465 | März 1986

SCHRIFTENREIHE SCHIFFBAU

Thomas Koch

Anforderungen an einen Standard für den Austausch schiffstechnischer Produktdaten über das Deutsche Forschungsnetz (DFN)

TUHH
Technische Universität Hamburg-Harburg

Anforderungen an einen Standard für den Austausch schiffstechnischer Produktdaten über das deutsche Forschungsnetz (DFN)

T. Koch, Hamburg, Technische Universität Hamburg-Harburg, 1986

Schriftenreihe Schiffbau
Schwarzenbergstraße 95c
D-21073 Hamburg

http://www.tuhh.de/vss

INSTITUT FÜR SCHIFFBAU DER UNIVERSITÄT HAMBURG

Bericht Nr. 465

Anforderungen an einen Standard für den Austausch schiffstechnischer Produktdaten über das Deutsche Forschungsnetz (DFN)

von

Thomas Koch

März 1986

ISBN 3 - 89220 - 465 - 9

Anforderungen an einen Standard für den Austausch schiffstechnischer Produktdaten über das Deutsche Forschungsnetz (DFN)

T. Koch

März 1986

Bericht Nr. 465

Inhaltsverzeichnis

1. Voraussetzungen und Möglichkeiten im DFN

Das Deutsche Forschungsnetz (DFN) wurde 1983 eingerichtet, um für Rechneranwender aus dem Wissenschaftsbereich Rechner-Kommunikationsdienste bereitzustellen.

Das DFN muß sich in seinen Protokollen, die für die verschiedenen Kommunikationsdienste erstellt werden, an den internationalen (ISO, CCITT), deutschen (DIN) sowie europäischen und amerikanischen (NBS, ANSI) Normen bzw. Normentwürfen orientieren. Im Bereich der Rechnerkommunikation existieren bereits einige Normentwürfe, von denen die wichtigste die ISO/DIS 7498 "Open Systems Interconnection Basic Reference Model" sein dürfte. Auf den Inhalt dieser Norm (die auch als OSI - Norm bezeichnet wird) wird im folgenden noch näher eingegangen. Zahlreiche Anforderungen an Netze und Aufgabenstellungen für Kommunikationsdienste sind jedoch noch weit von jeder Normung entfernt. Hier werden vom DFN vorläufige Protokolle und Definitionen entwickelt, die jedoch bei Inkrafttreten einer Norm u.U. angepaßt oder revidiert werden müssen.

Als Hardware wurde die Nutzung des DATEX-P Netzes der Deutschen Bundespost vorgesehen. Der Zugang ist für Universitäten daher leicht und kostengünstig realisierbar.

1.1. ISO/OSI Referenzmodell

Die bereits erwähnte ISO Norm definiert ein Referenzmodell für Kommunikationsvorgänge /Spies85, Kafka84/. Der Kommunikationsvorgang wird dabei auf mehreren Ebenen (Schichten) abgewickelt, wobei an den Endpunkten der Kommunikationsstrecke jeweils die entsprechenden Schichten Informationen austauschen. Das OSI - Referenzmodell definiert insgesamt 7 Schichten, zusätzlich wird die Kommunikations - Hardware (Schnittstellen, Modems, Leitungen) als nullte Schicht angesehen, auf der alle anderen aufbauen. Alle darüber angeordneten Schichten repräsentieren eine bestimmte hierarchisch angeordnete Funktion innerhalb der Kommunikationsabwicklung. Dabei ist wesentlich, daß durch entsprechenden Aufbau der einzelnen Schichten-Protokolle die Aufgaben einer Schicht völlig transparent für die übergeordneten Schichten sind, d.h. jede Schicht stellt für die nächsthöhere Leistungen bereit, die sie völlig selbständig erfüllen kann.

Die physikalische Schicht (physical control layer) stellt durch Schnittstellendefinitionen u.ä. die Möglichkeit zum Übertragen eines beliebigen Bitstromes bereit.

Die Datensicherungs- oder Verbindungsschicht (link control layer) enthält alle Einrichtungen zur Kontrolle und eine eventuelle Korrektur von Übertragungsfehlern. Dazu gehört auch eine Formatierung des Datenstromes z.B. in Form von Blöcken.

Die Netzwerk-Schicht (network control layer) organisiert die Wegfindung innerhalb des Netzwerkes.

Die Schichten 1 bis 3 können zusammengefaßt als Netz-orientiert (link) bezeichnet werden, da sie den Transport von Daten durch ein Netzwerk organisieren. Die X.25-Empfehlung beschreibt eine mögliche Realisierung der Schicht 3.

Die Transportschicht (transport end-to-end control layer) ist bereits unabhängig von der Art des Netzwerks, sie dient zur Datenflußkontrolle zwischen den einzelnen Sitzungen. Diese ist notwendig, da z.B. bei Netzen, die wie DATEX-P (der X.25-Empfehlung folgend) Daten in sogenannten Datenpaketen transportieren (und nicht zeichenweise), die fehlerfreie Ablieferung eines Pakets überprüft wird, es jedoch nicht ausgeschlossen ist, daß ein Paket verloren geht. Diese Kontrolle und die Initiierung einer erneuten Übertragung im Fehlerfall wird in der Transportschicht durchgeführt.

Die Sitzungsschicht (session control layer) dient dem Aufbau und der Erhaltung einer logischen Verbindung zwischen zwei Endanwenderprogrammen. Dabei wird auch die Form des Datenaustausches festgelegt.

Die Darstellungsschicht (presentation control layer) legt den Zeichensatz und die Datenkodierung und -formatierung fest, in der kommuniziert wird. Auch die Darstellung der Daten auf einem Drucker oder Terminal wird hier definiert. Auf diese Weise können z.B. erweiterte Bildschirmfunktionen verwendet werden. Hier kann bereits anwendungsbezogen unterschieden werden zwischen Dialogdaten, Filetransfer und ähnlichen Aufgaben. Eine Verschlüsselung der Daten kann ebenfalls in dieser Schicht durchgeführt werden.

Die Anwendungsschicht (process control layer) definiert die Kommandosprache eines Kommunikationsdienstes wie z.B. für Filetransfer oder RJE (Remote Job Entry).

Die Schichten 4 bis 7 sind Sitzungs-orientiert (session); die zugehörige Software ist für jeden Prozess jeweils einmal notwendig, während die Software für niedrigere Schichten nur einmal pro Netzanschluß benötigt wird.

Eine ebenfalls häufig anzutreffende Gliederung der Schichten bezeichnet die unteren vier Schichten als 'transport system' während die Schichten 5, 6 und 7 unter dem Begriff 'session services subsystem' zusammengefaßt werden /Mart81, S.165/.

1.2. Kommunikationsprotokolle

Das OSI-Referenzmodell behandelt nur die Gliederung der einzelnen Stufen des Kommunikationsvorganges. Wie die einzelnen Schichten konkret realisiert werden, muß in Protokollen festgelegt werden.

1.2.1. X.25-Empfehlung

Für das Transport-(Sub-)system existieren bereits eine Reihe allgemein verbreiteter und akzeptierter Protokolle, so die bereits erwähnte X.25-Empfehlung /Haas84/, die zu den bisher wichtigsten Normen gehört.

X.25 definiert die Schnittstelle zwischen einem offenen Daten-Paket-Vermittlungsnetz (wie DATEX-P) und den Benutzerrechnern, die das Netz nutzen. Ein Rechner, der das X.25 Protokoll erfüllt, übergibt die Daten in Paketform an das Netz-Interface (Modem). Dieses Paket enthält außer den zu übertragenden Daten zusätzliche Informationen in sogenannten Headern, deren Inhalt in X.25 spezifiziert wird. Diese Informationen geben das Format des Headers, logische Kanalnummer, Pakettyp, Rechner-ID und andere für die Netzvermittlung (gemäß Schicht 3) relevante Daten an. Die eigentlichen zu übertragenden Daten stammen aus höheren Schichten und werden völlig unverändert weitergegeben. Die Anwenderdaten werden so von den Headern der tiefer liegenden Schichten 'umhüllt', in der Literatur wird oft das Bild der russischen Holzpuppe als Bild dafür gewählt.

Einige Charakteristika des X.25-Protokolls sind /Kafka84, S.33/:

- Datenübertragungsgeschwindigkeit im Bereich von 50 Bit/s (Einheit: Baud, Zeichen: Bd) bis 48 kBd.
- Mehrfachnutzung von physikalischen Anschlüssen durch logische Kanäle (max. 4096 Kanäle pro Anschluß).
- Sicherung der Übertragung durch Fehlerkontrolle im Netz.
- fortlaufende Nummerierung der Datenpakete zur Sicherstellung der korrekten Reihenfolge

Ein weitergehendes Protokoll ist durch die S.70-Empfehlung (Grundlage z.B. des TELEFAX-Dienstes) definiert worden, die die vierte Schicht einschließt und auf X.25 aufbaut. Auch diese Empfehlung wird bei verschiedenen DFN-Kommunikationsdiensten berücksichtigt.

1.2.2. X.28-Empfehlung

Die X.25 basiert auf Paketdatentransfer. Diese Form von Datenaufbereitung kann zwar inzwischen vollständig durch Hardware (VLSI-Chips) realisiert werden, oft jedoch wird die Blockung/Entblockung von Daten durch entsprechende Software durchgeführt werden müssen. Auf zahlreichen Rechenanlagen existiert diese Software bereits, zumindest ist sie relativ leicht implementierbar. Der Prozessorzeitaufwand ist allerdings insbesondere bei kleinen und kleinsten Rechnern nicht zu vernachlässigen. Aus diesem Grund gibt es eine Vielzahl von Terminals die mit einer einfachen seriellen Schnittstelle ausgerüstet sind (dieses sind auch die Standardgeräte, die an Rechenanlagen angeschlossen werden). Diese Geräte unterstützen nur ein zeichenorientiertes Start/Stop-Protokoll und können keine X.25-Schnittstelle realisieren.

Derartige Terminals lassen sich über ein PAD (Packet Assembly/Disassembly) an ein X.25-Netz anschließen. Die Arbeitsweise des PAD wird in der X.3-Empfehlung spezifiziert. Ein PAD stellt eine Art Datenkonvertierer dar, der die einzelnen Zeichen in Paketen sammelt und gemäß dem X.25 Protokoll an das Netz weitergibt.

Das zeichenorientierte Protokoll zwischen einem PAD und einem Start/Stop-Terminal ist in der X.28 Empfehlung beschrieben. Das X.28 Protokoll geht über das einfache Start/Stop-Protokoll hinaus, da bestimmte Parameter wie Zeichenecho, Eingabeende-Zeichen, Verzögerungszeiten, Übertragungsgeschwindigkeit, Start/Stop-Kontrolle (flow control) usw. vom Terminal aus gesetzt werden können.

Mit Hilfe des X.28-Protokolls ist es möglich, Dialogstationen direkt an das Netz zu koppeln und interaktiv über das Netz auf entfernten Rechnern zu arbeiten.

1.2.3. Virtual File

Unter dem Begriff 'Virtual File' laufen international mehrere Normungsbestrebungen, die das Arbeiten mit Dateien standardisieren sollen. Bisher unterscheiden sich alle Betriebssysteme erheblich in den Konventionen, die die Namensgebung von Dateinamen oder -bezeichnungen und den Zugriff auf Dateien betreffen. Das 'Virtual File' definiert

eine genormte Dateinamensgebung und einen Satz von Standardoperationen, mit denen eine Datei manipuliert werden kann (z.B. Open, Close, Read usw.).

Insbesondere für die Nutzer von Datennetzen ist ein solcher Standard von großem Interesse, da Dateien im Netz dann alle als Virtual Files transportiert und angesprochen werden können. Da der Virtual File-Standard die Formatierung von Daten beschreibt, wäre damit ein Protokoll der sechsten Schicht des OSI-Modells realisiert.

Für existierende Betriebssysteme muß dann ein Interface bereitgestellt werden, welches Dateinamen und interne Dateioperationen auf die entsprechenden Eigenschaften des Virtual File abbildet. Zukünftige Betriebssysteme könnten natürlich das Virtual File direkt unterstützen, so daß ein Interface entfallen kann.

1.2.4. Virtual Terminal

Noch weiter divergierende (Nicht-)Standards existieren im Bereich der Terminalsteuerung. Fast alle Terminals unterstützen heute eine Vielfalt von mehr oder weniger intelligenten Zusatzfunktionen bei der Bildschirmdarstellung. Diese Funktionen werden vom Rechner aus über sogenannte 'Kontrollsequenzen', d.h. Zeichenfolgen, die nicht druckbare Zeichen enthalten, genutzt. Leider ist es bisher nicht möglich gewesen, diese Kontrollsequenzen zu standardisieren; es gibt daher fast so viele verschiedene Methoden wie Terminalfabrikate und -typen existieren.

Um im Netz Terminals verschiedenster Art über unterschiedliche Rechenanlagen oder direkt ansprechen und dabei erweiterte Terminalfunktionen nutzen zu können, ist es erforderlich, eine Norm für diese Funktionen und ihre Kommandosequenzen zu schaffen. Die realen Terminals werden über entsprechende Interfacesoftware angesteuert, die die Kommandosequenzen umsetzt. Wenn eine solche Norm verabschiedet worden ist, wird es natürlich auch Terminals geben, die als Virtual Terminal direkt angesprochen werden können.

Auch das Virtual Terminal Konzept stellt ein Protokoll für die 6. Schicht im OSI-Modell dar.

1.2.5. Einheitliche höhere Kommunikationsprotokolle (EHKP)

In der Bundesrepublik werden alle Normungsvorhaben der OSI-Schichten 4 - 7 in dem Projekt EHKP zusammengefaßt. Virtual File, Virtual Terminal aber auch Nachrichtenaustausch (CMBS) sowie Batch-Jobs (RJE) gehören zu diesen im EHKP unterstützten Kommunikationsdiensten.

Die EHKP-Definitionen werden in Einklang mit den anderen internationalen Normungsaktivitäten in diesem Bereich durchgeführt. Die DFN-Protokolle werden sich mit Sicherheit auch an den Ergebnissen der EHKP-Definitionen orientieren. Es ist jedoch noch nicht abzusehen, wie lange eine endgültige Festlegung dauern wird, so daß auf jeden Fall vorläufige Überlegungen im Bereich des DFN notwendig sind.

1.2.6. Grafische Kommunikationsprotokolle

Der Betrieb von grafischen Geräten wie Terminals, Workstations und Plottern am Netz bringt ähnliche Probleme wie der Betrieb von alphanumerischen Terminals und Druckern. Allerdings bestehen einige Unterschiede: Die Vielfalt der Geräte ist nicht ganz so groß, andererseits ist die Anzahl der möglichen Funktionen um Größenordnungen höher.

Normungsbestrebungen bestehen auf dem Gebiet der Grafik schon seit einiger Zeit, so daß hier schon wesentlich konkretere Vorstellungen bestehen. Um grafische Endgeräte direkt ansprechen zu können, wird eine Schnittstelle genormt, die die Bezeichung 'Virtual Device Interface' (VDI) hat.

Das VDI ist in seiner Konzeption auf andere bereits existierende Normenentwürfe wie VDM, GKS, CORE (s.u.) abgestimmt, die geräteunabhängige Standards für grafische Datenaufbereitung darstellen.

Im DFN werden sehr ehrgeizige Projekte im Bereich der Grafik-Kommunikationsdienste verfolgt /DFNG84/. Es sind dies:

- Grafischer Dialog: Es soll ermöglicht werden, einen Rechner bzw. ein intelligentes Grafikterminal als GKS-Workstation zu betreiben.

- Kommunikation zwischen Modelliersystemen: CAD-Systeme sollen untereinander auf unterschiedlichste Art auf mehreren Rechnern verknüpft arbeiten können.

- Transfer grafischer Daten: Auf der Basis des Filetransfer-Dienstes sollen Dateien, die grafische Daten enthalten, ausgetauscht werden.

Die einfachste Stufe stellt der letzte Punkt dar: Der Filetransfer beliebiger Dateien ist bereits möglich /DFNP84/. Der wesentliche, noch offene Punkt ist, wie die Datenformatierung in diesen Dateien festzulegen ist. Dies ist jedoch in erster Linie ein anwendungsbezogenes Problem, bei dem die Benutzer entsprechende Anforderungen definieren müssen. Der Austausch schiffstechnischer Daten, der im

folgenden untersucht werden soll, gehört auch in diesen Bereich.

Die Nutzung von grafischen Endgeräten wird im zuerst genannten Punkt behandelt werden. Dabei sind außer den Fragestellungen, die durch das VDI behandelt werden, noch weitergehende Fragen zu klären /DFNG84, S.35ff./.

Die Interaktion von verteilten Modelliersystemen (CAD, CAE usw.) soll in mehreren Stufen realisiert werden /DFNG84, S.44 ff./. Dazu bedarf es allerdings der Festlegung einer Anwenderschnittstelle für Modelliersysteme, die z.Z. noch nicht in Sicht ist. Diese Schnittstelle könnte auf den Definitionen des Filetransfers aufbauen.

1.3. Einrichtungen des Netzes

DATEX-P ist ein öffentliches Paketdatennetz, das von der Deutschen Bundespost betrieben wird. Es ist international an weitere Paketdatennetze angeschlossen. (In diesem Bereich sind jedoch die Protokolle des DFN vorläufig nicht anwendbar.)

Der Zugang erfolgt über einen gemieteten Netzanschluß, außerdem wird eine Identifikationsnummer (NUI) vergeben /DATEXP, Koch85/. Es gibt verschiedene Arten des DATEX-P Anschlusses:

- Akustikkoppler (300 Bd)
- DATEX-P20H (1200 Bd mit Postmodem)
- DATEX-P10H (X.25 Schnittstelle bzw. PAD beim Teilnehmer)

Für Teilnehmer des DFN wird in erster Linie der P10 Hauptanschluß verwendet werden, da die Rechenanlage meist über eine X.25-Schnittstelle verfügt oder ein PAD-Rechner (z.B. in Rechenzentren von Universitäten) vorhanden ist bzw. beschafft werden kann.

PAD's sind für viele neuere Rechner (vor allem Systeme, die den Dialogbetrieb unterstützen) als Hardwarekomponenten vorhanden, oder aber durch entsprechende Systemsoftware emulierbar (Software-PAD). Batch-orientierte Betriebssysteme unterstützen den Netzbetrieb oft nur unzureichend oder gar nicht, interaktive Betriebssysteme wie VAX/VMS, PRIMOS und natürlich UNIX integrieren, zumindest in ihren neueren Versionen, den Netzbetrieb in erheblichem Maße. Dies gilt nicht nur für die unteren Schichten des Transport-Systems, oft werden bestimmte Kommunikationsdienste (Filetransfer bzw. Filezugriff, Remote-Batch-Jobs) auf der Anwendungs-

schicht unterstützt, allerdings mangels Normung z.T. auf recht unterschiedliche Art.

Wesentlicher Kostenfaktor im Betrieb eines Netzes ist die Nutzungsdauer. Die innerhalb einer Zeiteinheit zu übertragende Datenmenge hängt dabei von der Baudrate (Übertragungsgeschwindigkeit) des Anschlusses ab. Die zu veranschlagende reale Übertragungszeit (einschließlich Verzögerungen im Netz) kann ungefähr nach folgender Faustformel bestimmt werden /Rein81, S.55; DFNM85, S.10/:

$$t_{Ü} = 8.3 \; / \; \text{Baudrate} * n_{Zeichen}$$

Für die Übertragung einer Datei von z.B. 100 kByte bei einer Übertragungsrate von 4800 Bd ergibt sich eine Zeitdauer von ungefähr 177 Sekunden.

2. Schiffstechnische Produktdaten

Der Entwurf, die Konstruktion und die Fertigungsplanung von Schiffen und meerestechnischen Produkten beinhalten die Erstellung von sehr großen Datenmengen, die im Zuge der zunehmenden Verwendung von Rechnern im Konstruktionsprozess immer häufiger in maschinenlesbarer Form vorliegen. Die Einführung weiterer rechnergestützter Methoden (zusammengefaßt als CAD-Methoden bezeichnet) verstärkt diesen Trend erneut. Die zukünftige weite Verbreitung von Rechnerarbeitsplätzen in Werften, Ingenieurbüros und Universitäten bewirkt einen Bedarf an Kommunikationsmöglichkeiten zwischen Rechenanlagen, wie sie durch das DFN theoretisch angeboten werden können.

Eine erste wichtige Stufe ist dabei der Austausch von Informationen in Form von Dateien. Der Kommunikationsdienst Filetransfer kann hier problemlos eingesetzt werden. Allerdings werden von Seiten des DFN keine Definitionen bezüglich des Inhalts dieser Dateien bereitgestellt, die Festlegung eines solchen Dateiformats ist ein reines Anwenderproblem.

Es ergibt sich daher die Notwendigkeit, einen Standard zu definieren, der ein Dateiformat festlegt, das den gesamten Bedarf des Austausches von schiffstechnischen Produktdaten abzudecken vermag.

Es soll zunächst untersucht werden, welche Anforderungen an eine solche Definition gestellt werden. Das gesamte Spektrum schiffstechnischer Daten läßt sich in Teilbereiche aufteilen, die jeweils bestimmte Daten zusammenfassen. Bisher wurden drei solche Bereiche im CAD-Arbeitskreis des FDS festgelegt:

- die Schiffsformdaten-Schnittstelle (SFS),
- das Schiffszeichnungsformat (SZF) und
- das Schiffbaufertigungsformat (SFF).

Es ist zu prüfen, ob diese Bereiche jeweils einzelne Datenformatdefinitionen benötigen, oder ob andere Lösungen gewählt werden können.

2.1. Allgemeine Anforderungen

Im folgenden werden einige grundsätzliche Forderungen aufgeführt, die in einem Standard für grafischen Datenaustausch beachten werden sollten.

2.1.1. Platzbedarf

Grafische Daten sind sehr umfangreich. Daher sind beim Transport durch ein Netz hohe Übertragungskosten zu erwarten. Diesen kann durch entsprechende Wahl des Datenformates begegnet werden. Es gibt hier mehrere Ansatzpunkte:

Übertragung binärer Daten anstelle von Klartext-Daten. Da binäre Daten nicht mehr lesbar sind, müssen zusätzliche Überprüfungsmöglichkeiten vorgesehen werden (Quersummen von Zahlenwerten, CRC-Prüfworte usw.). Eine Verwendung von binären Datenformaten erbringt eine Reduktion von bis zu 60% /IGES83/ bei numerischen Daten. Keine Reduktion ist bei Textdaten zu erwarten.

Eine weitere Möglichkeit ist die Verwendung optimierender Codierungen (Hufman-Code usw.) /Kell86/. Es gibt hier mehrere Methoden, von denen einige die Möglichkeit zur Fehlerüberprüfung einschließen. Die Verwendung solcher Codes erbringt bei anwendungstypischen Daten bis zu 70%, wobei numerische Daten und Texte gemischt sein können.

Es ist zu erwarten, daß von Seiten des DFN anwendungsunabhängige Datenkomprimierungsmethoden bereitgestellt werden, es lagen jedoch keine Informationen dazu vor.

2.1.2. Zeichensatz

Es sollte der volle ASCII-Zeichensatz einschließlich nicht druckbarer Zeichen unterstützt werden. Im ASCII-Zeichensatz nicht enthaltene Zeichen des EBCDIC-Zeichensatzes können nach einer Tabelle z.B. auf unübliche nicht druckbare Zeichen abgebildet werden (optional). Obwohl der Zeichensatz im Netz definiert sein wird, besteht die Möglichkeit, daß spezielle Zeichen aus nicht-ASCII-Zeichensätzen benötigt werden.

2.1.3. Erweiterungen

Die Standardisierung kann nur den aktuellen Stand der Entwicklung berücksichtigen. Daher ist es notwendig, die Möglichkeit zu Erweiterungen bereits einzuplanen. Da nicht alle denkbaren Eigenschaften standardisiert werden können, sind zusätzlich Vorkehrungen für benutzerspezifische Erweiterungen zu treffen. Viele neuere Normen (GKS, CORE, VDM) besitzen solche Möglichkeiten.

2.1.4. Symbolische Daten

Alle Daten sollten symbolisch, d.h. mit Hilfe eines Namens, darstellbar sein. Inwieweit verschiedene Datentypen zu unterscheiden sind, muß im Einzelfall entschieden werden. Die möglichen Namen sollten mindestens 8 bis 10 signifikante Zeichen enthalten. Groß- und Kleinschrift sollten unterschieden werden und einige nicht-alfanumerische Sonderzeichen wie z.B. '$' und '_' zugelassen sein, um die Kompatibilität zu allen wichtigeren Programmiersprachen zu erhalten.

2.1.5. Kompatibilität zu anderen Normen

Es existieren insbesondere im Bereich der grafischen Datenverarbeitung schon zahlreiche Normen (s.o.), die allgemein akzeptiert worden sind und sich daher vermutlich durchsetzen werden. Es ist daher unbedingt erforderlich, die Kompatibilität zu diesen Normen durch entsprechende Festlegungen zu erhalten, oder aber Schnittstellen zu diesen Normen vorzusehen.

2.1.6. Editierbarkeit

Es ist wünschenswert, an einer Datei, die grafische Daten enthält, mittels eines Texteditors Änderungen vornehmen zu können. Diese Forderung steht jedoch im Widerspruch zu Anforderungen des geringen Platzbedarfes, da Klartext-Codierung erforderlich ist und eine möglichst übersichtliche Darstellungsweise gewählt werden muß. Ein weiterer zu beachtender Punkt ist die Integrität der Datei. Falls Änderungen von Hand durchgeführt werden, muß die gesamte Information in der Datei noch immer konsistent sein.

2.1.7. Sequentielle Bearbeitung

Einige Rechner verfügen nicht über ausreichenden Speicherplatz, um alle Dateien problemlos vollständig verarbeiten zu können. Aus diesem Grund sollte das Dateiformat sequentiell verarbeitbar sein, um eventuell nicht benötigte Informationen überspringen zu können und unnötige Zwischenspeicherungen zu vermeiden.

Diese Forderung ist allerdings nicht von so großer Bedeutung; sie ist außerdem im allgemeinen nur mit großen Schwierigkeiten erfüllbar.

2.2. Schiffsformdaten-Schnittstelle (SFS)

Die SFS soll alle Daten beinhalten, die zur Darstellung und Generierung von Schiffskörperoberflächen und anderen Strömungskörpern benötigt werden. Die Darstellung dieser Daten erfolgt in einem dreidimensionalen Koordinatensystem, dessen Nullpunkt beliebig wählbar sein muß. Zur Generierung werden verschiedene geometrische Objekte benötigt:

- Koordinatensystem: es wird ein dreidimensionales kartesisches Koordinatensystem benötigt, in dem die Positionen aller Objekte durch ein Koordinatentripel anzugeben sind.

- Punkte: Objekte, die eine Position im Raum durch ein Koordinatentripel beschreiben.

- Linien: dem mathematischen Begriff 'Strecke' entsprechende Objekte, die zwei Punkte (zwei Koordinatentripel) durch einen Linienzug verbinden.

- Kreise, bzw. Kreisbögen: diese Objekte lassen sich auf verschiedene Weise beschreiben: durch Angabe von vier der Größen Mittelpunkt, Radius, Startpunkt, Endpunkt, Anfangswinkel, Endwinkel und Winkeldifferenz (es gibt entsprechend viele Möglichkeiten der Definition).

- andere analytische Kurvenabschnitte wie Parabel, Hyperbel usw.

- interpolierende Kurven: Polynome, Splines verschiedener Art. Es sind Stützpunkte anzugeben, ggf. auch die Koeffizienten der Polynome.

- Rotationsflächen: durch Rotation einer Kurve um eine Achse erzeugte Fläche.

- interpolierte Flächen: durch ein geordnetes (meist orthogonales Netz von zweidimensionalen Polynomen oder durch dreidimensionale Polynome (Spezialfälle: Splines, Coon'sche Flächen) definierte Flächen.

- Tangentialebenen: Angabe eines Tangentialebenen-Normalenvektors an einem Koordinatenpunkt.

- Krümmungsvektor: Angabe der Krümmung einer Kurve an einem Punkt durch einen Vektor, der die Position und den Abstand des Krümmungsmittelpunktes (Krümmungsradius) beschreibt.

Neben den rein geometrischen Objekten sind weitere Informationen erforderlich oder wünschenswert:

- geometrische Transformation: die Objektkoordinaten können durch eine Transformationsmatrix verschoben, rotiert und scaliert werden. Diese Fähigkeit erlaubt insbesondere im Zusammenhang mit entsprechender Strukturinformation (Segmente u.ä., s.u.) leistungsfähige und platzsparende Operationen.

- Viewports und Windows: durch Angabe von Viewportdimensionen und Fenster läßt sich die grafische Darstellung in einer Zeichnung spezifizieren. Dies ermöglicht z.B. die mehrfache Darstellung eines Körpers in verschiedenen Ansichten, wobei die Körperdefinition nur einmal benötigt wird. Für bestimmte häufig auftretende Darstellungen (wie z.B. Linienriß, Generalplan usw.) lassen sich Standarddefinitionen festlegen.

- allgemeine Informationen: Zeichnungsbeschriftung, Projektdaten, Copyright usw.

- Strukturierung der Daten: Informationen zur Verknüpfung von Daten zu übergeordneten Einheiten, Zuordnung von Prioritäten und ähnlichen Bewertungs- und Steuerungsgrößen. Damit ist es möglich, mehr Informationen aus den übertragenen Daten zu erhalten, die eine Weiterverarbeitung erleichtern. Die Verknüpfung kann auf mehrfache Weise erzielt werden: Segmente (im Sinne von GKS), Macro's, Directories u.ä.

2.3. Schiffszeichnungsformat (SZF)

Im Schiffszeichnungsformat sollen alle Informationen, die in den Konstruktionszeichnungen enthalten sind, darstellbar sein. Dabei ist zu beachten, daß nicht die Zeichungen als Endprodukt übertragen werden sollen (jedenfalls nicht ausschließlich), sondern vielmehr die darzustellenden Produkte vollständig in ihrer Geometrie, Topologie und weiteren allgemeinen Eigenschaften beschreibbar sein sollen, d.h. auch in einer Form, die von Modelliersystemen verarbeitet werden kann. Es ist zu erwarten, daß das SZF eine zentrale Rolle beim allgemeinen Austausch von Daten (z.B. auf dem Gebiet der Forschung) spielen wird.

Das SZF benötigt außer den Möglichkeiten der SFS noch weitere Eigenschaften:

- verschiedene Koordinatensysteme: Zusätzlich zum dreidimensionalen kartesischen Koordinatensystem sind weitere Systeme sinnvoll: Zylinderkoordinaten (z.B. Propellerentwurf) und sphärisches Koordinatenssytem

(hydrodynamische sowie elektrotechnische Probleme). Modifikationen dieser Systeme lassen sich durch entsprechende Transformationen definieren.

- Finite Elemente: Knotendaten und Elemente verschiedener Typen, Materialdaten. Die Informationen bedingen eine Verknüpfung von grafischen und nicht-grafischen Daten.

- Achsendarstellungen: ein sehr oft benötigtes grafisches Element sind Achsen, die die Erstellung von Diagrammen und Kurvendarstellungen ermöglichen. Da sich diese Objekte durch eine geringe Anzahl von Parametern beschreiben lassen, erscheint es sinnvoll, sie als Standardobjekt vorzusehen. Eine alternative Möglichkeit wäre die Definition durch Macro's.

- Detailinformationen: In wesentlich umfangreicherem Maße als bei der SFS sind Beschriftungen, Bemaßungen, Materialdaten und Fertigungsinformationen erforderlich. Es ist daher notwendig, ein einheitliches System (wie es z.B. bei der symbolischen Darstellung von Schweißnahtformen besteht) für verschiedene derartige Daten festzulegen.

- Datenbankinformationen: die vom DFN angestrebte Zusammenarbeit von Modelliersystemen erfordert auch den Austausch von CAD-Datenbankinformationen. Diese Aufgabe muß für schiffstechnische Programme innerhalb des SZF erfüllbar sein. Dabei werden in wesentlichem Umfange auch nicht-grafische Informationen übertragen.

2.4. Schiffsbaufertigungsformat (SFF)

Im SFF werden weitere vor allem nicht-grafische Daten benötigt:

- Materialinformationen: Es gibt genormte Werkstoff-Kennzahlen, die die Materialart beschreiben. Weitere benötigte Daten sind jedoch Halbzeugart (z.B. Platte, Profil, Rohr usw.), Bestellinformationen, Korrosionsschutz usw. Bei vielen dieser Daten existieren bereits Kennzahlensysteme, die von Branchenverbänden u.ä. Organisationen festgelegt wurden. Diese Systeme, die oft auch bei Bestellungen verwendet werden können, sollten berücksichtigt werden.

- Bearbeitungsinformationen: Art und Typ der benötigten Werkzeugmaschinen, Bearbeitungsdauer, besondere Anforderungen.

- numerische Steuerung (NC): Daten zur Bedienung von NC-Brennschneidmaschinen (hier existieren Hersteller-

normen, die sich bereits sehr weit durchgesetzt haben), NC-Bearbeitungsmaschinen (Drehbänke, Fräsen usw.) und Handhabungsautomaten ('Roboter'). Bei allen diesen Maschinen gibt es noch keine allgemein akzeptierten Standards, allerdings gibt es zahlreiche Ansätze, die im Bereich der Handhabungsautomaten schon zu Realisierungen von einer Anzahl von 'Steuersprachen' geführt haben, z.B. /Blum84/.

- Arbeitsplanung: Stücklisten, Bearbeitungsfolgen, Netzpläne. Eine Standardisierung bedingt eine genaue Kenntnis der in unterschiedlichen Unternehmen gebräuchlichen Methoden und Organisationsformen für diesen Bereich.

- Explosionszeichnungen und Zusammenbauanleitungen, Wartungsinformationen (bestehend aus Zeichungen und Text). Diese Daten bedingen umfangreiche Informationen über die Struktur des beschriebenen Produkts.

- Produktionsautomation: dieses Gebiet, das alle mit der Fertigung von Produkten verbundenen Vorgänge umfaßt, spielt bisher speziell im Schiffbau noch keine beachtenswerte Rolle. Es ist trotzdem darauf zu achten, einen etwaigen SFF-Standard im Hinblick auf zukünftige Entwicklungen zu definieren. Es gibt mehrere internationale Normungsaktivitäten (z.B. MAP), die Einfluß auf die Entwicklung der Automation haben werden. Z.Z. besteht jedoch in der Schiffbau-Branche kein Bedarf zum Austausch solcher Daten.

Wie aus den genannten Punkten zu ersehen ist, ist die Fertigungsautomation noch nicht soweit gediehen, daß bereits detaillierte Informationen vorliegen bzw. Normentwürfe bekannt sind. Es gibt jedoch zahlreiche Bestrebungen; die für das SFF wesentliche dürfte die auf den Erkenntnissen der IGES-Norm aufbauende, noch in der Entwicklung befindliche STEP-Norm sein, über die keine genauen Unterlagen zur Verfügung standen.

Bei den Überlegungen zur Definition eines SFF ist in der jetzigen Situation zu beachten, daß eine vorläufige 'Eigenkonstruktion' ohne Berücksichtigung anderer Standards zum Abweichen von der internationalen Entwicklung führen kann, andererseits ein Abwarten internationaler Normungen erhebliche Verzögerungen bei der Realisierung des SFF zur Folge hat.

3. Bestehende Normentwürfe

Im folgenden sollen die wichtigsten bereits vorhandenen Normentwürfe kurz dargestellt und im Hinblick auf ihre Verwendbarkeit für die SFS, das SZF und SFF untersucht werden. Dabei sind die nicht anwendungsbezogenen (meist internationalen) Normentwürfe und die bereits existierenden Vorschläge für die schiffstechnischen Schnittstellen-Definitionen /SFSB85/ und für die Produktdefinition /IGES83/ zu unterscheiden.

3.1. GKS

GKS ('Grafisches Kern System' oder 'Graphics Kernel System') stellt einen Standard für Anwendungsprogramme dar. Es stellt dem Programmierer, der grafische Probleme lösen will, einen umfangreichen Vorrat von Funktionen zur Darstellung und Beschreibung von grafischen Objekten zur Verfügung. GKS stellt also eine genormte Schnittstelle zu Anwendungsprogrammen her /GKS82, Hopg85/.

GKS erzeugt darstellbare Ansichten der Objekte in normalisierten Gerätekoordinaten ('normalized device coordinates' (NDC)), die von der jeweiligen hardwareabhängigen Treiber-(System)software auf das entsprechende Endgerät abgebildet werden können.

GKS ist in der derzeitigen aktuellen Version (7.2) beschränkt auf zweidimensionale Bilder. Es ist geplant, einen weiteren über GKS angeordneten Standard zu definieren, der die entsprechenden 3D-Funktionen enthält. Die Beschränkung auf 2D-Bilder wurde absichtlich vollzogen: die Darstellung dreidimensionaler Objekte ist bereits im Bereich der Modelliersoftware anzusiedeln, einem Bereich, in dem noch keine klaren Vorstellungen über allgemeine Konzepte herrschen, so daß die Definition eines 3D-Standards unweigerlich eine Fixierung bedeutet hätte. 3D-Darstellungen benötigen z.B. Funktionen für die Projektion von Objekten, Schattenwurf, Oberflächenhelligkeit, Sichtbarkeit usw.; dies alles sind derzeit noch Probleme der Forschung, auch wenn es bereits eine Vielzahl von Lösungen gibt.

Es werden sechs grafische Primitiva im GKS unterstützt: polyline, polymarker, text, fill area, cell array und die 'generalized drawing primitives' (GDP), die eine Vielzahl von grafischen Objekten (Kreise, Kreisbögen usw.) darstellen können, sofern diese von der Geräte-Hardware erzeugt werden können.

Zur Strukturierung der Objekte dient das Segment-Konzept: Ein Segment kann eine (theoretisch) beliebige Anzahl von grafischen Primitiva enthalten, die für zahlreiche Operationen gemeinsam angesprochen werden können.

Ein wesentliches Konzept im GKS ist das der logischen Workstation. Jede Workstation ist ein Ein- und/oder Ausgabegerät, das jedoch unterschiedlichster Art sein kann. Jeder Workstation sind entsprechende Eigenschaften zugeordnet. Es gibt sechs Typen von Workstations: nur für Ausgabe (mit nur einem Gerät pro Workstation), nur für Eingabe (mit beliebig vielen Eingabegeräten), Eingabe und Ausgabe (entsprechende Bedingungen), Metafile-Ausgabe (zum Schreiben einer geräteunabhängigen Grafikdatei (Metafile)), Metafile-Eingabe (zum Lesen dieser Datei) und die 'workstation independent segment storage' (WISS) Workstation, die ausschließlich zum Speichern von Segmenten dient und der kein physikalisches Ein- oder Ausgabegerät zugeordnet ist.

Das Anwenderprogramm verwendet unter GKS im Normalfall als Koordinatensystem Weltkoordinaten ('world coordinates (WC)), d.h. die Koordinaten, die die Anwendung in der Realität beschreiben. Mit Hilfe einer Transformation werden diese Koordinaten vom GKS auf NDC umgerechnet. Die NDC stellen ein karthesisches Koordinatensystem dar, das auf beiden Achsen das Intervall [0,1] darstellt. Diese Koordinaten, die geräteunabhängig sind, werden in einer weiteren Transformation auf die Gerätekoordinaten ('device coordinates' (DC)) umgerechnet, wobei diese Transformation nur einige Möglichkeiten erlaubt, wie Vergrößern und Verkleinern des Bildes, Ausschnittdarstellung usw.

Da GKS recht umfangreich sein kann, wurden verschiedene Level definiert, die entsprechende Teilimplementationen ermöglichen. Diese wurden unterschieden für Eingabe- und Ausgabeoperationen. Der input level wird durch die Buchstaben a bis c bezeichnet, der output level durch die Zahlen 0 bis 2. Der GKS-Level 2c stellt die vollständige Implementation dar. Durch die Unterscheidung von Eingabe und Ausgabe ist es z.B. möglich, Implementationen für eine reine Plotterausgabe zu erstellen, die keinerlei Eingabemöglichkeiten besitzen.

Für grafische Eingaben wurden umfangreiche Möglichkeiten in Form von logischen Geräten ('logical input devices') vorgesehen: locator, stroke, valuator, choice, string und pick. Der locator dient zur Angabe von Koordinatenwerten (x,y)-Positionen und kann z.B. durch ein Fadenkreuz oder ein grafisches Tablett realisiert sein. Stroke dient der Eingabe einer Folge von (x,y)-Positionen, Valuator erzeugt einen numerischen Wert. Choice erlaubt die Auswahl von Möglichkeiten z.B. aus einem Menü. Es kann durch eine Maus oder einen Lichtgriffel realisiert sein, natürlich auch durch eine Tastatur (bzw. Funktionstasten). String dient der Eingabe von Zeichenketten, sinnvollerweise über die Tastatur. Mit Hilfe des pick-devices kann ein dargestelltes Objekt ausgewählt werden.

Alle grafischen Primitiva und Funktionen haben mehrere Optionen, die das Erscheinungsbild beeinflussen. Diese werden als Attribute bezeichnet. Es gibt zwei Arten von Attributen: bündelbare und nicht-bündelbare.

Bündelbare Attribute lassen sich in Gruppen ('representations') zusammenfassen, wobei eine Kennzahl eine solche Gruppe auswählen kann. Dieses bietet vor allem den Vorteil der Geräteunabhängigkeit, da die Definition der Attribute eines Bündels (die geräteabhängig sein können) nur einmal zu erfolgen hat. Danach kann diese Darstellungsweise über die Kennzahl ausgewählt bzw. angesprochen werden.

Nicht-bündelbare Attribute lassen sich nicht zusammenfassen. Dazu gehören in erster Linie Textattribute und die Kennzahlen der Bündel (die ihrerseits wieder als Attribute betrachtet werden).

Es folgt eine Liste der wichtigsten GKS-Funktionen mit Kurzbeschreibung, gegliedert nach Funktionsbereichen und Level. Die Darstellung erfolgt an dieser Stelle relativ detailiert, da alle GKS Funktionen auch in ein Metafile abgebildet werden können, das dem Datenaustausch (auch über Netz) dienen kann. Es kann damit ein umfangreicher Satz an wichtigen grafik-orientierten Funktionen definiert werden.

3.1.1. Übergeordnete Kontrolfunktionen

Die Kontrollfunktionen dienen der Steuerung der Gesamtsystems, der Initialisierung und der Behandlung von Fehlerzuständen.

3.1.1.1. Level 0a

Open GKS: Initialisierung des Gesamtsystems, der Status wird von system-closed zu system-open geändert.

Close GKS: Schließen des Systems (Leeren evt. Puffer usw.), Status wird system-closed.

Open Workstation: Es wird eine logische Workstation erzeugt und ggf. einem oder mehreren Geräten zugeordnet. Jede logische Workstation erhält eine Beschreibungstabelle ('descriptor table'), die die wesentlichen Eigenschaften enthält.

Close Workstation: Vervollständigt alle Ausgaben zur Workstation und trennt die Workstation vom GKS. Die descriptor table wird entfernt.

Activate Workstation: Die Workstation wird aktiv, d.h. sie ist direkt für Ein- und Ausgabe ansprechbar.

Deactivate Workstation: Die Workstation wird inaktiv. Sie ist weiterhin vorhanden, aber nicht ansprechbar.

Update Workstation: Alle gepufferten Aktionen werden an der Workstation ausgeführt.

Clear Workstation: Außer der Funktion Update wird z.B. bei Grafikbildschirmen die Zeichenfläche gelöscht.

Escape: Mit dieser Funktion lassen sich nicht im GKS definierte besondere Funktionen ausführen, es ist ein Erweiterungsmechanismus, um nicht standardisierte Funktionen standardisiert verwenden zu können.

Error handling: Falls vom GKS ein Fehler gemeldet wird, kann dieser an die Error handling Funktion weitergegeben werden, die entsprechende Aktionen durchführt.

3.1.1.2. Level 1a

Create Segment: Erzeugt ein Segment, das durch eine eindeutige Identifikation ansprechbar ist. Dieses Segment steht an allen aktiven Workstations zur Verfügung (es ist mit diesen Workstations 'assoziiert').

Close Segment: Ein Segment wird abgeschlossen. Es existiert, wird dargestellt, kann jedoch nicht verändert oder gelöscht werden.

Rename Segment: Die Identifikation eines Segments wird geändert.

Delete Segment: Ein Segment wird auf allen Workstations, auf denen es vorhanden ist, gelöscht.

Delete Segment from Workstation: Ein Segment wird auf einer Workstation gelöscht. Das Segment existiert weiterhin und kann verändert werden. Der Zustand nach Ausführen dieser Funktion ist derselbe, als wenn die entsprechende Workstation zum Zeitpunkt der Segment-Generierung nicht aktiv war.

Redraw all Segments on Workstation: Alle mit der spezifizierten Workstation assoziierten Segmente werden dargestellt.

Message: Es Meldung wird in einem reservierten Bereich der Workstation-Zeichenfläche ausgegeben. Dieses ist eine Möglichkeit, um asynchrone Bildschirmausgaben wie Operatormeldungen u.ä. abzufangen und darzustellen, ohne die Grafikausgabe zu zerstören.

3.1.2. Koordinaten- und Transformationsfunktionen

Die grafischen Funktionen dienen der Darstellung eines Bildes auf den verschiedenen grafischen Ausgabegeräten in einer den jeweiligen Bedingungen angepaßten Form.

3.1.2.1. Level 0a

Set Normalization Transformation: Auswahl einer Transformation aus der Transformationstabelle. Die Tabelle wird durch Window- und Viewport-Definitionen erstellt.

Set Window: Angabe der Fenster-Koordinaten in 'world coordinates' für eine Transformation.

Set Viewport: Angabe der Viewport-Koordinaten in NDC für eine Transformation. Window und Viewport-Daten zusammen definieren eine Normalization Transformation.

Set Clipping Indicator: Ein- bzw. Ausschalten des Clipping für die momentan gewählte Transformation.

Set Workstation Window,
Set Workstation Viewport: Festlegung der sogenannten Workstation-Transformation. Diese bildet die NDC-Zeichenfläche auf die Gerätekoordinaten (DC) ab. Dabei wird das Seitenverhältnis des Bildes erhalten, Clipping wird immer durchgeführt.

3.1.3. Ausgabe-Funktionen

Mit den Ausgabe-Funktionen werden die grafischen Grundelemente (Primitiva) dargestellt, aus denen alle Bilder zusammengesetzt werden.

3.1.3.1. Level 0a

Polyline: Darstellung einer Folge von einer oder mehr verbundenen gerade Linien. Der Linienzug beginnt am ersten angegebenen Punkt und endet am letzten.

Polymarker: Darstellung einer Folge von einem oder mehr Punkten, die durch ein zentriertes Symbol (Marker) gekennzeichnet werden.

Text: Ausgabe einer Zeichenkette an einer in Welt-Koordinaten angegebenen Position.

Fill Area: Zeichnen einer durch ein Polygon begrenzten Fläche, die durch drei oder mehr Punkte bestimmt wird. Die Fläche kann dabei mit verschiedenen Arten von Mustern ausgefüllt werden.

Cell Array: Ein rechteckiges Zellengebiet, dessen Zellengröße angegeben wird und bei dem jede Zelle mit einer beliebigen darstellbaren Farbe gefüllt werden kann. Diese Funktion kann vor allem bei Rasterdarstellungen genutzt werden.

Generalized Drawing Primitive (GDP): Erzeugung eines GDP's (z.B. Kreis, Ellipse, Spline) mit Hilfe der gegebenen Punkte und einem GDP-Datensatz. Die GDP's sind implementationsabhängig.

3.1.4. Nicht-bündelbare Attribute

3.1.4.1. Level 0a

Set Character Height,
Set Character Up Vector,
Set Character Path,
Set Text Alignment: Attribute, die die Zeichendarstellung durch GKS beeinflussen (Zeichenhöhe, Zeichenorientierung, Schreibrichtung und Justifikation).

Set Polyline Index,
Set Polymarker Index,
Set Text Index,
Set Fill Area Index: Auswahl der entsprechenden Attributbündel für die einzelnen grafischen Primitiva.

3.1.5. Bündelbare Attribute

3.1.5.1. Level 0a

Set Linetype,
Set Linewidth Scale Factor,
Set Polyline Color Index: Einstellen der Linientype (durchgehend, gestrichelt, punktiert usw.), der Linienbreite und der Farbe für die Polyline-Primitiva.

Set Marker Type,
Set Marker Size Scale Factor,
Set Polymarker Color Index: Einstellen der Symboldarstellung, der Größe und der Farbe für den Polymarker.

Set Text Font and Precision,
Set Character Expansion Factor,
Set Character Spacing,
Set Text Color Index: Auswahl der Schriftart (Font), der Zeichenbreite, des Zeichenabstandes und der Farbe für Texte.

Set Fill Area Interior Style,
Set Fill Area Style Index,
Set Fill Area Color Index: Für das Fill Area Primitivum können die 'Füllmethode', das verwendete Füllmuster und die Farbe gesetzt werden. Die Füllmethode kann vier Werte annehmen: 'hollow' (leer), 'solid' (ausgefüllt), 'patterned' (gemustert) oder 'hatched' (schraffiert). Im hollow-Modus wird nur die Umrandung des Polygons gezeichnet (es entspricht also etwa einem polyline-Aufruf für ein geschlossenes Polygon), im solid-Modus wird eine geschlossene Fläche gezeichnet. In diesen beiden Modi spielt das Füllmuster keine Rolle, braucht also nicht angegeben zu werden. Der patterned-Modus füllt das Polygon mit einem beliebigen regelmäßigen Muster, das aus einer Füllmuster-Tabelle mit Hilfe des Füllmuster-Index ausgewählt wird. Gleiches gilt für den hatched-Modus, jedoch wird ein Schraffurmuster verwendet.

3.1.6. Segment Attribute

Die Segmente besitzen ebenso wie die Primitiva eine Anzahl von Attributen, die die Darstellung beeinflussen. Wichtig sind dabei vor allem die Funktionen zur Positionierung (Transformation) von Segmenten, die wesentlich die Leistungsfähigkeit von Segmenten erhöhen.

3.1.6.1. Level 1a

Set Visibility,
Set Highlighting,
Set Segment Priority: Segmente können sichtbar und unsichtbar gemacht werden, ebenso können sie hervorgehoben werden. Segmente können mit einer Priorität ausgestattet werden, die für verschienene Funktionen eine Bearbeitungsreihenfolge vorschreibt.

Evaluate Transformation Matrix,
Accumulate Transformation Matrix,
Set Segment Transformation: Mit diesen Funktionen kann aus einer Verschiebung (Translation), Drehung (Rotation) und Vergrößerung bzw. Verkleinerung (Skalierung) eine Transformationsmatrix berechnet werden, eventuell mehrere Transformationen nacheinander akkumuliert werden und die resultierende Transformation auf ein Segment (mit allen dazu gehörenden Primitiva) angewendet werden.

3.1.6.2. Level 1b

Set Detectability: Damit kann entschieden werden, ob ein Segment durch ein Pick-Device detektierbar sein soll oder nicht.

3.1.7. Workstation Attribute

Die Workstation-Attribute beeinflussen Umsetzung der GKS-Attribute in die an einer bestimmten Workstation real vorhandenen Attribute zur Darstellung der grafischen Objekte.

3.1.8. Eingabefunktionen

Für alle oben erwähnten Eingabegeräte gibt es Funktionen zum Lesen von Werten bzw. Zeichen, zum Initialisieren der Geräte, zum Auswählen verschiedener Funktionsweisen und entsprechender Parameter und Optionen.

Die Metafile-Funktionen erlauben die Erstellung und Verarbeitung der geräteunabhängigen Grafikdateien (Metafiles).

3.1.9.1. Level 0a

Write Item to GKSM: Schreiben eines Datensatzes in das Metafile, der keine GKS-Funktion spezifiziert.

Get Item Type from GKSM: Anfordern von Informationen über den nächsten im Metafile stehenden Datensatz.

Read Item from GKSM: Lesen eines GKSM-Datensatzes.

Interpret Item: Bearbeiten eines gelesenen GKSM-Datensatzes. Das Resultat ist das gleiche, als wenn die im Datensatz spezifierte Funktion durch Aufruf ausgeführt worden wäre.

Es ist zu beachten, daß das Schreiben von GKS-Informationen in ein Metafile automatisch durchgeführt wird, sobald eine entsprechende Metafile-Workstation eröffnet und aktiviert wurde.

3.1.10. Workstation-unabhängiger Segmentspeicher (WISS)

3.1.10.1. Level 2a

Associate Segment with Workstation: Ein im WISS gespeichertes Segment wird in den Workstation-abhängigen Speicher übertragen.

Copy Segment to Workstation: Ein im WISS gespeichertes Segment wird in eine Workstation übertragen, es wird aber eine Transformation durchgeführt.

Insert Segment: Die Primitiva eines Segmentes, das im WISS abgelegt wurde, werden an eine Workstation übergeben, nachdem eine Transformation durchgeführt wurde. Die Primitiva sind an der Ziel-Workstation nicht mehr als Segment ansprechbar.

3.1.11. Abfragefunktionen

Die momentanen Werte fast aller Parameter und Statuswerte des GKS lassen sich mit den Abfragefunktionen ('Inquiry functions') feststellen. Es sind wiederum die verschiedenen

Level berücksichtigt. Da Abfragefunktionen für den Datentransfer von geringer Bedeutung sind, wird an dieser Stelle nicht weiter darauf eingegangen.

3.2. CORE

Das CORE System ist älter als GKS, allerdings ist die Normung nicht so weit fortgeschritten (es wurde von GKS überholt). Dieses liegt nicht zuletzt daran, daß CORE ein vollständiges 3D-System ist, das zahlreiche Funktionen von Modelliersystemen enthält.

Die Abbildung von drei-dimensionalen Objekten bedingt einen mehrstufigen Abbildungsvorgang. Objekte werden mit ihren lokalen Objektkoordinaten mit Hilfe einer Modellierungstransformation auf die Weltkoordinaten transformiert. In diesen Weltkoordinaten ist ein 3D-Betrachtungsvolumen definiert, an dessen Flächen die Objekte abgeschnitten werden. Die 'geclippten' 3D-Weltkoordinaten werden durch eine Normalisierungstransformation in einen 3D-Viewport in NDCs abgebildet. Der 3D-Viewport wird durch eine Bildtransformation und eine Projektion in 2D-NDCs transformiert /Foley83, Kap. 8/. Der hier beschriebene Weg ist notwendig, falls eine 3D-Bildtransformation durchgeführt werden soll. Ist dieses nicht erforderlich, so ergeben sich eine Anzahl von Alternativen /Foley83, Kap.8; Harr85, Kap.8/.

Projektionen sind notwendig zur Übertragung des drei-dimensionalen Bildraumes auf eine Fläche. Es gibt eine große Anzahl von Transformationskonzepten und Projektionsmethoden. Im CORE-System wird das Objekt als im Weltkoordinatenraum fixiert betrachtet, während die Bildfläche positioniert werden kann. Dazu sind eine Anzahl von Betrachtungsparametern notwendig. Projektionsmethoden können im wesentlichen in Parallelprojektionen und Zentralprojektionen aufgeteilt werden. CORE unterstützt beide Typen und erlaubt damit praktisch alle üblichen Projektionen /Harr85, Kap.8/.

CORE besitzt folgende grafische Primitiva: marker, polymarker, line, polyline, text und polygon sowie eines zur Geometrie-Definition: move. Hier unterscheiden sich GKS und CORE: CORE besitzt einen hypothetischen Zeichenstift ('pen'), dessen momentane Position ('current pen position') stets verfügbar ist. Eine Linie zwischen den Punkten (0.0, 1.0) und (2.0, 3.0) wird unter GKS gezeichnet durch:

```
X (1) = 0.0; X (2) = 2.0;
Y (1) = 1.0; Y (2) = 3.0;
PolyLine (2, X, Y);
```

In CORE ergeben sich zahlreiche Möglichkeiten (X, Y enthalten bereits die Koordinaten, die Prozedurnamen entsprechen der Implementation in /Harr85/):

```
-    Move-Abs-2 (X (1), Y (1));
     Line-Abs-2 (X (2), Y (2));
```

oder

```
-    Move-Abs-2 (X (1), Y (1));
     dX = X (2) - X (1);  dY =  Y (2) - Y (1);
     Line-Rel-2 (dX, dY);
```

oder mit Polyline

```
-    Move-Abs-2 (X (1), Y (1));
     Polyline-Abs-2 (X (2), Y (2), 1);
```

usw.

Move-Abs-2 setzt jeweils die Position, Line-xxx-2 und Polyline-xxx-2 zeichnen die Objekte dann von dieser Position aus. Hinzu kommt die oben angedeutete Möglichkeit der relativen Koordinatenangaben. Insbesondere diese Möglichkeit erfordert das Vorhandensein der Zeichenstiftposition.

Natürlich lassen sich beide Methoden leicht austauschen. Es ist jedoch zu beachten, daß bisher weit verbreitete firmenspezifische Grafiksoftwarepakete wie CalComp oder BENSON zumindest teilweise das Konzept der Zeichenstiftposition beinhalten.

Es sollen hier nicht alle CORE-Funktionen aufgeführt werden, da insbesondere die 2D-Funktionen außer dem oben dargestellten konzeptionellen Unterschied im wesentlichen ähnlich oder vergleichbar sind. Nachfolgend wird auf einige Funktionen besonders eingegangen, die im Zusammenhang mit der 3D-Grafik notwendig oder sinnvoll werden.

Set Display Mode: Es lassen sich vier verschiedene Darstellungsweisen für Polygone definieren:

Fast - Darstellung als Drahtmodell,
Fill - Flächenfüllung der Polygonen, jedoch keine Beachtung der verdeckten Flächen,
Hidden-line - Drahtmodell ohne verdeckte Linien,
Hidden-surface - Volumendarstellung ohne verdeckte Flächen.

Die Verarbeitungsgeschwindigkeit nimmt natürlich zu den letzteren Modes hin ab.

Set Coordinate System Type: CORE erlaubt die Auswahl eines Rechts- oder Links-Hand Koordinatensystems.

Set Window-3,
Set Viewport-3: Die Window- und Viewport-Definitionen werden erweitert für drei-dimensionale Koordinaten. Es wird damit ein Weltkoordinaten-Quader definiert und seine Position in einem (virtuellen) Bildraum angegeben. Der Bildraum, der das Äquivalent zur Bildfläche bei 2D-Darstellungen ist, muß noch durch eine Projektion auf eine 2D-Bildfläche (z.B. den Bildschirm) abgebildet (transformiert) werden. Dazu sind weitere Daten notwendig, die durch die folgenden Funktionen bereitgestellt werden.

Set View Reference Point,
Set View Plane Normal,
Set View Plane Distance,
Set View Depth,
Set View Up: Diese Funktionen geben Daten an, die die Betrachterposition definieren: Position, Blickrichtung, Abstand der Projektionsfläche, Tiefe des betrachteten Bereichs und Aufwärtsrichtung des Betrachter.

Set Projection Type: Auswahl der Projektionsmethode: Parallel- oder Perspektivprojektion. Es wird zusätzlich ein Projektionsmittelpunkt benötigt.

Front/Back Clipping: Der Bildraum kann zusätzlich zum 2D-Bild auch an Vorder- und Rückseite abgeschnitten werden.

Image Transformation Type: Die an einem Segmentbild zugelassenen Transformationen können mit dieser Funktion auf einen der vier möglichen Werte festgelegt werden:

- No image transformation
- 2-dimensional translation
- 2-dimensional scale, rotation and translation
- 3-dimensional scale, rotation and translation

Bildtransformationen entsprechen der Workstation-Transformation des GKS. 3D Transformationen bedingen jedoch, daß bei diesen Transformationen noch Projektionen berechnet werden. Daher ist es sinnvoll, eine dem Anwendungsgebiet entsprechende Bildtransformation auszuwählen.

3.3. VDM und VDI

VDM (Virtual Device Metafile) und VDI (Virtual Device Interface) sind zwei Normentwürfe, die sowohl GKS als auch CORE unterstützen. Beide Normen definieren Schnittstellen, die etwa denen der logischen Workstations im GKS entsprechen. Daher sind nur grafische Informationen mit geringem Abstraktionsgrad (z.B. keine Segmente) vorhanden. Auf eine weitere Untersuchung kann daher hier verzichtet werden.

3.4. Beurteilung der grafischen Standards

GKS und CORE sind selbstverständlich nicht auf den Austausch von grafischen und nicht-grafischen Daten in Form von Dateien ausgerichtet. Die einzige Einrichtung, die in diese Richtung geht, ist das Metafile. Es ist aber wesentlich, alle Funktionen dieser Standards auf ihre Eigenschaften hin zu untersuchen, um festzustellen, wieweit derartige Möglichkeiten in einem Produktbeschreibungsstandard nötig oder nützlich sind. Viele grafische Funktionen sind sehr sorgfältig ausgearbeitet worden, nicht zuletzt aufgrund der umfangreichen Erfahrungen, die über viele Jahre hinweg mit den unterschiedlichen Konzepten gesammelt werden konnten.

Die wichtigsten Punkte seien nachstehend nochmals zusammengefaßt:

Ein vollständiger Satz von grafischen Primitiva ist notwendig, um alle Darstellungen von Objekten auf einfache Weise zu ermöglichen.

Allen Primitiva sind entsprechende Attribute zuzuordnen, die ebenfalls vielfältige Möglichkeiten schaffen sollten. Ob dabei ein Attributkonzept, wie es bei GKS vorgesehen ist, verwendet wird, ist nicht so wesentlich, da die Festlegung von sinnvollen Bündelungen von vielen Faktoren (z.B. auch nationalen Unterschieden) abhängig ist.

Grafische Transformationen und Darstellungsbeschreibungen wie Viewports und Windows sind Grundfunktionen, die jedes System, das grafische Daten behandelt, enthalten sollte. Projektionen sind für drei-dimensionale Objekte ebenso erforderlich, dabei lassen sich alle Arten von Projektionen mit einem wie in CORE gewählten Ansatz darstellen.

Eine Strukturierung der Daten ist notwendig. Es ist zu klären, ob das Segment eine ausreichende Datenstruktur ist. Vermutlich werden zusätzliche Möglichkeiten benötigt.

Die Einrichtung der Eingabe- und Ausgabelevel zeigt viele Vorteile. Es ist damit zu rechnen, daß nicht alle Implementationen einer Norm diese in vollem Umfang unterstützen, insbesondere dann nicht, wenn diese Norm sehr umfangreich ist. Ein Standard zum Austausch von schiffstechnischen Produktdaten wird sicherlich viele Einzelheiten berücksichtigen müssen. Daher ist es sinnvoll, diese schon bei der Festlegung funktionsmäßig in sinnvollen Gruppen zusammenzufassen. Eine Teilimplementation dieser Norm wäre dann in der Lage, bestimmte Funktionen vollständig zu unterstützen.

Erweiterbarkeit, wie sie durch die Escape-Funktionen oder das GDP-Primitivum bereitgestellt wird, ist für eine Norm unerläßlich. Dabei gibt es zwei Kategorien: benutzerdefinierte temporäre Eigenschaften und das generelle Konzept zur Einrichtung neuer Eigenschaften, die zukünftig in die Norm aufgenommen werden.

4. Anwendungsorientierte Normenentwürfe

Die hier diskutierten Normenentwürfe VDA/FS (mit ihrer Erweiterung SFS-Vorschlag der TU Berlin) und IGES sind in ihrem Konzept mehr auf die jeweiligen Anwendungen hin ausgerichtet. Allgemeine grafische Konzepte sind nur teilweise darin zu finden; oft ist dabei die Betrachtungsweise unterschiedlich zu der der oben dargestellten Grafiksysteme.

4.1. VDA/FS und SFS-Vorschlag

Die vom CAD-Arbeitskreis des VDA entwickelte VDA/Formschnittstelle, die auch als DIN-Normvorschlag /DIN/ vorliegt, soll in erster Linie dem Austausch von Formdaten dienen, wie sie z.B. für die geometrische Definition der Karosserieteile im Automobilbau benötigt werden. Die durchaus vergleichbaren Probleme lassen diesen Standard als für die Schiffsformschnittstelle geeignet erscheinen.

Die VDA/FS liegt in der Version 2.0 vor, die einige Erweiterungen gegenüber dem DIN-Vorschlag enthält /VDA86/.

4.1.1. Dateiformat

Die Datei besteht aus Sätzen mit fester Länge von 80 Zeichen, die dem 7-Bit Code der DIN 66003 (deutsches ASCII-Äquivalent) entsprechen müssen. Das Satzformat entspricht dem FORTRAN-Lochkartenformat (72 Zeichen + 8 Zeichen Nummerierung).

Es gibt zwei Arten von Informationen: Geometrische Elemente, nicht-geometrische Daten. Das Satzformat ist für alle Informationen gleich. Jedes geometrische Element und jede nicht-geometrische Information kann mit einem symbolischen Namen benannt werden, nach einem Gleichheitszeichen folgt der Informationstyp, nach einem Schrägstrich alle notwendigen Parameter, die auch über mehrere Fortsetzungssätze ausgedehnt werden können. Kommentarsätze sind ebenfalls an beliebiger Stelle möglich.

Ein symbolischer Name darf bis zu 8 Zeichen lang sein und nur aus Großbuchstaben A-Z oder Ziffern 0-9 bestehen, wobei das erste Zeichen ein Buchstabe sein muß. Die Parameterdaten werden als Liste von Integer- und Realzahlen angegeben, wobei die einzelnen Werte durch Kommata voneinander getrennt werden müssen.

Es darf nur ein drei-dimensionales karthesisches Koordinatensystem verwendet werden, das Weltkoordinaten wiedergibt. Die Koordinatenangaben sind dimensionsgebunden, die Einheit ist Millimeter.

4.1.2. Geometriedaten

Es gibt folgende Geometrie-Elementtypen (die mit einem Stern markierten sind erst in der Version 2.0 enthalten):

POINT: dreidimensionale Koordinaten eines Raumpunktes.

PSET: eine Folge einer beliebigen Anzahl von Punkten, angegeben durch Koordinatentripel.

MDI: eine Punkt-Vektor-Folge. Dieses Element entspricht dem mathematischen Begriff eines Ortsvektors.

CIRCLE: Kreis bzw. Kreisbogen im 3D-Raum. Angegeben werden Mittelpunkt, Radius, Anfangs- und Endwinkel sowie die Kreisebene (durch zwei Vektoren).

CURVE: Eine Kurve, die aus Polynomen beliebigen Grades zusammengesetzt wird. Es werden jeweils stückweise die Polynomkoeffizienten angegeben.

SURF: Dieses Element entspricht dem CURVE-Element für Flächendefinition. Es werden orthogonale Kurven definiert, die ebenso wie für das CURVE-Element aus Polynomstücken beliebigen Grades gebildet werden.

CONS: * Die Abbildung eines CURVE-Elements auf ein SURF-Element. Damit ist es möglich, beliebige Kurven im Raum zu definieren, die auf einer bereits festgelegten Fläche des Typs SURF liegt.

FACE: * Mit diesem Element lassen sich aus SURF und CONS-Elementen beliebig begrenzte Flächen definieren. Grundlage bilden dabei, wie bei den anderen Elementen, stückweise definierte Polynome.

TOP: * Mit dem TOP-Element lassen sich mehrere Flächenstücke, wie sie z.B. durch CONS oder FACE-Elemente definiert wurden zu einer beliebigen Gesamtfläche zusammenfassen.

4.1.3. Nichtgeometrische Daten

Jede Datei wird mit einem HEADER-Satz begonnen. Diesem folgen mindestens 20 Parameter-Datensätze, die Informationen über Absender, Ursprungsrechner und -Software, Empfänger usw. enthalten. Der symbolische Name des Headers ist die Bezeichnung der gesamten Datei. Dementsprechend gibt es einen END-Satz, der das Ende der gesamten Datei anzeigt und der als Namen ebenfalls den im Header verwendeten enthalten muß.

Die geometrischen Datensätze können mit Hilfe der BEGINSET und ENDSET-Sätze strukturiert werden. Es ist nur eine einfache Zusammenfassung von Datensätzen zu einem Set erlaubt, d.h. innerhalb eines Sets darf kein weiterer definiert werden und jeder geometrische Datensatz darf höchstens einem Set zugeordnet werden.

4.1.4. SFS-Vorschlag

Vom Institut für Schiffs- und Meerestechnik der TU Berlin liegt ein Entwurf vor, der die Anforderungen der Schiffbauformschnittstelle auf der Basis einer erweiterten VDA/FS Definition erfüllen soll. Der für diese Untersuchung vorliegende 3. Entwurf vom Juli 1985 nimmt Bezug auf die VDA/FS in der Version 1.0 /SFSB85/. Einige Erweiterungen des Vorschlages sind bereits in der VDA Version 2.0 berücksichtigt (Element CIRCLE mit etwas geänderten Parametern), daher werden hier nur die Unterschiede zur Version 2.0 gezeigt.

Zusätzliche geometrische Elemente (auch als Entities bezeichnet, s.u. unter IGES) sind:

LINE: Koordinaten zweier Raumpunkte, die durch eine gerade Linie verbunden werden sollen.

SLOPE: Angabe eines Tangenteneinheitsvektors an einem oder mehreren Raumpunkten.

CURVAT: Angabe eines Krümmungsvektors an Raumpunkten.

SLOCUR: Verknüpfung der Informationen von SLOPE und CURVAT für einen oder mehrere Raumpunkte.

COONS: Dieses Element definiert Coons'sche Flächenelemente. Es werden die Geometrie-Matrizen für ein oder mehrere Coons-Flächenelemente angegeben.

Als nichtgeometrischer Datensatz wurde der PROJECT-Satz vorgesehen. Dieser Satz wird als Parameterdatensatz des HEADER-Satzes betrachtet. Es folgen eine anzugebende Anzahl

von Parametersätzen, die technische Daten (z.B. Hauptabmessungen) des Projektes enthalten.

4.1.5. Bewertung des SFS-Vorschlages

Der SFS-Vorschlag soll hier nach den Kriterien bewertet werden, die in den Kapiteln 2.1 und 2.2 als Anforderungen an die Schiffsformdaten-Schnittstelle aufgestellt wurden.

4.1.5.1. Allgemeine Anforderungen

Platzbedarf: Es wird Klartext-Codierung und ein festes Satzformat verwendet. Dieses bedingt einen erheblichen Platzbedarf.

Zeichensatz: Der Zeichensatz ist beschränkt auf Ziffern, Großbuchstaben und einige Sonder- bzw. Satzzeichen.

Erweiterungen: Neue Keywords lassen sich jederzeit festlegen, es werden im Laufe weiterer Versionen neue Entities hinzukommen. Problematisch ist dabei, daß kein prinzipielles Konzept definiert wurde, in welcher Form Entities aufgebaut werden. Dieses führt im Laufe der Entwicklung leicht zur Erfindung neuer Problemlösungen, die vom generellen Konzept abweichen (müssen).

Symbole: Entities können mit einem Namen versehen werden, der jedoch nur aus Großbuchstaben bzw. Ziffern (FORTRAN-Konvention) bestehen darf und max. 8 Zeichen lang ist. Diese Einschränkungen erscheinen unnötig. Variablen z.B. zur Kennzeichnung von Konstanten können nicht verwendet werden.

Kompatibilität: Der SFS-Vorschlag wird selbstverständlich aufwärts-kompatibel zur VDA/FS sein. Da fast nur grafische Entities existieren, könnte dieser Standard eine Untermenge von IGES darstellen, gewisse Detailprobleme existieren sicherlich. Ähnliches gilt für die Kompatibilität zu GKS und CORE, da für alle fehlenden (aber notwendigen) grafischen Operationen Standardwerte festgelegt werden müßten.

Editierbarkeit: Aufgrund der Klartext-Codierung und des Satzformates möglich. Jedoch muß beim Editieren das Satzformat wieder hergestellt werden. Bestimmte (in der Version 2.0 hinzugefügte) Entities, die hierarchisch oberhalb der Basis-Entities angeordnet sind und auf solche Bezug nehmen (TOP, FACE, CONS) schränken die Editierbarkeit ein. Weitere derartige zukünftig hinzuzufügende Entites werden zur Einschränkung der Editierbarkeit führen.

Sequentielle Bearbeitung: ist möglich.

4.1.5.2. Spezifische Anforderungen

Die hier untersuchten Anforderungen entsprechen denen aus Abschnitt 2.2.

Koordinatensytem: Es wird ein dreidimensionales rechtwinkliges Koordinatensysten verwendet.

Punkte: POINT-Entity

Linien: LINE-Entity

Kreise: CIRCLE-Entity

Parabel, Hyperbel: Parabel ist in der CURVE-Entity enthalten, Hyperbel ist nicht möglich (in analytischer Form).

interpolierte Kurven: CURVE-Entity

Rotationsflächen: nicht möglich

interpolierte Flächen: SURF-Entity, Erweiterungen bzw. Spezialformen durch CONS, FACE, TOP, COONS.

Tangente, Krümmung: entweder durch SLOPE, CURVAT und SLOCUR-Entity oder durch MDI-Entity realisierbar.

geometrische Transformation: ist nicht möglich, jedoch wird im VDA-Arbeitskreis darüber beraten, eine Entity MATRIX zu schaffen.

Viewport/Windows: nicht möglich

allgemeine Informationen: im HEADER und PROJECT-Satz (ab V2.0 der VDA/FS).

Strukturierung: Es gibt die Entities BEGINSET und ENDSET, die jedoch nur eine Ebene erlauben. Dieses erscheint nicht ausreichend.

4.1.5.3. Allgemeine Beurteilung

Der SFS-Vorschlag ist sehr speziell auf geometrische Daten von Körperoberflächen ausgerichtet. Dadurch ist es möglich, eine einfache Norm anzubieten. Positiv sind die Lesbarkeit und Editierbarkeit der Datei und das übersichtliche Konzept der geometrischen Entities (Ausnahme: die Erweiterungen der VDA/FS V2.0) zu bewerten. Wesentliche Schwachpunkte dieses Vorschlages sind der erhebliche Platzbedarf der Dateien,

sowie die mangelnden Möglichkeiten zur symbolischen Darstellung und zur Strukturierung der Daten. Es fehlen einige nützliche grafische Grundfunktionen.

Da die SFS eine Untermenge des SZF bildet, ist zu überlegen, ob nicht eine gemeinsame Norm entwickelt werden sollte, die die verschiedenen Funktionen durch ein GKS-ähnliches Level-Konzept erfaßt. Die im SFS-Vorschlag enthaltene Normstruktur erlaubt es nicht, das SZF mit einzubeziehen.

4.2. IGES

Die Initial Graphics Exchange Specification wird von einer Interessengruppe beraten, in der alle wesentlichen amerikanischen Unternehmen vertreten sind, die im CAD-Bereich tätig sind oder CAD anwenden, andererseits ist das National Bureau of Standard (NBS) beteiligt. Diese Interessengruppe, die selbst den Namen IGES trägt, ist inzwischen eine ständige Einrichtung, die neben der IGES Spezifikation weitere Normungsprojekte wie PDES betreibt. Die IGES-Spezifikation wird als Vorstufe zu einer allgemeinen Produktbeschreibungsnorm gesehen, an der in erster Linie Erfahrungen in der Softwareerstellung und in der praktischen Anwendung gesammelt werden sollen. Bisher hat das IGES-Kommitee fast nur nationale Beteiligung, allerdings soll die Zusammenarbeit mit Normengremien wie DIN, SEI usw. stark entwickelt werden, zumal die IGES-Spezifikation als Vorschlag für eine ISO-Norm vorliegt.

Die IGES-Spezifikation erscheint z.Z. in der Version 3.0 vom August 1985, die Version 2.0 wurde in /IGES83/ veröffentlicht.

Die Zielsetzung des IGES-Standards ist es, nicht nur grafische Objekte, sondern komplette technische Zeichnungen mit Beschriftungen und technischen Informationen zu definieren. Eine topologische Struktur der Objekte läßt sich nur bis zu einem gewissen Grad festlegen und u.U. mit entsprechendem Software-Aufwand auch 'rückgewinnen'. (Diese Ziele werden jedoch in verstärktem Maße durch die PDES-Spezifikation unterstützt, über die kaum Informationen vorliegen und die sich bisher noch in der Konzeptionsphase befindet.)

IGES weicht daher wesentlich von software-nahen Standards wie GKS oder CORE ab. IGES enthält z.B. über 50 grafische Elemente (hier als geometrische Entities bezeichnet) und 32 Finite Elemente. Zusätzlich gibt es 'Hilfsprimitiva' wie Maßlinien, Bezeichnungen, Pfeile usw. (annotation entities) und Entities zur Strukturierung der Daten (structure entities).

Das IGES-Dateiformat erlaubt zwei Alternativen (in Version 3 vermutlich noch eine weitere), nämlich das ASCII-Format und ein Binär-Format.

Das ASCII-Format ist wie die VDA/FS als 'Lochkartenformat' mit 80 Zeichen langen Sätzen definiert, von denen die letzten 8 Zeichen zur Nummerierung verwendet werden. Die Nummer einer Karte besteht aus einem Buchstaben in Spalte 73, der den Dateiabschnitt angibt und einer vorzeichenlosen Zahl, die in jedem Abschnitt mit 1 beginnt und aufsteigend

gezählt wird. Die Datei wird in fünf Abschnitte unterteilt:

- start section, Kennbuchstabe S
- global section, G
- directory entry section, D
- parameter data section, P
- terminate section, T

Die start section enthält lesbaren Text zur Beschreibung und Identifikation der Datei. Die global section legt einige umdefinierbare Parameter für die Interpretation der Datei fest wie Satz-Ende-Zeichen, Trennzeichen, IGES Version, verwendetes Integer- und Real-Zahlenformat, Modellmaßstab usw.

Die directory entry section ist ein Verzeichnis aller Entities. Die Entities besitzen eine eindeutige Kennziffer. Jede Entity erhält genau einen Eintrag in der directory entry section, die Zeiger (pointer) auf die entsprechenden Sätze in der parameter data section enthält und einige allgemeine Parameter (wie z.B. eine zu verwendende Transformationsmatrix) festlegt. Jede Entity kann auch mit einem 8 Zeichen langen Namen versehen werden.

Die parameter data section enthält die jeweiligen Parameter für alle Entities in sequentieller Folge. Diese Daten sind nur durch den directory Eintrag für die jeweilige Entity erreichbar. Die Daten werden als Liste von Zahlen, durch Kommata getrennt, in die Parameter-Datensätze geschrieben.

In der terminate section gibt es nur einen Datensatz. In diesem werden die Identifikationsnummern des jeweils letzten Satzes eines Abschnitts verzeichnet. Es kann damit die Vollständigkeit der vorher gelesenen Daten überprüft werden.

Das Binär-Format besitzt einen weiteren Abschnitt, der vor den eben genannten anzuordnen ist. In dieser binary information section werden Angaben zum Format der binären Aufzeichung gemacht; der Kennbuchstabe ist B. Das Satzformat ist frei definierbar, die Satznummerierung ist den einzelnen Sätzen jeweils vorangestellt.

4.2.1. Geometrische Entities

Die geometrischen Entities belegen die Kennzahlen im Bereich von 100 bis 199. Bisher sind nur einige dieser Nummern verwendet worden, so daß genug Nummern für zukünftige Erweiterungen zur Verfügung stehen. Die Kennzahlen sind in der folgenden Übersicht in Klammern hinter der Entity-Bezeichnung angegeben.

CIRCULAR ARC (100): Kreis oder Kreisbogen um einen gegebenen Mittelpunkt.

COMPOSITE CURVE (102): Eine Kurve, die aus mehreren Stücken von anderen Entities zusammengesetzt wird: POINT, LINE, CIRCULAR ARC, CONIC ARC oder PARAMETRIC SPLINE.

CONIC ARC (104): Teil eines Kegelschnittes (Ellipse, Parabel oder Hyperbel), definiert durch die Koeffizienten der allgemeinen Kegelschnittgleichung.

COPIOUS DATA (106): Diese Entity faßt die Möglichkeiten von PSET und MDI der VDA-Norm zusammen. Es können Punktemengen durch Koordinatenpaare auf einer Fläche, Raumpunkte durch Koordinatentripel oder Ortsvektoren durch Sextupel angegeben werden.

PLANE (108): Durch die Koeffizienten einer Parametergleichung läßt sich eine begrenzte oder unbegrenzte Ebene definieren. Durch Pointer auf andere Kurven-Entites lassen sich die Ränder der Ebene darstellen.

LINE (110): Gerade Verbindung zweier Raumpunkte, die durch Koordinatentripel bestimmt werden.

PARAMETRIC SPLINE CURVE (112),
PARAMETRIC SPLINE SURFACE (114): IGES erlaubt eine Anzahl verschiedener Spline-Typen, mit denen eine Kurve bzw. eine Fläche (durch ein Gitter von Kurven) durch Interpolation darstellen läßt. Die vorgesehenen Splines sind (ohne Version 3.0):

- linear (Gerade)
- quadratisch (Parabel)
- kubisch
- Wilson-Fowler
- modifizierte Wilson-Fowler
- B-Spline

POINT (116): Ein Raumpunkt, der durch ein Koordinatentripel festgelegt wird. Die Darstellung kann durch ein Markersymbol erfolgen (optional).

RULED SURFACE (118): Durch Definition zweier parametrischer Kurven durch eine der bereits erwähnten Entities wird eine Fläche gebildet, in dem eine gerade Linie jeweils die Punkte gleicher Parameterwerte verbindet und längs der beiden Kurven bewegt wird.

SURFACE OF REVOLUTION (120): Eine erzeugende Kurve (Geneatrix), die durch eine Entity wie CIRCULAR ARC, CONIC ARC, LINE, PARAMETRIC SPLINE CURVE oder COMPOSITE CURVE definiert wurde, wird um eine gerade Achse rotiert. Die Geneatrix und die Achse werden durch Pointer auf andere geeignete Entities bestimmt.

TABULATED CYLINDER (122): Ähnlich der RULED SURFACE wird eine gerade Linie parallel zu sich selbst längs einer als Directrix bezeichneten Kurve bewegt.

TRANSFORMATION MATRIX (124): Eine dreidimensionale Transformationsmatrix wird durch eine 3x3 Rotationsmatrix und einen Translationsvektor aufgestellt. Es gibt mehrere Formen dieser Entity. Die Form 0 transformiert eine Entity, die einen Pointer auf die Transformation besitzt, indem zunächst eine Rotation durchgeführt und dann der Translationsvektor addiert wird. Die Formen 10, 11, 12 führen eine Koordinatensystemtransformation durch, wobei die Form 10 ein lokales kartesisches Koordinatensystem definiert, Form 11 ein zylindrisches und Form 12 ein sphärisches Koordinatensytem festlegen.

LINEAR PATH (106): Diese Entity ist der Polyline des GKS oder CORE vergleichbar. Sie unterscheidet sich von der COPIOUS DATA Entity (auf die durch einen Pointer Bezug genommen wird) dadurch, daß die Punkte als geordnete Menge betrachtet werden, die durch Liniensegmente verbunden werden. Dabei kann der Linienzug auch geschlossen sein, indem die Koordinaten des letzten Punktes gleich denen des ersten gesetzt werden. Die LINEAR PATH Entity kann sowohl 2-dimensionale als auch 3-dimensionale Punktkoordinaten enthalten.

SIMPLE CLOSED AREA (106): Es handelt sich um eine weitere Sonderform der COPIOUS DATA Entity, die ebenfalls eine geordnete Punktemenge beschreibt, die durch Liniensegmente verbunden wird. Es dürfen sich jedoch keine Linien überschneiden, so daß eine einfache Fläche definiert wird.

FLASH (125): Diese Entity definiert einen Punkt im X-Y-Koordinatenbereich, an dem eine zentrierte Kennzeichnungsfläche (Kreis, Rechteck, Ring, Oval oder ein 'SIMPLE CLOSED AREA') angeordnet wird.

RATIONAL B-SPLINE CURVE (126): Mit diesem Objekt lassen sich verschiedene analytische Kurven festlegen: Linien, Kreisbögen, Ellipsenbögen, Hyperbeln oder rationale Polynome und Splines.

RATIONAL B-SPLINE SURFACE (128): Entspricht der RATIONAL B-SPLINE CURVE für Flächen. Es können Ebenen, Kreiszylinder, Kegel, Kugelflächen, Torus, Rotationsflächen, TABULATED CYLINDER, RULED SURFACE sowie durch rationale Polynome und Spline erzeugte Flächen bestimmt werden.

NODE (134): Ein Raumpunkt, der eine Identifikation besitzt, die zur Definition von FINITE ELEMENT Entities verwendet werden kann. Es stehen mehrere lokale Koordinatensysteme zur Verfügung: kartesisches, zylindrisches und Kugel-Koordinatensystem.

FINITE ELEMENT (136): Diese Entity erlaubt die Generierung von FE-Elementen. Es gibt 32 verschiedene 0-, 1-, 2- und 3-dimensionale Typen verschiedener Topologien, Anzahl von Knoten (NODE).

4.2.2. Nicht-geometrische Entities

Die nicht-geometrischen Entities teilen sich auf in Beschriftungselemente (annotation entities) und Strukturierungselemente (structure entities). Für die annotation Entities sind die Kennzahlen im Bereich 200 bis 299 reserviert, einige Entities nehmen Bezug auf die COPIOUS DATA Entity (106). Die structure Entities belegen den Bereich der Kennzahlen 300 bis 499, wobei im Bereich 300 bis 399 die Definitions-Entities angeordnet sind, im Bereich 400-499 dagegen die Instance Entities, d.h. die Anwendungs-Entities der Definitions-Entities und vergleichbarer Entities. Makros haben ebenfalls Definitions-Entities. Die Makro-Aufrufe (macro instance entities) erhalten Kennzahlen im Bereich 600 bis 699.

4.2.2.1. Annotation Entities

Die Beschriftungselemente bestehen in erster Linie aus Bemaßungen und Hilfslinien (witness lines) wie Mittellinien usw. Die Bemaßungs-Entities besitzen Pointer auf einen Text (GENERAL NOTE), bis zu zwei Hilfslinien (WITNESS LINE) und einer beliebigen Anzahl von Pfeilen (LEADER).

ANGULAR DIMENSION (202): Bemaßung eines Winkels. Die Parameter enthalten Daten über Winkelpunkt und die Radien der Pfeilbögen.

CENTERLINE (106): Symmetrie- und Mittellinien festgelegt durch zwei Punkte (Sonderform von COPIOUS DATA).

DIAMETER DIMENSION (206): Durchmesser-Bemaßung. Als Parameter wird der Mittelpunkt des bemaßten Kreises bzw. Kreisbogens benötigt.

FLAG NOTE (208): Diese Entity dient zur Kennzeichnung von hervorzuhebenden Punkten z.B. in technischen Zeichnungen. Ein Text wird dabei von einem Wimpel umrandet. Zusätzlich kann ein Pfeil zu der bezeichneten Position gezeichnet werden. Diese Entity entspricht den amerikanischen Standards in technischen Zeichnungen. Für deutschen bzw. europäischen Gebrauch müßten entsprechende Äquivalente verwendet werden.

GENERAL LABEL (210): Ein allgemeiner Text, der durch einen oder mehrere Pfeile auf die erläuterte(n) Stelle(n) verweist.

GENERAL NOTE (212): Ein beliebiger Text, dessen Fonttyp und Position, Richtung, Größe usw. durch Parameter festgelegt wird.

LEADER (ARROW) (214): Mit dieser Entity wird ein Pfeil gezeichnet, der aus einer Linie oder einem Kreisbogen und einer Pfeilspitze unterschiedlichen Aussehens besteht. Die Parameter geben Position, Richtung, Typ der Pfeilspitze usw. an.

LINEAR DIMENSION (216): Eine Standardbemaßung, bestehend aus Text, zwei Pfeilen und zwei Bemaßungshilfslinien.

ORDINATE DIMENSION (218): Eine Bemaßung, die z.B. verwendet wird, um Abstände von einer gemeinsamen Basislinie zu bezeichnen. Bestandteile sind eine Hilfslinie, evt. ein Pfeil und Text.

POINT DIMENSION (220): Zur Bezeichnung einer Position wird mit dieser Entity ein Text durch eine Hilfslinie mit einem Punkt verbunden. Der Text kann optional von einem Kreis oder Sechseck umrandet werden. Ohne Umrandung wird die Verbindungslinie als Unterstreichung des Textes fortgeführt.

RADIUS DIMENSION (222): Radius- (Kreis- und Kreisbogen-) Bemaßung. Es wird ein Text durch einen Pfeil mit dem bemaßten Radius verbunden. Parameter sind u.a. die Mittelpunktkoordinaten des Kreises.

SECTION (106): Diese Entity (Sonderform der COPIOUS DATA Entity) definiert Schraffurmuster für Schnittflächen in technischen Zeichnungen.

WITNESS LINE (106): Hilfslinien, die aus zwei oder mehr Linienelementen gebildet werden. Es ist eine Sonderform der COPIOUS DATA Entity. Die Hilfslinien werden von allen Bemaßungsentities verwendet.

4.2.2.2. Structure Entities

ASSOCIATIVITY DEFINITION (302): Mit Hilfe dieser Entity lassen sich Strukturen von logischen Verbindungen zwischen Datenklassen herstellen. Die Datenklassen (Classes) bestehen aus Einträgen (in der Art eines directory) mit einem oder mehreren Teilen (Entities oder Datenwerte). Die Parameter geben die Anzahl der definierten Klassen und für jede Klasse die Anzahl von Teilen pro Eintrag an. Die Anzahl der Einträge pro Klasse kann unterschiedlich sein und wird erst in einer ASSOCIATIVITY INSTANCE Entity festgelegt. Alle verknüpften Entities können durch sogenannte back pointer ihrerseits auch auf die Verknüpfung (associativity) verweisen, in der sie enthalten sind. Die Einträge einer Klasse können geordnet sein (die Reihenfolge ist dann signifikant).

ASSOCIATIVITY INSTANCE (402): Hiermit wird eine zuvor definierte Verknüpfungsstruktur (s.o.) eingerichtet. Es gibt einige vordefinierte Verknüpfungen oder aber die mit der ASSOCIATIVITY DEFINITION Entity definierten. Als Parameter werden u.a. die Anzahl der Einträge pro Klasse und entsprechenden Pointer zu Entities bzw. Datenwerte benötigt.

Es gibt folgende vordefinierte Verknüpfungstypen:

- Group: Eine einzelne Klasse, die eine Anzahl von Entities mit back pointern zu einer logischen Entity zusammenfaßt. Diese Verknüpfung ist mit dem bereits dargestellten Segment (z.B. im GKS) vergleichbar.

- Group without back pointers: Wie die Group-Verknüpfung, jedoch enthalten die verknüpften Entities keinen Verweis auf die Gruppe.

- Views Visible: Eine Verknüpfung, die zwei Klassen enthält: Die Ansichten (Views) in denen eine Entity sichtbar ist (falls diese in mehreren Ansichten sichtbar sein soll) und als zweite Klasse die Entities, welche durch diese Verknüpfung verbunden werden.

- Views Visible, Pen Line Weight: Eine Erweiterung der Views Visible Klasse, die zusätzlich erlaubt, unterschiedliche Zeichenstifttypen, Linientypen und -breiten einzustellen.

- Entity Label Display: Eine Ein-Klassen-Verknüpfung, die eine evt. Entity-Beschriftung in unterschiedlichen Ansichten in verschiedenen Positionen und Ausrichtungen darstellt.

- View List: Diese Verknüpfung enthält zwei Klassen: eine Ansicht und eine Liste von Entities, die in dieser Ansicht dargestellt werden sollen.

- Signal String: Eine Verknüpfung mit vier Klassen. Diese Entity ist speziell auf die Bedürfnisse des Entwurfs von Leiterplatten für die Elektronik abgestimmt. Die erste Klasse enthält alle Bezeichnungen, die diese Leitungsführung hat; es ist eine geordnete Klasse. Die zweite Klasse ist eine Liste der Anschlußpunkte. Die dritte Klasse ist eine geordnete Liste zu den zur Darstellung verwendeten geometrischen Entities für eine schematische Darstellung (z.B. Schaltplan), während die vierte Klasse dieselbe Aufgabe zur Darstellung der realen Verhältnisse erfüllt.

- Single Parent: Diese Verknüpfung enthält nur eine Klasse. Diese enthält einen Pointer zu einer Parent-Entity und eine geordnete Liste von Children-Entities. Die Darstellung (display) der Children-Entities richtet sich nach den Darstellungsparametern der Parent-Entity (Transformationsmatrix usw.).

- Text Node: Eine Zwei-Klassen-Verknüpfung, die die Position und die Darstellungsparameter eine GENERAL NOTE Entity zur Mehrfachverwendung in verschiedenen Teilbildern (subfigures (s.u.)) mit verschiedenen Darstellungsparametern erlaubt.

- Connect Node: Diese Verknüpfung ist ebenfalls speziell für Elektotechnik aber auch für Rohrsystementwurf gedacht. Die Klasse 2 der Signal String Verknüpfung nimmt Bezug auf diese Entities. Die erste Klasse ist eine geordnete Liste die auf POINT-Entities verweist, wobei der erste Punkt den Node selbst beschreibt, während die folgenden die

damit verbundenen darstellen. Die zweite Klasse soll die Eigenschaften dieses Verbindungspunktes beschreiben und ist bisher undefiniert.

DRAWING (404): Mit Hilfe dieses Strukturelements können mehrere Ansichten (VIEW (s.u.)) zu einer Zeichnungsdefinition zusammengefaßt werden. Es können zusätzliche Beschriftungselemente, die in keiner Ansicht erfaßt werden (z.B. Schriftfeld) hinzugefügt werden.

LINE FONT DEFINITION (304): Diese Entity erlaubt die Festlegung eines Linienstils. Es können ein Muster (z.B. strichpunktiert), Linienbreite und andere Eigenschaften definiert werden.

MACRO DEFINITION (306): Die Makrodefinition erlaubt es, neue Entities selbst zu definieren. Benutzt werden diese Entities durch eine MACRO INSTANCE Entity. Jede neue Entity erhält eine eigene Kennzahl im Bereich 600 bis 699. Ein Makro wird durch ein Programm in einer Makrosprache definiert. Diese Sprache enthält folgende Befehle:

MACRO: zeigt den Beginn einer Makro-Definion an. Als Parameter folgen die Kennzahl und die Aufruf-Parameter.

LET: Dieser Befehl entspricht dem BASIC-Kommando LET (Zuweisung).

SET: Dieser Befehl bewirkt die Directory- und Parameter-Einträge beim Aufruf eines Makros.

REPEAT: Eine Schleife wird mit diesem Befehl begonnen. Das Schleifenende wird durch den Befehl CONTINUE angezeigt. Die Anzahl der Wiederholungen wird als Parameter im REPEAT-Befehl eingetragen.

CONTINUE: Schleifenende.

MREF: Erlaubt das geschachtelte Aufrufen eines Makros aus einem Makro. Der Aufruf darf nicht rekursiv sein.

ENDM: Ende einer Makro-Definition.

In der Version 3.0 der IGES sind insbesondere die Makro-Sprachelemente erheblich erweitert worden. Es sind Sprünge, Label, Verzweigungen usw. möglich. Zusätzlich können andere IGES-Dateien angesprochen werden ('EXTERNAL FILE REFERENCE'),

sodaß Makro-Bibliotheken u.ä. möglich sind. Über die Erweiterungen lagen jedoch keine näheren Informationen vor.

In den Makros können Variablen verwendet werden. Der Typ der Variablen wird (ähnlich wie in BASIC und FORTRAN) durch besondere Buchstaben oder Sonderzeichen als erstem Zeichen des Variablennamens bestimmt. Integer beginnen mit den Buchstaben I..N oder i..n, Real mit A..H, O..Z oder a..h, o..z, Double precision mit einem '!', String-Variablen mit einem '$' und Pointer mit einem '#'.

Es stehen eine ganze Anzahl den FORTRAN Bibliotheksfunktionen ähnliche Funktionen zur Verfügung. Vom Benutzer können keine Funktionen definiert werden.

MACRO INSTANCE (600 bis 699): Aufruf eines selbstdefinierten Makros. Die Parameter werden als Aufrufparameter weitergegeben.

PROPERTY (406): Die Property-Entity dient zur Zuordnung von Eigenschaften zu anderen Entities. Diese Eigenschaften können durch Text oder numerische Daten beschrieben werden. Es gibt (wie bei den associativity entities) vordefinierte und selbst zu definierende Formen. Z.Z. gibt es folgende vordefinierten Formen:

Definition Levels: Es wird ein System von logischen Funktionsebenen eingerichtet, in dem Entities mehreren Ebenen zugeordnet sind.

Region Restriction: Mit dieser Eigenschaft lassen sich Entities definieren, die Beschränkungen für bestimmte geometrische Bereiche festlegen. Z.B. kann so bestimmt werden, daß ein Teil außerhalb eines Bereiches angeordnet werden muß. Diese Eigenschaft ist im Hinblick auf Anwendungen der Elektrotechnik konzipiert.

Level Function: Eine Funktionsebenenhierarchie kann mit dieser Entity festgelegt werden, so daß funktional zusammengehörige Teile identifiziert werden können.

Region Fill Property: Vergleichbar dem GKS-Fillstyle-Attribut kann hiermit die Methode und gegebenenfalls das Muster zum Ausfüllen einer Fläche definiert werden.

Line Widening: Diese Property Entity dient zur Einstellung der Linienbreite und der dargestellten Linienenden (abgerundet oder rechteckig). Ein weiteres Attribut ist die Ausrichtung der gezeichneten Linie zur geometrisch definierten Linie (zentriert, rechts- oder linksbündig).

Drilled Hole: Die Zuordnung dieser Eigenschaft z.B. zu einer POINT-Entity kennzeichnet diese als Bohrloch (mit Angabe des Durchmessers, Oberflächenbearbeitung und evt. Plattierung). Auch diese Form wurde in erster Linie für den Leiterplattenentwurf geschaffen.

Reference Designator: Verknüpfung eines Bezeichnungstextes mit einer Entity, die ein elektrisches Bauteil darstellt.

Pin Number,
Part Number: Benennungen von elektrischen Anschlüssen und Teilen.

Hierarchy: Diese Form erlaubt die Einrichtung einer Hierarchie von Directory-Einträgen in Bezug auf die bei der Darstellung von Entities zu verwendenden Darstellungsparameter und Properties.

SUBFIGURE DEFINITION (308): Eine Anzahl von Entities und evt. weiteren Subfigures wird zu einem Teilbild zusammengefaßt, daß innerhalb der Gesamtzeichnung mehrfach dargestellt werden kann. Die Subfigure kann mit einem Namen versehen werden.

SINGULAR SUBFIGURE INSTANCE (408): Diese Entity bewirkt die Darstellung einer Subfigure Entity, wie sie durch die SUBFIGURE DEFINITION festgelegt wurde. Als Parameter werden ein Skalierungsfaktor, und die Position im Modellraum angegeben.

RECTANGULAR ARRAY SUBFIGURE INSTANCE (412): Eine Basis-Entity wird in regelmäßiger Anordnung in Spalten und Zeilen angeordnet. Die Basis-Entity kann eine Gruppe, eine Subfigure-Anwendung, ein Punkt, Linie, Kreisbogen, Kegelschnitt oder ein Rechteck- oder Rotationsfeld sein. Einzelne Anordnungen der Basis-Entity können nach einem Muster unsichtbar sein.

CIRCULAR ARRAY SUBFIGURE INSTANCE (414): Vergleichbar mit dem Rechteckfeld werden Basis-Entities längs des Radius' eines unsichtbaren Kreises angeordnet.

TEXT FONT DEFINITION (310): Diese Entity definiert das Erscheinungsbild eines neuen Zeichensatzes. Die Zeichen werden nach einem Kodierschema aus einzelnen Linien(-zügen) gebildet.

VIEW (410): Die Projektion eines 3D-Modells auf die X-Y-Ebene wird mit dieser Entity bestimmt. Als Parameter werden eine Ansichtskennzahl, ein Skalierungsfaktor und optional die Begrenzungsflächen eines Clipping-Raumes angegeben.

4.2.3. Bewertung

Zu den schon bei der Beurteilung der VDA/FS und ihrer schiffstechnischen Erweiterung angewendeten Kriterien, die in den Abschnitten 2.1 und 2.2 aufgeführt wurden, müssen bei der IGES-Beurteilung auch die Punkte des Abschnittes 2.3 für das Schiffbauzeichnungsformat berücksichtigt werden.

4.2.3.1. Allgemeine Anforderungen

Platzbedarf: Es stehen mehrere Formate zur Verfügung, unter anderem ein komprimiertes ASCII-Format (ab V3.0), das wohl einen guten Kompromiß bezüglich der sich widersprechenden Forderungen geringer Platzbedarf und Editierbarkeit darstellen dürfte. Das Binär-Format ist sehr platzsparend. Die Möglichkeit der Auswahl eines Formates ist sicherlich positiv zu bewerten, da jeweils anwendungsbezogen entschieden werden kann, welches Format verwendet werden sollte.

Zeichensatz: Es steht der gesamte ASCII-Zeichensatz zur Verfügung. Kritische Sonderzeichen, die z.B. national unterschiedliche Bedeutung haben können, finden keine Verwendung.

Erweiterungen: Das gesamte IGES-Konzept ist auf Erweiterungen ausgelegt. Für alle Arten von Entities sind in den jeweiligen Kennzahlenbereichen frei Nummern reserviert. Ebenso gibt es reservierte Form-Kennzahlen. In den Directory-Einträgen sind freie Plätze für Pointer und Parameter vorhanden. Durch das Pointer-Konzept ist eine klare Struktur für Erweiterungen vorgegeben. Inwieweit diese Struktur in Zukunft ausreichen wird, ist schwer zu beurteilen.

Symbole: Alle Entities können mit Namen versehen werden. Allerdings können diese Namen nicht zur Referenz von Entities innerhalb der Datei benutzt werden, da dieses ausschließlich über Pointer geschieht. Variablen und damit eine Parametrisierung von Entities sind innerhalb von Makros möglich, sonst jedoch nicht.

Kompatibilität: Da IGES die Basis zukünftiger Entwicklungen bildet, wird sicherlich eine mehr oder weniger begrenzte Kompatibilität z.B. zu PDES vorhanden sein. STEP scheint aufgrund seiner Konzeption nicht kompatibel zu werden. Zu den Grafik-Standards GKS und CORE bestehen z.T. erhebliche Unterschiede.

Editierbarkeit: Eine IGES-Datei ist ohne sehr komplexe Hilfsmittel nicht editierbar, da z.B. das Löschen eines Parameterdatensatzes eine Veränderung vieler Pointer zur Folge hat. Die Bereitstellung spezieller Editoren ist sehr aufwendig und daher unwahrscheinlich. Binäre Dateien lassen sich nur nach einer Umsetzung lesen und bearbeiten, auch dieses ist sehr aufwendig.

Sequentielle Bearbeitung: Es ist nur bedingt möglich, eine Datei sequentiell zu lesen. Auf jeden Fall muß immer das gesamte Directory gelesen werden. Intensive Verwendung von back pointern macht das sequentielle Bearbeiten unmöglich.

4.2.3.2. Spezifische Anforderungen

Koordinatensystem: Es steht allgemein nur ein rechtwinkliges Koordinatensytem zur Verfügung, mit Ausnahme der FE-Elemente, für die zusätzlich Zylinder - und Kugelkoordinaten verwendet werden können.

Punkte: POINT-Entity, NODE-Entity

Linien: LINE-Entity

Kreise: CIRCULAR ARC-Entity

Parabel, Hyperbel: CONIC ARC-Entity

interpolierte Kurven: PARAMETRIC SPLINE CURVE-Entity, RATIONAL B-SPLINE CURVE-Entity, COMPOSITE CURVE-Entity

Rotationsflächen: SURFACE OF REVOLUTION-Entity

interpolierte Flächen: PARAMETRIC SPLINE SURFACE-Entity, RATIONAL B-SPLINE SURFACE-Entity, RULED SURFACE-Entity, TABULATED CYLINDER-Entity

Tangente, Krümmung: COPIOUS DATA-Entity, PLANE-Entity

geometrische Transformation: TRANSFORMATION MATRIX-Entity

Viewport/Windows: Die Verwendung der geometrischen Operationen dieser Entities ist in IGES abweichend von GKS oder CORE realisiert. Die VIEW- und DRAWING-Entities beschreiben die Darstellung des Objekts in einer Zeichung. Die Sichtbarkeit einer Entity in einer Ansicht wird durch ihre Assoziierung mit der entsprechenden VIEW-Entity festgelegt. D.h., daß die Datei selbst bereits diese Sichtbarkeitsinformationen enthält. Bei GKS und CORE hingegen wird dieses Kriterium erst bei der Darstellung untersucht.

allgemeine Informationen: für diese Daten gibt es eine Vielzahl von Beschriftungs-Entities, die oft jedoch sehr speziell auf bestimmte Anwendungen ausgerichtet sind.

Strukturierung: es sind komplexe Verknüpfungen aller Art mit Hilfe der ASSOCIATIVITY-Entities möglich. Segmente (im Sinne von GKS oder CORE) lassen sich mit den Typen 'group' oder 'single parent' definieren. Es gibt Makros (ab V3.0 auch Makro-Bibliotheken), allerdings ist nur eine beschränkte Anzahl von Makros definierbar. Makros können keine symbolischen Bezeichnungen erhalten.

FE-Elemente: NODE-Entity, FINITE ELEMENT-Entity

Achsendarstellung: nicht vorhanden, aber als Makro definierbar.

Detailinformationen: außer speziellen, auf die Bedürfnisse der Elektronik abgestellten Entities existieren nur wenige Möglichkeiten. Entsprechende Anforderungen müßten ebenfalls durch Makros (z.B. für Schweißnahtsymbole) erfüllt werden. Auch hier dürfte die begrenzte Zahl von Makros hinderlich sein.

Datenbankinformationen: nicht-grafische Informationen können nur durch Beschriftungs-Entities (GENERAL LABEL) übermittelt werden. Dazu wäre jedoch eine Erweiterung des Standards zur Festlegung der Interpretation dieser Daten notwendig.

4.2.3.3. Allgemeine Beurteilung

Der IGES-Normvorschlag ist sehr umfangreich und komplex aufgebaut. Die Möglichkeiten der grafischen Entities sind ausreichend. Die Assiociativity Entities sind sehr leistungsfähig, aber kompliziert zu verarbeiten und daher aufwendig in ihrer Implementierung. Nicht-grafische Informationen lassen sich, soweit es um die reine Zeichnungs-

beschriftung geht, gut darstellen. Nicht so günstig ist die Situation bei noch abstrakteren Informationen. Hier sind zusätzliche Festlegungen notwendig.

Das Pointer-Konzept ist zusammen mit der Gliederung der Datei in mehrere Abschnitte die zentrale Struktur der IGES-Daten. Es besitzt viele Vorteile im Hinblick auf die Erweiterbarkeit und die Flexibilität in der Darstellung der Daten, ist jedoch nur durch Programme interpretierbar, die entsprechend aufwendig sein müssen. (Entsprechend lange hat die Verwirklichung erster IGES-Prozessoren gedauert.)

Es gibt deutliche Differenzen in der Betrachtungsweise der grafischen Entities und Funktionen im Vergleich zu den grafischen Standards GKS und CORE. Dieses wird besonders in Zukunft problematisch sein, da mit Modelliersystemen zu rechnen ist, die diese Standards zur Darstellungsgenerierung verwenden werden.

An mehreren Entities wird deutlich, daß einige Industriezweige besonderen Einfluß auf die Definition hatten. Dieses ist nicht negativ zu bewerten; es wird aber erforderlich sein, entsprechende detailorientierte Entities zu definieren. IGES ist daher nicht ohne Überarbeitung und Anpassung auf die Bedürfnisse des Schiffbaus anwendbar.

5. Ansätze und Schlußfolgerungen für den Entwurf einer Norm

Im folgenden sollen die Ergebnisse der bisher erfolgten Untersuchungen dazu verwendet werden, um Möglichkeiten für die Erstellung einer Norm für den grafischen Datenaustausch (abgekürzt GDA) aufzuzeigen. Schwerpunkte sollen dabei die Berücksichtigung möglichst vieler vorher angesprochener Punkte sein.

Ein anderer wesentlicher Aspekt ist die Umsetzung einer solchen Norm in die Realität (d.h. die Implementation auf verschiedenen Rechnern). Normentwürfe wie IGES sind sehr aufwendig in der Implementation und vor allem in der Validierung (d.h. Tests zur Überprüfung, ob alle Anforderungen der Norm erfüllt werden). Für Projekte wie IGES sind sehr große Testbibliotheken erforderlich, die über Jahre hinweg aufgebaut werden. Es ist daher sinnvoll, insbesondere über Möglichkeiten des Rechnereinsatzes bei der Implementation selbst nachzudenken.

5.1. Methoden zur Generierung von Parsern

Wesentlicher Bestandteil eines GDA-Datei Preprozessors ist ein sogenannter Parser, der die symbolischen Informationen (auch Zahlen sind hier als Symbole anzusehen) in eine maschinen-interpretierbare Form umsetzt. Der gesamte Vorgang der Dateiverarbeitung setzt sich aus den drei Schritten lexikalische, syntaktische und semantische Analyse zusammen, die je nach verwendeter Methode parallel oder nacheinander ablaufen /Zima82/.

Die **lexikalische Analyse** liest die Datei zeichenweise, eliminiert z.B. Kommentare und faßt bestimmte Zeichengruppen wie Bezeichner (identifier), Zahlen, Zeichenketten zu einer Einheit zusammen.

Die **syntaktische Analyse** prüft die richtige Abfolge von lexikalischen Einheiten. Diese Abfolge ist durch die Syntax der "Dateisprache" festgelegt, die Gesamtheit der Syntax-Vorschriften wird als **Grammatik** bezeichnet.

Die **semantische Analyse** prüft die inhaltliche Konsistenz der verarbeiteten Informationen (z.B. die Verwendung unzulässiger Werte, undefinierter Bezeichner usw.).

IGES definiert eine Sprache, die nur geringen syntaktischen Umfang hat, da nur lexikalische Einheiten wie Zahlen, Kommata und ein starres Satzformat verwendet werden. Dafür ist der semantische Umfang gewaltig. Ein IGES-Parser wird

daher eine große Anzahl von Konsistenzprüfungen durchführen und die Verflechtungen durch die Pointer analysieren müssen.

VDA/FS hingegen definiert eine syntaktisch umfangreichere Sprache, da eine Anzahl von Kennworten (keywords) existieren und diesen bestimmte, jeweils in Anzahl und Anordnung unterschiedliche Parameter zugeordnet sind. Kennworte sind die Grundbestandteile einer Sprache, die bei der Grammatik-Definition auch als **terminale Symbole** bezeichnet werden.

Es ist festzustellen, daß eine syntaktisch aufwendigere Sprache gleichzeitig den Vorteil besserer Lesbarkeit bietet. Andererseits erscheint eine solche Sprache ebenfalls aufwendig in der Implementation.

Es gibt jedoch die Möglichkeit, eine syntaktisch umfangreiche Grammatik zu definieren und diese sogar automatisch (d.h. durch ein Programm) in einen Parser umsetzen zu lassen, falls man einige kleinere Einschränkungen bei der Definition der Grammatik berücksichtigt /Wirth84, S.32/. Diese Möglichkeit wird durch die Klasse der 'recursive descent' Parser (bzw. Compiler) geboten, die z.B. für Compiler verschiedener Hochsprachen verwendet werden. Die Definition der Grammatik kann dann ebenfalls in einer gebräuchlichen Notationsweise erfolgen (z.B. der im folgenden verwendeten erweiterten Backus-Naur-Form, abgekürzt BNF), die ein Programm (ebenfalls ein Parser) wie eine Programmiersprache sehr einfacher Art lesen kann, um daraus Datenstrukturen zu bilden, die einem bereits bestehenden Parsergerüst als Informationen dafür dienen, wie im konkreten Fall eine Analyse durchgeführt werden soll. Dieses Konzept löst in erster Linie die Frage der Programmierung des Syntaxanalysators und erlaubt daher die Definition verständlicherer Sprachen auch für Anwendungen wie den GDA. Ein wesentlicher Punkt ist allerdings, daß solche Parser nur in einer Sprache geschrieben werden können, die Rekursion zuläßt.

Die semantische Analyse ist mit den Ergebnissen der syntaktischen Analyse dann relativ leicht durchführbar. Sie ist jedoch im Falle des GDA z.B. abhängig von den Anwendungsprogrammen, die die Informationen weiterverarbeiten sollen (z.B. CAD-Systeme u.ä.).

Die lexikalische Analyse ist für viele Anwendungen recht einfach, da nur Einheiten wie Zahlen (Real u. Integer), Keywords und Bezeichner vorkommen. Sie ist daher bereits im generierten Parser integriert.

In /Wirth84/ wird ein sehr einfacher BNF-Parser-Generator beschrieben, der für die nachfolgenden Versuche als Ausgangspunkt verwendet wurde. Dieser Generator verarbeitet eine erweiterte BNF, die folgende Elemente enthält:

1) Beliebige Zeichenketten, die auch das Unterstreichungszeichen enthalten dürfen, werden als nicht-terminale Symbole, d.h als syntaktische Einheiten angesehen. Die Bedeutung der Zeichenkette kann z.B. der semantischen Funktion entsprechend gewählt werden.

2) Zeichenketten, die in doppelten Anführungszeichen eingeschlossen sind, werden als terminale Symbole, d.h. die Kennworte der Sprache verarbeitet.

3) Runde Klammern haben gruppierende Funktion, ähnlich ihrer Funktion in mathematischen Formeln.

4) Eckige Klammern ('[' und ']') zeigen, daß die davon umschlossenen Einheiten optional sind, d.h. nicht vorhanden sein müssen.

5) Geschweifte Klammern werden um Einheiten angeordnet, die sich wiederholen können, und zwar null-, ein- oder mehrmals. Das bedeutet, daß diese auch entfallen können.

6) Ein senkrechter Strich trennt zwei Alternativen, er hat die logische Bedeutung des Wortes 'oder'.

Die Sprachdefinition erfolgt in Form von sogenannten Produktionen, die anzeigen, wie ein nicht-terminales syntaktisches Symbol in eine Folge von hierarchisch niedriger anzuordnenden Symbolen (terminal und nicht-terminal) zerlegt werden kann. Eine Produktion besteht aus einem nicht-terminalen Symbol, einem Gleichheitszeichen und einer Folge von syntaktischen Einheiten, abgeschlossen wird sie durch einen Punkt. Der generierte Parser arbeitet die Produktionen von einem Startsymbol aus beginnend so lange ab, bis alle im untersuchten Text vorhandenen terminalen Symbole abgearbeitet sind.

Die nicht-terminalen Symbole für Bezeichner und Zahlen (identifier und number), die bereits in der lexikalischen Analyse erkannt werden, werden den intern vereinbarten terminalen Symbolen "$ID" bzw. "$NUM" zugeordnet.

Der BNF-Parser-Generator wurde in der Sprache 'C' geschrieben (dieses geschah in erster Linie wegen der erforderlichen umfangreichen Rekursion). Ein Listing befindet sich im Anhang. Das Programm liest eine in BNF notierte Sprachdefinition, überprüft diese (mit gewissen Einschränkungen) und baut die Datenstrukturen für das ebenfalls bereits enthaltene Parser-Gerüst auf. Anschließend liest dieser neu entstandene Parser eine Beispieldatei in der gerade definierten Sprache und analysiert diese nach den gegebenen Regeln. Es besteht also die Möglichkeit, sehr schnell eine

konfliktfreie Grammatik zu entwickeln und auszubauen, zusätzlich wird immer der jeweils arbeitsfähige Parser erzeugt.

5.2. Entwurf einer einfachen GDA Sprache

Die nachfolgend beschriebene Sprachdefinition für Produktdaten-Dateien erhebt selbstverständlich keinen Anspruch auf Vollständigkeit. Es soll nur das Prinzip gezeigt werden, wie sich mit den oben beschriebenen Hilfsmitteln eine lesbare, editierbare Sprache schaffen läßt, die alle wesentlichen Strukturen enthält.

Eine Produktbeschreibungsdatei besteht aus einer beliebigen Folge von Zuweisungen an Variablen, Definitionen von Strukturen wie Segmenten und Makros oder Entities. Abgeschlossen wird eine Datei durch einen Prüfsatz, der die Anzahl der vorher verarbeiteten Einheiten der drei Gruppen enthält, optional können noch CRC-Werte angegeben werden.

```
prod_desc_file = { (symbol_assign | definition | entity) ";"}
                   check_record .
```

Die Definitionen dieser nicht-terminalen Symbole folgen in den nächsten Produktionen. Eine Zuweisung an eine Variable ist wie folgt aufgebaut:

```
symbol_assign  = identifier "=" expression .
```

Die Strukturdefinition kann alternativ aus vier verschiedenen Typen gebildet werden: der Segment-Definition, der Makro-Definition, der externen Segmentreferenz und der externen Makroreferenz. Externe Referenzen nehmen Bezug auf andere Dateien, die z.B. Segment- oder Makrobibliotheken enthalten. Ein Segment erhält einen Namen, in geschweiften Klammern folgt dann der Segmentkörper, der aus einer Folge von Anweisungen besteht. Gleiches gilt für die Makrodefinition, jedoch folgt auf den Makronamen eine Parameterliste. Die externen Referenzen geben jeweils den Namen des Objekts an, bei Makros folgt in Klammern die Zahl der benötigten Parameter. Ein Typprüfung für Makroparameter ist hier nicht vorgesehen, ebenso wie die Definition von Makros an dieser Stelle ebenfalls verkürzt behandelt wird.

```
definition      = "segment" identifier "{" { statement } "}" |
                  "macro" identifier
                        "(" [identifier {"," identifier} ] ")"
                        "{" { statement } "}" |
                  "extern"
                     ("segment" identifier |
                      "macro" identifier "(" number ")" ).
```

Eine Anweisung ist jeweils entweder eine Variablenzuweisung oder eine Entity, abgeschlossen durch ein Semicolon:

```
statement       = (symbol_assign | entity) ";".
```

Die Entity ist die Grundeinheit zur Erzeugung von Objekten. Diese Objekte können ein grafisches oder ein nicht-grafisches Primitivum sein, eine Segment- oder Makro-Anwendung oder aber ein GKS-Metafile Datensatz:

```
entity          = graphic_prim | nongraph_prim |
                     segment_instance | macro_instance |
                     gks_metafile_rec .
```

Der Prüfsatz, der die Datei abschließt, beginnt mit einem Kennwort, danach folgen die Zahlenwerte und eventuell die CRC-Werte:

```
check_record    = "$CHECK" number "," number "," number
                     ["," crc "," crc "," crc ] "end".
```

Die grafischen Primitiva werden durch ein vorangestelltes Kennwort bezeichnet, es folgt ein optionaler Bezeichner, unter dem diese Entity in der nachfolgenden Datei angesprochen werden kann. Fehlt dieser Bezeichner, besteht diese Möglichkeit nicht. Es folgt dann eine Parameterliste (ebenfalls optional), deren Bedeutung von der jeweiligen Entity abhängt. Ein spezielles grafisches Primitivum ist die Transformation, die durch ein vorangestelltes "transfm" gekennzeichnet wird. Auch hier kann ein Bezeichner angegeben werden, was in fast allen Fällen notwendig ist. Es folgt eine spezielle Liste von Ausdrücken.

```
graphic_prim    = (gp_keyword [identifier] [param_list]) |
                     ("transfm" [identifier] transform_expr).
```

Die nicht-grafischen Primitiva sind im Aufbau den grafischen Primitiva vergleichbar (mit einigen Einschränkungen), sie besitzen eigene Kennworte.

```
nongraph_prim  = ngp_keyword [identifier] [param_list].
```

Eine Segment-Anwendung beginnt mit dem Kennwort "draw", gefolgt von dem Bezeichner des Segments. Optional folgt dann ein Transformationsausdruck.

```
segment_instance = "draw" identifier [ ":" transform_expr ].
```

Eine Makro-Anwendung ist wieder der Segment-Anwendung vergleichbar, das Kennwort ist jedoch "call" (wie ein Unterprogrammaufruf); es muß eine entsprechende Anzahl von Parametern angegeben werden (die Überprüfung ist jedoch, genauso wie eine evt. Typprüfung, ein semantisches Problem) und es kann eine Transformation angewendet werden.

```
macro_instance = "call" identifier "(" [param_list] ")"
                        [":" transform_expr].
```

Die GKS-Metafile Sätze sollen hier nur stellvertretend dargestellt werden, sie sind daher nur in ihrem Grundschema, wie es in /GKS82/ definiert wurde, dargestellt. Wie GKSM-Sätze zu behandeln sind, kann hier nicht festgelegt werden.

```
gks_metafile_rec = "GKSM" number {"," item }.
```

Die Liste der grafischen Kennworte, die gleichzeitig die Liste der verfügbaren grafischen Grundbausteine wiedergibt, ist ebenfalls nur stellvertretend zu sehen, gleiches gilt für die nicht-grafischen Elemente. Diese Listen sind leicht erweiterbar.

```
gp_keyword      = "point" | "line" | "vector" | "curve" |
                  "surface" | "rotsurf" | "ratcurv" |
                  "ratsurf" | "points" | "circle" .

ngp_keyword     = "label" | "attrib" | "text" | "dimen" |
                  "link" | "value".
```

Die Parameterliste besteht aus einem oder mehreren Parametern, die verschiedene Erscheinungsformen haben können. Ein vorangestellter Doppelpunkt zeigt einen Transformationsausdruck an. Ein vorangestelltes 'commercial-at' ('@') führt eine in Klammern angeordnete Liste von Koordinaten an. Ein '&' zeigt eine nachfolgende Entity an. Damit können z.B. nicht-grafische Entities grafischen als Attribute zugeordnet werden. Ein Stern bewirkt die Referenz einer anderen Entity. Schließlich kann ein Parameter selbstverständlich eine Zahl oder ein Bezeichner sein, oder aber eine in Klammern angeordnete Liste solcher Größen. Allen Parameterwerten kann ein Deklarator, gefolgt von einem Gleichheitszeichen, vorangestellt sein. Der Deklarator kann den Verwendungszweck des Parameters genauer bezeichnen (s.u.).

```
param_list      = parameter {"," parameter } .


parameter       = (":" transform_expr) |
                    ("@" "(" coor_list ")" ) |
                    ("&" [decl "="] entity) |
                    ("*" [decl "="] identifier) |
                    ([decl "="]
                        (item | "(" item {"," item} ")")).
```

Ein Transformationsausdruck erlaubt die Anwendung einer aus Skalierung, Rotation und Translation zusammengesetzten Transformation eines entsprechenden Objekts. Entweder werden diese Daten direkt angegeben (die einzelnen Teile werden dabei durch die Kennworte "scale", "rot" und "trans" gekennzeichnet), oder es wird eine Referenz auf eine Transformations-Entity angegeben.

```
transform_expr = ("(" ["scale" coor_list] ["rot" coor_list]
                    ["trans" coor_list] ")" )
                  | ("*" identifier).
```

Ein Deklarator kann bestimmte Parameter kennzeichnen, die Eigenschaften einzelner Entity-Typen beschreiben. Welche Deklaratoren bei welchen Entities verwendet werden dürfen, ist wiederum ein semantisches Problem. Es kann theoretisch auch syntaktisch festgelegt werden, allerdings würde dadurch die Sprachdefinition sehr aufgebläht werden. Die Liste der hier verwendeten Deklaratoren ist ebenfalls nur als Beispiel anzusehen.

```
decl            = "pos" | "mag" | "angle" | "rad" | "font"
                    | "color" | "style".
```

Die restlichen Produktionen definieren einfache Datenelemente wie Koordinatenwerte und Datenwerte. Die Ausdrücke erlauben die Darstellung einfacher Formeln, die Operatoren wie "+", "-", "*" und "/" sowie Klammerung zulassen und die Operator-Folge im mathematischen Sinne berücksichtigen.

```
coor_list       = item "," item ["," item].

expression      = ["+" | "-"] term { ("+" | "-") term }.
term            = factor { ("*" | "/") factor }.
factor          = item | "(" expression ")".

item            = identifier | number.
```

Die CRC-Werte werden an dieser Stelle lediglich als Zahlenwerte definiert, die Zuordnung von Bezeichner- und Zahlenwert-Produktion zu den programm-internen Kennworten erfolgt in den letzten zwei Zeilen:

```
crc             = number.
identifier      = "$ID".
number          = "$NUM".
```

Da der generierte Parser wissen muß, an welcher Stelle die Analyse der nicht-terminalen Symbole beginnt, muß das Start-symbol, das bereits definiert wurde, bezeichnet werden:

```
$prod_desc_file.
```

Die gesamte in BNF notierte GDA-Definition ist nochmals im Anhang aufgeführt, gefolgt von einer kleinen Beispieldatei, die einige Möglichkeiten dieser 'Sprache' zeigt. Die Ausgabeliste, die die Ergebnisse des Parsers zeigt, ist ebenfalls enthalten. Die darin ausgedruckten Informationen können direkt der semantischen Analyse zugeführt werden.

Wesentliche hier nicht weiter vertiefte Punkte sind:

1) die Makroanweisungen, für die hier nur Variablen-zuweisungen und Entities zugelassen sind. Die vorliegende Definition ist nicht ausreichend. Es wäre jedoch unnötig, an dieser Stelle einfache Sprach-strukturen zu definieren. Entsprechende geeignete BNF-Notationen für alle wichtigen Hochsprachen sind in der Literatur vorhanden /JeWi75, Kern83/ und können bei Bedarf in vereinfachter Form leicht eingebaut werden.

2) die verschiedenen verfügbaren Variablentypen. In der vorliegenden Form gibt es nur Integer-Werte. Dies ist natürlich nicht ausreichend, da auf jeden Fall Real-Werte für Koordinaten und Faktoren und Zeichenketten für Textdaten benötigt werden. Weitere Datentypen (vielleicht nur innerhalb von Makros) wären bool'sche und Zeichen-Variable.

3) der Mechanismus für die externe Referenz von Segment- und/oder Makro-Bibliotheksdateien.

4) die Eigenschaften der einzelnen Entities. Es wurden einige der weiter oben behandelten Entities vorgesehen. Die Parameter dieser Entites können sich an diesen Konzepten orientieren.

5.3. Schlußbemerkung

Das vorstehende Beispiel zeigt die Möglichkeiten, die bestehen, um eine übersichtliche und leistungsfähige Sprachdefinition für eine Anwendung wie den Austausch grafischer Daten bzw. von Produktdaten in Form von Dateien zu schaffen. Einige wichtige vorher festgestellte Anforderungen werden auf einfache Weise realisiert, z.T. bedingt durch den relativ hohen Abstraktionsgrad der Syntax. Nicht alle Möglichkeiten wurden ausgenutzt und ebenso sind einige wichtige Anforderungen noch nicht berücksichtigt worden. Dies ist jedoch prinzipiell möglich.

Bei der hier gewählten Lösung wurde besonderer Wert auf gute Lesbarkeit der Datei gelegt, um das Verständnis dieses Ansatzes zu erleichtern. Es ist aber ebenfalls möglich, im Hinblick auf den Platzbedarf entweder alternativ oder als einzige Möglichkeit die Kennworte z.B. durch kurze binäre Kennzahlen (sog. 'token') darzustellen. Der Parser kann auch solche Daten, genauso wie auch binäre Zahlenwerte, verarbeiten, da nur geringfügige Änderungen an dem Verfahren der lexikalischen Analyse notwendig sind. Es gibt eine Vielzahl von Möglichkeiten, um hier platzsparende Alternativen zu schaffen. Für die Darstellung binärer Zahlen sollte das Zahlenformat gemäß dem IEEE-Standard verwendet werden, /Cava85, S.446 ff./ und /Stev81/, da dieser in Zukunft maßgebend für Fließkomma-Verarbeitung sein wird.

Durch die Generierungsmöglichkeiten, verbunden mit automatischer vollständiger Validierung ist der wesentliche Nachteil der aufwendigen und daher teueren und unsicheren Implementation, der einer solchen Lösung sicherlich zugerechnet wird, verringert worden. Da die verfügbare Zeit

ebenso wie die Leistungsfähigkeit des verwendeten Tischrechners begrenzt war, kann angenommen werden, daß ein solches Projekt unter Anwendung der heute verfügbaren Möglichkeiten sogar recht schnell und kostengünstig realisert werden kann /Thom84/. Durch die zwangsläufig 'umsonst' erhaltene Validierung ist eine solche Lösung dann sicherlich konkurrenzfähig.

Literatur

/Ang83/ Angell, Ian O.: Graphische Datenverarbeitung, München, 1983.

/Blum84/ Blume, Ch.; Frommherz, B.: IRDATA - Eine Einführung, in: Elektronik, Heft 25, 1984, S.60-66.

/Cava85/ Cavanagh, J.J.F: Digital Computer Arithmetic, New York, 1985.

/DATEXP/ DATEX-P Handbuch der Deutschen Bundespost, 1982.

/DFNG84/ DFN: Grafische Kommunikation in offenen Netzen, Version 2.0, Berlin, 1984.

/DFNM85/ DFN-Mitteilungen, Heft 1, 1985, S.10.

/DFNP84/ DFN: Protokollhandbuch, Version 1, Berlin, 1984.

/DIN/ DIN 66301: Format zum Austausch geometrischer Informationen, Berlin, 1984.

/Fis83/ Fischer, Wolf E.: Datenbanksystem für CAD-Arbeitsplätze, Informatik-Fachberichte der GI, Bd. 70, Berlin, 1983.

/Foley82/ Foley, J.D.; Van Dam, A.: Fundamentals of Interactive Computer Graphics, Reading, 1983.

/Haas84/ Haas, J.; Hofmann, W.: Die V.- und X.-Empfehlungen des CCITT, in: Datenkommunikation (Elektronik Sonderheft), 1984, S.119-122.

/Hopg83/ Hopgood, F.R.A. et.al.: Introduction to the Graphical Kernel System (GKS), London, 1983.

/GKS82/ ISO/DIS 7942: Information Processing - Graphical Kernel System (GKS) -Functional Description: GKS Version 7.2, ISO/TC97/SC5/WG2 N163 (1982).

/IGES83/ Initial Graphics Exchange Specification (IGES), Version 2.0, NBSIR 82-2631 (AF) (National Bureau of Standards), 1983.

/JeWi75/ Jensen, K.; Wirth, N.: PASCAL, User Manual and Report, New York, 1975.

/Kafka84/ Kafka, Gerhard: Einführung in die Datenfernverarbeitung, in: Datenkommunikation (Elektronik Sonderheft), 1984, S.3-85.

/Kern83/ Kernighan, Brian. W.; Ritchie, D.M: The C Programming Language, Englewood Cliffs, 1977.

/Kell86/ Keller, Rudolf: Frust für Fehlerteufel, in: c't, Heft 2, 1986, S.54-56.

/Koch85/ Koch, Klaus: Daten auf Draht, in: Chip, Heft 10, 1985, S.234-239.

/Mart81/ Martin, James: Computer Networks and distributed processing, Englewood Cliffs, 1981.

/Newm79/ Newman, W.M.; Sproull, R.F.: Principles of Interactive Computer Graphics, New York, 1979.

/Rein81/ Reinauer, Gerd: Der Aufbau von anwendergerechten CAD-Systemen, Schriftenreihe der österreichischen Computergesellschaft, Bd. 13, 1981.

/SFSB85/ Schiffsform-Schnittstelle (SFS), 3.Entwurf, Institut f. Schiffs- und Meerestechnik, TU Berlin, 1985.

/Spies85/ Spies, Gaby: Im Vergleich: Lokale Netzwerke für UNIX-Systeme, in: Elektronik, Heft 19, 1985, S.195-200.

/Stev81/ Stevenson, D: A Proposed Standard for Binary Floating-Point Arithmetic, in: IEEE Computer, Heft 3, 1981, S.51-62.

/Thom84/ Thomas, Ernst: Compilerbau unter UNIX, in: Elektronik, Heft 22, 1984, S.96-100.

/VDA86/ VDA-Flächenschnittstelle, Version 2.0, Entwurf, Verband der Automobilindustrie, Frankfurt, 1986.

/Wirth84/ Wirth, Niklaus: Compilerbau, Stuttgart, 1984.

/Zima82/ Zima, Hans: Compilerbau, Bd. 1, Mannheim, 1982.

Anhang

```
 1:  0   /*      bl.c - GetSym, find, error, link, makeident, maketerm,
 2:  0                          termlist, factor, term, expression,
 3:  0                          getch, GetTarget, scanterm, isterm, ispterm,
 4:  0                          parse, main                                    */
 5:  0
 6:  0   #include    <stdio.h>
 7:  0
 8:  0   #define     NIL     0
 9:  0   #define     ESC     '\033'
10:  0
11:  0   #define     IDENT_SIZE      20
12:  0   #define     MAXT            100
13:  0
14:  0   /*      default lexical terminal symbols        */
15:  0   #define     IDENT           "$ID"
16:  0   #define     NUMBER          "$NUM"
17:  0
18:  0   /*#define DEBUG 0        /* un-comment for full debug    */
19:  0
20:  0   struct Node {
21:  1              struct  Node    *suc, *alt;
22:  1              char    terminal;
23:  1              union {
24:  2                  char        *tsym;
25:  2                  int *nsym;          /* pointer to Header    */
26:  2                  } usym;
27:  1              };      /* Node */
28:  0
29:  0   struct      Header {
30:  1              char    *sym;
31:  1              struct  Node    *entry;
32:  1              struct  Header  *succ;
33:  1              };  /* Header */
34:  0
35:  0   struct      Header  *list, *sentinel, *hd;
36:  0   struct      Node    *q, *r, *s;
37:  0   char        sym, noerr;
38:  0   char        *ts [MAXT];
39:  0   int tssize = 0;
40:  0   char        empty [] = "";
41:  0   char        down;                   /* parser direction flag        */
42:  0   int plevel;                  /* parser tree level            */
43:  0   char    pident [IDENT_SIZE+1];
44:  0   char    idname [IDENT_SIZE+1];
45:  0   int     numbval;
46:  0
47:  0   FILE    InFile;                 /* EBNF input file      */
48:  0   FILE    OutFile;                /* parser output file   */
49:  0
50:  0
51:  0   /* GetSym - get next nonblank char for bnf parsing      */
52:  0   GetSym ()
53:  0   {
54:  1    do {
55:  2       sym = getc (InFile);
56:  2       putchar (sym);
57:  2    } while (sym <= ' ' && sym != EOF && sym != CPMEOF);
```

```
 58:  1      }   /* GetSym */
 59:  0
 60:  0
 61:  0      /* find - lookup nonterminal symbol and insert new one  */
 62:  0      find (s, h)
 63:  0      char        *s;
 64:  0      struct      Header  **h;
 65:  0      {
 66:  1       struct     Header  *h1;
 67:  1
 68:  1      #ifdef DEBUG
 69:  1       fprintf (OutFile, "find symbol '%s'", s);
 70:  1      #endif
 71:  1       h1 = list;
 72:  1       sentinel->sym = s;
 73:  1       while (strcmp (h1->sym, s))
 74:  1          h1 = h1->succ;
 75:  1       if (h1 == sentinel) {                /* insert */
 76:  2      #ifdef DEBUG
 77:  2          fprintf (OutFile, "*insert*\n");
 78:  2      #endif
 79:  2          sentinel = alloc (sizeof (struct Header));
 80:  2          h1->succ = sentinel;
 81:  2          h1->entry = NIL;
 82:  2          *h = h1;
 83:  2          return (NO);
 84:  2          }
 85:  1      #ifdef DEBUG
 86:  1       else
 87:  1          putc ('\n', OutFile);
 88:  1      #endif
 89:  1       *h = h1;
 90:  1       return (YES);              /* already defined and found    */
 91:  1      }   /* find */
 92:  0
 93:  0
 94:  0      /* error - report errors */
 95:  0      void error (msg)
 96:  0       char       *msg;
 97:  0      {
 98:  1       fprintf (OutFile, "\nincorrect syntax: %s\n",msg);
 99:  1       fprintf (ERROUT, "\nincorrect syntax: %s\n",msg);
100:  1       noerr = FALSE;
101:  1      }   /* error */
102:  0
103:  0
104:  0      /* link - insert parse control tree links       */
105:  0      void link (p, q)            /*  (p, q: NodePtr);    */
106:  0       struct     Node    *p, *q;
107:  0      {                          /* insert q in  places indicated by linked chain p */
108:  1       struct     Node    *t;
109:  1
110:  1       while (p != NIL) {
111:  2          t = p;  p = t->suc;  t->suc = q;
112:  2          }
113:  1      }   /* link */
114:  0
```

```
115:  0
116:  0     /* makeident - read bnf input by nonterminal symbol rules       */
117:  0     void makeident (ident)
118:  0      char       *ident;
119:  0     {
120:  1      char       *i;
121:  1
122:  1      i = ident;
123:  1      while (isdigit (sym) || isalpha (sym) || sym == '_') {
124:  2         *ident++ = sym;
125:  2         GetSym ();
126:  2         }
127:  1     /*  if (sym <= ' ' && sym != EOF && sym != CPMEOF)
128:  1         GetSym;
129:  1     */
130:  1      *ident = 0;
131:  1     #ifdef DEBUG
132:  1      fprintf (OutFile, "makeident returns '%s'\n", i);
133:  1     #endif
134:  1     }   /* makeident    */
135:  0
136:  0
137:  0     /* maketerm - read bnf input by terminal symbol rules   */
138:  0     void maketerm (ident)
139:  0      char       *ident;
140:  0     {
141:  1      char       *i;
142:  1
143:  1      i = ident;
144:  1      while (sym != '"' && (ident-i) <= IDENT_SIZE ) {
145:  2         *ident++ = sym;
146:  2         GetSym ();
147:  2         }
148:  1      if (sym != '"')
149:  1         error ("unclosed terminal symbol string");
150:  1      else
151:  1         GetSym ();
152:  1      *ident = 0;
153:  1     #ifdef DEBUG
154:  1      fprintf (OutFile, "maketerm returns '%s'\n", i);
155:  1     #endif
156:  1     }   /* maketerm     */
157:  0
158:  0
159:  0     /* termlist - lookup terminal symbol and add new one into list  */
160:  0     void termlist (sym)
161:  0      char       *sym;
162:  0     {
163:  1      int        i, j;
164:  1
165:  1     #ifdef DEBUG
166:  1      fprintf (OutFile, "termlist - enter '%s', ", sym);
167:  1     #endif
168:  1      for (i = 0; i < tssize; i++) {
169:  2         if ( (j = strcmp (ts [i], sym)) == 0) {
170:  3     #ifdef DEBUG
171:  3             fprintf (OutFile, "already in table at %d\n", i+1);
```

```
172:  3     #endif
173:  3             return;
174:  3             }
175:  2         if (j > 0) {
176:  3             if (tssize >= MAXT) {
177:  4     #ifdef DEBUG
178:  4                 error ("too many terminal symbols");
179:  4     #endif
180:  4                 exit ();
181:  4                 }
182:  3             for (j = tssize++; j > i; j--)              /* copy all pointers one
183:  3                 ts [j] = ts [j-1];
184:  3             ts [i] = alloc (strlen (sym)+1);
185:  3             strcpy (ts [i], sym);
186:  3     #ifdef DEBUG
187:  3             fprintf (OutFile, "new size is %d\n", tssize);
188:  3     #endif
189:  3             return;
190:  3             }
191:  2         }
192:  1
193:  1      if (tssize >= MAXT) {
194:  2         error ("too many terminal symbols");
195:  2         exit ();
196:  2         }
197:  1      ts [tssize] = alloc (strlen(sym)+1);
198:  1      strcpy (ts [tssize++], sym);
199:  1     #ifdef DEBUG
200:  1      fprintf (OutFile, "new size is %d\n", tssize);
201:  1     #endif
202:  1     }   /* termlist     */
203:  0
204:  0
205:  0     /* factor - analyse bnf 'factor' syntax structure       */
206:  0     void factor (p, q, r, s)
207:  0      struct     Node    **p, **q, **r, **s;
208:  0     {
209:  1      auto       struct      Node    *a;
210:  1      auto       struct      Header  *h;
211:  1      auto       char        *ident;
212:  1
213:  1      if ('a' <= tolower (sym) && tolower (sym) <= 'z') {  /* nonterminal symbol
214:  2         ident = alloc (IDENT_SIZE+1);
215:  2         makeident (ident);
216:  2         fprintf (OutFile, "nonterminal symbol\t'%s'\n", ident);
217:  2         find (ident, &h);
218:  2         a = alloc (sizeof (struct Node));
219:  2         a->terminal = FALSE;
220:  2         a->usym.nsym = (int *) h;
221:  2         a->alt  = NIL;
222:  2         a->suc = NIL;
223:  2         *p = *q = *r = *s = a;
224:  2         }
225:  1      else if (sym == '"' ) {       /* terminal symbol */
226:  2         GetSym ();
227:  2         ident = alloc (IDENT_SIZE + 1);
228:  2         maketerm (ident);
```

```
229:  2          fprintf (OutFile, "terminal symbol\t\t'%s'\n", ident);
230:  2          termlist (ident);
231:  2          a = alloc (sizeof (struct Node));
232:  2          a->terminal = TRUE;
233:  2          a->usym.tsym = ident;
234:  2          a->alt = NIL;
235:  2          a->suc = NIL;
236:  2          *p = *q = *r = *s = a;
237:  2          }
238:  1       else if (sym == '(') {
239:  2          GetSym ();
240:  2          expression (p, q, r, s);
241:  2          if (sym == ')') {
242:  3              GetSym ();
243:  3              }
244:  2          else
245:  2              error ("unclosed parenthesis");
246:  2          }
247:  1       else if (sym == '[') {
248:  2          GetSym ();
249:  2          expression (p, q, r, s);
250:  2          a = alloc (sizeof (struct Node));
251:  2          a->terminal = TRUE;
252:  2          a->usym.tsym = empty;
253:  2          a->alt  = NIL;
254:  2          a->suc  = NIL;
255:  2          (*q)->alt = (*s)->suc = a;
256:  2          *q = *s = a;
257:  2          if (sym == ']') {
258:  3              GetSym ();
259:  3              }
260:  2          else
261:  2              error ("unclosed brackets");
262:  2          }
263:  1       else if (sym == '{') {
264:  2          GetSym ();
265:  2          expression (p, q, r, s);
266:  2          link (*r, *p);
267:  2          a = alloc (sizeof (struct Node));
268:  2          a->terminal = TRUE;
269:  2          a->usym.tsym = empty;
270:  2          a->alt = NIL;
271:  2          a->suc = NIL;
272:  2          (*q)->alt = a;
273:  2          *q = *r = *s = a;
274:  2          if (sym == '}') {
275:  3              GetSym ();
276:  3              }
277:  2          else
278:  2              error ("unclosed braces");
279:  2          }
280:  1       else
281:  1          error ("unexpected symbol");
282:  1       }   /* factor */
283:  0
284:  0
285:  0       /* term - analyse bnf term syntax structure     */
```

```
286:  0   void term (p, q, r, s)
287:  0    struct     Node    **p, **q, **r, **s;
288:  0   {
289:  1    auto       struct      Node    *p1, *q1, *r1, *s1;
290:  1
291:  1    factor (p, q, r, s);
292:  1    while (('a' <= tolower (sym) && tolower (sym) <= 'z') ||
293:  1                   sym == '"' || sym == '(' || sym == '[' || sym == '{') {
294:  2       factor (&p1, &q1, &r1, &s1);
295:  2       link (*r, p1);
296:  2       *r = r1; *s = s1;
297:  2       }
298:  1   }   /* term */
299:  0
300:  0
301:  0   /* expression - analyse bnf expression syntax structure */
302:  0   void expression (p, q, r, s)        /* (Var p, q, r, s : NodePtr);  */
303:  0    struct     Node    **p, **q, **r, **s;
304:  0   {
305:  1    auto       struct      Node    *q1, *s1;
306:  1
307:  1    term (p, q, r, s);
308:  1    while (sym == '|') {
309:  2       GetSym ();
310:  2       term (&((*q)->alt), &q1, &((*s)->suc), &s1);
311:  2       *q = q1; *s = s1;
312:  2       }
313:  1   }   /* expression   */
314:  0
315:  0
316:  0   getch ()                  /* get next InFile char */
317:  0   {
318:  1    char       c;
319:  1
320:  1    c = getc (InFile);
321:  1    putchar (c);
322:  1    return (c);
323:  1   }   /* getch        */
324:  0
325:  0
326:  0   /* GetTarget - program input scanner    */
327:  0   void GetTarget ()
328:  0   {
329:  1    char       *i;
330:  1
331:  1    i = pident;
332:  1    if (sym <= ' ')
333:  1       while ( (sym = getch ()) <= ' ' && sym != EOF)
334:  1           ;
335:  1
336:  1    if (sym != EOF) {
337:  2       if (isdigit (sym)) {
338:  3           do {
339:  4               *i++ = sym;
340:  4               sym = getch ();
341:  4           } while (isdigit (sym));
342:  3           *i = 0;
```

```
343:   3              if (!isterm (pident)) {
344:   4                  numbval = atoi (pident);
345:   4                  strcpy (pident, NUMBER);
346:   4                  }
347:   3      #ifdef DEBUG
348:   3              fprintf (OutFile, "GetTarget returns '%s', value is '%d'\n", pident,
349:   3      #endif
350:   3              }
351:   2          else if (isalpha (sym) || sym == '_') {
352:   3              do {
353:   4                  *i++ = sym;
354:   4                  sym = getch ();
355:   4              } while (isdigit (sym) || isalpha (sym) || sym == '_');
356:   3              *i = 0;
357:   3              if (!isterm (pident)) {
358:   4                  strcpy (idname, pident);
359:   4                  strcpy (pident, IDENT);
360:   4      #ifdef DEBUG
361:   4                  fprintf (OutFile, "GetTarget returns '%s', ident is '%s'\n", pic
362:   4      #endif
363:   4                  }
364:   3      #ifdef DEBUG
365:   3              else
366:   3                  fprintf (OutFile, "GetTarget returns '%s'\n", pident);
367:   3      #endif
368:   3              }
369:   2          else {
370:   3              FOREVER {
371:   4                  *i++ = sym; *i = 0;
372:   4                  if (! ispterm (pident))
373:   4                      break;
374:   4                  sym = getch ();
375:   4              }
376:   3              *(--i) = 0;                         /* delete wrong char     */
377:   3              if (strlen (pident) == 0) {
378:   4                  error ("unknown lexical item '%c'", sym);
379:   4                  sym = ' ';  /* delete char, otherwise trapped here  */
380:   4                  }
381:   3      #ifdef DEBUG
382:   3              fprintf (OutFile, "GetTarget returns '%s'\n", pident);
383:   3      #endif
384:   3              }
385:   2          }
386:   1      }   /* GetTarget    */
387:   0
388:   0
389:   0      /* scanterm - print list of terminal symbols    */
390:   0      void scanterm ()
391:   0      {
392:   1       int        i;
393:   1
394:   1       fprintf (OutFile, "\n\n--- list of terminal symbols ---\n\n");
395:   1       for (i = 0; i < tssize; i++) {
396:   2          fprintf (OutFile, "#%3d - %s\n", i, ts [i]);
397:   2          }
398:   1      }   /* scanterm     */
399:   0
```

```
400:  0
401:  0     /* isterm - check if symbol is terminal */
402:  0     isterm (s)
403:  0      char       *s;
404:  0     {
405:  1      int        i;
406:  1
407:  1      for (i = 0; i < tssize; i++) {
408:  2         if (strcmp (ts [i], s) == 0)
409:  2             return (YES);
410:  2         }
411:  1      return (NO);
412:  1     }   /* isterm       */
413:  0
414:  0
415:  0     /* ispterm - check if symbol is part of terminal        */
416:  0     ispterm (s)
417:  0      char       *s;
418:  0     {
419:  1      int        i;
420:  1
421:  1      for (i = 0; i < tssize; i++) {
422:  2         if (index (ts [i], s) == 0)
423:  2             return (YES);
424:  2         }
425:  1      return (NO);
426:  1     }   /* ispterm      */
427:  0
428:  0
429:  0     /* parse - parse program under control of bnf tree      */
430:  0     void parse (goal, match)
431:  0      struct     Header  *goal;
432:  0      char       *match;
433:  0     {
434:  1      auto       struct      Node    *s;
435:  1      auto       char        first;
436:  1
437:  1      if (!down) {
438:  2         fprintf (OutFile, "\ngoal symbol(s):");
439:  2         down = YES;
440:  2         }
441:  1      plevel++;
442:  1      first = YES;
443:  1
444:  1      s = goal->entry;
445:  1
446:  1      do {
447:  2         if (s->terminal) {
448:  3             if (strcmp (s->usym.tsym, pident) == 0) {
449:  4                 if (down) {
450:  5                     if (first)
451:  5                         fprintf (OutFile, "   '%s'\n", goal->sym);
452:  5                     else
453:  5                         putc ('\n', OutFile);
454:  5                     down = first = NO;
455:  5                     }
456:  4                             /* display parse analysis report        */
```

```
457:  4                 fprintf (OutFile, "at level %3d matched:\t'%s'", plevel, pident
458:  4                 if (strcmp (pident, IDENT) == 0)
459:  4                     fprintf (OutFile, "\tidentifier '%s'", idname);
460:  4                 else if (strcmp (pident, NUMBER) == 0)
461:  4                     fprintf (OutFile, "\tvalue '%d'", numbval);
462:  4                 putc ('\n', OutFile);
463:  4
464:  4                 *match = TRUE;
465:  4                 GetTarget ();
466:  4                 }
467:  3             else {
468:  4                 *match = (s->usym.tsym == empty);
469:  4                 if (!match) {
470:  5                     first = NO;
471:  5                     fprintf (OutFile, "   '%s'", goal->sym);
472:  5                     }
473:  4                 }
474:  3             }
475:  2         else {
476:  3             if (first) {
477:  4                 fprintf (OutFile, "   '%s'", goal->sym);
478:  4                 first = NO;
479:  4                 }
480:  3             parse (s->usym.nsym, match);
481:  3             }
482:  2         s = *match ? s->suc : s->alt;
483:  2       } while (s != NIL);
484:  1
485:  1       plevel --;
486:  1      #ifdef DEBUG
487:  1       fprintf (OutFile, "... parse exit\n");
488:  1      #endif
489:  1      }   /* parse        */
490:  0
491:  0
492:  0      main (argc, argv)                          /* ebnf parser */
493:  0       int         argc;
494:  0       char        *argv[];
495:  0      {
496:  1       int         l;
497:  1       char        *ident;
498:  1       char        *table, *tblend;
499:  1       struct      Header  *hdp;                  /* parser goal  */
500:  1       char        fn [15], fnp [15];             /* filename buffers     */
501:  1
502:  1       table = alloc (1);
503:  1
504:  1       if (argc <= 1)
505:  1          strcpy (fn, "EBNF.BNF");
506:  1       else {
507:  2          strcpy (fn, argv [1]);
508:  2          if (index (fn, ".") == -1) {
509:  3              strcpy (fnp, fn);
510:  3              strcat (fn, ".BNF");
511:  3              }
512:  2          else {
513:  3              strcpy (fnp, fn);
```

```
514:  3              fnp [index (fn, ".")] = 0;
515:  3              }
516:  2           }
517:  1        strcat (fnp, ".PRS");
518:  1
519:  1        if ((InFile = fopen (fn, "r")) == ERR) {
520:  2           fprintf (ERROUT, "%s: can't open", fn);
521:  2           exit ();
522:  2           }
523:  1        if ((OutFile = fopen (fnp, "w")) == ERR) {
524:  2           fprintf (ERROUT, "%s: can't open", fnp);
525:  2           exit ();
526:  2           }
527:  1        noerr = TRUE;
528:  1        sentinel = alloc (sizeof (struct Header));
529:  1        list = sentinel;
530:  1
531:  1           /* read productions and build data structure */
532:  1        fprintf (OutFile, "Backus Naur Form  language analysis\n\n");
533:  1        while (TRUE) {
534:  2           GetSym ();
535:  2           if (sym == '$')  break;
536:  2           ident = alloc (IDENT_SIZE + 1);
537:  2           makeident (ident);
538:  2           find (ident, &hd);
539:  2           if (sym == '=')
540:  2               GetSym ();
541:  2           else
542:  2               error ("'=' expected");
543:  2           expression (&(hd->entry), &q, &r, &s);
544:  2           link (r, NIL);
545:  2           if (sym != '.')
546:  2               error ("'.' expected");
547:  2           }  /* while */
548:  1
549:  1        if (noerr ) {          /* read goal symbol */
550:  2           GetSym ();
551:  2           ident = alloc (IDENT_SIZE + 1);
552:  2           makeident (ident);
553:  2           putchar ('\n');
554:  2           if (sym != '.')
555:  2               error ("'.' expected");
556:  2           if (find (ident, &hdp))
557:  2               fprintf (OutFile, "\nstart symbol is '%s'\n", ident);
558:  2           else {
559:  3               error ("start symbol not defined\n");
560:  3               exit ();
561:  3               }
562:  2            }
563:  1
564:  1        tblend = alloc (1);
565:  1        if (tblend <= 0)
566:  1           error ("dynamic memory overflow");
567:  1        else
568:  1           fprintf (OutFile, "\n%u bytes dynamic memory used\n", tblend - table);
569:  1
570:  1        if (noerr ) {
```

```
 571:  2          hd = list;
 572:  2                              /* check whether all symbols are defined */
 573:  2          while (hd != sentinel) {
 574:  3              if (hd->entry == NIL ) {
 575:  4                  fprintf (OutFile, "undefined symbol %s\n",hd->sym);
 576:  4                  fprintf (ERROUT, "undefined symbol %s\n",hd->sym);
 577:  4                  noerr = FALSE;
 578:  4                  }
 579:  3              hd = hd->succ;
 580:  3              }
 581:  2          }
 582:  1
 583:  1       scanterm ();
 584:  1
 585:  1       if (noerr) {                       /* read and parse sentences     */
 586:  2          sym = getc (InFile);
 587:  2          FOREVER {
 588:  3              GetTarget ();              /* write target symbol to pident        */
 589:  3              if (sym == CPMEOF || sym == EOF) break;
 590:  3              fprintf (OutFile, "\n\nparsing ...\n\n");
 591:  3              down = NO; plevel = 0;
 592:  3              parse (hdp, &noerr);
 593:  3              if (!noerr)
 594:  3                  fprintf (OutFile, " *** incorrect\n");
 595:  3              }
 596:  2          }
 597:  1       if (noerr) fprintf (OutFile, "\n\tno syntax errors\n");
 598:  1       fclose (InFile);
 599:  1       fclose (OutFile);
 600:  1      }  /* main ebnf parser */
```

cross reference list:

```
CPMEOF          :    57   589
DEBUG           :    68    76    85   131   153   165   170   177   186
                    199   347   360   364   381   486
EOF             :    57   333   336   589
ERR             :   519   523
ERROUT          :    99   520   524   576
ESC             :     9
FALSE           :   100   219   577
FILE            :    48
FOREVER         :   370   587
GetSym          :    52   125   146   151   226   239   242   248   258
                    264   275   309   534   540   550
GetTarget       :   327   465   588
Header          :    29    32    35    64    66    79   210   431   499
                    528
IDENT           :    15   359   458
IDENT_SIZE      :    11   144   214   227   536   551
InFile          :    55   320   519   586   598
MAXT            :    12    38   176   193
NIL             :     8    81   110   221   222   234   235   253   254
                    270   271   483   544   574
NO              :    83   411   425   454   470   478   591
NUMBER          :    16   345   460
Node            :    20    21    31    36   106   108   207   209   218
```

```
prod_desc_file  = { (symbol_assign ! definition ! entity) ";" }
                        check_record .
symbol_assign   = identifier "=" expression .
definition      = "segment" identifier "{" { statement } "}" !
                  "macro" identifier "(" [identifier {"," identifier} ] ")"
                        "{" { statement } "}" !
                  "extern"
                        ("segment" identifier !
                         "macro" identifier "(" number ")" ).
statement       = (symbol_assign ! entity) ";".
entity          = graphic_prim ! nongraph_prim ! segment_instance !
                        macro_instance ! gks_metafile_rec .
check_record    = "$CHECK" number "," number "," number
                        ["," crc "," crc "," crc ] "end".
graphic_prim    = (gp_keyword [identifier] [param_list]) !
                        ("transfm" [identifier] transform_expr).
nongraph_prim   = ngp_keyword [identifier] [param_list].
segment_instance= "draw" identifier [ ":" transform_expr ].
macro_instance  = "call" identifier "(" [param_list] ")" [":" transform_expr].
gks_metafile_rec= "GKSM" number {"," item }.
gp_keyword      = "point" ! "line" ! "vector" ! "curve" ! "surface" !
                        "rotsurf" ! "ratcurv" ! "ratsurf" !
                        "points" ! "circle" .
ngp_keyword     = "label" ! "attrib" ! "text" ! "dimen" ! "link" !
                        "value".
param_list      = parameter {"," parameter } .
parameter       = (":" transform_expr) !
                        ("@" "(" coor_list ")" ) !
                        ("&" [decl "="] entity) !
                        ("*" [decl "="] identifier) !
                        ([decl "="] (item ! "(" item {"," item} ")")).
transform_expr  = ("(" ["scale" coor_list] ["rot" coor_list]
                        ["trans" coor_list] ")" )
                        ! ("*" identifier).
decl            = "pos" ! "mag" ! "angle" ! "rad" ! "font" ! "color" ! "style".
coor_list       = item "," item ["," item].
expression      = ["+" ! "-"] term { ("+" ! "-") term }.
term            = factor { ("*" ! "/") factor }.
factor          = item ! "(" expression ")".
item            = identifier ! number.
crc             = number.
identifier      = "$ID".
number          = "$NUM".

$prod_desc_file.
```

```
        point   p1      @(Ø, Ø, Ø);
        point   p2      @(1, 1, 1);
        line            *p1, *p2, style = 4;

x1 = 5+(1Ø/3);
x2 = 1Ø;
dotted = 2;

        point   p3      @(x1, Ø, Ø), style = 1;
        point   p4      @(x2, Ø, Ø), style = 1, font = 3;
        line            *p3, *p4, style = dotted;


        transfm t1      (scale 2, 1);
        line            *p1, *p4, :*t1;

segment s1      {
        circle          @(x1, Ø, Ø), @(x2,Ø,Ø), angle = 1ØØ;
        line            @(x1, Ø, Ø), @(12, 12, Ø);
        };

macro   m1 (xx, yy, zz) {
        line            @(xx, yy, zz), @(Ø, Ø, Ø);
        };

extern  segment ex1;
extern  macro   mac1 (2);

        draw    s1:*t1;
        call    m1 (x1, 3, 5):(scale 2, 1);

$CHECK 3, 4, 11 end
```

Backus Naur Form language analysis

```
nonterminal symbol      'symbol_assign'
nonterminal symbol      'definition'
nonterminal symbol      'entity'
terminal symbol         ';'
nonterminal symbol      'check_record'
nonterminal symbol      'identifier'
terminal symbol         '='
nonterminal symbol      'expression'
terminal symbol         'segment'
nonterminal symbol      'identifier'
terminal symbol         '{'
nonterminal symbol      'statement'
terminal symbol         '}'
terminal symbol         'macro'
nonterminal symbol      'identifier'
terminal symbol         '('
nonterminal symbol      'identifier'
terminal symbol         ','
nonterminal symbol      'identifier'
terminal symbol         ')'
terminal symbol         '{'
nonterminal symbol      'statement'
terminal symbol         '}'
terminal symbol         'extern'
terminal symbol         'segment'
nonterminal symbol      'identifier'
terminal symbol         'macro'
nonterminal symbol      'identifier'
terminal symbol         '('
nonterminal symbol      'number'
terminal symbol         ')'
nonterminal symbol      'symbol_assign'
nonterminal symbol      'entity'
terminal symbol         ';'
nonterminal symbol      'graphic_prim'
nonterminal symbol      'nongraph_prim'
nonterminal symbol      'segment_instance'
nonterminal symbol      'macro_instance'
nonterminal symbol      'gks_metafile_rec'
terminal symbol         '$CHECK'
nonterminal symbol      'number'
terminal symbol         ','
nonterminal symbol      'number'
terminal symbol         ','
nonterminal symbol      'number'
terminal symbol         ','
nonterminal symbol      'crc'
terminal symbol         ','
nonterminal symbol      'crc'
terminal symbol         ','
nonterminal symbol      'crc'
terminal symbol         'end'
nonterminal symbol      'gp_keyword'
nonterminal symbol      'identifier'
nonterminal symbol      'param_list'
```



```
terminal symbol         'transfm'
nonterminal symbol      'identifier'
nonterminal symbol      'transform_expr'
nonterminal symbol      'ngp_keyword'
nonterminal symbol      'identifier'
nonterminal symbol      'param_list'
terminal symbol         'draw'
nonterminal symbol      'identifier'
terminal symbol         ':'
nonterminal symbol      'transform_expr'
terminal symbol         'call'
nonterminal symbol      'identifier'
terminal symbol         '('
nonterminal symbol      'param_list'
terminal symbol         ')'
terminal symbol         ':'
nonterminal symbol      'transform_expr'
terminal symbol         'GKSM'
nonterminal symbol      'number'
terminal symbol         ','
nonterminal symbol      'item'
terminal symbol         'point'
terminal symbol         'line'
terminal symbol         'vector'
terminal symbol         'curve'
terminal symbol         'surface'
terminal symbol         'rotsurf'
terminal symbol         'ratcurv'
terminal symbol         'ratsurf'
terminal symbol         'points'
terminal symbol         'circle'
terminal symbol         'label'
terminal symbol         'attrib'
terminal symbol         'text'
terminal symbol         'dimen'
terminal symbol         'link'
terminal symbol         'value'
nonterminal symbol      'parameter'
terminal symbol         ','
nonterminal symbol      'parameter'
terminal symbol         ':'
nonterminal symbol      'transform_expr'
terminal symbol         '@'
terminal symbol         '('
nonterminal symbol      'coor_list'
terminal symbol         ')'
terminal symbol         '&'
nonterminal symbol      'decl'
terminal symbol         '='
nonterminal symbol      'entity'
terminal symbol         '#'
nonterminal symbol      'decl'
terminal symbol         '='
nonterminal symbol      'identifier'
nonterminal symbol      'decl'
terminal symbol         '='
nonterminal symbol      'item'
```

```
terminal symbol         '('
nonterminal symbol      'item'
terminal symbol         ','
nonterminal symbol      'item'
terminal symbol         ')'
terminal symbol         '('
terminal symbol         'scale'
nonterminal symbol      'coor_list'
terminal symbol         'rot'
nonterminal symbol      'coor_list'
terminal symbol         'trans'
nonterminal symbol      'coor_list'
terminal symbol         ')'
terminal symbol         '*'
nonterminal symbol      'identifier'
terminal symbol         'pos'
terminal symbol         'mag'
terminal symbol         'angle'
terminal symbol         'rad'
terminal symbol         'font'
terminal symbol         'color'
terminal symbol         'style'
nonterminal symbol      'item'
terminal symbol         ','
nonterminal symbol      'item'
terminal symbol         ','
nonterminal symbol      'item'
terminal symbol         '+'
terminal symbol         '-'
nonterminal symbol      'term'
terminal symbol         '+'
terminal symbol         '-'
nonterminal symbol      'term'
nonterminal symbol      'factor'
terminal symbol         '*'
terminal symbol         '/'
nonterminal symbol      'factor'
nonterminal symbol      'item'
terminal symbol         '('
nonterminal symbol      'expression'
terminal symbol         ')'
nonterminal symbol      'identifier'
nonterminal symbol      'number'
nonterminal symbol      'number'
terminal symbol         '$ID'
terminal symbol         '$NUM'

start symbol is 'prod_desc_file'

6460 bytes dynamic memory used


--- list of terminal symbols ---

#  0 - $CHECK
#  1 - $ID
#  2 - $NUM
```



```
#  3 - &
#  4 - (
#  5 - )
#  6 - *
#  7 - +
#  8 - ,
#  9 - -
# 10 - /
# 11 - :
# 12 - ;
# 13 - =
# 14 - @
# 15 - GKSM
# 16 - angle
# 17 - attrib
# 18 - call
# 19 - circle
# 20 - color
# 21 - curve
# 22 - dimen
# 23 - draw
# 24 - end
# 25 - extern
# 26 - font
# 27 - label
# 28 - line
# 29 - link
# 30 - macro
# 31 - mag
# 32 - point
# 33 - points
# 34 - pos
# 35 - rad
# 36 - ratcurv
# 37 - ratsurf
# 38 - rot
# 39 - rotsurf
# 40 - scale
# 41 - segment
# 42 - style
# 43 - surface
# 44 - text
# 45 - trans
# 46 - transfm
# 47 - value
# 48 - vector
# 49 - {
# 50 - }


parsing ...


goal symbol(s):   'prod_desc_file'   'symbol_assign'   'entity'
'graphic_prim'   'gp_keyword'
at level   4 matched:   'point'
```

```
goal symbol(s):   'identifier'
at level   4 matched:   '$ID'   identifier 'p1'

goal symbol(s):   'param_list'   'parameter'
at level   5 matched:   '@'
at level   5 matched:   '('

goal symbol(s):   'coor_list'   'item'   'number'
at level   8 matched:   '$NUM'  value '0'
at level   6 matched:   ','

goal symbol(s):   'item'   'number'
at level   8 matched:   '$NUM'  value '0'
at level   6 matched:   ','

goal symbol(s):   'item'   'number'
at level   8 matched:   '$NUM'  value '0'
at level   5 matched:   ')'
at level   1 matched:   ';'

goal symbol(s):   'symbol_assign'   'entity'   'graphic_prim'   'gp_keyword'
at level   4 matched:   'point'

goal symbol(s):   'identifier'
at level   4 matched:   '$ID'   identifier 'p2'

goal symbol(s):   'param_list'   'parameter'
at level   5 matched:   '@'
at level   5 matched:   '('

goal symbol(s):   'coor_list'   'item'   'number'
at level   8 matched:   '$NUM'  value '1'
at level   6 matched:   ','

goal symbol(s):   'item'   'number'
at level   8 matched:   '$NUM'  value '1'
at level   6 matched:   ','

goal symbol(s):   'item'   'number'
at level   8 matched:   '$NUM'  value '1'
at level   5 matched:   ')'
at level   1 matched:   ';'

goal symbol(s):   'symbol_assign'   'entity'   'graphic_prim'   'gp_keyword'
at level   4 matched:   'line'

goal symbol(s):   'param_list'   'parameter'
at level   5 matched:   '*'

goal symbol(s):   'identifier'
at level   6 matched:   '$ID'   identifier 'p1'
at level   4 matched:   ','

goal symbol(s):   'parameter'
at level   5 matched:   '*'

goal symbol(s):   'identifier'
```



```
at level   6 matched:   '$ID'   identifier 'p2'
at level   4 matched:   ','

goal symbol(s):   'parameter'   'decl'
at level   6 matched:   'style'
at level   5 matched:   '='

goal symbol(s):   'item'   'number'
at level   7 matched:   '$NUM'  value '4'
at level   1 matched:   ';'

goal symbol(s):   'symbol_assign'   'identifier'
at level   3 matched:   '$ID'   identifier 'x1'
at level   2 matched:   '='

goal symbol(s):   'expression'   'term'   'factor'   'item'   'number'
at level   7 matched:   '$NUM'  value '5'
at level   3 matched:   '+'

goal symbol(s):   'term'   'factor'   'item'
at level   5 matched:   '('

goal symbol(s):   'expression'   'term'   'factor'   'item'   'number'
at level  10 matched:   '$NUM'  value '10'
at level   7 matched:   '/'

goal symbol(s):   'factor'   'item'   'number'
at level  10 matched:   '$NUM'  value '3'
at level   5 matched:   ')'
at level   1 matched:   ';'

goal symbol(s):   'symbol_assign'   'identifier'
at level   3 matched:   '$ID'   identifier 'x2'
at level   2 matched:   '='

goal symbol(s):   'expression'   'term'   'factor'   'item'   'number'
at level   7 matched:   '$NUM'  value '10'
at level   1 matched:   ';'

goal symbol(s):   'symbol_assign'   'identifier'
at level   3 matched:   '$ID'   identifier 'dotted'
at level   2 matched:   '='

goal symbol(s):   'expression'   'term'   'factor'   'item'   'number'
at level   7 matched:   '$NUM'  value '2'
at level   1 matched:   ';'

goal symbol(s):   'symbol_assign'   'entity'   'graphic_prim'   'gp_keyword'
at level   4 matched:   'point'

goal symbol(s):   'identifier'
at level   4 matched:   '$ID'   identifier 'p3'

goal symbol(s):   'param_list'   'parameter'
at level   5 matched:   '@'
at level   5 matched:   '('
```

```
goal symbol(s):  'coor_list'  'item'  'identifier'
at level   8 matched:  '$ID'  identifier 'x1'
at level   6 matched:  ','

goal symbol(s):  'item'  'number'
at level   8 matched:  '$NUM'  value '0'
at level   6 matched:  ','

goal symbol(s):  'item'  'number'
at level   8 matched:  '$NUM'  value '0'
at level   5 matched:  ')'
at level   4 matched:  ','

goal symbol(s):  'parameter'  'decl'
at level   6 matched:  'style'
at level   5 matched:  '='

goal symbol(s):  'item'  'number'
at level   7 matched:  '$NUM'  value '1'
at level   1 matched:  ';'

goal symbol(s):  'symbol_assign'  'entity'  'graphic_prim'  'gp_keyword'
at level   4 matched:  'point'

goal symbol(s):  'identifier'
at level   4 matched:  '$ID'  identifier 'p4'

goal symbol(s):  'param_list'  'parameter'
at level   5 matched:  '@'
at level   5 matched:  '('

goal symbol(s):  'coor_list'  'item'  'identifier'
at level   8 matched:  '$ID'  identifier 'x2'
at level   6 matched:  ','

goal symbol(s):  'item'  'number'
at level   8 matched:  '$NUM'  value '0'
at level   6 matched:  ','

goal symbol(s):  'item'  'number'
at level   8 matched:  '$NUM'  value '0'
at level   5 matched:  ')'
at level   4 matched:  ','

goal symbol(s):  'parameter'  'decl'
at level   6 matched:  'style'
at level   5 matched:  '='

goal symbol(s):  'item'  'number'
at level   7 matched:  '$NUM'  value '1'
at level   4 matched:  ','

goal symbol(s):  'parameter'  'decl'
at level   6 matched:  'font'
at level   5 matched:  '='

goal symbol(s):  'item'  'number'
```



```
at level   7 matched:  '$NUM'  value '3'
at level   1 matched:  ';'

goal symbol(s):  'symbol_assign'  'entity'  'graphic_prim'  'gp_keyword'
at level   4 matched:  'line'

goal symbol(s):  'param_list'  'parameter'
at level   5 matched:  '*'

goal symbol(s):  'identifier'
at level   6 matched:  '$ID'  identifier 'p3'
at level   4 matched:  ','

goal symbol(s):  'parameter'
at level   5 matched:  '*'

goal symbol(s):  'identifier'
at level   6 matched:  '$ID'  identifier 'p4'
at level   4 matched:  ','

goal symbol(s):  'parameter'  'decl'
at level   6 matched:  'style'
at level   5 matched:  '='

goal symbol(s):  'item'  'identifier'
at level   7 matched:  '$ID'  identifier 'dotted'
at level   1 matched:  ';'

goal symbol(s):  'symbol_assign'  'entity'  'graphic_prim'
at level   3 matched:  'transfm'

goal symbol(s):  'identifier'
at level   4 matched:  '$ID'  identifier 't1'

goal symbol(s):  'transform_expr'
at level   4 matched:  '('
at level   4 matched:  'scale'

goal symbol(s):  'coor_list'  'item'  'number'
at level   7 matched:  '$NUM'  value '2'
at level   5 matched:  ','

goal symbol(s):  'item'  'number'
at level   7 matched:  '$NUM'  value '1'
at level   4 matched:  ')'
at level   1 matched:  ';'

goal symbol(s):  'symbol_assign'  'entity'  'graphic_prim'  'gp_keyword'
at level   4 matched:  'line'

goal symbol(s):  'param_list'  'parameter'
at level   5 matched:  '*'

goal symbol(s):  'identifier'
at level   6 matched:  '$ID'  identifier 'p1'
at level   4 matched:  ','
```

```
goal symbol(s):   'parameter'
at level   5 matched:   '*'

goal symbol(s):   'identifier'
at level   6 matched:   '$ID'   identifier 'p4'
at level   4 matched:   ','

goal symbol(s):   'parameter'
at level   5 matched:   ';'

goal symbol(s):   'transform_expr'
at level   6 matched:   '*'

goal symbol(s):   'identifier'
at level   7 matched:   '$ID'   identifier 't1'
at level   1 matched:   ';'

goal symbol(s):   'symbol_assign'   'definition'
at level   2 matched:   'segment'

goal symbol(s):   'identifier'
at level   3 matched:   '$ID'   identifier 's1'
at level   2 matched:   '('

goal symbol(s):   'statement'   'symbol_assign'   'entity'   'graphic_prim'
'gp_keyword'
at level   6 matched:   'circle'

goal symbol(s):   'param_list'   'parameter'
at level   7 matched:   '@'
at level   7 matched:   '('

goal symbol(s):   'coor_list'   'item'   'identifier'
at level  10 matched:   '$ID'   identifier 'x1'
at level   8 matched:   ','

goal symbol(s):   'item'   'number'
at level  10 matched:   '$NUM'  value '0'
at level   8 matched:   ','

goal symbol(s):   'item'   'number'
at level  10 matched:   '$NUM'  value '0'
at level   7 matched:   ')'
at level   6 matched:   ','

goal symbol(s):   'parameter'
at level   7 matched:   '@'
at level   7 matched:   '('

goal symbol(s):   'coor_list'   'item'   'identifier'
at level  10 matched:   '$ID'   identifier 'x2'
at level   8 matched:   ','

goal symbol(s):   'item'   'number'
at level  10 matched:   '$NUM'  value '0'
at level   8 matched:   ','
```



```
goal symbol(s):   'item'   'number'
at level  10 matched:   '$NUM'  value '0'
at level   7 matched:   ')'
at level   6 matched:   ','

goal symbol(s):   'parameter'   'decl'
at level   8 matched:   'angle'
at level   7 matched:   '='

goal symbol(s):   'item'   'number'
at level   9 matched:   '$NUM'  value '100'
at level   3 matched:   ';'

goal symbol(s):   'statement'   'symbol_assign'   'entity'   'graphic_prim'
'gp_keyword'
at level   6 matched:   'line'

goal symbol(s):   'param_list'   'parameter'
at level   7 matched:   '@'
at level   7 matched:   '('

goal symbol(s):   'coor_list'   'item'   'identifier'
at level  10 matched:   '$ID'   identifier 'x1'
at level   8 matched:   ','

goal symbol(s):   'item'   'number'
at level  10 matched:   '$NUM'  value '0'
at level   8 matched:   ','

goal symbol(s):   'item'   'number'
at level  10 matched:   '$NUM'  value '0'
at level   7 matched:   ')'
at level   6 matched:   ','

goal symbol(s):   'parameter'
at level   7 matched:   '@'
at level   7 matched:   '('

goal symbol(s):   'coor_list'   'item'   'number'
at level  10 matched:   '$NUM'  value '12'
at level   8 matched:   ','

goal symbol(s):   'item'   'number'
at level  10 matched:   '$NUM'  value '12'
at level   8 matched:   ','

goal symbol(s):   'item'   'number'
at level  10 matched:   '$NUM'  value '0'
at level   7 matched:   ')'
at level   3 matched:   ';'

goal symbol(s):   'statement'   'symbol_assign'   'entity'   'graphic_prim'
'nongraph_prim'
at level   2 matched:   ')'
at level   1 matched:   ';'

goal symbol(s):   'symbol_assign'   'definition'
```

```
at level   2 matched:   'macro'

goal symbol(s):   'identifier'
at level   3 matched:   '$ID'   identifier 'm1'
at level   2 matched:   '('

goal symbol(s):   'identifier'
at level   3 matched:   '$ID'   identifier 'xx'
at level   2 matched:   ','

goal symbol(s):   'identifier'
at level   3 matched:   '$ID'   identifier 'yy'
at level   2 matched:   ','

goal symbol(s):   'identifier'
at level   3 matched:   '$ID'   identifier 'zz'
at level   2 matched:   ')'
at level   2 matched:   '('

goal symbol(s):   'statement'   'symbol_assign'   'entity'   'graphic_prim'
'gp_keyword'
at level   6 matched:   'line'

goal symbol(s):   'param_list'   'parameter'
at level   7 matched:   '@'
at level   7 matched:   '('

goal symbol(s):   'coor_list'   'item'   'identifier'
at level  10 matched:   '$ID'   identifier 'xx'
at level   8 matched:   ','

goal symbol(s):   'item'   'identifier'
at level  10 matched:   '$ID'   identifier 'yy'
at level   8 matched:   ','

goal symbol(s):   'item'   'identifier'
at level  10 matched:   '$ID'   identifier 'zz'
at level   7 matched:   ')'
at level   6 matched:   ','

goal symbol(s):   'parameter'
at level   7 matched:   '@'
at level   7 matched:   '('

goal symbol(s):   'coor_list'   'item'   'number'
at level  10 matched:   '$NUM'  value '0'
at level   8 matched:   ','

goal symbol(s):   'item'   'number'
at level  10 matched:   '$NUM'  value '0'
at level   8 matched:   ','

goal symbol(s):   'item'   'number'
at level  10 matched:   '$NUM'  value '0'
at level   7 matched:   ')'
at level   3 matched:   ';'
```



```
goal symbol(s):   'statement'   'symbol_assign'   'entity'   'graphic_prim'
'nongraph_prim'
at level   2 matched:   ')'
at level   1 matched:   ';'

goal symbol(s):   'symbol_assign'   'definition'
at level   2 matched:   'extern'
at level   2 matched:   'segment'

goal symbol(s):   'identifier'
at level   3 matched:   '$ID'   identifier 'ex1'
at level   1 matched:   ';'

goal symbol(s):   'symbol_assign'   'definition'
at level   2 matched:   'extern'
at level   2 matched:   'macro'

goal symbol(s):   'identifier'
at level   3 matched:   '$ID'   identifier 'mac1'
at level   2 matched:   '('

goal symbol(s):   'number'
at level   3 matched:   '$NUM'  value '2'
at level   2 matched:   ')'
at level   1 matched:   ';'

goal symbol(s):   'symbol_assign'   'entity'   'graphic_prim'
'nongraph_prim'   'segment_instance'
at level   3 matched:   'draw'

goal symbol(s):   'identifier'
at level   4 matched:   '$ID'   identifier 's1'
at level   3 matched:   ':'

goal symbol(s):   'transform_expr'
at level   4 matched:   '*'

goal symbol(s):   'identifier'
at level   5 matched:   '$ID'   identifier 't1'
at level   1 matched:   ';'

goal symbol(s):   'symbol_assign'   'entity'   'graphic_prim'   'nongraph_prim'
'macro_instance'
at level   3 matched:   'call'

goal symbol(s):   'identifier'
at level   4 matched:   '$ID'   identifier 'm1'
at level   3 matched:   '('

goal symbol(s):   'param_list'   'parameter'   'item'   'identifier'
at level   7 matched:   '$ID'   identifier 'x1'
at level   4 matched:   ','

goal symbol(s):   'parameter'   'item'   'number'
at level   7 matched:   '$NUM'  value '3'
at level   4 matched:   ','
```

```
goal symbol(s):  'parameter'  'item'  'number'
at level   7 matched:  '$NUM'  value '5'
at level   3 matched:  ')'
at level   3 matched:  ':'

goal symbol(s):  'transform_expr'
at level   4 matched:  '('
at level   4 matched:  'scale'

goal symbol(s):  'coor_list'  'item'  'number'
at level   7 matched:  '$NUM'  value '2'
at level   5 matched:  ','

goal symbol(s):  'item'  'number'
at level   7 matched:  '$NUM'  value '1'
at level   4 matched:  ')'
at level   1 matched:  ';'

goal symbol(s):  'symbol_assign'  'entity'  'graphic_prim'  'nongraph_prim'
'check_record'
at level   2 matched:  '$CHECK'

goal symbol(s):  'number'
at level   3 matched:  '$NUM'  value '3'
at level   2 matched:  ','

goal symbol(s):  'number'
at level   3 matched:  '$NUM'  value '4'
at level   2 matched:  ','

goal symbol(s):  'number'
at level   3 matched:  '$NUM'  value '11'
at level   2 matched:  'end'

       no syntax errors
```