Universität Rostock

Fachbereich Informatik

Institut für Computergrafik

Entwicklung eines interaktiven Werkzeugs zur Ermittlung geeigneter Visualisierungstechniken

Studienarbeit

Bearbeiter: Uwe Rauschenbach
Matrikelnummer: 089206593

Betreuer: Prof. Dr. Heidrun Schumann
Kai Lukoschek

Abgabedatum: 4. August 1994

Inhaltsverzeichnis

1. Einführung

Mit der stürmischen Entwicklung der Computerhardware in den letzten Jahren wurde es möglich, leistungsfähige Systeme zur grafischen Darstellung von Daten zu implementieren, sogenannte Visualisierungssysteme. Diese enthalten eine Vielzahl von Visualisierungstechniken, die nicht-grafische Daten in grafischer Form darstellen. Meist sind sie um weitere Techniken erweiterbar.

Parallel dazu wurden immer bessere Meßverfahren und automatische Meßgeräte für eine Vielzahl wissenschaftlicher Anwendungsgebiete entwickelt. Diese liefern täglich gigantische Datenmengen, deren grafische Auswertung eine der wenigen Möglichkeiten ist, die Daten überhaupt sinnvoll zu interpretieren und zu überschauen.

Jedoch sind die Wissenschaftler bei der Auswahl einer Darstellungsart für die Visualisierung einer Datenmenge weitgehend auf sich gestellt. Wünschenswert ist somit eine Möglichkeit, den Wissenschaftler bei der Auswahl einer geeigneten Visualisierungstechnik zu unterstützen. In den letzten Jahren wurden dazu verschiedene Ansätze entwickelt.

In Erweiterung verschiedener vorliegender Arbeiten[1] wurde am FB Informatik der Universität Rostock ein formaler Ansatz zur Ermittlung geeigneter Visualisierungstechniken für multivariate Daten entwickelt[2]. Dieser wird in Abschnitt 2.1 kurz vorgestellt.

Ziel der vorliegenden Arbeit ist es, auf der Grundlage der in diesem Ansatz entwickelten Kriterien ein System zu entwerfen und zu implementieren, das den Benutzer bei der Auswahl einer geeigneten Visualisierungstechnik für sein Visualisierungsproblem unterstützt.

---

1 Siehe [LUKO93b], Seite 1f.
2 Siehe [LUKO93a].

2. Abstecken des Problemfeldes

2.1. Allgemeines

Im folgenden Abschnitt werden ausgehend von der allgemeinen Beschreibung eines Visualisierungsproblems und eines an der Universität Rostock entwickelten Beschreibungsansatzes das Feld der Arbeit abgesteckt und Teilprobleme erarbeitet.

2.2. Darstellung der zugrundeliegenden Konzepte

2.2.1. Beschreibung eines Visualisierungsproblems

Ein *Visualisierungsproblem* wird durch den *Gegenstand* und das *Ziel* der Visualisierung beschrieben. Während ersterer die zu visualisierenden Daten beschreibt, stellt letzteres den Zweck dar, der mit der Visualisierung verfolgt wird.

Als Gegenstand der Visualisierung werden in der vorliegenden Arbeit multivariate Daten angenommen. Das sind mehrdimensionale Daten, die über einem Meßgitter vorliegen. Die Daten können in Form einer Meßwerttabelle dargestellt werden, deren Spalten *Merkmale* und deren Zeilen *Beobachtungsfälle* genannt werden. Die Merkmale kann man nach zwei Gesichtspunkten klassifizieren:

1. Nach dem Raumbezug:
   Merkmale, die das Meßgitter beschreiben, heißen *Dimensionen*, alle anderen Merkmale heißen *abhängige Merkmale*.

2. Nach dem Wertebereich:
   Ein Merkmal, zwischen dessen Werten eine Ordnungsrelation besteht, heißt *ordinales Merkmal*. Sind die Werte eines Merkmals Elemente einer Aufzählungsmenge, besteht also keine Ordnungsrelation, so handelt es sich um ein *nominales Merkmal*.

Somit besteht die Spezifikation eines Visualisierungsproblems aus zwei Teilen: der Einschränkung der Datenmenge (also der Auswahl eines interessierenden Ausschnittes aus den zu visualisierenden Daten) und der Festlegung von *Interpretationszielen* (die ausdrücken, was der Benutzer aus der grafischen Darstellung erkennen will).

Bei der Einschränkung der Datenmenge kann der Benutzer durch einen von Theisel[3] entwickelten informationstheoretischen Ansatz unterstützt werden, der in einer Datenanalyse ermittelt, in welchen Merkmalen relevante Information enthalten ist und zwischen welchen Merkmalen Korrelationen bestehen.

2.2.2. Vorstellung des funktionalen Ansatzes

Um aus der Beschreibung eines Visualisierungsproblems eine Menge geeigneter Techniken abzuleiten, kommt der sogenannte *funktionale Ansatz*[4] zur Anwendung.

Dieser besteht aus drei Schritten:

- Expressivität
- Effektivität
- Angemessenheit

Die Expressivität stellt dar, inwieweit eine Technik gewisse Interpretationsziele für eine bestimmte Datenmenge erfüllt. Die Effektivität basiert auf Erkenntnissen der Wahrnehmungspsychologie und drückt aus, welche Parameter der Daten auf welche Parameter der Visualisierungstechnik abgebildet werden sollen und wie gut der Nutzer in der Lage ist, die grafisch dargebotene Information zu erfassen[5]. Die Angemessenheit schließlich drückt den Aufwand (Rechenaufwand und kognitiver Aufand des Nutzers) aus, der mit der Visualisierung verbunden ist.

Im Rahmen der vorliegenden Arbeit spielt nur das Kriterium der Expressivität[6] eine Rolle. Dieses definiert sechs Interpretationsziele[7,8]: Identifizierungsproblem, Lokalisierungsproblem, Vergleichsproblem, Verteilungsproblem, Korrelationsproblem und Häufigkeitsproblem.

---

3 Siehe [THEI92] und [THEI93].
4 Siehe [LUKO93a].
5 Wenn der Nutzer beispielsweise farbenblind ist, ist eine Darstellung durch verschiedene Farben nicht angebracht.
6 Ausführliche Beschreibung und Formeln siehe [LUKO93b].
7 Für eine Diskussion siehe [LUKO93b], Seite 4f.
8 In [THEI94] wird eine mathematisch fundierte Klassifizierung von Interpretationszielen vorgestellt. Diese wird jedoch in der vorliegenden Arbeit nicht berücksichtigt.

Das Expressivitätskriterium beinhaltet zwei Berechnungsschritte. Zunächst wird die *Eignung* einer Visualisierungstechnik für die Erfüllung eines Interpretationszieles als Funktion von Parametern der beschriebenen Datenmenge[9] ermittelt. Faßt man die Eignungen einer Technik für die Erfüllung aller Interpretationsziele in einem Vektor zusammen, erhält man den *Eignungsvektor* dieser Technik.

Für den zweiten Schritt wird ein Vektor von *Wichtungen* von Interpretationszielen benötigt. Das ist der sogenannte *Wichtungsvektor*. In einer Vergleichsrechnung wird für jede Visualisierungstechnik und jedes Interpretationsziel die Eignung mit der Wichtung verglichen. Das Ergebnis des Vergleichs heißt *Abstand*, durch Zusammenfassen der Abstände für eine Technik und alle Interpretationsziele erhält man den *Abstandsvektor* dieser Technik. Aufsummieren der Elemente des Abstandsvektors ergibt den *Gesamtabstand* der Technik. Dieser Abstand wird als Maß für die Eignung der Technik für das entsprechende Visualisierungsproblem angenommen. Die Technik wird als *geeignet* bezeichnet, wenn ihre Eignung größer als ein *Schwellwert* ist.

Falls keine geeignete Technik gefunden werden kann, wird versucht, durch *Kombination* von Techniken alle Interpretationsziele zu erfüllen. Die dieser Lösung zugrundeliegende Idee ist die Tatsache, daß verschiedene Techniken im allgemeinen unterschiedliche Eignungen für unterschiedliche Interpretationsziele aufweisen.

2.3. Ziel der Arbeit

Nach der Beschreibung des funktionalen Ansatzes sollen nun Ziel der Arbeit und sich ergebende Teilprobleme abgeleitet werden.

Ziel der Arbeit ist es, ein Werkzeug zu entwerfen und zu entwickeln, das das im funktionalen Ansatz dargestellte Kriterium der Expressivität implementiert und den Nutzer durch eine leicht zu bedienende grafische Oberfläche unterstützt. Die Integration der Kriterien Effektivität und Angemessenheit soll möglich sein, wird jedoch in vorliegender Arbeit nicht diskutiert.

Als Prämisse für die Implementation galt, daß die Werkzeuge *netCDF*[10] für die Datenmodellierung und *Tcl/Tk*[11] für die User-Interface-Gestaltung einzusetzen waren. Um schnell das Konzept anhand eines Prototypen präsentieren zu können, wurde das Werkzeug *Xf*[12],

9 Für eine Beschreibung relevanter Parameter einer Datenmenge siehe [ARND93b].
10 Siehe [REW93].
11 Siehe [OUST93].
12 Siehe [MAH93].

das auf Tcl/Tk basiert, eingesetzt. Der damit erstellte erste Prototyp wurde im Verlauf der Arbeit mit dem Projekt immer weiter vervollkommnet.

Im folgenden werden zu lösende Teilprobleme abgeleitet, deren Lösung in den Abschnitten 3 bis 6 diskutiert werden soll.

2.4. Ableitung zu lösender Probleme

2.4.1. Datendefinitions-Tool

Die Spezifizierung der zu visualisierenden Datenmenge erfordert die Eingabe von *Metadaten*, die in Form von *Attributen* der Datenmenge hinzugefügt werden. Attribute können nach zwei Kriterien klassifiziert werden:

1. Nach dem Einflußbereich (Scope):
   Es werden globale Attribute (die Charakteristika der gesamten Datenmenge beschreiben) und Variablenattribute (die Charakteristika der einzelnen Merkmale[13] beschreiben) unterschieden[14].

2. Nach dem Lebenszyklus:
   Es werden *Datenstrukturattribute*, die fest an die Beschreibung der Datenmenge gebunden sind und diese strukturieren (wie die Festlegung, ob eine Variable ordinal oder nominal ist) und *Visualisierungsattribute*, die variable Parameter des Visualisierungsproblems beschreiben (wie die Angabe, ob ein Merkmal visualisiert werden soll oder nicht) unterschieden.

Die Klassifikation nach dem Lebenszyklus impliziert eine Trennung von initialer Datenbeschreibung (die je Datensatz nur einmal durchgeführt werden muß) und Beschreibung des Visualisierungsproblems (die viele Male über demselben Datensatz durchgeführt werden kann).

Aufgrund der Wichtigkeit der initialen Datenbeschreibung sollte der Benutzer bei deren Definition durch ein Tool unterstützt werden, da auf diesen Definitionen viele der automatischen Datenanalysen aufbauen und somit Fehlentscheidungen auf diesem Gebiet gravierende Folgen auf die Visualisierung haben. Das Tool sollte außerhalb der Visualisierungs-

---

[13] Der Begriff "Variable" anstelle von "Merkmal" ist aus der netCDF-Terminologie übernommen (Sicht der Implementation), ist aber hier synonym zu betrachten.
[14] Siehe Anhang B für eine Auflistung aller verwendeten Attribute.

hilfe realisiert werden, da die Metadaten der initialen Datenbeschreibung an die Datenwerte gekoppelt und nicht von in der Visualisierungshilfe manipulierten Parametern abhängig sind. Es sollte weiterhin das Einlesen von Nutzerdaten in verschiedenen Formaten und deren Konvertierung in das netCDF-Format unterstützen.

Dieses Tool wird im Rahmen einer eigenständigen Arbeit angefertigt[15].

2.4.2. Konfliktlösekomponente für die Datenbeschreibung

Sowohl Datenmenge als auch Interpretationsziele beeinflussen die Abstandsvektoren einer Technik im funktionalen Ansatz (der Abstand ist eine Funktion von Datencharakteristika und Interpretationszielen). Die bisherige Variante des funktionalen Ansatzes[16] geht von der Annahme aus, daß die zu visualisierende Datenmenge einmalig vom Nutzer spezifiziert und danach die Eignungsberechnung durchgeführt wird. In dem Fall, daß keine geeignete Technik gefunden wird, wird der Benutzer nur bei der Einschränkung schlecht erfüllter Interpretationsziele unterstützt, da diese direkt aus den Abstandsvektoren ablesbar sind.

Um dem Nutzer mehr Kontrolle über den Konfliktlöseprozeß zu geben, sollte auch die Einschränkung der zu visualisierenden Datenmenge unterstützt werden, das heißt, das System sollte eine Darstellung anbieten, aus der der Nutzer eine Strategie zur Einschränkung der Datenmenge ableiten kann. Die Abstandsvektoren sollten also um Abstände zu den Parametern der Datenmenge erweitert werden.

Die automatische Konfliktlösung sollte auch weiterhin nur versuchen, die Erfüllung der Interpretationsziele zu optimieren. Eine automatische Einschränkung der Datenmenge ist im allgemeinen nicht möglich, da die Entscheidung, welche Daten visualisiert werden sollen und welche nicht, letztendlich beim Nutzer liegt.

Die Erweiterung der Abstandsfunktion um Abstände zu den Datenparametern wird in einer eigenständigen Arbeit bearbeitet[17].

---

15 Siehe [KAIE94].
16 Siehe [LUKO93b].
17 Siehe [THIE94].

2.4.3. Erarbeitung eines Benutzeroberflächenkonzeptes

Die Benutzeroberfläche der Visualisierungshilfe soll auf dem Prinzip der direkten Manipulation aufbauen und durch Fenstertechnik dem Benutzer erlauben, jeweils nur den ihn interessierenden Ausschnitt aus dem Visualisierungsproblem anzuzeigen. Folgende Aufgaben sind zu unterstützen:

1) Interaktive Spezifikation der Datenmenge
2) Visualisierung der Ergebnisse des informationstheoretischen Ansatzes
3) Interaktive Spezifikation der Interpretationsziele
4) Visualisierung von Konflikten zwischen Eignung von Techniken und Aspekten des Visualisierungsproblems (Abstände aus dem erweiterten funktionalen Ansatz) und Ermöglichen ihrer Lösung
5) Grafische Darstellung der Ergebnisse
6) Anzeige von Erklärungstexten und Hilfefunktion
7) Laden und Speichern von Visualisierungsproblemen

Weiterhin sind Lösungen zu finden, wie die interne Repräsentation der Abstände, der Schwellwerte für die Abstandsberechnung und der Wichtungen der Interpretationsziele[18] in eine für den Nutzer intuitiv verständliche externe Repräsentation umgewandelt werden kann.

2.4.4. Überarbeitung und Implementation der automatischen Konfliktlösung

Der vorgestellte graphentheoretische Ansatz[19] zur automatischen Konfliktlösung (also zur Kombination von Techniken, wenn keine Einzeltechnik geeignet ist) muß auf seine Eignung für die Implementierung sowie auf Schwachstellen untersucht, daraus ein Algorithmus entwickelt und implementiert werden.

2.4.5. Realisierung einer Online-Kopplung zum IRIS-Explorer

Das Ziel eines Nutzers bei der Benutzung einer Visualisierungshilfe ist letztendlich die Visualisierung der Daten. Somit ist eine gewisse Kopplung der Visualisierungshilfe mit einem Visualisierungssystem wünschenswert. Im vorliegenden Projekt wurde das System IRIS-Explorer[20] eingesetzt. Schwenck[21] untersuchte drei Möglichkeiten der Kopplung einer

---

18 Siehe [LUKO93b].
19 Siehe [LUKO93b].
20 Für eine Kurzbeschreibung des IRIS-Explorers siehe auch Abschnitt 5.

Visualisierungshilfe mit dem IRIS-Explorer. Er wählte die Möglichkeit, die Visualisierungshilfe parallel zum Visualisierungssystem laufenzulassen. Eine vierte Möglichkeit wurde von ihm außer acht gelassen: der Aufruf des Visualisierungssystems aus der Visualisierungshilfe heraus.

Diese Vorgehensweise hat gegenüber der von Schwenck gewählten folgende Vorteile:

- Der Benutzer wird nicht mit Anweisungen belastet, bestimmte Maps zu starten. Er erhält auch bei Unkenntnis der Bedienung des IRIS-Explorers eine grafische Repräsentation seiner Daten. Mit dem Explorer vertraute Benutzer können darüber hinaus Modifikationen in der Parameterbelegung vornehmen.
- Die ausgewählte Datenmenge und eine anfängliche Parameterbelegung kann dem IRIS-Explorer über ein temporäres File und ein Einlesemodul übergeben werden.

Dem stehen folgende Nachteile gegenüber:

- Der Zeitaufwand des Nutzers ist geringfügig höher, da bei jeder Änderung des Visualisierungsproblems der Explorer neu gestartet wird.
- Das System ist in etwas höherem Maße als Schwencks System abhäng vom IRIS-Explorer. Diese Abhängigkeit läßt sich jedoch auf wenige Programmodule beschränken.

Falls keine zur Visualisierung geeignete Einzeltechnik gefunden werden kann, kombiniert die Visualisierungshilfe, wie bereits oben angegeben, mehrere Techniken. Die hohe Anzahl möglicher Kombinationen und das variable Parametermapping machen es notwendig, ein Programmodul zu implementieren, das aus einer Liste von Visualisierungstechniken eine Explorer-Map generiert.

Dieses als Map-Generator bezeichnete Modul wird in einer eigenständigen Arbeit entwickelt[22].

[21] Siehe [SCHW93], Seite 69f.
[22] Siehe [DAMM94].

2.4.6. Dependency-Management

In der Menge der Attribute, die ein Visualisierungsproblem beschreiben, existieren gewisse Abhängigkeiten. Eine Änderung eines Attributes zieht die Notwendigkeit nach sich, andere Attribute neu zu berechnen und diese geänderten Daten an der Benutzeroberfläche darzustellen.

Bei einem direktmanipulativen System ist es wichtig, daß Änderungen sofort visuelle Auswirkungen haben. Ziel ist es, daß der dem Nutzer in den offenen Fenstern sichtbare Ausschnitt aus der Gesamtattributmenge immer konsistent ist.

Ein sehr mächtiger Mechanismus, das zu erreichen, ist der *Model-View-Controller* der Sprache *Smalltalk*. In der vorliegenden Arbeit soll ein ähnlicher Mechanismus unter Nutzung von Tcl/Tk entworfen und implementiert werden.

3. Benutzeroberflächenkonzept

3.1. Problemstellung

In diesem Abschnitt soll das direktmanipulative, fensterorientierte Benutzeroberflächenkonzept der Visualisierungshilfe vorgestellt werden. Dabei wird insbesondere auf die Lösung der in Abschnitt 2.4.3 aufgeworfenen Probleme eingegangen.

3.2. Externe Repräsentation interner Größen

3.2.1. Berechnung einer prozentualen Eignung

Die Eignung einer Technik für die Erfüllung aller Interpretationsziele wird durch ihren Gesamtabstand (eine real-Zahl) repräsentiert. Zur Orientierung des Benutzers über die Eignung einer Technik sollten jedoch keine Abstände, sondern die Eignung in Prozent der maximalen Eignung an der Benutzeroberfläche angeboten werden. Dabei werden folgende Mappings vorgenommen:

- Eine Eignung von 0% entspricht dem maximalen Abstand.
- Eine Eignung von 100% entspricht dem Abstand 0.
- Das Mapping wird linear angenommen.
- Die Mapping-Funktion muß bijektiv sein, da sowohl Ausgaben (Konfliktdarstellung) als auch Eingaben (Setzen von Schwellwerten) benötigt werden.

Folgende Funktionen leisten das Gewünschte[23]:

$$D_{\max} = \sum_{j=1}^{N} b_j$$

$$E\% = \begin{cases} 100 * (1 - \frac{D}{D_{\max}}) & (D_{\max} > 0) \\ 100 & (D_{\max} = 0) \end{cases}$$

$$D = D_{\max} * (1 - \frac{E\%}{100})$$

wobei

D	den Gesamtabstand der aktuellen Technik,
$D_{\max}$	den maximal möglichen Abstand,
$E\%$	die prozentuale Eignung,
b_j	die Wichtung des j-ten Interpretationszieles und
N	die Anzahl der Interpretationsziele darstellt.

3.2.2. Spezifizierung der Interpretationsziele

Im funktionalen Ansatz wird als Wichtung der Interpretationsziele eine Zahl im Intervall [0,1] angenommen. Es kann dem Benutzer jedoch nicht zugemutet werden, bei der Eingabe der Wichtung eines Interpretationszieles eine real-Zahl einzugeben. Statt dessen wird das Intervall in fünf gleich große diskrete Abschnitte eingeteilt und fünf Stufen der Wichtigkeit verbal festgelegt (siehe Tabelle 3.1).

Wichtung	verbale Wichtung
0	unwichtig
0.25	interessant
0.5	wichtig
0.75	sehr wichtig
1	äußerst wichtig

Tabelle 3.1: Wichtung von Interpretationszielen

Interaktiv können die Interpretationsziele nun durch Radiobuttons spezifiziert werden (siehe Abbildung 3.1). Thema einer Untersuchung (die jedoch nicht durchgeführt wurde) wäre es, zu prüfen, ob die angenommene Unterteilung des Intervalls in gleichgroße Teilabschnitte und die Anzahl von

Abb. 3.1: Interaktive Spezifikation von Interpretationszielen

[23] Vgl. auch [LUKO93b], Seite 5f, für die Formeln zur Eignungsberechnung.

fünf Teilabschnitten realistisch ist. Insbesondere interessant wäre, wie das aus der Auswertung psychologischer Fragebögen bekannte Phänomen der Vermeidung extremer Beurteilungen bei der Spezifikation von Interpretationszielen nach o.a. Schema auftritt.

3.3. Visualisierung von Konflikten

Bei der Visualisierung von Konflikten sollte dem Benutzer gleichzeitig eine Möglichkeit aufgezeigt werden, wie diese gelöst werden können. Die Widersprüche selbst werden durch Balkendiagramme visualisiert, mit diesen Balkendiagrammen assoziierte Bedienelemente erlauben die Manipulation der Parameter, zu denen ein Widerspruch besteht.

Widersprüche zu den Interpretationszielen werden durch die Abstandsvektoren ausgedrückt. Diese können somit in einem Balkendiagramm dargestellt und die Balken den einzelnen Interpretationszielen zugeordnet werden (siehe Abbildung 3.2). Ein langer Balken bedeutet hier einen großen Widerspruch und somit einen aussichtsreichen Kandidaten für eine Einschränkung. Bei der Einschränkung eines Interpretationszieles (durch Anklicken eines Radiobuttons links vom aktiven Radiobutton) bewegt sich auch das Ende des Balkens nach links, so daß die Vorgehensweise intuitiv ist.

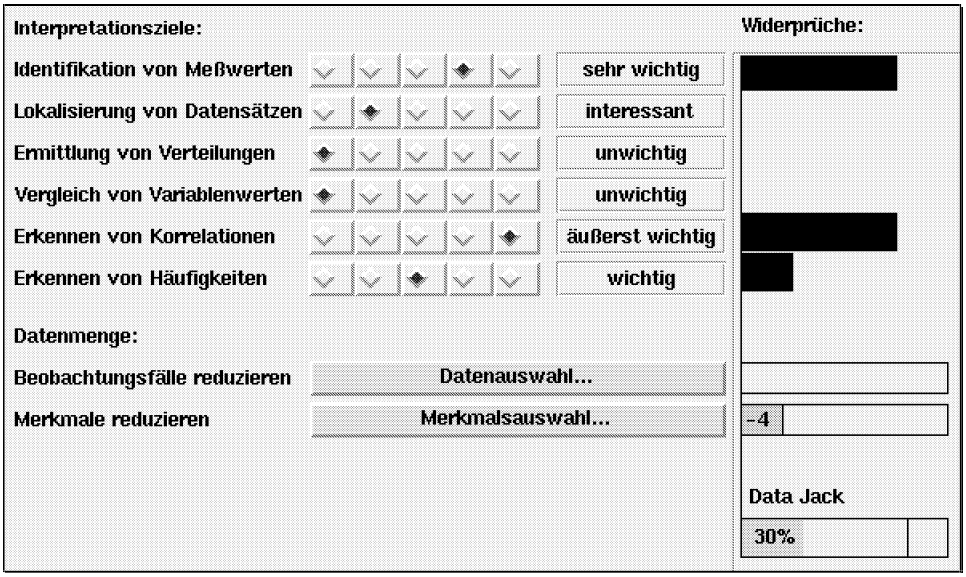


Abb. 3.2: Ausschnitt aus dem Fenster zur Visualisierung von Konflikten

Widersprüche zu Attributen der Datenbeschreibung werden als Differenz zwischen dem aktuellen Wert und dem für eine Eignung maximal zulässigen Wert eines Attributes ausgedrückt. Gegenwärtig werden Widersprüche zu den Attributen *Anzahl ausgewählter*

Merkmale und *Anzahl ausgewählter Beobachtungsfälle* berechnet[24]. Diese Differenz wird absolut als Zahl und relativ als Balken dargestellt (siehe Abbildung 3.2). Die Länge des Balkens repräsentiert hierbei den prozentualen Anteil des aktuell gewählten Attributwertes, für den eine Visualisierung noch möglich ist. Ein leeres Rechteck bedeutet, daß eine Einschränkung der Datenmenge keinen Einfluß auf die Eignung hat. Auch hier sind Attribute mit langen Balken die aussichtsreichsten Kandidaten für eine Einschränkung.

Um die prozentuale Eignung einer Technik darzustellen, wird eine ähnliche Repräsentation gewählt: Der Prozentwert wird als Text und als Balken ausgegeben (siehe Abbildung 3.2). Eine senkrechte Linie repräsentiert den Wert, den die Eignung erreichen muß, damit die Technik als geeignet angenommen wird.

Da die Balken bei den verschiedenen Teilaspekten Widersprüche zu Interpretationszielen, Widersprüche zu Datenbeschreibung und Eignung der Technik verschiedene Sachverhalte darstellen, wurde eine unterschiedliche farbliche Gestaltung gewählt.

3.4. Datenauswahlkonzepte

3.4.1. Allgemeines

Aus dem Datenbankbereich sind zwei Grundkonzepte der Auswahl einer Daten-Teilmenge aus einer relational strukturierten Datenmenge[25] bekannt: *Selektion* und *Projektion*. Auf die Begriffswelt multivariater Daten übertragen, bedeutet Selektion die Auswahl von Beobachtungsfällen und Projektion die Auswahl von Merkmalen. In der Benutzeroberfläche der Visualisierungshilfe soll die Selektion als Funktion *Datenauswahl* und die Projektion als Funktion *Merkmalsauswahl* angeboten werden.

Weiterhin ermöglicht die Visualisierung von Daten die Abbildung von Intervallen von Werten auf diskrete Attributwerte. Bespielsweise könnte ein Temperaturbereich auf eine Farbe auf der Landkarte abgebildet werden. Daher soll für ordinale Merkmale die Angabe von *Intervallen* möglich sein.

---

[24] Die Berechnung von Datenwidersprüchen ist Gegenstand einer separaten Arbeit von Thiessen [THIE94].

[25] Daten, die in Tabellenform darstellbar sind. Multivariate Daten gehören zu dieser Klasse.

In den folgenden Abschnitten wird auf die Merkmalsauswahl und die Spezifizierung von Intervallen eingegangen. Die Datenauswahl wird im Rahmen einer eigenständigen Arbeit bearbeitet[26].

3.4.2. Merkmalsauswahl und Korrelationen

Die Merkmalsauswahl muß dem Benutzer eine einfache Auswahl der zu visualisierenden Merkmale gestatten und jederzeit einen Überblick darüber ermöglichen, ob ein Merkmal visualisiert wird oder nicht. Mit Hilfe der Anzeige von Informationsgehalt, Korrelationen und Raumbezug (Dimensionen) wird dem Benutzer eine Unterstützung bei der Auswahl gegeben. Raumbezug und Informationsgehalt können einfach in der Auswahlbox für die zu visualisierenden Merkmale (Abbildung 3.3) dargestellt werden. Ein Merkmal mit relevanter Information wird im Fenster *Merkmalsauswahl* durch ein "i" links vom Merkmalsnamen gekennzeichnet, ein Merkmal mit Raumbezug (Dimension) durch ein "D". Korrelationen werden als eigenständiger Aspekt der Datenmenge in einem eigenen Fenster dargestellt (Abbildung 3.4).

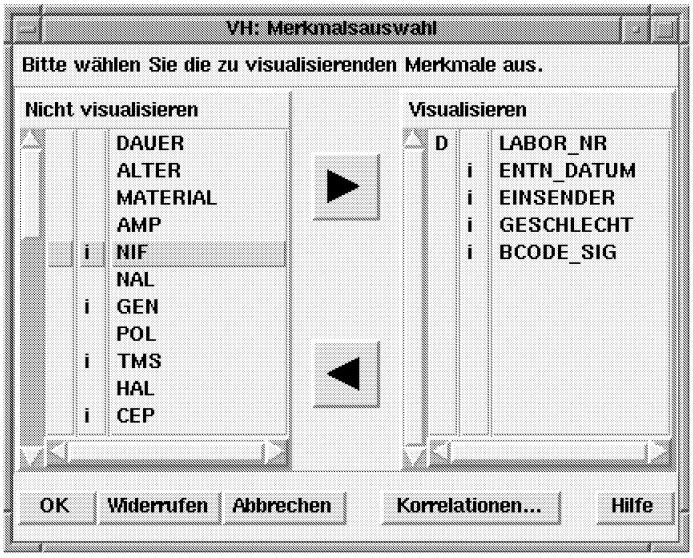


Abb. 3.3: Fenster zur Merkmalsauswahl

[26] Siehe [KAIE93] und [KAIE94].

In der linken Listbox des Fensters *Korrelationen* werden Merkmale, die Bestandteil eines Tupels korrelierender Merkmale sind, durch ein "k" links vom Merkmalsnamen gekennzeichnet. Anklicken eines solchen Merkmals stellt in der rechten Listbox im Fenster alle Tupel von Merkmalen dar, die mit diesem Merkmal korrelieren. Ein "v" links vom Merkmalsnamen bedeutet, daß das entsprechende Merkmal zur Visualisierung ausgewählt wurde. Der Benutzer kann nun ihn interessierende korrelierende Merkmale im Fenster *Merkmalsauswahl* auswählen.

Abb. 3.4: Fenster zur Anzeige von Korrelationen

3.4.3. Intervalle

Intervalle können auf zwei Arten spezifiziert werden: als Intervallgröße oder als Intervallanzahl. In der vorliegenden Realisierung wird beides angeboten (siehe Abbildung 3.5). Wenn der Benutzer die Intervallgröße ändert, wird die Intervallanzahl automatisch neu berechnet und umgekehrt.

In der vorliegenden Version wird nur eine gleichmäßige Intervallaufteilung unterstützt. Es wäre Gegenstand weiterer Überlegungen, hier alternative Unterteilungsverfahren (z.B. logarithmisch) zu entwickeln und zu implementieren.

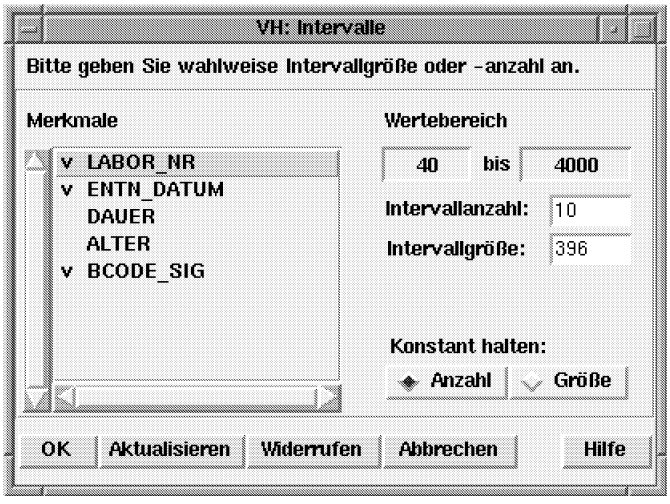


Abb. 3.5: Fenster zur Spezifizierung von Intervallen

Da sich bei der Einschränkung der Datenmenge im Zuge der Datenauswahl die Grenzen des Wertebereiches eines Merkmals ändern können, ist in einem solchen Fall ebenfalls eine Neuberechnung entweder von Intervallgröße oder Intervallanzahl notwendig. Mit Hilfe zweier Radiobuttons kann der Benutzer bestimmen, ob bei

dieser Neuberechnung die Intervallanzahl oder die Intervallgröße konstant gehalten werden soll.

3.5. Anzeige geeigneter Techniken

Das Ziel der Arbeit mit der Visualisierungshilfe ist die Ermittlung geeigneter Visualisierungstechniken oder Technikkombinationen. Alle Techniken oder Kombinationen werden dem Benutzer in einer Liste angeboten. Dem Benutzer wird ermöglicht, durch grafische Sinnbilder (Icons) das Erscheinungsbild einzelner Techniken intuitiv zu erfassen. Eine raumsparende textuelle Repräsentation wird ebenfalls angeboten.

Schließlich ist die Auswahl einer Technik oder Kombination und der Start des Visualisierungssystems IRIS-Explorer mit dieser möglich.

All diese Funktionen sind im Fenster *Empfehlung* realisiert (siehe Abbildung 3.6). Die Umschaltung zwischen grafischer und textueller Repräsentation erfolgt durch den Aktionsknopf *Icons benutzen*, der Start der Visualisierung mit einer ausgewählten Technikkombination durch Betätigen des Aktionsknopfes *Visualisierung starten*.

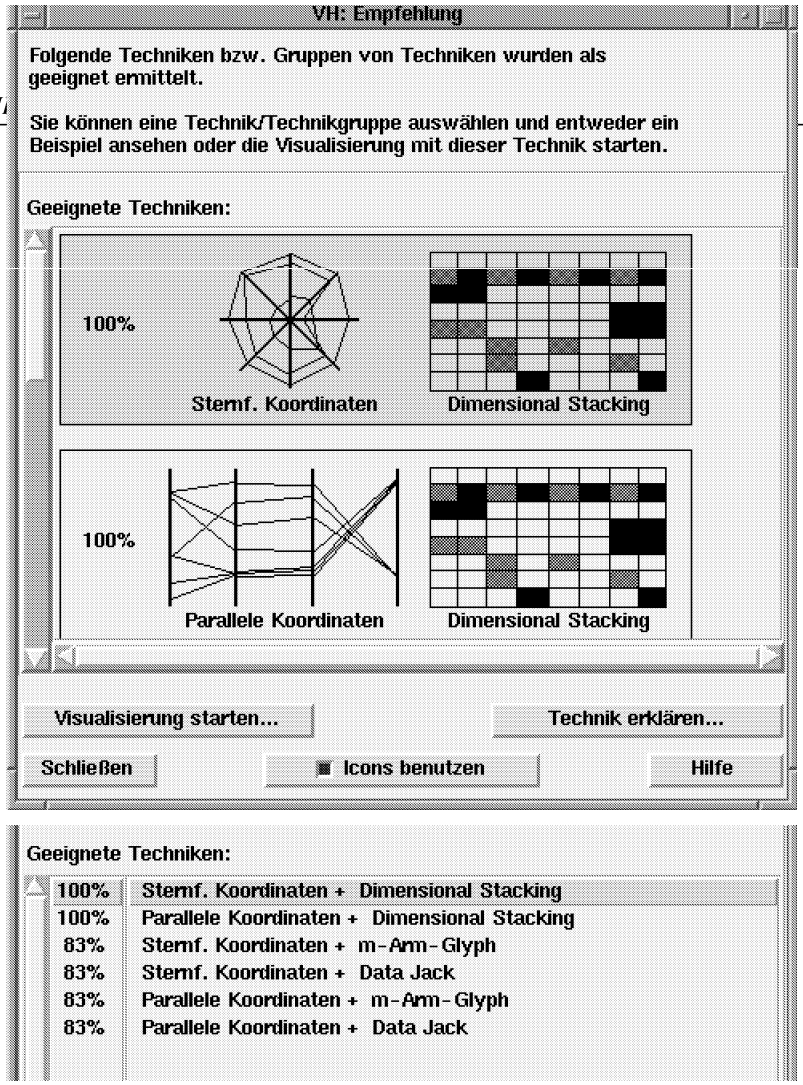


Abb. 3.6: Fenster zur Anzeige der Ergebnisse
Der Textmodus ist nur als Ausschnitt dargestellt.

3.6. Erklärungs- und Hilfesystem

Eine Visualisierungshilfe sollte dem Benutzer mehr Unterstützung bieten als die Errechnung geeigneter Visualisierungstechniken. Wichtig für das Verständnis der einzelnen Techniken sind Erklärungen und Beispiele zu ihren Grundprinzipien und Erscheinungsbildern sowie eine Online-Hilfe, die Bedienhinweise und Hintergrundinformationen zu den in den einzelnen Fenstern dargestellten Informationsausschnitten bereitstellt.

Weiterhin ist für den unerfahrenen Benutzer eine automatische Nutzerführung sehr günstig. Diese sollte den Benutzer durch das System führen, indem sie ihm die jeweils als nächstes möglichen Bearbeitungsschritte anbietet und sie erklärt. Prinzipiell ist die Nutzerführung ein Programmteil, der dem Nutzer einen Interaktionsgraphen[27] veranschaulicht.

Von diesen Vorschlägen wurden die Mechanismen zur Darstellung der Erklärungs- und Hilfetexte sowie zur Beispielanzeige implementiert. Weiterhin wurden erste unvollständige Hilfe- und Erklärungstexte erstellt. Beispielmaps zur Anzeige sowie eine automatische Nutzerführung konnten aufgrund der kurzen Zeit nicht implementiert werden.

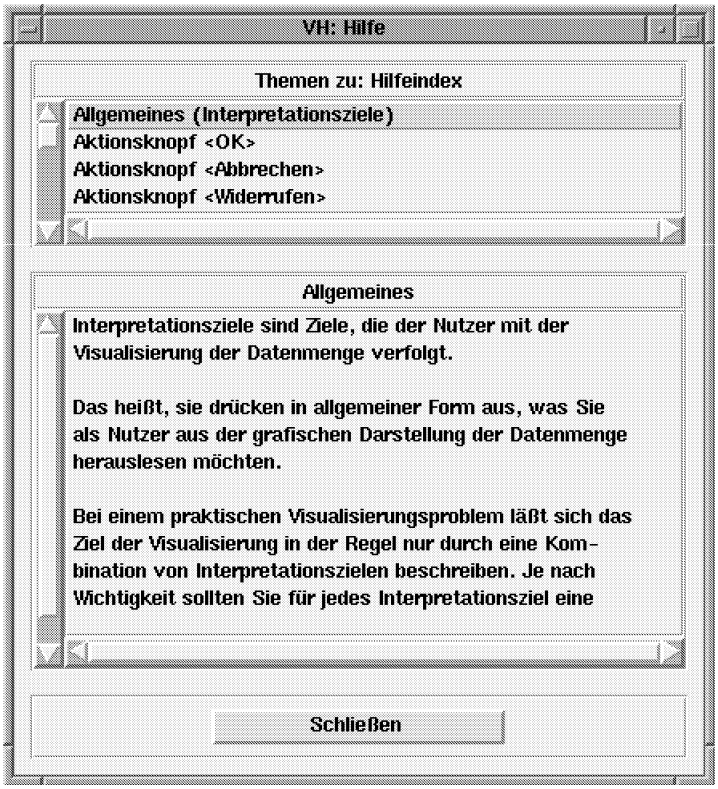


Abb. 3.7: Hilfefenster.

Die Hilfeinformation wurde dreistufig strukturiert. Die oberste Stufe adressiert einen Hilfeindex, der eine Liste von Hilfethemen (zweite Stufe) enthält. Der Text des Hilfethemas stellt die unterste Stufe dar. Die oberste Stufe dient dazu, den Kontext festzulegen. Das ist entweder das Fenster, aus dem die Hilfe angefordert wurde, oder ein globaler Hilfeindex. Die Syntax des Help-Files ist in Anhang F dokumentiert. Das Hilfefenster mit dem globalen Hilfeindex als oberste Ebene ist in Abbildung 3.7 dargestellt.

---

[27] Als Interaktionsgraph soll hier ein endlicher Automat verstanden werden, dessen Knoten Tätigkeiten des Benutzers (wie Spezifizierung von Interpretationszielen) darstellen und dessen Kanten von Bedingungen abhängige Übergänge zwischen diesen Zuständen darstellen.

Im Erklärungsfenster (siehe Abbildung 3.8) werden alle Techniken in einer Iconleiste angeboten. Durch Anklicken des Icons einer Technik bekommt der Benutzer den Erklärungstext zur entsprechenden Technik angezeigt. Betätigen des Aktionsknopfes *Beispiel* startet den IRIS-Explorer zur Anzeige einer Beispiel-Map. Die Erklärungstexte sind ebenfalls im Helpfile abgelegt (siehe Anhang F).

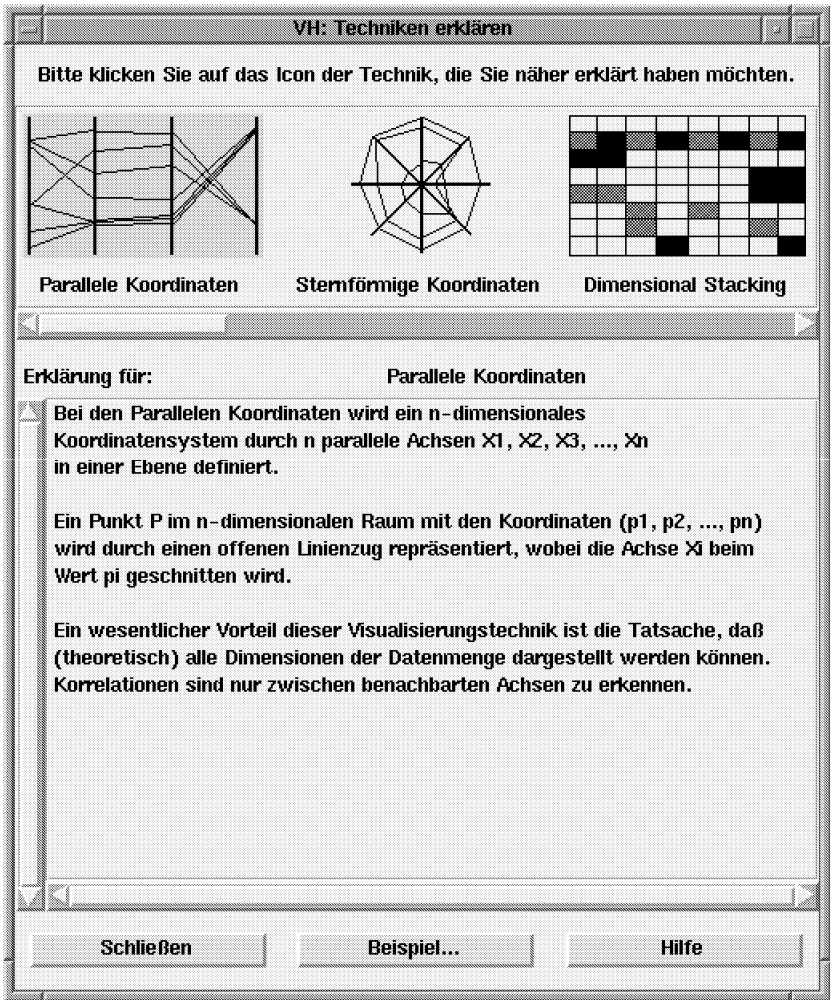


Abb. 3.8: Erklärungsfenster

4. Automatische Konfliktlösung

4.1. Problemstellung

Zu einer gegebenen Visualisierungstechnik, die als "am besten geeignete" für ein gegebenes Visualisierungsproblem gefunden wurde[28], deren Abstand aber größer als ein gewisser Schwellwert *W* ist, sollen eine oder mehrere weitere Techniken zugeordnet werden, so daß:

- der resultierende Gesamtabstand kleiner oder gleich *W* wird
- dieses mit einer minimalen Menge zusätzlicher Techniken erreicht wird.

Für eine nachfolgende Auswahl durch den Nutzer oder anhand eines Effektivitätskriteriums[29] ist es wünschenswert, daß möglichst alle Lösungen, die diese Bedingungen erfüllen, generiert werden.

Es existiert ein graphentheoretischer Ansatz von Lukoschek[30] zur Lösung dieses Problems. Dieser soll auf seine Eignung untersucht und ggf. angepaßt werden.

4.2. Grundsätzliche Überlegungen

Es handelt sich hier um ein kombinatorisches Optimierungsproblem. Gesucht werden alle Kombinationen von Techniken, die eine Gütefunktion (einen Gesamtabstand nicht größer als *W* zu besitzen) erfüllen. "Kombination" wird hier im Sinne der mathematischen Definition von "Kombinationen von *n* Elementen zur *k*-ten Klasse ohne Wiederholungen von Elementen" gebraucht.

---

[28] Siehe [LUKO93b], Seite 3ff.
[29] Siehe [LUKO93a].
[30] Siehe [LUKO93b], Seite 23ff.

Formal:

Gesucht werden alle Mengen M mit folgenden Eigenschaften:

$$M := \{t_0\} \cup \{t_i : t_i \in T\} \quad \text{mit} \quad |M| = k_{\min} \wedge GAbst(M) \leq W \tag{4.1}$$

Dabei sind

t_0	die lt. funktionalem Ansatz "am besten geeignete Technik"
t_i	eine Visualisierungstechnik aus der Menge T
T	die Menge der verfügbaren Visualisierungstechniken außer t_0
$GAbst(M)$	die (weiter unten spezifizierte) Gesamtabstandsfunktion, die auf dem funktionalen Ansatz basiert,
W	der Schwellwert
k_{min}	die Größe der geringstmächtigen Menge M von Techniken, für die $GAbst(M) \leq W$ gilt

4.3. Algorithmus zum Erzeugen aller Kombinationen C_n^k

Die Funktion dieses Algorithmus ist es, rekursiv aus einer Liste von n Techniken Listen von k Techniken zu erzeugen, die die Kombinationen dieser n Elemente zur k-ten Klasse repräsentieren. Gestartet wird mit einer leeren Ausgabeliste und einer Eingabeliste, die alle n Techniken enthält.

Der Pseudocode in Listing 4.1 beschreibt den Algorithmus. Dabei wird von "Call-by-reference" (wie z.B. für Arrays in C verwendet) ausgegangen. Um die Funktion reentrant zu machen, sind daher lokale Variablen zum Sichern der aktuellen Parameter erforderlich.

```
PROCEDURE Combine(ListOfTechniques inlist, outlist; INTEGER k);
LOCAL VARS:   ListOfTechniques local_inlist, local_outlist;
              Technique tech;
BEGIN
    IF LENGTH(outlist)=k THEN Output(outlist)
    ELSE BEGIN
        local_inlist:=inlist;
        WHILE local_inlist NOT EMPTY
        BEGIN
            tech:=FirstElement(local_inlist);
            DeleteFirstElement(local_inlist);
            local_outlist:=outlist;
            Append(local_outlist, tech);
            Combine(local_inlist, local_outlist, k)
        END;
    END;
END;
```

Listing 4.1: Allgemeiner Algorithmus zum Erzeugen aller Kombinationen von n Elementen zur k-ten Klasse

Als Beispiel wird in Abbildung 4.1 der Rekursionsbaum für die Erzeugung aller Kombinationen der Elemente (a,b,c,d) zur zweiten Klasse dargestellt. OUT kennzeichnet dabei einen Knoten, in dem das Ergebnis ausgegeben wird und keine weitere Rekursion erfolgt, CALL einen Knoten, der die Ergebnisliste weiter aufbaut und rekursiv die Funktion Combine aufruft, und EXIT einen Knoten, in dem kein Ergebnis vorliegt, der aber wegen einer leeren Eingabeliste auch keine Rekursion mehr ausführen kann.

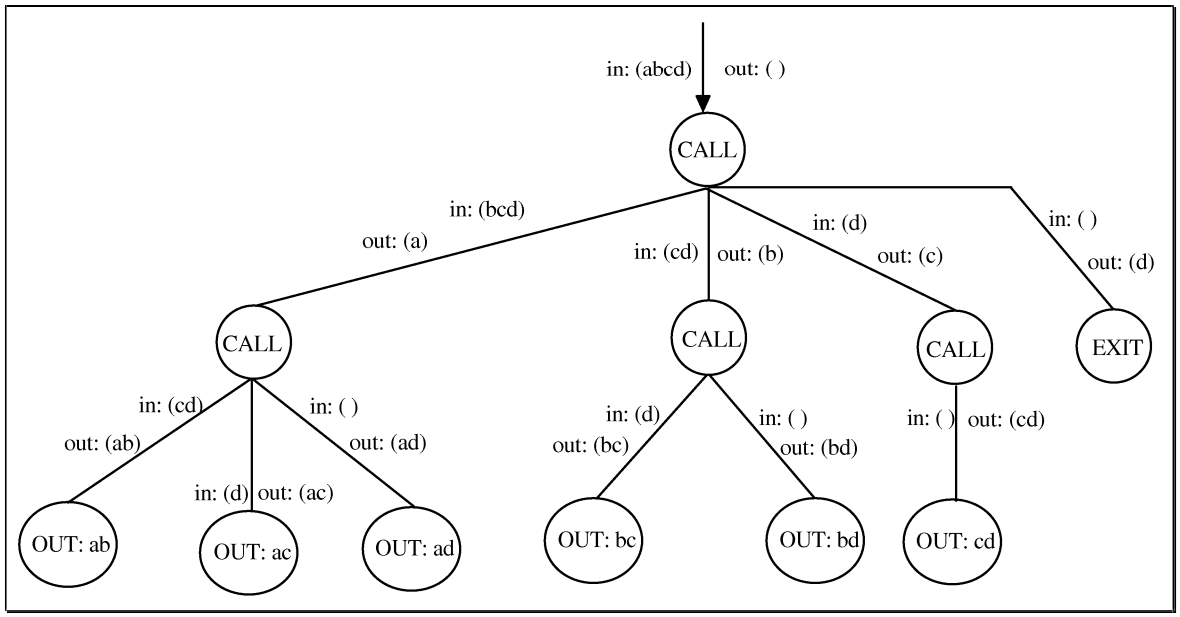


Abb. 4.1: Rekursionsbaum zur Erzeugung aller Kombinationen der Elemente (a,b,c,d) zur zweiten Klasse

4.4. Komplexitätsbetrachtungen

4.4.1. Komplexität

Im folgenden werden die Komplexität des Algorithmus in Listing 4.1 untersucht und Schlußfolgerungen für die Eignung für das vorliegende Problem gezogen.

Anzahl der OUT-Blätter im Rekursionsbaum: $N_b = \binom{n}{k}$ (4.2)

Anzahl der rekursiven Aufrufe: $N_r = 1 + \sum_{i=1}^{k} \binom{n}{i}$ (4.3.1)

$$N_r = 1 + \frac{n}{1} + \frac{n \cdot (n-1)}{1 \cdot 2} + \ldots + \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{1 \cdot 2 \cdot \ldots \cdot k} \quad (4.3.2)$$

Zur Erklärung: Das erste Glied der Summe widerspiegelt den Wurzelknoten im Aufrufbaum, jede i-te Ebene im Baum *(i=1(1)k)* erzeugt die Kombinationen von *n* zur *i*-ten Klasse.

N_b ist ein Maß für die (mögliche) Anzahl der Ergebnismengen. N_r ist ein Maß für die Rechenzeit und für den benötigten Speicherplatz für lokale Variablen. Das letzte Glied der Summe in Formel (4.3.2) ist ein Maß für die Ordnung der Komplexität.

Der Algorithmus hat somit eine Komplexität der Ordnung $O(n,k) = N_b = \binom{n}{k}$.

Die Komplexität stellt kein Problem dar für die vorliegenden Datenmengen (15 Visualisierungstechniken und 6 Interpretationsziele, also $n=15$ und $k \leq 6$). Die Rechenzeit für die Generierung aller Kombinationen ($n=15$ und $k=6$) liegt auf einem PC 486DX-40 unter zwei Sekunden. Einige ausgewählte Werte von N_b und N_r für $n=15$ in Abhängigkeit von k zeigt Tabelle 4.1.

k	N_b	N_r
6	5005	9949
3	455	473
2	105	121

Tabelle 4.1: Ausgewählte Komplexitätswerte

4.4.2. Reduktion der Datenmenge

Zur Beherrschung der Komplexität bei einer größeren Anzahl von Techniken und Zielen empfiehlt sich eine Einschränkung von *n* und *k* vor der Berechnung. Zunächst könnten durch ein *Prefiltering* Techniken anhand der Beschreibung des Visualisierungsproblems als nicht relevant verworfen und konkrete Interpretationsziele auf eine minimale Menge

abstrakter Interpretationsziele abgebildet werden (falls das möglich ist). Danach könnten all die abstrakten Interpretationsziele verworfen werden, deren Wichtung gleich 0 ist.

Danach würde die Eignungsberechnung gemäß funktionalem Ansatz erfolgen. Im Anschluß können alle die Techniken verworfen werden, deren Abstand gleich dem maximalen Abstand (d.h. der Summe der Wichtungen der Interpretationsziele) ist.

In der Praxis tritt es nun häufig auf, daß mehrere Techniken gleiche Abstandsvektoren aufweisen[31]. Diese könnten jeweils zu einer *Klasse* zusammengefaßt werden, wodurch *n* verringert wird. Weiterhin sollte *k* für praktisch nutzbare Ergebnisse nicht größer als ein gewisser Wert (Vorschlag: 3 bis 4) werden, da eine größere Anzahl gleichzeitig angebotener Visualisierungstechniken verwirrend wirkt.

Ist die Datenmenge trotz Reduktion immer noch zu groß für ein interaktives Antwortverhalten, so könnte man sich mit einer Teilanzahl von Lösungen zufriedengeben.

4.5. Berücksichtigung einer Gesamtabstandsfunktion und dynamische Berechnung von k

Um im Algorithmus eine Gesamtabstandsfunktion zu berücksichtigen, ist im Abbruchkriterium nicht nur die Mächtigkeit, sondern auch der Wert der Gesamtabstandsfunktion für die aktuelle Menge zu testen. Ist die Mächtigkeit der Menge gleich *k*, erfolgt keine weitere Rekursion. Ist außerdem der Wert der Gesamtabstandsfunktion kleiner oder gleich *W*, wird die aktuelle Menge ausgegeben.

Die dynamische Berechnung von *k* ermöglicht es, nur die geringstmächtigen Mengen auszugeben, deren Wert der Gesamtabstandsfunktion kleiner oder gleich *W* ist. Als Initialwert von *k* könnte das mögliche Maximum für *k* gewählt werden, also die Anzahl der Interpretationsziele, deren Wichtung größer als 0 ist. Das entspricht dem Extremfall, daß für jedes Interpretationsziel eine Technik eingesetzt wird.

Wenn die Gesamtabstandsfunktion berechnet wird, erfolgt gleichzeitig eine Prüfung, ob die Mächtigkeit der aktuellen Menge kleiner als die globale Variable *k* ist, der Gesamtabstand der Menge jedoch kleiner oder gleich *W*. Ist dies der Fall, wird *k* gleich der Mächtigkeit der aktuellen Menge gesetzt. Gleichzeitig können alle bisher ermittelten Lösungen gelöscht werden, da die Mächtigkeit aller dieser Mengen größer als k ist.

---

[31] Vgl. [LUKO93b], Seite 9ff.

Ist eine Menge Lösung (s.o.), so wird sie an die Ergebnisliste (eine globale Variable) angehängt.

Günstig wäre es, wenn das kleinste *k* vor dem Anhängen der ersten Ergebnis-Menge an die Ergebnisliste feststehen würde. Das würde nicht nur verhindern, daß Mengen mit einer Mächtigkeit größer als *k* an die Ergebnisliste angehängt werden, sondern auch ein modifiziertes Abbruchkriterium ermöglichen: "Tote" Rekursionen (die Teilbäume im Rekursionsbaum erzeugen würden, die nur EXIT-Knoten als Blätter besitzen) könnten dadurch verhindert werden, daß nur dann eine Rekursion erfolgt, wenn in `local_inlist` noch genug Elemente enthalten sind, um `outlist` auf die Länge *k* anwachsen zu lassen.

Während der dynamischen Berechnung von *k* muß die Rekursion jedoch bis zur leeren `local_inlist` laufen, da *k* noch nicht feststeht.

Um sicherzustellen, daß das kleinste *k* vor Anhängen des ersten Ergebnisses feststeht, sollte die Liste der zu kombinierenden Elemente so sortiert werden, daß das erste Element die größte und das letzte Element die geringste Abstandsverbesserung bringt.

Obwohl dieses Verfahren wünschenswert wäre, wird im Abschnitt 4.7 gezeigt, daß es für die Konfliktlösung nach Modifikation des graphentheoretischen Ansatzes impraktikabel ist.

4.6. Modifikation des graphentheoretischen Ansatzes

Der graphentheoretische Ansatz[32] geht von der Annahme aus, daß eine Technik t_0 am besten zur Lösung eines gegebenen Visualisierungsproblems geeignet, jedoch ihr Abstand D_0 größer als ein gegebener Schwellwert W ist. Ausgehend von der Eignung der verbleibenden Techniken für die Erfüllung einzelner Interpretationsziele wird dieser Abstand durch Hinzunahme weiterer geeigneter Techniken so verbessert, daß der Schwellwert unterschritten wird. Bei diesem Herangehen treten zwei Probleme auf, die in Tabelle 4.2 illustriert werden.

Technik	(Gesamt-) Abstandsvektor I_1	I_2	I_3	(Gesamt-) Abstand
T_1	0.25	1	1	2.25
T_2	0.4	0.4	1	2.0
T_3	1	0.25	1	2.25
T_4	0.75	0.75	0	1.5
T_5	0.75	0	0.75	1.5
{T_4, T_1, T_3}	0.25	0.25	0	0.5
{T_4, T_2}	0.4	0.4	0	0.8

Tabelle 4.2: Beispielwerte zur Illustration der Probleme mit dem bisherigen graphentheoretischen Ansatz

[32] Siehe [Luko93b], Seite 24ff.

Problem 1: Erzeugung der Kombinationen ist problematisch bei mehreren "am besten geeigneten Techniken"

Da der Algorithmus von einer Liste aller zu kombinierenden Techniken ausgeht, wäre die Vorgehensweise folgende:

1. Initialisiere T mit der Liste aller Techniken außer der ersten "am besten geeigneten" Technik t_{01}, die für mindestens ein Interpretationsziel die größte Verbesserung des Abstandes bringen.
2. Generiere alle Mengen von Techniken, die die folgende Gütefunktion erfüllen: Nimmt man $k-1$ Techniken aus der Menge T zu t_{01} hinzu, so ist der Gesamtabstand von $\{t_{01}\} \cup T$ kleiner oder gleich W.
3. Wiederhole (1) und (2) für alle am besten geeigneten Techniken t_{0i} ($i>1$).

Diese Vorgehensweise erzeugt jedoch einige Mengen doppelt.

Ein einfaches Beispiel (siehe Tabelle 4.2):
Gegeben sind die Techniken T_1 bis T_5 und $W=1.0$. Die Techniken T_4 und T_5 sind "am besten geeignet" - ihre Abstände sind gleich, jedoch schlechter als W.
Der erste Durchlauf (Verbesserung von T_4) würde nun folgende Resultate generieren: $\{T_4, T_1\}$, $\{T_4, T_3\}$, $\{T_4, T_5\}$. Im zweiten Durchlauf (Verbesserung von T_5) würden generiert: $\{T_5, T_1\}$, $\{T_5, T_4\}$.
Jedoch sind $\{T_5, T_4\}$ und $\{T_4, T_5\}$ äquivalent. Somit würden die "am besten geeigneten" Techniken eine Sonderbehandlung oder Nachbearbeitung erfordern.

Problem 2: "Übersehen" von Lösungen

Durch die Auswahl der Techniken über die Interpretationsziele können Techniken, die für mehrere Interpretationsziele mittelmäßig gut geeignet sind, aber zu einer größeren Gesamtverbesserung des Gesamtabstandes führen, "übersehen" werden.

Ein Beispiel dafür (Werte siehe Tabelle 4.2):
Die Technik T_1 sei für das Interpretationsziel I_1 am besten geeignet, T_2 sei für die Interpretationsziele I_1 und I_2 mittelmäßig gut geeignet. Technik T_3 schließlich habe die beste Eignung für I_2. Der funktionale Ansatz habe eine Technik T_4 als insgesamt am besten geeignet ermittelt. Der Schwellwert W für die Akzeptanz einer Technik als geeignet für das Visualisierungsproblem sei 0.9.
Nach dem graphentheoretischen Ansatz würde eine Menge von $\{T_4, T_1, T_3\}$ als geeignet gefunden, da T_1 für I_1 und T_3 für I_2 am besten geeignet sind.

Jedoch erfüllt auch die Menge $\{T_4, T_2\}$ die Akzeptanzbedingung. Der graphentheoretische Ansatz würde diese Lösung nicht finden, obwohl der Vorteil gegenüber der Menge von drei Techniken in einer größeren Übersichtlichkeit besteht.

Zur Lösung dieser Probleme wird der graphentheoretische Ansatz abgewandelt: Es wird eine imaginäre Technik t_b als Ausgangspunkt genommen, deren Abstand gleich einem angenommenen schlechtestmöglichen Abstand ist. Dann wird versucht, durch Kombination dieser Technik mit anderen Techniken eine Menge zu konstruieren, deren Gesamtabstand kleiner oder gleich dem Schwellwert ist.

Die hinzuzufügenden Techniken werden anhand der erreichbaren Gesamtverbesserung des Gesamtabstandes und nicht nach der Verbesserung des Gesamtabstandes für nur ein Interpretationsziel gewählt.

4.7. Dynamische Berechnung der erreichbaren Gesamtverbesserung

Tabelle 4.3 illustriert die Berechnung der erreichbaren Gesamtverbesserung: Ausgehend vom kumulativen Abstandsvektor einer Menge von Techniken wird ein Verbesserungsvektor[33] errechnet. Die Summe der Komponenten des Verbesserungsvektors ist die Gesamtverbesserung. Diese ändert sich jedoch dynamisch für jede neue Ausgangsmenge von Techniken. Beispielsweise wäre die erreichbare Gesamtverbesserung für die Hinzunahme der Technik T_1 zur Menge $\{T_4, T_2\}$ nur noch 0.15, da lediglich für das Interpretationsziel I_1 noch eine Verbesserung erreicht werden könnte.

[33] Spalten 6 bis 8 von Tabelle 4.3

Technik (- Menge)	(kumulativer) Abstandsvektor I_1	I_2	I_3	(Gesamt-) Abstand	Verbesserung gegenüber {T_4} für I_1	I_2	I_3	Gesamtverbesserung
T_2	0.4	0.4	1	2.0	0.35	0.35	0	0.7
T_1	0.25	1	1	2.25	0.5	0	0	0.5
T_3	1	0.25	1	2.25	0	0.5	0	0.5
T_4	0.75	0.75	0	1.5	0	0	0	0
{T_4}	0.75	0.75	0	1.5				

Technik (- Menge)	(kumulativer) Abstandsvektor I_1	I_2	I_3	(Gesamt-) Abstand	Verbesserung gegenüber {T_4, T_2} für I_1	I_2	I_3	Gesamtverbesserung
T_1	0.25	1	1	2.25	0.15	0	0	0.15
T_3	1	0.25	1	2.25	0	0.15	0	0.15
T_2	0.4	0.4	1	2.0	0	0	0	0
T_4	0.75	0.75	0	1.5	0	0	0	0
{T_4, T_2}	0.4	0.4	0	0.5				

Tabelle 4.3: Dynamische Berechnung der erreichbaren Gesamtverbesserung
Techniken sind absteigend nach Gesamtverbesserung sortiert.

Der kumulative Abstandsvektor einer Technik-Menge nach der Hinzunahme einer weiteren Technik kann dann aus dem alten kumulativen Abstandsvektor dieser Menge und dem Verbesserungsvektor der hinzugefügten Technik errechnet werden.

Die folgende formale Berechnungsvorschrift beschreibt die Berechnung der Verbesserungsvektoren und der Gesamtverbesserung.

Verbesserungsvektor $\overline{V}_t^T$ für Technik t gegenüber Technik-Menge T:

$$V_{t,i}^T = \begin{cases} D_{T,i} - D_{t,i} & falls > 0 \\ 0 & sonst \end{cases} \tag{4.4}$$

Gesamtverbesserung g_t^T für Technik t gegenüber Technik-Menge T:

$$g_t^T = \sum_i V_{t,i}^T \tag{4.5}$$

Subtrahieren des Verbesserungsvektors $\overline{V_t^T}$ einer hinzugenommenen Technik t in den kumulativen Abstandsvektor $\overline{D_T}$ einer Technik-Menge T:

$$D'_{T,i} = D_{T,i} - V_{t,i}^T \tag{4.6}$$

In den Formeln (4.4) bis (4.6) sind

i der Laufindex über die Interpretationsziele,

$\overline{D_t}$ der Abstandsvektor der Technik t gemäß funktionalem Ansatz[34]

Da die durch Hinzufügen einer Technik erreichbare Gesamtverbesserung nicht nur von der Eignung dieser Technik, sondern auch vom Gesamtabstand der zu verbessernden Technik-Menge abhängt, wird sie mit wachsender Mächtigkeit der Technik-Menge kleiner. Somit muß für jede Menge eine Neuberechnung der Gesamtverbesserungen der verbleibenden Techniken und ggf. eine Neusortierung der Inputliste (siehe Abschnitt 4.5) erfolgen.

Daraus ergibt sich die Frage, ob die Sortierung sicherstellt, daß k_{min} vor der Ausgabe des ersten Ergebnisses feststeht. Das ist offensichtlich nicht der Fall. Nimmt man das Beispiel aus Tabelle 4.2 und einen Schwellwert von $W=0.6$, so würden die folgenden Lösungen in der aufgeführten Reihenfolge generiert werden:

Nr.	Technik-Menge	I_1	I_2	I_3	Gesamtabstand
1	$\{T_4, T_2, T_1, T_3\}$	0.25	0.25	0	0.5
2	$\{T_4, T_1, T_3\}$	0.25	0.25	0	0.5

Tabelle 4.4: Beispiel für das Problem mit der dynamischen Berechnung von k_{min}

Hier liegt ein Problem für größere Datenmengen: Im schlimmsten Fall wird k_{min} erst im am weitesten rechten Unterbaum des Wurzelknotens gefunden, also viele Kombinationen umsonst generiert. Für die vorliegenden Datenmengen (siehe Komplexitätsbetrachtungen) ist dieses Problem jedoch nicht akut. Verfahren zur Reduktion der Datenmenge (siehe Abschnitt 4.4.2.) können hier in einem gewissen Rahmen Verbesserungen bringen.

Sollten aufgrund größerer Datenmengen Verfahren der Datenreduktion nicht greifen, so kann anstelle des modifizierten graphentheoretischen Ansatzes der ursprüngliche graphentheoretische Ansatz verwendet werden. Dieses Verfahren erzeugt zwar nur lokal optimale (und damit schlechtere) Lösungen, ein entscheidender Vorteil ist jedoch, daß es die endgültige Berechnung von k_{min} vor dem Start der Rekursion ermöglicht, was die Datenmenge nochmals wesentlich einschränkt. Der Grund dafür ist, daß sich die erreichbaren Verbesserungen nicht dynamisch ändern, da die Verbesserungen bezüglich einzelner Interpretationsziele und nicht als Gesamtverbesserung betrachtet werden.

---

[34] Siehe [LUKO93b], Seite 5.

Weiterhin könnte der Initialwert von k_{min} so gewählt werden, daß er der Länge der geringstmächtigen Menge entspricht, die aus den Visualisierungstechniken mit der besten Eignung für einzelne Interpretationsziele zusammengesetzt wurde und deren Gesamtabstand kleiner oder gleich W ist.

Beispiel: In Tabelle 4.2 würde bei $W=0.9$ $k_{min}=3$ gewählt werden. Zunächst würde T_4 als Technik mit dem kleinsten Gesamtabstand gefunden werden - der Abstand zu I_3 wäre damit 0, der Gesamtabstand 1.5. Als nächstes würde die Technik hinzugenommen, die bezüglich eines der verbleibenden Interpretationsziele I_1 und I_2 die größte Verbesserung bringt - hier wären T_1 und T_3 möglich. Wenn T_1 gewählt wird, beträgt der Gesamtabstand 1, und lediglich I_2 kann noch verbessert werden. Das wird durch Hinzufügen von T_3 erreicht. Somit ist die Ergebnismenge $\{T_4, T_1, T_3\}$. Dieses Vorgehen entspricht dem ursprünglichen graphentheoretischen Ansatz, es ermittelt ein lokales Optimum.

Der Eignungsvektor $b_{\bar{b}}$ der imaginären schlechtestmöglichen Technik $t_{\bar{b}}$ wird durch Addieren eines Offsets zu jedem Element des Wichtungsvektors b errechnet. Das ist notwendig, damit auch in dem Fall, daß der bestmögliche Eignungsvektor genau dem Wichtungsvektor b entspricht[35], eine Verbesserung möglich ist und somit ein Resultat generiert werden kann.

4.8. Algorithmus zur automatischen Konfliktlösung

Nach den vorangegangenen Betrachtungen kann jetzt der Algorithmus zur automatischen Konfliktlösung formuliert werden.

4.8.1. Datenstrukturen

Den Kern des Algorithmus bilden die Datenstrukturen Technik-Liste und Ergebnis-Liste.

Die Technik-Liste ist eine Liste von Tupeln der Form $(t, g_t^T, \overline{V}_t^T)$, wobei t den Schlüssel der verwendeten Technik, g_t^T die durch zusätzlichen Einsatz der Technik t erreichbare Gesamtverbesserung des Gesamtabstandes eines Ergebnistupels (siehe unten) und $\overline{V}_t^T$ den Vektor der Verbesserungen der Abstände zu den einzelnen Interpretationszielen darstellen. Der aktuelle Inhalt einer Technik-Liste hängt immer von einer Menge von Techniken T ab, die verbessert werden soll und als Ergebnistupel gespeichert ist. Zur Berechnung von g_t^T und $\overline{V}_t^T$ werden die Formeln (4.4) und (4.5) verwendet. Nach jeder Neuberechnung werden

[35] Ein Beispiel wäre die Spezifizierung aller b_i *als Null.*

Tupel mit $g_t^T = 0$ entfernt und die Technik-Liste aufsteigend nach g_t^T sortiert, so daß deren letztes Element die höchste Gesamtverbesserung bringt.

Die Ergebnisliste besteht aus Ergebnistupeln, von denen jedes eine Menge von Techniken enthält, deren Gesamtabstand kleiner oder gleich W ist und die genau k_{min} Techniken enthält. Ein Ergebnistupel hat die Form (T, d_T, $\overline{D}_T$), wobei T eine Menge (Liste) von Visualisierungstechniken, d_T der Gesamtabstand dieser Menge und $\overline{D}_T$ der kumulative Abstandsvektor dieser Menge sind. Bei jedem Hinzufügen einer Technik t zu T wird $\overline{D}_T$ nach Formel (4.6) neu berechnet. Die folgende Formel beschreibt die Berechnung von d_T mit i als Index über die Interpretationsziele.

$$d_t = \sum_i D_{T,i} \tag{4.7}$$

4.8.2. Programmablauf

Zunächst werden die Abstände $\overline{D}_t$ für alle Techniken t nach dem funktionalen Ansatz[36] berechnet.

Ein Ergebnistupel wird mit einer leeren Menge von Techniken und $\overline{D}_T = \overline{b}_b$, also dem Wichtungsvektor der Interpretationsziele als schlechtestmöglichem Abstandsvektor initialisiert. Aufgrund dieses Ergebnistupels und der vom funktionalen Ansatz ermittelten Abstandsvektoren wird nun wie oben angegeben eine Technik-Liste erstellt und sortiert.

Als nächstes erfolgt ein Test, ob die Technik-Liste überhaupt Elemente enthält: Ist das nicht der Fall, so ist keine Verbesserung möglich, und ein Abbruch mit Fehlermeldung erfolgt.

Falls eine Verbesserung möglich ist, werden eine leere Ergebnisliste und k_{min} nach Formel (4.8) initialisiert.

$$k_{\min} = |B| \quad mit \quad B := \{b_i : b_i > 0\} \tag{4.8}$$

wobei

$\overline{b}$	der Wichtungsvektor der Interpretationsziele[37] und
i	der Index über alle Interpretationsziele sind.

Danach wird die Funktion CombineTechniques (Listing 4.2) rekursiv ausgeführt.

---

[36] Siehe [LUKO93b], Seite 4ff.
[37] Vgl. [LUKO93b], Seite 6.

```
TYPE TechTupel = (t, g, V);
       TechList = List OF TechTupel;
       ResultTupel = (T, d, D);
       ResultList = List OF ResultTupel;

FUNCTION CombineTechniques
              (IN TechList techList;
              IN ResultTupel resTup;
              INOUT Integer kMin;
              IN Float W;
              OUT ResultList resList);

LOCAL_VARS
   TechList lTechList;
   TechTupel techListEntry;
   ResultTupel lResTup;

BEGIN
    lTechList:=techList;

    ## Anpassen von kMin
    IF (resTup.d<=W) AND (kMin>LENGTH(resTup.T) THEN
    BEGIN
        kMin:=LENGTH(resTup.T);
        DeleteList(resList);
    END;

    ## Rekursionsabbruch, ggf. Ergebnis anhängen
    IF LENGTH(resTup.T)=kMin THEN
    BEGIN
        IF resTup.d<=W THEN AppendToList(resList, resTup);
        RETURN;
    END ELSE BEGIN
        ## Berechnen neuer Kombinationen
        FOREACH Entry OF techList DO
        BEGIN
            lResTup:=resTup;
        Append(lResTup, Entry);
        lResTup.d:=lResTup.d-Entry.g;
        FOR i:=0 TO NR_GOALS DO lResTup.D[i]:=lResTup.D[i]-Entry.V[i];

        ## Neuberechnen von g und V in allen Einträgen der Technik-Liste
        ## nach Formel (4.4) und (4.5) basierend auf lResTup.d und
        ## den Abständen aus dem funktionalen Ansatz
        ## Entfernen aller Einträge mit g=0
        ## Sortieren aufsteigend nach g
        ReComputeImprovments(lResTup.d, lTechList);

        ## Rekursion
            CombineTechniques (techList, resTup, kMin, W, resList);
        END
    END
END
```

Listing 4.2: Rekursion zur Generierung aller notwendigen Technikkombinationen

Das Löschen der angehängten Technik aus der Technik-Liste (DeleteFirstElement in Listing 4.1) ist hier nicht erforderlich, da diese Technik nach dem Anhängen keine weitere Verbesserung bringen kann, das zugehörige *g* somit 0 und die Technik entfernt wird.

4.9. Schlußbemerkungen

Im vorliegenden Abschnitt wurde ein Algorithmus zur automatischen Konfliktlösung bei der Ermittlung geeigneter Visualisierungstechniken für ein Visualisierungsproblem basierend auf dem funktionalen Ansatz von Lukoschek vorgestellt.

Dieser Algorithmus findet alle Kombinationen von Visualisierungstechniken mit einem Gesamtabstand besser als ein Schwellwert und minimaler Länge und ist damit eine Erweiterung des von Lukoschek vorgestellten graphentheoretischen Ansatzes zur Konfliktlösung.

Verfahren zur Datenreduktion wurden vorgeschlagen, jedoch nicht weiter vertieft oder implementiert. Eine Implementation des Algorithmus in C liegt im Anhang D vor.

5. Online-Kopplung der Visualisierungshilfe zum IRIS-Explorer

5.1. Problemstellung

Für die im Projekt "Visualisierungshilfe" vorgesehene Online-Kopplung von Visualisierungshilfe und IRIS-Explorer ist es notwendig, daß der Explorer aus der Visualisierungshilfe heraus gestartet werden kann. Das wird sowohl zur Anzeige von Beispielen als auch für die Visualisierung der ausgewählten Datenmenge mit der von der Visualisierungshilfe errechneten Technik bzw. Technikkombination benötigt. Dabei werden folgende Forderungen gestellt:

(1) Der Explorer[38] muß aus der Visualisierungshilfe heraus mit einer Map[39] zur Darstellung gestartet werden können.

(2) Der Explorer muß aus der Visualisierungshilfe heraus beendet werden können.

(3) Die Visualisierungshilfe muß jederzeit darüber informiert sein, ob der Explorer aktiv ist oder nicht. Insbesondere muß beim Beenden der Visualisierungshilfe ein etwa aktiver Explorer mit beendet werden.

Im Verlauf der Implementation des Map-Generators[40] ergab sich weiterhin die Forderung nach einer Einbindung einer neueren Explorer-Version (2.1) anstelle der bisher verwendeten Version 1.0.

---

[38] Wenn im Rahmen dieser Arbeit vom Explorer die Rede ist, ist immer der IRIS-Explorer Version 1.0 gemeint.
[39] Der Begriff einer Map wird in Abschnitt 5.2.1. erklärt.
[40] Vgl. [DAMM94].

5.2. Analyse des Explorers

Im folgenden werden die Prozeßstruktur und die Benutzeroberfläche des Explorers kurz analysiert und Schlußfolgerungen für die Lösung der Probleme (1) bis (3) gezogen.

5.2.1. Benutzeroberfläche des Explorers

Der Explorer ist ein komplexes System, das mindestens drei Fenster öffnet. In der Version 1.0 sind das das Menüfenster, das Librarian-Fenster und das Map-Editor-Fenster. Version 2.1 öffnet ein Librarian-Fenster, ein Map-Editor-Fenster und ein Log-Fenster. Das Librarian-Fenster enthält Bausteine *(Module)*, die im Map-Editor mit Datenflüssen zu Modulkombinationen *(Maps)* verbunden werden, die dann zur Visualisierung von Daten eingesetzt werden. Beispielsweise könnte eine einfache Map aus einem Dateneinlesemodul, einem Datenfiltermodul und einem Datenausgabemodul bestehen. Im Map-Editor ist jedes Modul durch ein Icon mit zugehörigen Datenflüssen dargestellt. Jedes Icon kann zu einem eigenen Fenster vergrößert werden, in dem Eingabe- und Ausgabeelemente enthalten sein können: Das Fenster des Dateneinlesemoduls der obigen einfachen Map könnte eine Dateiauswahlbox sein, das Filtermodul-Fenster Parametereinstellungen erlauben und das Ausgabemodul-Fenster ein Drawing Area enthalten. Diese Fenster können auch wieder in den Map-Editor hinein ikonisiert werden. Der Explorer Version 1.0 kann durch Aktivieren des Menüeintrages Admin/Quit im Menüfenster oder durch Schließen des Menüfensters beendet werden. Die Beendigung der Version 2.1 erfolgt durch Aktivieren des Menüeintrages File/Quit im Map-Editor-Fenster oder durch Schließen des Map-Editor-Fensters.

5.2.2. Prozeßstruktur des Explorers

Der Explorer besteht aus mehreren kommunizierenden Prozessen. Die Aufrufstruktur ist in Abbildung 5.1 dargestellt. Hierbei bestehen keine wesentlichen Unterschiede zwischen den Versionen 1.0 und 2.1.

Das Startmodul *explorer* ruft das User-Interface-Modul *gui* und den globalen Communicator *gc* auf. Dieser ruft seinerseits den lokalen Communicator *lc* auf, welcher die Module startet, die in der aktuellen Map enthalten sind. Obwohl der Explorer die Möglichkeit bietet, Module auf entfernten Maschinen ablaufen zu lassen, wurde dies nicht getestet. Die obige Beschreibung bezieht sich somit nur auf lokale Module, was aber für die Visualisierung multivariater Daten ausreichend zu sein scheint.

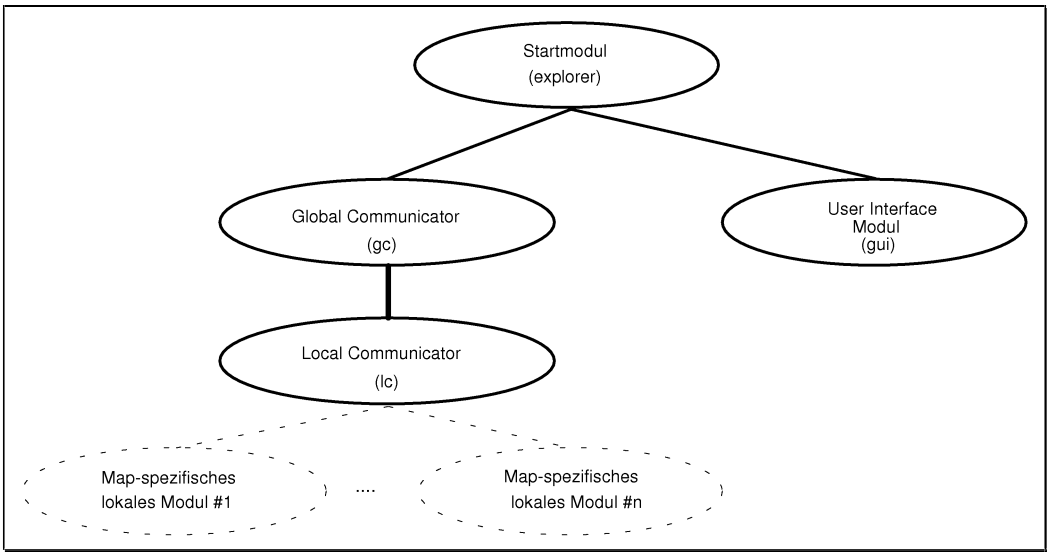


Abb. 5.1: Prozeßstruktur des Explorers

5.2.3. Probleme und Lösungsansätze

Aus den beiden vorangegangenen Abschnitten können einige Probleme erkannt und erste Schlußfolgerungen für eine Strategie zur Realisierung der Explorer-Steuerung abgeleitet werden.

Da das Startmodul beim Starten des Explorers aus der Visualisierungshilfe ein Kindprozeß von ihr ist, bietet es sich an, die Steuerung des Explorers über dieses zu realisieren: Explorer beenden heißt, das Startmodul zu beenden. Das Ende des Startmoduls kann als Ende eines Kindprozesses von der Visualisierungshilfe erkannt werden, wodurch Forderung (3) erfüllt wird. Jedoch bleiben alle anderen Explorer-Module aktiv, wenn das Startmodul *explorer* von außen (mit dem UNIX-Befehl *kill*) beendet wird. Das Beenden des Startmoduls von außen ist also nicht die Lösung für Forderung (2). Beenden der Module *gui* und *gc* von außen führt in jedem Fall zum Beenden aller anderen Module (auch des Startmoduls), somit ist das die gesuchte Lösung für (2) und (3).

Der Explorer kann jedoch auch über sein eigenes Menü oder durch Schließen des Menüfensters (Version 1.0) bzw. des Map-Editor-Fensters (Version 2.1) beendet werden. Wird der Explorer über den Menüeintrag Quit beendet, so führt das zum Beenden aller anderen Module. Wird jedoch das Menü- bzw. Map-Editor-Fenster geschlossen, werden zwar *gui* beendet und alle Fenster geschlossen, aber abhängig vom Systemzustand nicht alle anderen Prozesse, insbesondere nicht das Startmodul, beendet. Das bedeutet, daß Forderung (3) nur schwer zu erfüllen ist: Obwohl die Fenster nicht mehr dargestellt werden,

ist das Startmodul bei dieser Möglichkeit des Beendens weiterhin aktiv, so daß die Visualisierungshilfe nicht das Ende des Explorer-Prozesses erkennen kann.

Als eine mögliche Lösung für dieses Problem wurde das Entfernen des Menüfensters vom Bildschirm mit Hilfe von Routinen des X Window Systems getestet. Insbesondere beim Explorer Version 2.1 ergaben sich hier Probleme aufgrund der mangelnden Synchronisation. Zum Schließen eines Fensters muß nämlich die gesamte Window-Hierarchie rekursiv durchlaufen und das zu schließende Fenster gesucht werden. Wenn zwischen dem Erfragen des Zeigers auf ein Fenster und dem Dereferenzieren dieses Zeigers das Fenster durch vom System oder vom Benutzer ausgelöste Ereignisse geschlossen wird, zeigt der Zeiger auf einen undefinierten Speicherbereich. Ein Programmabsturz ist die Folge. Daher wurde diese Möglichkeit wieder verworfen.

Alternativ muß deshalb sichergestellt werden, daß die "Restprozesse" von der Visualisierungshilfe bei deren Beenden oder vor dem Neustart des IRIS-Explorers beendet werden.

Im folgenden Abschnitt sollen die obigen Lösungsansätze genauer betrachtet und eine Strategie zur Steuerung des Explorers abgeleitet werden.

5.3. Strategie zur Realisierung der Explorer-Steuerung

Aufgrund der Komplexität des Explorers ist anzustreben, daß nicht mehr als eine Instanz des Explorers gleichzeitig aktiv ist. Das heißt, daß vor jedem Start des Explorers geprüft wird, ob eine Explorer-Instanz aktiv ist. Ist das der Fall, wird diese zunächst beendet, bevor eine neue Explorer-Instanz gestartet wird. Das ist auch zum Erfüllen von Forderung (3) notwendig.

Aufgrund der Forderung (2) aus Abschnitt 1 kann die Visualisierungshilfe nach dem Start des Explorers nicht einfach auf dessen Beendigung warten, da sie weitere Eingaben entgegennehmen muß, um den Explorer-Prozeß wieder zu beenden. Ein Beenden des Explorers über sein eigenes Menü ist ja nach dem Unsichtbarmachen des Menüfensters nicht mehr möglich. Daher sollte das Warten einem Prozeß übertragen werden, der parallel zur Visualisierungshilfe läuft und mit dieser kommuniziert.

Folgende Strategie zur Prozeßsteuerung bietet sich an: Die Existenz des Startmoduls *explorer* dient als Indikator für die Existenz einer Explorer-Instanz. Drei Prozesse sind notwendig, um die Prozeßsteuerung zu realisieren:

A: der ursprüngliche Prozeß Visualisierungshilfe, der als User Interface fungiert und den Prozeß B erzeugt.

B: ein Kindprozeß von A, der seinerseits den Explorer als Prozeß C erzeugt und den Prozeß A von Existenz und Prozeßidentifikatoren der Prozesse *gui* und *gc* informiert. Danach wartet B auf die Beendigung von C, informiert A über diese und beendet sich schließlich selbst.

C: das Startmodul des Explorers als Kindprozeß von B.

Somit ergibt sich die in Abbildung 5.3 dargestellte Prozeßstruktur für den "Normalfall".

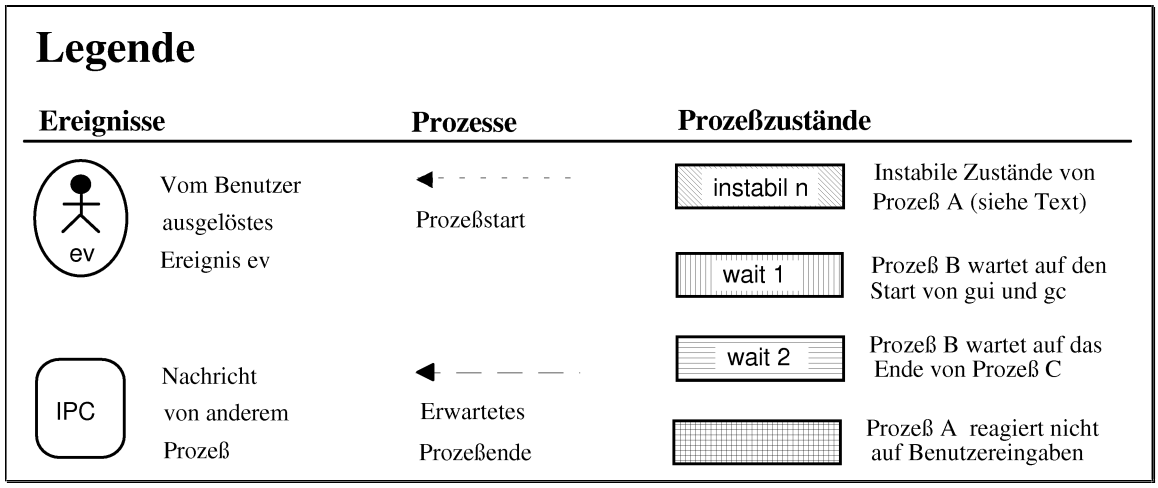


Abb. 5.2: Legende für die Abbildungen 5.3 und 5.4

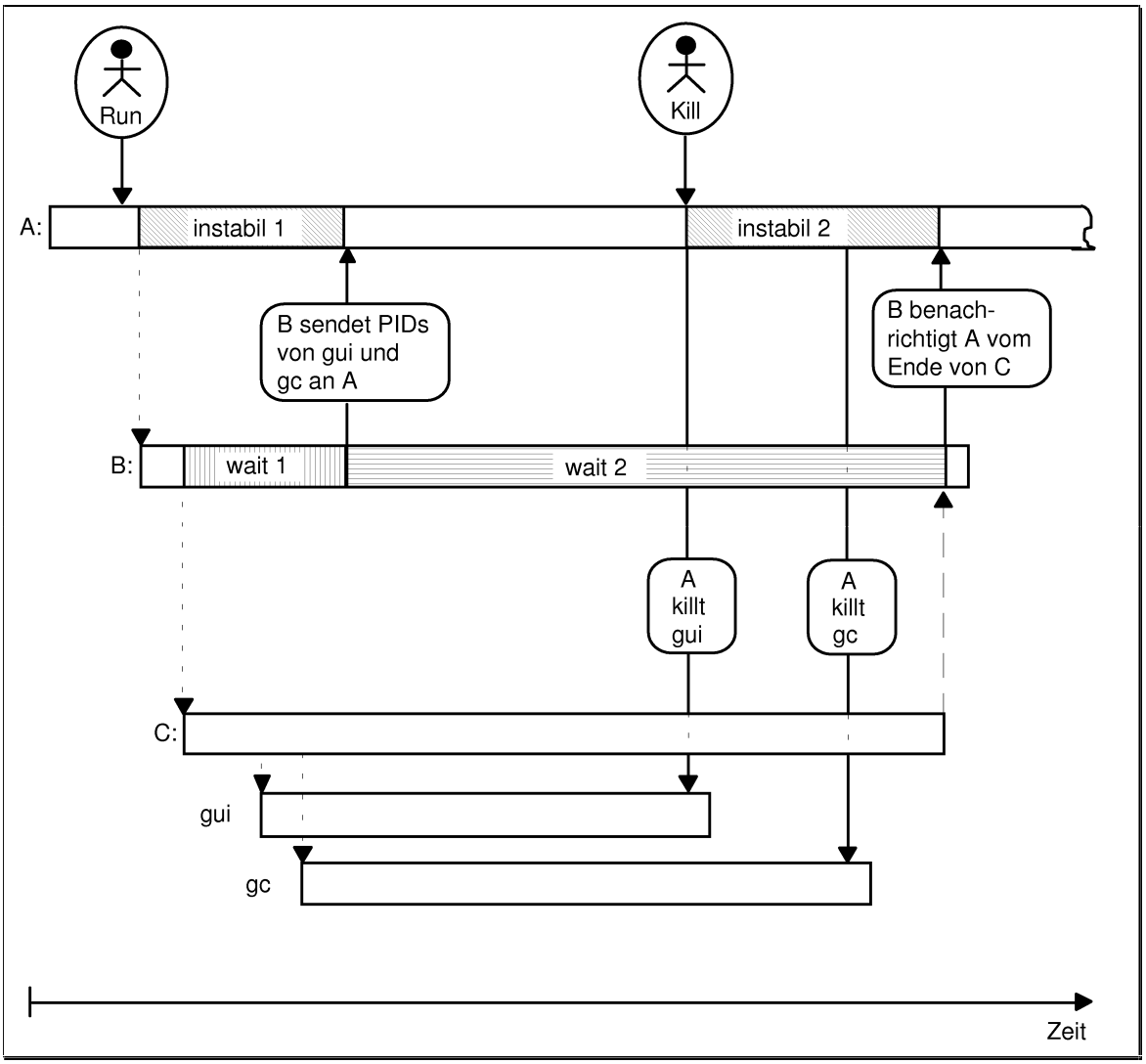


Abb. 5.3: Prozeßstruktur und Interprozeßkommunikation bei der Steuerung des Explorers aus der Visualisierungshilfe. Die Interprozeßkommunikation zwischen Teilprozessen des Explorers ist nicht dargestellt.

Die schraffiert dargestellten Zeitintervalle *"instabil 1"* und *"instabil 2"* von A stellen Problemzonen dar, weil in diesen Intervallen Prozesse gestartet oder beendet werden. Daher entsprechen die im Datenbereich von Prozeß A gespeicherten Prozeßidentifikatoren nicht zu jedem Zeitpunkt den laufenden Prozessen. Also müssen während dieser Zeitintervalle die Prozesse A und B synchronisiert werden.

Während das für *"instabil 1"* problemlos möglich ist (A wartet einfach auf ein Signal von B), sind für *"instabil 2"* weitere Überlegungen notwendig. Nach dem Senden Signals "Beenden" an *gui* kann nicht gewartet werden, da das "Beenden"-Signal weiterhin an *gc* zu senden ist. Wurde dieses gesendet, kann Prozeß A auf das Signal von B warten, das mitteilt, daß der Explorer beendet wurde. Es muß sichergestellt werden, daß Prozeß A während dieses Ablaufes keine weiteren Ereignisse vom Benutzer entgegennimmt und behandelt, da

sonst z.B. ein "Run"-Ereignis zum Start einer neuen Explorer-Instanz führen könnte, obwohl die alte noch nicht beendet ist.

Während des Zustandes *"wait 1"* von Prozeß B wartet dieser aktiv, bis zwei Prozesse *gui* und *gc* gefunden werden, die Kindprozesse des Prozesses C sind. Während des Zustandes *"wait 2"* wird B inaktiv, bis er das Signal "C ist terminiert" erhält.

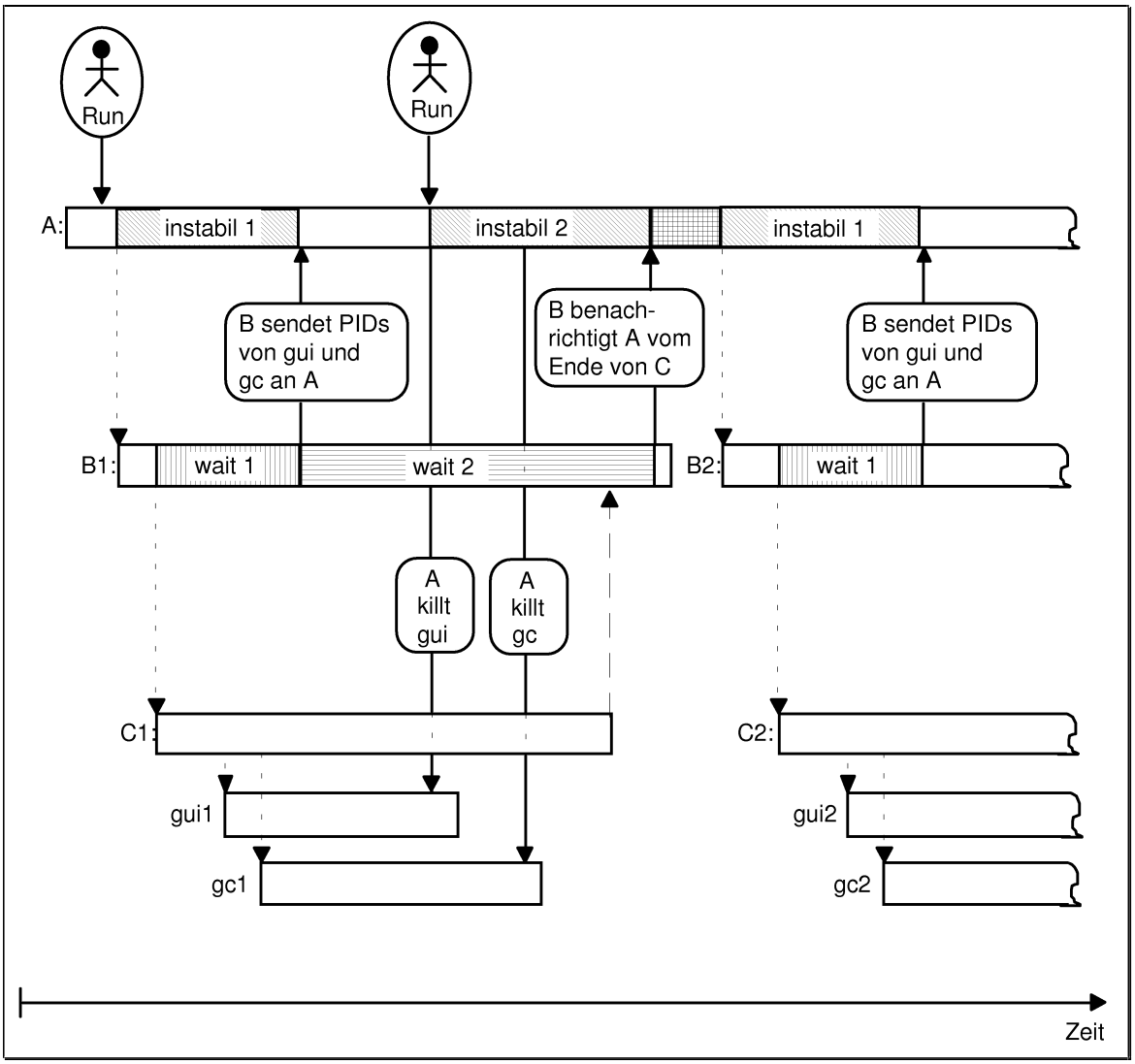


Abb. 5.4: Erneutes Auslösen von "Run" ohne vorheriges "Kill": Da noch eine Explorer-Instanz existiert, wird diese vor dem Start einer neuen Instanz beseitigt.

Abbildung 5.4 illustriert den Fall, daß beim Auslösen des Run-Kommandos bereits eine Explorer-Instanz aktiv ist. Sie wird in diesem Fall zunächst beendet und dann eine neue Instanz gestartet. Weiterhin muß beim Beenden der Visualisierungshilfe geprüft werden, ob noch eine Explorer-Instanz aktiv ist, und diese ggf. beendet werden.

5.4. Realisierung der Explorer-Steuerung

5.4.1. Realisierung von Synchronisation und Interprozeßkommunikation

Für die Interprozeßkommunikation werden zwei Konzepte benutzt: Pipes und Signale. Dabei dienen Pipes zum Datenaustausch und Signale zur Benachrichtigung von Prozessen über den Eintritt gewisser Ereignisse.

Eine Pipe unter UNIX ist ein Paar von File-Descriptoren, von denen einer zum Schreiben und einer zum Lesen geöffnet ist. Der sendende Prozeß schreibt seine Daten in den Write-Teil der Pipe, der empfangende Prozeß liest die Daten aus dem Read-Teil. Falls die Pipe leer ist, wartet der lesende Prozeß. Das läßt sich gleichzeitig zur Synchronisation von Prozessen verwenden.

Ein Signal unter UNIX ist eine vom Betriebssystem versandte Meldung, die einen Signal-Handler in dem Prozeß aufruft, an den das Signal gesandt wurde. Gleichzeitig beendet ein Signal den Wartezustand eines Prozesses, in den er durch den *wait*-Befehl versetzt wurde. In Verbindung mit dem *wait*-Befehl kann ein Signal also ebenfalls zur Synchronisation verwendet werden.

Wie können nun mit diesen Hilfsmitteln die Synchronisation und Kommunikation der drei Prozesse A, B und C realisiert werden? Für das Senden der PIDs von *gui* und *gc* bietet es sich an, eine Pipe zu benutzen. Durch das Lesen aus der Pipe wird auch das Warten des Prozesses A während des Intervalls *"instabil 1"* (siehe Abbildung 5.3) realisiert. Die Benachrichtigung des Prozesses A, daß der Explorer beendet wurde, erfolgt durch ein Signal. Dieses übernimmt gleichzeitig die Synchronisation während des Intervalls *"instabil 2"*: Nach dem Senden des *kill*-Signals an *gui* und *gc* wird A mittels des *wait*-Befehls in einen Wartezustand versetzt, den das Signal beendet.

Der Zustand *"wait 2"* des Prozesses B wird ebenfalls durch einen *wait*-Befehl eingeleitet und durch das vom Betriebssystem mit der Beendigung von Prozeß C automatisch generierte Signal *"death of child"* beendet. Auf den Zustand *"wait 1"* wird im Abschnitt 4.2. eingegangen.

In Abschnitt 3 wurde gefordert, daß während des Zustandes "*instabil 2"* von Prozeß A keine Ereignisse behandelt werden. Das läßt sich dadurch erreichen, daß die Befehle *"kill gui"*, *"kill gc"* und "Warte auf Nachricht von B" nacheinander aus derselben Ereignisroutine aufgerufen werden.

5.4.2. Realisierung des Auffindens der Prozesse gui und gc

Während der Phase *"wait 1"* des Prozesses B versucht dieser herauszufinden, wie die Prozeßidentifikatoren der Kindprozesse *gui* und *gc* des Prozesses C lauten. Da ein direkter Zugriff auf die Prozeßtabelle schwierig ist, kommt hier ein Kindprozeß B' von B zum Einsatz, der die Suche nach *gui* und *gc* übernimmt. Die Standardausgabe von B' wird über eine Pipe an B übergeben. B erzeugt jetzt zyklisch B', bis beide Prozesse gefunden werden konnten (polling).

Die Suche nach *gui* und *gc* wird dabei durch eine Shell-Pipe realisiert, die in der Ausgabe des UNIX-Kommandos *ps* nach dem PID von Prozeß C (als Elternprozeß) und den Strings *gui* und *gc* (als Prozeßnamen) sucht und die PIDs der entsprechenden Prozesse zurückgibt. Diese Variante ist relativ einfach, erfordert aber evtl. bei Portierung der Visualisierungshilfe auf ein anderes UNIX-System die Anpassung an das Ausgabeformat des dort installierten *ps*-Kommandos.

5.4.3. Schnittstellen

Die Komponente zur Explorer-Steuerung wird als Modul realisiert, das folgende Schnittstellen aufweist:

1. Explorer Start Callback: Eine Routine, die von der Visualisierungshilfe aufgerufen wird, um den Explorer zu starten. Als Parameter werden Name der Map und ein Flag, ob der Map-Editor ikonisiert werden soll, benötigt.
2. Explorer Kill Callback: Eine Routine, die von der Visualisierungshilfe aufgerufen wird, um den Explorer zu beenden. Es werden keine Parameter benötigt.
3. Kill Signal Handler: Eine Routine der Visualisierungshilfe, die über ein Signal aufgerufen wird, wenn der Explorer beendet ist. Diese übernimmt z.B. das Deaktivieren von Aktionsknöpfen, bzw. Menüeinträgen, die das Beenden des Explorers ermöglichen.

5.4.4. Aufrufparameter

Um ein Map-File automatisch beim Start zu laden, wird der Explorer von Prozeß C mit den folgenden Parametern gestartet:

explorer -map <mapname>.map

Das ist sowohl mit dem Explorer der Version 1.0 als auch mit Version 2.1 möglich. Mit Version 1.0 erzeugte Maps können problemlos mit der Version 2.1 dargestellt werden. Umgekehrt gilt das nicht.

Explorer 2.1 gestattet zusätzlich das Erstellen von script-Files. Dies ermöglicht eine automatische Map-Generierung[41]. Zum Anzeigen eines automatisch generierten Scripts wird der Explorer mit folgenden Parametern gestartet:

explorer -script <scriptname>.skm

5.5. Zusammenfassung

Im vorliegenden Kapitel wurde einen Möglichkeit vorgestellt, den Explorer aus der Visualisierungshilfe heraus zu starten und zu beenden. Da der Explorer als komplexes System vorliegt, mußten Behelfslösungen (wie die Suche nach den Explorer-Prozessen *gui* und *gc*) verwendet werden. Dadurch ist das System relativ langsam und anfällig. Eine elegante Lösung wäre eine Explorer-Bibliothek, die in eigene Programme eingebunden werden kann. Diese kann jedoch nur von Silicon Graphics, dem Hersteller des IRIS-Explorers, geliefert werden.

41 Für eine genauere Beschreibung siehe [DAMM94].

6. Dependency-Management

6.1. Problemstellung

Bei einem direktmanipulativen System ist es wichtig, daß Änderungen sofort visuelle Auswirkungen haben. Ziel ist es, daß der dem Nutzer in den offenen Fenstern sichtbare Ausschnitt aus der Gesamtattributmenge immer konsistent ist.

Ausgehend von der Menge der Attribute und den darin existierenden Abhängigkeiten soll im folgenden ein Mechanismus vorgeschlagen werden, der bei Änderung eines Attributes die davon abhängigen Attribute neu berechnet und diese geänderten Daten an der Benutzeroberfläche darstellt. Dabei wird aus Performancegründen Wert darauf gelegt, daß nur solche Attribute neu berechnet werden, die auch in gerade offenen Fenstern dargestellt werden.

Da der Prototyp der Visualisierungshilfe in der Technik des *Rapid Prototyping* in Tcl/Tk erstellt wurde, wird zusätzlich eine Art der Integration gefordert, die mit geringen Änderungen am Tcl-Code des Prototypen auskommt, die also quasi als eine Schicht unter die vom Prototypen benutzten Tcl-Variablen "untergeschoben" werden kann.

6.2. Attribute und Abhängigkeiten

6.2.1. Attribute zur Beschreibung eines Visualisierungsproblems

Die Beschreibung eines Visualisierungsproblems (vgl. Abschnitt 2.1.1) wird in netCDF als Menge von *Attributen* abgespeichert[42]. Dabei werden *globale Attribute* und *Merkmalsattribute* unterschieden. Globale Attribute beschreiben Charakteristika der gesamten Datenmenge (z.B. Anzahl der Beobachtungsfälle). Merkmalsattribute beschreiben Charakteristika eines Merkmals (z.B. Ordinalität). Während globale Attribute also genau einen Wert repräs-

[42] Siehe [REW93].

sentieren, kann man Merkmalsattribute als eine Liste von Werten auffassen, die je Merkmal ein Element enthält. Nach diesem Schema sind die netCDF-Attribute in einer globalen Tcl-Variablen gespeichert.

6.2.2. Abhängigkeiten zwischen den Attributen

Abhängigkeiten bestehen einerseits zwischen den Attributen, andererseits zwischen der Datenmenge und Attributen. Allgemein soll hier von *Aspekten* gesprochen werden: Ein Aspekt ist entweder die Datenmenge oder ein Attribut. Unter einer *Abhängigkeit* wird in diesem Abschnitt die Notwendigkeit verstanden, bei Änderung eines Aspektes einen abhängigen Aspekt neu zu berechnen.

Attribute enthalten alle zur Beschreibung des Visualisierungsproblems relevanten Informationen. Relevante Informationen, die in der Datenmenge selbst enthalten sind (z. B. Minimum, Maximum, Korrelationen usw.) werden durch Routinen zur *Datenanalyse* aus der Datenmenge extrahiert und in Attributen abgespeichert. Attribute können also auf verschiedene Art und Weise gewonnen werden (vgl. Anhang B):

a) essentielle Attribute
Diese können nicht im Rahmen einer Datenanlyse aus den Daten oder aus anderen Attributen erzeugt werden. Sie sind auch nicht im Rahmen der Visualisierungshilfe interaktiv modifizierbar, sondern müssen durch das Datendefinitions-Tool erzeugt werden.

b) berechenbar und abhängig von Benutzereingaben
Diese Attribute können aus anderen Aspekten berechnet werden. Sie ändern sich, wenn diese Aspekte interaktiv geändert werden.

c) berechenbar und unabhängig von Benutzereingaben
Diese Attribute hängen von Aspekten ab, die interaktiv nicht modifiziert werden.

d) interaktiv modifizierbar
Diese Attribute können interaktiv eingegeben werden. Beim Erzeugen des netCDF-Files durch das Datendefinitions-Tool müssen sie mit Default-Werten belegt werden.

Abhängigkeiten können in einem gerichteten kreisfreien Graphen (DAG[43]) dargestellt werden. Die Kreisfreiheit wird gefordert, um Endlosschleifen beim Durchlaufen des Abhängigkeitsgraphen zu vermeiden. In Abbildung 6.1 ist ein Ausschnitt aus dem Abhängigkeitsgraphen veranschaulicht. Für eine Darstellung aller Abhängigkeiten wird auf Anhang B verwiesen.

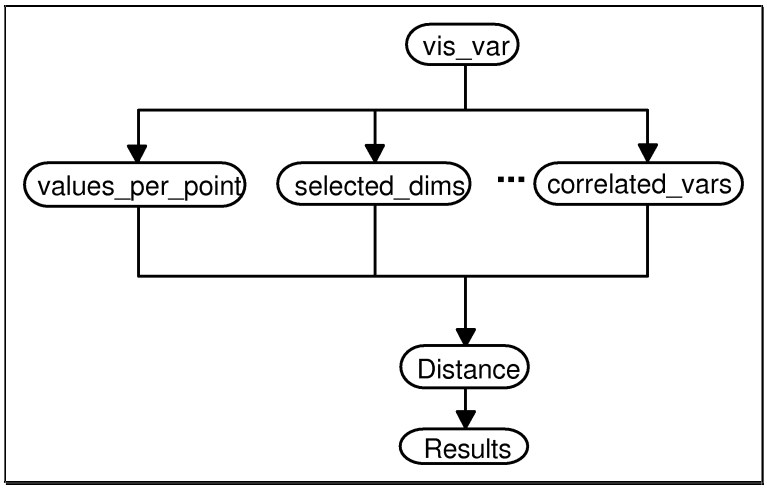


Abb. 6.1: Ausschnitt aus dem Abhängigkeits-DAG. Bedeutung der Attribute siehe Anhang B.

6.3. Ein dynamischer Ansatz zum Dependency-Management

6.3.1. Überblick

Das Dependency-Management muß sicherstellen, daß alle benötigten Attribute zu jedem Zeitpunkt in einem konsistenten Zustand sind. Attribute, die zum aktuellen Zeitpunkt nicht benötigt werden, müssen auch nicht konsistent sein. Das führt zu einer Einsparung an Rechenzeit.

Es muß somit ein Mechanismus gefunden werden, der den in Abschnitt 6.2.2 eingeführten Abhängigkeitsgraphen verwaltet und alle benötigten Attribute konsistent hält. Dieser muß weiterhin die drei verschiedenen Arten von Aspekten gleich behandeln.

6.3.2. Behandlung unterschiedlicher Aspekte

Der Aspekt *Datenmenge* wird durch Markierung einzelner Datensätze zur Visualisierung modifiziert. Die Routine, die die Markierung durchführt, kann gleichzeitig die Anzahl der markierten Datensätze feststellen und als globales Attribut abspeichern. Somit kann eine Änderung der Datenmenge auf die Änderung eines globalen Attributes zurückgeführt werden.

[43] DAG = Directed Acyclic Graph

Bei der Änderung eines Merkmalsattributes ist neben dem geänderten Attribut auch das Merkmal interessant, zu dem es gehört. Nur abhängige globale Attribute und abhängige Merkmalsattribute dieses Merkmals müßten dann modifiziert werden. Eine Analyse der Abhängigkeiten, in die Merkmalsattribute einbezogen sind, ergab jedoch, daß die notwendigen Berechnungen nicht umfangreich sind. Auch wird i.d.R. die Anzahl der Merkmale 100 nicht überschreiten. Somit kann auf eine gesonderte Behandlung der Merkmale verzichtet und das System weiter vereinfacht werden. Merkmalsattribute werden wie globale Attribute behandelt: bei einer Änderung eines Merkmalsattributes wird die gesamte abhängige Attributmenge neu berechnet. Hier tritt jedoch ein potentielles Problem auf: In Schleifen, die dasselbe Attribut mehrerer (oder u.U. aller) Merkmale modifizieren, würde bei jeder Änderung eines Merkmalsattributs eine Neuberechnung initiiert. Um das zu umgehen, kann vor Beginn der Schleife die Neuberechnung verboten und erst nach Verarbeitung aller Merkmale explizit gestartet werden.

6.3.3. Konsistenzdaten und Konsistenzroutinen

Der Dependency-Mechanismus muß folgende Zugriffe auf die Daten berücksichtigen:

a) Initialisierung
Hierbei wird ein konsistenter Zustand der Ausgangsdatenmenge aus den im netCDF-File gespeicherten Informationen hergestellt.

b) Schreibzugriffe
Erfolgt ein Schreibzugriff auf einen Aspekt der Datenmenge, so müssen alle abhängigen Aspekte als ungültig markiert werden. Gleichzeitig wird eine Nachricht an alle Fenster gesendet, die mindestens einen der als ungültig markierten bzw. geänderten Aspekte darstellen. Diese müssen dann ihren Inhalt durch Aufruf der *Redraw-Prozedur* neu darstellen.

c) Lesezugriffe
Bei einem Lesezugriff auf einen Aspekt wird geprüft, ob dieser als ungültig markiert ist. Ist das der Fall, erfolgt seine Neuberechnung, und er wird als gültig markiert. Lesezugriffe auf einen Aspekt werden i.d.R. von Routinen zum Neuzeichnen der Fensterinhalte oder zur Neuberechnung anderer Aspekte initiiert. Dieser Mechanismus stellt somit sicher, daß immer nur die Aspekte neu berechnet werden, die das System auch benötigt. Das Markieren als gültig nach der Neuberechnung stellt sicher, daß ein gültiger Aspekt beim nächsten Lesezugriff nicht nochmals berechnet wird.

d) Synchronisation

Bei der Synchronisation werden alle Aspekte der Datenmenge als Attribute im netCDF-File abgespeichert.

Die Architektur des Dependency- Mechanismus ist in Abbildung 6.2 dargestellt. Zum Verwalten oben genannter Zugriffe sind *Konsistenzdaten* und *Konsistenzroutinen* notwendig. Konsistenzdaten bestehen aus einer *Repräsentation des Abhängigkeitsgraphen* (sta-tisch) und aus einer Menge von *Dirty-Flags* (dynamisch). Im Dirty-Flag eines Aspektes wird gespeichert, ob dieser gültig ist oder nicht. Konsistenzroutinen realisieren die Zugriffsarten "Lesen" und "Schreiben". Die Routine "Init" ist für die Initialisierung der Datenbasis verantwortlich, und "WT" (Write Through) übernimmt das Zurückschreiben der Werte in netCDF-Attribute (Synchronisation). Die Konsistenzroutinen für das Lesen von Aspekten rufen die Daten-Neuberechnungsroutinen auf, falls notwendig. Eine Besonderheit ist hier, daß eine solche Berechnungsroutine mehrere Aspekte gleichzeitig berechnen kann.

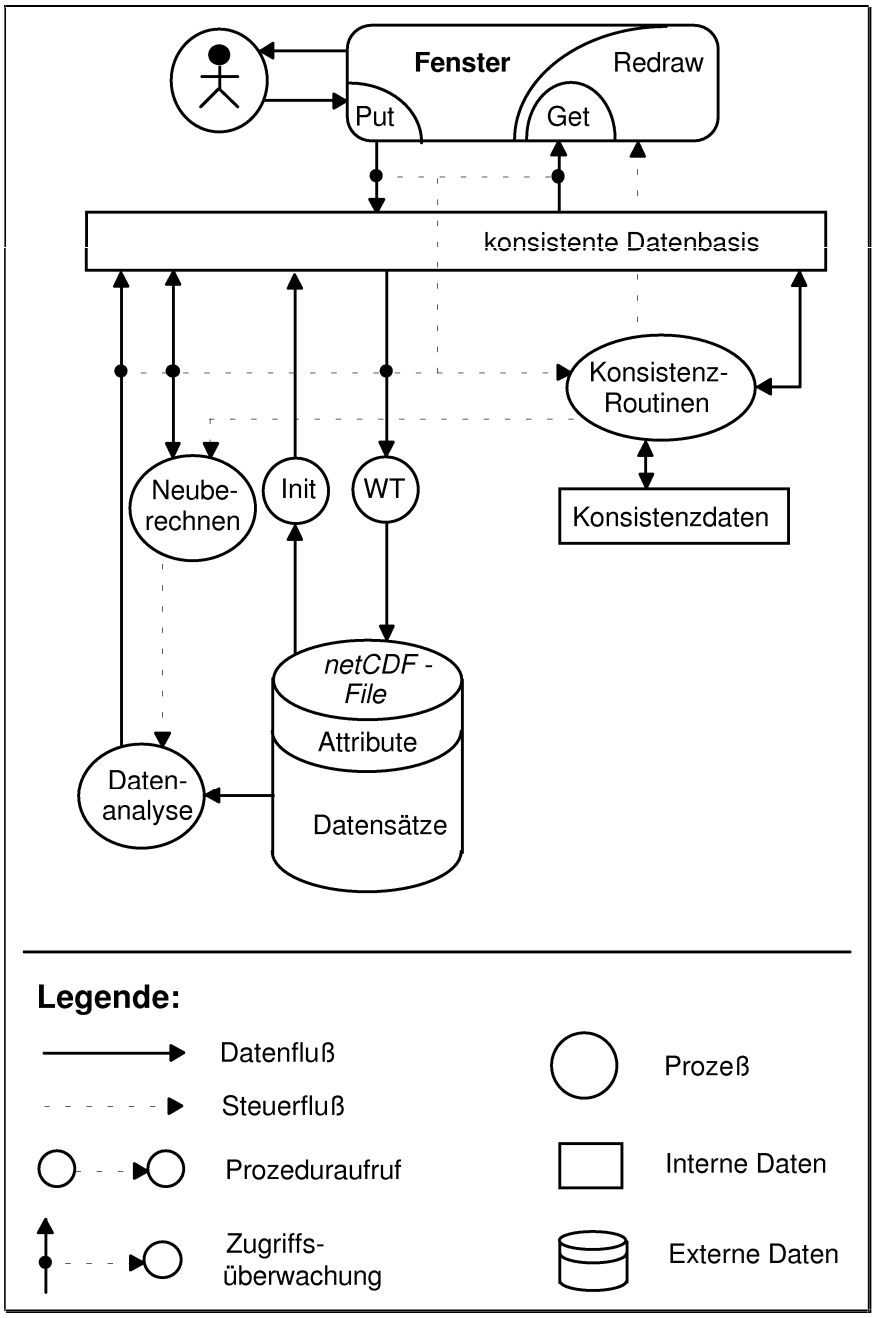


Abb. 6.2: Architektur des Dependency-Mechanismus

In einem solchen Fall muß die Berechnungsroutine die Dirty-Flags aller berechneten Aspekte zurücksetzen.

6.3.4. Verarbeitung des Abhängigkeitsgraphen

Die Verarbeitung des Abhängigkeitsgraphen spielt eine zentrale Rolle im Dependency-Mechanismus. Eine erste Idee zur Verarbeitung des Abhängigkeitsgraphen ist in Listing 6.1 dargestellt. Die Prozedur `SetzeDirty` wird aufgerufen, wenn ein Schreibzugriff auf einen Aspekt erfolgt.

```
PROCEDURE SetzeDirty(Aspekt1)
BEGIN
    FOREACH Aspekt2 ABHÄNGIG_VON Aspekt1 DO
        SET Dirty_Flag[Aspekt2] 1
        SetzeDirty(Aspekt2)
    DONE
    Benachrichtige_abhängige_Fenster(Aspekt1)
END
```

Listing 6.1: Pseudocode für das Durchlaufen des Abhängigkeitsgraphen

Bei Experimenten mit der Erstimplementation des Dependency-Mechanismus stellte sich jedoch heraus, daß dieser einfache Ansatz zu tieferen Rekursionen als notwendig, zu unnötigen Mehrfachberechnungen von Aspekten und im Extremfall zum Programmabbruch wegen entstehender Kreise im Graphen führte.

Die Lösung ist eine Verarbeitung in zwei Stufen: Zunächst wird der Abhängigkeitsgraph durchlaufen und alle in derselben Schachtelungstiefe befindlichen Aspekte in einer Liste für diese Tiefe notiert. Gleichzeitig können die Dirty-Flags gesetzt werden - bis auf das des Wurzelknotens. In der zweiten Stufe werden die Listen rückwärts ausgewertet und für jeden Aspekt von diesem abhängige Fenster an eine *Redraw-Liste* angehängt, falls sie in dieser noch nicht enthalten sind. Danach werden alle Fenster in der Redraw-Liste der Reihe nach benachrichtigt. Das heißt, die Fenster, deren dargestellte Aspekte von den meisten Aspekten abhängen, werden zuerst neu gezeichnet. Da das Neuzeichnen in geordneter Reihenfolge erfolgt, kann während des Neuzeichnens der Aufruf der Prozedur `SetzeDirty` verboten werden. Damit ist die Bildung von Kreisen und die Mehrfachberechnung von Aspekten ausgeschlossen.

Listing 6.2 zeigt die verbesserte Variante.

```
GLOBAL DAG2Level

PROCEDURE DoSetDirty(Aspekt1, Level)
BEGIN
    IF DAG2Level.max < Level THEN
    BEGIN
        DAG2Level.max:=Level
        DAG2Level.list[Level]:=EMPTY
    END
    FOREACH Aspekt2 ABHÄNGIG_VON Aspekt1 DO
        SET Dirty_Flag[Aspekt2] 1
        IF NOT (Aspekt2 IN DAG2Level.list[Level]
        THEN APPEND Aspekt2 TO DAG2Level.list[Level]
        DoSetDirty(Aspekt2, Level+1)
    DONE
END

PROCEDURE CallRedraw
    RedrawList:=EMPTY
    FOR DAG2Level.max DOWNTO 0 DO
        FOREACH Aspekt IN DAG2Level.list[i] DO
             FOREACH Window ABHÄNGIG_VON Aspekt DO
               IF NOT (Window IN RedrawList)
               THEN APPEND Window TO RedrawList
            DONE
        DONE
    DONE
    FOR i:=0 TO LENGTH(RedrawList) DO
        Window:=LISTENTRY(RedrawList, i)
        Benachrichtige_Fenster(Window)
    DONE
END

PROCEDURE SetzeDirty(Aspekt)
BEGIN
    IF NOT VERBOTEN(Setze_Dirty) THEN
        VERBIETE(SetzeDirty)
        DAG2Level.max:=0
        DAG2Level.liste[0]:=Aspekt
        SET Dirty_Flag(Aspekt) 0
        DoSetDirty(Aspekt, 1)
        CallRedraw
        ERLAUBE(SetzeDirty)
    END
END
```

Listing 6.2: Pseudocode für das zweistufige Durchlaufen des Abhängigkeitsgraphen

6.4. Implementation des Dependency-Mechanismus

Der Dependency-Mechanismus wurde zum Teil in Tcl/Tk und zum Teil in C implementiert. C-Routinen werden zur Datenanalyse, zum Initialisieren und Synchronisieren eingesetzt. Konsistenzdaten und Konsistenzroutinen wurden in Tcl implementiert. Das Listing ist in Anhang E abgedruckt. Neuberechnung von Aspekten wird teilweise von Tcl-, teilweise von C-Routinen übernommen.

Fenster wurden als Tk-Widgets mit assoziierten Redraw-Routinen in Tcl implementiert.

Zur Zugriffsüberwachung wurden Tcl-Traces eingesetzt. Ein Trace ist ein Tcl-Kommando, das vom Tcl-Interpreter bei Variablenzugriffen aufgerufen wird. Write-Traces rufen die oben diskutierten Routinen zum Setzen der Dirty-Flags und zum Neuzeichnen von Widgets auf. Read-Traces starten eine Neuberechnung des gelesenen Aspektes, falls dessen Dirty-Flag gesetzt ist. Durch die Traces ist eine weitgehende Transparenz der Neuberechnung von Aspekten erreichbar. Erst beim Lesezugriff auf eine Tcl-Variable wird entschieden, ob diese neu zu berechnen ist oder nicht. Damit ist ein Weiternutzen des ursprünglichen Codes des Prototypen mit geringfügigen Änderungen gewährleistet.

7. Schlußbetrachtungen

In der vorliegenden Arbeit wurde ein interaktives Werkzeug vorgestellt, das den Benutzer bei der Auswahl geeigneter Visualisierungstechniken unterstützt. Es wurde auf der Grundlage des von Lukoschek vorgestellten Expressivitätskriteriums[44] erstellt. Zur Zeit wird an einer Einbindung des Effektivitätskriteriums[45] gearbeitet.

Das Werkzeug ermöglicht die interaktive Spezifizierung der Komponenten eines Visualisierungsproblems, die Anzeige von Widersprüchen und den Start des Visualisierungsproblems IRIS-Explorer mit einer geeigneten Visualisierungstechnik. Der Benutzer wird durch eine direktmanipulative, fensterorientierte Oberfläche mit automatischem Dependency-Management, Erklärungs- und Hilfefunktion unterstützt.

Das größte Problem des Nutzers bei der Arbeit mit der Visualisierungshilfe ist nach Meinung des Autors die Spezifizierung der Interpretationsziele. Es scheint relative schwierig zu sein, ein konkretes Interpretationsziel in eine gewichtete Kombination abstrakter Ziele zu transformieren. Das in [THEI94] entwickelte mathematische System von Interpretationszielen ist noch bedeutend schwerer zu handhaben als das in der gegenwärtigen Version der Visualisierungshilfe implementierte heuristisch ermittelte. Auf diesem Gebiet einerseits und bei der Wahl der für die Eignungsberechnung eingesetzten Funktionen andererseits liegen die größten Schwachstellen der Visualisierungshilfe, hier besteht Bedarf für weitere Forschungen. Auch eine Erweiterung der Schnittstelle zum IRIS-Explorer um weitere Visualisierungssysteme oder an eine Erweiterung der Datenklasse über die multivariaten Daten hinaus wären lohnenswerte Forschungsgebiete.

Für eine Auflistung von Einschränkungen des gegenwärtigen Systems wird auf Anhang C verwiesen.

---

[44] Siehe [LUKO93b].
[45] Siehe [LUKO93a].

Literaturverzeichnis

[ARND92] S. Arndt: *"Visualisierung mikrobiologischer Daten"*, Diplomarbeit, Universität Rostock, Fachbereich Informatik, 1992.

[ARND93a] S. Arndt: *"Ansätze zur systematischen Auswahl von visuellen Repräsentationen bzw. Techniken zu ihrer Erzeugung"*, ICG-Bericht 2/1993, Universität Rostock, Fachbereich Informatik, Institut für Computergrafik, 1993.

[ARND93b] S. Arndt: *"Spezifizierung multivariater Daten zum Zwecke einer systematischen Auswahl ihrer visuellen Repräsentation"*, ICG-Bericht 4/1993, Universität Rostock, Fachbereich Informatik, Institut für Computergrafik, 1993.

[BROD92] K. W. Brodlie: *"Scientific Visualisation: techniques and applications"*, Springer, Berlin 1992.

[DAMM94] V. Damm: *"Automatische Map-Erzeugung" (vorläufiger Arbeitstitel)*, Studienarbeit, Universität Rostock, Fachbereich Informatik, z.Zt. in Bearbeitung, voraussichtlich 1994.

[KAIE93] T. Kaie: *"Visualisierung mikrobiologischer Daten - Entwichklung eines interaktiven Werkzeugs zur Markierung von Datensätzen"*, Projektpraktikum, Universität Rostock, Fachbereich Informatik, 1993.

[KAIE94] T. Kaie: *"Entwurf und Implementation von Werkzeugen zur Datenspezifikation" (vorläufiger Arbeitstitel)*, Studienarbeit, Universität Rostock, Fachbereich Informatik, z.Zt. in Bearbeitung, voraussichtlich 1994.

[LUKO93a] K. Lukoschek: *"Kriterien zur Wirksamkeit visueller Repräsentationen wissenschaftlicher Daten"*, ICG-Bericht 1/1993, Universität Rostock, Fachbereich Informatik, Institut für Computergrafik, 1993.

[LUKO93b] K. Lukoschek: *"Die Expressivität als ein Kriterium für die Wirksamkeit visueller Repräsentationen"*, Preprint CS-07-93, Universität Rostock, Fachbereich Informatik, 1993.

[MAH93] A. Mahler und S. Delmas: *"Doing More With Shape Tools"*, STONE System Documentation, Technische Universität Berlin, 1993.

[OUST93] J. K. Ousterhout: *"Tcl and the Tk Toolkit"*, Draft 8/12/93, Addison Wesley Publishing Company, 1993.

[REW93] R. Rew et al.: *"NetCDF User's Guide"*, Version 2.3, University Corporation for Atmospheric Research, 1993.

[SCHU93] H.Schumann: *"Entwurf einer Visualisierungshilfe für die Darstellung wissenschaftlich-technischer Daten"*, ICG-Bericht 3/93, Universität Rostock, Fachbereich Informatik, Institut für Computergrafik, 1993.

[SCHW93] M. Schwenck: *"Visualisierung von Volumendaten"*, Diplomarbeit, Universität Rostock, Fachbereich Informatik, 1993.

[THEI92] H. Theisel: *"Analyse mehrdimensionaler Daten mit Hilfe der Informationstheorie"*, Rostocker Informatik-Berichte, Heft 14, Universität Rostock, Fachbereich Informatik, 1992.

[THEI93] H. Theisel: *"Erweiterungen und Anwendungen eines informationstheoretischen Ansatzes zur Analyse mehrdimensionaler Daten"*, Studienarbeit, Universität Rostock, Fachbereich Informatik, 1993.

[THEI94] H. Theisel: *"Automatische Auswahl geeigneter Visualisierungstechniken für allgemeine wissenschaftliche Datensätze"*, Diplomarbeit, Universität Rostock, Fachbereich Informatik, 1994.

[THIE94] A. Thiessen: *"Berechnung von Widersprüchen zwischen der Eignung von Visualisierungstechniken und Parametern der spezifizierten Datenmenge" (vorläufiger Arbeitstitel)*, Studienarbeit, Universität Rostock, Fachbereich Informatik, z.Zt. in Bearbeitung, voraussichtlich 1994.

Anhangsverzeichnis

Anhang A:
Benutzerhandbuch der Visualisierungshilfe

1. Kurzbeschreibung

Bei der Visualisierungshilfe handelt es sich um ein Programm, das in der Lage ist, multivariate Daten einzulesen und im Dialog mit dem Benutzer eine für die Visualisierung dieser Datenmenge geeignete Visualisierungstechnik zu ermitteln.

Unter multivariaten Daten werden in diesem Zusammenhang Daten verstanden, die über einem Meßgitter vorliegen und in Tabellenform (vgl. Tabelle 1) darstellbar sind. Solche Daten fallen bei vielen wissenschaftlichen Experimenten an. Die Spalten der Tabelle (oft als Variablen bezeichnet) werden in diesem Handbuch ***Merkmale*** genannt. Das Meßgitter wird durch sogenannte ***Dimensionen*** beschrieben (Zeit in Tabelle 1). Man unterscheidet ***ordinale*** und ***nominale*** Merkmale. Während bei ersteren (z.B. Temperatur) eine Ordnungsrelation zwischen den einzelnen Werten besteht, existiert diese bei letzteren (z.B. Farbe) nicht.

Zeit [min]	Farbe	Temperatur [°K]
0	grau	300
1	rot	1000
2	weiß	2000

Tabelle 1: Beispiel einer (einfachen) Meßwerttabelle bei einem Aufheizversuch

Die Visualisierungshilfe ermittelt wichtige Parameter dieser Datenmenge und erlaubt deren interaktive Einschränkung. Damit wird der ***Gegenstand der Visualisierung*** beschrieben. Das ***Ziel der Visualisierung*** beinhaltet den Zweck, den der Benutzer mit der grafischen Darstellung der Datenmenge verfolgt. Dieser wird in Form von Interpretationszielen interaktiv spezifiziert. Aus dem durch Gegenstand und Ziel der Visualisierung beschriebenen ***Visualisierungsproblem*** ermittelt das System eine Menge geeigneter Visualisierungstechniken, die dem Benutzer zur Auswahl angeboten werden.

Eine Online-Kopplung zum Visualisierungssystem IRIS-Explorer erlaubt die Visualisierung der Daten mit der als geeignet ermittelten Technik direkt aus der Visualisierungshilfe heraus[1]. Ein Hilfesystem und die Möglichkeit, Erklärungstexte und Visualisierungsbeispiele[2] anzuzeigen, unterstützen den Benutzer.

2. Systemanforderungen, Installation und Start

Benötigt wird ein Rechner mit einem UNIX-Betriebssystem und dem X Window System. An Softwarebibliotheken sind netCDF Version 2.3, Tcl Version 7.3 und Tk Version 3.6 notwendig. Diese sind vor der Übersetzung der Visualisierungshilfe zu installieren.

Im Makefile der Visualisierungshilfe sind dann entsprechend den dort eingefügten Kommentaren einige Pfade zu setzen und in der Variablen EXPLORER festzulegen, welche Version des IRIS-Explorers eingebunden werden soll. Das System wurde mit den Versionen 1.0 (nur Beispielanzeige) und 2.1 (Beispielanzeige und Visualisierung) getestet. Falls kein IRIS-Explorer verfügbar ist, kann die Alternative "EXPLORER=None" im Makefile gewählt werden, die die Online-Kopplung deaktiviert. Falls eine andere Version als 1.0 oder 2.1 vorhanden ist, gilt folgende Regel: Wenn die Version die Option *-script* unterstützt, ist die Online-Kopplung der Version 2.1 einzubinden, ansonsten die der Version 1.0. Eine Garantie für die Lauffähigkeit kann hier jedoch nicht übernommen werden.

Nach Konfigurieren des Makefiles erfolgt durch Eingabe von *make* das Übersetzen und durch Eingabe von *make install* die Installation des Programms. Weitere Informationen zur Installation sind in der Datei README und im Makefile enthalten.

Die Visualisierungshilfe wird durch Eingabe von *vh* gestartet. Als optionaler Parameter kann der Name der zu ladenden netCDF-Datei angegeben werden.

---

1 Noch nicht vollständig implementiert.
2 Beispiele noch nicht implementiert. Erklärungstexte noch nicht vollständig implementiert.

3. Bedienung

3.1. Kurzbeschreibung der Benutzeroberfläche

Die Benutzeroberfläche der Visualisierungshilfe ist direktmanipulativ und fensterorientiert. Jedes Fenster stellt einen Ausschnitt aus dem Visualisierungsproblem dar, der gewisse Teilaspekte des Visualisierungsproblems anzeigt oder deren interaktive Modifikation gestattet. Die Konsistenz der Information zwischen den einzelnen Fenstern wird automatisch gewährleistet, d.h., nach jeder Änderung in einem der Fenster wird der Inhalt aller von dieser Änderung abhängigen Fenster automatisch neu berechnet. Da das bei großen Datenmengen und vielen offenen Fenstern eine Weile dauern kann, ist dieser Mechanismus vom Benutzer über den Checkbutton *Immer aktualisieren* im Menüfenster (siehe Abschnitt 3.4) abschaltbar. Bei abgeschalteter automatischer Aktualisierung kann der Benutzer über den Aktionsknopf *Aktualisieren* im Menüfenster die Aktualisierung erzwingen.

Eingabefenster, die über einen Aktionsknopf *Aktualisieren* verfügen, geben eine Änderung nicht automatisch weiter, sondern erfordern dafür das Betätigen dieses Aktionsknopfes.

3.2. Ablauf einer Sitzung mit der Visualisierungshilfe

Der folgende Abschnitt beschreibt kurz die Schritte, aus denen normalerweise eine Sitzung mit der Visualisierungshilfe besteht. Sie ist als Anregung für den unerfahrenen Nutzer gedacht. Das Konzept der Benutzeroberfläche gestattet große Flexibilität in der Reihenfolge der Schritte und somit eine interaktive Erkundung von Alternativen, da die Auswirkungen von Änderungen unmittelbar sichtbar werden.

Für eine detaillierte Beschreibung der einzelnen Schritte wird auf die nachfolgenden Abschnitte verwiesen.

1. NetCDF-Datei laden, die den Gegenstand der Visualisierung enthält

Dazu wird über das Menü *Datei/Laden* eine Dateiauswahlbox geöffnet und dort die zu ladende Datei angeklickt.

2. Gegenstand der Visualisierung spezifizieren

Dazu existieren vier Menüeinträge im Menü *Visualisierungsproblem*.

Anklicken des Eintrages *Merkmalsauswahl* bringt ein Fenster auf den Bildschirm, das es ermöglicht, Merkmale auszuwählen (das entspricht der Auswahl von Spalten in der Meßwerttabelle).

Der Eintrag *Datenauswahl* öffnet ein Fenster, das die Auswahl von Datensätzen (Zeilen in der Meßwerttabelle) gestattet. Die Zeilen werden über logische Ausdrücke ausgewählt.

Durch Aktivieren des Eintrages *Korrelationen* wird ein Fenster geöffnet, das mögliche Korrelationen zwischen Merkmalen anzeigt und so bei der Merkmalsauswahl hilft.

Hinter dem Eintrag *Intervalle* verbirgt sich die Möglichkeit, die Werte ordinaler Variablen in Intervalle einzuteilen. Bei der Visualisierung werden alle Werte innerhalb eines Intervalls wie ein Wert behandelt, also z.B. in derselben Farbe dargestellt.

3. Ziel der Visualisierung spezifizieren

Das geschieht im Fenster *Interpretationsziele*, das über den gleichnamigen Menüeintrag im Menü *Visualisierungsproblem* erreichbar ist.

4. Empfehlung ansehen bzw. Konflikte lösen

Um das Ergebnis- oder das Konfliktfenster zu öffnen, werden die Menüeinträge *Techniken/Empfehlung* bzw. *Techniken/Konflikte* angeklickt. Beim Öffnen des Fensters *Konflikte* wird das Fenster *Interpretationsziele* automatisch geschlossen.
Wenn eine oder mehrere geeignete Techniken bzw. Technikkombinationen gefunden wurden, kann jetzt die Visualisierung durch Auswahl einer Technik im Empfehlungsfenster und anschließendes Betätigen des Aktionsknopfes *Visualisierung...* gestartet werden.

Falls keine geeignete Technik existiert, ist das Visualisierungsproblem zu umfangreich. Dann müssen entweder die Datenmenge oder die Interpretationsziele eingeschränkt werden. Zur Hilfestellung kann das Fenster *Konflikte* geöffnet werden, das anzeigt, welche Kandidaten für eine Einschränkung die aussichtsreichsten sind.

3.3. Beschreibung generischer Aktionsknöpfe

Im folgenden werden die Aktionsknöpfe und ihre Funktion beschrieben, die in allen oder in mehreren Fenstern enthalten sind.

OK
Schließt ein Fenster, in dem Änderungen möglich sind. Macht Änderungen dadurch permanent.

Schließen
Schließt ein Fenster, in dem keine Änderungen möglich sind.

Abbrechen
Nimmt alle Änderungen seit dem letzten Öffnen des Fensters zurück und schließt das Fenster.

Aktualisieren
Gibt die in einem Fenster vorgenommen Änderungen an das System weiter. Das wird angewendet, wenn die Änderungen entweder aufwendige Berechnungen nach sich ziehen oder aus mehreren Interaktionsschritten bestehen (z.B. Eingabe einer Zahl).

Widerrufen
Nimmt alle Änderungen seit dem letzten Öffnen des Fensters zurück.

Hilfe
Zeigt Hilfeinformation zum Fenster an.

3.4. Beschreibung der einzelnen Fenster

Fenster *Hauptfenster*

Aktivierung: Programmstart

Funktion: "Steuerzentrale"

Beschreibung: Im Hauptfenster können über das Menü alle Aktionen der Visualisierungshilfe ausgelöst und der Programmstatus abgelesen werden.

Über den Aktionsknopf *Explorer beenden* kann der eingebundene IRIS-Explorer beendet werden.

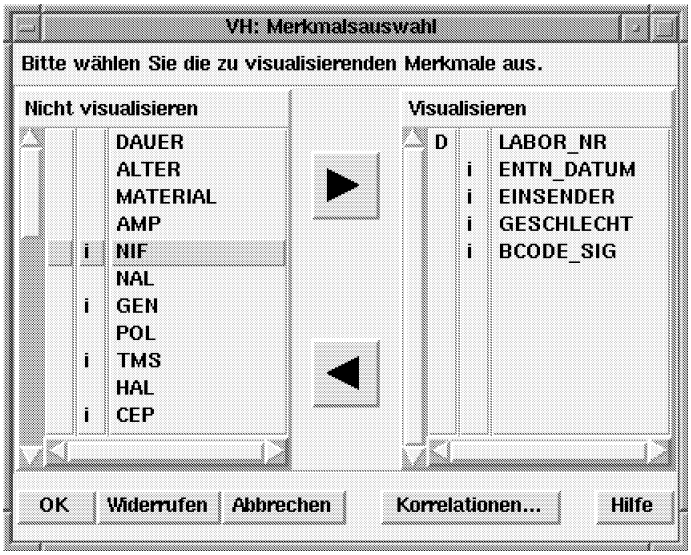
Der Aktionsknopf *Aktualisieren* erzwingt eine Aktualisierung aller offenen Fenster. Er übernimmt jedoch nicht die Aktualisierung in Fenstern, die selbst über einen Aktionsknopf *Aktualisieren* verfügen. Mit dem Checkbutton *Immer aktualisieren* kann die automatische Aktualisierung abgeschaltet werden.

Fenster *Dateiauswahlbox*

Aktivierung: Menü *Datei/Laden* bzw. Menü *Datei/Speichern*

Funktion: Auswahl der zu ladenden oder zu speichernden Datei

Fenster *Merkmalsauswahl*



Aktivierung: Menü *Visualisierungsproblem/Merkmalsauswahl*

Aktualisieren: automatisch

Funktion: Auswählen von zu visualisierenden Merkmalen

Beschreibung: Durch Anklicken mit der Maus kann in einer der beiden Listen ein Merkmal ausgewählt werden, durch Ziehen mit der Maus auch mehrere. Betätigen der mit den Pfeilen markierten Aktionsknöpfe verschiebt das Merkmal in die durch den Pfeil angedeutete Richtung. Betätigen des Aktionknopfes *Korrelationen* öffnet das Korrelationsfenster, das mögliche Korrelationen zwischen den Merkmalen anzeigt. Wenn der Benutzer an Korrelationen interessiert ist, kann er hier interessante Merkmalskombinationen ablesen und sie dann im Fenster *Merkmalsauswahl* zur Visualisierung auswählen.

Die "i's" in der zweiten Spalte jeder Liste sagen aus, daß in den Werten des jeweiligen Merkmals relevante Information enthalten sein kann, es sich also lohnt, dieses Merkmal zu visualisieren. Das ist jedoch kein Dogma. Ein "D" in der ersten Spalte sagt aus, daß es sich bei dem entsprechenden Merkmal um eine Dimension (also eine Variable mit Zeit- oder Raumbezug bzw. einen Index zur Identifikation der Beobachtungsfälle) handelt.

Fenster *Korrelationen*

Aktivierung: Menü *Visualisierungsproblem/Korrelationen*

Aktualisieren: nicht

Funktion: Ablesen möglicher Korrelationen

Beschreibung: Durch Anklicken eines Merkmals in der linken Liste werden in der rechten Liste alle Merkmale dargestellt, die mit diesem möglicherweise korrelieren. Ein "k" bedeutet, daß zu dem entsprechenden Merkmal Korrelationen existieren, ein "v", daß das entsprechende Merkmal zur Visualisierung ausgewählt wurde (vgl. Fenster *Merkmalsauswahl*).

Weiterhin wird die Länge des größten Tupels korrelierender Merkmale angezeigt.

Fenster *Datenauswahl*

Hier wird auf die Studienarbeit von Tom Kaie[3] verwiesen.

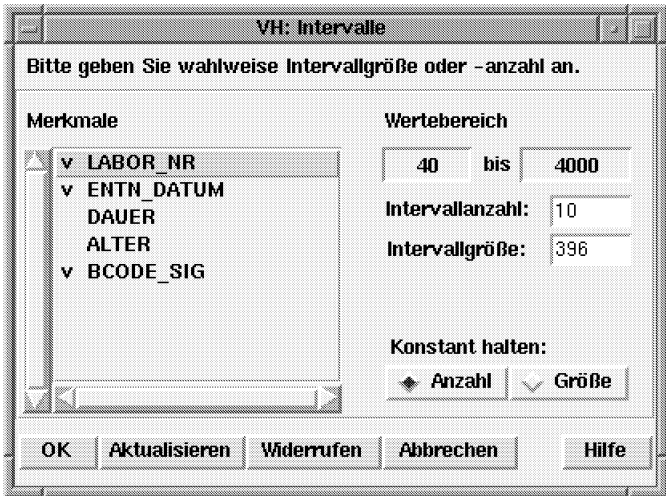
Fenster *Nutzer-Optionen*

Hier wird auf Arbeiten von Kai Lukoschek[4] verwiesen.

[3] T. Kaie: *"Entwurf und Implementation von Werkzeugen zur Datenspezifikation" (vorläufiger Arbeitstitel)*, Studienarbeit, Universität Rostock, Fachbereich Informatik, z.Zt. in Bearbeitung, voraussichtlich 1994.

[4] K. Lukoschek: *"Implementation des Effektivitätskriteriums" (vorläufiger Arbeitstitel)*, Universität Rostock, Fachbereich Informatik, z.Zt. in Bearbeitung, voraussichtlich 1994.

Fenster *Intervalle*



Aktivierung: Menü *Visualisierungsproblem/Intervalle*

Aktualisieren: Aktionsknopf

Funktion: Einteilung des Wertebereiches ordinaler Variablen in Intervalle[5]

Beschreibung: Nach dem Anklicken eines Merkmalsnamens kann entweder die Größe oder die Anzahl der Intervalle geändert werden, in die der Wertebereich des entsprechenden Merkmals eingeteilt werden soll. Der jeweils andere Aspekt wird dann automatisch neu berechnet (z.B. bei Änderung der Intervallanzahl die Intervallgröße). Unterschiedliche Werte im selben Intervall werden bei der Visualisierung wie ein Wert betrachtet, also gleich dargestellt.

Die Radiobuttons *Konstant halten* erlauben die Auswahl des Aspekts (Intervallgröße oder Intervallanzahl), der bei einer Änderung der zu visualisierenden Datenmenge im Fenster *Datenauswahl* konstant bleiben soll.

5 Die vorliegende Version unterstützt nur ganzzahlige Werte.

Fenster *Interpretationsziele*

Aktivierung: Menü *Visualisierungsproblem/Interpretationsziele*

Aktualisieren: automatisch

Funktion: Festlegen der Wichtung der Interpretationsziele

Beschreibung: Durch Anklicken der Radiobuttons lassen sich für die einzelnen Interpretationsziele Wichtungen vergeben. Diese werden verbal in den Feldern am rechten Fensterrand dargestellt. Aktivieren des Aktionsknopfes "Hilfe" öffnet das Hilfefenster, das u.a. Erklärungen zu den einzelnen Interpretationszielen enthält.

Beim Öffnen des Fensters *Interpretationsziele* wird ein etwa offenes Fenster *Konflikte* automatisch geschlossen, da eine Spezifikation der Interpretationsziele auch im Fenster *Interpretationsziele* möglich ist.

Fenster *Konflikte*

Aktivierung: Menü *Techniken/Konflikte*

Aktualisieren: automatisch

Funktion: Ablesen von Konflikten zwischen beschriebenem Visualisierungsproblem und Eignung der Techniken

Beschreibung: Widersprüche zwischen beschriebenem Visualisierungsproblem und Eignung der Techniken werden grafisch dargestellt. Die schwarzen Balken neben den Interpretationszielen stellen die Größe der Widersprüche dar. Die Interpretationsziele mit dem längsten Balken sollten somit zuerst eingeschränkt werden. Die notwendige Reduktion der Datenmenge wird einmal als Zahl (absoluter Wert) und andererseits als Grafik (relativer Wert) dargestellt. Die Zahl -4 in der Spalte "Data Jack" in obiger Abbildung bedeutet also, daß für eine Eignung der Technik "Data Jack" die Zahl der ausgewählten Merkmale um 4 reduziert

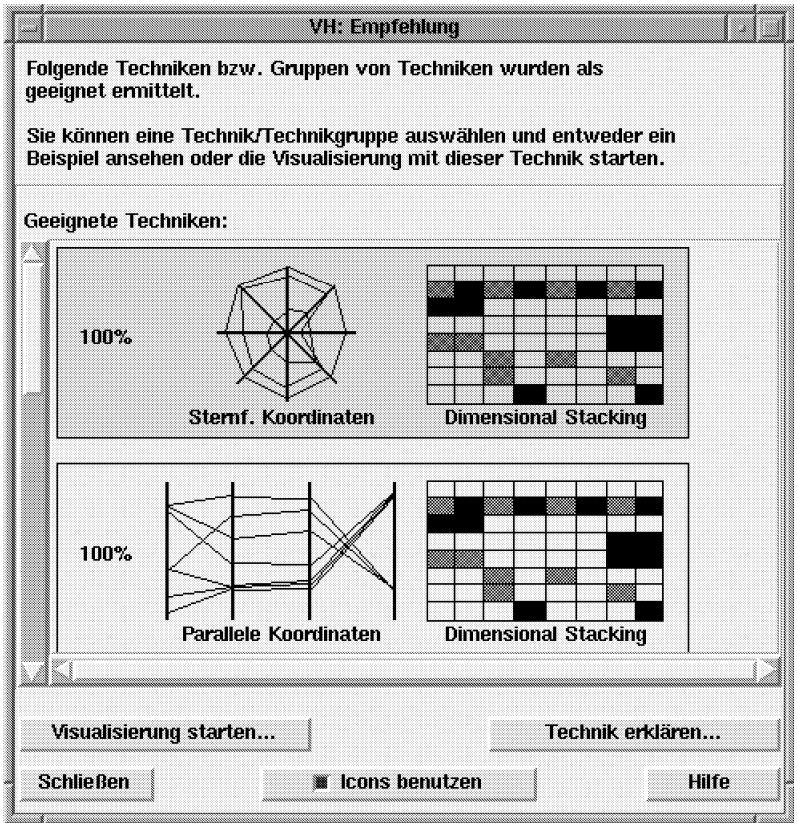
werden muß. Das sind relativ gesehen etwa vier Fünftel, wie sich aus der Länge des grauen Balkens ergibt. Dieser gibt an, wie hoch die Anzahl der Merkmale (bzw. analog der Beobachtungsfälle) für eine Eignung der Technik maximal sein darf. Ein leeres Rechteck bedeutet, daß eine Einschränkung der Datenmenge keinen Einfluß auf die Eignung hat.

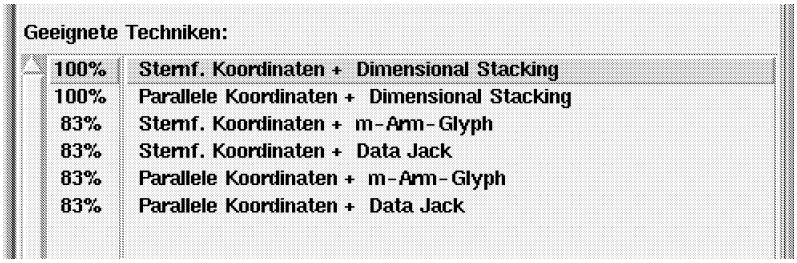
Im unteren Teil wird die prozentuale Eignung der Einzeltechniken für das Visualisierungsproblem durch einen hellblauen Balken angezeigt. Die senkrechte Linie im den Balken umgebenden Rahmen repräsentiert den Schwellwert, ab dem eine Technik als geeignet angenommen wird.

Wenn Konflikte auftreten, heißt das nicht, daß das System keine geeigneten Techniken findet. Falls keine Einzeltechnik geeignet ist, versucht es nämlich, Techniken zu kombinieren.

Beim Öffnen des Fensters *Konflikte* wird ein etwa offenes Fenster *Interpretationsziele* automatisch geschlossen, da eine Spezifikation der Interpretationsziele auch im Fenster *Konflikte* möglich ist.

Fenster *Empfehlung*





Aktivierung: Menü Techniken/Empfehlung

Aktualisieren: nicht

Funktion: Darstellung geeigneter Techniken, Auswahl einer Technik(kombination), Start der Visualisierung

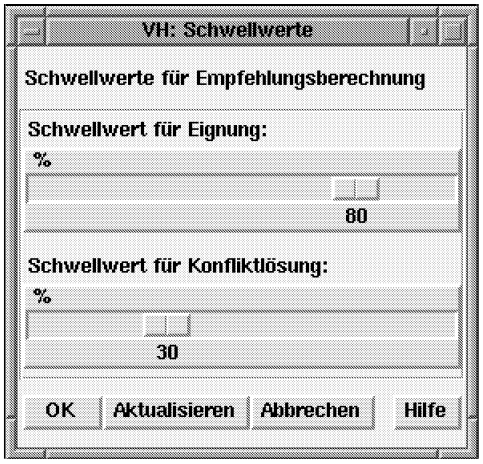
Beschreibung: Geeignete Techniken können entweder als Text oder grafisch dargestellt werden. Mit Hilfe des Checkbuttons kann zwischen raumsparender textueller und intuitiver

grafischer Darstellung umgeschaltet werden. Im Bild ist die textuelle Darstellung nur als Ausschnitt dargestellt.

Durch Anklicken einer Technik(kombination) und Betätigen des Aktionsknopfes *Visualisierung starten...* wird der IRIS-Explorer mit der ausgewählten Technik(kombination) gestartet[6].

Betätigen des Aktionsknopfes *Technik erklären...* öffnet das Fenster *Technik erklären*.

Fenster *Schwellwerte*



Aktivierung: Menü *Techniken/Schwellwerte*

Aktualisieren: Aktionsknopf

Funktion: Einstellen der Schwellwerte für die Eignung

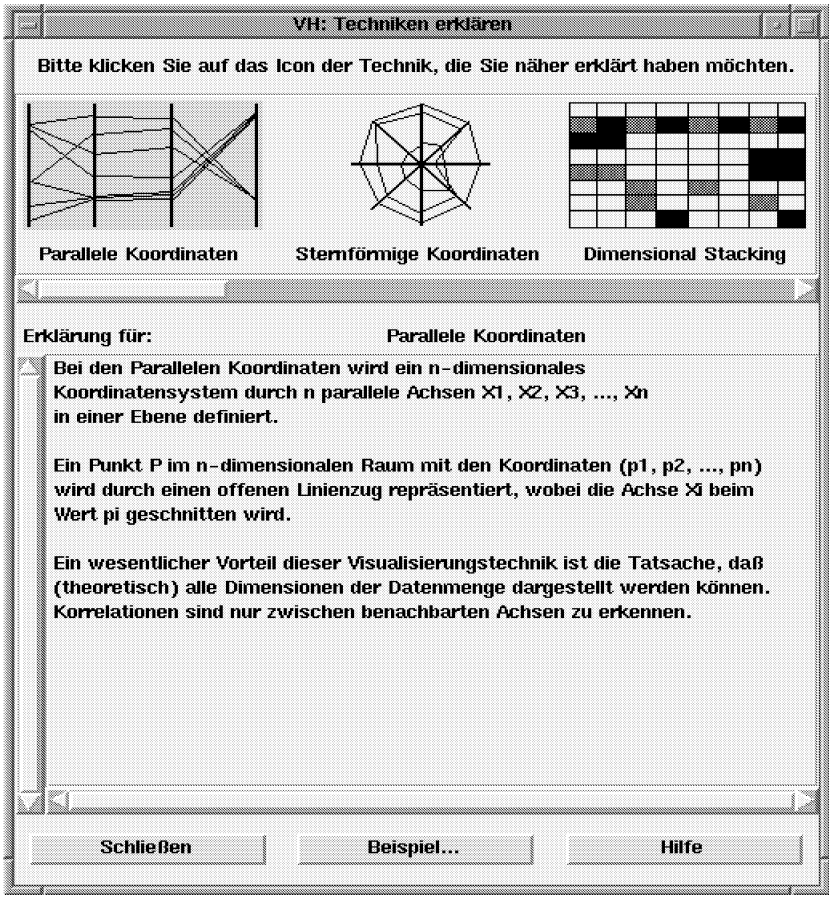
Beschreibung: Die Visualisierungshilfe berechnet die Eignung einer Technik für ein Visualisierungsproblem prozentual bezüglich der maximalen Eignung. Wenn dieser Eignungswert eine Schwelle überschreitet, wird die Technik als geeignet angenommen. Weiterhin werden Techniken, die schlechter als ein unterer Schwellwert sind, nicht in die Darstellung der Konflikte einbezogen.

Diese Schwellwerte können im Fenster *Schwellwerte* gesetzt werden.

---

6 Nur, wenn Explorer 2.1 eingebunden ist. Noch nicht vollständig implementiert.

Fenster *Technik erklären*



Aktivierung: Menü *Techniken/Erklärung* oder *Hilfe/Technik erklären*

Aktualisieren: nicht

Funktion: Anzeige von Erklärungstexten[7], Aufruf des IRIS-Explorers zur Anzeige von Beispiel-Maps[8]

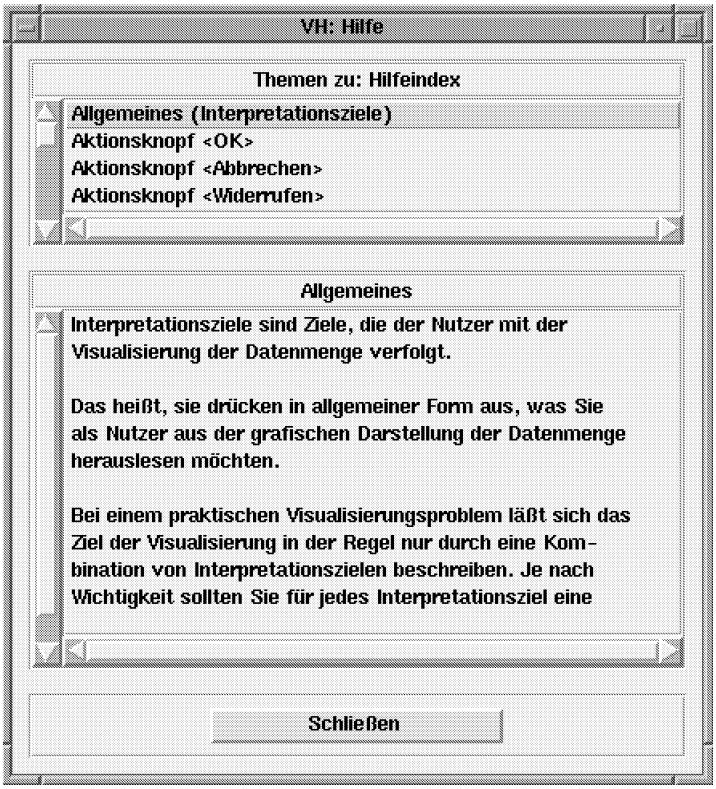
Beschreibung: Durch Anklicken eines Icons in der Iconleiste wird ein Erklärungstext zur entsprechenden Visualisierungstechnik angezeigt. Betätigen des Aktionsknopfes *Beispiel...* startet die Anzeige einer Beispiel-Map mit dem IRIS-Explorer.

---

7 Es sind noch nicht alle Erklärungstexte implementiert.

8 Nur, wenn Explorer V 1.0 oder 2.1 während der Installation eingebunden worden ist. Beispiel-Maps noch nicht implementiert.

Fenster *Hilfe*



Aktivierung: Menü *Hilfe/Index* oder Aktionsknöpfe *Hilfe*[9]

Aktualisieren: nicht

Funktion: Anzeige von Hilfetexten

Beschreibung: Durch Anklicken eines Hilfethemas in der Liste im oberen Teil des Fensters wird der Hilfetext zum entsprechenden Thema angezeigt. Wurde die Hilfe durch Betätigen des *Hilfe*-Aktionsknopfes in einem Fenster aufgerufen, so zeigt die Liste nur die für dieses Fenster relevanten Hilfethemen.

[9] Fensterbezogene Hilfetexte sind bisher nur für die Fenster *Interpretationsziele* und *Konflikte* implementiert.

Anhang B:

NetCDF-Datenschnittstelle der Visualisierungshilfe

1. NetCDF-Filestruktur aus Nutzersicht

Im folgenden wird eine Übersicht über alle Attribute gegeben, die Teil des von der Visualisierungshilfe eingelesenen netCDF-Files sind.

Zur Einführung in netCDF und in die verwendete Datenbeschreibung wird auf [REW93] und auf Arbeiten von Arndt [ARND93b] verwiesen.

Zur Zeit existiert noch kein Datendefinitions-Tool, das den Benutzer bei der Erstellung der Datenbeschreibung unterstützt. Das ist Gegenstand aktueller Arbeiten. Bis zur Verfügbarkeit dieses Werkzeuges können zum Setzen von Default-Werten für Variablenattribute globale Attribute verwendet werden.

Die Datenwerte selbst müssen momentan einen der Typen

- double (Gleitkommazahlen)
- char (Zeichenketten) oder
- long (ganze Zahlen) haben.

Bei Zeichenketten ist zu beachten, daß diese zusätzlich zu den Dimensionen, von denen sie im Experiment abhängig sind, noch von einer Stringlängen-Hilfsdimension (z.B. *stringlength*) abhängen müssen. Deren Maximalwert muß der größtmöglichen Stringlänge entsprechen. Bei einer Stringlängen-Hilfsdimension muß das Variablenattribut *describes* gleich "*stringlength*" gesetzt werden (siehe Tabelle "Attribute für alle Variablen"). In der *data*-Sektion eines netCDF-Files müssen Datenwerte solcher Merkmale rechts mit Leerzeichen oder Nullbytes auf die erforderliche Länge aufgefüllt werden.

Die Spalte *Default* gibt an, ob das Attribut unbedingt spezifiziert werden muß (Spalte leer) oder ob es berechnet bzw. interaktiv modifiziert werden kann und daher die Angabe eines Default-Wertes ausreicht (Spalte enthält Default).

Globale Attribute

Attributname	Typ	Zulässige Werte	Default	Bedeutung/Bemerkungen
type_of_grid	string	(bisher nicht spezifiziert)[1]		Gitterart (bisher nicht verwendet)
nature_of_domain	string	"points", "regions", "enumerated set"		Wirkungsbereich der Meßpunkte (bisher nicht verwendet)
selected_dims	integer	ganze Zahlen >=0	0	Anzahl zur Visualisierung ausgewählter Dimensionen
values_per_point	integer	ganze Zahlen >=0	0	Anzahl zur Visualisierung ausgewählter abhängiger Merkmale
quality	string	"complete", "incomplete"	"in-complete"	Vollständigkeit der Datenmenge (bisher nicht verwendet)
composition	string	"heterogeneous", "homogeneous"	"hetero-geneous"	Zusammensetzung der Datenmenge (bisher nicht verwendet)
correlated_vars	integer	ganze Zahlen >=0	0	Anzahl der Merkmale in der größten Korrelation nach der Merkmalsauswahl
num_of_tupels	integer	ganze Zahlen >=0	0	Anzahl von Tupeln korrelierender Merkmale
tupel0, tupel1, ...	string	tcl-Liste[2]	(nicht angeben)	Tupel korrelierender Merkmale
vh_Defaults	integer	0 oder 1	(nicht angeben)	komplette Datenanalyse wird durchgeführt, wenn diese Attribut fehlt oder 1 ist
num_of_observations	integer	ganze Zahlen >=0	0	Anzahl der Beobachtungsfälle
IG1, IG2, ..., IG6	string	"0", "0.25", "0.5", "0.75", "1"	"0"	Wichtungen der Interpretationsziele
upper_thresh	integer	0..100	80	Schwellwert für Eignung
lower_thresh	integer	0..100	30	Schwellwert für Konfliktlösung
colorblind	integer	0 oder 1	0	Benutzer ist farbenblind (1) oder nicht (0)
show_dim	integer	0 oder 1	1	Ausgewählte Dimensionen sollen visualisiert werden (1) oder nicht (0)

1 Einen beliebigen String angeben.
2 Liste von netCDF-IDs der beteiligten Merkmale, zuerst Dimensionen, aufsteigend sortiert, dann abhängige Merkmale, aufsteigend sortiert.

Attribute für alle Variablen

Attributname	Typ	Zulässige Werte	Default	Bedeutung/Bemerkungen
describes[3]	string	"dimension"[4], "stringlength"[5], "marker"[6]		Sagt aus, welche Art von Dimension vorliegt
ord	string	"ordinal", "nominal"		Unterscheidung, ob Variable ordinal oder nominal
type_of_values	string	bisher nur "scalar"		Typ der einzelnen Meßwerte bisher nicht verwendet)
info	integer	0 oder 1	0	Ist in den Werten des Merkmals nach der Datenauswahl relevante information enthalten (1) oder nicht (0)
vis_var	integer	0 oder 1	0	Soll das Merkmal visualisiert werden (1) oder nicht (0)

Attribute nominaler Variablen

Attributname	Typ	Zulässige Werte	Default	Bedeutung/Bemerkungen
valid_values	integer	ganze Zahlen >0		Anzahl der gültigen Werte
v0, v1, ...	string			Die gültigen Werte
num_of_values	integer	ganze Zahl	0	valid_values nach Datenauswahl

Attribute ordinaler Variablen

Attributname	Typ	Zulässige Werte	Default	Bedeutung/Bemerkungen
valid_range	integer, integer	Paare ganzer Zahlen[7]		Wertebereich
min	ínteger	ganze Zahl	0	Minimaler Wert des Merkmals nach der Datenauswahl
max	ínteger	ganze Zahl	0	Maximaler Wert des Merkmals nach der Datenauswahl
num_of_intervals	integer	ganze Zahl	0	Anzahl der Intervalle des Wertebereiches
size_of_intervals	integer	ganze Zahl	0	Größe der Intervalle des Wertebereiches
const_intervals	string	"size", "count"	"count"	Zeigt an, ob bei Änderung von Minimum und Maximum Größe oder Anzahl der Intervalle konstant bleiben soll

---

3 Dieses Attribut ist nur für im netCDF-File unter *dimensions* definierte Variablen und die Hilfvariable vh_internal_Marker (reserviert) zulässig und erforderlich.

4 Merkmal beschreibt Dimension mit Raum- oder Zeitbezug.

5 Merkmal beschreibt Stringlänge.

6 Merkmal beschreibt Markierung von Datensätzen. Für interne Zwecke reserviert.

7 Die erste Zahl muß kleiner sein als die zweite.

Schließlich soll als Beispiel ein netCDF-File für Tabelle 1 aus Anhang A angeführt werden. Es kann mit dem Werkzeug *ncgen* aus der netCDF-Distribution in ein binäres netCDF-File übertragen werden.

```
netcdf Beispiel {
dimensions:
  // Physikalische Dimension
  Zeit = UNLIMITED;
  // Hilfsdimension
  stringlength = 5 ;

variables:
  short stringlength(stringlength) ;
    stringlength:describes = "stringlength";
  long Zeit(Zeit) ;
    Zeit:describes = "dimension" ;
    Zeit:valid_range = 1, 5;
    Zeit:ord = "ordinal";
  long Temp(Zeit);
    Temp:valid_range = 0, 3000;
    Temp:ord ="ordinal";
  char Farbe(Zeit, stringlength);
    Farbe:valid_values = 3 ;
    Farbe:v0 = "grau" ;
    Farbe:v1 = "rot" ;
    Farbe:v2 = "weisz" ;
    Farbe:ord = "nominal";

// Globale Attribute
:type_of_values = "scalar";
:values_per_point = 0;
:composition = "heterogeneous";
:nature_of_domain = "points";
:quality = "incomplete";
:type_of_grid = "???";
:num_of_observations = 3;
:IG1 = "0";
:IG2 = "0";
:IG3 = "0";
:IG4 = "0";
:IG5 = "0";
:IG6 = "0";
:lower_thresh = 30;
:upper_thresh = 80;
:colorblind = 0;
:show_dims = 1;
:selected_dims = 0;
:correlated_vars = 0;
:num_of_tupels = 0;

// Default-Werte
:num_of_intervals = 1;
:num_of_values = 0;
:size_of_intervals = 1;
:const_intervals = "count";
:info = 0;
:vis_var = 0;

data:
  Zeit = 0, 1, 2;
  Temp = 300, 1000, 2000;
  Farbe =
      "grau ",
      "rot  ",
      "weisz" ;
}
```

2. NetCDF-Filestruktur aus Entwicklersicht

Im folgenden wird eine Übersicht über alle Attribute gegeben, die intern in der Visualisierungshilfe verwendet werden.

Sie können nach verschiedenen Gesichtspunkten gruppiert werden.

1) Scope:

- V - Variablenattribut
- O - Variablenattribut, ordinale Variablen
- N - Variablenattribut, nominale Variablen
- G - globales Attribut
- T - temporäres globales Attribut (nicht gespeichert)

2) Herkunft:

- s - aus interner Struktur des netCDF Files synthetisiert
- d - zur Datenbeschreibung vom netCDF vorgeschriebenes Attribut
- ds - aus vorgeschriebenen Attributen synthetisiert
- z - zur Datenbeschreibung im Rahmen der Visualisierungshilfe zusätzlich eingeführt
- zs - aus zusätzlich eingeführten Attributen synthetisiert
- h - internes Hilfsattribut

3) Rechte des Programms vh in Bezug auf Attribut:

- r - vh liest Attribut
- w - vh liest und schreibt Attribut

4) Speicherung:

- 0 - von vh nicht im netCDF gespeichert
- 1 - von vh im netCDF gespeichert
- 2 - im netCDF enthalten, aber von vh nicht gespeichert

Vier Arten von Attributen können unterschieden werden.

a) essentielle Attribute

Diese können nicht im Rahmen einer Datenanlyse aus den Daten oder aus anderen Attributen erzeugt werden. Sie sind auch nicht im Rahmen der Visualisierungshilfe interaktiv modifizierbar, sondern müssen durch das Datendefinitionstool erzeugt werden.

Attribut	Scope	Herkunft	Rechte	gespeichert	Originalattribut
ID	V	s	r	2	(intern)
Name	V	s	r	2	(intern)
AllDims	G	s	r	2	(intern)
AllVars	G	s	r	2	(intern)
ord	V	z	r	2	ord
valid_values	N	d	r	2	valid_values
Range	V	ds	r	2	valid_range (ordinal) v0, v1, ... (nominal)
type_of_values	V	z	r	2	type_of_values
type_of_grid	G	z	r	2	type_of_grid
nature_of_domain	G	z	r	2	nature_of_domain

b) berechenbar und abhängig von Benutzereingaben

Diese Attribute können aus anderen Attributen oder aus der Datenmenge berechnet werden. Sie ändern sich, wenn interaktiv Attribute oder Datenmenge geändert werden.

Attribut	Scope	Herkunft	Rechte	gespeichert	Originalattribut
min	O	z	w	1	min
max	O	z	w	1	max
num_of_intervals	O	z	w	1	num_of_intervals
size_of_intervals	O	z	w	1	size_of_intervals
num_of_values	N	z	w	1	num_of_values
info	V	z	w	1	info
selected_dims	G	z	w	1	selected_dims
values_per_point	G	z	w	1	values_per_point
quality	G	z	w	1	quality
composition	G	z	w	1	composition
correlated_vars	G	z	w	1	correlated_vars
Results [8]	T	z	w	0	(kein)
EffResults [9]	T	z	w	0	(kein)
Distance [10]	T	z	w	0	(kein)
AUXResults [11]	T	h	w	0	(kein)
AUXDistance [12]	T	h	w	0	(kein)

[8] Liste geeigneter Technik-Kombinationen nach Anwendung des Expressivitätskriteriums (Hilfsattribut)
[9] Liste geeigneter Technik-Kombinationen nach Anwendung des Effektivitätskriteriums (Hilfsattribut)
[10] Liste von Abstandswerten nach Anwendung des Expressivitätskriteriums (Hilfsattribut)
[11] Hilfsattribut zur Konvertierung der Ergebnisse
[12] Hilfsattribut zur Konvertierung des Abstandes

c) berechenbar und unabhängig von Benutzereingaben

Diese Attribute hängen von Teilen der Datenmenge ab, die interaktiv nicht modifiziert werden.

Attribut	Scope	Herkunft	Rechte	gespeichert	Originalattribut
num_of_tupels	G	z	w*[13]	1	num_of_tupels
Tupels	G	zs	w*	1	tupel0, tupel1, ...
vh_Defaults	G	h	w	1	vh_Defaults

d) interaktiv modifizierbar

Diese Attribute können interaktiv eingegeben werden. Beim Erzeugen des netCDF-Files müssen sie mit Default-Werten belegt werden.

Attribut	Scope	Herkunft	Rechte	gespeichert	Originalattribut
num_of_intervals	O	z	w	1	num_of_intervals
size_of_intervals	O	z	w	1	size_of_intervals
const_intervals	O	z	w	1	const_intervals
vis_var	V	z	w	1	vis_var
num_of_observations	G	z	w	1	num_of_observations
IG1..IG6	G	z	w	1	IG1..IG6
upper_thresh	G	z	w	1	upper_thresh
lower_thresh	G	z	w	1	lower_thresh
colorblind	G	z	w	1	colorblind
show_dim	G	z	w	1	show_dim

[13] w*: Eigentlich müßte der Datenbeschreibungs-Generator diese Attribute erzeugen. Da dieser nicht rechtzeitig fertig wird, erzeugt VH diese Attribute beim Laden, falls sie fehlen.

3. Abhängigkeiten zwischen den Attributen

Zwischen den Attributen bestehen gewisse Abhängigkeiten, die darstellen, welche anderen Attribute bei der Änderung eines bestimmten Attributs neu berechnet werden müssen. Diese können als gerichteter azyklischer Graph dargestellt werden. Da der Graph recht groß wird, soll aus Gründen der Platzersparnis die Tabellenform bevorzugt werden.

Attribut	abhängige Attribute
vis_var	values_per_point, selected_dims, quality, composition, correlated_vars
num_of_observations	min, max, num_of_values, info, quality, Distance
num_of_intervals	info, Distance
const_intervals	num_of_intervals, size_of_intervals
min	num_of_intervals, size_of_intervals
max	num_of_intervals, size_of_intervals
IG1, IG2, ..., IG6	Distance
num_of_values	Distance
selected_dims	Distance
values_per_point	Distance
correlated_vars	Distance
upper_thresh	Distance
lower_thresh	Distance
Distance	Results
Results	EffResults
colorblind	EffResults
show_dim	EffResults
num_of_tupels	correlated_vars
Tupels	correlated_vars

Bemerkungen:

1) Die wechselseitige Abhängigkeit zwischen *num_of_intervals* und *size_of_intervals* wird nicht modelliert, da das zu einem Kreis im Abhängigkeitsgraphen führen würde. Das Widget, in dem diese eingegeben werden, muß die Abhängigkeit selbst sicherstellen.

2) Für Berechnungen wird nur *num_of_intervals* verwendet.

Anhang C:

Bugs und Einschränkungen der Visualisierungshilfe

Der Prototyp "Visualisierungshilfe" ist in einem Entwicklungsstadium, in dem normalerweise mit umfangreichen Tests begonnen werden könnte. Es sind garantiert noch relativ viele Fehler enthalten. Die folgende Liste enthält einige bekannte Fehler und Einschränkungen:

1. Bei Fehlern im netCDF-File erfolgt teilweise Programmabsturz.
2. Fehlende Attribute im netCDF File führen zum Programmabbruch. Wenn das netCDF-File mit dem (noch zu implementierenden) Datenbeschreibungs-Generator erstellt wurde, tritt das nicht auf. Dieser stellt sicher, daß alle Attribute vorhanden sind.

   Beim Erstellen eines netCDF-Files von Hand (mit ncgen) muß das fehlende Attribut von Hand in das cdl-File eingetragen und dieses neu übersetzt werden. Default-Werte von Variablenattributen können als globale Attribute spezifiziert werden. Für empfohlene Defaults siehe Anhang B.
3. *size_of_intervals*, *num_of_intervals*, *min*, *max* können momentan nur als ganze Zahlen angegeben werden.
4. Es sind nicht alle Hilfe- und Erklärungstexte implementiert.
5. Es sind noch keine Beispielmaps und Visualisierungsmaps implementiert.
6. Die Explorer-Online-Kopplung ist fehleranfällig, da eine vollständige Synchronisation nicht möglich ist. Insbesondere werden beim Schließen des Hauptfensters des IRIS-Explorers zwar dessen Fenster vom Bildschirm entfernt, die zugehörigen Prozesse jedoch nur teilweise beendet. In diesen Fällen bleibt der Aktionsknopf *Explorer beenden* weiterhin aktiv. Bei dessen Betätigen werden die "Restprozesse" des Explorers aus dem Speicher entfernt, was daran zu erkennen ist, daß o.g. Aktionsknopf inaktiv geschaltet wird.

Beim Beenden der Visualisierungshilfe oder vor dem Neustart des Explorers werden die "Restprozesse" jedoch auf jeden Fall beendet.

7. Beim Betätigen der Radiobuttons zum Wichten der Interpretationsziele oder zum Einstellen der Nutzeroptionen werden teilweise die Werte intern zwar gesetzt, aber das Erscheinungsbild der Radiobuttons selbst nicht korrekt geändert. Dieser Fehler scheint seine Ursache im Tk zu haben. Die Radiobuttons werden korrekt aktualisiert, wenn nach dem Anklicken der Mauszeiger in ein anderes Gebiet auf dem Bildschirm und dann wieder über die radiobuttons bewegt wird.

Anhang D:

Listing der automatischen Konfliktlösung

Relevante Auszüge aus *Func.h*

```
Anzahl Techniken und Interpretationsziele
#define NR_TECH 15
#define NR_GOALS 6
```

```
Struktur fuer Schwellwerte
     upper:    Techniken mit einem Gesamtabstand <= upper werden fuer die
               Anzeige von Konflikten verwendet
     lower:    Techniken mit einem Gesamtabstand <= lower werden als
               geeignet angenommen
typedef struct tThreshStruct {double upper, lower; } tThresh;
```

```
Struktur fuer Datenbeschreibung
     k:        Anzahl der Merkmale im groeszten Tupel mit relevanter
               gemeinsamer Info
     m:        Anzahl Merkmale
     n:        Anzahl Datensaetze
     w:        mittl. Umfang des Wertebereiches
typedef struct tDataSpecStruct {int k, m, n, w;} tDataSpec;
```

```
Ergebnis der automatischen Konfliktloesung
     len:      Laenge des Tupels
     tech:     Array von Techniken
     abstand:  Gesamtabstand des Tupels
     abs_vect:Abstandsvektor des Tupels
     next:     Naechstes Tupel
typedef struct tAutoResultStruct
      { int len, tech[NR_TECH];
          double abstand;
          double abs_vect[NR_GOALS];
          struct tAutoResultStruct *next;
        } tAutoResult;
```

Relevante Auszüge aus *Func.c*

```
Offset, der zum Wichtungsvektor zu addieren ist, um die
theoretische) schlechteste Technik zu erhalten
```

```
#define BBAD_OFS 0.25
```

```
Techniklisten-Record
     key:          Schluessel der Technik
     improvment:   Durch zusaetzlichen Einsatz der Technik
                   erreichbare Gesamtverbesserung
     imp_vect:     Durch zusaetzlichen Einsatz der Technik
                   erreichbare Verbesserung der einzelnen Int-Ziele
```

```
typedef struct tTechListRecStruct
      {
        int key;
        double improvment;
        double imp_vect[NR_GOALS];
      } tTechListRec;
```

```
Technikliste
     len:          Anzahl Records
     list:         Feld von Records

Eigenschaft: list ist aufsteigend nach improvment sortiert
```

```
typedef struct tTechListStruct
      {
         int len;
         tTechListRec list[NR_TECH];
      } tTechList;
```

```
Erstellen einer Technik-Liste

Letzte Aenderung: 19.07.1994 (Uwe Rauschenbach)

Parameter:
     - abstand:    Abstandsvektoren der Techniken (aus fkt. Ansatz) (IN)
     - b:          Abstandsvektor der virtuellen schlechtestmoeglichen
                   Technik (IN)
     - techlist:   Zeiger auf Technikliste (OUT)
```

```
void MakeTechList(double abstand[NR_TECH][NR_GOALS],
                  double b[NR_GOALS],
                  tTechList *techList)
{
   int i,j;

   for (i=0; i<NR_TECH; i++)
   {
      techList->list[i].key=i;
   }
   techList->len=NR_TECH;
   ReComputeImprov(abstand, b, techList);
}
```

```
Sortieren der techList aufsteigend nach improvment

Letzte Aenderung: 19.07.1994 (Uwe Rauschenbach)

Parameter:
     - list:   Zeiger auf Technikliste (INOUT)
```

```
void SortList(tTechList *list)
{
   int sort, i, last, end;
   tTechListRec tmp, *first, *next;

   last=list->len;
   do
   {
      sort=0;
      end=last;
      next=&(list->list[0]);

      for (i=1; i<end; i++)
      {
         first=next;
         next=&(list->list[i]);
         if (first->improvment > next->improvment)
         {
            tmp=*first;
            *first=*next;
            *next=tmp;
            sort++;
            last=i;
         }
      }
   } while (sort);
}
```

```
Loeschen einer Ergebnisliste

Letzte Aenderung: 19.07.1994 (Uwe Rauschenbach)

Parameter:
     - res:    Zeiger auf Zeiger auf zu loeschende Ergebnisliste (IN/OUT)
```

```
void DiscardResults(tAutoResult **res)
{
   tAutoResult *tmp;

   tmp=*res;
   while (tmp)
   {
      free(tmp);
      tmp=*res=(*res)->next;
   }
   *res=NULL;
}
```

```
Neuberechnen der Improvements nach dem Hinzufuegen einer Technik

Letzte Aenderung: 19.07.1994 (Uwe Rauschenbach)

Parameter:
     - abstand:   Abstandsvektoren der Techniken (aus fkt. Ansatz) (IN)
     - bad:       Abstandsvektor der zu verbessernden Technik
                  (-kombination) (IN)
     - techlist:  Zeiger auf Technikliste, in der die Improvements
                  modifiziert werden sollen(IN/OUT)
```

```
void ReComputeImprov(double abstand[NR_TECH][NR_GOALS],
                 double bad[NR_GOALS],
                 tTechList *techList)
{
   double tmp;
   int i, idx, j;

   /* idx bezeichnet Element, in das gespeichert wird
    * Nur Elemente, deren Verbesserung >0 ist, bleiben
    * erhalten (idx++), andere werden ueberschrieben
    * Wichtig! Alle Felder des Record muessen ueber-
    * schrieben werden!
    */
   idx=0;
   for (i=0; i<techList->len; i++)
   {
      techList->list[idx].improvment=0;
      for (j=0; j<NR_GOALS; j++)
      {
         tmp = bad[j]-abstand[techList->list[i].key][j];
         tmp = (tmp>0?tmp:0);
         techList->list[idx].improvment += tmp;
         techList->list[idx].imp_vect[j] = tmp;
      }
      if (techList->list[idx].improvment)
      {
         /* Ueberschreiben des key-Feldes zum Sicherstellen der
         * Konsistenz */
         techList->list[idx].key=techList->list[i].key;
         /* Speichern auf naechstes Element */
         idx++;
      }
   }

   techList->len=idx;
   SortList(techList);
}
```

```
Anhaengen eines Technik-Tupels an die Ergebnisliste

Letzte Aenderung: 19.07.1994 (Uwe Rauschenbach)

Parameter:
     - list:   Zeiger auf Zeiger auf Ergebnisliste (IN/OUT)
     - rec:    Zeiger auf Technik-Tupel (IN)

Seiteneffekte: Speicherplatz dynamisch allokiert
```

```
void AppendToList(tAutoResult **list, tAutoResult *rec)
{
   tAutoResult *pTmp, *pAfter, *pBefore;

   pTmp=malloc(sizeof(*pTmp));
   *pTmp=*rec;

   if (!(*list))
   {
      /* Leere Liste */
      *list = pTmp;
      pTmp->next=NULL;
   }
   else
   {
      /* Einfuegen in Liste */
      pBefore=*list;
      while (pBefore && (pBefore->abstand <= rec->abstand))
      {
         pAfter=pBefore;
         pBefore=pBefore->next;
      }
      /* Einfuegen am Anfang? */
      if (pBefore==*list) *list=pTmp;
      else pAfter->next=pTmp;

      pTmp->next=pBefore;
   }
}
```

```
Rekursives Berechnen geeigneter Technikkombinationen

Letzte Aenderung: 19.07.1994 (Uwe Rauschenbach)

Parameter:
     - abstand:   Abstandsvektoren aus fkt. Ansatz ("durchgereicht") (IN)
     - techList:  Zeiger auf Technik-Liste (IN)
     - resTup:    Zeiger auf Ergebnistupel (IN/OUT)
     - kMin:      Zeiger auf minimale Tupellaenge (glob. Variable)
                  (IN/OUT)
     - W:         Schwellwert fuer Eignung (IN)
     - resList:   Zeiger auf Ergebnisliste (IN/OUT)
```

```
void Combine(
   double abstand[NR_TECH][NR_GOALS],
   tTechList *techList,
   tAutoResult *resTup,
   int *kMin,
   double W,
   tAutoResult **resList)
{
   tTechList lTechList;
   tTechListRec techListEntry;
   tAutoResult lResTup, *pTmp;
   int i, j, k, it, testIdx, testEl;

   lTechList=*techList;
   /* Anpassen der Minimallaenge */
   if ((resTup->abstand <= W) && (*kMin>resTup->len))
   {
      DiscardResults(resList);
      *kMin=resTup->len;
   }

   if (resTup->len==*kMin)
   {
      if (resTup->abstand<=W) AppendToList(resList, resTup);
      return;
   }
   else
   {
      for (it=techList->len-1; it>=0; it--)
      {
         techListEntry=techList->list[it];

         lResTup=*resTup;
         lResTup.tech[lResTup.len]=techListEntry.key;
         lResTup.len++;
         lResTup.abstand -= techListEntry.improvment;
         for (i=0; i<NR_GOALS; i++) lResTup.abs_vect[i] -=
            techListEntry.imp_vect[i];

         ReComputeImprov(abstand, lResTup.abs_vect, &lTechList);

         Combine(abstand, &lTechList, &lResTup, kMin, W, resList);
      };
   }
}
```

```
Automatische Konfliktloesung

Letzte Aenderung: 19.07.1994 (Uwe Rauschenbach)

Parameter:
     - lower:  Unterer Schwellwert (IN)
     - b:      Liste mit Wichtungen (IN)
     - data:   Datencharacteristika (IN)
     - result: Liste von geeigneten Technik-Kombinationen (INOUT)

Return:  0     keine Loesung
         1..   Laenge des kuerzesten Tupels in result

Seiteneffekte:  -Der Speicher für result wird dynamisch allokiert und muß
                  von der rufenden Prozedur freigegeben werden.
                 -*result muß mit leerer Liste initialisiert sein.
```

```
int auto_resolve( double lower, double b[NR_GOALS], tDataSpec data,
            tAutoResult **result)
{
   double abstand[NR_TECH][NR_GOALS];
   double D[NR_TECH];
   double bbad[NR_GOALS];
   int good, bad;
   int liste[NR_TECH+1];
   tThresh thresh;
   int i,j,tech,ti, kMin;

   tTechList techList;
   tAutoResult resTup, *resList;

   double maxDist, improv, neededImprov;

   maxDist=b[0];;
   for (i=1; i<NR_GOALS; i++) maxDist += b[i];
   thresh.upper=thresh.lower=maxDist;
   funkt_ansatz(b, thresh, data, &abstand, &D, &liste, &good, &bad);

   /* Initialisiere kMin -> schlimmster Fall: eine Technik je Ziel */
   kMin=NR_GOALS;

   /* Initialisiere bad: Schlechtestmoegliche, virtuelle Technik
    * Problem: Wenn alle b[i]==0 -> keine Verbesserung mgl.
    * Loesung: Offset addieren, um Sonderfall zu berueksichtigen */
   for (i=0; i<NR_GOALS; i++) bbad[i]=b[i]+BBAD_OFS;

   MakeTechList(abstand, bbad, &techList);
   if (!techList.len) return 0;

   /* Initialisiere Ergebnislisten */
   resList=NULL;

   resTup.len=0;
   resTup.abstand=maxDist+NR_GOALS*BBAD_OFS;
   for (i=0; i<NR_GOALS; i++) resTup.abs_vect[i]=bbad[i];
   resTup.next=NULL;
   Combine (abstand, &techList, &resTup, &kMin, lower, &resList);

   *result=resList;
   return kMin;
}
```

Anhang E:

Listing des Dependency-Mechanismus

Relevante Auszüge aus *Depend.tcl*

```
        >>>>>>>>> Beschreiben der Dependency-Struktur <<<<<<<<<
```

```
Initialisieren der Variablennamen

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.InitVarNames {} {
   global RW_Vars       # Read/Write Variablen -> vh aendert diese
   global RO_Vars       # Read variablen -> vh aendert diese nicht
   global AUX_Vars      # Hilfsvariablen

   # Variablen, die von anderen Variablen abhaengig sind, muessen in der
   # Liste nach diesen stehen!
   set RW_Vars    {vh_Defaults num_of_intervals size_of_intervals \
                   const_intervals min max num_of_values info vis_var \
                   selected_dims values_per_point quality composition \
                   num_of_observations correlated_vars IG1 IG2 IG3 IG4 \
                   IG5 IG6 upper_thresh lower_thresh num_of_tupels \
                   Tupels Distance Results colorblind show_dim EffResults}
   set RO_Vars    {ID Name ord valid_values Range type_of_values AllDims \
                   AllVars type_of_grid nature_of_domain }
   set AUX_Vars   {AUXResults AUXDistance}
}
```

```
Initialisieren von Listen der Widgets,
die Wert einer Variablen darstellen

Widget musz nur in Liste aufgenommen werden,
wenn geänderte Variable auch anderweitig als
durch Eingabe in dem betreffenden Widget geaendert
werden kann

Keine -> "0"

Letzte Aenderung: 26.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.InitDependants {} {
   global Dependants RW_Vars RO_Vars AUX_Vars

   foreach i $RW_Vars {set Dependants($i) {0}}
   foreach i $RO_Vars {set Dependants($i) {0}}
   foreach i $AUX_Vars {set Dependants($i) {0}}

   set Dependants(info) {topVarSel}
   set Dependants(Distance) {topConflicts}
   set Dependants(vis_var) {topVarSel topIntervals topCorr topEff}
   set Dependants(correlated_vars) {topCorr}
   set Dependants(EffResults) {topResults}
   set Dependants(min) {topIntervals}
   set Dependants(max) {topIntervals}
   set Dependants(num_of_intervals) {topIntervals}
   set Dependants(size_of_Intervals) {topIntervals}
   set Dependants(upper_thresh) {topConflicts}
   set Dependants(num_of_obeservations) {topConflicts}
   set Dependants(selected_dims) {topConflicts}
   set Dependants(values_per_point) {topConflicts}
}
```

```
Konsistenzroutine 1 initialisieren
-> Abhaengigkeitsgraph wird beschrieben

Letzte Aenderung: 26.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.InitKR1 {} {
   global KR1 RO_Vars RW_Vars AUX_Vars

   foreach i $RO_Vars {set KR1($i) {}}
   foreach i $RW_Vars {set KR1($i) {}}
   foreach i $AUX_Vars {set KR1($i) {}}

   set KR1(vis_var)     {values_per_point selected_dims quality \
                         composition correlated_vars}

   set KR1(num_of_observations)     {min max num_of_values info quality \
                                     Distance}

   set KR1(num_of_intervals)  {info Distance}

   set KR1(const_intervals) {num_of_intervals size_of_intervals}
   set KR1(min) {num_of_intervals size_of_intervals}
   set KR1(max) {num_of_intervals size_of_intervals}

   set KR1(IG1) {Distance}
   set KR1(IG2) {Distance}
   set KR1(IG3) {Distance}
   set KR1(IG4) {Distance}
   set KR1(IG5) {Distance}
   set KR1(IG6) {Distance}

   set KR1(num_of_values) {Distance}
   set KR1(selected_dims) {Distance}
   set KR1(values_per_point) {Distance}
   set KR1(correlated_vars) {Distance}
   set KR1(upper_thresh) {Distance}
   set KR1(lower_thresh) {Distance}

   set KR1(num_of_tupels) {correlated_vars}
   set KR1(Tupels) {correlated_vars}

   set KR1(colorblind) {EffResults}
   set KR1(show_dim) {EffResults}

   # Hilfsabhängigkeiten fuer globale Variablen
   # Results und Distance
   set KR1(Distance) {Results AUXDistance}
   set KR1(Results) {EffResults}
   set KR1(EffResults) {AUXResults}
}
```

```
Konsistenzroutine 2 initialisieren
 -> Routinen zur Neuberechnung eines oder mehrerer
    Datenaspekte (netCDF-Attribute)
 -> WICHTIG: Am Ende jeder KR2-Prozedur
    Dirty-Flag aller berechnetet Variablen auf 0 setzen

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.InitKR2 {} {
   global RO_Vars RW_Vars AUX_Vars
   global KR2

   foreach i $RO_Vars {set KR2($i) "DummyKR2"}
   foreach i $RW_Vars {set KR2($i) "DummyKR2"}
   foreach i $AUX_Vars {set KR2($i) "DummyKR2"}

   set KR2(size_of_intervals) Depend.Intervals
   set KR2(num_of_intervals) Depend.Intervals

   set KR2(values_per_point) Depend.KR_SelVar
   set KR2(selected_dims) Depend.KR_SelVar

   set KR2(Results) "Depend.Results"
   set KR2(Distance) "Depend.Distance"

   set KR2(correlated_vars) "Depend.CorrVars"

   set KR2(min) "Depend.AnalyseVarAtts"
   set KR2(max) "Depend.AnalyseVarAtts"
   set KR2(num_of_values) "Depend.AnalyseVarAtts"
   set KR2(info) "Depend.AnalyseVarAtts"

   set KR2(num_of_tupels) "Depend.AnalyseGlobAtts"
   set KR2(Tupels) "Depend.AnalyseGlobAtts"

   set KR2(EffResults) "Depend.Eff"
}
```

```
Initialisieren der WT (WriteThrough) Routinen
-> Zurueckschreiben der Variablen ins netCDF File

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.InitWT {} {
   global RW_Vars
   global WT

   foreach i $RW_Vars {set WT($i) "DummyWT"}

   # Variablenattribute
   set WT(num_of_intervals) WriteVarCallback
   set WT(size_of_intervals) WriteVarCallback
   set WT(const_intervals) WriteVarCallback
   set WT(min) WriteVarCallback
   set WT(max) WriteVarCallback
   set WT(num_of_values) WriteVarCallback
   set WT(info) WriteVarCallback
   set WT(vis_var) WriteVarCallback

    # globale Attribute
    set WT(selected_dims) WriteGlobCallback
    set WT(values_per_point) WriteGlobCallback
    set WT(quality) WriteGlobCallback
    set WT(composition) WriteGlobCallback
    set WT(num_of_observations) WriteGlobCallback
    set WT(correlated_vars) WriteGlobCallback
    set WT(IG1) WriteGlobCallback
    set WT(IG2) WriteGlobCallback
    set WT(IG3) WriteGlobCallback
    set WT(IG4) WriteGlobCallback
    set WT(IG5) WriteGlobCallback
    set WT(IG6) WriteGlobCallback

    set WT(upper_thresh) WriteGlobCallback
    set WT(lower_thresh) WriteGlobCallback

    set WT(vh_Defaults) WriteGlobCallback

    set WT(colorblind) WriteGlobCallback
    set WT(show_dim) WriteGlobCallback

    set WT(num_of_tupels) WriteGlobCallback
    set WT(Tupels) WriteGlobCallback
}
```

```
   >>>>>>>>> Traces zur Realisierung des Dependency-Mechanismus <<<<<<<<<
```

```
Initialisieren der traces, die den Dependency-
Mechanismus ausloesen

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.InitTraces {} {
   global netCDF
   global Results Distance

   Depend.EnableTraces {}

   trace variable netCDF r Traces.Read
   trace variable netCDF w Traces.Write

   trace variable Distance r Traces.AUX
   trace variable Results r Traces.AUX
}
```

```
Verbieten der Traces
 -> guenstig, wenn viele aufeinanderfolgende
    Schreibzugriffe auf dasselbe Attribut erfolgen
    (nicht bei jedem Einzelzugriff Ausloesen des Dep-Mech)
 -> Wichtig: nach letztem Schreibzugriff Aufruf von
    Depend.EnableTraces

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.DisableTraces {} {
   global enableTraces

   set enableTraces 0
}
```

```
Erlauben der Traces
 -> Aufruf nach letztem Schreibzugriff nach Depend.DisableTraces
 -> Registriert Schreibzugriff auf alle Vriablen in var_list

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter:
     - var_list:  Liste aller Attribute, auf die nach
                  Depend.DisableTraces ein Schreibzugriff erfolgte
```

```
proc Depend.EnableTraces {var_list} {
   global enableTraces WT netCDF

   set enableTraces 1
   foreach i $var_list {
      Traces.Write netCDF $i dummy
   }
}
```

```
Trace, der bei Schreibzugriff auf eine Variable aufgerufen wird
 -> loest Dependency-Mechanismus aus

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter:
     - name:    Name der Variablen, auf die Schreibzugriff erfolgte
     - element: Falls Variable Array, Name des geschriebenen Array-
                Elements
     - op:          r / w / u fuer Read / Write / Unset
```

```
proc Traces.Write {name element op} {
   global Dirty netCDF KR1 WT enableTraces

   if {[string compare $element Results]!=0 &&
       [string compare $element Distance]!=0} {
      set Dirty($element) 0
   }

   if {!$enableTraces} {return}

   Depend.Dirty $element
}
```

```
Trace, der bei Lesezugriff auf eine Variable aufgerufen wird
 -> Testet, ob Variable gueltig (Dirty=0)
 -> Falls nicht, aufruf von KR2 zum Neuberechnen des Attributs

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter:
     - name:      Name der Variablen, auf die Lesebzugriff erfolgte
     - element:   Falls Variable Array, Name des gelesenen Array-Elements
     - op:        r / w / u fuer Read / Write / Unset
```

```
proc Traces.Read {name element op} {
   global Dirty netCDF KR2 KR1 enableTraces

   if {!$enableTraces} {return}

   if {[string compare $element ""]!=0} {
      # Call KR2 to recompute if dirty:
      if {$Dirty($element)==1} {
         $KR2($element)
         set Dirty($element) 0
      }
   }
}
```

```
Trace, der bei Lesezugriff auf die globalen
Hilfsvariablen Distance und Results aufgerufen wird

Notwendig, um Lesezugriffe auf diese mit netCDF(Distance) und
netCDF(Results) zu synchronisieren

 -> zugehoerige Dirty-Flags: Dirty(AUXDistance) und
    Dirty(AUXResults)

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter:
     - name:   Name der Variablen, auf die Schreibzugriff erfolgte
     - element:   Falls Variable Array, Name des geschriebenen Array-
                  Elements
     - op:        r / w / u fuer Read / Write / Unset
```

```
proc Traces.AUX {name element op} {
   global Dirty enableTraces

   if {!$enableTraces} {return}

   # name ist "Distance" oder "Results"
   if {$Dirty(AUX$name)==1} {
      Depend.Convert$name
      set Dirty(AUX$name) 0
   }
}
```

```
>>>>>>>>> Dirty-Management und Redraw <<<<<<<<<
```

```
Setzen der Dirty-Flags aller abhaengigen Attribute auf 1
NICHT extern aufrufen!

Die Dirty-Flags aller abhaengigen Variablen werden auf 1 gesetzt
und die Namen der abhaengigen Variablen in DAG2Level gesammelt

DAG2Level enthaelt den Teil des Abhaengigkeits-Graphen (DAG), dessen
Wurzelknoten "element" ist, in Stufenform (siehe Studienarbeit)

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter:
     - element: Das Attribut des Arrays netCDF, das dirty zu setzen ist
     - lv:      Die aktuelle Ebene im Teil-DAG (Wurzel=0)
```

```
proc Depend.SetDirty {element lv} {
   global netCDF Dirty KR1 DAG2Level
   if {$DAG2Level(max)<$lv} {
      set DAG2Level(max) $lv
      set DAG2Level($lv) {}
   }
   foreach dep $KR1($element) {
      set Dirty($dep) 1
      if {[lsearch -exact $DAG2Level($lv) $dep]==-1} {
         lappend DAG2Level($lv) $dep
      }
      Depend.SetDirty $dep [expr $lv + 1]
   }
}
```

```
Neuzeichnen aller abhaengigen Widgets nach einem Schreibzugriff
NICHT extern aufrufen!

- Schritte:
     1. DAG2Level von den Blaettern aus durchlaufen, dabei aus globaler
        Variable Dependants die Widgets entnehmen, die vom aktuellen
        Attribut abhaengig sind
        -> Falls diese noch nicht in Redraw-List, anhaengen
     2. Neuzeichnen aller Widgets in Redraw-List

Struktur der Redraw-List stellt sicher, dasz jedes Widget
nur einmal neu gezeichnet werden musz, da sich nach dem Neu-
zeichnen eines Widgets die unterliegenden Attribute nicht
mehr aendern koennen

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.CallRedraw {} {
   global DAG2Level Dependants top.PermApply

   if {!${top.PermApply}} {
      return
   } else {
      set RedrawList {}
      for {set i $DAG2Level(max)} {$i>=0} {incr i -1} {
         foreach att $DAG2Level($i) {
            foreach dep $Dependants($att) {
               if {$dep != 0} {
                  if {[lsearch -exact $RedrawList $dep]==-1} {
                     lappend RedrawList $dep
                  }
               }
            }
         }
      }
      foreach dep $RedrawList {${dep}.Redraw}
   }
}
```

```
Ausloesen des Dependency-Mechanismus in 2 Schritten
nach einem Schreibzugriff auf eine Variable

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter:
     - element: Variable, auf die der Schreibzugriff erfolgte
```

```
proc Depend.Dirty {element} {
   global Dirty DAG2Level EnableDirty
   set Dirty(File) 1
   if {!$EnableDirty} {return}
   # Setze Wurzelknoten
   set DAG2Level(max) 0
   set DAG2Level(0) $element
   set EnableDirty 0
   Depend.SetDirty $element 1
   Depend.CallRedraw
   set EnableDirty 1
}
```

```
Erzwingen eines Redraw aller Widgets

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.ForceRedraw {} {
   global RW_Vars Dirty Dependants

   Menu.Status "Aktualisiere Abhängigkeiten..."
   foreach i $RW_Vars {
      foreach j $Dependants($i) {
         if {$j!="0"} {
            ${j}.Redraw
         }
      }
   }
   Menu.StatusDone
}
```

```
Synchronisieren der tcl-Variablen mit dem
netCDF-File (z.B. vor Save aufgerufen)

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.Sync {} {
   global KR2 Dirty RW_Vars netCDF WT

   Menu.Status "Überarbeite Datenabhängigkeiten..."

   # Variablen, die von anderen Variablen abhängig sind, vor diesen
   # synchronisieren, um Mehrfachaktualisierungen und Kreise im Aufruf-
   # Graphen zu vermeiden.
   # Wird durch Decrementschleife und Struktur der Variablen RW_Vars
   # sichergestellt.
   for {set i [expr [llength $RW_Vars] -1]} {$i>=0} {incr i -1} {
      set element [lindex $RW_Vars $i]
      if {$Dirty($element)==1} {
         $KR2($element)
         set Dirty($element) 0
      }
      $WT($element) $netCDF($element) $element
   }
   set netCDF(vh_Defaults) 0
   Menu.StatusDone
}
```

```
Erlauben des Redraw-Mechanismus

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.InitWidgetEnd {} {
   global EnableDirty
   set EnableDirty 1
   Menu.StatusDone
}
```

```
Routine zum Sperren des Redraw-Mechanismus waehrend
der Initialisierung eines Widgets

Notwendig, weil sonst der Dependency-Mechanismus
waehrend des in der Initialisierungsroutine aufgerufenen
Redraw dasselbe Redraw beim Schreibzugriff auf Variablen,
die Dirty sind, nochmals aufrufen wuerde

Der resultierende Kreis in den Traces wuerde vom tcl-Interpreter
erkannt und mit Fehlermeldung quittiert

WICHTIG! Am Ende der Init-Routine Depend.InitWidgetEnd aufrufen!

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.InitWidgetStart {} {
   global EnableDirty
   set EnableDirty 0
}
```

```
Setzen der Attribute, die berechnet werden koennen,
auf Dirty
 -> Erzwingen einer Neuberechnung
 -> Angewandt, wenn netCDF File erstmalig geladen wird,
    um Default-Werte mit berechneten Werten zu ueberschreiben
 -> gesteuert ueber netCDF-Attribut vh_Defaults

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.SetDefaultsDirty {} {
   global RW_Vars Dirty netCDF

   foreach i $RW_Vars {
      set Dirty($i) $netCDF(vh_Defaults)
   }
   set Dirty(Results) 1
   set Dirty(Distance) 1
   set Dirty(AUXResults) 1
   set Dirty(AUXDistance) 1
}
```

```
>>>>>>>>> Initialisierung <<<<<<<<<
```

```
Initialisieren der Variablen Dirty
 -> zeigt an, ob ein Attribut gueltig ist (0)
    oder nicht (1)
 -> wenn Dirty(att)=1, dann wird beim Lesezugriff
    auf netCDF(att) diese Variable durch
    Aufruf der zugehoerigen KR2 neu berechnet.

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.InitDirty {} {
   global Dirty RW_Vars RO_Vars netCDF

   foreach i $RW_Vars {set Dirty($i) $netCDF(vh_Defaults)}
   foreach i $RO_Vars {set Dirty($i) 0}

   set Dirty(File) 0

   set Dirty(Results) 1
   set Dirty(Distance) 1
   set Dirty(AUXResults) 1
   set Dirty(AUXDistance) 1
}
```

```
    >>>>>>>>> Routinen zur Berechnung von Datenaspekten (KR2) <<<<<<<<<

   >>>>>>>>> Hier werden zwei Routinen als Beispiel angeführt. <<<<<<<<<
```

```
Hilfsroutine zum Konvertieren der Abstandsvektoren
aus einer Liste (netCDF(Distance)) in ein Array
(Distance)

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.ConvertDistance {} {
   global netCDF

   # Return Variables
   global Distance

   if {[llength $netCDF(Distance)]==0} {
      set Distance(Error) 1
   } else {
      set Distance(Error) 0
      set Distance(Good_cnt) [lindex $netCDF(Distance) 0]
      set Distance(Good_Tech) [lindex $netCDF(Distance) 1]
      set Distance(Good_dist) [lindex $netCDF(Distance) 2]
      set Distance(Bad_cnt) [lindex $netCDF(Distance) 3]
      set Distance(Bad_Tech) [lindex $netCDF(Distance) 4]
      set Distance(Bad_dist) [lindex $netCDF(Distance) 5]
      set Distance(DistVect) [lindex $netCDF(Distance) 6]
      set Distance(m_neu) [lindex $netCDF(Distance) 7]
      set Distance(n_neu) [lindex $netCDF(Distance) 8]
   }
}
```

```
Berechnen der Abstandsvektoren als Liste

Letzte Aenderung: 18.7.94 (Uwe Rauschenbach)

Parameter: keine
```

```
proc Depend.Distance {} {
   global Dirty netCDF

   Menu.Status "Überarbeite Konflikte..."
   set m [expr $netCDF(selected_dims) + $netCDF(values_per_point)]
   set w 0
   for {set i 0} {$i<$netCDF(AllVars)} {incr i} {
      if {[lindex $netCDF(vis_var) $i] == 1} {
         if {[lindex $netCDF(ord) $i]=="ordinal"} {
            set w [expr $w + [lindex $netCDF(num_of_intervals) $i]]
         } else {
            set w [expr $w + [lindex $netCDF(num_of_values) $i]]
         }
      }
   }

   if {$m == 0} {
      set w 0
   } else {
      set w [expr $w / $m]
   }

   if {$m==0 || $netCDF(num_of_observations)==0} {
      set netCDF(Distance) ""
      Menu.StatusDone
      return 1
   }

   set params [Utils.Percent2Float $netCDF(upper_thresh)]
   lappend params [Utils.Percent2Float $netCDF(lower_thresh)]
   lappend params $netCDF(correlated_vars)
   lappend params $m
   lappend params $netCDF(num_of_observations)
   lappend params $w
   for {set i 1} {$i < 7} {incr i} {
      lappend params $netCDF(IG$i)
   }

   puts stdout "Params for CalcDist: $params"
   set err [catch "CalcDistCallback $params" returnList]

   Menu.StatusDone
   if {$err} then {
      puts stdout "Error computing advice - Aborting...";
      AlertBox "Fehler bei Empfehlungsberechnung: $returnList"
      return 1
   } else {
      return 0
   }
}
```

Anhang F:

Struktur des Helpfiles

Relevante Auszüge aus *vh.hlp*

Aufbau des Help-Files

```
<Helpfile> ::= <Index> <Helptext> <Explaintext>
```

Aufbau eines Indexeintrages

```
<Index> ::= <IdxStartLabel> <CR> <IdxName> <CR> <IdxText> <IdxEndLabel>
            <CR>

// Mit <string> kann in den entsprechenden Hilfeindex eingestiegen werden
<IdxStartLabel> ::= ".Idx_" <string>

// <string> wird als Ueberschrift des Index dargestellt
<IdxName> ::= <string>

<IdxText> ::= <IdxText1> {<IdxText1>}

// <string1> dient als Schluessel für Hilfetext und darf
// keine Leerzeichen enthalten
// <string2> ist Eintrag in der Liste des Hilfeindex
<IdxText1> ::= <string1> " " <string2> <CR>

<IdxEndLabel> ::= ".Idx_END"
```

Aufbau eines Hilfeeintrages

```
<HelpText> ::= <HelpEntry> {<HelpEntry>}

<HelpEntry> ::=   <HelpStartLabel> <CR> <HelpName> <CR> <HelpText1>
                  <HelpEndLabel> <CR>

// <string1> dient als Schluessel für Hilfetext (wie im Index)
<HelpStartLabel> ::= ".Help_" <string1>

// <string> wird als Titel der Hilfebox angezeigt
<HelpName> ::= <string>

<HelpText1> ::= {<string> <CR>}

<HelpEndLabel> ::= ".Help_END
```

Aufbau eines Erklärungseintrages

```
<ExplainText> ::= <ExplainEntry> {<ExplainEntry>}

<ExplainEntry> ::= <ExplainStartLabel> <CR> <ExplainName> <CR>
                   <ExplainText1> <ExplainEndLabel> <CR>

// <string1> dient als Schluessel für Erklaerungstext
<ExplainStartLabel> ::= ".Explain_" <string1>

// <string> wird als Titel der Erklaerungsbox angezeigt
<ExplainName> ::= <string>

<ExplainText1> ::= {<string> <CR>}

<ExplainEndLabel> ::= ".Explain_END
```

Beispiel (Dummy-Hilfeindex)

```
.Idx_Proto
Prototyp-Index
Proto-OVERVIEW Prototyp
.Idx_END
```

Beispiel (Dummy-Hilfetext)

```
.Help_Proto-OVERVIEW
Hilfe zum Prototyp

Dieser Hilfetext ist NOCH NICHT IMPLEMENTIERT.


Benutzen Sie Hilfe/Index, um alle implentierten
Hilfethemen zu sehen.
.Help_END
```

Bild zum Beispiel