Forschungszentrum Karlsruhe
Technik und Umwelt

Wissenschaftliche Berichte
FZKA 5998

Das VISART-Konzept einer standardisierten Schnittstelle zwischen Codes und Auswerteprogrammen

Teil 4: VISARTOP – ein Programm zum Bearbeiten von VISART-Dateien

S. Kleinheins

Institut für Neutronenphysik und Reaktortechnik
Projekt Nukleare Sicherheitsforschung

Juni 1999

Forschungszentrum Karlsruhe
Technik und Umwelt

Wissenschaftliche Berichte

FZKA 5998

Das VISART-Konzept einer standardisierten Schnittstelle zwischen Codes und Auswerteprogrammen

Teil 4: VISARTOP—ein Programm zum Bearbeiten von VISART-Dateien

Siegfried Kleinheins

Institut für Neutronenphysik und Reaktortechnik
Projekt Nukleare Sicherheitsforschung

Forschungszentrum Karlsruhe GmbH, Karlsruhe
1999

Als Manuskript gedruckt

Forschungszentrum Karlsruhe GmbH
Postfach 3640, 76021 Karlsruhe

Mitglied der Hermann von Helmholtz-Gemeinschaft
Deutscher Forschungszentren (HGF)

ISSN 0947-8620

Zusammenfassung

VISART ist ein standardisiertes Format für Postprocessor-Dateien mit Daten, wie sie von Fluiddynamik- und ähnlichen Codes über einer mehrdimensionalen Geometrie und der Zeit errechnet werden. Das Programm VISARTOP kann aus den in einer VISART-Datei gespeicherten Größen, z.B. zum Zwecke der grafischen Darstellung oder der Visualisierung, neue Größen ableiten und sie, wahlweise zusammen mit den alten Größen, im VISART-Format auf eine neue Datei schreiben. Die Vorschriften zur Verknüpfung von Größen zu neuen Größen werden durch arithmetische, logische und andere Ausdrücke in weitgehend vertrauter Schreibweise in der Eingabe von VISARTOP definiert, wobei die Größen lediglich durch ihre Identifikationen (d.h. die Namen, unter denen sie in der VISART-Datei stehen) spezifiziert werden. Daneben lassen sich mit VISARTOP noch einige weitere nützliche Operationen auf VISART-Dateien ausführen, wie Formatumwandlung, Zusammenfassung von Fortsetzungs-Files und Reduzierung des Dateiinhalts.

The VISART Concept of a Standardized Interface between Codes and Evaluation Programs

VISARTOP—a Program for Processing VISART Files

Abstract

VISART is a standardized format for postprocessor files with data calculated by fluid-dynamics and similar codes over a multi-dimensional geometry and time. The VISARTOP program can derive new quantities from the quantities stored in a VISART file, e.g. for the purpose of graphical display or visualization, and write them on a new file in the VISART format optionally along with the old quantities. The directives for combining quantities to new quantities are defined in the VISARTOP input by arithmetical, logical and other expressions in mostly familiar notation where the quantities are specified by their identifications only (i.e. the names under which they are stored in the VISART file). In addition, some other useful operations on VISART files can be performed with VISARTOP such as format conversion, combining of continuation files and reduction of the file contents.

Übersicht über das VISART-Konzept und seine Dokumentation

Welche Aufgabe ist zu lösen ?

Die rechnerische Simulation von Problemen der Fluiddynamik, Elektrodynamik, Neutronik usw. reduziert sich meist auf die numerische Lösung von zeitabhängigen partiellen Differentialgleichungen mittels räumlicher und zeitlicher Diskretisierung. Die dafür verwendeten Computer-Codes errechnen dabei eine Fülle von Daten über der Geometrie und der Zeit. Zu ihrer Auswertung sind diese Daten in geeigneter Form auf einer Postprocessor-Datei bereitzustellen.

Was ist VISART ?

VISART ist in erster Linie der Name eines (zunächst im FZK-Bereich) standardisierten Formats für Postprocessor-Dateien mit nach obiger Art errechneten Daten, die zur Auswertung erstellt werden, aber noch keine Steuerinformation für Grafik usw. enthalten. Im weiteren Sinne ist VISART der Name eines auf dieser Standardisierung aufbauenden Konzepts einer einheitlichen Schnittstelle zwischen Codes und Auswerteprogrammen, die es erlaubt, mit minimalem Implementierungsaufwand einer Vielzahl von dafür geeigneten Codes eine Vielzahl von Diensten zur Weiterbehandlung der berechneten Daten zur Verfügung zu stellen.

Welche Codes erstellen VISART-Dateien ?

Postprocessor-Ausgabe im VISART-Format wurde (Stand Dezember 1998) bereits in die folgenden, im FZK betreuten oder entwickelten Codes eingebaut: SIMMER-II.12, AFDM, SIMMER-III, HYDSOL, MATTINA, FLUTAN, KADI2D, MAX3D, BFCPIC, GASFLOW.

Was kann man mit VISART-Dateien anfangen ?

VISART-Dateien können zunächst einmal mit einem Dienstprogramm in wählbarem Umfang und Detail „durchblättert" werden. Mit einem anderen Programm können die Werte ausgewählter Größen gezielt über der Geometrie und/oder der Zeit in Zahlenform „angeschaut" werden.

Die graphische und anderweitige Auswertung von Größen einer VISART-Datei ist sodann mit einigen im Hause oder im Auftrag entwickelten Auswerteprogrammen möglich, die das VISART-Dateiformat direkt lesen können.

Die Visualisierung der Daten einer VISART-Datei mit kommerziellen Visualisierungssystemen (bis jetzt AVS/Express, Gsharp, Tecplot-7), die jeweils eigene Formate für ihre Eingabedateien verlangen, geschieht über das Zwischenschalten eines für VISART angefertigten Adapterprogramms, das die Eingabedatei des jeweiligen Systems für die gerade gewählten Größen und Darstellungskoordinaten (z.B. über der Geometrie—auch Schnitten oder Ausschnitten daraus—und/oder der Zeit) erstellt.

Schließlich steht ein mächtiges Werkzeug zum Bearbeiten von VISART-Dateien und zum Ableiten neuer Größen aus den in der VISART-Datei enthaltenen Daten zur Verfügung.

Alle erwähnten Dienste können über eine zentrale graphische Oberfläche auf X-Terminals für Workstations aufgerufen und gesteuert werden.

Wo findet man was ?

Teil 1 der VISART-Dokumentation führt in das VISART-Konzept ein und stellt die graphische Benutzeroberfläche zum Aufruf der Dienste vor. Das Programm VISARTUL zum Durchblättern und Prüfen von VISART-Dateien wird beschrieben, und es wird ein Überblick über die anderen zur Verfügung stehenden Dienste gegeben. (Dieser Teil ist die Grundlage für die nachfolgenden Teile.)

Teil 2 der VISART-Dokumentation definiert den VISART-Standard für den Aufbau der Postprocessor-Dateien.

Teil 3 der VISART-Dokumentation beschreibt das Adapterprogramm VISAPTER zum Erstellen von Eingabedateien für die Visualisierung mit AVS/Express, Gsharp, Tecplot-7 usw. und zum Anschauen ausgewählter Werte in Zahlenform.

Teil 4 der VISART-Dokumentation beschreibt das Werkzeug VISARTOP zum Bearbeiten von VISART-Dateien und zum Ableiten neuer Größen.

Teil 5 der VISART-Dokumentation beschreibt das Adapterprogramm VISARVOL zum Erstellen von Eingabedateien für die zweidimensionale Darstellung von Volumenanteilen mit Tecplot usw..

Auswerteprogramme, die VISART-Dateien direkt lesen und verarbeiten können, werden nicht in dieser Reihe, sondern eigenständig dokumentiert. Das gleiche gilt für den Einbau der VISART-Schnittstelle in die einzelnen Codes.

Inhalt

1. Einleitung

VISART ist der Name eines (zunächst im FZK-Bereich) standardisierten Formats für Postprocessor-Dateien [1b] mit über der Geometrie und der Zeit errechneten Daten, wie sie z.B. von Fluiddynamik-Codes zur anschließenden grafischen oder sonstigen Auswertung erstellt werden.

Die Postprocessor-Ausgabe eines Codes wird, schon aus Platzgründen, nur die Werte einer beschränkten Anzahl von Größen umfassen. Bei der Erstellung des Codes und der Programmierung der Ausgabe ist meist auch noch nicht vollständig absehbar, welche Größen beim Postprocessing später einmal benötigt werden könnten. In der Vergangenheit haben deshalb spezielle Auswerteprogramme (z.B. der T6P-Postprocessor [3, Vol. VIII] für die SIMMER- und AFDM-Codes) zum Zwecke der grafischen Darstellung weitere Größen aus den auf der Postprocessor-Datei angebotenen abgeleitet. Dies war jedoch beschränkt auf wenige, in den Auswerteprogrammen vorgesehene, Größen.[1]

Im VISART-Konzept einer einheitlichen Schnittstelle zwischen Codes und Auswerteprogrammen ist eine Bearbeitung und Ableitung von Größen unabhängig von den erzeugenden Codes und von den Auswerteprogrammen naheliegend. Zweckmäßigerweise geschieht dies mit einem eigenen Bearbeitungswerkzeug, das auf eine VISART-Datei angewendet wird und eine neue VISART-Datei mit den bearbeiteten und abgeleiteten Größen erzeugt (Bild 1).

Das hier beschriebene Programm VISARTOP ist in erster Linie für die Erzeugung neuer Größen aus vorhandenen Größen einer VISART-Datei durch arithmetische, logische und andere Operationen gedacht. Die in die Operationen eingehenden Gruppen werden in der Steuereingabe durch ihre Identifikationen spezifiziert; ihre Verknüpfungen zu neuen Größen werden durch arithmetische, logische und andere Ausdrücke in weitgehend vertrauter Schreibweise angegeben. An die Identifikationen angehängte Qualifikatoren erlauben die Auswahl von Vektorkomponenten zur Bildung von skalaren Gruppen oder die Bildung von vektoriellen Gruppen aus skalaren Komponenten, die Auswahl von Werten vorhandener Gruppen zur Bildung von Gruppen über Subnetzen oder von Gruppenelementen, oder die Bildung von Gruppen über dem Netz aus Gruppen über Subnetzen oder Einzeldaten. Die Resultate der Operationen werden als Gruppen auf die neue VISART-Datei geschrieben, in der Regel zusätzlich zu den bereits vorhandenen Gruppen.

Daneben lassen sich mit VISARTOP noch andere Operationen auf eine VISART-Datei als ganze oder auf Gruppen derselben ausführen:

- Die Umwandlung einer unformatierten Datei in eine formatierte und umgekehrt.
- Die Vereinigung von Fortsetzungs-Files (aus Restart-Läufen) zu einer alle Files umfassenden Datei.
- Die Reduktion des Dateiumfangs durch Auswahl von Problemzyklen und/oder von Gruppen.

VISARTOP ist ein Fortran-Programm, das im Stapelbetrieb aufgerufen wird, und zwar am besten über eine grafische Benutzeroberfläche, mit der die Steuereingabe zusammengestellt wird. Die grafische Benutzeroberfläche wurde bereits im Teil 1 der VISART-Dokumentation [1a] vorgestellt. Im vorliegenden Bericht, dem Teil 4 der VISART-Dokumentation, werden VISARTOP, seine Steuereingabe, seine Ein- und Ausgabedateien und Anwendungen des Programms beschrieben. (Der Teil 4 ist eine aktualisierte Neufassung eines unveröffentlichten Berichts vom Juli 1993.) Zum Verständnis des Berichts ist die Vertrautheit mit dem einführenden Teil 1 der Dokumentation [1a] Voraussetzung, besonders bezüglich der verwendeten Terminologie.

[1] Neuerdings bieten auch kommerzielle Visualisierungssysteme wie Gsharp und Tecplot-7 die Möglichkeit, die darzustellenden Daten zu modifizieren und zu kombinieren.

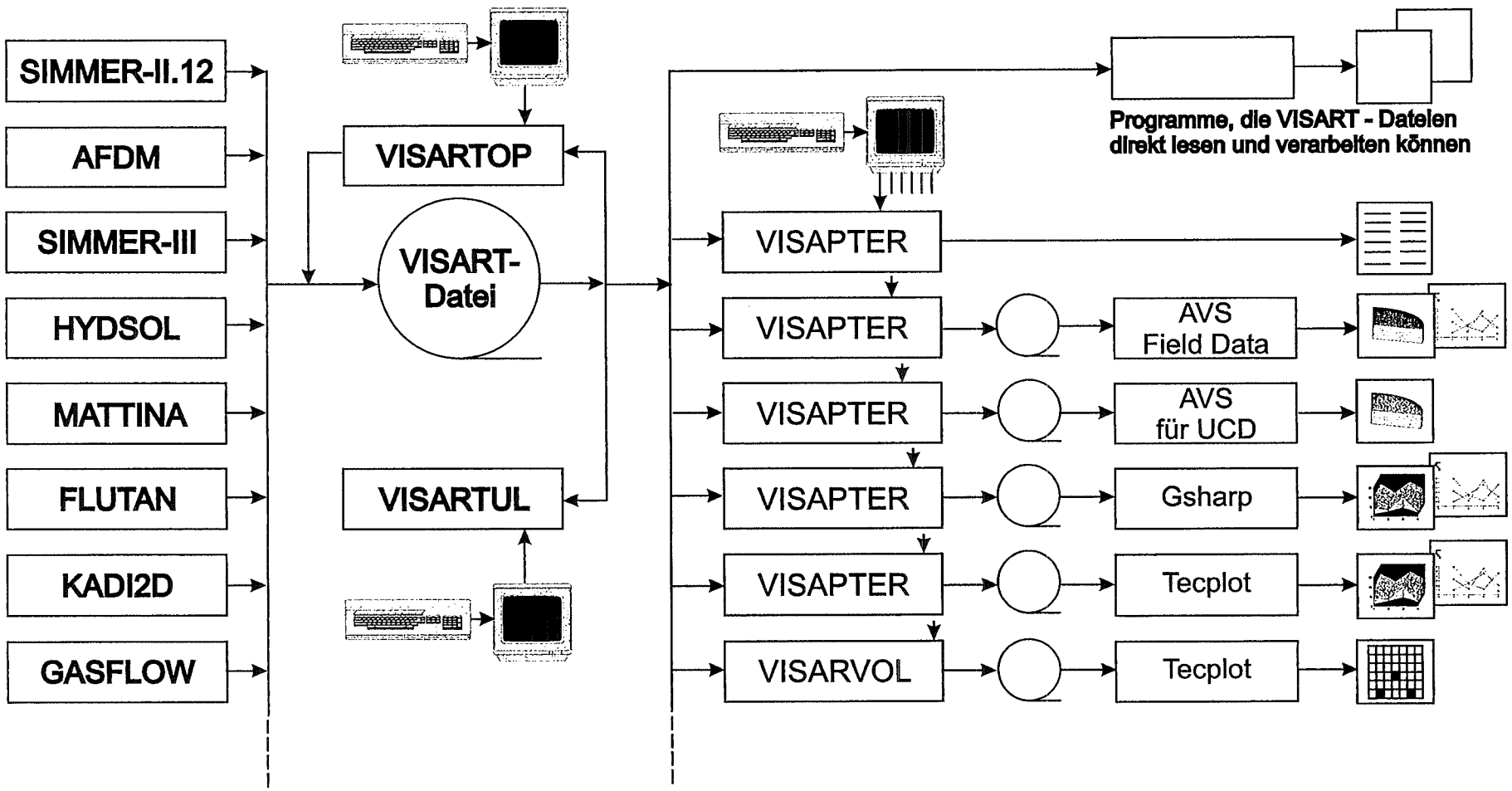

Bild 1 : Datenfluß über die VISART - Schnittstelle

2. Fähigkeiten des Programms

In diesem Kapitel werden die Optionen des Programms VISARTOP erläutert. Sie können jeweils einzeln, aber auch in beliebigen Kombinationen zusammen, in einem Aufruf des Programms realisiert werden. In allen Fällen wird aus einer vorhandenen VISART-Datei eine neue, bearbeitete VISART-Datei erzeugt.

2.1 Änderung der Datei-Formatierung

In der Regel werden VISART-Dateien in unformatierter Darstellung [1b] erstellt. Für die Übertragung zwischen Computern verschiedener Wortlängen und/oder Zahlendarstellungen, ggf. zum Korrigieren von Fehlern, usw. benötigt man jedoch manchmal die Datei in formatierter Darstellung [1b]. VISARTOP kann eine VISART-Datei in unformatierter Darstellung in eine VISART-Datei in formatierter Darstellung umwandeln, und umgekehrt.

2.2 Vereinigung von Fortsetzungs-Files

Bei längeren Rechnungen fallen in jedem Restart-Lauf eines dafür eingerichteten Codes VISART-Dateien als Postprocessor-Ausgabe an, die Fortsetzungen der VISART-Datei des Erstlaufs bzw. des vorherigen Restart-Laufs sind (und deshalb als Fortsetzungs-Files bezeichnet werden). Aus Gründen der Übersichtlichkeit und Verwaltung möchte man evtl. alle zusammengehörigen Files in *einer* VISART-Datei vereinigen. VISARTOP kann solche Verkettungen durchführen.

2.3 Auswahl von Problemzyklen und Gruppen

Zum Durchschauen, Visualisieren, Archivieren, Weitergeben, usw. der Daten genügt manchmal eine Teilmenge der von den Läufen eines Codes auf die VISART-Datei geschriebenen Problemzyklen und/oder Gruppen. VISARTOP kann VISART-Dateien mit reduzierter Datenmenge erstellen. Die zu übernehmenden Problemzyklen werden durch Angabe der Identifikationen der Zyklusgruppen oder durch Angabe der Problemzeit- oder Zyklusnummerintervalle spezifiziert. Die zu übernehmenden Gruppen werden durch Angabe ihrer Identifikationen spezifiziert; alternativ können auch die nicht zu übernehmenden Gruppen durch Angabe ihrer Identifikationen spezifiziert werden.

2.4 Bildung neuer und modifizierter Gruppen

Der Wunsch nach der Bildung neuer und modifizierter Gruppen wurde bereits in der Einleitung begründet. Die Bildungsvorschriften werden durch die Eingabe von Anweisungen spezifiziert, in denen ein Gleichheitszeichen auftritt. Links vom Gleichheitszeichen steht die Identifikation einer neu zu bildenden Größe, rechts davon ein Ausdruck mit Identifikationen bereits vorhandener Größen und mit Operatoren, die die Größen verändern oder verknüpfen. Im folgenden werden die Bestandteile der Anweisungen definiert und der Aufbau der Ausdrücke beschrieben.

2.4.1 Gruppen und Singles, Variablen und Konstanten

Gruppen sind im VISART-Standard [1b] definierte geordnete Datenmengen, bestehend aus Kennsatz, ggf. Spezifikationssatz und Datensätzen, die jeweils einer (skalaren oder vektoriellen) physikalischen Größe (im Falle von Gruppen 9/19 auch mehreren Größen) zugeordnet sind. Solche Gruppen stehen in der Eingabedatei von VISARTOP und werden, abhängig von der Steuereingabe, in die Ausgabedatei übernommen und/oder in VISARTOP zu neuen Gruppen (entsprechend neuen Größen) kombiniert und auf die Ausgabedatei geschrieben.

Alle als Operanden in den Ausdrücken der Steuereingabe auftretenden Gruppen mit Identifikationen aus der VISART-Datei werden im Programm eingelesen und gespeichert. Eingelesen und gespeichert wird bei regulären Netzen auch die Geometrie-Gruppe `GEOMETRY` (siehe 4.1), die die Netzkoordinaten enthält.

In VISARTOP neu erzeugte Gruppen können permanent oder temporär sein. Sie erhalten die in den Anweisungen der Steuereingabe links vom Gleichheitszeichen stehenden Identifikationen. **Permanente Gruppen** werden in die Ausgabedatei geschrieben. Ihre Identifikationen beginnen mit einem Buchstaben. **Temporäre Gruppen** dagegen werden nur im Programm gespeichert, aber nicht ausgegeben. Ihre Identifikationen beginnen mit einem $-Zeichen. [1]

Singles sind im Gegensatz zu Gruppen (skalare oder vektorielle) Einzeldaten, die als Konstante explizit in den Ausdrücken der Steuereingabe stehen, oder als Gruppenelemente (s. 2.4.2) aus Gruppen entnommen sind, oder aus Konstanten und/oder Gruppenelementen oder durch Funktionen erzeugt werden. Singles sind immer temporär, d.h. sie werden im Programm als Konstante oder unter einer Identifikation, beginnend mit einem $-Zeichen, als neue Größe gespeichert, und sie können mit anderen Singles und Gruppen verknüpft werden, aber sie können nicht ausgegeben werden. Vom Programm werden automatisch zur Verfügung gestellt der Single `$IJKBAR`, der (als Vektor) die Anzahl der Maschen in den Koordinatenrichtungen enthält, sowie die Singles `$TIME` und `$CYCLE`, die die Problemzeit bzw. Zyklusnummer des jeweiligen Problemzyklus enthalten.

Außer den **Konstanten**, die explizit in den Ausdrücken als (skalare) Daten (numerische, logische oder Zeichen) stehen, heißen alle oben erwähnten Größen, einschließlich der eingelesenen, **Variablen**. Im allgemeinen ändern sich ihre Werte mit den Problemzyklen. Auch die Größen `GEOMETRY` und `$IJKBAR` werden formal als Variablen behandelt, obwohl ihre Werte über den Problemzyklen konstant bleiben.

In Tabelle 1 werden die in diesem Abschnitt eingeführten Begriffe an Beispielen veranschaulicht.

Die Identifikationen von permanenten und temporäre Gruppen und von Singles, ggf. qualifiziert durch Versionsangabe, Komponente, Index und Ortsangabe (siehe 2.4.2), können demnach sowohl auf der linken Seite der Anweisungen als auch in den Ausdrücken auf der rechten Seite der Anweisungen stehen. Temporäre Gruppen und Singles müssen vor der Verwendung in Ausdrücken durch eine Anweisung definiert worden sein; Konstanten können nur in Ausdrücken stehen.

[1] Die Identifikation `$DUMMY` hat eine Sonderfunktion; siehe 3.2.1.

2.4.2 Qualifikatoren

Unter Qualifikatoren verstehen wir an die Identifikationen von Variablen angefügte Zeichenfolgen zur Kennzeichnung der Version von Variablen, der Vektorkomponenten von Variablen mit vektoriellen Größen und der Verschiebung oder Auswahl der Werte von Gruppen, sowie im Falle von Gruppen 21 zur Ortsangabe der zugeordneten Größe.

Alle in den Ausdrücken auf der rechten Seite der Anweisungen vorkommenden Variablen und die temporären Variablen auf der linken Seite werden in zwei **Versionen** im Programm gespeichert, nämlich für den jeweils aktuellen Problemzyklus und für den vorherigen (das bezieht sich jeweils auf die gemäß Abschnitt 2.3 übernommenen Problemzyklen). Erstere werden durch die Identifikation allein bezeichnet; letztere werden durch ein an die Identifikation angefügtes Gänsefüßchen[1] " gekennzeichnet.

Für alle in den Ausdrücken auf der rechten Seite der Anweisungen vorkommenden Variablen mit vektoriellen Größen und für alle solchen temporären auf der linken Seite können auch **Vektorkomponenten** spezifiziert werden. Dies geschieht durch ein an die Identifikation angefügtes At-Zeichen („Klammeraffe") @ mit nachfolgender Komponentennummer. Die Spezifizierung von Vektorkomponenten auf der rechten Seite der Anweisungen erlaubt die Zerlegung von Variablen mit vektoriellen Größen in (skalare) Komponenten; die Spezifizierung auf der linken Seite erlaubt den Aufbau derselben aus den Komponenten. Wenn keine Vektorkomponente spezifiziert wird, wird die Variable als Vektor, mit allen Komponenten, bzw. als Skalar angesehen.[2]

Alle in den Ausdrücken auf der rechten Seite der Anweisungen vorkommenden Gruppen[3] und alle temporären Gruppen auf der linken Seite können mit Indexausdrücken in geschweiften Klammern versehen werden. Indexausdrücke bezeichnen Teilmengen der Werte einer Gruppe auf zwei unterschiedliche Arten. Zum einen können sie bei den Gruppen 5 und 15 eine **Verschiebung** in der Maschenzuordnung der gespeicherten Werte für eine oder mehrere Koordinatenrichtungen des Netzes bewirken (benötigt für Operationen über versetzte Maschen hinweg), oder sie können bei den anderen Gruppen eine Verschiebung in der Positionszuordnung der gespeicherten Werte schlechthin bewirken (z.B. für die Bildung von Differenzenquotienten über der Zeit bei den Gruppen 20 und 21); wir sprechen dann von einem *Verschiebeindex*. Zum anderen können sie bei den Gruppen 5 und 15 eine **Auswahl** unter den Maschen treffen, um ein Subnetz des gegebenen Netzes — das im Extremfall auch aus nur *einer* Masche, einem Gruppenelement, bestehen kann — durch Konstanthalten einer oder mehrerer Koordinaten des Systems zu bilden, oder sie können bei den anderen Gruppen ein Gruppenelement auswählen; wir sprechen dann von einem *Auswahlindex*. Wenn kein Indexausdruck spezifiziert wird, werden alle Werte der Gruppe ohne Verschiebung beibehalten. (Näheres über die Verwendung von Indexausdrücken findet man im Abschnitt 3.2.2.)

In den Ausdrücken auf der rechten Seite der Anweisungen vorkommende einzulesende Gruppen 21 und alle solche permanente Gruppen auf der linken Seite können zur **Ortsangabe** der zugeordneten Größe mit einem Maschenindex oder einer Punktkoordinate bzw. einem solchen Paar oder Tripel, durch Kommata getrennt und in geschweiften Klammern eingeschlossen, versehen werden. Die **Ortsangabe** muß identisch mit den im Spezifikationssatz der Gruppe 21 stehenden bzw. einzusetzenden Maschenindizes bzw. Punktkoordinaten sein.

Tabelle 2 enthält Beispiele für die Verwendung von Qualifikatoren.

---

[1] unter MVS: einen an die Identifikation angefügten Backslash \.

[2] Die Variable `GEOMETRY`, Gruppe 4, kann nur mit Spezifizierung einer Komponente auftreten. Sie wird dann wie eine skalare Gruppe 5, die für jede Masche deren Koordinatenwerte in der spezifizierten Koordinatenrichtung enthält, aufgefaßt.

[3] außer der Gruppe 4 mit der Variablen `GEOMETRY`.

2.4.3 Operationen und Ausdrücke

Auf der rechten Seite der Anweisungen stehen **Ausdrücke** zur Erzeugung von neuen Variablen aus eingelesenen Gruppen, Konstanten und bereits definierten temporären Variablen. Diese Ausdrücke können geschachtelt sein; ihr Aufbau ergibt sich aus den folgenden rekursiven Definitionen (zur Notation siehe 3.1.1): [1]

Operand := *Konstante* | *Variablenbezeichnung* | (*Ausdruck*) | *Funktion*

Term := [*monadischer Operator*] *Operand*

Ausdruck := *Term* | *Term* ? *Term* : *Term* | *Ausdruck dyadischer Operator Term*

Funktion := *Funktionsname* [([*Ausdruck* [,*Ausdruck* [,*Ausdruck*]]])]

Konstante steht für eine explizite Konstante; *Variablenbezeichnung* steht für die Identifikation einer Variablen mit wahlweiser Anfügung von Qualifikatoren. Zum Beispiel macht die Spezifikation einer Vektorkomponente aus einem Vektor einen Skalar; die Angabe eines Auswahlindexes verringert die Anzahl der Werte in der Gruppe bis hin zu einem Gruppenelement, wodurch aus der Gruppe ein Single wird (siehe auch 3.2.2).

Tabelle 3 listet die möglichen **Operationen** und die zugehörigen Operatoren. Monadische Operatoren stehen vor dem Operanden, dyadische Operatoren stehen zwischen den Operanden. Der Fragezeichen-Operator steht nach der Bedingung, der Doppelpunkt-Operator zwischen den Alternativen. Die Reihenfolge der Operationen muß ggf. durch Klammern vorgegeben werden; andernfalls werden die Operationen von links nach rechts abgearbeitet.

Als Ergebnisse unzulässiger Operationen (Kehrwert von Null, Quadratwurzel aus negativen Zahlen, Logarithmus von nicht-positiven Zahlen, Division durch Null, falls der Zähler nicht ebenfalls Null ist) wird ein quasi-unendlicher Wert (siehe 5.4) zurückgegeben. Es obliegt dem Benutzer, solche Fälle durch Inspektion oder bedingte Zuweisungen zu überwachen.

Die für die jeweiligen Operationen zulässigen Typen der Operanden und die den resultierenden Ausdrücken der Operationen zugewiesenen Typen sind in Tabelle 3 ebenfalls aufgelistet.

[1] Der Ausdruck $Term_1$? $Term_2$: $Term_3$ bedeutet eine bedingte Zuweisung:
wenn $Term_1$ gleich `.true.`, dann Zuweisung von $Term_2$, sonst von $Term_3$.

Jeder aus den Operationen resultierende Teilausdruck, wie auch der gesamte Ausdruck, ist formal eine Gruppe oder ein Single, ein Vektor oder ein Skalar. Dabei gilt das folgende:

- Gruppen, die durch eine bedingte Zuweisung[1] oder durch eine dyadische Operation verknüpft werden, müssen in der Gruppenkennzahl zueinander passen. Referenzgruppen 6, 16 und 20, die als Operanden auftreten, werden wie ihre Subgruppen 7, 17 bzw. 21 behandelt. Singles haben die Gruppenkennzahl 0 und passen zu allen Gruppenkennzahlen. Der resultierende Ausdruck erhält die Gruppenkennzahl des höchstwertigen der beiden Operanden-Terme[2] zugewiesen. Tabelle 4 listet die erlaubten Kombinationen von Gruppenkennzahlen und die resultierenden Gruppenkennzahlen.

- Gruppen, die durch eine bedingte Zuweisung[1] oder durch eine dyadische Operation verknüpft werden, müssen in der Länge (Anzahl der Werte) übereinstimmen. Singles haben die Länge 1 und können untereinander und mit Gruppen jeder Länge verknüpft werden, wobei jeder Wert der Gruppe mit dem Wert des Singles verknüpft wird. Der resultierende Ausdruck erhält demnach die größere der Längen der beiden Operanden-Terme[2] zugewiesen. Tabelle 5 listet die möglichen Kombinationen von Gruppen und Singles und die resultierenden Eigenschaften.

- Gruppen mit vektoriellen Größen, die durch eine bedingte Zuweisung[1] oder durch eine dyadische Operation verknüpft werden, müssen in der Vektordimension (Anzahl der Komponenten) übereinstimmen. Gruppen mit skalaren Größen haben die Vektordimension 0 und können untereinander und mit Vektoren jeder Dimension verknüpft werden, wobei jede Komponente des Vektors mit dem Skalar verknüpft wird. Der resultierende Ausdruck erhält demnach die größere der Vektordimensionen der beiden Operanden-Terme[2] zugewiesen. Tabelle 5 listet die möglichen Kombinationen von Vektoren und Skalaren und die resultierenden Eigenschaften.

Tabelle 6 listet die derzeit implementierten **Funktionen** und Tabelle 7 die Syntax der zugehörigen Argumente und Resultate. Wie bei Operationen, ist der aus einem Funktionsaufruf resultierende Teilausdruck formal eine Gruppe oder ein Single, ein Vektor oder ein Skalar. Anzahl, Kennzahl, Vektordimension und Typ der Argumente und Kennzahl, Vektordimension und Typ des Funktionsresultats sind von Funktion zu Funktion verschieden und der Tabelle 7 zu entnehmen.

---

[1] Die Verknüpfungsregeln bei bedingten Zuweisungen beziehen sich zunächst einmal auf die durch den Operator : verknüpften Operanden-Terme $Term_2$ und $Term_3$ (siehe Fußnote auf der vorigen Seite). Daneben muß aber auch der aus dieser Verknüpfung resultierende Teilausdruck durch den Operator ? mit dem Operanden-Term $Term_1$ verknüpfbar sein, wobei zusätzlich zu den obigen Regeln noch gefordert wird: ist $Term_1$ eine Gruppe, dann muß auch der Teilausdruck eine Gruppe mit der gleichen Kennzahl und mit der gleichen Länge sein; ist $Term_1$ ein Vektor, dann muß auch der Teilausdruck ein Vektor sein.

[2] bei bedingten Zuweisungen: der beiden Operanden-Terme $Term_2$ und $Term_3$ (siehe Fußnote auf der vorigen Seite)

2.4.4 Anweisungen

Anweisungen definieren neu erzeugte permanente Gruppen, die auf die Ausgabedatei geschrieben werden, oder temporäre Variablen, die in den Ausdrücken nachfolgender Anweisungen benötigt werden. Durch eine Anweisung erhält die Variable, die aus dem Ausdruck auf der rechten Seite resultiert, die auf der linken Seite stehende Identifikation. Alternativ kann man auch sagen: die Variable mit der auf der linken Seite stehenden Identifikation erhält den Ausdruck auf der rechten Seite zugewiesen.

Eine Anweisung hat den Aufbau

Name = *Ausdruck*

oder

Temp.-Variablenbezeichnung = *Ausdruck*

Name steht für die Identifikation einer permanenten Gruppe (bei Gruppen 21 mit wahlweiser Anfügung einer Ortsangabe); *Temp.-Variablenbezeichnung* steht für die Identifikation einer temporären Variablen (Single oder temporäre Gruppe) mit wahlweiser Anfügung von Qualifikatoren. Die Identifikation einer permanenten Gruppe kann nur in *einer* Anweisung einem Ausdruck zugeordnet werden. Temporären Variablen können dagegen mehrfach Ausdrücke zugewiesen werden. Dadurch kann z.B. eine vektorielle Größe aus ihren skalaren Komponenten oder eine Gruppe aus ihren Gruppenelementen aufgebaut werden.

Durch die erste Anweisung an eine temporäre Variable *ohne* Qualifikator wird nur die Gruppenkennzahl der Variablen und der Typ der enthaltenen Größe endgültig definiert; bei den Gruppen 5 und 15 auch die Subdimension, die Lokation der Maschenwerte, die Belegung des Netzes und die Reihenfolge der Maschen im Netz. Durch die erste Anweisung an eine temporäre Variable *mit* Qualifikation einer Vektorkomponente wird außerdem zusätzlich die Länge der Variablen endgültig festgelegt, bei Spezifizierung eines Verschiebe- oder Auswahlindexes die Vektordimension. Die jeweils übrigen Charakteristiken werden dagegen nur vorläufig, und die Werte der Variablen nur vorläufig oder teilweise definiert. Durch mehrere Anweisungen an die gleiche temporäre Variable mit komplementären Qualifikatoren läßt sich diese dann vervollständigen. Zum Beispiel wird bei Spezifikation von Vektorkomponenten jeder Komponente ein (skalarer) Ausdruck zugewiesen; bei Angabe eines Auswahlindexes wird einem Teil der Variablen ein Ausdruck zugewiesen, der einer kleineren Gruppe (bis hin zu einem Single) entspricht. (Siehe auch 3.2.2.)

3. Steuerung des Programms

In diesem Kapitel wird die Steuereingabe des Programms VISARTOP beschrieben.

3.1 Steuereingabe

Das Programm ist für den Stapelbetrieb entworfen; es erwartet eine komplette Steuereingabe auf einer Datei und läuft vom Start bis zum Programmende oder Fehlerabbruch ohne Wechselwirkung mit dem Benutzer.

Hier wird der Aufbau der Steuereingabe beschrieben, wie sie vom Benutzer zu erstellen ist bzw. wie sie von der grafischen Benutzeroberfläche (siehe 3.4) aus der Einstellung der „Widgets“ erzeugt wird.

Alle Eingabe wird von VISARTOP als `CHARACTER`-Kette gelesen, syntaktisch analysiert und als Wertzuweisungen an Variable oder als Einträge in Tabellen interpretiert. Die Angaben und Anweisungen können sich beliebig über die Zeilen (von Spalte 1 bis 72) und auch über mehrere Zeilen erstrecken; Leerzeichen und Zeilenende sind bedeutungslos (außer in Namen und Konstanten). Kommentar wird zwischen den Zeichen `/*` und `*/` eingeschlossen; er kann überall dort stehen, wo bedeutungslose Leerzeichen stehen dürfen.[1]

Die Steuereingabe besteht aus drei Teilen, wobei Teil 2 und 3 sowohl für das Kopf-Paket als auch für die Rumpf-Pakete der Datei benötigt werden:

Teil 1: Angabe der Formatierung und Gliederung der Ausgabedatei und Auswahl der zu übernehmenden Problemzyklen;
Teil 2: Auswahl der zu übernehmenden Gruppen für das Kopf-Paket;
Teil 3: Anweisungen zur Erzeugung neuer Gruppen für das Kopf-Paket;
Teil 2: Auswahl der zu übernehmenden Gruppen für die Rumpf-Pakete;
Teil 3: Anweisungen zur Erzeugung neuer Gruppen für die Rumpf-Pakete.

In allen Teilen der Eingabe werden die einzelnen Angaben und Anweisungen durch einen Strichpunkt abgeschlossen, die Teile selbst durch einen Punkt.

Das Programm VISARTOP erwartet die Steuereingabe als Standardeingabe. Der Abschluß des Teils 3 für die Rumpf-Pakete wird als Ende der Eingabe erkannt.

[1] unter MVS dürfen die Zeichen `/*` nicht am Zeilenanfang des Eingabestroms stehen!

3.1.1 Syntaktische Definitionen

In der folgenden syntaktischen Beschreibung der Eingabeteile schließen eckige Klammern Angaben und Anweisungen ein, die nicht immer vorkommen müssen. Falls nichts anderes gesagt wird, sind Reihenfolge und Wiederholung von Zeilen in eckigen Klammern beliebig. Ein senkrechter Strich bedeutet ein exklusives Oder.

Eine *Integerkonstante* ist eine nicht-negative ganze Zahl, eine *Realkonstante* eine nicht-negative Zahl, die einen Dezimalpunkt enthalten und mit einer Ziffer beginnen muß (Exponenten sind bisher nicht erlaubt). Eine *logische Konstante* kann `.TRUE.` oder `.FALSE.` sein. Eine *Zeichenkonstante* ist jede in Hochkommata eingeschlossene Zeichenkette. Von Konstanten werden nur die ersten 16 Stellen eingelesen; gespeichert werden sie in Einfachworten (alle Integer- und logische Konstanten, sowie Realkonstanten bei DBLPP-Option „off") oder in Doppelworten (alle Zeichenkonstanten, sowie Realkonstanten bei DBLPP-Option „on").

Ein *Name* ist eine Zeichenkette aus Buchstaben, Ziffern und Unterstrichen, die mit einem Buchstaben beginnen muß. Die Unterstriche werden nach dem Einlesen durch Leerzeichen ersetzt. Sollen Unterstriche als Bestandteile von *Name* bestehen bleiben, so sind sie mit einem Backslash \ zu „quoten". (Wegen ?, * und + in Namen siehe 3.1.3.) *$name* beginnt mit $ anstelle eines Buchstabens. Von Namen werden nur die ersten 8 Stellen eingelesen und gespeichert.

Eine *Konstante* ist dann definiert durch

Konstante := *numerische Konstante* | *logische Konstante* | *Zeichenkonstante*

numerische Konstante := *Integerkonstante* | *Realkonstante*

Eine *Variablenbezeichnung* ist definiert durch

Variablenbezeichnung := *Name* ["] [@*s*] [{*Indexausdruck*}] |
Name {*Ortsangabe*} |
Temp.-Variablenbezeichnung

eine *Temp.-Variablenbezeichnung* durch

Temp.-Variablenbezeichnung := *$name* ["] [@*s*] [{*Indexausdruck*}]

", @*s*, und {*Indexausdruck*} sind Qualifikatoren für die Version, die Vektorkomponente bzw. den Verschiebe- oder Auswahlindex. Hierbei ist *s*= 1|2|3; ein *Indexausdruck* ist definiert durch

Indexausdruck := *Verschiebeindexausdruck* | *Auswahlindexausdruck*

Verschiebeindexausdruck := [*Kette*] [,[*Kette*] [,[*Kette*]]]

Auswahlindexausdruck := *Integerkonstante* [,[*Integerkonstante*] [,[*Integerkonstante*]]] |
& *Integerkonstante* [,[*Int.konstante*] [,[*Int.konstante*]]] |
& *Name* [,[*Integerkonstante*]]

wobei *Kette* eine Zeichenkette aus den Zeichen < und > bedeutet und das Und-Zeichen & eine indirekte Auswahl kennzeichnet (siehe 3.2.2).

Die Zeichenketten im Verschiebeindexausdruck können jeweils nur 4 Stellen lang sein. Die Integerkonstanten im Auswahlindexausdruck können, außer der ersten, die 8 Stellen lang sein kann, jeweils nur 4 Stellen lang sein; ist die erste Integerkonstante mehr als 4 Stellen lang, so kann der Indexausdruck nicht mehr als die ersten beiden Integerkonstanten enthalten. Der Name im (indirekten) Indexausdruck kann 8 Stellen lang sein, die ggf. darauffolgende Integerkonstante 4 Stellen lang.

Eine *Ortsangabe* ist definiert durch

Ortsangabe := *Integerkonstante* [,[*Integerkonstante*] [,[*Integerkonstante*]]] |
Realkonstante [,[*Realkonstante*] [,[*Realkonstante*]]]

Ein *Operator* ist definiert durch

Operator := *monadischer Operator* | *dyadischer Operator* | ? | :

wobei *monadische Operatoren* und *dyadische Operatoren* der linken Spalte der Tabelle 3 zu entnehmen sind.

Ein *Funktionsname* ist von der Form #*nn*, wo *nn* für eine Ziffernpaar, gemäß der Tabelle 6, steht.

Beispiele:

Konstante:	`7 0.5 1000000. .TRUE. 'QUATSCH'`		
Variablenbezeichnung:	`TEMPERAT`	`$TEMPER`	
	`TEMPERAT"`	`$TEMPER"`	(Version)
	`IMPULS@2`	`$IMPULS@2`	(Vektorkomp.)
	`TEMPERAT{,<}`	`$TEMPER{,<}`	(Verschiebung)
	`IMPULS{5,7}`	`$IMPULS{5,7}`	(Auswahl)
	`ANFANG{&MASSE,1}`	`$ANFANG{&MASSE,1}`	(indir. Auswahl)
	`ORDNUNG{&3,5,4}`	`$ORDNUNG{&3,5,4}`	(indir. Auswahl)
	`IMPULS_T{5,7}`		(Ortsangabe)
	`TEMPER_T{10.7,12.8}`		(Ortsangabe)
Nicht erlaubt:	`.5 1.E6 IMPULS{,7}`		

3.1.2 Teil 1 der Eingabe

Teil 1 der Eingabe dient zur Angabe der Formatierung und Gliederung der Ausgabedatei und zur Auswahl der in die Ausgabedatei zu übernehmenden Problemzyklen.

Syntax des Teils 1:

e; *f*; *g*;

[*Realkonstante*$_1$; *Realkonstante*$_2$; [*Realkonstante*$_1$; *Realkonstante*$_2$; [...]]]
[*Integerkonstante*$_1$; [*Integerkonstante*$_2$; [*Integerkonstante*$_1$; *Integerkonstante*$_2$; [...]]]]
[Name; [Name; [...]]]

.

Bedeutung der Angaben:

e	Anzahl der Fortsetzungs-Files (einschl. erster File) der Eingabedatei.
f	Formatierung der Ausgabedatei: 0, wenn unformatiert; 1, wenn formatiert; leer, wenn Übernahme der Formatierung der Eingabedatei in die Ausgabedatei.
g	Gliederung der Ausgabedatei: 1, wenn Zusammenfassung der Fortsetzungs-Files der Eingabedatei zu *einer* Ausgabedatei. leer, wenn Übernahme der Fortsetzungs-Files der Eingabedatei in die Ausgabedatei.
Realkonstante	Problemzeitpunkt; ein Konstantenpaar schließt ein zu übernehmendes Problemzeitintervall ein.
Integerkonstante	Zyklusnummer; ein Konstantenpaar schließt ein zu übernehmendes Zyklusintervall ein; ist nur eine Konstante angegeben, so bedeutet dies, daß jeder durch die Zyklusnummer teilbare Zyklus zu übernehmen ist.
Name	Identifikation der zu übernehmenden Problemzyklen.

Anmerkungen:

Die Menge der zu übernehmenden Problemzyklen bestimmt sich aus dem Durchschnitt der vereinigten Problemzeitintervalle, der vereinigten Zyklusintervalle und der vereinigten Identifikationen. Wenn alle Angaben fehlen, werden alle Problemzyklen übernommen.

3.1.3 Teil 2 der Eingabe

Teil 2 der Eingabe dient zur Auswahl der in die Ausgabedatei zu übernehmenden Gruppen. Er wird sowohl für das Kopf-Paket als auch für das Rumpf-Paket der Datei benötigt.

Syntax des Teils 2:

p;

[*Name*; [*Name*; [*Name*; [...]]]]

[$Name_0$; [_*Name*; | _*Name*{*Ortsangabe*}; [_*Name*; | _*Name*{*Ortsangabe*}; [...]]]] [1]
[-$Name_0$; [_*Name*; | _*Name*{*Ortsangabe*}; [_*Name*; | _*Name*{*Ortsangabe*}; [...]]]] [1 2]

.

Bedeutung der Angaben:

p	Art der Auswahl der zu übernehmenden Gruppen: 0, wenn die nicht zu übernehmenden Gruppen spezifiziert werden; 1, wenn die zu übernehmenden Gruppen spezifiziert werden.
Name	Identifikation einer zu übernehmenden bzw. nicht zu übernehmenden Gruppe.
$Name_0$	Identifikation einer zu übernehmenden bzw. nicht zu übernehmenden Referenzgruppe 6, 16 oder 20 (im Falle *p*=0 mit Minuszeichen versehen, falls die Gruppe nur zum Bezug aufgeführt ist, aber nicht gelöscht werden soll).

Den Identifikationen der Subgruppen 7, 17 und 21 wird ein Unterstrich vorangestellt, der als Bezug auf die vorangegangene Referenzgruppe 6, 16 bzw. 20 mit der Identifikation $Name_0$ dient.

Anmerkungen:

Die Menge der zu übernehmenden Gruppen bestimmt sich demnach nach folgender Tabelle:

p	keine Identifikationen angegeben	Identifikationen angegeben
0	alle Gruppen werden übernommen	alle Gruppen außer den angegebenen werden übernommen
1	keine Gruppen werden übernommen	nur die angegebenen Gruppen werden übernommen

Die Referenzgruppen 6, 16 und 20 können nur dann entfernt werden, wenn auch alle zugehörigen Subgruppen 7, 17 bzw. 21 entfernt werden.

In *Name*, aber nicht in $Name_0$, dürfen auch die Platzhalterzeichen ?, * und + vorkommen. ? steht für ein beliebiges, nicht leeres, Zeichen. * und + stehen für eine Kette aus keinem, einem oder mehreren beliebigen, nicht leeren, Zeichen. Die durch * bezeichneten Ketten können nachfolgende, explizit eingegebene, Zeichen und Leerstellen innerhalb des Doppelwortes nach rechts schieben. Die durch + bezeichneten Ketten können nur die nachfolgenden Leerstellen überschreiben; nachfolgende Zeichen bleiben fest an ihrer eingegebenen Stelle im Doppelwort. Mit einem Namen, der Platzhalterzeichen enthält, können also mehrere Identifikationen abgedeckt werden. Beispiel: `A*Z` deckt alle Namen ab, die mit `A` beginnen, mit `Z` enden und dazwischen keine Leerzeichen enthalten; `A+____Z` deckt alle Namen ab, die mit A beginnen, mit `Z` in der achten Stelle enden und davor auch Leerzeichen enthalten können.

[1] _*Name*{*Ortsangabe*} nur für Gruppen 21
[2] nur für *p*=0

3.1.4 Teil 3 der Eingabe

Teil 3 der Eingabe enthält Anweisungen zur Erzeugung neuer Gruppen. Er wird sowohl für das Kopf-Paket als auch für das Rumpf-Paket der Datei benötigt. [1]

Syntax des Teils 3:

[*Name* = *Ausdruck* ;]
[*Temp.-Variablenbezeichnung* = *Ausdruck* ;]

[$Name_0$] ;
[_*Name* = *Ausdruck_* ;]
[_*Name* {*Ortsangabe*} = *Ausdruck_* ;] [1]
[_*Temp.-Variablenbezeichnung* = *Ausdruck_* ;]

.

Bedeutung der Angaben:

Name	Identifikation einer durch den Ausdruck auf der rechten Seite definierten permanenten Gruppe.
Temp.-Variablenbezeichnung	Identifikation einer temporären Variablen, ggf. mit Qualifikatoren, der der Ausdruck auf der rechten Seite zugewiesen wird.
$Name_0$	Identifikation einer zu übernehmenden Referenzgruppe 6, 16 oder 20 (leer, wenn eine solche Gruppe nicht übernommen werden soll); der Strichpunkt allein markiert den Beginn eines Abschnittes.
Ausdruck	zusammengesetzt aus *Konstanten*, *Variablenbezeichnungen*, *Funktionsnamen*, *Operatoren* und runden Klammern, gemäß folgender rekursiver Definition (siehe 2.4.3):

Operand := *Konstante* | *Variablenbezeichnung* | (*Ausdruck*) | *Funktion*

Term := [*monadischer Operator*] *Operand*

Ausdruck := *Term* | *Term* ? *Term* : *Term* | *Ausdruck dyadischer Operator Term*

Funktion := *Funktionsname* [([*Ausdruck* [,*Ausdruck* [,*Ausdruck*]]])]

Die Identifikationen auf der linken Seite der Anweisungen können (innerhalb der Namenskonventionen) beliebig gewählt werden. Den Identifikationen der Subgruppen 7, 17 und 21 muß ein Unterstrich vorangestellt werden und sie können nur innerhalb eines markierten Abschnitts (siehe 3.2.1) auftreten. Sie beziehen sich dann auf die Referenzgruppe 6, 16 bzw. 20, die am Beginn des Abschnitts unter $Name_0$ aufgeführt ist. *Ausdruck_* bedeutet einen Ausdruck, in dem an Subgruppen 7, 17 und 21 nur solche des jeweiligen Abschnitts vertreten sein dürfen.

Anmerkungen:

Die Subgruppen 7, 17 und 21 können nur dann erzeugt werden, wenn die zugehörigen Referenzgruppen 6, 16 bzw. 20 übernommen wurden (siehe 3.1.3).

Wenn die Gruppe $Name_0$ im Teil 3 der Eingabe aufgeführt ist, wird sie in die Ausgabedatei übernommen, auch wenn sie im Teil 2 der Eingabe nicht aufgeführt ist.

[1] nur für Gruppen 21.

3.2 Ergänzungen zum Teil 3 der Eingabe

3.2.1 Abschnittsstruktur

Der Teil 3 der Steuereingabe wird parallel zum Einlesen der Pakete der VISART-Eingabedatei abgearbeitet. Das heißt, daß der Teil 3 für das Kopf-Paket parallel zum Inhalt des Kopf-Pakets der Datei interpretiert wird, der Teil 3 für die Rumpf-Pakete parallel zum Inhalt jedes Rumpf-Pakets.

Das hat zunächst einmal zur Folge, daß Variable aus dem Kopf-Paket der Datei, die mit Variablen aus den Rumpf-Paketen verknüpft werden sollen, bereits im Teil 3 für das Kopf-Paket an temporäre Variable zugewiesen werden müssen, da sie beim Abarbeiten des Rumpf-Pakets nicht mehr von der Eingabedatei gelesen werden können.

Sodann muß der Teil 3 der Eingabe sowohl für das Kopf-Paket als auch für die Rumpf-Pakete in Abschnitte gegliedert werden, die zur Anordnung der Gruppen im Kopf-Paket bzw. in den Rumpf-Paketen der Eingabedatei passen. Zu jeder Referenzgruppe 6, 16 oder 20 und den nachfolgenden Gruppen im Kopf-Paket bzw. in den Rumpf-Paketen gehört parallel ein Abschnitt der Eingabe. Der Beginn eines Abschnitts wird durch die Identifikation der betr. Referenzgruppe 6, 16 oder 20 mit nachfolgendem Strichpunkt markiert. Die Zugehörigkeit von Subgruppen 7, 17 bzw. 21 zum Abschnitt muß in den Anweisungen durch Namen und Variablenbezeichnungen mit vorangestelltem Unterstrich hergestellt werden. Bei den anderen Gruppen und Singles in den Anweisungen des Abschnitts ist dies nicht nötig (kann jedoch bei temporären Gruppen und Singles sinnvoll sein, um den Geltungsbereich der Variablen auf den jeweiligen Abschnitt zu begrenzen). Ein Abschnitt wird beendet mit dem Beginn eines neuen Abschnitts oder durch das nächste Rumpf-Paket. Gibt es keine Anweisungen innerhalb eines Abschnitts, so wird er lediglich durch einen Strichpunkt markiert. Wenn überhaupt keine Abschnitte im Paket angesprochen werden, können auch die Strichpunkte entfallen[1]. Beim Abarbeiten eines Abschnitts können nur die bis zum jeweiligen Abschnittsende in der Eingabedatei vorkommenden Gruppen und, im Falle der Subgruppen 7, 17, bzw. 21, dem Abschnitt zugehörige Gruppen eingelesen und verknüpft werden.

Wenn die Auswahl eines Gruppenelements aus den Werten einer Gruppe durch indirekte Auswahl erfolgt (siehe 3.2.2) so gelten folgende Besonderheiten. Die beschreibende Gruppe muß vor der Interpretation der Auswahl eingelesen oder als temporäre Variable erzeugt worden sein. Wird die Gruppe außer zur Beschreibung nicht weiter benötigt, so wird das Einlesen im Kopf- bzw. Rumpf-Paket durch eine Anweisung mit der Identifikation `$DUMMY` auf der linken Seite und der Identifikation der beschreibenden Gruppe auf der rechten Seite veranlaßt; darüber hinaus hat diese Anweisung keine Wirkung. Die Kennzahl der beschreibenden Gruppe muß zu der Kennzahl der Werte enthaltenden Gruppe nach folgender Tabelle passen:

Beschreibende Gruppe	Werte enthaltende Gruppe
6	7
9	9
9	19
19	19
16	17

Um die Zugehörigkeit der beschreibenden Gruppe zu der die Werte enthaltenden Gruppe festzulegen, brauchen beide Identifikationen nicht identisch zu sein, sondern es genügt, wenn die Identifikation der ersteren Gruppe und die der letzteren Gruppe in möglichst vielen Zeichen vom Anfang

[1] Dies gilt jedoch nicht, wenn im Kopf-Paket Referenzgruppen 6 stehen, auf die sich leere Referenzgruppen 16 in den Rumpf-Paketen beziehen. In diesem Fall müssen die Referenzgruppen 6 im Eingabeteil 3 für das Kopf-Paket abschnittsgemäß mit Identifikation und Strichpunkt aufgeführt werden.

der Identifikationen her übereinstimmen (siehe Beispiele 6 und 7). (Falls mehrere gleichweit übereinstimmende Identifikationen für die beschreibende Gruppe in Frage kommen, wird die zuerst eingelesene oder erzeugte dafür genommen.)

3.2.2 Details der Qualifikatoren

Indexausdrücke in geschweiften Klammern als Qualifikatoren von Variablen für den Verschiebe- oder Auswahlindex wurden in 2.4.2 eingeführt. Im folgenden wird ihre Verwendung näher beschrieben.

Ein Verschiebeindex besteht aus bis zu drei Zeichenketten, jeweils leer oder aus den Zeichen > oder < zusammengesetzt, die durch Kommata getrennt und in geschweiften Klammern eingeschlossen sind. Die Kette > bedeutet, daß die Zählung der Maschen in der betr. Koordinatenrichtung bzw. die Zählung der Positionen in der Gruppe um 1 versetzt nach rechts bzw. oben bzw. hinten beginnt; die Kette >> bedeutet 2 Versetzungen; usw.. Die Kette < bedeutet, daß die Zählung der Maschen in der betr. Koordinatenrichtung bzw. die Zählung der Positionen in der Gruppe um 1 versetzt von rechts bzw. oben bzw. hinten aufhört; die Kette << bedeutet 2 Versetzungen; usw.. In beiden Fällen wird durch die Qualifikation mit dem Verschiebeindex also eine um 1 Wert bzw. 2 Werte, usw., (ggf. in der betr. Koordinatenrichtung) kleinere Gruppe bezeichnet.

Für Verschiebeindexausdrücke an Variablen in Ausdrücken gilt:

- Bei der Verschiebung der Maschen- oder Positionszuordnung der Gruppe gehen Werte aus der Gruppe verloren; die Anzahl der Werte in der resultierenden Gruppe ist entsprechend kleiner.[1] Im Falle der Gruppen 5 und 15 wird dann die Verschlüsselung der Lokation im Spezifikationssatz um den Code für die Belegungsverschiebung des Netzes ergänzt (siehe [1b], Zusatztabelle 4.3.6).[2]

Für Verschiebeindexausdrücke an temporären Variablen auf der linken Seite von Anweisungen gilt:

- Die Angabe einer Verschiebung ist nur in der ersten Anweisung an eine Variable möglich. Durch die Verschiebung der Maschen- oder Positionszuordnung der Gruppe werden außer den durch den zugewiesenen Ausdruck belegten Stellen weitere Stellen in der Gruppe reserviert, deren Werte aber nicht definiert; die Anzahl der Werte in der Gruppe wird um die reservierten erhöht.[1] Im Falle der Gruppen 5 und 15 wird die Verschlüsselung der Lokation im Spezifikationssatz um den Code für die Belegungsverschiebung ergänzt.[2] Hier ist eine Verschiebung nur so weit möglich, als die möglichen Lokationen auf den Maschen des Netzes nicht überschritten werden; bei einer Verschiebung einer vollen Belegung des Netzes mit Werten in den Maschenmittelpunkten (Lokationsverschlüsselung 00)[3] wird ausnahmsweise die Lokation, statt die Belegung, um eine halbe Masche verschoben, sodaß die Werte auf den Maschenhüllen zu liegen kommen.

Ein Auswahlindex besteht aus bis zu drei Integerkonstanten (sog. direkte Auswahl), oder aus einem Und-Zeichen & gefolgt von bis zu drei Integerkonstanten oder einem Namen und einer Integerkonstanten (sog. indirekte Auswahl), die durch Kommata getrennt und in geschweiften Klammern eingeschlossen sind. Bei den Gruppen 5 und 15 ist nur direkte Auswahl möglich. Hier bezeichnen die von Null verschiedenen Integerkonstanten die für die jeweilige Koordinatenrichtung des Netzes festgehaltenen Maschenindizes, wodurch Gruppen über Subnetzen bis hin zu Gruppenelementen definiert werden. Bei den anderen Gruppen gibt bei direkter Auswahl die Integerkonstante (nur *eine* ist erlaubt) die Position des auszuwählenden Wertes in der Gruppe an.[4] Bei indirekter

[1] Ausnahme: Bei den Subgruppen 7, 17 und 21 wird formal die Anzahl der Werte beibehalten.

[2] Diese erweiterte Lokationsverschlüsselung ist jedoch nur innerhalb von VISARTOP definiert und muß vor der Ausgabe der permanenten Gruppen durch geeignete Kombinationen der temporären Gruppen wieder beseitigt werden.

[3] auch für die Lokationen 01 mit Verschiebung > und 10 mit Verschiebung <

[4] Die Integerkonstante 999999 deutet auf den letzten Maschenindex in der betreffenden Koordinatenrichtung bzw. auf die letzte Position in der Gruppe.

Auswahl ist die Position des auszuwählenden Wertes in der Gruppe gleich der Position der im Indexausdruck angegebenen Konstanten in einer bereits definierten Gruppe passender Kennzahl, die zur Beschreibung der Werte enthaltenden Gruppe dient. Z.B. wird bei indirekter Auswahl mit einem Namen (und ggf. laufender Nummer) die Position des Gruppenelements aus der Position des Namens (und ggf. der laufenden Nummer bei Mehrfachauftreten des Namens) in der Beschreibungsgruppe bestimmt (siehe auch 3.2.1).

Für Auswahlindexausdrücke an Variablen in Ausdrücken gilt:

- Die Auswahl aus der Gruppe definiert ein Gruppenelement, oder, bei den Gruppen 5 und 15, auch Gruppen über Subnetzen. Die Anzahl der Werte wird auf 1 gesetzt bzw. bei den Subnetzen entsprechend verringert; dort werden auch die das Subnetz definierenden Indizes in den Spezifikationssatz übernommen.

Für Auswahlindexausdrücke an temporären Variablen auf der linken Seite von Anweisungen gilt:

- Die Angabe einer Auswahl ist nicht in der ersten Anweisung an eine Variable erlaubt; sie setzt voraus, daß die Gruppe einschließlich der Gruppenlänge bereits definiert ist. Dagegen ist die Angabe einer Auswahl in beliebig vielen weiteren Anweisungen an die Variable möglich. Der Auswahlindex bezeichnet die Position(en) in der Gruppe, deren Werte durch den Wert bzw. die Werte des zugewiesenen Ausdrucks überschrieben werden sollen.

Vektorkomponentennummern mit vorangestelltem At-Zeichen als Qualifikatoren von Variablen für die Vektorkomponente wurden ebenfalls in 2.4.2 eingeführt. Für Vektorkomponentennummern an temporären Variablen auf der linken Seite von Anweisungen gilt:

- Die Angabe von Vektorkomponenten ist in der ersten und in weiteren Anweisungen an eine Variable möglich. Die Komponentennummer bezeichnet die Komponente der Variablen, die den (skalaren) Ausdruck zugewiesen bekommt. Die Vektordimension der Variablen wird bestimmt durch das Maximum der angegebenen Komponentennummern.

3.3 Eingabebeispiele

Die folgenden Bespiele enthalten jeweils die komplette Eingabe für einen VISARTOP-Lauf.

Eingabe zur Umwandlung eines unformatierten VISART-Files in einen formatierten VISART-File:

```
1; 1; ;.
0;.
.
0;.
.
```

Eingabe zur Zusammenfassung von 5 VISART-Files in 1 VISART-File:

```
5; ; 1;.
0;.
.
0;.
.
```

Eingabe für eine Übernahme der Formatierung und Gliederung der Eingabedatei und für die Übernahme alle Problemzyklen zwischen den Problemzeiten 5 ms und 10 ms:

```
1; ; ; 0.005; 0.01;.
0;.
.
0;.
.
```

Eingabe zur Erzeugung eines VISART-Files, der nur die angegebenen Gruppen enthält (TIMEALL hat die Kennzahl 20):

```
1; ; ;.
0;.
.
1; RLBR1; RLBR2; TIMEALL; _SEL____T; _S2PHAS_T; .
.
```

Eingabe zur Erzeugung eines VISART-Files, der alle Gruppen außer den angegebenen enthält (auch TIMEALL mit Kennzahl 20 bleibt enthalten):

```
1; ; ;.
0;.
.
0; ALPHG; ALPHL; -TIMEALL; _P_____T; .
.
```

Eingabe zur Erzeugung eines VISART-Files, der zusätzlich zu den vorhandenen Gruppen die angegebenen neuen Gruppen erzeugt (TIMEALL hat die Kennzahl 20):

```
1; ; ;.
0;.
.
0;.
RLBR = RLBR1 + RLBR2 ;
TIMEALL ;
_PBAR___T = _P_____T / 100000.   ;
.
```

Weitere Beispiele für den Teil 3 der Eingabe sind in Kapitel 6 zu finden.

3.4 Eingabe über die Benutzeroberfläche

Wenn das Programm VISARTOP über die grafische Benutzeroberfläche [1a] aufgerufen wird, wird die Steuereingabe durch das Tcl/Tk-Skript der Benutzeroberfläche aus den Einstellungen der „Widgets" im VISARTOP-Fenster und in den zugehörigen Unterfenstern aufgebaut und nach Betätigung der Taste „Ausführen" auf die Datei `VISART_IN` im jeweiligen Arbeitsverzeichnis geschrieben. Diese Datei wird dem Programm beim Aufruf als Eingabedatei übergeben.

Die Widgets (das sind hier Tasten, Wähltasten, Eingabekästchen, Anzeigefelder) sind jeweils einem bestimmten Eingabeparameter zugeordnet und setzen dessen Wert nach der Stellung der Taste(n) oder auf den im Kästchen oder Anzeigefeld eingegebenen Wert. Zwei Tasten öffnen nach Betätigung ein Unterfenster für die Einstellung weiterer Gruppen von Parametern. Die Funktion der Widgets im VISARTOP-Fenster ist durch in die VISART-Benutzeroberfläche eingebaute Hilfetexte dokumentiert. In diesen Hilfen werden auch die Zuordnung der Widgets zu den Eingabeparameter und die den Tastenstellungen entsprechenden Werte angegeben.

Bild 2 zeigt das VISARTOP-Fenster. Alle Widgets unterhalb der Anzeigefelder für die Dateien dienen zur Eingabe von Steuerparametern.

Mit der Wähltastenleiste „unformatiert", „formatiert", „übernommen" wird der Parameter f spezifiziert. Mit der Wähltastenleiste „zusammengefaßt", „übernommen" wird der Parameter g spezifiziert. (Der Parameter e wird implizit über die Spezifizierung der VISART-Datei(en) im linken oberen Anzeigefeld gesetzt.)

In der Kästchenleiste „Zu uebernehmende Rumpf-Pakete ..." werden die Problemzeitintervalle, und/oder Zyklusintervalle und/oder Zyklusidentifikationen eingetragen.

Es folgen zwei identische Felder von Tasten und Feldern, von denen sich das erste auf das Kopf-Paket, das zweite auf die Rumpf-Pakete bezieht. Mit der Wähltastenleiste „Zu uebernehmende ...", „Nicht zu uebernehmende ..." wird der Parameter p für das Kopf- bzw. die Rumpf-Pakete spezifiziert. In der daruntersteheden Kästchenleiste können die zugehörigen Gruppenidentifikationen eingetragen werden. Zur Eingabe der Anweisungen für die Erzeugung neuer Gruppen (sowie zu deren Ein- oder Abspeicherung von bzw. auf Dateien) kann mit der Taste „ja" ein Unterfenster (Bild 2.a) geöffnet werden, in dem mehr Platz für das die Anweisungen aufnehmende Kästchen zur Verfügung steht, als im VISARTOP-Fenster unter der „ja"-Taste.

Identifikationen werden in die Kästchen der Benutzeroberfläche ohne Apostrophe eingesetzt. In den Identifikationen stehende Leerzeichen werden deshalb als Unterstriche eingegeben. Sind Unterstriche Teil der Identifikation, so sind sie bei der Eingabe durch einen vorgesetzten Backslash zu „quoten".

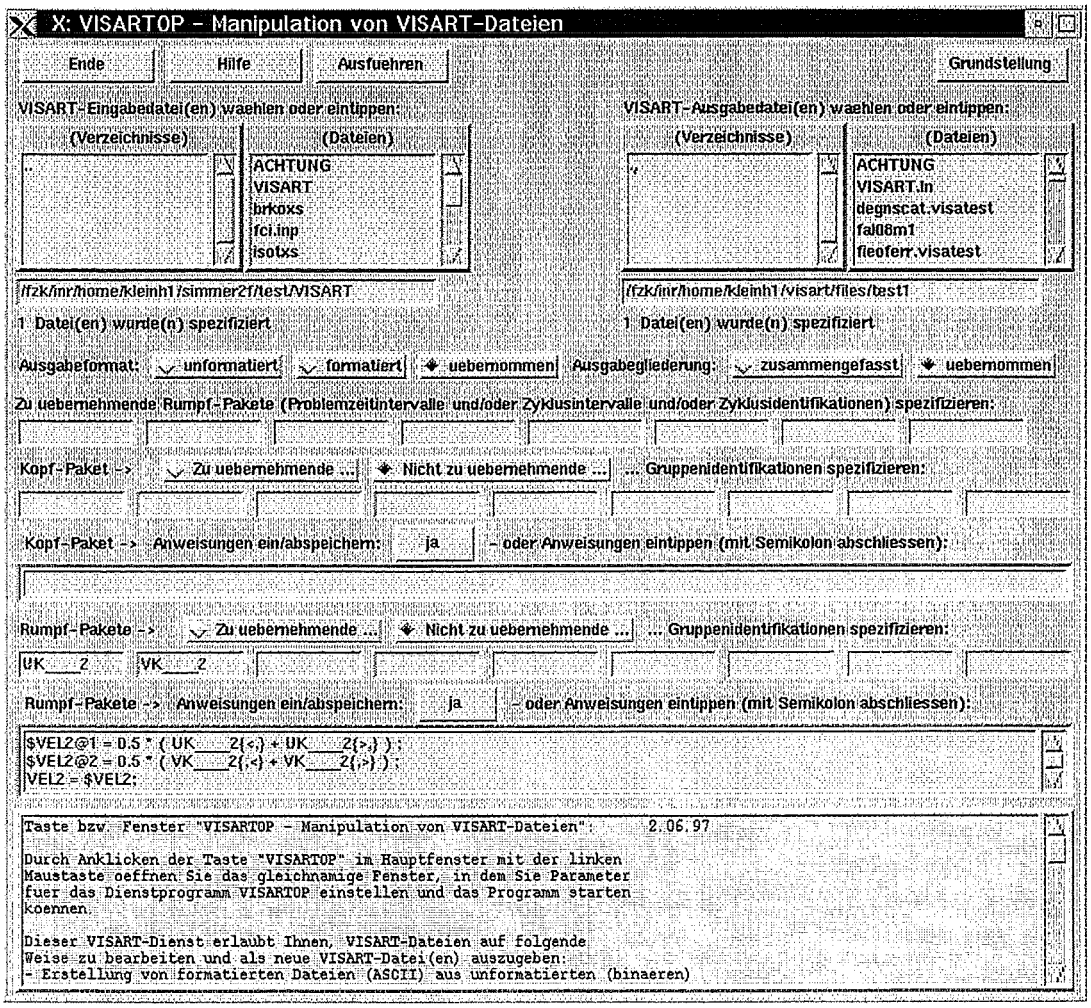

Bild 2: VISARTOP-Fenster

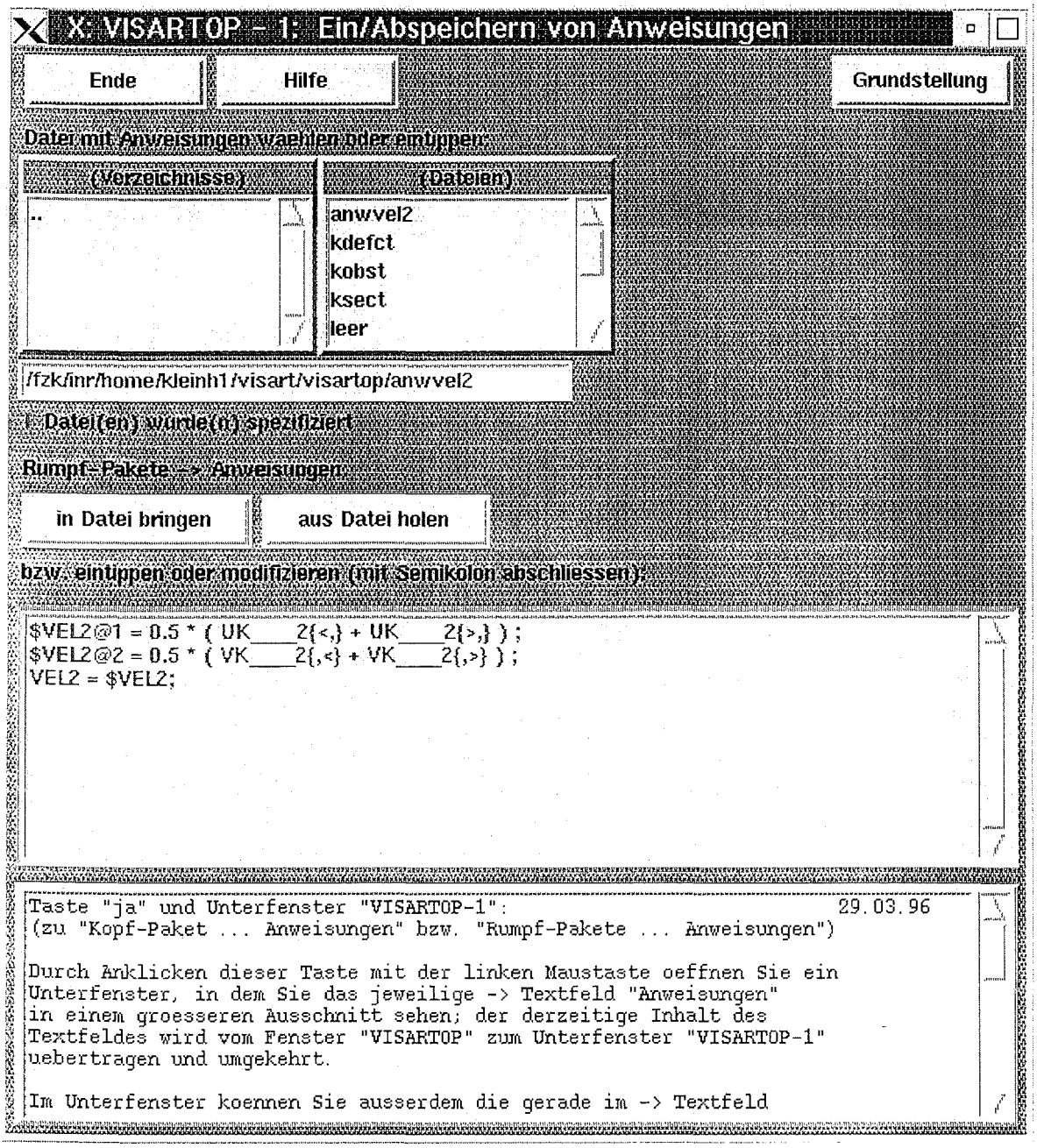

Bild 2.a: VISARTOP-Unterfenster für die Rumpf-Pakete

4. Ein- und Ausgabedateien des Programms

In diesem Kapitel wird die VISART-Eingabedatei des Programms VISARTOP und seine Ausgabedatei beschrieben.

4.1 VISART-Dateistruktur und -inhalt

Im VISART-Standard ist nur die Form, nicht aber der Inhalt der Postprocessor-Dateien festgelegt. Allerdings werden von VISARTOP einige wenige Einschränkungen der Struktur der Dateien und einige Konventionen vorausgesetzt, die im folgenden aufgeführt werden.

Bezüglich Anzahl und Format der Dateien gilt:

- Die von VISARTOP erwarteten VISART-Dateien können aus einem Erstlauf und beliebig vielen Restart-Läufen einer Anwendung bestehen. Solche Fortsetzungsdateien werden hier als „Files" *einer* VISART-„Datei" bezeichnet. VISARTOP akzeptiert eine „Datei" im obigen Sinne, mit einem oder mehreren „Files", als Eingabe (siehe 3.1.2). Die Gliederung der eingelesenen Datei kann für die ausgegebene Datei übernommen werden (siehe 3.1.2).
- Die von VISARTOP erwarteten VISART-Dateien dürfen formatiert oder unformatiert sein und im letzteren Fall einfache oder doppelte Genauigkeit der `REAL`-Daten haben. Wenn sie formatiert, d.h. in ASCII geschrieben sind, sind sie ganz unabhängig von ihrer Herkunft lesbar. Wenn sie unformatiert, d.h. binär sind, sind sie im allgemeinen nur lesbar, wenn VISARTOP unter dem gleichen Betriebssystem und dem gleichen Processor-Typ läuft, wie diejenigen, mit denen die Dateien geschrieben wurden. Für die ausgegebene Datei gilt das gleiche, wobei eine Umwandlung der Formatierung möglich ist (siehe 3.1.2).
- Da VISARTOP in Fortran geschrieben ist, müssen unformatierte VISART-Dateien, die z.B. mit C-Programmen erstellt werden, auch die interne Darstellung der Fortran-Records nachbilden (mit einem 4-Byte-Wort am Anfang und Ende jedes Records, in dem die Anzahl der Bytes des Records steht), wenn sie von VISARTOP gelesen werden sollen.

Bezüglich der Struktur gilt:

- VISARTOP erwartet, daß alle Rumpf-Pakete einer VISART-Datei gleichartig aufgebaut sind, d.h. aus der gleichen Anzahl und Anordnung gleicher Gruppen bestehen. Dies gilt auch über alle Fortsetzungs-Files (aus Restart-Läufen eines Codes) einer VISART-Datei hinweg.
- VISARTOP kann Größen der Gruppen/Subgruppen 5/15, 6/16, 7/17, 9/19, 20 und 21 aus VISART-Dateien verarbeiten. Größen der Gruppen/Subgruppen 51/151, 61/161 und 71/171 werden dagegen bisher noch nicht akzeptiert.
- In einem Kopf- oder Rumpf-Paket können mehrere Referenzgruppen 6/16, mit jeweils zugehörigen Subgruppen 7/17, und mehrere Referenzgruppe 20, mit jeweils zugehörigen Subgruppen 21, auftreten.
- Bei der Angabe von Größen der Gruppen 5/15 über Teilnetzen [1a, 1b] müssen die Identifikationen einer Größe in allen Gruppen, die jeweils ein Teilnetz überdecken, die gleichen sein. Die Reihenfolge der Teilnetze, gegeben durch die Reihenfolge der Gruppen, muß für alle Größen die gleiche sein. Die Teilnetze können, aber müssen sich nicht, überlappen.

Bezüglich der Konventionen gilt:

- Das Programm VISARTOP setzt voraus, daß die Identifikationen der Gruppen der Eingabedatei, ggf. nach Ersatz der Platzhalterzeichen, den folgenden Spezifikationen genügen: zusammengesetzt aus Buchstaben, Ziffern, Unterstrichen und Leerzeichen, wobei das erste Zeichen ein Buchstabe sein muß.
- VISARTOP erwartet die Geometrie-Gruppe 4 unter der Identifikation `GEOMETRY`.

Bezüglich der Verteilung auf Kopf-Paket und Rumpf-Pakete der VISART-Datei gilt:

- Für VISARTOP können die Gruppen mit den Angaben zur Netzgeometrie (Maschenspezifizierung, -belegung, -koordinaten) in der VISART-Datei an verschiedener Stelle stehen: 5 bzw. 6/7 im Kopf-Paket, 15 bzw. 16/17 im ersten Rumpf-Paket oder 15 bzw. 16/17 in jedem Rumpf-Paket, was sich danach richtet, ob die Geometrie und die Größen auf der Netzhülle über der Problemzeit konstant bleiben oder sich ändern. Im einzelnen gilt:
- Die Referenzgruppen 6/16 mit der Angabe der belegten Maschen oder Maschenhüllen bei defektiven Netzen (sowie die Subgruppe 7/17 mit dem Oberflächennormalenvektor) können sowohl im Kopf-Paket, als auch in den Rumpf-Paketen stehen. Stehen sie als Gruppen 6 im Kopf-Paket, so dürfen die (syntaktisch erforderlichen) entsprechenden Gruppen 16 in den Rumpf-Paketen leer sein (siehe Tab. 3.3.2, sowie B.3 in [1b]).
- Die Gruppen mit den Netzkoordinaten (Gruppen 5/15 oder Subgruppen 7/17), sowie mit der Angabe der nicht belegten Maschen bei „schwach defektiven" Netzen (Gruppen 5/15) können sowohl im Kopf-Paket als auch in den Rumpf-Paketen stehen (siehe Tab. B.2 in [1b]).

4.2 Ausgabedatei

Das in 4.1 Ausgeführte gilt für die Ausgabedatei von VISARTOP ebenso wie für die Eingabedatei.

Die Ausgabedatei ist, abgesehen von Änderungen der Gliederung und der Formatierung, hinsichtlich der übernommenen Gruppen identisch mit der Eingabedatei.[1] Immer übernommen werden die obligatorischen Gruppen im Kopf-Paket (Beschreibungsgruppen und Geometrie-Gruppe), wobei bei Zusammenfassung von Fortsetzungsfiles die Gruppen aus dem ersten File entnommen werden.

In VISARTOP neu erstellte Gruppen werden im Kopf-Paket bzw. in jedem Rumpf-Paket der Ausgabedatei hinter die übernommenen Gruppen des Abschnitts (siehe 3.2.1), in dem sie laut Steuereingabe spezifiziert wurden, eingefügt.[2]

[1] dies gilt auch in Bezug auf das alte oder neue Format des Spezifikationssatzes der Gruppen 5/15 [1b].

[2] Das Format des Spezifikationssatzes der Gruppen 5/15 (alt oder neu [1b]) richtet sich nach dem auf der Eingabedatei verwendeten.

5. Technik des Programms

In diesem Kapitel werden Aufruf, Aufbau und Charakteristiken des Programms VISARTOP beschrieben.

5.1 Programmoptionen

Das Programm VISARTOP ist in Fortran geschrieben und damit in jedem Betriebssystem implementierbar. Es läuft zur Zeit unter UNIX auf IBM-RS/6000-Workstations (sowie unter MVS auf IBM-Großrechnern, wofür es ursprünglich entworfen wurde).

Das Programm verwendet im wesentlichen Fortran-77; aus Fortran-90 wurden für eine dynamische Speicherverwaltung einige Element im Zusammenhang mit ALLOCATE-Befehlen und das Module-Konzept übernommen.

Die Versionsführung geschieht mit dem Werkzeug HISTORIAN [2]. Die aktuelle HISTORIAN-Source von VISARTOP (z.Z. Release D.4) steht unter DCE/DFS in der Datei

```
/fzk/inr/@sys/VISART/sources90/visartop90
```

Aus ihr wird in einem HISTORIAN-Lauf zunächst eine compilierbare Fortran-Source, unter Berücksichtigung der spezifizierten Optionen, erstellt. Die folgenden Optionen können kombiniert werden:

- `UNIX`: Falls diese Option „on“ ist, wird die Fortran-Source für UNIX erzeugt (andernfalls für MVS).
- `ENGLISH`: Falls diese Option „off“ ist, wird eine Fortran Source mit deutschsprachigen Nachrichten und Fehlermeldungen im Ausführungsprotokoll erzeugt; andernfalls mit englischsprachigen Texten.
- `DBLPP`: Zum Lesen von unformatierten VISART-Dateien mit einfacher Wortlänge der `REAL`-Variablen und von formatierten VISART-Dateien [1b] muß diese Option „off“ sein (das Programm erkennt aus der ersten Beschreibungsgruppe der VISART-Datei, ob es sich um eine unformatierte oder um eine formatierte Datei handelt, und stellt sich demgemäß darauf ein). Zum Lesen und Verarbeiten von unformatierten VISART-Dateien mit doppelter Wortlänge der `REAL`-Variablen muß diese Option „on“ sein.
- `SCREEN`: Falls diese Option „off“ ist, wird die Protokollausgabe mit maximal 81 Zeichen pro Zeile geschrieben; andernfalls mit maximal 133 Zeichen.

Zum automatischen Erzeugen der Fortran-Source mit unterschiedlichen Kombinationen von Optionen stehen unter DCE/DFS Shell-Scripts unter den Namen

```
/fzk/inr/@sys/VISART/sources90/visartop??.s
```

zur Verfügung, wo `??` die Optionenkombination bezeichnet (siehe 5.2). (Die Shell-Scripts setzen auch Datum und Uhrzeit der Fortran-Source-Erstellung in die dafür vorgesehenen `DATA`-Statements in VISARTOP ein, deren Inhalt bei der Ausführung des Programms ins Protokoll gedruckt wird.)

Wegen der Rekursivität der Subroutine `XPRESS` muß beim Compilieren der Fortran-Source die Option `-qnosave` gesetzt sein.

5.2 Programmaufruf und -argumente

Ausführbare Binärprogramme von VISARTOP stehen unter DCE/DFS im Verzeichnis

```
/fzk/inr/@sys/VISART/bin
```

und zwar als Datei

- `visartop` für die HISTORIAN-Option `UNIX`;
- `visartop-e` für die HISTORIAN-Optionen `UNIX, ENGLISH`;

Beim Aufruf von VISARTOP aus der VISART-Benutzeroberfläche werden bei Sprachauswahl „deutsch" (über die Menütaste „Sprache" im Hauptfenster der Benutzeroberfläche) die Binärprogramme ohne die Option `ENGLISH` ausgeführt, bei Sprachauswahl „english" die Binärprogramme mit der Option `ENGLISH`.

Das Programm VISARTOP erwartet die Steuereingabe auf seiner Standardeingabe (Unit 5) und schreibt das Ausführungsprotokoll auf seine Standardausgabe (Unit 6). Das Ausführungsprotokoll besteht aus der Wiedergabe der Steuereingabe, dem Ausdruck der Kennsätze und Spezifikationssätze aller übernommenen und erzeugten Gruppen[1] und eventuellen Fehlermeldungen. Fehlermeldungen beginnen mit dem Namen der betroffenen Subroutine; bei Fehlern, die dem Teil 3 der Eingabe zugeordnet werden können, folgt eine Markierung der Stelle in der Eingabezeile oder die Identifikation auf der linken Seite der betreffenden Anweisung; dann folgt die Diagnostik des Fehlers.

Beim Aufruf von VISARTOP über die grafische Benutzeroberfläche wird die Steuereingabe der von den Tcl/Tk-Skripts der Benutzeroberfläche aufgebauten Datei `VISART_IN` entnommen (siehe 3.4). Die Protokollausgabe wird auf die Datei `VISART_OUT` im jeweiligen Verzeichnis gelegt und im Textfeld des VISARTOP-Fensters angezeigt. Eventuelle Fehlerausgabe wird auf die Datei `VISART_ERR` im jeweiligen Verzeichnis gelegt und in eigenen Dialogfenstern angezeigt.

Neben der Steuereingabe erwartet VISARTOP eine VISART-Datei als Eingabe, aus einem oder mehreren Files bestehend (siehe 4.1), und erstellt eine Datei als Ausgabe, ebenfalls aus einem oder mehreren Files bestehend (siehe 4.2). Die Namen der laut Steuereingabe benötigten Ein- und Ausgabedateien werden als Argumente beim Programmaufruf angegeben:

```
visartop [ 10/1 [ 10/2 ... ]  [ 20/1 [ 20/2 ... ]  ]  ]
```

wo die Unit-Nummern für die Datei-/File-Namen laut nachfolgender Tabelle stehen. Beim Aufruf von VISARTOP über die grafische Benutzeroberfläche werden die im VISARTOP-Fenster eingestellte VISART-Datei, ggf. mit ihren Fortsetzungs-Files, und die eingestellte Ausgabedatei, ggf. mit ihren Fortsetzungs-Files, als Argumente übergeben.

[1] Der Ausdruck entfällt aber, wenn alle Gruppen übernommen und keine erzeugt werden.

Falls keine Datei-/File-Namen als Argumente angegeben werden, verwendet VISARTOP die in der folgenden Tabelle stehenden:

Unit	Name	Status	Format	Bedeutung
10/1	VISART1	old	form./unform.	1. gelesener VISART-File
10/2	VISART2	old	form./unform.	2. gelesener VISART-File
...	...	...	...	...
10/9	VISART9	old	form./unform.	9. gelesener VISART-File
10/10	VISART10	old	form./unform.	10. gelesener VISART-File
...	...	...	...	...
10/n	VISARTn	old	form./unform.	n. gelesener VISART-File
20/1	VISARTN1	unknown	form./unform.	1. erzeugter VISART-File
20/2	VISARTN2	unknown	form./unform.	2. erzeugter VISART-File
...	...	...	...	...
20/9	VISARTN9	unknown	form./unform.	9. erzeugter VISART-File
20/10	VISARTN10	unknown	form./unform.	10. erzeugter VISART-File
...	...	...	...	...
20/n	VISARTNn	unknown	form./unform.	n. erzeugter VISART-File

5.3 Programmaufbau

Das Hauptprogramm von VISARTOP steuert den Programmablauf in folgenden Phasen:

(Initialisierung:)

Feststellen der benötigten Länge der Tabelle B (Subroutinen INITA0, TEIL01, TEIL02, TEIL03).

(9) Einlesen des Teils 1 der Steuereingabe und Füllen der Tabelle T (Subroutine TEIL1).

- Öffnen des anstehenden Files der VISART-Datei; Lesen und Kopieren der Beschreibungsgruppen und der Geometriegruppe; Allokieren und Initialisieren der dimensionsabhängigen Felder der Tabelle B (Subroutinen CPHAD1, INITAB, CPHAD2).

(Für das Kopf-Paket des VISART-Files:)

- Einlesen des Teils 2 der Steuereingabe und Füllen der Tabelle C (Subroutine TEIL2).
- Einlesen des Teils 3 der Steuereingabe und Füllen der Tabelle B sowie der Zeichenkette STRING (Subroutine TEIL3).
- Allokieren der Hilfsfelder.
- Lesen, ggf. Kopieren und ggf. Bearbeiten der Gruppen des Kopfpakets bis zur nächsten Zyklusgruppe oder bis zum File-Ende (Subroutine SHIFTV, ZYKLUS).
- Falls Zyklusgruppe folgt: Kopieren der Zyklusgruppe und weiter mit Rumpf-Paketen; falls File-Ende und Fortsetzungsfiles vorhanden sind: zurück nach (9); andernfalls Stop.

(Für die Rumpf-Pakete des VISART-Files:)

- Einlesen des Teils 2 der Steuereingabe und Füllen der Tabelle C (Subroutine TEIL2).
- Einlesen des Teils 3 der Steuereingabe und Füllen der Tabelle B sowie der Zeichenkette STRING (Subroutine TEIL3).
- Allokieren der Hilfsfelder.

(Für jedes Rumpf-Paket des VISART-Files:)

(10) Lesen, ggf. Kopieren und ggf. Bearbeiten der Gruppen des Kopfpakets bis zur nächsten Zyklusgruppe oder bis zum File-Ende (Subroutine SHIFTV, ZYKLUS).

- Falls Zyklusgruppe folgt: Kopieren der Zyklusgruppe und zurück nach (10); falls File-Ende und Fortsetzungsfiles vorhanden sind: zurück nach (9); andernfalls Stop.

Die für das Kopf-Paket und für jedes Rumpf-Paket gerufene Subroutine ZYKLUS ist wie folgt aufgebaut:

- Setzen eines Pointers auf den Anfang der Zeichenkette STRING.

(10) Ggf. Überlesen nicht zu übernehmender Rumpf-Pakete des VISART-Files. Für das Kopf-Paket und für zu übernehmende Rumpf-Pakete des VISART-Files: Lesen des Kennsatzes der nächsten Gruppe (Subroutine PSNEXT).

- Für Gruppen ungleich 10, 6/16, 20: Lesen der weiteren Sätze der Gruppe (Subroutine REED); ggf. Speichern der Werte im dafür vorgesehenen Eintrag der Tabelle B; Kopieren als Gruppe auf den Ausgabe-File (Subroutine RITE) nach Maßgabe der Tabelle C. Zurück nach (10).
- Für Gruppen 10, 6/16, 20:

  (11) Abarbeiten des Abschnittes der Zeichenkette STRING, auf den der zugehörige Pointer zeigt, gemäß folgender Fallunterscheidung:

  . Falls der Pointer auf eine Anweisung für eine temporäre Variable zeigt: Berechnung des Ausdrucks auf der rechten Seite, Zwischenspeichern der Werte im Eintrag 0 der Tabelle B (Subroutine XPRESS) und Übertragen der Werte in den für die Variable vorgesehenen Eintrag der Tabelle B (Subroutine ASSIGN). Weiterzählen des Pointers; zurück nach (11).

  . Falls der Pointer auf eine Anweisung für eine permanente Variable zeigt: Berechnung des Ausdrucks auf der rechten Seite, Zwischenspeichern der Werte im Eintrag 0 der Tabelle B (Subroutine XPRESS) und Schreiben der Gruppe auf den Ausgabe-File (Subroutine RITE). Weiterzählen des Pointers; zurück nach (11).

  . Falls der Pointer auf eine nicht-leere Abschnittsmarkierung zeigt (die zu der zuletzt vom VISART-File gelesenen Referenzgruppe 6/16 oder 20 gehört): Lesen der weiteren Sätze der Gruppe (Subroutine REED); Speichern der Werte im dafür vorgesehenen Eintrag der Tabelle B und Kopieren als Gruppe auf den Ausgabe-File (Subroutine RITE). Weiterzählen des Pointers; zurück nach (10).

  . Falls der Pointer auf eine leere Abschnittsmarkierung zeigt (die zu der zuletzt vom VISART-File gelesenen Referenzgruppe 6/16 oder 20 gehört): Lesen der weiteren Sätze der Gruppe (Subroutine REED); Kopieren als Gruppe auf den Ausgabe-File (Subroutine RITE) nach Maßgabe der Tabelle C; Weiterzählen des Pointers; zurück nach (10).

  . Falls der Pointer auf das Ende der Zeichenkette STRING zeigt (die zu der zuletzt vom VISART-File gelesenen Zyklusgruppe 10 oder zum File-Ende gehört): Ende der Subroutine ZYKLUS.

Die von der Subroutine ZYKLUS gerufene rekursive Subroutine XPRESS wertet den in der Zeichenkette STRING verschlüsselten Ausdruck aus, auf den der im 1. Argument stehende Pointer zeigt, und gibt ihn als Tabelleneintrag in den Feldern zurück, die durch die nachfolgende Argumentgruppe spezifiziert sind (zusammengefaßt als Φ_1). Die weiteren Argumentgruppen bezeichnen Hilfsfelder für Zwischenergebnisse (zusammengefaßt als Φ, Φ_2, Φ_{H1}, Φ_{H2}). Das Programm ist wie folgt aufgebaut:

- Löschen der Merker für den monadischen Opearator und den Abfrageoperator; Setzen des Merkers für sonstige Operatoren auf das Gleichheitszeichen =.

(5) Aufruf der Subroutine GET, die die in der Zeichenkette STRING verschlüsselte Operandenbezeichnung bzw. den Operator bzw. die Funktionsbezeichnung bzw. den Strichpunkt, auf den der aktuelle Pointer zeigt, entschlüsselt und zurückgibt. Anschließend Weiterzählen des Pointers und je nach Fall:

. Falls ein Operator (siehe Tabelle 3) angetroffen wurde: Setzen des betreffenden Merkers auf den Operator; zurück nach (5).

. Falls eine Operandenbezeichnung angetroffen wurde: Übertragen des Operanden von der Tabelle B nach Φ_2; ggf. Verschieben der Werte (Subroutine IVERSH); weiter bei (50).

. Falls eine Funktionsbezeichnung ohne nachfolgende Argumente angetroffen wurde: Aufruf der Funktion (Subroutine FUNCTN) mit der Funktionsnummer als 1. Argument); Rückgabe des Ergebnisses in Φ_2; weiter bei (50).

. Falls eine Funktionsbezeichnung mit nachfolgenden Argumenten angetroffen wurde: Rekursiver Aufruf von XPRESS für die Ausdrücke der einzelnen Argumente; Rückgabe der Ergebnisse in Φ_{H1}, Φ_2, bzw. Φ_{H2}. Aufruf der Funktion (Subroutine FUNCTN) mit der Funktionsnummer als 1. Argument und den weiteren Argumenten Φ_{H1}, Φ_2, bzw. Φ_{H2}; Rückgabe des Ergebnisses in Φ_2; weiter bei (50).

. Falls eine öffnende Klammer angetroffen wurde: Rekursiver Aufruf von XPRESS für den nachfolgenden Ausdruck; Rückgabe des Ergebnisses in Φ_2; weiter bei (50).

. Falls eine schließende Klammer oder ein Strichpunkt angetroffen wurde: Ende der Subroutine XPRESS.

(50) Falls ein monadischer Operator gemerkt wurde: Verarbeiten des Operanden Φ_2 mit dem gemerkten monadischen Operator; Abspeichern des Ergebnisses in Φ_2. Anschließend je nach Fall:

. Falls ein Abfrageoperator (?)[1] gemerkt wurde: Abspeichern des Operanden in Φ_0; zurück nach (5).

. Falls als sonstiger Operator das Gleichheitszeichen (=) gemerkt wurde: Abspeichern des Operanden in Φ_1; zurück nach (5).

. Falls als sonstiger Operator ein dyadischer Operator gemerkt wurde: Verarbeiten des Operanden Φ_1 mit dem Operanden Φ_2 gemäß dem gemerkten Operator und Abspeichern des Ergebnisses in Φ_1; zurück nach (5).

. Falls als sonstiger Operator der :-Operator gemerkt wurde: Wahl des Operanden Φ_1 oder des Operanden Φ_2 nach Maßgabe des Operanden Φ_0 und Abspeichern des Ergebnisses in Φ_1; zurück nach (5).

[1] In der Zeichenkette STRING ist der ?-Operator *vor* dem Ausdruck mit der Bedingung verschlüsselt.

5.4 Einige Programmdetails

Im Programm VISARTOP werden die Tabellen T, C, B geführt, mit der jeweiligen Dimension `MAXKT=10`, `MAXKC=100`, `MAXKB`. Letztere Dimension wird aus der Eingabe (Teil 3) abgeleitet, ist aber durch `MAXKBO=101` nach oben begrenzt.

- Die Tabelle T besteht aus drei Teiltabellen, jeweils für eine der drei Möglichkeiten der Auswahl von Rumpf-Paketen im Teil 1 der Eingabe. Jeder Eintrag definiert Anfang oder Ende eines Problemzeitintervalls bzw. die Identifikation eines Problemzyklus.
- Die Tabelle C enthält Einträge für die Identifikationen aller gemäß Teil 2 der Eingabe zu übernehmenden bzw. nicht zu übernehmenden Gruppen (wobei eingegebene Namen mit den Platzhalterzeichen * und + expandiert sind und mehrfache Einträge in der Tabelle einnehmen).
- Die Tabelle B (bestehend aus mehreren Feldern für die einzelnen Parameter und Werte) enthält Einträge für alle in den Ausdrücken des Teils 3 der Eingabe vorkommenden Konstanten und eingelesenen Gruppen, sowie für alle durch die Anweisungen erzeugten Singles und temporären Gruppen. Jeder Eintrag umfaßt den Kennsatz, den Spezifikationssatz und die Datensätze der Gruppe in den beiden letzten Versionen bzw. entsprechende Daten für Singles. Die Felder für die Datensätze werden dynamisch für die Maschenzahl des jeweiligen Problems in der VISART-Datei dimensioniert. Der nullte Eintrag der Tabelle ist als Arbeitsbereich vorgesehen, die nächsten drei Einträge für die Variablen `GEOMETRY` (zusammen mit `$IJKBAR`), `$CYCLE` bzw. `$TIME`.

Weiter wird in VISARTOP die Zeichenkette `STRING` geführt, mit der Dimension `MAXKS=20000`. Dieses Feld enthält in verschlüsselter Form die Anweisungen des Teils 3 der Eingabe für das Kopf-Paket bzw. für die Rumpf-Pakete. Die Bezeichnungen der in den Ausdrücken vorkommenden Konstanten und Variablen sowie der auf der linken Seite der Anweisungen stehenden temporären Variablen sind jeweils durch eine Zeichenkette aus `ML=18` Zeichen ersetzt. In dieser Kette sind die Nummer des zugehörigen Eintrags in der Tabelle B, sowie die Qualifikatoren der Variablen verschlüsselt. Die Identifikationen permanenter Gruppen auf der linken Seite der Anweisungen stehen unverschlüsselt, aber ggf. mit verschlüsselter Ortsangabe, als Zeichenkette aus `MN=21` Zeichen im Feld. Operatoren sind durch jeweils *ein* Zeichen verschlüsselt (siehe Tabelle 3; die monadischen Operatoren werden ohne Prozentzeichen verwendet; die relationalen Operatoren werden durch Ziffern verschlüsselt; der Fragezeichen-Operator steht hier *vor* dem Ausdruck, der die Bedingung abgibt).

Der Wert des Auswahlindexes, wenn er auf die letzte Position der entsprechenden Koordinatenrichtung verweisen soll, beträgt `MAXDEF` = 999999 (siehe Fußnote zu 3.2.2 und Beispiel 11). Die quasi-unendlichen Werte, die als Ergebnis unzulässiger Operationen (siehe 2.4.3) zurückgegeben werden, sind `UNGULT` = 6.666666E66 bzw. `INGULT` = 666666666.

5.5 Einschränkungen usw.

Wie in Abschnitt 2.4.3 angemerkt, muß bis auf weiteres die Priorität *aller* dyadischen Operationen durch Klammern vorgegeben werden (d.h. die Regeln „Punkt vor Strich“ usw. gelten nicht automatisch).

An Variablenbezeichnungen auf der linken Seite von Anweisungen kann jeweils nur *einer* der Qualifikatoren Vektorkomponente und Indexausdruck angefügt werden. Für die Versionsangabe auf der linken Seite besteht keine Anwendung. An Variablenbezeichnungen in Ausdrücken müssen mehrfache Qualifikatoren in der Reihenfolge „Version, Vektorkomponente, Indexausdruck“ angefügt werden.

Zur Zeit sind noch nicht alle der beschriebenen Möglichkeiten in VISARTOP voll implementiert. Verschiebeindizes und Auswahlindizes sind nur für bestimmte Koordinatensysteme, Maschenwertlokationen, Netzbelegungen und Maschenreihenfolgen realisiert. Nicht funktionsfähige Situationen sind durch Abfragen und Fehlermeldungen abgesichert.

5.6 Aufbau der Benutzeroberfläche

Die grafische Benutzeroberfläche für die VISART-Dienste unter UNIX wurde bereits im Teil 1 der VISART-Dokumentation [1a] vorgestellt. Sie ist in der Skriptsprache Tcl/Tk implementiert und setzt die Installierung des entsprechenden Interpreters voraus, der u.a. die Tk-Funktionen mit der Xlib-Library realisiert.

Die Tcl/Tk-Skripte der VISART-Benutzeroberfläche stehen unter DCE/DFS im Verzeichnis

`/fzk/inr/@sys/VISART/tcltk`

Der Aufruf des Skripts `visart` öffnet das Hauptfenster der Benutzeroberfläche. Von dort gelangt man durch Anklicken der Tasten „VISARTOP“ in das Skript `visartop.tcl`, das das entsprechende Fenster (siehe 3.4) öffnet und ggf. in die Skripte für weitere Unterfenster verzweigt.

6. Anwendungsbeispiele

In diesem Kapitel sind Beispiele für die VISARTOP-Steuereingabe zusammengestellt, die meist praktischen Anwendungen entnommen sind. Angeführt ist jeweils nur der Teil 3 der Steuereingabe[1] mit Anweisungen zur Erzeugung neuer Gruppen, wie sie z.B. in den Kästchen der grafischen Oberfläche (siehe 3.4) anzugeben sind.

Beispiel 1 :

Im Plot-Programm HYDPLOT6 für den damaligen Fluid-Dynamics-Postprocessor-File von SIMMER-II war die Erzeugung (und Darstellung) der folgenden Größen aus den eingelesenen Daten programmiert:

- Die totale makroskopische Brutmaterialdichte;
- Die totale makroskopische Spaltmaterialdichte.

VISARTOP würde diese Größen aus den auf der Fluid-Dynamics-VISART-Datei der KV-Version des SIMMER-II.12-Codes [4] stehenden Gruppen 15 gemäß den folgenden Anweisungen erzeugen:

```
RBR1 = RSBR1 + RSBR3 + RLBR1 + RLBR6 + RGBR1 ;
RBR2 = RSBR2 + RSBR4 + RLBR2 + RLBR7 + RGBR2 ;
```

Beispiel 2 :

Im Plot-Programm NEUPLOT1 für den damaligen Integral-Data-Postprocessor-File von SIMMER-II war die Erzeugung (und Darstellung) der folgenden Größen aus den eingelesenen Daten programmiert:

- Die Reaktivität in Dollar, aus der Reaktivität und dem Anteil der verzögerten Neutronen;
- die Änderung der Dichten der drei nuklearen Materialien zwischen zwei Problemzyklen.

VISARTOP würde diese Größen aus den auf der Neutronics-VISART-Datei der KV-Version des SIMMER-II.12-Codes [4] stehenden Gruppen 15 bzw. 21 gemäß den folgenden Anweisungen erzeugen:

```
DROBR1 = ROBR1 - ROBR1";
DROBR2 = ROBR2 - ROBR2";
DROBR3 = ROBR3 - ROBR3";
TIMEREA;
_DOLLAR = _REACT / _TEBETA ;
```

Will man statt der Dichteänderung den Differenzenquotienten der Dichte über der Zeit, so lauten die entsprechenden Anweisungen:

```
DR1ODT = ( ROBR1 - ROBR1") / ( $TIME - $TIME") ;
DR2ODT = ( ROBR2 - ROBR2") / ( $TIME - $TIME") ;
DR3ODT = ( ROBR3 - ROBR3") / ( $TIME - $TIME") ;
```

Beispiel 3 :

Als neue Größe soll das Maximum zweier Dichten aus der Fluid-Dynamics-VISART-Datei der KV-Version des SIMMER-II.12-Codes [4] gebildet werden:

```
RGBRX = ( RGBR2 > RGBR3) ?  RGBR2 :  RGBR3 ;
```

[1] für die Rumpf-Pakete, wenn nichts anderes gesagt ist.

Beispiel 4 :

Aus den in der Fluid-Dynamics-VISART-Datei der KV-Version des SIMMER-II.12-Codes [4] stehenden Geschwindigkeitsvektoren sollen die Skalargrößen der Geschwindigkeitsbeträge erzeugt werden:

```
ABSVELG = %W ( %Q VELG@1 + %Q VELG@2 ) ;
ABSVELL = %W ( %Q VELL@1 + %Q VELL@2 ) ;
```

Beispiel 5 :

Im Plot-Programm PLOTALFA für den damaligen Fluid-Dynamics-Postprocessor-File von SIMMER-II war die Erzeugung (und Darstellung) der folgenden Größen aus den von der Datei eingelesenen und in der Steuereingabe eingegebenen Daten programmiert:

- Die Volumenfraktionen der 5 festen und 6 flüssigen Komponenten aus den makroskopischen und mikroskopischen Dichten der Komponenten.

VISARTOP würde diese Größen aus den auf der Fluid-Dynamics-VISART-Datei der KV-Version des SIMMER-II.12-Codes [4] stehenden Gruppen 15 gemäß den folgenden Anweisungen erzeugen (in der Gruppe 9 im Kopf-Paket, `INITIAL`, stehen die mikroskopischen Dichten):

(Eingabe für das Kopf-Paket:)

```
$RHOS1 = INITIAL{4} ;
$RHOS3 = INITIAL{6} ;
$RHOS5 = INITIAL{8} ;
$RHOS6 = INITIAL{9} ;
$RHOS7 = INITIAL{10} ;
$RHOL5 = INITIAL{17} ;
$RHOL6 = INITIAL{18} ;
$RHOL8 = INITIAL{20} ;
```

(Eingabe für das Rumpf-Paket:)

```
ALPHS1 = ( RSBR1 + RSBR2 ) / $RHOS1 ;
ALPHS2 = ( RSBR3 + RSBR4 ) / $RHOS3 ;
ALPHS3 = RSBR5 / $RHOS5 ;
ALPHS4 = RSBR6 / $RHOS6 ;
ALPHS5 = RSBR7 / $RHOS7 ;
ALPHL1 = ( RLBR1 + RLBR2 ) / RHOLMI1 ;
ALPHL2 = RLBR3 / RHOLMI2 ;
ALPHL3 = RLBR4 / RHOLMI3 ;
ALPHL4 = RLBR5 / $RHOL5 ;
ALPHL5 = ( RLBR6 + RLBR7 ) / $RHOL6 ;
ALPHL6 = RLBR8 / $RHOL8 ;
```

Beispiel 6 :

Aus den in der Fluid-Dynamics-VISART-Datei der KV-Version des SIMMER-II.12-Codes [4] stehenden Integralgrößen der Gruppe 19, INTGRLVL, soll die Masse der Liquid-Komponente 3 extrahiert und zur Bildung der Massenanteile in den Maschen verwendet werden (INTGRLNM ist die beschreibende Gruppe 9 zu den Gruppen 19; SML ist der Name für die Liquid-Komponenten-Massen):

(Eingabe für das Kopf-Paket:)

```
$VOL = VOLUMES ;
$DUMMY = INTGRLNM ;
```

(Eingabe für das Rumpf-Paket:)

```
MASSRAT3 = RLBR3 * $VOL / INTGRLVL{&SML,3} ;
```

Beispiel 7 :

Der FLUTAN-Code [7] beruht auf einem defektiven Netz [1a, 1b]. Mittels der in der VISART-Datei des Codes stehenden Referenzgruppe 16, SI_INDEX, soll die in der Subgruppe 17, SITL, stehende Temperatur in der Masche mit den Koordinaten $(1, 16, 2)$ gefunden werden:

```
SI_INDEX ;
_$TL = _SITL{&1,16,2} ;
```

Beispiel 8 :

Aus den in der VISART-Datei des MATTINA-Codes (des ehemaligen IVA3-Codes [6]) stehenden Größen über dem dreidimensionalen Netz des Codes sollen die Größen über dem zweidimensionalen Subnetz ausgewählt werden, das durch Konstanthalten der zweiten Koordinate entsteht; z.B. für die Gruppe 15, P, und $j = 1$:

```
PS = P{0,1,0} ;
```

Beispiel 9 :

In der Fluid-Dynamics-VISART-Datei der KV-Version des AFDM-Codes [3] und des SIMMER-II.12-Codes [4] stehen sowohl die Feldgeschwindigkeiten in den Maschenmittelpunkten als Vektoren als auch die Feldgeschwindigkeitskomponenten auf den Maschenhüllen (jeweils Gruppen 15). Mit VISARTOP könnte man erstere aus den letzteren erzeugen, z.B. für SIMMER:

```
$VELLS@1 = 0.5 * ( UL{<,} + UL{>,} ) ;
$VELLS@2 = 0.5 * ( VL{,<} + VL{,>} ) ;
VELLS = $VELLS ;
```

Beispiel 10 :

In einer Subroutine des MATTINA-Codes (des ehemaligen IVA3-Codes [6]), die die Ausgabe für ein spezielles Plotprogramm erstellte, war die Wichtung von Geschwindigkeitskomponenten auf den Hüllen der Maschen mit den Volumenfraktionen der jeweils stromaufwärts liegenden Maschen programmiert (jeweils Gruppen 15). Aus den in der VISART-Datei des MATTINA-Codes stehenden Geschwindigkeitskomponenten und Volumenfraktionen würde VISARTOP die gewichteten Größen gemäß den folgenden Anweisungen erzeugen (hier für Feld 1; man beachte, daß die Geschwindigkeiten auf den Maschenhüllen, die Volumenfraktionen in den Maschenmittelpunkten definiert sind):

```
$ALL{>,,} = AL___1 ;
$ALR{<,,} = AL___1 ;
$UUAL = ( UU___1 > 0.  )  ?  ( UU___1 * $ALL ) : ( UU___1 * $ALR ) ;
$ALB{,,>} = AL___1 ;
$ALT{,,<} = AL___1 ;
$WWAL = ( WW___1 > 0.  )  ?  ( WW___1 * $ALB ) : ( WW___1 * $ALT ) ;
```

Anschließend kann wie in den Beispielen 9 und 8 weiter verfahren werden, um die Größe mit einem Plotprogramm als Vektorfeld über einem zweidimensionalen Subnetz darzustellen.

Beispiel 11 :

Aus der in der Neutronics-VISART-Datei der KV-Version des SIMMER-II.12-Codes [4] als Gruppe 21, PINTG, stehenden normierten Energie soll die normierte Leistung, ebenfalls als Gruppe 21, PHISTAR, erzeugt werden (man beachte, daß die Anzahl der Differenzenquotientenwerte um eins geringer ist, als die Anzahl der Problemzeitwerte; sie werden hier dem größeren der beiden herangezogenen Problemzeitwerte zugeordnet):

```
TIMEREA;
_$PHISTAR{>} = ( _PINTG{>} - _PINTG{<} ) / ( TIMEREA{>} - TIMEREA{<} ) ;
_$PHISTAR{1} = (_PINTG{1} - _PINTG"{999999}) /
               (TIMEREA{1} - TIMEREA"{999999}) ;
_PHISTAR = _$PHISTAR ;
```

Beispiel 12 :

Im Kopf-Paket der Fluid-Dynamics-VISART-Datei der KV-Version des SIMMER-II.12-Codes [4] (und anderer Codes [3], [5], [6]) stehen außer der obligatorischen Gruppe 4, GEOMETRY, auch zwei fakultative Gruppen 5, WIDTHS und VOLUMES, mit den Maschenweiten bzw. den Maschenvolumina. Mit VISARTOP könnte man letztere aus der Geometrie-Gruppe erzeugen (Eingabe für das Kopf-Paket):

```
$KOORD1 = GEOMETRY@1 ;
$KOORD2 = GEOMETRY@2 ;
$WIDTHSS@1 = ( $KOORD1{>,} - $KOORD1{<,} ) ;
$WIDTHSS@2 = ( $KOORD2{,>} - $KOORD2{,<} ) ;
WIDTHSS = $WIDTHSS ;
$UMFANG = 3.14159 * ( $KOORD1{>,} + $KOORD1{<,} ) ;
VOLUMESS = $WIDTHSS@1 * $WIDTHSS@2 * $UMFANG ;
```

Beispiel 13 :

In der Fluid-Dynamics-VISART-Datei der KV-Version des SIMMER-II.12-Codes [4] stehen unter den Integralgrößen der Gruppen 19 auch die Massen der Komponenten (siehe Beispiel 6). Mit VISARTOP könnte man diese aus den makroskopischen Dichten der Komponenten in den Maschen und den Maschenvolumina (beides Gruppe 15) errechnen und daraus eine Gruppe 19, INTGRL, aufbauen; z.B. für die Masse der Liquid-Komponente 3:

(Eingabe für das Kopf-Paket:)

```
$VOL = VOLUMES ;
```

(Eingabe für das Rumpf-Paket:)

```
$INTGRL = #02 ( 119 , 0.  )  ;
........  = ;
$MASSL3 = #05 ( RLBR3 * $VOL ) ;
$INTGRL{16} = $MASSL3 ;
........  = ;
INTGRL = $INTGRL ;
```

Beispiel 14 :

Die in der VISART-Datei des FLUTAN-Codes [7] für das defektive Netz als Subgruppe 17 gespeicherte Temperatur SITL (siehe Beispiel 7) soll gemäß der Referenzgruppe 16 auf eine Gruppe 15, TL, über dem vollen Netz expandiert werden:

```
SI_INDEX ;
TL = #06 (_SITL , 0.  )  ;
```

Eine Gruppe 15, DEFCT, mit den „Löchern" dieses Netzes kann ergänzend dazu erzeugt werden:

```
_$SIDEFCT = #13 ( 0 ) ;
DEFCT = #06 (_$SIDEFCT , 1 ) ;
```

Beispiel 15 :

Die in der VISART-Datei des FLUTAN-Codes [7] als Subgruppe 17 gespeichete Temperatur, SSTL, auf der Netzhülle soll gemäß deren Referenzgruppe 16, SS_INDEX, und dem Oberflächennormalenvektor, Subgruppe 17, SS_NORMV, auf eine Gruppe 15 über dem Netz expandiert werden, und zwar hier für die Netzhüllflächen senkrecht zur 1. Koordinatenrichtung:

```
;
SS_INDEX ;
_$SURF1R = ( _SS_NORMV@1 < 0.  )  ?  SS_INDEX : 0 ;
_$SURF1L = ( _SS_NORMV@1 > 0.  )  ?  SS_INDEX : 0 ;
$STL1R{>,,} = #16 ( _SSTL , 0., _$SURF1R ) ;
$STL1L{<,,} = #16 ( _SSTL , 0., _$SURF1L ) ;
$STL1 = $STL1R + $STL1L ;
STL1 = $STL1 ;
```

Aus der Gruppe 15, STL1, können auch zweidimensionale Subnetze für die verschiedenen Ebenen ausgeschnitten werden (siehe Beispiel 8); z.B.:

```
STL1_0 = $STL1{0,,} ;
```

Beispiel 16 :

In einem quasi-eindimensionalen r, z-Netz ($i = 1$) (Fluid-Dynamics-VISART-Datei der KV-Version des SIMMER-III-Codes [5]) soll die Höhe einer Flüssigkeitssäule (gegeben durch die Grenze 0.028 für die Volumenfraktion ALPLK_1 der Flüssigkeit) in eine Gruppe 19, Z_COORD, geschrieben werden, um die Höhe über der Zeit darstellen zu können:

```
$ONE@1 = 0 ;
$ONE@2 = 1 ;
$LOGICAL = ALPLK_1 <= 0.028 ;
$FRSTTRU = #08 ( $LOGICAL , 1 ) ;
/* (Wenn kein .TRUE. gefunden:  #08 bringt Fehlermeldung) */
$LASTFLS = $FRSTTRU - $ONE ;
/* Wenn nur .TRUE. gefunden:  Hoehe 0 ausgeben */
$Z_COORD = ( $LASTFLS@2 > 0 ) ?
    ( (#09(GEOMETRY@2,$LASTFLS) + #09(GEOMETRY@2,$FRSTTRU)) / 2.  )  :  0.  ;
/* (Wenn nur .TRUE. gefunden:  Hoehe 0 wird ausgegeben) */
Z_COORD = #02 ( 1 , $Z_COORD ) ;
```

Beispiel 17 :

In einem zweidimensionalen r, z-Netz (Fluid-Dynamics-VISART-Datei der KV-Version des SIMMER-III-Codes [5]) soll der Fluß des Feldes 2, gegeben durch die axiale Geschwindigkeit VK__2, über die obere axiale Netzhülle integriert, in einer Gruppe 19, FLOW, ausgegeben werden:

```
$R = GEOMETRY@1 ;
$RAREA = 3.14159 * ( %Q $R{>,} - %Q $R{<,} ) ;
$RFLOW = VK__2{,>} * $RAREA ;
$Z@1 = 0 ;
$Z@2 = 999999 ;
$FLOW = #05 ( $RFLOW , $Z ) ;
FLOW = #02 ( 1 , $FLOW ) ;
```

Literatur

[1a] S. Kleinheins:
Das VISART-Konzept einer standardisierten Schnittstelle zwischen Codes und Auswerteprogrammen
Teil 1: Einführung und Überblick
FZKA 5995, 1997

[1b] S. Kleinheins:
Das VISART-Konzept einer standardisierten Schnittstelle zwischen Codes und Auswerteprogrammen
Teil 2: VISART — ein standardisiertes Postprocessor-Dateiformat
FZKA 5996, 1997

[1c] S. Kleinheins:
Das VISART-Konzept einer standardisierten Schnittstelle zwischen Codes und Auswerteprogrammen
Teil 3: VISAPTER — ein Programm zum Anschluß einiger Visualisierungssysteme an VISART-Dateien
FZKA 5997, 1999

[1e] S. Kleinheins:
Das VISART-Konzept einer standardisierten Schnittstelle zwischen Codes und Auswerteprogrammen
Teil 5: VISARVOL — ein Programm zur Visualisierung von Volumenanteilen aus VISART-Dateien
FZKA 5999, 1999

[2] HISTORIAN Plus User's Manual, Release 4.3.137
HPCSA, Austin, Texas, August 1991

[3] W.R. Bohl et al.:
AFDM: An Advanced Fluid-Dynamics Model, Vol. I - VIII
LA-11692-MS, LANL, September 1990

[4] W.R. Bohl, L.B. Luck:
SIMMER-II: A Computer Program for LMFBR Disrupted Core Analysis
LA-11415-MS, LANL, June 1990

[5] S. Kondo et al.: SIMMER-III: A Computer Program for LMFR Core Disruptive Accident Analysis, Version 2.A: User's Manual
PNC ZN9410 96-207, PNC, O-arai, June 1996

[6] N.I. Kolev:
IVA3: Computer Code for Modelling of Transient Three Dimensional Three Phase Flow in Complicated Geometry
KfK 4950, Dezember 1991

[7] H. Borgwaldt, W. Baumann, G. Willerding:
FLUTAN Input Specifications
KfK 5010, Mai 1992

Tabellen

Tabelle 1: Beispiele für Konstante und Variable

		temporär		permanent
	eingelesene Gruppe	erzeugte Gruppe	erz. Single	erzeugte Gruppe
skalar	DRUCK, RHO, VOL	$MASSE=RHO*VOL	$S=0.5	QS=$S*DRUCK+2.0
vektoriell	VEL	$WVEL=VEL*VOL	$V=VEL{3,4}	QV=$WVEL/$V+VEL

Tabelle 2: Beispiele für Qualifikatoren

	linke Seite der Anweisungen	rechte Seite der Anweisungen
Version		RHO-RHO", $TIME-$TIME"
Komponente	$VEC@1=..., $VEC@2=...	VEL@1*VEL@1+VEL@2*VEL@2
Verschiebung Auswahl	$UU{>,}=... $DREIDIM{0,1,0}=... $UU{1,}=...	(U{<,}+U{>,})/2.0 DREIDIM{0,1,0} INTEGRAL(7), INTEGRAL(MASSE,2)
Ortsangabe	DRUCK_T{3,4}=...	DRUCK_T{3,4}

Tabelle 3: Operationen und Operatoren

1 2 3

Operator	Operation	Operanden-Typ bzw. Typ des Resultats[3]				
	Monadische Operationen	R		I	L	C
+	keine[1]	R		I	-	-
-	Vorzeichenumkehr[1]	R		I	-	-
%M	Vorzeichenumkehr	R		I	-	-
%K	Kehrwert	R		-	-	-
%A	Betrag	R		I	-	-
%S	Signum	R		I	-	-
%Q	Quadrat	R		I	-	-
%W	Wurzel	R		-	-	-
%E	Exponentialfunktion	R		-	-	-
%L	Natürlicher Logarithmus	R		-	-	-
%R	Umwandlung in `REAL`	-		R	-	-
%I	Umwand. in `INTEGER`	I		-	I	-
%G	Umwand. in `LOGICAL`[2]	L		L	-	-
%N	Negation	-		-	L	-
	Dyadische Operationen	R,R	R,I	I,I	L,L	C,C
	arithmetische und logische					
+	Addition	R	R	I	-	-
-	Subtraktion	R	R	I	-	-
*	Multiplikation	R	R	I	-	-
/	Division	R	R	I	-	-
&	logisches Und	-	-	-	L	-
|	logisches Oder	-	-	-	L	-
	relationale					
<	kleiner	L	L	L	-	-
<=	kleiner oder gleich	L	L	L	-	-
>	größer	L	L	L	-	-
>=	größer oder gleich	L	L	L	-	-
==	identisch	L	L	L	-	L
!=	nicht identisch	L	L	L	-	L
	Sonstige Operationen	R		I	L	C
?	Bedingungsabfrage	-		-	L	-
:	Alternativentrennung	R		I	L	C
=	Zuweisung	R		I	L	C

[1] nur am Anfang eines Ausdrucks zulässig

[2] Operand $\neq 0$ gibt Resultat `.true.`

[3] R: `REAL`, I: `INTEGER`, L: `LOGICAL`, C: `CHARACTER`

Tabelle 4: Kombinationen von Gruppenkennzahlen

	5	6	7	9	15	16	17	19	20	21
4	5	-	-	-	15	-	-	-	-	-
5	5	-	-	-	15	-	-	-	-	-
6	-	-	7	-	-	-	-	-	-	-
7	-	7	7	-	-	-	-	-	-	-
9	-	-	-	9	-	-	-	19	-	-
15	15	-	-	-	15	-	-	-	-	-
16	-	-	-	-	-	-	17	-	-	-
17	-	-	-	-	-	17	17	-	-	-
19	-	-	-	19	-	-	-	19	-	-
20	-	-	-	-	-	-	-	-	-	21
21	-	-	-	-	-	-	-	-	21	21

Tabelle 5: Kombinationen von Gruppen, Singles, Vektoren, Skalaren

	Single skalar	Single vektoriell	Gruppe skalar	Gruppe vektoriell
Single skalar	Single skalar	Single vektoriell	Gruppe skalar	Gruppe vektoriell
Single vektoriell	Single vektoriell	Single vektoriell	Gruppe vektoriell	Gruppe vektoriell
Gruppe skalar	Gruppe skalar	Gruppe vektoriell	Gruppe skalar	Gruppe vektoriell
Gruppe vektoriell	Gruppe vektoriell	Gruppe vektoriell	Gruppe vektoriell	Gruppe vektoriell

Tabelle 6: Funktionen

#01 Erzeugung einer Gruppe 5 oder 15 [2] mit konstantem Wert (Argument 1) in allen Maschen[1].

#02 Erzeugung einer Gruppe 9 oder 19 mit vorgegebener Anzahl (Argument 1) von Elementen und konstantem Wert (Argument 2) in allen Elementen.

#03 Erzeugung eines Singles mit dem Wert des Maximums (ggf. jeder Komponente) der Maschenwerte[1] einer beliebigen Gruppe (Argument 1).

#04 Erzeugung eines Singles mit dem Wert des Minimums (ggf. jeder Komponente) der Maschenwerte[1] einer beliebigen Gruppe (Argument 1).

#05 Erzeugung eines Singles mit dem Wert der Summe (ggf. jeder Komponente) der Maschenwerte[1] einer beliebigen Gruppe (Argument 1).

#06 Expansion einer Subgruppe 7 oder 17 [2] (Argument 1) auf eine Gruppe 5 bzw. 15 gemäß den Koordinaten in der zugehörigen Referenzgruppe 6 bzw. 16; mit Default-Werten (Argument 2) der Gruppe 5 bzw. 15.[3]

#07 Kompression einer Gruppe 5 bzw. 15 [2] (Argument 1) in eine Subgruppe 7 oder 17 gemäß den Koordinaten in der zugehörigen Referenzgruppe 6 bzw. 16.

#08 Erzeugung eines Singles mit dem Wert der Position von `.true.` in einer beliebigen Gruppe (Argument 1); bei mehrfachem Vorkommen von `.true.` in der Gruppe: mit der Position des n-ten Auftretens (Argument 2).

#09 Erzeugung eines Singles mit dem Wert einer beliebigen Gruppe (Argument 1) an einer gegebenen Position (Argument 2)

#10 Erzeugung einer Gruppe 9/19 mit den Werten der Positionen von `.true.` in einer beliebigen Gruppe (Argument 1).

#11 Erzeugung einer Gruppe 9/19 mit den Werten einer beliebigen Gruppe (Argument 1) an den Positionen, die durch eine Gruppe 9/19 (Argument 2) gegeben sind.

#12 Erzeugung einer skalaren Gruppe 5/15 auf den Lokationen -88 mit konstanten Werten auf der inneren Netzhülle (Argument 1) und auf der äußeren Netzhülle (Argument 2) in den Koordinatenrichtungen.

#13 Erzeugung einer Subgruppe 7 oder 17 [2] mit konstantem Wert (Argument 1) in den gemäß der zugehörigen Referenzgruppe 6 bzw. 16 definierten Koordinaten.

#14 Erzeugung einer Gruppe (Kennzahl wie Argument 1) mit den Positionen der Werte einer beliebigen Gruppe (Argument 1) in sortierter Reihenfolge (numerisch bzw. alphabetisch).

#15 Erzeugung einer Gruppe (Kennzahl wie Argument 1) mit den Werten einer beliebigen Gruppe (Argument 1), sortiert nach den Positionen in einer anderen Gruppe (Argument 2, Kennzahl wie Argument 1).

#16 Expansion einer Subgruppe 7 oder 17 [2] (Argument 1) auf eine Gruppe 5 bzw. 15 gemäß den Koordinaten in der beschreibenden Subgruppe 7/17 [2] (Argument 3) mit Default-Werten (Argument 2) der Gruppe 5 bzw. 15.[3] [4]

#17 Kompression einer Gruppe 5 bzw. 15 [2] (Argument 1) in eine Subgruppe 7 oder 17 gemäß den Koordinaten in der beschreibenden Subgruppe 7/17 [2] (Argument 2).[4]

[1] [2] [3] [4] Fußnoten siehe nächste Seite.

Tabelle 6 (Forts.): Funktionen

#90 Erzeugung einer Gruppe 5 oder 15 mit konstantem Wert (Argument 1) in allen Maschen[1], für Lokation 99.

#91 Erzeugung einer Gruppe 5 oder 15 mit konstantem Wert (Argument 1) in allen Maschen[1], für Lokation 1.

#92 Erzeugung einer Gruppe 5 oder 15 mit konstantem Wert (Argument 1) in allen Maschen[1], für Lokation 2.

#93 Erzeugung einer Gruppe 5 oder 15 mit konstantem Wert (Argument 1) in allen Maschen[1], für Lokation 4.

#94 Erzeugung einer Gruppe 5 oder 15 mit konstantem Wert (Argument 1) in allen Maschen[1], für Lokation 10.

#95 Erzeugung einer Gruppe 5 oder 15 mit konstantem Wert (Argument 1) in allen Maschen[1], für Lokation 20.

#96 Erzeugung einer Gruppe 5 oder 15 mit konstantem Wert (Argument 1) in allen Maschen[1], für Lokation 40.

#97 Erzeugung einer Gruppe 5 oder 15 mit konstantem Wert (Argument 1) in allen Maschen[1], für Lokation 11.

#98 Erzeugung einer Gruppe 5 oder 15 mit konstantem Wert (Argument 1) in allen Maschen[1], für Lokation 22.

#99 Erzeugung einer Gruppe 5 oder 15 mit konstantem Wert (Argument 1) in allen Maschen[1], für Lokation 44.

---

[1] Bei den Funktionen `#01`, `#03`, `#04`, `#05` für Gruppen 5/15 [2] gibt das Argument 2, falls allein vorhanden, den Maschenindex in der betr. Koordinatenrichtung an, der konstant gehalten werden soll; Argument 2 und 3, falls beide vorhanden, geben den Bereich der Maschenindizes in der betr. Koordinatenrichtung an, der überdeckt werden soll. (Die Angabe 0 als Maschenindex läßt den Maschenindex frei variieren.) Bei der Funktion `#01` werden die nicht überdeckten Maschen mit dem Wert 0 bzw. 0. bzw. '' bzw. `.FALSE.` gefüllt.

[2] Für die Gruppen 5/15 bzw. 7/17 ist nur die Lokation 0 implementiert.

[3] Bei mehrfachem Vorkommen der gleichen Koordinaten(-Paare/Tripel) wird nur das erste Vorkommen berücksichtigt.

[4] Wenn eine der Koordinaten in der beschreibenden Subgruppe ≤ 0 ist, wird sie übersprungen.

Tabelle 7: Argumente und Resultate von Funktionen

#*nn*	Resultat Kz.	Vd.	Typ	Argument 1 Kz.	Vd.	Typ	Argument 2 Kz.	Vd.	Typ	Argument 3 Kz.	Vd.	Typ
#01	5/15	=A1	=A1	0	d	t	[0	D	I	[0	D	I]]
#02	9/19	=A2	=A2	0	0	I	0	d	t			
#03	0	=A1	=A1	5/15 ≠5/15	d d	R/I R/I	[0	D	I	[0	D	I]]
#04	0	=A1	=A1	5/15 ≠5/15	d d	R/I R/I	[0	D	I	[0	D	I]]
#05	0	=A1	=A1	5/15 ≠5/15	d d	R/I R/I	[0	D	I	[0	D	I]]
#06	5/15	=A1	=A1	7/17	d	t	0	=A1	=A1			
#07	7/17	=A1	=A1	5/15	d	t						
#08	0 0	D 0	I I	5/15 ≠5/15	0 0	L L	0 0	0 0	I I			
#09	0 0	=A1 =A1	=A1 =A1	5/15 ≠5/15	d d	t t	0 0	D 0	I I			
#10	9/19 9/19	D 0	I I	5/15 ≠5/15	0 0	L L						
#11	9/19 9/19	=A1 =A1	=A1 =A1	5/15 ≠5/15	d d	t t	9/19 9/19	D 0	I I			
#12	5/15	0	=A1	0	D	t	0	D	=A1			
#13	7/17	=A1	=A1	0	d	t						
#14	=A1	0	I	k	0	t						
#15	=A1	=A1	=A1	k	d	t	=A1	0	I			
#16	5/15	=A1	=A1	7/17	d	t	0	=A1	=A1	7/17	D	I
#17	7/17	=A1	=A1	5/15	d	t	7/17	D	I			
#90 ... #99: wie #01												

Erläuterungen:
Kz.: Kennzahl (k: beliebig);
Vd.: Vektordimension (d: beliebig; D: Netzdimension des Koordinatensystems);
Typ (t: beliebig; I: Integer; R: Real; L: Logical);
=A*n*: wie Argument *n*.