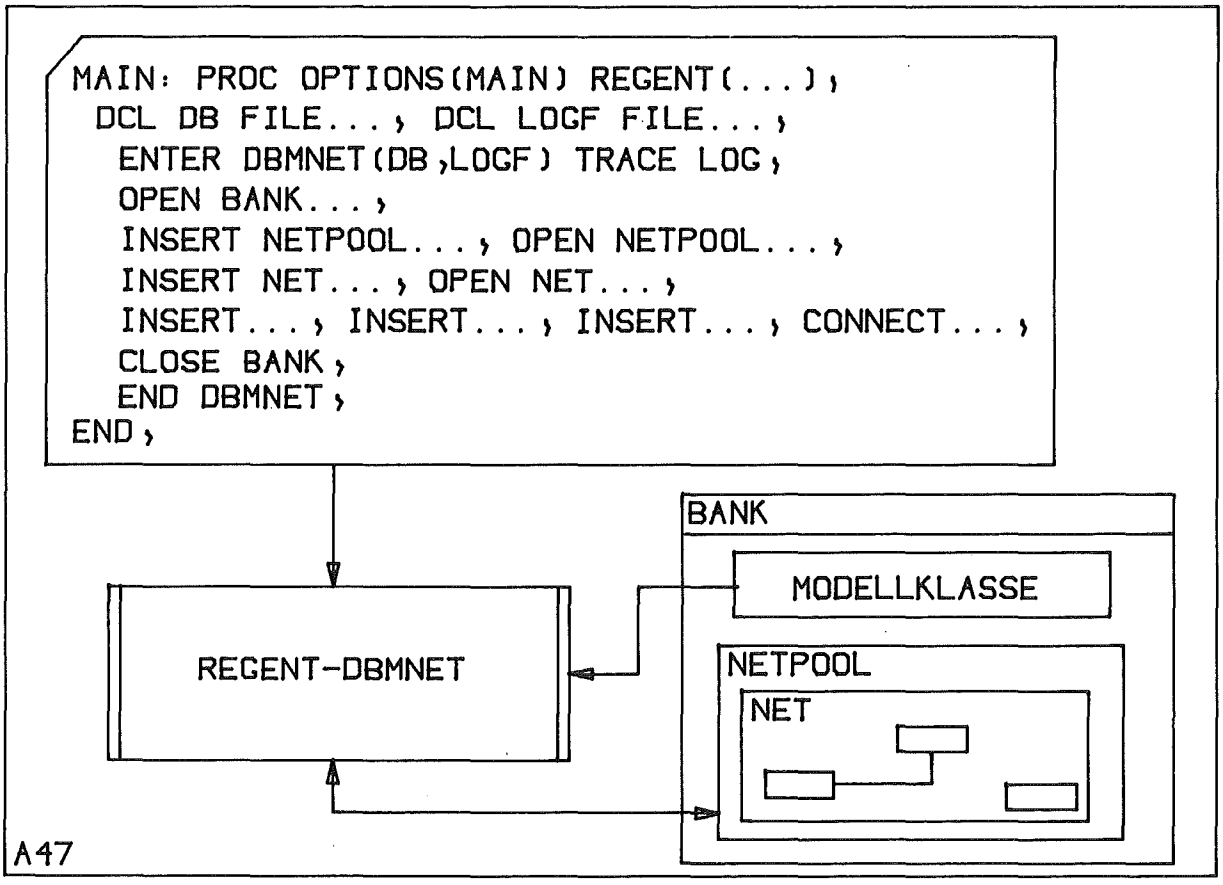


Abb.50: Modellieren im Stapelbetrieb

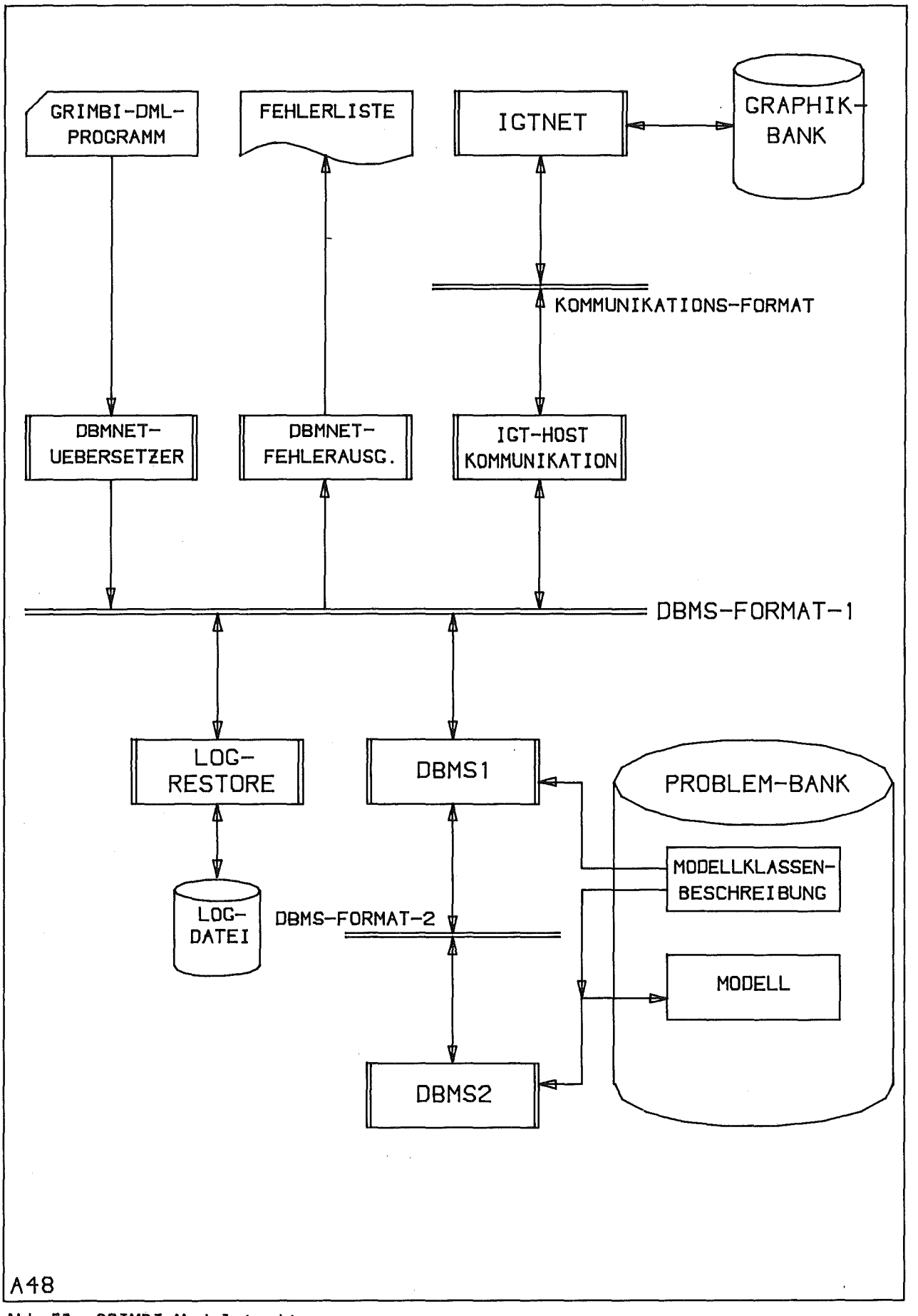


Abb.51: GRIMBI-Modulstruktur

Die Zweiteilung des DBMS-Systems wurde vorgenommen, um bei einer fortgeschrittenen GRIMBI-Version den DBMS1-Teil und das Schema auf das intelligente Terminal übernehmen zu können, so daß dort bereits eine weitreichende Vorprüfung erfolgen kann.

Datenverwaltungssystem

Typisch für das funktionale Modellieren mit GRIMBI sind Datenstrukturen, in denen sogenannte aktuelle Interessenbereiche abgegrenzt werden können, d.h. Bereiche auf die sich Zugriffsaktivität während einer Arbeitsphase im wesentlichen beschränkt. Als Basis der DBMNET-Algorithmen wurde daher eine Dateiverwaltung implementiert,die es gestattet, möglichst effektiv direkt auf Sätze einer Satzmenge zuzugreifen, wobei eine beliebige Anzahl von Satzmengen definierbar ist. Diese Satzmengen werden bezüglich der Ein/Ausgabe als Einheit betrachtet, so daß nach einem Zugriff auf einen Satz dieser Menge die Gesamtmenge (der Interessenbereich) im Arbeitsspeicher liegt. Das Auslagern erfolgt automatisch in Abhängigkeit von der Arbeitsspeicherbelegung und der Zugriffswahrscheinlichkeit, einer Zahl, die beim Zugriff auf eine Satzmenge vom Anwender dieser Satzmenge zugeordnet wird. Mit ihr bestimmt man, in welcher Reihenfolge das System die Satzmengen bei Bedarf auslagert.

Diese Automatik ist in der ersten GRIMBI-Version noch nicht realisiert, das Verwaltungssystem führt aber eine umfangreiche Zugriffsstatistik, um für den weiteren Ausbau Einflußparameter zur Verfügung zu haben.

Dem Entwurf des Datenverwaltungssystems liegen folgende Randbedingungen zugrunde:

- Ausschließliche Verwendung von PL/1-Sprachelementen, um eine weitgehende Übertragbarkeit der Programme zu erzielen.

- Implementierung in einem Paging-Betriebssystem, was insbesondere eine physische Zusammenfassung der zeitlich gemeinsam zu bearbeitenden Datenstrukturen im Arbeitsspeicher erfordert (Minimierung des Working-Sets /70/), um die Paging-Rate gering zu halten. Diese physische Zusammenfassung der Daten eines Interessenbereiches senkt aber auch gleichzeitig die Ein/Ausgabe-Rate des Datenverwaltungssystems.

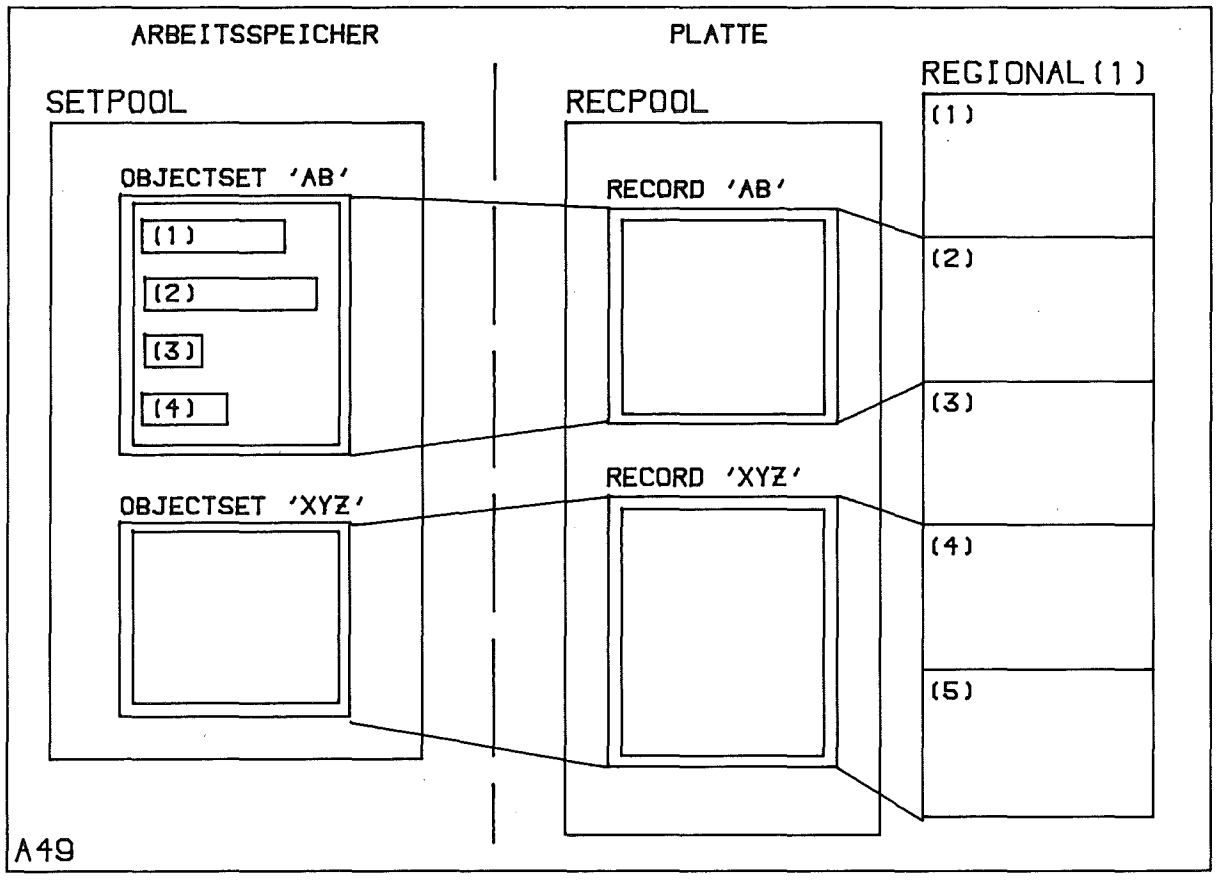


Abb.52: Datenverwaltungssystem

Das System /82/ gliedert sich in eine Satzmengen-Verwaltung (RECORDPOOL-Verwaltung) auf externen Speichern (Platten) und ein Teilsystem zur Verwaltung einer Menge von Satzmengen (Objektmengen) im Arbeitsspeicher (SETPOOL-Verwaltung). Einem SETPOOL im Arbeitsspeicher ist ein RECORDPOOL auf einem externen Speicher zugeordnet (Abb.52).

(1) RECORDPOOL-Verwaltung

Ein RECORDPOOL ist eine Menge von benannten Sätzen beliebiger Länge. Sie werden auf eine PL/1-REGIONAL(1)-Datei /83/ (Records fester Länge mit den Schlüsseln 1...n) abgebildet. Ein Anwender-Satz wird durch einen oder mehrere REGIONAL(1)-Sätze fester Länge dargestellt. Im Gegensatz zu einer PL/1-REGIONAL(3)-Datei wird freigegebener Platz wieder verfügbar, und der Zugriff auf einen Satz erfordert keine sequentielle Suche, sondern erfolgt direkt. Ein RECORDPOOL ist durch folgende Attribute gekennzeichnet:

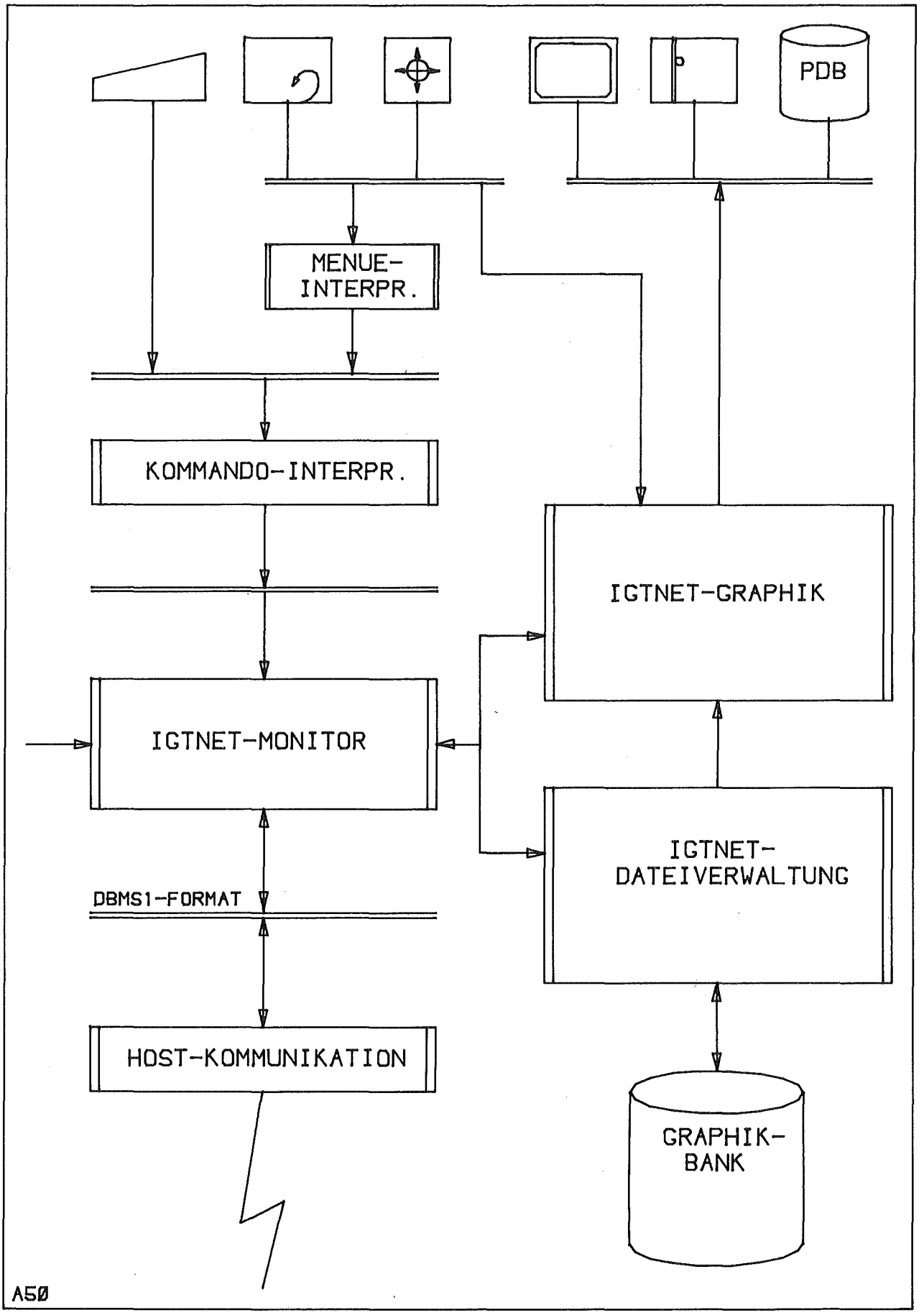


Abb. 53: IGTNET-Architektur

- Name
- Datum und Uhrzeit der Initialisierung
- Datum und Uhrzeit des letzten Zugriffs
- Belegter Platz (in Bytes)
- Vorübergehend nicht nutzbarer Platz
- Zahl der Sätze
- Belegter Platz für Katalog
- Zahl der Katalog- und Satzzugriffe
- Länge der RECORD-Liste im Katalog
- Länge und Inkrement des Katalogs

(2) SETPOOL-Verwaltung

Mit Hilfe der RECORDPOOL-Verwaltung ist die SETPOOL-Verwaltung implementiert worden. Ein SETPOOL ist eine Menge von benannten OBJECTSETs, ein OBJECTSET eine Menge von OBJECTs. OBJECTs enthalten die eigentlichen Anwenderdaten, sie sind durch eine Länge charakterisiert, zu ihrer Identifikation vergibt das System Schlüssel. Die Anwenderdaten sind dem System nicht bekannt, die kleinsten System-Elemente sind die OBJECTs. Auf die Datenelemente eines OBJECTs greift man mit PL/1-Mitteln zu (PL/1-BASED-Variblen).

Ein SETPOOL wird auf einen RECORDPOOL abgebildet, ein OBJECTSET auf einen RECORD eines RECORDPOOLs. Das System übernimmt die Platzverwaltung im OBJECTSET.

SETPOOL-Attribute sind:

- Name
- aktuelle und maximale Anzahl von enthaltenen OBJSET-Elementen
- aktuell und maximal belegter Platz
- Typ (beliebiger INTEGER-Wert, BIN FIXED(31))

und im geöffneten Zustand:

- aktuelle und maximale Zahl der OBJSET-Elemente im Arbeitsspeicher
- maximal verfügbarer Platz für OBJSET-Elemente im Arbeitsspeicher (Anwender-Vorgabe)
- Zahl der Puffer-Reorganisationen
- Zahl der OBJSET-Compresses
- Art der Katalogbehandlung (Arbeitsspeicher-resident oder nicht)

Ein OBJSET-Element hat folgende Attribute:

- Name
- Länge
- aktueller und maximaler Füllstand
- aktuelle und maximale Anzahl von enthaltenen DATAOBJ-Elementen
- Inkrement der Erweiterung
- aktuelle und maximale Zahl der Erweiterungen

und im eröffneten Zustand ergänzend:

- Zugriffswahrscheinlichkeit (Anwender-Angabe)
- Änderungszustand (geändert, nicht geändert)
- Arbeitsspeicherresidenz

Ein eröffneter OBJECTSET wird mit Hilfe einer PL/1-Area und einiger Kontrollinformationen und der zugehörigen Zugriffsroutinen realisiert. Diese Technik garantiert den Zusammenhalt der Daten des jeweiligen Interessenbereiches im Arbeitsspeicher (kleiner Working-Set). Es können gleichzeitig mehrere OBJECTSETs eröffnet und damit im Arbeitsspeicher sein. Gemäß der beim Zugriff vergebenen Prioritäten der OBJECTSETs und maximaler Arbeitsspeicher-Freigabe für die OBJSETs, werden diese automatisch vom System komplett ein- und ausgelagert.

3.5.4. IGTNET

Um im Timesharing-Betrieb auf einem Großrechner akzeptable Antwortzeiten zu erhalten, wurden diejenigen Aufgaben, bei denen vom Anwender sofortige Reaktionen erwartet werden, auf ein intelligentes Terminal ausgelagert. Das Software-Paket IGTNET bearbeitet auf einem TEKTRONIX-4081-Terminal dazu die gesamte Graphik und den Dialog in folgenden Teilaufgaben (Abb.47):

(1) Syntaktische Kommando- und Menü-Bearbeitung,
(2) Initialisieren einer Bank (INIT GRBANK) gemäß der Graphik der Modellklassen-Definition (GRIMBI-DDL)
(3) Positionieren von Symbolen, Aufbau der Leitungsführung beim Modellieren (MODELLING-INSERT-CONNECT)
(4) Identifizieren von Symbolen, Anschlüssen, Attributen und Leitungen durch "Zeigen"

(5) Graphisches Editieren von Blockdiagrammen (EDIT-Befehle), die den Inhalt der Problemdaten-Bank repräsentieren, durch Verschieben von Symbolen und Symbolgruppen und durch Änderungen der Leitungsführung (Erhalt der Topologie)
(6) Nachträgliches Ergänzen eines GRIMBI-Modells durch Graphik (COMPLETE)
(7) Zeichnen von Blockdiagrammen gemäß der Schema-Graphik (DRAWING- Befehle)
(8) Ausgabe von Blockdiagrammen auf Display, Plotter und im TEKTRONIX- PDB-Format /80/ (PLOT)
(9) Realisierung der "Split-Screen"-Technik (SWITCH)
(10) Retten von Blockdiagrammen auf den Host-Rechner und Restauration
(11) Kommunikation mit dem Host-Rechner

Insbesondere werden alle graphischen Daten auf dem IGT gehalten, Bilddaten werden zwischen den beiden Rechnern nur bei der Initialisierung der IGT-Datenbasis und für die Erstellung von Sicherheitskopien auf dem Host-Rechner und zur Restauration übertragen. Graphische Aufgaben, d.h. die Aufgaben des DRAWING- und OFFLINE-Betriebszustandes kann IGTNET autonom ohne Host-Unterstützung bearbeiten (Kapitel 3.2).

IGTNET-Architektur

Diese Leistungen des IGTNET-Systems für die TEKTRONIX-4081 wird durch eine Softwarestruktur erzielt, die aus Abb.53 hervorgeht. Ein Monitor steuert vier Moduln an, die die Graphik, die Datenbank, den Dialog und die Rechner-Rechner-Kommunikation bearbeiten. Implementierungsbasis für IGTNET ist das TEKTRONIX-4081-DGSS /81/, ein Softwarepaket zur Unterstützung der Graphik, der Dialogbearbeitung und der Kommunikation mit dem Host-Rechner.

3.5.5. Die graphische GRIMBI-Umgebung

Zeichnungen als Darstellungsmittel sind eines der GRIMBI-Produkte. Da für die freie graphische Bearbeitung von Zeichnungen und für die Zeichnungsausgabe über verschiedene Ausgabegeräte bereits ausreichende Software zur Verfügung steht, wurden derartige allgemeine Fähigkeiten nicht in GRIMBI aufgenommen. GRIMBI bietet aber, um die vorhandenen Hilfsmittel voll nutzen zu können, zwei Schnittstellen (Abb.54):

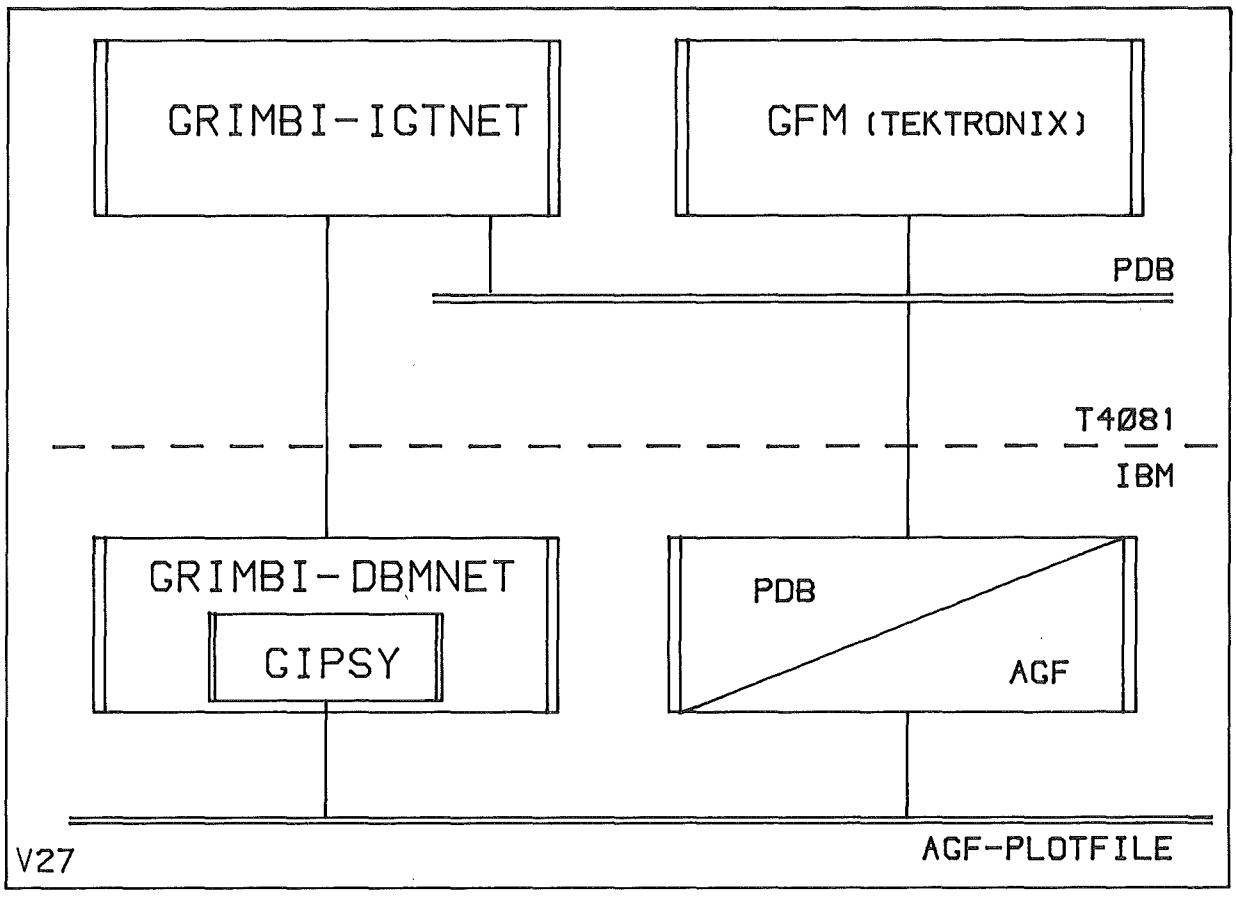


Abb.54: GRIMBI-Graphik-Umgebung

Die TEKTRONIX-PDB-Schnittstelle /16/ und die AGF-Plotfile-Schnittstelle /84/. GRIMBI-Blockdiagramme können in diesen Formaten ausgegebenwerd en, sodaß man sie entweder auf der TEKTRONIX-4081 mit dem GFM (graphic function manager) /16/ interaktiv editieren kann oder über den AGF-Plotfile eine Vielzahl von graphischen Ausgabegeräten erreicht. Auch eine ergänzende Bearbeitung mit GIPSY ist dadurch möglich (/85/,/86/,/71/,/87/).

3.6. Arbeiten mit GRIMBI

Die Arbeit mit GRIMBI beginnt im allgemeinen mit einer Problemanalyse, in der man ermittelt, welche Datentypen in dem zu bearbeitenden Problembereich zum Modellieren benötigt werden. Das Ergebnis dieser Phase wird mit der in Kapitel-3.1. beschriebenen Datendefinitionssprache (DDL) formuliert, normalerweise erst einmal ohne die graphische Beschreibung der Datentypen. Ein REGENT-DBANET-Programm, wie es das folgende Beispiel zeigt, initialisiert auf der Betriebssystemdatei TS0846.BANK.DATA eine GRIMBI-BANK mit einer Modellklassen-Beschreibung.

Beispiel: Initialisieren (Abb.49)

```
//IRE846XX JOB (.......),LEINEMANN
// EXEC REGENT
//P.SYSIN DD *
  INIT: PROC OPTIONS(REGENT);
   ENTER DBANET(FAULTTRE) DDN('BANKDD') NEW('R','W','U') DBFILE(DBFILE)
                          STEXT('Modellklasse Fehlerbaum');
    DESCRIBE ...;
    ...
    DESCRIBE...;
   END DBANET;
   FINISH;
  END INIT;
//G.BANKDD DD DSN=TS0846.BANK.DATA,DISP=SHR
```

Hat man die zugehörigen Analysealgorithmen in das System eingebracht (z.B. als REGENT-Subsystem mit Moduln, die für den Datenzugriff die DBMNET-POL, d.h. die GRIMBI-DML verwenden), so prüft man in einer Testphase im Stapelbetrieb (Fehlerprotokolle), ob die Modellklassenbeschreibung und das Analysemodell konsistent sind, indem man mit der GRIMBI-DML Testmodelle entwickelt und sie analysiert.

Nach erfolgreichem Abschluß dieser Arbeiten ergänzt man des logische Datenmodell durch die gewünschte Symbolik, falls eine interaktive Arbeit vorgesehen ist.

Modelliert wird dann anschließend im Stapelbetrieb mit Programmen, wie sie das nächste Beispiel zeigt, oder interaktiv am Bildschirm. Die dazu bereitstehenden Operationen werden über die DBMNET-POL (GRIMBI-DML) oder am Bildschirm-Arbeitsplatz durch Kommandos bzw. Menüs aktiviert.

Beispiel Modellieren (Abb.50)

```
//IRE846YY JOB (...),LEINEMANN
// EXEC REGENT
//P.SYSIN DD *
   MODEL: PROC OPTIONS(REGENT);
   DCL DBFILE FILE ENV(REGIONAL(1),F);
   DCL LOGFILE FILE RECORD ENV(V BLKSIZE(500) CONSECUTIVE);
   ENTER DBMNET(DBFILE,LOGFILE) STATISTICS TRACE LOG LOGFILE('CMDFILE');
    OPEN BANK(FAULTTRE) BANKDD('BANKDD') PASS('U');
    INSERT NETPOOL.....;
    OPEN NETPOOL.....;
    INSERT NET...;
    OPEN NET...:
    INSERT...;
    ...
    CONNECT...;
    ...
    CLOSE BANK;
   END DBMNET;
   FINISH;
   END MODEL;
//G.BANKDD DD DSN=TSO846.BANK.DATA,DISP=SHR
```

Im Stapelbetrieb erstellte Modelle, d.h. Modelle, denen keine Graphik zugeordnet ist, können auch interaktiv weiter bearbeitet werden. Eröffnet man eine derartige Bank, deren Modellklassenbeschreibung natürlich einen Graphik-Teil enthalten muß (der aber auch nachträglich eingefügt werden kann, (siehe Beispiel), am interaktiven Arbeitsplatz, so fordert das System den Operateur auf , eine Bank auf dem IGT zu initialisieren (implizietes COMPLETE). Wird dann ein NETPOOL (Modell) eröffnet, so wird der Anwender gezwungen, alle Elemente des Bereiches zu positionieren und alle Verbindungen zu verlegen. Erst danach ist es möglich, das Modell interaktiv weiter zu bearbeiten.

Die Möglichkeiten des interaktiven Arbeitens mit GRIMBI zeigt die Menükarte in Abb.55 für die Modellklasse "Fehlerbäume". Bei anderen Modellklassen wird das Symbolfeld ersetzt. Die Menükarte umfaßt die Befehle, die weitgehend graphische Eingaben (Positionieren, Identifizieren) erfordern und sich durch intensive Interaktion auszeichnen. Sie bilden die zentralen Befehle des Modellierens. Die anderen Befehle, wie beispielsweise CHANGE PROPERTY, die einen großen Anteil von Texteingaben erfordern oder auch relativ selten benötigt werden, gibt man im interaktiven Betrieb als Kommandos über die Tastatur ein.

Der linke Teil der Menükarte (2/3) zeigt die jeweils gültigen Symbole. Wenn man ein Symbol identifiziert (PICK), bewirkt man implizit ein INSERT und die Aufforderung, das Symbol im aktuellen Netz zu positionieren. Die Befehlsgruppen unter den Symbolen dienen dem Bearbeiten der Modellstruktur. Sie erfordern sowohl eine Interaktion mit dem Host-Rechner (der logischen (Problem-) Datenbank), als auch mit der graphischen Datenbank auf dem Satelliten.

Die Operationen auf der rechten Seite der Menükarte bewirken keine Interaktion mit dem Host-Rechner, es sind reine Graphik-Befehle an das Teilsystem IGTNET. Sie beeinflussen die Geometrie der Darstellung (PLOTPARMS), den Informationsinhalt der Zeichnung (PLOTTYPE) und ermöglichen das Editieren, d.h. das graphische Verändern der Darstellung, ohne den logischen Gehalt der Datenbank zu verändern.

Die Abb.56 zeigt für die Modellklasse "Fehlerbäume" alle graphischen Details der Symbole, wie sie mit der GRIMBI-DDL definiert wurden: die Basisgraphik, die Attributgraphik, die Textfenster (punktiert), die Anschlußgraphik, die Pickbereiche (strichliert) für das Gesamtsymbol, die Attribute und Anschlüsse und die Anschlußkoordinaten (Kreise).

Beispiel: Nachträgliches Einfügen eines Graphik-Schemas

```
//IRE846XX JOB (.......),LEINEMANN
// EXEC REGENT
//P.SYSIN DD *
  INIT: PROC OPTIONS(REGENT);
   ENTER DBANET(FAULTTRE) DDN('BANKDD') OLD('U') DBFILE(DBFILE)
                          STEXT('Modellklasse Fehlerbaum');
    OPEN GRAHIC(10);
     OPEN OBJTYPE....;
      BASEGRAPHIC; ...; END BASEGRAPHIC;
      ...
     CLOSE OBJTYPE;
    ...
    CLOSE GRAPHIC;
   END DBANET;
   FINISH;
  END INIT;
//G.BANKDD DD DSN=TSO846.BANK.DATA,DISP=SHR
```

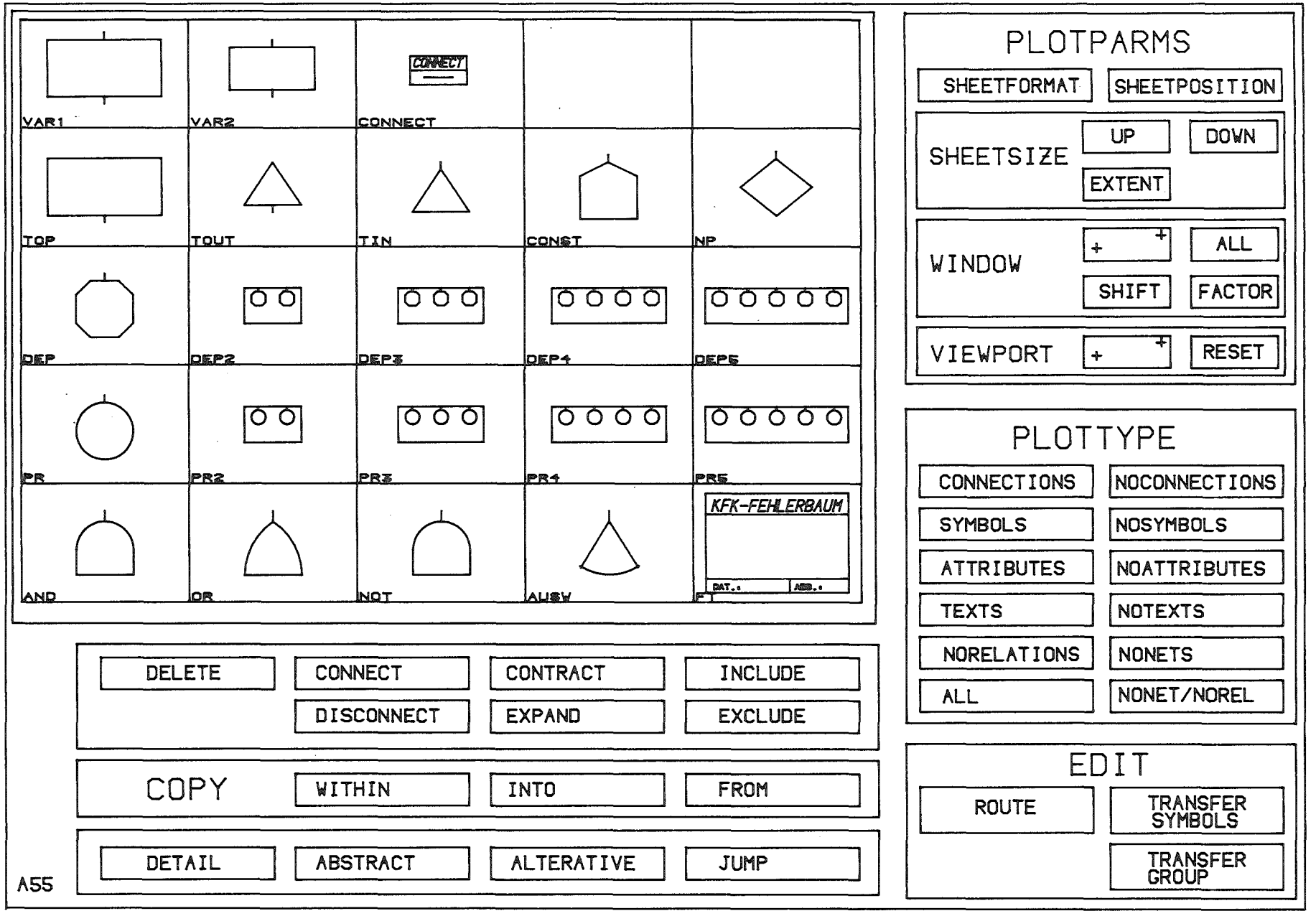


Abb.55: Menüfeld für interaktives Arbeiten mit GRIMBI

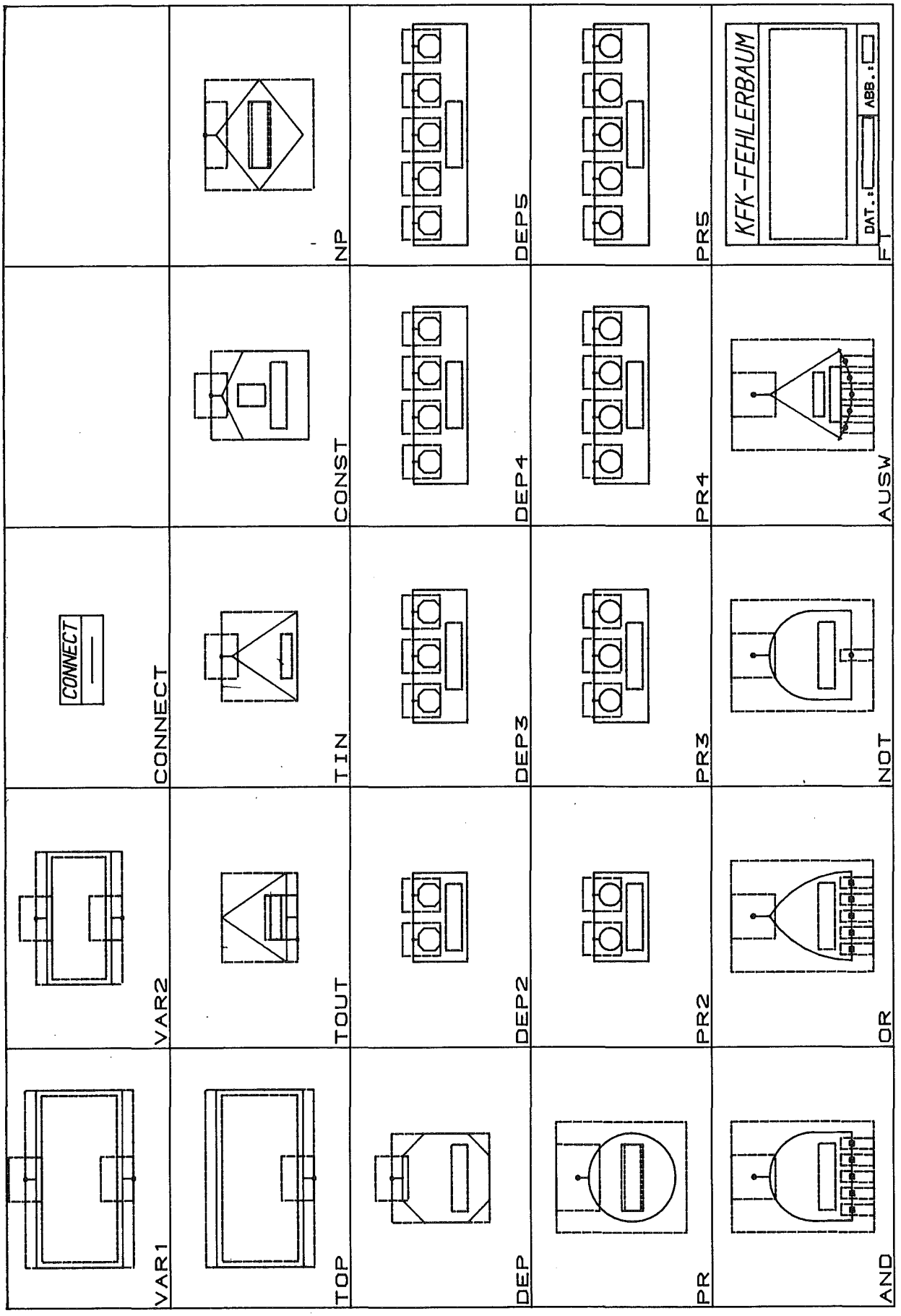


Abb.56: DUMP SYMLIB für Fehlerbaum-Modellklasse mit allen Optionen)

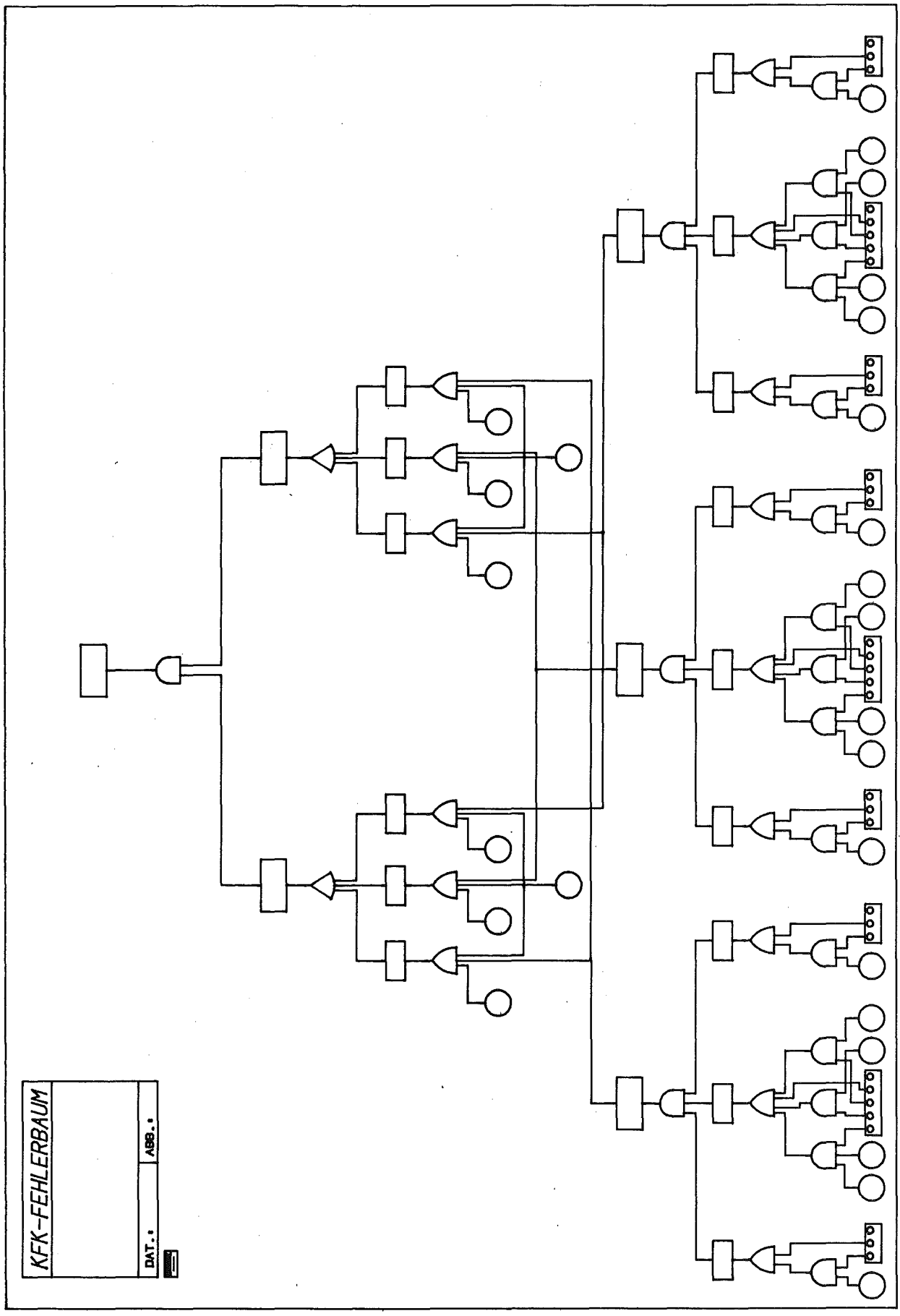


Abb.57: Beispiel eines GRIMBI-Fehlerbaumes

Wird ein interaktiv erstelltes Modell (d.h. ein Modell mit einer graphischen Repräsentation) im Stapelbetrieb verändert, so registriert GRIMBI diese Aktion und fordert den Anwender bei einer nachfolgenden interaktiven Sitzung zu graphischen Ergänzungen auf (im Extremfall zum Neuaufbau der gesamten Graphik).

Das folgende Beispiel zeigt eine interaktive Sitzung mit GRIMBI, bei der zu Beginn eine Bank auf dem IGT initialisiert wird.

Beispiel

System	Anwender	Kommentar
	IPL GRIMBI	
GOING ONLINE?	YES	
<HOST MODE>	LOGON....	
	EXEC GRIMBI BANK.DATA	
TRACE?	YES	Trace auf SYSOUT=A
LOGGING?	YES für Restart	
STATISTICS?	NO	
ENTER COMMAND:	INIT GRBANK	Initialisiere Bank auf IGT Übertragen des graphischen Schemas (Symbolik)
ENTER IGT-FILE:	TEST.LIB	auf TEST.LIB
ENTER COMMAND:	OPEN BANK TEST.LIB	
ENTER COMMAND:	DUMP SYMLIB	Zeichnet alle Symbole der Modellklasse (Abb.56)
ENTER COMMAND:	INIT NETPOOL..	
ENTER COMMAND:	OPEN NETPOOL...	
ENTER COMMAND:	INIT NET...	
ENTER COMMAND:	OPEN NET...	
ENTER COMMAND:	INSERT ...	
ENTER COMMAND:	CONNECT ...	
ENTER COMMAND:	CHANGE ...	
ENTER COMMAND:	PLOT	Fehlerbaumbeispiel in Abb.57
ENTER COMMAND:	CLOSE BANK	
ENTER COMMAND:	END	

Dieses Sitzungsbeispiel zeigt das Modellieren mit GRIMBI, wobei sowohl der Host-Rechner als auch das IGT benötigt werden. Beginnt man die Sitzung dagegen mit "GOING ONLINE? NO", so stehen für die Bearbeitung einer Bank, die im ONLINE-Betrieb erstellt wurde, lediglich die EDIT-Befehle (Kapitel 3.2.) zur Verfügung, so daß man nur graphisch editieren kann. Man kann in diesem OFFLINE-Zustand des IGT mit GRIMBI aber auch Blockdiagramme gemäß einer Modellklasse erstellen, wobei dann aber keine problemabhängigen Prüfungen der Eingabe erfolgen, sondern nur blockdiagramm-typische Restriktionen gelten. Für die Art der Arbeit verwendet man eine im ONLINE-Betrieb initialisierte, leere BANK. Eine solche Bank kann später weder im ONLINE-Betrieb verwendet noch der Analyse zugeführt werden.

4. Ausblick

Im nächsten Schritt des GRIMBI-Projektes wird das System in die Praxis überführt, und zwar im Bereich der Fehlerbaumanalyse. Dabei sollen vor allem auch synergetische Aspekte untersucht werden, insbesondere die folgenden Fragestellungen:

- Nutzt ein Ingenieur ein solches Hilfsmittel vom Entwurfsbeginn an, oder läßt er einen Grobentwurf, der nach herkömmlicher Art und Weise am Schreibtisch erstellt wurde, durch eine Hilfskraft in das System eingeben, um ihn dann interaktiv zu überarbeiten?

- Was hält ihn gegebenenfalls davon ab, sofort mit dem System zu arbeiten?

Darüberhinaus sollen weitere Einsatzmöglichkeiten von GRIMBI untersucht werden, wie beispielsweise die Software- bzw. die allgemeine Systemspezifikation /88/, /89/,/90/, /91/, /92/, /93/, die Anwendung im Bereich der funktionalen Anlagenplanung (R&I-Planung, Kapitel 1) und die Kopplung mit der daran anschließenden räumlichen Anlagenplanung.

In diesem Zusammenhang treten auch Probleme einer zentralen Datenbasis in den Vordergrund, wie

- Modelltransformationen (Abb.2),

- Mehrfachzugriff,

- Aufbau eines zentralen Modells, Datenmodell eines zentralen Modells.

5. Zusammenfassung

Das CAD-System GRIMBI unterstützt das funktionale Modellieren, bei dem es darum geht, aus vorgegebenen Elementen oder Teilsystemen komplexe Systeme bestimmter Funktion aufzubauen und sie dann gegebenenfalls zu analysieren. Bewährtes Hilfsmittel in diesem Problembereich ist die Blockdiagramm-Methode, deren Operanden und Operationen einschließlich ihrer graphischen Unterstützung für den EDV-Einsatz aufbereitet wurden. Sie bilden die Basis für die Bearbeitung und Verwaltung der funktionellen Modelle (Blockdiagramm-Informationen) mit GRIMBI.

Wie Datenbanksysteme kennt GRIMBI zwei Phasen. In einem Schritt der Arbeitsvorbereitung definiert man mit einer Datendefinitions-Sprache (GRIMBI-DDL), die auf einem problemorientierten Datenmodell basiert, eine Modellklasse. Sie umfaßt die Beschreibung

- der Objekttypen dieser Klasse durch Attribute, Anschlüsse und Strukturen,
- der Relationen und ihrer problemgegebenen Restriktionen,
- der Organisationsstrukturen zur Gliederung des Gesamtsystems in überschaubare Teilsysteme und
- der externen Darstellung dieser Modelle durch eine parametrisierte Symbolik.

Derartige Modellklassen bilden den Rahmen für die andere GRIMBI-Phase, das Modellieren eines Systems und seiner Auswertung. Dazu dient eine Datenmanipulationssprache (GRIMBI-DML), deren Operationen den Modellklassen-Restriktionen unterliegen. Neben dem Aufbau der Problemstruktur unterstützt sie auch die Bearbeitung von Teilsystemen und Entwurfsalternativen. Die GRIMBI-DML steht als Erweiterung einer Gastsprache (PL/1) im Stapelbetrieb (vor allem zur Analyse) und als selbständige Kommando/Menü-Sprache im interaktiven Betrieb zur Verfügung.

Im interaktiven Betrieb werden die Informationen durch Blockdiagramme mit der Modellklassen-Symbolik dargestellt und durch Zeigen identifiziert.

GRIMBI zeichnet sich durch eine hohe Flexibilität der Problemanpassung aus:

- Änderungen oder Neudefinitionen von Modellklassen, die im Forschungs- und Entwicklungsbereich häufig sind, erfordern lediglich einen Datendefinitionslauf und gegebenenfalls die Bearbeitung eines Analysemoduls, aber keinerlei Bearbeitung von Systemmoduln.
- Der Wechsel von Modellklassen ist dynamisch möglich, im interaktiven Betrieb können sogar zwei Banken verschiedener Modellklassen parallel bearbeitet werden ("Split-Screen"-Technik).

Aufgrund der Verwendung eines intelligenten graphischen Terminals (IGT) und einer geeigneten Aufgaben- und Datenverteilung zwischen IGT und Gast-Rechner bietet GRIMBI neben problemgerechten Antwortzeiten auch die Möglichkeit einer abgestuften Nutzung seiner Fähigkeiten entsprechend den Erfordernissen:

- als System zum Modellieren mit problemtypischen Restriktionen und Hilfen, wobei auf dem Gast-Rechner die Problemdaten und auf dem IGT der Dialog und die gesamte Graphik bearbeitet werden (MODELLING-Betrieb) und
- als Blockdiagramm-Zeichensystem unter Verwendung der Modellklassen- Graphik und blockdiagramm-typischer Restriktionen und Hilfen ohne Gast-Rechner-Unterstützung (OFFLINE-DRAWING-Betrieb).

GRIMBI ist als Subsystem des integrierten CAD-Systems REGENT implementiert und ergänzt REGENT durch eine problemspezifische Datenbankkomponente und eine graphisch orientierte Dialogkomponente, es realisiert innerhalb der REGENT-Methodenbank die Methode des funktionalen Modellierens, durch die vorhandene Analysemethoden ergänzt werden.

Die Abb.58 zeigt, wie die drei GRIMBI-Teilsysteme GRIMBI-DBANET (Datendefinition), GRIMBI-DBMNET (Datenmanipulation) und GRIMBI-IGTNET (interaktive graphische Bearbeitung) die einzelnen in Kapitel 1 genannten Modellierungsphasen unterstützen.

Mit GRIMBI steht ein Werkzeug bereit, das bereits in einem frühen Konstruktions-/Planungsstadium, nämlich dem funktionellen Modellieren, geeignete, umfassende EDV-Hilfsmittel verfügbar macht, indem es nicht nur zeichnerische Unterstützung bietet, sondern eine problemangepaßte Informationsverarbeitung realisiert, die auf Datenbanktechniken fußt und sie durch interaktive, graphische Methoden ergänzt.

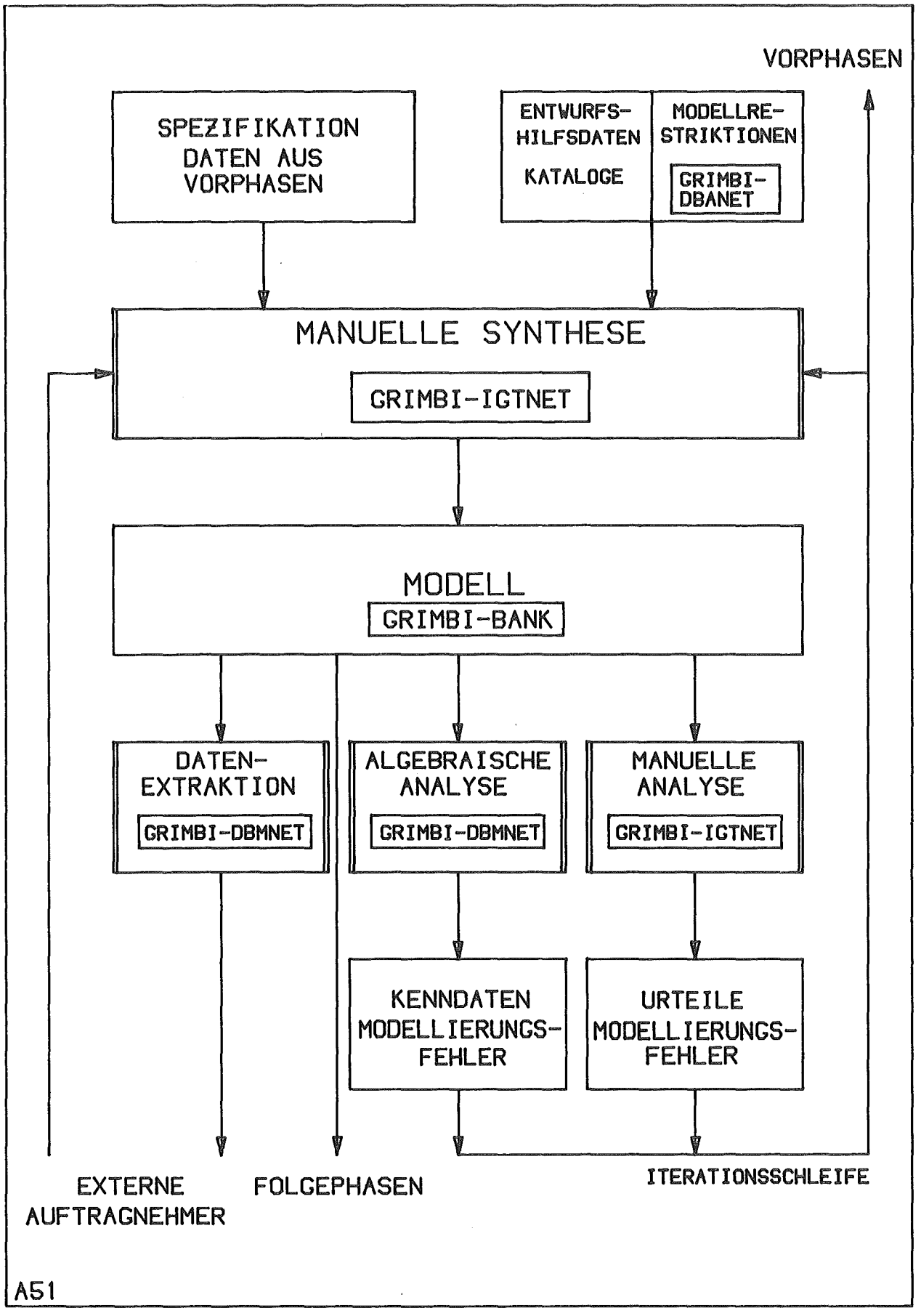


Abb. 58: Unterstützung des Modellierens durch GRIMBI

6. Literaturverzeichnis

****

/1/ Roth,K.:Methodisches Ermitteln von Funktionsstruktur und Gestalt. VDI-Berichte (1974)219 S.89-99

/2/ Schlechtendahl,E.G.: Grundzüge des integrierten CAD - Systems REGENT. Angewandte Informatik 18(1976) S.490-496

/3/ Beitz,W.: Übersicht über Konstruktionsmethoden. Konstruktion 24(1972) S.68-72

/4/ DIN 28004: Fließbilder verfahrenstechnischer Anlagen.

/5/ Grabowski,H.,Schlingensiepen,J.,Sebregondi,H.-P.,Kollmann,F.-G.: Rechnerunterstützte Anlagenplanung. KfK-CAD 141, 1979.

/6/ DIN 19227: Bildzeichen und Kennbuchstaben für Messen, Steuern, Regeln in der Verfahrenstechnik.

/7/ VDI 4008: Boolesches Modell.

/8/ DIN 25424 Fehlerbaumanalyse

/9/ DIN 25419: Störfallablaufanalyse.

/10/ Smidt,D.: Reaktor-Sicherheitstechnik. Springer, Berlin 1979.

/11/ Caldarola,L.: Generalized Fault Tree Analysis Combined with State Analysis KfK-2530, Februar (1980)

/12/ Leinemann,K.,Schumann,U.: Ein Beitrag zu Grundlagen des rechnergestützten Entwurfs und einer Konstruktionssprache. KfK-Ext.8/72-5

/13/ Pichler,F.: Einige Anwendungen gerichteter Graphen in der mathematischen Systemtheorie. In: Mühlbacher,J.(ed.): Datenstrukturen, Graphen, Algorithmen. Hansen, München (1978) S.18-36

/14/ Chen,P.P.: The Entity-Relationship Model - Toward a Unified View of Data. ACM Trans. on Database Systems 1(1976)1 S.9-36

/15/ Dreger,W.: Grundlagen der Systemtechnik.
IBM Nachrichten 23(1973)218 S.823-829

/16/ TEKTRONIX: PLOT80 Graphic Function Manager. Beaverton (1978)

/17/ Interactive Graphic Design System, M&S Europe B.V., Amsterdam,
Report-Nr.77-033,77-129,78-057

/18/ Belady,L.A.,Blasgen,M.W.,Evangelisti,C.J.,Tennison,R.D.: A computer graphics system for block diagram problems. IBM SYST. J. (1971)2 S.143-161

/19/ CABLOS - CAD Softwaresystem. Applied Graphics Systems Deutschland,München

/20/ Bayegan,H.M.: CASS : A Computer - Aided Schematic System. Proc. of CAD78-Conf. Brighton, IPC Science and Technology Press, (1978) S.416-423

/21/ LEADS, Computer Aided Design Centre, Cambridge

/22/ AGS877, Applicon Inc., Burlington, USA

/23/ Evangelisti,C.J.,Morse,S.P.: Graphical modelling using contextually implied functions. The Computer Journal 14(1971)4 S.382-389

/24/ Morse,S.P.: Interactive graphic system for modeling symbolic networks.
Computer Design (1973)2 S.75-79

/25/ Baskin,H.B.,Morse,S.P.: A multilevel modeling structure for interactive graphic design. IBM SYST. J. (1968)3&4 S.218-228

/26/ Grauer,H.,Jarsch,V.: User-oriented and problem-independent computer-aided generation of schematic drawings in a simple low-cost environment.
Proc. of CAD78-Conf. Brighton,
IPC Science and Technology Press, (1978) S.573-577

/27/ Mancusi,M.D.,Wild,J.C.: Interactive graphics for schematic editing, a working tool.
Proc. 9. ACM/IEEE Design Automation Workshop (1972) S.301-304

/28/ Goebl,R.: Interaktive graphische Editoren. In: Weiss,J.(Hrsg.):
Graphische Datenverarbeitung 79, Schriftenreihe der Oestreichischen Computer Gesellschaft 4(1979) S.213-228

/29/ Atiya,J.: An interactive graphics schematic drawing system. CAD,IEE Conf. Proc. Nr.111, 8.-11.4.1974,Southampton

/30/ Feger,O.: Ein Programm zur automatischen Konstruktion von Stromlaufplaenen. Angewandte Informatik (1974)11 S.483-487

/31/ Evans,D.S.,Katzenelson,J.: Data structure and man-machine communication for network problems. Proceedings of the IEEE 55(1967)7 S.1135-1144

/32/ Amkreutz,J.: Ein Graphenprozessor als Planungs- und Entwurfsinstrument im Bauwesen. In: Mühlbacher,J.(ed.): Datenstrukturen, Graphen, Algorithmen. Hanser, München (1978) S.133-161

/33/ Marovac,N.,Elliot,W.S.: A network-oriented language - A new approach to network design, using interactive graphics. Computer & Graphics 2(1977) S.235-239

/34/ Marovac,N.: The structure of the network definition language NEDLAN. Its use in defining networks in CAD using interactive computer graphics. Comput. & Graphics 2(1976) S.23-29

/35/ Marovac,N.: A method for defining general networks for CAD, using interactive computer graphics. The Computer Journal 17(1974)4 S.332-336

/36/ Dumas,J.M.,Nizard,E.,Prunet,F.,Vaucanson,J.P.: A tool for designing graphical and distributed C.A.D. systems. CAD80-Conference Proceedings, IPC Science and Technology Press, März (1980) S.14-21

/37/ Konkart,R.,Alff,E.,Hornung,C.: GRADAS Graphisches Informationssystem auf der Grundlage einer relationalen Datenbank. Informatik Fachberichte 11(1977) S.262-276

/38/ Glatzer,V.,Encarnacao,J.L.: DATAS - Datenstrukturen in assoziativer Speicherung. Angewandte Informatik (1972)9 417-424

/39/ Atkinson,M.P.: PIXIN: A data language for network modelling. Information Processing 74 - North Holland Publishing Company (1974) S.296-300

/40/ Musgrave,G.(ed.): Computer-Aided Design of digital electronic cuircuits and systems. North Holland, Amsterdam (1979)

/41/ Bogo,G.,Lux,G.A.,Payan,C.: CASSANDRE and the computer aided logical system design. Information Processing 71 - North Holland Publishing Company (1972) S.1056-1065

/42/ Strong,W.L.,Thomas,K.D.: Computer graphics in power plant design. IEEE 1978 Power Egineering Society Summer Meeting, Los Angeles, 16.-21. Juli 1978 S.1-7

/43/ Irani,K.B.: On network linguistics and conversational design of queuing networks. Journal of the ACM 18(1971)4 S.616-629

/44/ Bass,L.,Wynhold,H.W.,Porterfield,W.R.: Fault Tree Graphics. In: Barlow,R.E.,Fussell,J.B.(eds.):Reliability and Fault Tree Analysis. SIAM, Philadelphia, (1975) S.913-927

/45/ Wedekind,H.: Datenbanksysteme I. Bibliographisches Institut, Mannheim (1974)

/46/ Date,C.J.: An introduction to data base systems. Addison-Wesley, Reading (1977)

/47/ Grabowski,H.,Eigner,M.: Anforderungen an CAD - Datenbanksysteme. VDI-Z 121(1979) S.33-45

/48/ Tsubaki,M.: DPLS - Database, Dynamic Program Control & Open-Ended POL Support. ACM Workshop on Databases for Interactive Design, Waterloo, Canada, 15.-16.9.75, S.146-160

/49/ Olle,W.T.: The CODASYL Approach to Data Base Management. Chichester (1978)

/50/ Tsichritzis,D.C.,Lochovskey,F.H.: Data Base Management Systems. New York (1977)

/51/ Schlageter,G.,Stucky,W.: Datenbanksysteme: Konzepte und Modelle. Teubner, Stuttgart (1977)

/52/ Kerschberg,L.,Klug,A.,Tsichritzis,D.: A Taxonomy of Data Models. In: Lockemann,P.C.,Neuhold,E.J.(eds.): Systems for Large Data Bases, North Holland Publishing Comp., (1976) S.43-64

/53/ Fischer,Blume: Datenbanksysteme für CAD - Anwendungen.
KfK-CAD 111,(1978)

/54/ Astrahan.M.M. et al: System R: Relational Approach to Database Management.
ACM Trans. on Database Systems 1(1976)2 S.97-137

/55/ Leesley,M.E.,Buchmann,A.P.: Databases for Process Plant Design.
CAD78-Conf. Proc., IPC Sience and Technology Press, (1978) S.270-277

/56/ Lafue,G.M.E.: Integrating language and database for CAD application.
Computer-aided design 11(1979)3 S.127-130

/57/ Grabowski,H.,Eigner,M.: Semantic Datamodel Requirements and Realization with a Relational Datastructure.
Computer Aided Design 11(1978)3 S.158-168

/58/ Grabowski,H.,Eigner,M.,Rausch,W.: CAD Data-Structure for Minicomputers.
CAD78-Conf. Proc., IPC Sience and Technology Press, (1978) S.530-548

/59/ Melanson,G.S.,Spurlin,S.A.: A fundamental approach to data base implementation for design automation. Proc. of the Workshop on Data Bases for Interactive Design, Waterloo, 15.-16.9.1975, S.112-119

/60/ Codd,E.F.: A relational model of data for large shared data banks.
CACM 13(1970) S.377-387

/61/ Lockemann,P.C.,Mayr,H.C.: Rechnergestützte Informationssysteme.
Springer, Berlin (1979)

/62/ Leinemann,K.: Rechnergestützte Anlagenplanung: Systemanalyse und Lösungsansätze für den EDV-Einsatz. KfK 2831 B, September (1979)

/63/ IBM: Information Management System/360. IBM-Form GH20-0765.

/64/ Codd,E.F.: A Data Sublanguage Founded on the Relational Calculus. In:
ACM-SIGFIDET Workshop on Data Description, Access and Control
(1971) S.35-68.

/65/ Encarnacao,J.L.: Computer Graphics, Oldenbourg, München (1975)

/66/ Leinemann,K.,Schlechtendahl,E.G.: The REGENT - System for CAD. In: Allan,J.J.III(ed):CAD Systems. Proc. IFIP Working Conf. on CAD Systems, Austin (Texas) (1975) S.143-167

/67/ Schlechtendahl,E.G.,Enderle,G.,Leinemann,K.,Schuster,R.: Das Programmsystem REGENT im praktischen Einsatz. Informatik Fachberichte 11(1977) S.287-301

/68/ Enderle,G.: Definition, Übersetzung und Anwendung benutzerorientierter Sprachen als Erweiterung von PL/1 in dem System für das rechnerunterstützte Entwickeln und Konstruieren REGENT. KFK 2204, (1975)

/69/ Enderle,G.: Problemorientierte Sprachen im REGENT-System. Angewandte Informatik 18(1976)12 S.543-549

/70/ Leinemann,K.: Dynamische Datenstrukturen des integrierten CAD - Systems REGENT. Angewandte Informatik 19(1977) S.26-31

/71/ Schuster,R.: Graphische Fähigkeiten als Bestandteile eines Systems für den rechnergestützten Entwurf. Angewandte Informatik 19(1977) S.155-163

/72/ Schuster,R.: System und Sprache zur Behandlung graphischer Informationen im rechnergestützten Entwurf. KFK 2305 (1976)

/73/ Leinemann,K.: A CAD-System for interactive graphical handling of hierarchical block diagram information. In: Vandoni,C.E. (ed.): EUROGRAPHICS 80, Proc. of the international conference and exhibition, Genf, 3.-5.9.80, North-Holland Publishing Company, Amsterdam. S.179-189

/74/ Leinemann,K.: GRIMBI - Ein CAD-System für das funktionale Modellieren. In: Wilhelm,R.: CAD-Fachgespräch, GI - 10.Jahrestagung, Informatik-Fachberichte Bd.34, Springer, Berlin (1980) S.51-65

/75/ Leinemann.K.: Ein Ansatz zur formalen Objektbeschreibung für das rechnergestützte Entwickeln und Konstruieren. Angewandte Informatik (1974)9 S.403-406

/76/ Denert.E.,Fank,R.: Datenstrukturen. BI-Wissenschaftsverlag, Mannheim(1977)

/77/ Liskov,B.,Zilles,S.: Programming with Abstract Data Types. Proc. Symp. on Very High Level Languages, SIGPLAN Notices 9(1974)4 S.50-59

/78/ Jensen,K.,Wirth,N.: PASCAL User Manual and Report. Springer, Berlin (1978)

/79/ IBM: SPF General Information Manual. IBM-Form GH20-1974

/80/ TEKTRONIX: PLOT80 Picture Data Base. Beaverton (1977)

/81/ TEKTRONIX: PLOT80-DGSS Reference Manual. Beaverton (1978)

/82/ Leinemann,K.,Steil,A.: Ein Programmpaket zur Verwaltung von Satzmengen in PL/1 auf einer REGIONAL(1)-Datei, unveröffentlicht.

/83/ IBM: PL/1-Optimizing Compiler, Language Reference Manual.

/84/ Enderle,G.,Giese,I.,Krause,M.,Meinzer,H.P.: AGF-Plotfile - Eine Datei zum Speichern und Transportieren graphischer Information. KfK 2776 (1976)

/85/ Kühl,D.,Leinemann,K.: Erfahrungen mit dem Einsatz des AGF-PLOT-FILE-Formats. Angewandte Informatik (1979)9 S.401-405

/86/ Leinemann,K.,Royl,P.,Zimmerer,W.: Integration graphischer Systeme für Darstellungszwecke. KfK-Nachr. 12(1980)1-2 S.75-80

/87/ Enderle,G.,Bechler,K.H.,Katz,F.,Leinemann,K.,Olbrich,W., Schlechtendahl,E.G.,Stölting,K.: GIPSY-Handbuch. KfK 2878, März (1980)

/88/ Estrin,G.: A methodology for design of digital systems - Supported by SARA at the age of one. AFIPS Conference Proceedings 47(1978) S.313-324

/89/ Campos,I.M.,Estrin,G.: SARA aided design of software for concurrent systems. AFIPS Conference Proceedings 47(1978) S.324-336

/90/ Teichroew,D.,Hershey,A.: PLS/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems. IEEE Trans. on Software Engineering SE3(1977)1 S.41-48

/91/ Encarnacao,J.: Logic Design Techniques and Tools. Siemens Forsch.-und Entwickl.-Ber. 7(1978)6 S.332-348

/92/ Ross,D.T.,Schoman,K.E.: Structured analysis for requirements definition. IEEE Transactions on Software Engineering SE3(1977)1 S.6-15

/93/ Ross,D.T.: Structured analysis(SA): A language for communicating ideas.
IEEE Transactions on Software Engineering SE3(1977)1 S.16-33

Anhang 1: GRIMBI - Datendefinitionssprache

Notation

< >	variable Sprachelemente (non-terminals)
::=	Substitutionsanweisung
|	trennt Alternativen
$\{\ \}_n^m$	Elemente zwischen {} dürfen n- bis m-mal wiederholt werden
{ }	darf entfallen
ABC	Sprachkonstanten (terminals)
/* */	Kommentarklammer

DDL-Syntax

```
<datadefinition> ::= ENTER DBANET(<dbname>) DBFILE(<dbfile>) <disp>
                           {DDN(<ddname>)} {STEXT(<stext>)};
                     <ddl-body>
                     END DBANET;

<disp>     ::= NEW(<read>,<write>,<update>) | OLD(<update>)

<ddl-body>::= <logic>           |  /* if NEW */
              <logic> <graphic> |  /* if NEW */
              <graphic>            /* if OLD */


<logic>    ::= <describe> {<describe>}0n
<describe>::= DESCRIBE 1 <problemtype> <systemtype>;

<problemtype> ::= CHAR(8)-string
```

```
<systemtype>::=
      BASEOBJECT {<standard-attribs>}
                 {<attriblist>}
                 <portlist>;
    | RELATION   {<standard-attribs>}
                 {FROMDOMAINE(<fromlist>)}
                 {TODOMAINE(<tolist>)}
                 {TYPE(<type>)}
                 {RESTRICTION(<restrictionlist>)}
                 {RESTRENTRY(<entry>)}
                 {<attriblist>}
                 {<tupelattrib>};
    | NET        {<standard-attribs>}
                 {NETDOMAINE(<net-problemtype-list>)}
                 {<attriblist>};
    | DESCRIPTOR {<standard-attribs>}
                 {SUBNETDOMAINE(<snobjtype-list>)}
                 {FIXPORTS}
                 {<attriblist>}

                 {,2 <portname> <snport-type>}_0^n;
    | SUBNETPORT {<standard-attribs>}
                 <porttype>
                 {FAN(<fanmin>,<fanmax>)}
                 {DIMENSION(<dim>)}
                 {<attriblist>};
    | SYSTEMPORT {<standard-attribs>}
                 <porttype>
                 {FAN(<fanmin>,<fanmax>)}
                 {DIMENSION(<dim>)}
                 {<attriblist>};
    | COMPLEX    {<standard-attribs>}
                 {2 <attrname> ATTRIBUTE

                   LIKE(<structure-object-attrib>)}_0^n
                {INSERT <structure-object-name> <complex-problemtype>

                                                  <properties>;}_2^n

               {CONNECT <port> <port> RELATION(<relname>,<value>);}_1^n
      END COMPLEX;
```

```
<complex-problemtype> ::= <baseobject> | <subnetport> | <relation>

<properties> ::= {<attrname>(<valuelist>)}0n
<standard-attribs> ::=  STEXT(<stext>)
                      | SUBTYPE(<objtype-list>)
                      | SUPERTYPE(<objtype-list>)


<objtype-list> ::= <problemtype> {,<problemtype>}0n

<net-problemtype-list> ::= <net-problemtype> {,<net-problemtype>}0n
<net-problemtype> ::= <baseobject-type> | <descriptor-type>
                     |<relation-type>   | <systemport-type> |<complex-type>


<snobjtype-list> ::= {<subnet-objtype>}1n
<subnet-objtype> ::=  <baseobject-type> | <descriptor-type>
                    | <relation-type>   | <complex-type>   | <subnetport-type>


<attriblist>   ::= 2 <attrib> {,2 <attrib>}0n

<attrib>       ::= <attrname> ATTRIBUTE <datatype>
                                        {DIMENSION(<dim>)}
                                        {VALUESET(<valueset>)}
                                        {DEFAULT(<default>)}
                                        {CLASS(<class>)}
                                        {REQUIRED}

<datatype>     ::=  BIN FIXED(15)   | BIN FIXED(31)
                  | BIN FLOAT(21)   | BIN FLOAT(53)
                  | CHAR(<n>)       | BIT(<n>)

<valueset>     ::= (<from> TO <to>) | (<v_1>,...,<v_n>)

<restrictionlist> ::=  <operand> <operation> <operand>

                      {,<operand> <operation> <operand>}0n
```

```
<operand>          ::=   FROM.<attrname> {(<ind>)}
                       | FROM..<attrname>{(<ind>)}
                       | TO.<attrname>   {(<ind>)}
                       | TO..<attrname>  {(<ind>)}
                       | REL.<attrname>  {(<ind>)}
                       | <const>

<operation>        ::= = | ¬= | < | > | <= | >=
<tupelattrib>      ::= 2 <tattrname> TUPELATTRIBUTE <datatype>
                                     {DIMENSION(<dim>)}
                                     {VALUESET(<valueset>)}
                                     {DEFAULT(<default>)}
                                     {CLASS(<class>)}
                                     {REQUIRED}

<portlist>      ::=   2 <portname> {,2 <portname>}_0^n

<port>          ::=   <portname> PORT <porttype>
                                 {DIMENSION(<dim>)}
                                 {FAN(<fanmin>,<fanmax>)}
                                 {<portattr-list>}

<porttype>         ::= IN | OUT | INOUT


<portattr-list>  ::=  3 <attrib> {,3 <attrib>}_0^n
```

Graphik-Teil

```
<graphic>          ::= OPEN GRAHIC(<grid>);

                        {<objtype-representation>}_1^n
                       CLOSE SCHEMA;
```

```
<objtype-representation>
      ::=  OPEN BASEOBJECT <problemtype>;
            <basegraphic>;
            <pick>;
            <portgraphic>;
            <portcoord>;
            <attribgraphic>;
            <textwindows>;
           CLOSE OBJTYPE;
         | OPEN OBJTYPE RELATION <problemtype>;
           <basegraphic>;
           <pickarea>;
           <textwindow>;
           TUPELGRAPHIC LINESTYLE(<linestyle>
                        TEXTWINDOW(<dx>,<dy>,<tlen>)
                        {FONT(<font>)}{SHEARING(<shearing>)}
           CLOSE OBJTYPE;
         | OPEN OBJTYPE NET <problemtype>;
           <basegraphik>;
           CLOSE OBJTYPE;
         | OPEN OBJTYPE DESCRIPTOR <name>;
           DESCRIPTOR; /*Beschreibt Deskriptorsymbol*/
            <basegraphic>;
            <attributgraphic>;
            <pickarea>;
            <textwindow>;

            {<desc-port-graphic>}_1^np
            /*np: Zahl der SUBNETPORT-Typen in SUBNETDOMAINE-Klausel*/
            PORTPOSITION <x> <y> <dx> <dy>;
           END DESCRIPTOR;
           SUBNET; /*Beschreibt Zeichnungs-Schriftfeld*/
            <basegraphic>;
            <attributgraphic>;
            <pickarea>;
            <textwindow>;
           END SUBNET;
          CLOSE OBJTYPE;
```

```
        | OPEN OBJTYPE <port-type> <problemtype>;
           <basegraphic>;
           <textwindow>;
           <attributgraphic>;
           <pickarea>;
          CLOSE OBJETTYPE;

   <port-type> ::= SYSTEMPORT | SUBNETPORT

   <desc-port-graphic> ::= DESCPORTGRAPHIC <subnetporttype>;

                           {<gr-object>}_1^n;
                           POSITIONPOINT <x> <y>;
                           ATTACHPOINT <x> <y>;
                           PICKAREA <x> <y> <dx> <dy>;
                          END DESCPORTTYPE;

<gr-repr> ::=  <pick>        | <portcoord>     | <basegraphic>
             | <portgraphic> | <attribgraphic> | <textwindow>
             | <tupelgraphic>


<pick>    ::= PICKAREA {<what> <where>}_1^n
<what>    ::= * | <portname>(<ind1>)
                | <portname>.<attr>(<ind1,ind2>)
                | .<attr>(<ind2>)

<where>   ::= <ix> <iy> <idx> <idy>
<portcoord>     ::= PORTCOORD <portname>(<ind1>) <ix> <iy>

                           {<portname>(<ind1>) <ix> <iy>}_0^n

<textwindow>    ::= TEXTWINDOW {<x> <y> <dx> <dy>

                              RASTER(<lines>,<cols>)}_0^n

<basegraphic>   ::= BASEGRAPHIC;

                     {<gr-object>}_1^n
                    END BASEGRAPHIC;

<portgraphic>   ::= PORTGRAPHIC <portname>(<ind1>);

                       {<gr-object>}_1^n
                    END PORTGRAPHIC;
```

```
<attribgraphic> ::= ATTRGRAPHIC <attrname>{(<ind)}
                        {<comp><value>,{<comp><value>});

                    {<gr-object>}1n
                    END ATTRGRAPHIC;
          <comp>::=  > | < | >= | <= | = | ¬=

<tupelgraphic>  ::= TUPELGRAPHIC LINESTYLE(<linestyle>
                                 TEXTWINDOW(<dx>,<dy>,<tlen>)
                                 {FONT(<font>)} {SHEARING(<shearing>)}

<gr-object> ::= <poly>  | <points> | <text> | <circle> | <gr-attrib>


<poly>      ::= POLY <poly-type> {<x> <y>}1n
<poly-type> ::= OPEN | CLOSE {HATCH(<alpha>,<dx>)}

<points>    ::= POINTS {<x> <y>}1n;
<text>      ::= TEXT <x> <y> <alpha> STRING(<string>);
<circle>    ::= CIRCLE <x> <y> <radius> <alpha1> <alpha2>
                           {HATCH(<alpha>,<dx>)};

<gr-attrib> ::= GRATTRIBUTE {<attrparm>}1n;
<attrparm>  ::= FONT(<font> | <SHEARING(<shearing>) | WIDTH(<width>)
              | HIGHT(<hight>) | LINESTYLE(<linestyle>)
              | GRAIN(<grain>) | SYMBOL(<symbol>)

<from>,<to>,<v_1>,<v_n>                 ::= Konstanten der erlaubten Typen

<ix>, <iy>, <idx>, <idy>, <font>,
<linestyle>, <t1>, <t2>, <grain>, <n>,
<dim>, <ind1>, <ind2>, <fanmin>,
<fanmax>, <grid>, <class>, <cols>       ::= INTEGER-Zahl 0...255

<x>, <y>, <dx>, <dy>, <shearing>,
<alpha>, <alpha1>, <alpha2>             ::= REAL-Zahl

<dbname>, <ddname>, <problemtype>,
<portname>, <attrname>, <read>, <write>,
<update>                                ::= CHAR(8)-string

<stext>, <string>                       ::= CHAR-string
```

Anhang 2: GRIMBI - Datenmanipulationssprache

Notation

< >	variable Sprachelemente (non-terminals)
<< >>	bei interaktivem Betrieb: PICK oder automatische Generation
::=	Substitutionsanweisung
|	trennt Alterativen
$\{\}_n^m$	Elemente zwischen { } dürfen n- bis m-mal wiederholt werden
{ }	darf entfallen
ABC..	Sprachkonstanten (terminals)

DML-Syntax

```
<dml-program> ::= ENTER DBMNET(<dbfile>,<logfile>)
                                <iotype>
                                {STATISTICS}
                                {LOGFILE(<logddn>)}
                                {TRACE}
                                {LOG};

                    {<dml-cmd>}1n
                  END DBMNET;

<iotype>      ::= BATCH | T4081

<dml-cmd>  ::=  <open>     | <close>      | <insert>      | <delete>
              | <connect>  | <disconnect> | <include>     | <exclude>
              | <contract> | <expand>
              | <detail>   | <abstract>   | <alternative> | <jump>
              | <search>   | <get-ops>    | PUSH          | POP
              | <copy>     | <dump>       | <change>      | <check>
              | <split>    | <join>
              | <window>   | <viewport>   | <transfer>    | <route>
              | <plot>
              | ACTIVATE | SAVE | RESTORE | COMPLETE
```

```
<open>      ::=  OPEN <open-obj>;
<open-obj>  ::=  BANK (<dbname>) {BANKDD(<ddn>)} {PASS(<pass>)}
                | NETPOOL(<npname>) {PASS(<pass>)}
                | NET (<netname>)

<close>     ::= CLOSE <close-obj> {LET};
<close-obj>::= BANK | NETPOOL | NET

<insert>    ::= INSERT <<objname>> <problemtype&>;

<problemtype&> ::=  NETPOOL          {SPACE(<n_objs>,<len_obj>,<incr>)}
                                     {STEXT(<stext>)}
                                     {PASS(<read>,<write>,<update>)}
                  | <net-desc-type>  {STEXT(<stext>)}
                                     {SPACE(<n_objs>,<len_obj>,<incr>)}

                                     {<property>}_0^n
                  | SUBNET(<subnet-name>)

                  | <probtype> {STEXT(<stext>)} {<property>}_0^n

<probtype>  ::= "Problemtyp vom DDL-Typ BASEOBJ, SUBNETPORT, COMPLEX,
                                        SYSTEMPORT, RELATION"


<property>  ::= {<attrname>(<valuelist>)}_1^n;

<valuelist> ::= <const> {,<const>}_1^n

<delete>    ::=    DELETE <delobj> {ALL};
<delobj>    ::=    NETPOOL(<netpoolname>) | NET(<netname>) | <<objname>>

<connect>   ::=  CONNECT <portref> <portref> REL(<relname> {<tupelvalue>};
                | CONNECT <<portref>> <<portref>> <<relname>> {<tupelvalue>};
<disconnect>::=   DISCONNECT <connection>;
<connection>::=   <<portref>> | <<portref>> <<portref>>
                | <<line>>    | <<rel-name>>
<portref>   ::=   <objname>.<portname>{(<ind>)}

<include>   ::=   INCLUDE <<sonname>> <<fathername>>;
<exclude>   ::=   EXCLUDE <<sonname>> {ALL};
```

```
<contract>  ::=  CONTRACT <<objlist>> TO(<descriptortype>)
                                          {<property>}0n
<expand>    ::=  EXPAND <<descname>>;

<detail>       ::=  DETAIL <<descname>> <detail-disp>;
<detail-disp>  ::=  NEW | OLD(<ind>)

<abstract>     ::=  ABSTRACT {INTO(<netname>)};

<alternative>  ::=  ALTERNATIVE <which>;
<which>        ::=  NEW | IND(<ind>) | NEXT | ACTIVE

<jump>         ::=  JUMP <<subnetportname>>;


<change>       ::=  CHANGE <<objname>> {<property>}1n;
                  | <ch-graphics>;

<search>       ::=  SEARCH  <s-opt> {PUSH} {FOUND(<bitvar>)};
                  | RSEARCH <r-opt> {PUSH} {FOUND(<bitvar>)};

<s-opt>        ::=  <direct> | <relative>

<direct>       ::=  NAME     (<objname>)
                  | KEY      (<objkey>)
                  | TYPE     (<problemtype>,<n>)

<relative>     ::=  SON      (<ind>)
                  | FATHER
                  | BROTHER  (<port>{(<ind>)})
                  | RELATION (<port>{(<ind>)})
                  | FROMOBJ  TUPEL(<ind>)
                  | TOOBJ    TUPEL(<ind>)

<r-opt>        ::=  NAME      (<relname>)
                  | KEY       (<relkey>)
                  | TYPE      (<reltype>)
                  | PORT      (<port>)
```

```
<get-ops> ::= OBJNAME(<name>) {KEY (<key-var>)}
                              {SYSTYPE(<systype-var>)}
                              {PROBLEMTYPE(<problemtype-var>)};
            | OBJKEY (<key>)  {NAME(<name-var>)}
                              {SYSTYPE(<systype-var>)}
                              {PROBLEMTYPE(<problemtype-var>)};
            | OBJIND (<ind>)  {NAME(<name-var>)}
                              {KEY(<key-var>)}
                              {SYSTYPE(<systype-var>)}
                              {PROBLEMTYPE(<problemtype>)};
            | OBJNUMB <of>    TO(<var>);
            | SGET({<portname>.}<attrname>{(<ind1>{,<ind2>})} TO(<var>);

  <of>     ::= ALL | SYSTYPE(<systype> | PROBLEMTYPE(<problemtype>)

<copy>     ::=  COPY <copy-opt>;

<copy-opt>::=  WITHIN <<obj-list>>
             | INTO <newnetname> <<objlist>>
             | FROM <fromnetname>

<spit>     ::= <<tupellist>>  /*bei interaktivem Betrieb*/
             | <tupellist> TO <newrelname>

<join>     ::= <<relname1>> {WITH} <<relname2>>

<restart> ::= RESTART <restart-file>;
```

Für interaktiven Betrieb:

```
PLOT <unit>; <unit> ::= DISPLAY | PLOTTER <dx>  | PDBFILE <pdbfile>

<ch-graphics>  ::=   CHANGE SYMBOLSIZE(<symsize>);
                   | CHANGE SHEETFORMAT;
                   | CHANGE SHEETSIZE <size>;
                   | CHANGE SHEETPOSITION <<newpos>>;

                   | CHANGE PLOTTYPE {<ptypes>}1n;

  <ptypes> ::= CONNECTIONS {(<relationtype-list>)} | NOCONNECTIONS
             | SYMBOLS {(<objtype-list)>} | NOSYMBOLS
             | ATTRIBS  | NOATTRIBS
             | TEXTS {(<text-window-number-list>)} | NOTEXTS
             | NORELATIONS | NONETS | ALL

<window> ::= WINDOW <wopt>;
      <wopt> ::= FACTOR <factor> | SHIFT <<w-pos>> | ALL | <<w-ll>> <<w-ur>>

<viewport> ::= VIEWPORT <vopt>;
         <vopt> ::= RESET | <<v-ll>> <<v-ur>>


<transfer> ::= TRANSFER {<<obj-name-list>>}1n
<rout>     ::= ROUTE <<connection>>;

<dump>     ::= DUMP SYMLIB {<symparm>}04
<symparm>  ::= PICKAREAS | TEXTWINDOWS | ATTRGRAPHIC |PORTCOORDS

<dbname>,<ddname>,<pass>,<npname>,<netname>,
<objname>,<read>,<write>,<update>,<subnet-name>,
<delobj>,<relname>,<portname>,<sonname>,<fathername>,
<descriptortype>,<descname>,<subnetportname>,
<problemtype>,<port>,<reltype>,<portname>,<attrname>,
<systype>,<subnet-name>,<new-subnet-name>,
<restart-file>,<logddn>,<logfile>,dbfile>          ::= CHAR(8)-Konstante

<stext>                                            ::= CHAR-string

<n_obj>,<len_obj>,<incr>,<ind>,<n>,<relkey>,
<objkey>,<symsize>                                 ::= INTEGER-Konstante
<...-var>,<var>                                    ::= Variable geeigneter Typen
```