INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen

Oettingenstrasse 67 D–80538 München

Ludwig
Maximilians-
Universität
München

Entwurf eines statistischen Systems zur Analyse graphischer Modelle

Angelika Elisabeth Blauth

Diplomarbeit

Beginn der Arbeit:	16. 12. 1997
Abgabe der Arbeit:	15. 06. 1998
Aufgabensteller:	Prof. Dr. François Bry
	Prof. Dr. Iris Pigeot
Betreuer:	Prof. Dr. François Bry
	Prof. Dr. Iris Pigeot
	Dr. Angelika Caputo

Erklärung

Hiermit versichere ich, daß ich alle Teile der Arbeit einschließlich der beschriebenen Software selbständig verfaßt habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

München, den 15.06.1998

Angelika Blauth

Zusammenfassung

Die Diplomarbeit beinhaltet den Entwurf und die teilweise Implementierung eines Programms zur Analyse graphischer Modelle. Sie entstand in Zusammenarbeit mit dem Institut für Statistik. Alle Teile der Diplomarbeit sind im World Wide Web unter der Adresse *http://www.pms.informatik.uni-muenchen.de/publikationen/* zu finden.
Die vorliegende Ausarbeitung behandelt nach einer kurzen Einführung zunächst die grundlegenden Konzepte graphischer Modelle, insoweit sie für den Entwurf eines Programms zur Analyse dieser Modelle von Relevanz sind. Der Entwurf des Programms erfolgt im objektorientierten Paradigma. Um seine Modellierung zu verdeutlichen, wird die *Unified Modelling Language* als Notation verwendet, deren Elemente in dem sich anschließenden Kapitel beschrieben werden. Die weiteren Kapitel stellen den realisierten Entwurf aus technischer wie aus der Sicht des Anwenders vor. Im letzten Kapitel wird die Arbeit zusammengefaßt und ein kurzer Ausblick gegeben. In einem Anhang werden grundlegende statistische Begriffe vorgestellt, die die Grundlagen graphischer Modelle ergänzen. Die restlichen Anhänge stellen wichtige Begriffe der Objektorientierung und der Unified Modelling Language übersichtlich dar.

Für die überaus gute Betreuung und die Schaffung einer äußerst angenehmen Atmosphäre möchte ich mich recht herzlich bedanken bei (in alphabetischer Reihenfolge):

François Bry, Angelika Caputo, Vanessa Didelez, Astrid Heinicke und Iris Pigeot.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Graphische Modelle stellen ein Teilgebiet der multivariaten Statistik dar. Sie werden beispielsweise in soziologischen und psychologischen Fragestellungen eingesetzt, in denen eine Vielzahl von Variablen gleichzeitig untersucht werden. Dabei interessiert man sich vor allem für die Zusammenhänge zwischen den betrachteten Variablen. In Anbetracht der meist großen Zahl an Variablen kann sich diese Zusammenhangsstruktur sehr komplex gestalten. Die Idee graphischer Modelle besteht nun in der Verbindung eines statistischen Modells der Zusammenhangsstruktur eines multivariaten Zufallsvektors und der Darstellung dieser Struktur in einem Graphen, wobei die betrachteten Variablen als Knoten und die Beziehungen zwischen den Variablen als Kanten dargestellt werden. Ein Forschungsschwerpunkt im Bereich graphischer Modelle ist die Entwicklung und Implementierung von Modellanpassungsstrategien, d.h. von Verfahren, mit denen ein graphisches Modell ausgewählt werden kann, das einen vorhanden Datensatz möglichst gut beschreibt. Die Modellanpassung ist im Rahmen graphischer Modelle bisher allerdings ein mühsames und zeitaufwendiges Unterfangen, da Zwischenergebnisse innerhalb der Modellanpassung nur textuell vorliegen und mehrere Programme bei der Modellanpassung verwendet werden müssen.

Das Thema dieser Arbeit ist der Entwurf eines Programms zur Analyse von Daten mit Hilfe graphischer Modelle. Dieses Programm soll vor allem zwei Dinge leisten: Zum einen die Durchführung der Modellanpassung und zum anderen die Visualisierung des zugehörigen Graphen zu verschiedenen Schritten der Analyse. Der Schwerpunkt der vorliegenden Arbeit liegt dabei auf folgenden Themen. Das Programm muß Anforderungen sowohl aus der Sicht der Informatik als auch aus der Sicht der Anwendung genügen. Da die Modellanpassung bei graphischen Modellen derzeit noch Gegenstand der Forschung ist, ist es notwendig, ein generisches System zu entwickeln. Dies bedeutet, daß sich im Programm mehrere Verfahren zur Modellanpassung integrieren lassen. Deshalb ist es gerade bei der Entwicklung dieses Programms im besonderen Maße notwendig, einen gut durchdachten Entwurf zu erstellen, der es erlaubt, das Programm in mehreren Richtungen zu erweitern. Ein weiterer wichtiger Aspekt betrifft die geeignete Visualisierung des Graphen, da gerade hierin die Vorteile einer computergestützten Modellanpassung zur Geltung kommen. Ein deutlich zunehmender Trend in Forschung und Praxis besteht gerade darin, gewonnene Ergebnisse nicht nur als Protokoll darzustellen, sondern sie geeignet zu visualisieren, wodurch eine rasche Erfassung dieser Ergebnisse ermöglicht werden soll. Ein Beispiel ist auch die Medizin, in der verschiedene Visualisierungstechniken angewandt werden, um Diagnosen besser und schneller erstellen zu können. Auch im Rahmen der Software–Entwicklung werden wichtige Zusammenhänge durch Diagramme dargestellt, wie in der vorliegenden Arbeit noch zu sehen ist. In der Statistik bieten gerade graphische Modelle eine Möglichkeit, komplexe Zusammmenhänge zwischen verschiedenen Merkmalen bzw. Variablen mittels eines Graphen überschaubar darzustellen. Da graphische Modelle insbesondere dann eingesetzt werden, wenn eine Vielzahl von Variablen erhoben wird, erhält man oft sehr große und damit unübersichtliche Graphen. Dieser Effekt ist natürlich dann kontraproduktiv zu dem eigentlichen Ziel, die Assoziationsstruktur der Variablen durch die Darstellung im Graphen zu veranschaulichen und auf einem Blick erfaßbar zu machen. Aus diesem Grund wird hier angestrebt, dem Benutzer unterschiedliche Eingriffsmöglichkeiten zur Verfügung zu stellen, die je nach Anforderungen eine möglichst gute und übersichtliche Darstellung des Graphen ermöglichen. Da das Programm auch von Benutzern mit eher geringen Kenntnissen auf dem Gebiet graphischer Modelle genutzt werden soll, ist auch ein umfangreiches Hilfesystem geplant, das mit der Bedienung des Programms und mit den statistischen Methoden vertraut macht und ihn so bei der statistischen Analyse unterstützt.

Innerhalb dieser Arbeit wurde das grundlegende Konzept eines solchen Programms erstellt und teilweise implementiert. Der Schwerpunkt der Arbeit liegt dabei auf einem umfassenden Design des Programms, das leicht zu erweitern und zu warten ist. Wichtige Teilaspekte sind dabei die Darstellung des Graphen in jedem Schritt der Modellanpassung sowie die Eingriffsmöglichkeiten in den Graphen von Seiten des Benutzers. Hier wurde Wert darauf gelegt, die Bedienung möglichst einfach und intuitiv zu gestalten.

Die Arbeit gliedert sich in acht Teile. In dem sich an die Einleitung anschließenden zweiten Kapitel werden die grundlegenden Konzepte graphischer Modelle erläutert. Dazu werden die wichtigsten Begriffe aus dem Bereich der Statistik und der Graphentheorie eingeführt, die für graphische Modelle von Bedeutung sind. Um überhaupt Aussagen über die Assoziationsstruktur der Variablen anhand eines Graphen treffen zu können, muß das statistische Modell bestimmten Anforderungen genügen, die ebenfalls in diesem Kapitel beschrieben werden. Außerdem erfolgt eine Kategorisierung der verschiedenen Arten von graphischen Modellen, und es wird eine Möglichkeit zur Modellanpassung schrittweise skizziert. Das Kapitel schließt mit einem Einblick in die aktuelle Forschung im Bereich graphischer Modelle, wobei kurz aufgezeigt wird, inwieweit diese in das zu erstellende Programm einfließt.
Kapitel 3 geht detailliert auf die Modellanpassung eines graphischen Modells ein. Es wird ein Algorithmus der hier relevanten Selektionsstrategie vorgestellt, der auch im Programm Verwendung findet. Das erfordert zunächst erneut eine Einführung grundlegender Begriffe. Danach erfolgt eine Beschreibung des Algorithmus, und es wird erläutert, wie sich die Ergebnisse des Algorithmus in einen Graphen übersetzen lassen. Der Ablauf des Algorithmus wird anhand eines Beispiels vertiefend dargestellt.

Das Programm wird mittels objektorientierter Analyse und Design erstellt. Die dabei gefundene Darstellung der Modellierung erfolgt in der "Unified Modelling Language", deren Notation in Kapitel 4 eingeführt wird. Hier erfolgt zunächst ein Einblick in objektorientierte Analyse und Design. Anschließend werden die einzelnen Elemente der Notation, in verschiedene Diagramme aufgeteilt, erläutert.

Die nächsten drei Kapitel befassen sich mit der Modellierung und der Benutzerführung des Programms. In Kapitel 5 wird das grundlegende Konzept und die grundlegende Bedienung des Programms dargelegt. Hierbei wird auf die Modellierung der Benutzerschnittstelle eingegangen. Dazu werden allgemein gültige Anforderungen an eine Benutzerschnittstelle genannt und erläutert, wie sie im Programm realisiert sind. Bei der Vorstellung der Modellierung des Programms wird zunächst der allgemeine Programmaufbau beschrieben. Von dort ausgehend werden die einzelnen Bestandteile des Programms genauer hervorgehoben. Zuletzt werden noch Fragen diskutiert, die in der vorliegenden Arbeit nicht explizit behandelt wurden und erste Lösungsvorschläge präsentiert.
Die Erstellung und Bedienung der graphischen Darstellung des statistischen Modells ist ein wichtiger Aspekt im Programm. In Kapitel 6 werden Realisation und Konzepte dieses Aspekts diskutiert. Hierbei wird zunächst die Modellierung der Elemente

des Graphen und dessen graphischer Darstellung vorgestellt. Es folgt ein Abschnitt über die Eingriffsmöglichkeiten, die ein Benutzer bei der Modellanpassung innerhalb des Graphen besitzt. An dieser Stelle wird auch die realisierte Benutzerführung des Graphen anderen denkbaren Ansätzen gegenübergestellt. Den Abschluß des Kapitels bilden die Erörterung von über die Arbeit hinausgehenden Fragen.
Kapitel 7 geht detailliert auf die Benutzerführung des Programms ein. Es werden die zur Erstellung eines graphischen Modells im Programm auszuführenden Schritte beschrieben und anhand von Screenshots verdeutlicht. Dieses Kapitel dient also als eine Art Benutzerhandbuch des Programms im jetzigen Entwicklungsstadium. Abschließend werden in Kapitel 8 die wichtigen und entscheidenden Punkte der Arbeit zusammengefaßt. Ebenfalls finden sich hier Anmerkungen zum Umgang mit der zur Verfügung gestellten Software sowie zum Entwicklungsprozeß insgesamt.
Im Anhang der Arbeit finden sich Ausführungen zu den Bereichen Statistik (Anhang A), Objektorientierung (Anhang B) und der UML–Notation (Anhang C).

Kapitel 2

Graphische Modelle

Die Theorie der graphischen Modelle ist ein Teilgebiet der multivariaten Statistik, d.h. die Problemstellung umfaßt stets mehrere Variablen gleichzeitig. Dabei werden bei graphischen Modellen speziell die Beziehungen zwischen allen Variablen untersucht, welche dann in einem Graphen dargestellt werden. Die Knoten des Graphen repräsentieren Variablen und die Kanten Beziehungen zwischen den Variablen. Zunächst besteht ein graphisches Modell neben dem Graphen noch aus der Spezifizierung der gemeinsamen Verteilung aller betrachteten Variablen, wie zum Beispiel der multivariaten Normalverteilung. Der Graph stellt dann bestimmte Eigenschaften dieser Verteilung dar. Dabei wird auch deutlich, daß der Graph in diesem Falle nicht, wie sonst häufig bei der Verwendung graphischer Darstellungen in der Statistik, die Daten selbst veranschaulicht, sondern ein zugrundeliegendes statistisches Modell. Einerseits werden die Abhängigkeiten zwischen Variablen, andererseits aber auch Unabhängigkeiten zwischen Variablen abgebildet.

In den folgenden Abschnitten wird der Leser mit dem Konzept graphischer Modelle vertraut gemacht und erhält einen Einblick in einen Teilbereich der aktuellen Forschung. Zu Beginn wird eine kurze informelle Einführung in die statistischen und graphentheoretischen Grundlagen graphischer Modelle gegeben. Es folgt ein Abriß über die verschiedenen Typen graphischer Modelle, die sich beispielsweise durch den zugrundeliegenden Graphen unterscheiden. Im Rahmen dieser Arbeit sind vor allem sogenannte graphische Kettenmodelle von Interesse, die deshalb besonders ausführlich dargestellt werden. In praktischen Fragestellungen steht man dabei zunächst dem Problem der Modellanpassung gegenüber, d.h. man möchte zu vorliegenden Daten einen Kettengraphen erstellen, der die in der Realität vorliegende Assoziationsstruktur möglichst genau widerspiegelt. Ein Verfahren zur schrittweisen Anpassung eines Kettengraphen wird von (Cox und Wermuth 1996) angedeutet und findet sich in Heinicke (1995) im Detail beschrieben. Da der schrittweise Aufbau eines Kettengraphen mit Hilfe eines Programms ein wichtiger Bestandteil dieser Arbeit ist, wird das zugrundeliegende Prinzip dazu anhand eines fortlaufenden Beispiels im nächsten Abschnitt behandelt. Zum Abschluß wird auf Erweiterungen graphischer Modelle eingegangen, die im Rahmen des Projekts, zu dem auch die vorliegende Arbeit gehört, angestrebt werden. Dabei wird auch aufgezeigt, wie diese in der zu erstellenden Software realisiert werden.

Der Schwerpunkt dieser Arbeit liegt nicht auf einer detaillierten Beschreibung der Theorie graphischer Modelle. Aus diesem Grund wird nur kurz in diese Thematik eingeführt. Die folgende Literaturliste soll aber dem interessierten Leser die Möglichkeit bieten, sich näher mit diesem Gebiet auseinanderzusetzen. Eine grundlegende Monographie zur Theorie graphischer Modelle ist (Lauritzen 1996). Sie gibt einen guten Überblick und erläutert zudem ausführlich den mathematisch theoretischen Hintergrund. Ein eher anwendungsorientiertes Buch ist dagegen (Edwards 1995), das sich hauptsächlich mit *ungerichteten Graphen* beschäftigt. Zudem bietet es eine Beschreibung des Programmes MIM, das ein System zur Analyse hierarchischer Modelle bietet. Ebenfalls anwendungsorientiert ist (Cox und Wermuth 1996). Darüber hinaus beschäftigt sich dieses Buch auch allgemein mit der Problematik statistischer Modellbildung. Viele Beispiele finden sich in (Whittaker 1991), wobei der graphentheoretische Kontext von graphischen Modellen dort nur angedeutet wird. Als erste Lektüre, die auch die Ereignisdatenanalyse im Kontext graphischer Modelle behandelt, ist schließlich (Vermunt 1997) zu nennen.

2.1 Grundlegende Konzepte

2.1.1 Wichtige Begriffe aus der Statistik und der Graphentheorie

Zuerst wird ein informeller Überblick über einige wichtige Begriffe aus der Statistik gegeben. Die formalen Definitionen finden sich in Anhang A und in den jeweils angegeben Literaturverweisen. Ein grundlegender Begriff in der Statistik ist die *Dichte*. Diese beschreibt die Verteilung eines zu untersuchenden Merkmals anhand einer nicht–

negativen Funktion. Eine formale Definition findet sich in (Fahrmeir, Künstler, Pigeot und Tutz 1997) S. 85. Ein wichtiges Konzept graphischer Modelle ist die statistische Unabhängigkeit. Zwei Variablen X und Y heißen *statistisch unabhängig*, wenn die Wahrscheinlichkeit dafür, daß X eine bestimmte Ausprägung annimmt, nicht von der Ausprägung von Y abhängt und umgekehrt. Werden mehr als zwei Zufallsvariablen betrachtet, so muß das obige Konzept erweitert werden. Dazu wird u.a. der Begriff der bedingten Unabhängigkeit eingeführt. Es seien zum Beispiel drei Zufallsvariablen X, Y und Z gegeben. Wenn X und Y für jedes $Z = z$ unabhängig sind, sagt man, daß X und Y *bedingt unabhängig* sind gegeben Z.

$$\text{Hierfür schreibt man } X \perp Y \mid Z.$$

Betrachtet man eine Menge V von Zufallsvariablen, sagt man, daß zwei Variablen X und Y *marginal unabhängig* sind, wenn X und Y unabhängig sind. Dabei werden die Einflüsse der übrigen Variablen aus V nicht beachtet. So kann es sein, daß bei drei Variablen X, Y und Z sowohl X und Y als auch Y und Z sowie X und Z marginal unabhängig sind, aber alle drei zusammen betrachtet abhängig voneinander sind.
Es kann auch der Fall auftreten, daß bei einer Betrachtung von nur zwei Zufallsvariablen X und Y eine marginale Abhängigkeit zwischen diesen beiden gefunden wird, aber bei Aufnahme einer weiteren Variable Z in die Analyse eine Unabhängigkeit zwischen X und Y gegeben Z, also $X \perp Y \mid Z$, festgestellt wird. Ebenso ist es möglich, daß eine zuvor positive Beziehung, d.h. große Werte von X ziehen große Werte von Y nach sich, bei der Hinzunahme anderer Variablen das Vorzeichen wechselt, hier also große Werte von X mit kleinen Werten von Y einhergehen. Andererseits kann es aber auch vorkommen, daß eine in das Modell mit aufgenommene Variable bedingte Abhängigkeiten im Modell indiziert, die eigentlich nicht vorhanden sind. Phänomene dieser Art sind unter dem Begriff *Simpsonsches Paradoxon* bekannt. Solchen irreführenden Ereignissen kann nur dadurch begegnet werden, indem bei der Auswertung von Studien alle relevanten Variablen gemeinsam betrachtet werden, da es sonst zu Verzerrungen in der Interpretation kommen kann. Dieser Sachverhalt wird anhand eines Beispiels aus (Edwards 1995) S.8f, verdeutlicht.

Beispiel 2.1 *In einer fiktiven Studie wird untersucht, ob Frauen bei der Zulassung zu amerikanischen Universitäten benachteiligt werden. Tabelle 2.1 zeigt (fiktive) Daten für eine ausgewählte Universität. Es scheint hier eine Benachteiligung von Bewerberinnen zu geben, da nur 41,6% von ihnen zugelassen werden, während im Gegensatz dazu 58,3% der männlichen Bewerber einen Studienplatz erhalten.*

Geschlecht	Anz. Bewerber	Anz. Zulassungen	Anteil Zulass. in %
männlich	600	350	58.3
weiblich	600	250	41.6

Tabelle 2.1: Bewerber unterteilt nach Geschlecht und Zulassung

Betrachtet man die Häufigkeitstabelle allerdings aufgeschlüsselt nach verschiedenen Fachbereichen, so zeigt sich die in Tabelle 2.2 wiedergegebene Situation.

Fachbereich	Geschlecht	Anz. Bewerber	Anz. Zulassungen	Anteil Zulass. in %
I	männlich	100	25	25
	weiblich	300	75	25
II	männlich	200	100	50
	weiblich	200	100	50
III	männlich	300	225	75
	weiblich	100	75	75

Tabelle 2.2: Bewerber unterteilt nach Geschlecht, Fachbereich und Zulassung

Innerhalb der Fachbereiche ist nun die Zulassungsrate für Männer und Frauen gleich. Dies würde also der obigen Aussage widersprechen, daß Bewerberinnen bei den Zulassungen für Universitäten benachteiligt werden. Der Unterschied zu der obigen Tabelle 2.1 läßt sich damit erklären, daß sich die Fachbereiche sehr in den Zulassungsraten unterscheiden und sich Männer eher in den Fachbereichen mit höheren Zulassungsraten bewerben.

Es folgt nun eine Zusammenstellung von Begriffen aus der Graphentheorie, die im Rahmen graphischer Modelle relevant sind.
Ein *Graph* $\mathcal{G} = (V, E)$ besteht aus einer endlichen Menge V von *Knoten* (engl. vertices) und einer endlichen Menge E von *Kanten* (engl. edges). Dabei ist $E \subseteq V \times V$. Ein Graph heißt *einfach*, wenn höchstens eine Kante zwischen zwei verschiedenen Knoten existiert. In den folgenden Erläuterungen sind Kanten der Form (η, η) nicht zugelassen. Kanten $(\eta, \lambda) \in E$, für die gilt, daß stets auch (λ, η) in E enthalten sind, heißen *ungerichtet*; Kanten, für die gilt, daß nur (η, λ) oder (λ, η) in E enthalten sind, heißen *gerichtet*. Besteht ein Graph nur aus gerichteten bzw. nur aus ungerichteten Kanten, nennt man ihn gerichtet bzw. ungerichtet. In einer zum Graphen gehörigen Darstellung

wird eine gerichtete Kante durch einen *Pfeil*, eine ungerichtete Kante durch eine *Linie* dargestellt.
Gibt es einen Pfeil von η nach λ, nennt man η ein *Elternteil* von λ und λ ein *Kind* von η. Die Menge der Elternteile von λ wird mit $pa(\lambda)$ (engl. parents) bezeichnet, die Menge der Kinder von η mit $ch(\eta)$ (engl. children). Sind zwei Knoten λ und η durch eine Linie verbunden, so heißen beide *adjazent* oder *benachbart*; sind sie hingegen weder durch einen Pfeil noch durch eine Linie verbunden, heißen sie *non–adjazent*. Die Menge aller Nachbarn von η wird mit $ne(\eta)$ (engl. neighbours) bezeichnet.
Die *Umgebung* $bd(\Psi)$ (engl. boundary) einer Teilmenge von Knoten Ψ, ist die Menge der Knoten aus $V \setminus \Psi$, die Elternteile oder Nachbarn zu Knoten in Ψ sind. Es gilt: $bd(\Psi) = pa(\psi) \cup ne(\psi)$ für alle $\psi \in \Psi$. Der *Abschluß* $cl(\Psi)$ (engl. closure) von Ψ ist die Menge Ψ vereinigt mit ihrer Umgebung.
Unter einem *Pfad* der Länge n von η nach λ versteht man eine Folge von $\eta = \eta_0, \ldots, \eta_n = \lambda$ Knoten, so daß gilt $(\eta_{i-1}, \eta_i) \in E, i = 1, \ldots, n$. Eine Menge S von Knoten aus V heißt (η, λ)–Separator, falls alle Pfade von η nach λ durch S gehen. S nennt man dann auch *separierende Menge* oder auch *trennende Menge* von $A, B \subseteq V$, falls es für alle $\eta \in A$ und $\lambda \in B$ einen (η, λ)–Separator gibt .
Knoten η, für die es einen Pfad von η nach λ gibt, aber keinen von λ nach η, heißen *Ahnen* $an(\lambda)$ (engl. ancestors) von λ. Die *Nachkommen* $de(\eta)$ (engl. determinants) von η bezeichnen Knoten λ, zu denen ein Pfad von η nach λ führt, aber nicht umgekehrt. Die *Nicht–Nachkommen* werden mit $nd(\eta)$ (engl. non–determinants) bezeichnet, und es gilt: $nd(\eta) = V \setminus (de(\eta) \cup \{\eta\})$.
Tabelle 2.3 gibt einen Überblick über die verwendeten Begriffe der Graphentheorie.

Bezeichnung	**Engl.**	**Bedeutung**
$pa(\lambda)$	parents	Eltern eines Knotens λ
$ch(\lambda)$	children	Kinder eines Knotens λ
$ne(\lambda)$	neighbours	Nachbarn eines Knotens λ
$bd(\Psi)$	boundary	Umgebung einer Knotenmenge Ψ
$cl(\Psi)$	closure	Abschluß einer Knotenmenge Ψ
$an(\Psi)$	ancestors	Ahnen einer Knotenmenge Ψ
$de(\Psi)$	descendants	Nachkommen einer Knotenmenge Ψ
$nd(\Psi)$	non–descendants	Nicht–Nachkommen einer Knotenmenge Ψ

Tabelle 2.3: Begriffe aus der Graphentheorie

2.1.2 Darstellung und Eigenschaften graphischer Modelle

In einem graphischen Modell werden Assoziationen zwischen den Variablen mit Hilfe eines Graphen dargestellt. Dabei werden die Zufallsvariablen im Graphen durch Knoten und ihre paarweisen Assoziationen durch Kanten repräsentiert. Genauer werden die zugrundeliegenden Variablen dabei folgendermaßen abgebildet: Diskrete Variablen werden durch einen Punkt dargestellt und stetige durch einen Kreis (im Englischen: **d**iscrete – **d**ot, **c**ontinuous – **c**ircle). Zudem werden symmetrische Assoziationen zwischen Variablen durch eine Linie wiedergegeben und einseitige Assoziationen durch einen Pfeil von der erklärenden zur erklärten Variable.

Dabei werden drei verschiedene Arten von Graphen unterschieden, von denen zwei bereits erwähnt wurden: ungerichtete und gerichtete Graphen. Eine dritte Gruppe sind Kettengraphen, die aus Elementen der anderen zwei Arten bestehen, d.h. die sowohl gerichtete als auch ungerichtete Kanten besitzen.
Zudem kann man noch eine weitere Einteilung und zwar in *Konzentrations–* und *Kovarianzgraphen* vornehmen. Dabei repräsentieren fehlende Kanten im Konzentrationsgraphen bedingte Unabhängigkeiten und im Kovarianzgraphen marginale. Der jeweilige Typ wird in der graphischen Darstellung durch die Art der Kanten deutlich gemacht: Bei Konzentrationsgraphen werden sie durch eine durchgezogene Linie, bei Kovarianzgraphen gestrichelt dargestellt. Die folgenden Überlegungen zu graphischen Modellen beziehen sich auf Konzentrationsgraphen, da diese in der Fachliteratur ausführlich behandelt werden. Sie gelten aber auch in ähnlicher Weise für Kovarianzgraphen.

Um Aussagen über Unabhängigkeiten aus einem Graphen entnehmen zu können, müssen bestimmte Eigenschaften an das zugehörige statistische Modell gestellt werden. Insbesondere müssen dazu in den einzelnen Modellen die verschiedenen Markov–Eigenschaften erfüllt und äquivalent sein. Da in dieser Arbeit hauptsächlich Kettengraphen behandelt werden, wird nur ihre Darstellung ausführlich beschrieben. Das Prinzip der Markov–Eigenschaften wird allerdings anhand von ungerichteten Graphen vorgestellt, da dieses hier sehr anschaulich formuliert werden kann. Die Übertragung dieser Eigenschaften auf Kettengraphen wird dann nur noch kurz angeschnitten.

Ungerichtete Graphen Die Markov–Eigenschaften für einen Graphen $\mathcal{G} = (V, E)$ lauten:
Ein *Wahrscheinlichkeitsmaß* P auf $\mathcal{X}$ erfüllt

(P) die *paarweise Markov–Eigenschaft* bezüglich $\mathcal{G}$, falls für jedes Paar (η, λ) non–adjazenter Knoten gilt:

$$\eta \perp \lambda \mid V \setminus \{\eta, \lambda\};$$

(L) die *lokale Markov–Eigenschaft* bezüglich $\mathcal{G}$, falls für jeden Knoten $\eta \in V$ gilt:

$$\eta \perp V \setminus cl(\eta) \mid bd(\eta);$$

(G) die *globale Markov–Eigenschaft* bezüglich $\mathcal{G}$, falls für jedes Tripel (A, B, S) disjunkter Teilmengen von V, wobei S trennende Menge von A und B ist, gilt:

$$A \perp B \mid S.$$

Für jeden ungerichteten Graphen $\mathcal{G}$ und für jede Wahrscheinlichkeitsverteilung P auf $\mathcal{X}$ gilt die folgende Beziehung zwischen den obigen drei Markov–Eigenschaften:

$$(G) \Rightarrow (L) \Rightarrow (P).$$

Um aber Aussagen über die Unabhängigkeit verschiedener Variablen anhand des Graphen treffen zu können, muß zusätzlich auch die andere Implikationsrichtung gelten, also insgesamt:

$$(G) \Leftrightarrow (L) \Leftrightarrow (P).$$

Dies ist bei ungerichteten Graphen der Fall, wenn P eine stetige und positive Dichte besitzt.

Beispiel 2.2 *Gegeben sei ein Modell für vier stetige Variablen W, X, Y und Z, bei dem die paarweise bedingten Unabhängigkeiten $W \perp Z \mid (X, Y)$ und $X \perp Z \mid (W, Y)$ gelten. Das führt zu den in Abbildung 2.1 dargestellten Graphen, in dem die Kanten zwischen W und Z und zwischen X und Z fehlen.*
Im Graphen werden die Knotenmengen $A_1 = \{W\}$ und $B_1 = \{Z\}$ durch die Menge $S_1 = \{Y\}$ separiert. Außerdem wird $A_2 = \{X\}$ von $B_2 = \{Z\}$ durch $S_2 = \{Y\}$ separiert. Daraus kann man aufgrund der globalen Markoveigenschaft schließen, daß auch $W \perp Z \mid Y$ und $X \perp Z \mid Y$ gilt. Die Markov–Eigenschaften erlauben damit eine Übertragung von einer graphentheoretischen Eigenschaft, Separation, zu einer statistischen Eigenschaft, bedingter Unabhängigkeit.

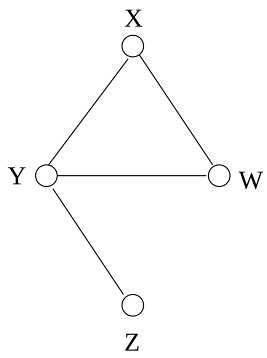


Abbildung 2.1: Beispiel für einen ungerichteten Graphen

Das folgende Beispiel ist Whittaker (1991, S. 1–6) entnommen.

Beispiel 2.3 *Es wird untersucht, ob zwischen den Prüfungsnoten von Studenten in den einzelnen Teilgebieten der Mathematik ein Zusammenhang besteht. Hierzu wurden die Noten von 88 Studenten in den Fächern Mechanik, Vektorrechnung, Algebra, Analysis und Statistik verglichen. Es ergab sich dabei das nachstehende graphische Modell:*

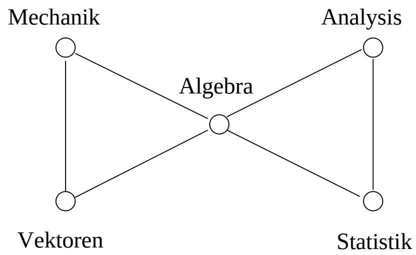


Abbildung 2.2: Graphisches Modell für die Ergebnisse der Prüfungen in den Teilgebieten der Mathematik

Aus dem Graphen kann man nun mit Hilfe der globalen Markov–Eigenschaft ablesen, daß die Noten in Mechanik und Vektorrechnung bedingt unabhängig von den Noten in Analysis und Statistik sind, wenn die Note in Algebra gegeben ist. Zudem zeigt er an, daß zum Beispiel die Noten in Algebra und Analysis allein ausreichen, um eine Statistiknote vorherzusagen, daß aber alle vier Noten der anderen Fächer herangezogen werden müssen, um die Algebranote vorherzusagen.

Kettengraphen Bei Kettengraphen werden die Kanten in numerierte paarweise disjunkte Teilmengen $S_1 \dot\cup S_2 \dot\cup \dots \dot\cup S_k = V$ aufgeteilt. Die Teilmengen $S_i, i = 1, \dots k$,

werden *Kettenglieder* genannt. Variablen im selben Kettenglied haben eine symmetrische Assoziationsstruktur, weshalb vorliegende Assoziationen durch Linien dargestellt werden. Die Kettenglieder selbst hingegen sind hinsichtlich ihrer asymmetrischen Assoziationsstruktur geordnet, wobei S_{i+1} auf S_i folgt für $i = 1, \ldots, k-1$. Assoziationen zwischen Variablen in verschiedenen Gliedern werden durch Pfeile in Richtung der erklärten Variable dargestellt.
Die Einteilung in verschiedene Kettenglieder entspricht der Aufteilung der Variablen in primäre Zielgrößen, verschiedene Stufen von Intermediaten und primäre Einflußgrößen. Intermediaten sind Variablen, die Zielgrößen und Einflußgrößen zugleich sind. Es ist dabei allgemeine Konvention, daß primäre Zielgrößen ganz links im Graph stehen und primäre Einflußgrößen ganz rechts. Die Intermediaten liegen dazwischen und sind so angeordnet, daß sie Zielgrößen für die rechts von ihnen angeordneten Variablen und Einflußgrößen für die links von ihnen stehenden Variablen sind. Um die verschiedenen Stufen der Variablen kenntlich zu machen, werden Variablen auf derselben Stufe in Kästen zusammengefaßt.

Beispiel 2.4 *Abbildung 2.3 zeigt ein Beispiel für einen Kettengraphen:*

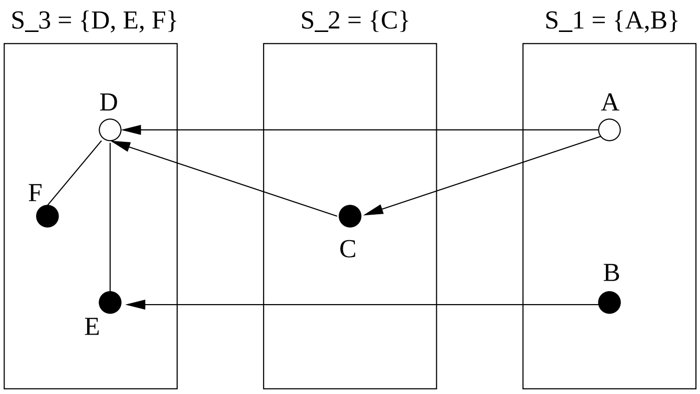


Abbildung 2.3: Beispiel für einen Kettengraphen

Dieser Graph besteht aus den stetigen Variablen A und D und aus den diskreten Variablen B, C, E und F. Dabei sind $A, B \in S_1$ die primären Einflußgrößen und $D, E, F \in S_3$ die primären Zielgrößen. Es gibt nur eine Stufe von Intermediaten, nämlich das Kettenglied S_2, welches die Variable C enthält.

Wenn die gegebene Modellverteilung die entsprechenden Markov–Eigenschaften für Kettengraphen erfüllt, können aus dem Graphen die bedingten Unabhängigkeiten von

Variablen entnommen werden. Zwei Variablen sind genau dann bedingt unabhängig, wenn keine Verbindung, also weder Pfeil noch Linie, zwischen ihnen existiert. Die bedingenden Variablen setzen sich aus den weiter rechts stehenden Variablen und den restlichen Variablen des Kastens der betrachteten Variable, die am weitesten links steht, zusammen. Im Beispielgraphen aus Abbildung 2.2 fehlt die Verbindung zwischen den Variablen E und F. Dies bedeutet, daß E und F bedingt unabhängig sind, gegeben die Variablen A, B, C und D. Ebenso fehlt die Verbindung zwischen B und C. C ist die weiter links stehende Variable. Da sie die einzige im Rechteck ist, gilt, daß B und C bedingt unabhängig sind gegeben A. Auch die Verbindung zwischen A und B fehlt. Das heißt A und B sind bedingt unabhängig gegeben die leere Menge, da A und B ganz rechts stehen. Dies wiederum bedeutet, daß A und B auch marginal unabhängig sind. Enthält der Graph sowohl stetige als auch diskrete Variablen, so ist eine Modellverteilung, die die Markov–Eigenschaften für Konzentrationsgraphen erfüllt, die sogenannte *Conditional Gaussian Verteilung*, kurz CG–Verteilung (siehe (Lauritzen und Wermuth 1989))

Sind die Assoziationen bestimmter Variablen nicht von Interesse, oder sollen sie in der Analyse nicht oder erst später berücksichtigt werden, so besteht die Möglichkeit, dies durch einen doppelt umrandeten Kasten deutlich zu machen. Im Beispielgraphen aus Abbildung 2.2 tritt der Fall nicht auf. Diese Art der Darstellung wird aber im Lauf der Graphentwicklung benötigt und dort genauer beschrieben (s. Abschnitt 2.2).

2.2 Skizzierung der schrittweisen Erstellung eines Kettengraphen

In praktischen Situationen liegen zunächst die beobachteten Daten vor aus denen der Kettengraph im Rahmen einer Modellanpassung sukzessive aufgebaut werden soll. Will man nun einen Kettengraphen erstellen, so sind zuerst Entscheidungen hinsichtlich der Variablen, die in das Modell miteinbezogen werden sollen, zu treffen. Diese Entscheidungen basieren auf Fachwissen aus dem entsprechenden Bereich der Studie und sind demnach inhaltlich begründet. So muß entschieden werden, welche Variablen überhaupt in das Modell aufzunehmen und wie sie grundlegend anzuordnen sind. Das Ergebnis einer solchen Anordnung wird *Abhängigkeitskette* genannt.

Die Wahl der Variablen hat dabei große Auswirkung auf das zu erstellende Modell, da, wie in Abschnitt 2.1.1 beschrieben, die Nichtbeachtung wichtiger Variablen zu irreführenden Aussagen führen kann. Eine Lösung dieses Dilemmas besteht jedoch nicht darin, möglichst viele Variablen in die Analyse aufzunehmen, da das Modell dann nur sehr schwer interpretierbar ist. Außerdem kann dann ein Fall des Simpsonschen Paradoxons eintreten: die hinzugekommenen Variablen begründen eine Abhängigkeit im Modell, die eigentlich nicht vorhanden ist.
Weiter legt die Anordnung der Variablen in einer Abhängigkeitskette die gesamte nachfolgende statistische Modellierung fest. So kann zwar die Struktur der Kette im Laufe der Analyse zumindest beim Konzentrationsgraphen weiter aufgebrochen werden; es ist jedoch nicht möglich, die Position der Variablen grundlegend zu ändern, da die geänderte Abhängigkeitskette eine komplette Neuberechnung zur Folge hätte.
Zudem müssen Überlegungen angestellt werden, welche Art von Unabhängigkeiten modelliert werden sollen, also bedingte oder marginale. Je nachdem wird ein Konzentrationsgraph oder ein Kovarianzgraph erstellt. Dabei ist zu berücksichtigen, daß ein Kovarianzgraph nicht immer berechnet werden kann, da geeignete statistische Verteilungen dafür noch ein Gegenstand aktueller Forschung sind (s. Abschnitt 2.3).
Im folgenden wird die Konstruktion eines Kettengraphen anhand eines fortlaufenden Beispiels näher erläutert. Die Beschreibung des Beispiels lehnt sich dabei an die Darstellung in (Pigeot und Streit 1994) an.

Beispiel 2.5 *Dem Beispiel liegt eine Studie zugrunde, die sich mit psychologischen Ursachen für einen Studienabbruch beschäftigt. Nach gründlichen Voruntersuchungen wurden folgende Variablen in das Modell aufgenommen, welche zur besseren Übersicht in Tabelle 2.4 angeordnet sind.*
Die in Abbildung 2.4 dargestellte Abhängigkeitskette wurde anschließend so formuliert, daß sie mit Erfahrungen und Ergebnissen aus früheren Studien verträglich ist und zudem den zeitlichen Ablauf widerspiegelt.
Für die postulierte Abhängigkeitskette soll nun ein Konzentrationsgraph erstellt werden.

Nach der Festlegung der Abhängigkeitskette, beginnt die Konstruktion des Kettengraphen. Dazu werden Regressionsanalysen für die einzelnen Variablen der Abhängigkeitskette durchgeführt. Die Berechnung verläuft dabei im Kovarianzgraphen etwas anders als im Konzentrationsgraphen, wie unten näher ausgeführt wird.
Im Konzentrationsgraphen werden die einzelnen Variablen der Reihe nach in Abhängigkeit zu allen Variablen gesetzt, die weiter rechts in der Abhängigkeitskette ste-

Variable	**Abkürzung**
Studienabbruch	AB
Gruppenintegration	GI
Motivation	M
Leistung nach Selbsteinschätzung	L
Wechsel der weiterführenden Schule	WWS
Wiederholung einer Klasse in der weiterführenden Schule	W
Integration in die Klasse der weiterführenden Schule	I
Wechsel der Grundschule	WGS
Bildungsniveau des Vaters	BV

Tabelle 2.4: Variablen im Modell

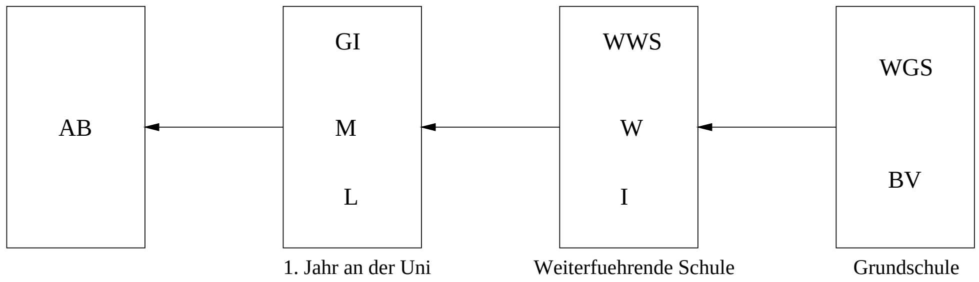


Abbildung 2.4: Abhängigkeitskette für die Determinanten eines Studienabbruchs

hen und auch zu denen, die in derselben Komponente enthalten sind. Hierbei kommen bekannte Regressionstechniken zum Zug. Mit einer geeigneten Selektionsstrategie wird versucht, in den jeweiligen Regressionen mit möglichst wenig erklärenden Variablen die Struktur der Daten zu erfassen. Mit Hilfe statistischer Variablenselektionsverfahren werden aus der Menge der potentiellen erklärenden Variablen die ausgewählt, die einen statistisch bedeutsamen Einfluß auf die jeweilige Zielgröße haben. Jede dieser Variablen wird im Graphen mit der betrachteten Zielgröße durch eine Kante (das kann ein Pfeil oder eine Linie sein) verbunden. Jede Kante entspricht einer bedingten Abhängigkeit. Man ist also daran interessiert, mit möglichst wenig bedingten Abhängigkeiten auszukommen, um die Zielvariablen zu erklären. Die Analyse beginnt mit den primären Zielgrößen als Zielvariablen und wird schrittweise bis zu den primären Einflußgrößen durchgeführt. Sind dabei mehrere Variablen in demselben Kettenglied enthalten, wird pro Variable ein Modell aufgestellt, um so auch die Abhängigkeitsstruktur innerhalb

eines Kettenglieds zu analysieren. Dies führt zu einem System von Regressionsmodellen, das unter hier nicht näher spezifizierten Annahmen insgesamt die Struktur der Daten in dem gerade untersuchten Kettenglied darstellt.
Prinzipiell findet dieser Ablauf auch bei der Erstellung eines Kovarianzgraphen statt. Im Unterschied zum Konzentrationsgraphen werden hier aber die Variablen nur in Abhängigkeit zu den weiter rechts stehenden Variablen gesetzt. Variablen in der gleichen Komponente werden zunächst nicht berücksichtigt. Die Analyse der Zusammenhänge in einer Komponente wird separat in einem eigenen Schritt vorgenommen. Zusätzlich wird jeder Schritt der Analyse durch fachliches Wissen genau reflektiert, da die statistischen Verfahren nur eine Hilfe sind, um adäquate Ergebnisse zu erzielen, aber alleine nicht ausreichen. So sollte man auch ggf. Änderungen am Graphen vornehmen, wenn die berechnete Struktur im Widerspruch zu früheren Erfahrungen im Forschungsbereich steht.

Beispiel 2.6 *(Fortsetzung von Beispiel 2.5) Bei der Studie über die Beweggründe für einen Studienabbruch ist die primäre Zielgröße die Variable AB und steht somit in der Abhängigkeitskette am weitesten links. Sie wird deshalb zuerst in Abhängigkeit zu den restlichen Variablen gesetzt, was in Abbildung 2.5 dargestellt ist. Der doppelt umrandete Kasten weist dabei darauf hin, daß die Abhängigkeitsstruktur zwischen den Variablen innerhalb dieses Kastens in dieser Stufe der Analyse nicht von Interesse ist.*

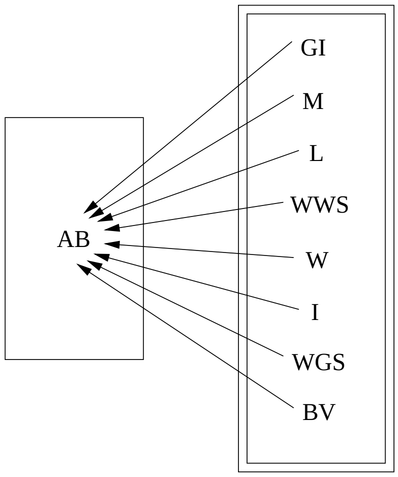


Abbildung 2.5: Vor Regressionsanalyse auf AB

Im Laufe der Regressionsanalyse wird festgestellt, daß nur die Variablen Gruppenintegration (GI), Leistung nach Selbsteinschätzung (L) und die Wiederholung einer Klasse *in der weiterführenden Schule den Studienabbruch direkt beeinflussen, was zu dem Graphen führt, der in Abbildung 2.6 dargestellt ist.*

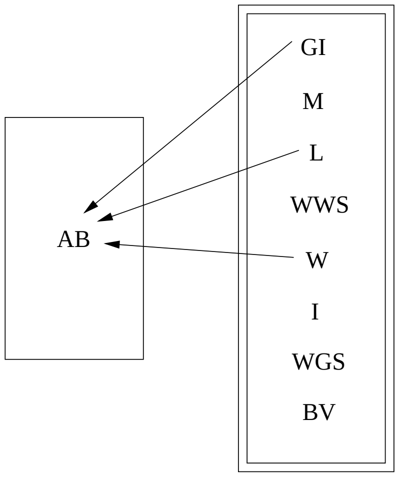


Abbildung 2.6: Ergebnis der Regressionsanalyse auf AB

Im nächsten Schritt werden nun die Variablen im Glied "1. Jahr an der Uni" betrachtet. Es entstehen dabei drei Regressionsmodelle mit jeweils Gruppenintegration (GI), Motivation (M) und Leistung nach Selbsteinschätzung (L) als Zielvariablen und allen anderen Variablen als mögliche Einflußgrößen. Hier ergibt sich das in Abbildung 2.7 gezeigte Modell, wobei die einseitige Abhängigkeitsstruktur zwischen GI, M und L nicht direkt aus der Analyse gefolgert werden kann, sondern auf Erfahrungen im Bereich der Untersuchung beruht.

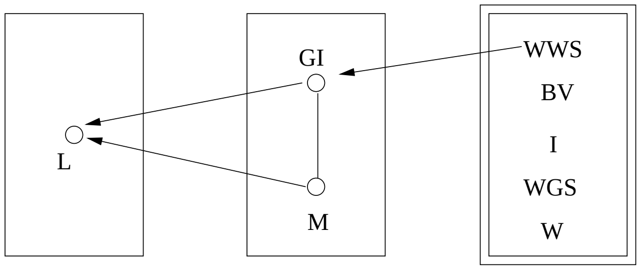


Abbildung 2.7: Abhängigkeitsstruktur der Universitätsfaktoren

Nun werden die erhaltenen Ergebnisse aus den obigen Schritten zusammengesetzt und man erhält Abbildung 2.8.

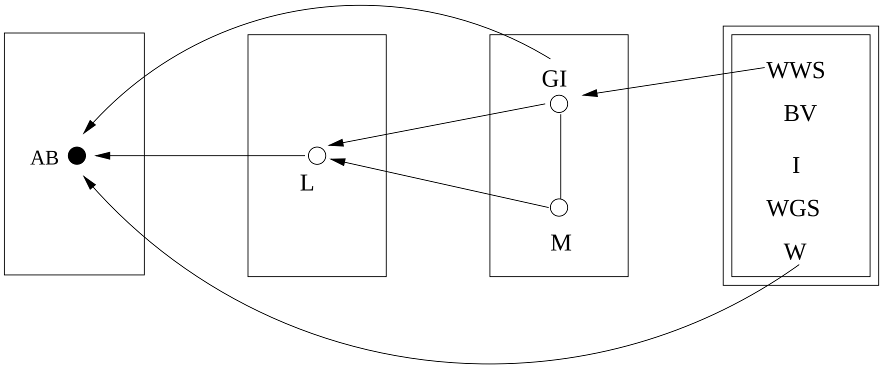


Abbildung 2.8: Kettengraph nach Untersuchung der Universitätsfaktoren

Indem die vorgestellten Prinzipien auch auf die restlichen Komponenten angewendet werden, ergibt sich das in Abbildung 2.9 gezeigte Gesamtbild.

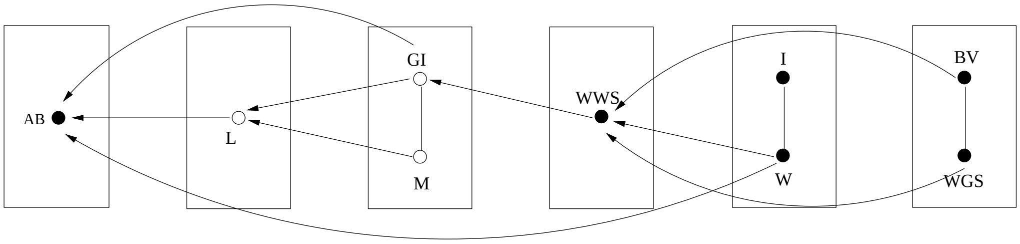


Abbildung 2.9: Kettengraph für die Determinanten eines Studienabbruchs

Man kann sehen, daß statistisch bedeutsame Gründe für einen Studienabbruch die Leistung nach Selbsteinschätzung, die Integration in die Gruppe in der Universität und die Tatsache, ob eine Klasse in der weiterführenden Schule wiederholt wurde, sind. Hingegen gibt z.B. die Integration in der Klasse einer weiterführenden Schule keinen Hinweis darauf, ob das Studium abgebrochen wurde.

2.3 Einblick in die aktuelle Forschung im Bereich graphischer Modelle

Ein aktueller Forschungsbereich ist die Suche nach alternativen Verteilungen, die in bestimmten Situationen angebrachter sind als zum Beispiel die CG–Verteilung. Ein solcher Bereich ist die multivariate Ereignisanalyse, in der beispielsweise Verweildauern

in bestimmten Zuständen oder Wartezeiten auf Ereignisse die interessierenden Variablen sind. Verweildauervariablen besitzen Verteilungen, deren Wertebereich auf positive reelle Zahlen beschränkt ist. Beispiele hierfür sind die Gamma– oder die Weibullverteilung. Über die CG–Verteilung hinausgehende Verteilungsannahmen würden den Einsatzbereich graphischer Modelle entscheidend erweitern. Außerdem sind bis jetzt in der Literatur nur wenige Verteilungen bekannt, die die Analyse marginaler Unabhängigkeiten, also die Bildung von Kovarianzgraphen, erlauben. Eine Verteilungsklasse, die von (Koehler und Symanowski 1995) diskutiert wird, leistet dies, wie (Caputo 1998) zeigt. Mit dieser Verteilungsklasse ist es zum einen prinzipiell möglich, Kovarianzgraphen zu erstellen, und zum anderen erweitert sie den Einsatzbereich graphischer Modelle auf die Analyse von Ereignisdaten.
Allerdings ist die Verteilungsklasse immun gegen das Simpsonsche Paradoxon, d.h., daß die Phänomene des Simpsonschen Paradoxons nicht in Erscheinung treten können. Dies hat zur Folge, daß diese Verteilungsfamilie nur in Situationen adäquat ist, bei denen das Simpsonsche Paradoxon nicht auftreten kann. Dies schränkt den Anwendungsbereich dieser Verteilungsklasse erneut ein. Die Arbeit von (Caputo 1998) stellt eine ausführliche Abhandlung zu den Eigenschaften dieser Verteilungsklasse im Rahmen graphischer Modelle dar. Ein Ziel bei der Entwicklung des Programms ist es, auch die Analyse von Ereignisdaten zu ermöglichen. Da dieses Gebiet aber noch erforscht wird, kann in der vorliegenden Arbeit nur ein allgemeiner Rahmen bereitgestellt werden, der eine Integration dieser Modelle ermöglicht.

Ein weiterer Forschungsbereich betrifft die Berücksichtigung fehlender Daten in graphischen Modellen. In Erhebungen kommt es typischerweise vor, daß nicht alle Fragen beantwortet werden oder nicht für alle Messungen Werte vorliegen. Man steht also vor dem Problem sogenannter fehlender Werte oder *Missing values*. Im Rahmen graphischer Kettenmodelle wird dieses Problem bis jetzt meist so gehandhabt, daß die Versuchsreihen mit fehlenden Werten aus der Analyse gestrichen werden. Dies führt unter Umständen zu einem bedeutend kleineren Stichprobenumfang und somit zu einem Informationsverlust. Es sind aber auch durchaus andere Möglichkeiten denkbar, wie mit fehlenden Werten umzugehen ist. Ein Beispiel ist, Personen mit fehlenden Werten für einzelne Variablen nicht komplett aus der Stichprobe zu entfernen, sondern die Teilvektoren, für die Werte vorliegen, in die Analyse miteinzubeziehen. Dabei impliziert man, daß von einer zufälligen Nichtbeantwortung der Fragen ausgegangen wird, die aber sicherlich nicht immer gegeben ist. Diese Lösung ist auch im Programm realisiert. Liegt der Nichtbeantwortung von Fragen ein bestimmter

Mechanismus zugrunde, so führt dies zu einer unter Umständen ernstzunehmenden Verzerrung und somit möglicherweise zu irreführenden Interpretationen. Im Rahmen dieses Projektes soll deshalb nach alternativen Methoden gesucht werden, mit diesem Phänomen umzugehen. So wäre es zum Beispiel wünschenswert Meßzahlen zur Verfügung zu haben, mit deren Hilfe man entscheiden kann, ob die Annahme des zufälligen Fehlens bestimmter Werte in einer gegebenen Datensituation sinnvoll ist.

Eine andere Aufgabe ist es, einen Wissenschaftler bei der Erstellung eines graphischen Modells zu unterstützen. So ist die Analyse eines Datensatzes mit herkömmlichen Modellen in der Statistik schon gründlich untersucht worden, aber ein angemessenes Modellwahlverfahren zur Erstellung eines Graphen wurde bislang eher selten diskutiert. Dies liegt zum Teil daran, daß es hier kein wissenschaftlich begründetes eindeutiges Konzept, nur Verfahren gibt, die heuristisch begründet oder rein explorativ zu verstehen sind. Ein Schwerpunkt dieser Arbeit liegt daher gerade darin, eine gute Unterstützung für die Erstellung graphischer Modelle bereitzustellen, was in den folgenden Kapiteln genauer diskutiert wird.

Kapitel 3

Selektionsstrategie für die Erstellung eines graphischen Modells

Dieses Kapitel beschäftigt sich mit der Frage der Modellanpassung eines vorhandenen Datensatzes. Im Rahmen graphischer Modelle wird dabei der Typ der zugrundeliegenden multivariaten Verteilung als Modellannahme vorausgesetzt. Damit reduziert sich das Problem der Modellanpassung auf die Bestimmung eines Graphen, der die Assoziationsstruktur der Daten erzeugenden Variablen widerspiegelt, und auf die Schätzung der zugehörigen Parameter. Für die Selektion eines Graphen aus der Menge aller möglichen Graphen zu einer vorgegebenen Variablenmenge gibt es kein allgemeingültiges Verfahren. In der Praxis hat sich allerdings eine Selektionsstrategie bewährt, die auf Variablenselektion beruht. Diese Selektionsverfahren haben das Ziel, Modelle zu finden, die mit möglichst wenig Variablen auskommen, aber dennoch hinreichend genau die Strukturen der Daten abbilden. In diesem Kapitel wird diejenige Selektionsstrategie erläutert, die im Programm zur Erstellung eines graphischen Modells verwendet wird. Der Zweck des Kapitels liegt darin, einerseits eine Grundlage zu schaffen, die eine Implementierung der Selektionsstrategie ermöglicht. Andererseits ist ein Einblick in den Ablauf der Selektionsstrategie erforderlich, um die Benutzerführung des Programms und die Modellierung einzelner Programmteile zu verstehen, was in den restlichen Kapiteln der Arbeit erfolgt. Dieser Abschnitt kann in diesem Sinne als eine Ergänzung der Grundlagen graphischer Modelle, wie sie in Kapitel 2 beschrieben wurden, betrachtet werden.

Das hier verwendete Selektionsverfahren kombiniert gängige Selektionsstrategien aus der Statistik, deren prinzipielle Vorgehensweisen und die dafür nötigen Hilfsmittel in Abschnitt 3.1 dargestellt werden. Eine mathematisch exakte Erläuterung der statistischen Standardverfahren, die in den einzelnen Berechnungsschritten der Regression zur Anwendung kommen, wird außer Acht gelassen, da die Implementierung dieser Verfahren nicht Bestandteil der Arbeit ist. Es wird zudem nicht auf eine inhaltliche Begründung der vorgenommenen Schritte eingegangen. Diese findet man in Heinicke (1996, S. 58ff), sowie eine exakte Definition der Standardverfahren in Cox und Wermuth (1996, Kapitel 4). Im Anschluß wird die Selektionsstrategie anhand eines abstrakten Algorithmus erklärt, der sowohl für die Erstellung eines Konzentrations– als auch eines Kovarianzgraphen verwendbar ist. Darauf folgt ein Abschnitt, der beschreibt, wie aus den Ergebnissen der Selektionsstrategie ein Graph erstellt wird. Um den Ablauf des Algorithmus noch zu verdeutlichen, werden die einzelnen Schritte und die Überführung in einen Graphen in Abschnitt 3.4 anhand eines einfachen Beispiels zur Erstellung eines Konzentrationsgraphen durchgeführt.

3.1 Grundlegende Begriffe

Dieser Abschnitt stellt wichtige Begriffe bereit, die in dem in Abschnitt 3.2 beschriebenen Algorithmus verwendet werden. Dieser setzt sich, grob beschrieben, aus verschiedenen Regressionen zusammen. Im Zusammenhang mit Regressionsmodellen treten üblicherweise die Begriffe Haupteffekt, Interaktion und nichtlinearer Effekt auf, die im folgenden kurz erläutert werden. Als Haupteffekt wird eine Variable bezeichnet, die einen direkten Einfluß über das Regressionsmodell auf die betrachtete Zielgröße ausübt. Eine Interaktion ist eine Kombination mehrerer Einflußvariablen in der Art, daß mit der Aufnahme einer Interaktion in das Modell der Tatsache Rechnung getragen wird, daß der Einfluß einer Variable auf die Zielgröße je nach Ausprägung weiterer Variablen unterschiedlich sein kann. Dabei sind Interaktionen in diesem Programm auf paarweise Interaktionen beschränkt, da der Einfluß höherer Interaktionen auf die Zielvariable inhaltlich meist schwer interpretierbar ist. Ergänzend zu den Haupteffekten, die als linear modelliert werden, wird zusätzlich die Möglichkeit nichtlinearer Effekte berücksichtigt. Unter einer nichtlinearen Transformation wird hier stets das Quadrieren einer Variablen verstanden, da eine quadratische Transformation die meisten anderen genügend glatten nichtlinearen Transformationen hinreichend gut approximiert. Das Programm ist aber so ausgelegt, daß auch andere nichtlineare Transformationen betrachtet werden können (siehe Abschnitt 5.2).

3.1.1 Screening

(Cox und Wermuth 1994) schlagen ein Screening–Verfahren vor, mit dem Hinweise darüber erhalten werden können, ob neben den Haupteffekten Interaktionen oder auch nichtlineare Transformationen von Einflußgrößen in der Modellierung zu berücksichtigen sind, um die Zusammenhangsstruktur der Daten im Modell adäquat zu erfassen. Es werden zwei Screenings durchgeführt, eines für die hier interessierenden paarweisen Interaktionen und eines für nichtlineare Transformationen, also hier speziell für die quadrierten Variablen. Das Ergebnis jedes Screenings wird neben den mathematischen Berechnungen graphisch visualisiert. Dabei wird jede im Modell enthaltene Variable in Abhängigkeit zu den Variablen gesetzt, die auch bei der Erstellung des jeweiligen Graphen als Einflußgrößen in Frage kommen (siehe Abschnitt 2.2). Jeder Punkt im Plot entspricht einer solchen Untersuchung. Eine detaillierte Beschreibung des Verfahrens, wie etwa der Größen, die gegeneinander aufgetragen werden, wird hier nicht gegeben, da dies zu tief in die statistische Analyse führen würde. Um nun Aussagen treffen zu können, welche Variablen als Interaktionen oder als nichtlineare Transformationen in der jeweiligen Regression zu berücksichtigen sind, ist die Diagonale des jeweiligen Plots von Bedeutung. Weicht nämlich ein Punkt stark von der Diagonalen ab, sollte man den dahinterstehenden Interaktionen bzw. nichtlinearen Transformationen in der jeweiligen Regression Beachtung schenken. Wann die Abweichung von der Diagonalen als bedeutend eingeschätzt wird, ist nicht scharf definiert und beruht auf Augenmaß. Das Vorgehen bei diesem Screening ist vergleichbar mit der Erstellung von Q–Q–Plots (siehe Fahrmeir, Künstler, Pigeot und Tutz (1997, S. 93ff)). In Ergänzung zu einer rein visuellen Interpretation des Plots läßt sich noch eine quantitative Unterstützung angeben, die auf einem geeigneten kritischen Wert des zugrundeliegenden statistischen Tests beruht und in Analogie zu (Cox und Wermuth 1994) erfolgt. Hier wird deutlich, daß die Screenings lediglich eine explorative Hilfestellung bei der Auffindung von Interaktionen und nichtlinearen Transformationen liefern, wenn auch eine überaus wichtige.

3.1.2 Standard–Selektionsverfahren

Eine Vielzahl der üblichen Statistikprogramme stellt zwei prinzipiell verschiedene Modellwahlstrategien für Regressionsanalysen, nämlich die schrittweise Vorwärtsselektion und die schrittweise Rückwärtsselektion, bereit. Die beiden Verfahren beruhen auf unterschiedlichen Strategien.
Die schrittweise Vorwärtsselektion setzt mit einem geeigneten „Nullmodell“ an. Ein Nullmodell ist dabei ein minimales Modell, das zur Erklärung der Zielvariable geeignet ist. Es wird dann in jedem Schritt um diejenige Variable erweitert, die ein bestimmtes Einschlußkriterium am besten erfüllt. Das Einschlußkriterium stellt sicher, daß die aufgenommene Variable die Vorhersage der Zielgröße wesentlich verbessert, und somit die Erweiterung des Modells um diese Variable überhaupt sinnvoll ist. Die schrittweise Vorwärtsselektion wird solange angewandt, bis entweder alle betrachteten Variablen mit in das Modell aufgenommen wurden oder keine der übrigen Variablen das Einschlußkriterium erfüllt.
Das andere gängige Verfahren, die schrittweise Rückwärtsselektion, geht von einem Haupteffektmodell aus, d.h. von einer Regression, in der alle Modellvariablen enthalten sind. Es kommt dann zu einem schrittweisen Ausschluß je einer Variablen, und zwar derjenigen, die am weitesten unter einem gegebenen Ausschlußkriterium liegt. Dies wird solange fortgesetzt, bis entweder ein Nullmodell entsteht oder alle Variablen über dem Ausschlußkriterium liegen.

3.2 Algorithmus zur Selektionsstrategie

Der im Programm zu implementierende Algorithmus zur Selektionsstratgie kombiniert die in Abschnitt 3.1 beschriebenen Verfahren. Ein schematischer Ablauf ist in Abbildung 3.1 dargestellt.
Zunächst werden die Variablen dem oben beschriebenen Screening unterworfen und die dabei gefundenen Ergebnisse abgespeichert. Danach wird der Graph aus einer Folge von Regressionen aufgebaut, deren Ablauf im folgenden beschrieben wird. Allgemein ist dabei zu beachten, daß sobald eine Variable, die im Haupteffektmodell vorhanden ist, aus der Regression ausgeschlossen wird, auch alle Interaktionen, in denen diese Variable enthalten ist, aus dem Modell entfernt werden. Bei jeder Regression wird geprüft, ob beim Screening eine zu berücksichtigende Interaktion oder nichtlineare Transformation festgestellt wurde. Falls dies der Fall ist, wird eine Vorwärtsselektion

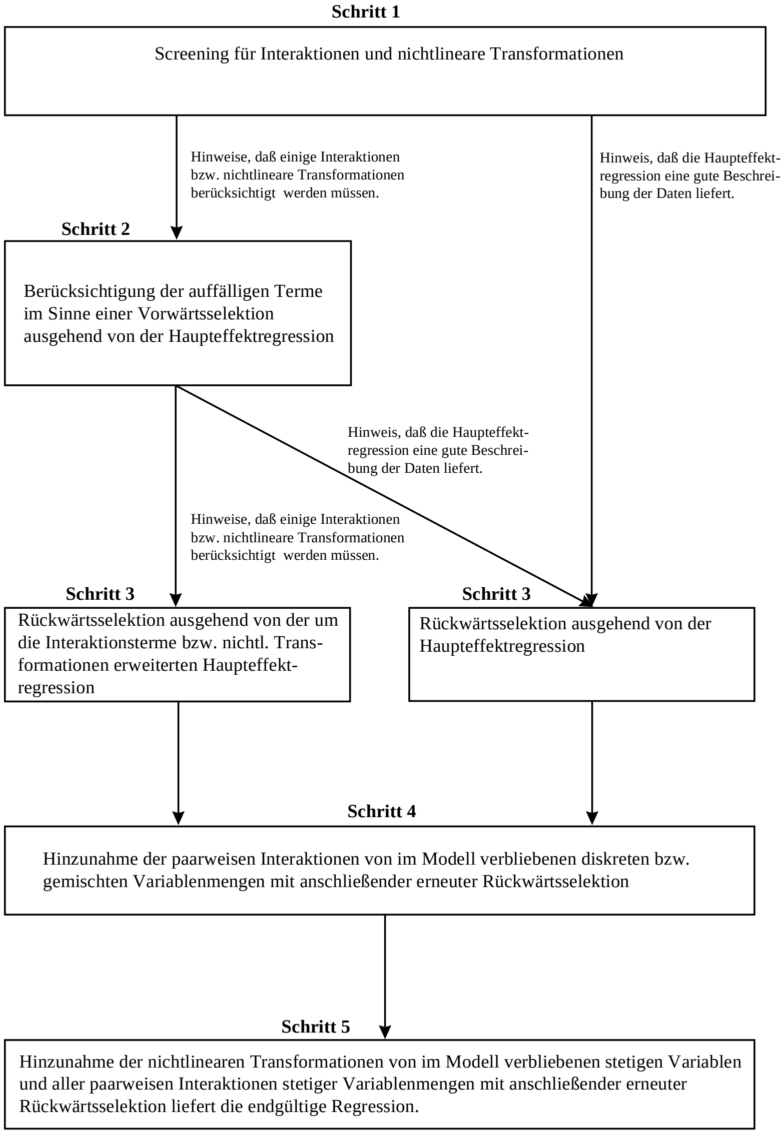


Abbildung 3.1: Schematischer Ablauf der Selektionsstrategie

durchgeführt, in der untersucht wird, welche Interaktionen bzw. nichtlinearen Transformationen mit in das Modell aufgenommen werden. Ausgangsbasis ist aber hier nicht das Nullmodell, sondern das Haupteffektmodell für die jeweilige Variable. Daraufhin wird eine Rückwärtsselektion durchgeführt. Als nächstes werden alle paarweisen Interaktionen von den übriggebliebenen diskreten oder gemischten Variablenmengen in das Modell aufgenommen und darauf basierend wieder eine Rückwärtsselektion ausgeführt. Zuletzt werden die verbliebenen stetigen Variablen automatisch quadriert bzw. von den stetigen Variablenmengen paarweise Interaktionen gebildet, in das Modell aufgenommen und wiederum eine Rückwärtsselektion durchgeführt. Ist dieses Schema für alle Variablen erfolgt, resultiert ein minimales Modell, das die notwendigen Haupteffekte, Interaktionen und nichtlinearen Anteile in Form von quadrierten Effekten berücksichtigt.

3.3 Erstellung des Graphen aus den Ergebnissen der Selektionsstrategie

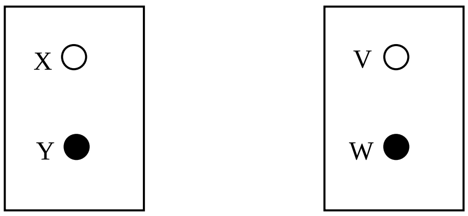
Die Ergebnisse der Selektionsstrategie für jede einzelne Variable werden wie folgt in einen Graphen umgewandelt, der dadurch sukzessive aufgebaut wird. Von jedem Haupteffekt im verbleibenden Modell, der nicht in derselben Kettenkomponente liegt wie die Zielvariable, wird ein Pfeil zur Zielvariablen eingetragen. Zudem wird die Zielvariable mit allen Haupteffekten des Modells durch eine Linie verbunden, die in derselben Komponente liegen wie sie. Im Modell vorhandene Interaktionen und nichtlineare Transformationen werden im Graphen nicht direkt dargestellt, da sie – obwohl äußerst wichtig für die Modellfindung im Rahmen der Selektionsstrategie – im Sinne von Markov–Eigenschaften eines graphischen Modells nicht zu interpretieren sind. Liegt eine Variable nur als nichtlineare Transformation vor, so wird von der Variable mit dem dazugehörigen Haupteffekt aus die Linie bzw. der Pfeil eingetragen.

3.4 Beispiel

Das folgende Beispiel illustriert den Vorgang der Variablenselektion und die anschließende Generierung des zugehörigen Kettengraphen. Im Rahmen dieses Beispiels soll vor allem die Kombination von Vorwärts– und Rückwärtsselektion transparent gemacht werden. Das Beispiel ist rein fiktiv, und entsprechend sind die Ergebnisse der Vergleiche mit den Ein– und Ausschlußkriterien (hier mit A bzw. E bezeichnet) willkürlich gewählt. Außerdem beinhaltet der oben beschriebene Selektionsalgorithmus weitere spezielle Regeln, auf die im Rahmen dieser Arbeit nicht eingegangen wird, die aber indirekt im Beispiel Eingang finden.

Im Programm und folgenden Beispiel wird das Screening zunächst für alle Variablen durchgeführt. Entsprechend beginnt die Beschreibung der Variablenselektion für jede Zielvariable mit dem zweiten Schritt des Algorithmus aus Abbildung 3.1. Der Ablauf wird für die erste Zielgröße im Detail dargestellt. Für die weiteren Zielgrößen werden nur die jeweilige Ausgangssituation und die Ergebnisse der Teilschritte angegeben, da die auszuführenden Schritte sich stets wiederholen.

Abhängigkeitskette:
Ausgegangen wird von zwei Kettenkomponenten mit je zwei Variablen, und zwar jeweils einer stetigen ($\circ$) und einer diskreten ($\bullet$), wie nachfolgende Abbildung illustriert.



I Screening

Zunächst wird das Screening für alle in der Abhängigkeitskette enthaltenen Variablen durchgeführt, und zwar eines für die nichtlinearen Transformationen und eines für die paarweisen Interaktionen. Die Ergebnisse der dafür nötigen Regressionen sind in den beiden untenstehenden Tabellen angegeben. In der Spalte *auffällig* steht für diejenigen Regressionen ein *ja*, die in Schritt 2 des Algorithmus mitberücksichtigt werden müssen.

A Screening auf nichtlineare Transformationen

Zielgröße	**Einflußgrößen**	**auffällig**
X	Y, Y^2 V, V^2 W, W^2	ja
Y	X, X^2 V, V^2 W, W^2	
V	W, W^2	
W	V, V^2	

B Screening auf Interaktionen

Zielgröße	**Einflußgrößen**	**auffällig**
	$Y, V, Y \times V$	
X	$Y, W, Y \times W$	
	$V, W, V \times W$	ja
	$X, V, X \times V$	
Y	$X, W, X \times W$	ja
	$V, W, V \times W$	ja

II Variablenselektion

Die Variablenselektion wird für die Zielvariable X ausführlich dargestellt, für die restlichen Variablen wird der Ablauf nur schematisch beschrieben, da er für jede Zielvariable analog abläuft. In der Beschreibung des Beispiels gibt es folgende Konvention. Wird das Ein– bzw. Ausschlußkriterium für eine Variable unter Einbeziehung anderer Variablen im Modell neu berechnet, so zeigt eine fortlaufende Zahl von Apostrophen an, daß dieser Wert sich von einem für diese Variable vorher berechneten unterscheiden kann.

A **Zielgröße** X

Schritt 2 des Algorithmus:

Schritt 2 des Algorithmus überprüft mittels einer Vorwärtsselektion, ob die im Screening „auffälligen“ Variablen in das Modell mit aufgenommen werden müssen.

<u>Ausgangssituation:</u>

- Grundlage ist die Haupteffektregression, d.h. hier die Einflußgrößen Y, V und W, da sie in derselben Kettenkomponente wie X bzw. in der rechts davon vorhanden sind.
- Mögliche weitere Einflußgrößen sind V^2 und $V \times W$, da sie im Screening für X auffällig sind.

<u>Vorwärtsselektion:</u>

- Bestimmung der Reihenfolge der Aufnahme:
  In den folgenden Schritten wird untersucht, welche auffällige Variable als erstes in Betracht kommt, in das Modell mitaufgenommen zu werden.

Dazu werden die einzelnen Einschlußkriterien der Variablen aus dem Screening berechnet und verglichen. Die Variable mit dem höchsten Wert wird zunächst in das Modell eingefügt.

(i) Berechne Einschlußkriterium E_{V^2} für Regression auf X mit den Einflußgrößen Y, V, W und V^2.

(ii) Berechne Einschlußkriterium $E_{V \times W}$ für Regression auf X mit den Einflußgrößen Y, V, W und $V \times W$.

(iii) Da hier $E_{V^2} > E_{V \times W}$ gilt, wird V^2 vorläufig mit in das Modell aufgenommen.

- Entscheidung über Aufnahme von V^2:
  Nun muß untersucht werden, ob das Einschlußkriterium für V^2 auch größer ist als das generelle Einschlußkriterium. Nur dann wird die Variable endgültig in das Modell eingefügt.

  (i) Berechne Einschlußkriterium E_{V^2} für Regression auf X mit den Einflußgrößen Y, V, W und V^2 (entspricht obigem Punkt (i)).

  (ii) Da hier $E_{V^2} > E$, wird V^2 aufgenommen.

- Entscheidung über Aufnahme von $V \times W$:
  Da $V \times W$ die letzte auffällige Variable ist, muß nicht mehr berechnet werden, welche Variable als nächstes hinzugenommen wird. Durch den Einschluß von V^2 kann sich aber das Einsschlußkriterium für $V \times W$ verändert haben und muß somit neu berechnet und mit dem allgemeinen Einschlußkriterium verglichen werden.

  (i) Berechne Einschlußkriterium $E'_{V \times W}$ für Regression auf X mit den Einflußgrößen Y, V, W, V^2 und $V \times W$.

  (ii) Da hier $E'_{V \times W} < E$, wird $V \times W$ nicht aufgenommen.

Schritt 2 ist hiermit beendet, da alle im Screening auffälligen Variablen betrachtet wurden.

Schritt 3 des Algorithmus:

In Schritt 3 des Algorithmus wird versucht, die nach Schritt 2 im Modell vorhandenen Einflußgrößen für X mittels einer Rückwärtsselektion auf die einzuschränken, die einen relativ hohen Einfluß auf die Zielvariable X haben.

Ausgangssituation:

- Einflußgrößen: Y, V, W und V^2, da diese nach Schritt 2 im Modell sind.
- Möglicher Ausschluß von Y, V, W und V^2.

Rückwärtsselektion:

In den folgenden Schritten wird untersucht, welche Variable als erstes in Betracht kommt, aus dem Modell entfernt zu werden. Dazu werden die einzelnen Ausschlußkriterien der Variablen berechnet und verglichen. Die Variable mit dem niedrigsten Wert wird zunächst aus dem Modell entfernt.

- Bestimmung der Reihenfolge des Ausschlusses:
  (i) Berechne Ausschlußkriterium A_Y für Regression auf X mit Einflußgrößen V, W, V^2.
  (ii) Berechne Ausschlußkriterium A_V für Regression auf X mit Einflußgrößen Y, W, V^2.
  (iii) Berechne Ausschlußkriterium A_W für Regression auf X mit Einflußgrößen Y, V, V^2.
  (iv) Berechne Ausschlußkriterium A_{V^2} für Regression auf X mit Einflußgrößen Y, V, W.
  (v) Da hier $A_W < A_{V^2} < A_V < A_Y$, wird W zuerst ausgeschlossen.
- Entscheidung über Ausschluß von W:
  Nun muß untersucht werden, ob das Ausschlußkriterium für W kleiner ist als das generelle Ausschlußkriterium. Nur dann wird die Variable endgültig aus dem Modell entfernt.
  (i) Berechne Ausschlußkriterium A_W für Regression auf X mit Einflußgrößen Y, V, V^2 (entspricht obigem Punkt (iii)).
  (ii) Da hier $A_W < A$, wird W aus dem Modell entfernt.

Neue Ausgangssituation:

Nach dem Ausschluß von W, sind nur noch Y, V und V^2 im Modell vorhanden. Es wird nun analog zu oben versucht, Variablen aus dem Modell zu entfernen.

- Einflußgrößen: Y, V und V^2
- möglicher Ausschluß von: Y, V, V^2

Rückwärtsselektion:

- Bestimmung der Reihenfolge des Ausschlusses:
  (i) Berechne Ausschlußkriterium A'_Y für Regression auf X mit Einflußgrößen V, V^2.
  (ii) Berechne Ausschlußkriterium A'_V für Regression auf X mit Einflußgrößen Y, V^2.
  (iii) Berechne Ausschlußkriterium A'_{V^2} für Regression auf X mit Einflußgrößen Y, V.
  (iv) Da hier $A'_Y < A'_{V^2} < A'_V$, wird Y zuerst ausgeschlossen.
- Entscheidung über Ausschluß von Y:
  (i) Berechne Ausschlußkriterium A'_Y für Regression auf X mit Einflußgrößen V, V^2 (entspricht obigem Punkt (i)).
  (ii) Da hier $A'_Y > A$, wird Y nicht aus dem Modell entfernt.

Da schon das kleinste der obigen Ausschlußkriterien, nämlich A'_Y, größer ist als das Ausschlußkriterium, ist Schritt 3 hiermit beendet.

Schritt 4 des Algorithmus:

In Schritt 4 werden zunächst alle paarweisen Interaktionen von im Modell verbliebenen diskreten bzw. gemischten Variablenmengen in das Modell hinzugenommen. Durch eine anschließende Rückwärtsselektion wird wie in Schritt 3 versucht, Variablen aus dem Modell zu entfernen.

Situation nach dem 3. Schritt:

- Einflußgrößen: Y, V und V^2
- davon stetig: V und V^2
- davon diskret: Y

Da Y die einzige diskrete Variable ist, werden die Interaktionen $Y \times V$ und $Y \times V^2$ in das Modell hinzugenommen.

Neue Ausgangssituation:

- Einflußgrößen: $Y, V, V^2, Y \times V$ und $Y \times V^2$
- möglicher Ausschluß von: $Y, V, V^2, Y \times V$ und $Y \times V^2$

Die Rückwärtsselektion läuft nun analog zu Schritt 3 ab und wird deshalb hier in Stichpunkten beschrieben. Nur Besonderheiten werden erwähnt.

Rückwärtsselektion:

- Bestimmung der Reihenfolge des Ausschlusses:
  (i) Berechne Ausschlußkriterium A''_Y für Regression auf X mit Einflußgrößen $V, V^2, Y \times V, Y \times V^2$.
  (ii) Berechne Ausschlußkriterium A''_V für Regression auf X mit Einflußgrößen $Y, V^2, Y \times V, Y \times V^2$.
  (iii) Berechne Ausschlußkriterium A''_{V^2} für Regression auf X mit Einflußgrößen $Y, V, Y \times V, Y \times V^2$.
  (iv) Berechne Ausschlußkriterium $A_{Y \times V}$ für Regression auf X mit Einflußgrößen $Y, V, V^2, Y \times V^2$.
  (v) Berechne Ausschlußkriterium $A_{Y \times V^2}$ für Regression auf X mit Einflußgrößen $Y, V, V^2, Y \times V$.
  (vi) Da hier $A''_{V^2} < A_{Y \times V^2} < A_{Y \times V} < A''_V < A''_Y$, wird V^2 zuerst ausgeschlossen.
- Entscheidung über Ausschluß von V^2:
  (i) Berechne Ausschlußkriterium A''_{V^2} für Regression auf X mit Einflußgrößen $Y, V, Y \times V, Y \times V^2$ (entspricht obigem Punkt (iii)).
  (ii) Da hier $A''_{V^2} < A$, wird Y^2 aus dem Modell entfernt. Außerdem wird die gemischte Interaktion $Y \times V^2$ entfernt, da Interaktionen nur betrachtet werden, wenn auch die zugehörigen Haupteffekte im Modell sind.

Neue Ausgangssituation:

- Einflußgrößen: $Y, V, Y \times V$
- möglicher Ausschluß von: $Y, V, Y \times V$

Rückwärtsselektion:

- Bestimmung der Reihenfolge des Ausschlusses:
  (i) Berechne Ausschlußkriterium A'''_Y für Regression auf X mit Einflußgrößen $V, Y \times V$.
  (ii) Berechne Ausschlußkriterium A'''_V für Regression auf X mit Einflußgrößen $Y, Y \times V$.
  (iii) Berechne Ausschlußkriterium $A'_{Y \times V}$ für Regression auf X mit Einflußgrößen Y, V.

(iv) Da hier $A'_{Y\times V} < A'''_Y < A'''_V$, wird $Y \times V$ zuerst ausgeschlossen.

- Entscheidung über Ausschluß von $Y \times V$:
  (i) Berechne Ausschlußkriterium $A'_{Y\times V}$ für Regression auf X mit Einflußgrößen Y, V (entspricht obigem Punkt (iii)).
  (ii) Da hier $A'_{Y\times V} > A$, wird $Y \times V$ nicht entfernt.

Schritt 4 ist hiermit beendet, da die Ausschlußkriterien der verbliebenen Variablen größer als das allgemeine Ausschlußkriterium sind.

Schritt 5 des Algorithmus:

In Schritt 5 werden zunächst alle nichtlinearen Transformationen von im Modell verbliebenen stetigen Variablen gebildet und in das Modell aufgenommen. Danach wird mittels Rückwärtsselektion versucht, die Einflußgrößen auf die wichtigen zu beschränken.

Situation nach dem 4. Schritt:

- Einflußgrößen: $Y, V, Y \times V$
- davon stetig: V
- davon diskret: Y

Da V^2 die einzige stetige Variable ist, wird V^2 in das Modell aufgenommen.

Neue Ausgangssituation:

- Einflußgrößen: $Y, V, Y \times V, V^2$
- möglicher Ausschluß von: $Y, V, Y \times V, V^2$

Rückwärtsselektion:

- Bestimmung der Reihenfolge des Ausschlusses:
  (i) Berechne Ausschlußkriterium A''''_Y für Regression auf X mit Einflußgrößen $V, Y \times V, V^2$.
  (ii) Berechne Ausschlußkriterium A''''_V für Regression auf X mit Einflußgrößen $Y, Y \times V, V^2$.
  (iii) Berechne Ausschlußkriterium $A''_{Y\times V}$ für Regression auf X mit Einflußgrößen V, Y, V^2.
  (iv) Berechne Ausschlußkriterium A'''_{V^2} für Regression auf X mit Einflußgrößen $V, Y, Y \times V$.

(v) Da hier $A''_{Y\times V} < A'''_{V^2} < A''''_{Y} < A''''_{V}$, wird $Y \times V$ zuerst ausgeschlossen.

- Entscheidung über Ausschluß von $Y \times V$:
  (i) Berechne Ausschlußkriterium $A''_{Y\times V}$ für Regression auf X mit Einflußgrößen V, Y, V^2 (entspricht obigem Punkt (iii)).
  (ii) Da hier $A''_{Y\times V} < A$, wird $Y \times V$ aus dem Modell entfernt.

Neue Ausgangssituation:

- Einflußgrößen: Y, V, V^2
- möglicher Ausschluß von: Y, V, V^2

Diese Rückwärtsselektion entspricht dem 2. Durchlauf im Schritt 3 des Algorithmus. Dort wurde keine der drei Variablen ausgeschlossen. Schritt 5 ist somit beendet und damit auch die Selektionsstrategie für X.

Endgültige Einflußgrößen für X: Y, V, V^2.

B **Zielgröße** Y

Schritt 2 des Algorithmus:

Ausgangssituation:

- Haupteffektregression, d.h. Einflußgrößen: X, V, W
- mögliche weitere Einflußgrößen: $X \times W, V \times W$

Ergebnis der Vorwärtsselektion:

Weder $X \times W$ noch $V \times W$ werden aufgenommen.

Schritt 3 des Algorithmus:

Ausgangssituation:

- Einflußgrößen: X, V, W
- möglicher Ausschluß von X, V, W

Ergebnis der Rückwärtsselektion:

V wird aus dem Modell entfernt.

Schritt 4 des Algorithmus:

Situation nach dem 3. Schritt:

- Einflußgrößen: X, W
- davon stetig: X
- davon diskret: W

Neue Ausgangssituation:

- Einflußgrößen: $X, W, X \times W$
- möglicher Ausschluß von $X, W, X \times W$

Ergebnis der Rückwärtsselektion:

$X, W, X \times W$ bleiben im Modell.

Schritt 5 des Algorithmus:

Situation nach dem 4. Schritt:

- Einflußgrößen: $X, W, X \times W$
- davon stetig: X
- davon diskret: W

Neue Ausgangssituation:

- Einflußgrößen: $X, W, X \times W, X^2$
- möglicher Ausschluß von $X, W, X \times W, X^2$

Ergebnis der Rückwärtsselektion:

X^2 wird aus dem Modell entfernt.

Endgültige Einflußgrößen für Y: $X, W, X \times W$.

C **Zielgröße** V

Schritt 2 des Algorithmus:

Ausgangssituation:

- Haupteffektregression, d.h. Einflußgröße: W
- mögliche weitere Einflußgrößen: keine

Vorwärtsselektion entfällt, da keine Variablen im Screening für V auffällig waren.

Schritt 3 des Algorithmus:

Ausgangssituation:

- Einflußgröße: W
- möglicher Ausschluß von W

Ergebnis der Rückwärtsselektion:

W bleibt im Modell.

Schritt 4 und 5 des Algorithmus entfallen, da man aus W allein keine Interaktion bilden kann und W eine diskrete und keine stetige Variable ist.

Endgültige Einflußgröße für V: W.

D **Zielgröße** W

Schritt 2 des Algorithmus:

Ausgangssituation:

- Haupteffektregression, d.h. Einflußgröße: V
- mögliche weitere Einflußgrößen: keine

Vorwärtsselektion entfällt, da keine Variablen im Screening für W auffällig waren.

Schritt 3 des Algorithmus:

Ausgangssituation:

- Einflußgröße: V
- möglicher Ausschluß von V

Ergebnis der Rückwärtsselektion:

V bleibt im Modell.

Schritt 4 des Algorithmus entfällt, da aus V allein keine Interaktion gebildet werden kann.

Schritt 5 des Algorithmus:

Ausgangssituation:

- Einflußgröße: V, V^2
- möglicher Ausschluß von V, V^2

Ergebnis der Rückwärtsselektion:

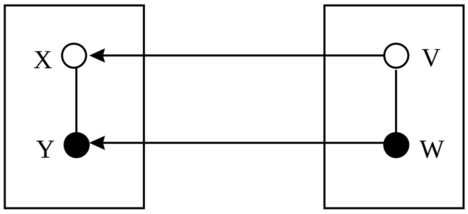
V^2 wird aus dem Modell entfernt.

Endgültige Einflußgröße für W: V.

Die folgende Tabelle gibt einen Überblick über die Einflußgrößen der einzelnen Variablen:

Zielgröße	**Einflußgrößen**
X	Y, V, V^2
Y	$X, W, X \times W$
V	W
W	V

Aus der oben beschriebenen Modellanpassungsstrategie und dem sich daraus ergebenden Resultat (vgl. auch Tabelle) läßt sich der endgültige Graph wie folgt ablesen.



Kapitel 4

Unified Modeling Language (UML)

Es ist eine allgemein bekannte Tatsache, daß Softwareentwicklung immer komplexer wird. Standen am Anfang der Computerära meist numerische Berechnungen im Vordergrund, die eine ziemlich einfache Ein/Ausgabe – Struktur besaßen, müssen heutzutage bei der Programmentwicklung verschiedenste Aspekte beachtet werden. Dazu gehören zum Beispiel eine gut durchdachte Benutzeroberfläche oder eine komplexe Dateiverwaltung. Um solchen gesteigerten Anforderungen gerecht zu werden, ist eine detaillierte Analyse und ein sorgfältiges Design des Programmes erforderlich, bevor mit der eigentlichen Implementierung begonnen wird. In der Analysephase der Programmentwicklung wird versucht, die Anwendungswelt durch die Informatik zu rekonstruieren. Dazu wird zunächst festgestellt, was das Programm leisten soll, aber noch nicht, wie dies geschehen soll. Innerhalb der Designphase dagegen wird die Anwendungswelt in einem abstrakten Modell abgebildet. Hierbei wird also der interne Aufbau des Programms modelliert. Bei der Softwareentwicklung gibt es nun verschiedene Ansätze zur Abwicklung der Design– und Analysephase. Diese Ansätze werden (Entwicklungs–)Methoden genannt. Die Methoden bieten ein Grundschema für einen erfolgreichen Programmentwurf, können aber kein „Kochrezept" sein, da die Entwicklung eines Programms vor allem auch ein kreativer Prozeß ist.
Das objektorientierte Paradigma hat sich als ein vernünftiges Konzept zur Realisierung von Softwaresystemem herauskristallisiert. Die Idee dazu ist über 25 Jahre alt. In diesem Zeitraum entwickelten sich auch die ersten objektorientierten Programmiersprachen. Das Prinzip der Objektorientierung kann man dabei als einen weiteren Schritt in der Datenabstraktion betrachten. Allgemein ist Datenabstraktion ein Prinzip, das den Programmcode für den Menschen verständlicher machen soll. So fing man zu Beginn der Computerära damit an, daß man, anstatt alles der Reihe nach aufzuschreiben, Pro-

zeduren einführte, die immer wieder aufgerufen werden können. Dies liefert einen reduzierten Code und somit einen besseren Überblick über das Geschriebene. Als nächstes wurden zusammenhängende Daten in sogenannten *records* gesammelt und somit die Strukturierung von Programmen verbessert, da zusammenhängende Daten jetzt in einer Einheit und nicht lose über das Programm verteilt waren. Die Objektorientierung setzt nun schließlich die Idee um, zu den Daten auch Prozeduren, die miteinander in Verbindung stehen, in einer Einheit, genannt *Klasse*, zu sammeln. Allerdings entstanden objektorientierte Entwicklungsmethoden erst Anfang der 90er Jahre, welche sich dann aber rasch in der Softwarebranche etablierten. Zu jeder Methode gehört auch eine eigene Notation, die die jeweiligen Elemente des aufzustellenden Modells darstellt. Als deutlich beliebteste Methoden stellten sich *OOD* (Object Oriented Design) von (Booch 1994) und *OMT* (Object Modeling Technique) von Rumbaugh (Rumbaugh, Blaha, Premerlani, Eddy und Lorensen 1991) heraus. Daneben existieren aber auch noch zahlreiche andere Ansätze, wie zum Beispiel *OOSE* (Object Oriented Software Engineering) von Jacobsen (Jacobsen, Christerson, Jonsson und Overgaard 1992). Eine Schwierigkeit bei der Verwendung von objektorientierten Entwicklungsmethoden liegt darin, daß es nicht nur zahlreiche Methoden mit einer eigenen Notation, sondern auch keine standardisierten Fachbegriffe für die Konzepte der Objektorientierung gibt.

Eine Besserung dieser Situation erhofft man sich von der Einführung der *UML* (Unified Modeling Language). Sie ist eine objektorientierte Modellierungssprache für die Spezifikation, Visualisierung, Erzeugung und Dokumentation von Software–Systemen und wurde von Booch, Rumbaugh und Jacobsen entwickelt. Die UML führt die früheren Ansätze der Autoren zusammen und wird als Nachfolgerin der anderen drei Sprachen angesehen. Zum jetzigen Zeitpunkt befindet sich die UML in der Version 1.1 und wird von der Object Management Group (OMG) als Standard anerkannt. Außerdem wird sie von verschiedenen bekannten Firmen, wie z.B. Microsoft, Hewlett–Packard, Oracle und IBM unterstützt, was vermuten läßt, daß sich die Sprache zu einem Quasi–Standard in der Industrie entwickelt.

Im Gegensatz zu den anderen objektorientierten Ansätzen ist die UML keine Entwicklungsmethode, sondern die Beschreibung einer einheitlichen Notation und eines Metamodells, was in Abschnitt 4.1 näher vorgestellt wird. Ob eine zugehörige Methode von den Autoren der UML in Zukunft folgen wird, ist derzeit noch unklar. In dieser Arbeit wird die UML verwendet, um die Programmkonzeption näher zu erläutern. Als Visualisierungwerkzeug wird dabei die Demo–Version von Rational Rose 4.0 eingesetzt.

Das folgende Kapitel gibt einen Überblick über die Zielsetzungen, die der Entwicklung dieser Sprache zugrunde liegen und bietet eine kurze Erläuterung der wichtigsten Notationselemente, soweit sie für die vorliegende Arbeit relevant sind. Die Darstellung folgt dabei (Rational Software Corporation 1997a), (Rational Software Corporation 1997b) und (Oestereich 1997), wobei die ersten beiden Titel im WWW unter der Adresse *http://www.rational.com/uml* zu finden sind. Das oben angesprochene Metamodell wird dagegen nicht erläutert, da eine Kenntnis dessen zum Verständnis dieser Arbeit nicht erforderlich ist. Eine genaue Darstellung des Metamodells findet sich in (Rational Software Corporation 1997c). Die in diesem Kapitel verwendeten Begriffe aus dem Bereich des objektorientierten Software–Engineerings werden in Anhang B der Arbeit eingeführt. In Anhang C findet sich zusätzlich ein Überblick über die Notationselemente der UML.

4.1 Zielsetzung

Bei der Entwicklung der UML werden folgende Zielsetzungen realisiert:

Bereitstellung einer einfachen, aussagekräftigen und visuellen Modellierungssprache zur Entwicklung von (Software–)Modellen.
Die Erfahrung zeigt, daß objektorientierte Modellierungssprachen für die Analyse und das Design „normaler" Anwendungen nur dann von einem breiten Publikum akzeptiert werden und sich zu einem Standard entwickeln können, wenn sie auf Konzepten aufgebaut sind, die schon weit verbreitet und leicht zu erlernen sind. Zudem muß die Sprache aussagekräftig sein, denn nur dann können Modelle entstehen, die innerhalb eines Entwicklungsteams ausgetauscht werden können, ohne daß im großen Maß zusätzliche Informationen weitergegeben werden müssen.

Ausbaufähigkeit der Sprache.
In bestimmten Bereichen der Software–Entwicklung werden spezielle Elemente benötigt, die von der UML nicht bereitgestellt werden. Aus diesem Grund ist die UML so definiert, daß sie leicht erweitert werden kann, ohne die grundlegenden Konzepte neu definieren zu müssen.

Unabhängigkeit von speziellen Programmiersprachen und Entwicklungsprozessen.
Bei der Entwicklung der UML wurde im Gegensatz zu anderen Modellierungssprachen, wie zum Beispiel der OOD, bewußt auf eine dazugehörige Methode verzichtet. Als Grund dafür wird von den Entwicklern von UML angegeben, daß eine Methode

die spezifischen Rahmenbedingungen des Anwendungsbereiches, des organisatorischen Umfelds und noch vieles mehr berücksichtigen muß. Um allen diesen Anforderungen gerecht werden zu können, sind verschiedene Methoden für verschiedene Situationen angebracht. Die UML kann deswegen als Basis für zahlreiche objektorientierte Entwicklungsmethoden fungieren. Sie ist zusätzlich von speziellen Programmiersprachen unabhängig, da sie eine große Basis an Modellierungselementen bereitstellt. Dies hat zur Folge, daß nicht jede objektorientierte Sprache alle Modellierungselemente von UML unterstützt. Bei der Programmerstellung muß deshalb darauf geachtet werden, daß nur Konzepte verwendet werden, die in der jeweiligen Sprache auch enthalten sind.

Bereitstellung einer formellen Basis zum Verständnis der Modellierungssprache.
Ein gewisser Grad an formeller Beschreibung einer Sprache ist für deren Verständnis notwendig. Das statische Modell der UML wird durch ein Metamodell beschrieben, das selbst in UML Klassendiagrammen erläutert ist. Ein derartiges Vorgehen ist für die Spezifizierung von Modellformaten üblich.

Unterstützung von Entwicklungskonzepten auf einer höheren Abstraktionsebene, wie Kollaborationsdiagrammen (collaboration diagrams), Rahmen (frames), Mustern (patterns) und Komponenten (components).
Um optimale Resultate aus der Objektorientierung und der Wiederverwendbarkeit zu ziehen, ist eine klar definierte Semantik der obigen Konzepte erforderlich.

Integration bewährter Konzepte.
Wie schon erwähnt, integriert die UML die bekanntesten objektorientierten Modellierungssprachen und bildet eine Sprache, die auf bewährten Methoden der Objektorientierung aufbaut. Dies erleichtert und beschleunigt die Einarbeitung für Entwickler, die schon einmal im objektorientierten Paradigma gearbeitet haben, und ist sicherlich ein Grund für den schnellen Erfolg der UML.

4.2 Wichtige Notationselemente

Wie bei objektorientierten Modellierungssprachen üblich, werden die Elemente eines Modells durch eine graphische Notation in verschiedenen Diagrammen dargestellt. Zusätzlich können wichtige Aussagen eines Diagramms durch eine angefügte verbale Beschreibung näher erläutert werden. Ein Diagramm bietet eine Sicht auf einen Modellausschnitt. Das Diagramm sollte auf die wichtigen Elemente beschränkt werden, um die Übersichtlichkeit zu gewährleisten. Im folgenden werden die grundlegenden Modellelemente gegliedert nach Diagrammtypen erläutert. Dabei ist zu berücksichtigen, daß einige Elemente in verschiedenen Diagrammen auftreten. In dieser Arbeit wird ein Element in dem Diagramm eingeführt, in dem es primär auftritt. Zu jedem Diagramm wird mindestens ein Beispiel, das dem zu realisierenden Programm entnommen ist, aufgezeigt. Auf Besonderheiten, die bei der Modellierung mittels der Implementierungssprache Delphi zu berücksichtigen sind, wird an der jeweiligen Stelle kurz eingegangen. Die deutschen Übersetzungen der Begriffe richten sich nach (Oestereich 1997). Eine Art von Diagrammen, nämlich die Implementierungsdiagramme (*implementation diagrams*), werden nicht beschrieben, da sie für diese Arbeit keine praktische Bedeutung haben. Genaue Angaben dazu findet man unter Rational Software Corporation (1997b, S. 132ff).

4.2.1 Anwendungsfalldiagramm

Ein Anwendungsfalldiagramm (*use case diagram*) beschreibt eine Folge von typischen Interaktionen zwischen dem Anwender und dem System. Es beschreibt hierbei nicht das interne Verhalten des Systems, sondern dient lediglich als Hilfsmittel zur Anforderungsanalyse: Es wird gezeigt, was geschehen soll, aber nicht, wie es im Detail aussieht. Das Diagramm besteht aus Akteuren (*actors*) und einer Menge von Anwendungsfällen (*use cases*), die von einer Linie abgeschlossen sind, die die Grenze des Systems kennzeichnet. Die Systemgrenze dient dazu, die Trennung zwischen den im Programm zu realisierenden Anwendungsfällen und den daran beteiligten Akteuren hervorzuheben. Zwischen den einzelnen Elementen des Diagramms gibt es verschiedene Beziehungen. Im folgenden werden die einzelnen Bestandteile eines Anwendungsfalldiagramms genauer beschrieben.

Anwendungsfall Ein Anwendungsfall ist die Beschreibung einer typischen Interaktion zwischen dem Anwender und dem System. Er wird durch eine Ellipse graphisch dargestellt, die den Namen des Anwendungsfalls zum Inhalt hat. Zudem existiert zu jedem Anwendungsfall eine Beschreibung, die die durchzuführenden Aktionen ausführlich erläutert. Hierbei ist es auch günstig, Beispieldialoge aus dem zu entwerfenden Programm, sogenannte explorative Prototypen, mitanzugeben.

Akteur Ein Akteur ist ein Objekt, das an einem Anwendungsfall beteiligt ist. Dies kann zum Beispiel ein Benutzer des Systems oder auch eine Systemkomponente sein.

Der Akteur wird als Rechteck oder, wenn es sich bei dem Objekt um eine Person handelt, typischerweise auch als Strichmännchen dargestellt.

Beziehungen In einem Anwendungsfalldiagramm gibt es mehrere Beziehungen zwischen den Diagrammelementen. Zum einen gibt es zwischen einem Akteur und einem Anwendungsfall die sogenannte Kommunikationsbeziehung (*communication relationship*). Diese bedeutet, daß der Akteur an einem Anwendungsfall beteiligt ist. Eine solche Beziehung wird durch eine Linie angezeigt.
Zum anderen können auch die Anwendungsfälle selbst miteinander in Verbindung stehen. Ein Pfeil von einem Anwendungsfall zu einem anderen zeigt dabei, daß das Verhalten des Anwendungsfalls, auf den der Pfeil zeigt, ein Teil des Anwendungsfalles ist, von dem der Pfeil ausgeht.

Beispiel 4.1 *Als konkretes Beispiel aus dem Programm dienen hier die Schritte, die der Anwender beim Einlesen eines neuen Datensatzes auszuführen hat. Einen Überblick gibt Anwendungsfalldiagramm 4.1:*
Konkret wird hier für eine genauere Beschreibung der Anwendungsfälle der Fall „Skalenniveau festlegen“ behandelt. Die Abkürzung „Af5“ steht hierbei für Anwendungsfall 5.

Af5 **Skalenniveau festlegen**

Szenario:

I Für jede Beobachtungsvariable das Skalenniveau durch Anklicken festlegen.

II Ein Häkchen vor dem Variablennamen zeigt an, daß das Skalenniveau für diese Variable bestimmt wurde.

III Sind alle Variablen belegt, kann die Schaltfläche „Fertig“ aufgerufen werden.

IV Mit „Fertig“ Dialog beenden.

4.2.2 Klassendiagramm

Klassendiagramme (*class diagrams*) geben die statische Struktur eines Modells wieder. Es werden Klassen, ihr interner Aufbau und Beziehungen zu anderen Klassen dargestellt. Zeitliche Abläufe können hingegen nicht mit Hilfe von Klassendiagrammen

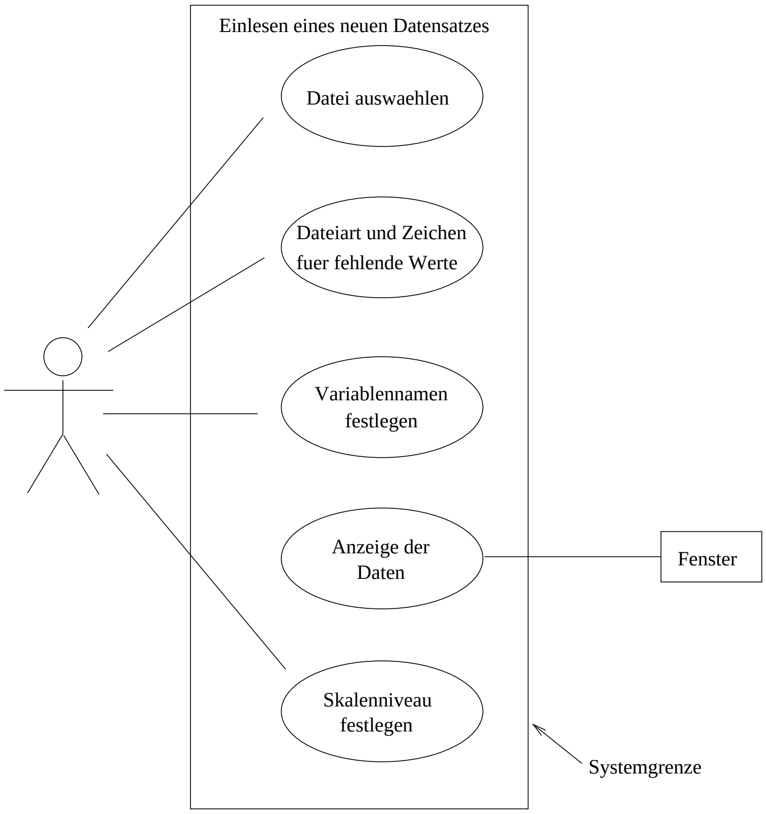


Abbildung 4.1: Anwendungsfalldiagramm für das Einlesen eines neuen Datensatzes

veranschaulicht werden. Hierzu gibt es in der UML andere Konzepte, die in Abschnitt 4.2.3 näher erläutert werden.

4.2.2.1 Basiselemente von Klassendiagrammen

Klasse Eine Klasse (*class*) ist der wesentliche Bestandteil eines Klassendiagramms. Sie beschreibt eine Menge von Objekten, die ähnliche Struktur, Verhalten und Beziehungen besitzen. Die Definition einer Klasse erfolgt durch die Angabe ihrer Attribute und Operationen. Ein **Attribut** (*attribute*) ist eine Dateneigenschaft, die für jedes Objekt einer Klasse vorliegt und für jedes Objekt einen individuellen Wert annimmt. Eine **Operation** (*operation*) ist eine Dienstleistung, die ein Objekt ausführen kann. Sie ist mit einer Funktion oder Prozedur im traditionellen imperativen Paradigma vergleichbar. Eine Operation wird durch ihre Signatur beschrieben, welche sich aus dem Operationsnamen, den Übergabeparametern und dem Rückgabetyp zusammensetzt.
Die graphische Darstellung einer Klasse erfolgt durch ein Rechteck, in dem zumindest

Abbildung 4.2: Dialogfeld für die Festlegung des Skalenniveaus der Modellvariablen

der Name der Klasse angegeben ist. Dieser beginnt mit einem Großbuchstaben und ist ein Substantiv im Singular. Dem Klassennamen kann der Name des zugehörigen *Pakets* vorangestellt werden, wobei beide Angaben durch zwei Doppelpunkte voneinander getrennt werden. Die Definition eines Pakets erfolgt weiter unten. Zusätzlich können Attribute und Operationen der Klasse aufgeführt werden. Dazu wird das Rechteck durch horizontale Linien in bis zu drei Rubriken unterteilt. Der obere Bereich enthält dann den Namen, der mittlere die Attribute und der untere die Operationen der Klasse. Abbildung 4.3 zeigt die verschiedenen Möglichkeiten der Notation einer Klasse.

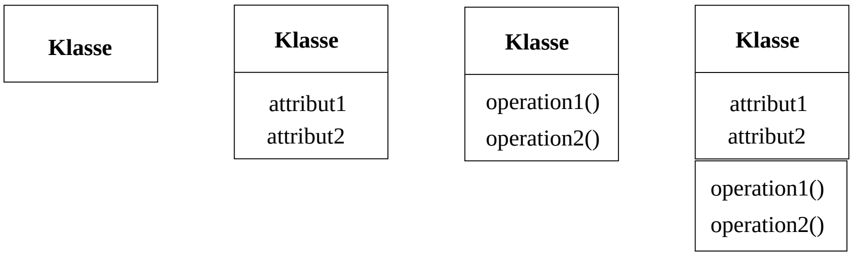


Abbildung 4.3: Verschiedene Darstellungsmöglichkeiten von Klassen

Sowohl vor den Attributen als auch vor den Operationen kann die zugehörige **Sichtbarkeit** (*visibility*) notiert werden. Sie gibt an, welche Klassen auf das Element zugreifen können. Hierbei gibt es drei Arten von Sichtbarkeitsstufen. Auf ein Element der Sichtbarkeitsstufe **public** kann jede Klasse zugreifen. Dies wird mit `"+"` gekennzeichnet. **Protected** bedeutet, daß die Klasse selbst, ihre *Unterklassen* (s. Abschnitt 4.2.2.2) und die als *„friend“* deklarierten Klassen dieses Element ansprechen können. Die Notation hierfür lautet `"#"`. Die größte Einschränkung beim Zugriff auf Klassenelemente ist die Sichtbarkeitsstufe **private**. Besitzt ein Element diese Stufe, können nur die Klasse selbst oder die als „friend“ deklarierten Klassen darauf zugreifen. Das Zeichen dafür ist `"-"`. Ein „friend“ einer Klasse ist dabei eine andere Klasse, die auf alle Bestandteile der einen Klassen vollen Zugriff hat, ansonsten aber unabhängig von ihr ist. In Delphi wird das friend–Konzept dadurch ermöglicht, indem friends einfach in dieselbe Quelldatei geschrieben werden. Tabelle 4.1 faßt die verschiedenen Sichtbarkeitsstufen zusammen.

Sichtbarkeit	**Zugriff**	**Zeichen**
public	alle Klassen	+
protected	Klasse selbst, friends und Unterklassen	#
private	Klasse selbst und friends	–

Tabelle 4.1: Arten von Sichtbarkeit

Beispiel 4.2 *Als Beispiel für den Aufbau einer Klasse dient die Klasse TOneValue, welche eine Realisation einer Variable abspeichert. Insgesamt werden in diesem Abschnitt die Beispiele aus dem Teil des Programms entnommen, welcher sich mit dem Einlesen und Abspeichern von Daten beschäftigt. Abbildung 4.4 zeigt den Aufbau der Klasse in UML–Notation. TOneValue ist einfach aufgebaut: Eine Realisation einer*

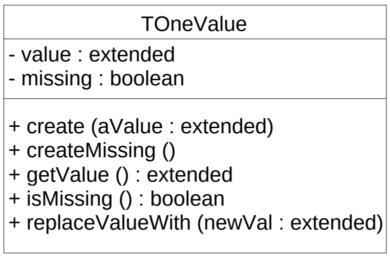


Abbildung 4.4: Aufbau der Klasse TOneValue

Variablen ist entweder vom Typ Extended, was einer Fließkommazahl mit sehr hoher

Genauigkeit entspricht, oder sie ist nicht vorhanden. In der Klasse wird dies durch die Attribute value *und* missing *modelliert. Die beiden Attribute haben die Sichtbarkeitsstufe private, um zu verhindern, daß ihre Werte von einer anderen Klasse geändert werden können. Das Lesen dieser Variablen ist den anderen Klassen aber durch die public–Operationen* getValue *bzw.* isMissing *möglich. Attribute einer Klasse als private zu deklarieren und durch geeignete Operationen, welche die Sichtbarkeitsstufe public haben, anderen Klassen ein Lese– bzw. Schreibrecht zu gewähren, ist ein in der objektorientierten Programmierung durchaus geläufiges Konzept. Die Operationen* create *und* createMissing *erzeugen jeweils ein neues Objekt der TOneValue–Klasse, je nachdem ob ein richtiger Wert oder das Zeichen für fehlende Daten eingelesen wird. Mit Hilfe der Operation* replaceValueWith *kann schon vorhandener Wert durch einen anderen ersetzt werden.*

Arten von Klassen Innerhalb des objektorientierten Paradigmas unterscheidet man verschiedene Spezialfälle von Klassen, welche im folgenden kurz aufgeführt werden. In dieser Arbeit werden nur diejenigen Arten erläutert, welche auch mit Delphi modelliert werden können.

Eine besondere Art von Klassendiagrammen sind **Objektdiagramme** (*object diagrams*). Sie zeigen das Verhalten von Objekten. Ein **Objekt** (*instance*) ist ein bestimmtes Exemplar einer Klasse, d.h. ein Objekt enthält eine Ausprägung aller Attribute einer Klasse und kann die in der Klasse definierten Operationen ausführen. Das Objektdiagramm zeigt den Zustand des Modells zu einem gewissen Zeitpunkt. Es wird häufig dazu benutzt, um Beispiele für Datenstrukturen zu geben. Die Notation eines Objekts ist prinzipiell dieselbe wie die einer Klasse, wobei die Rubrik für Operationen entfällt, da diese keine objekt–individuellen Ausprägungen besitzen, sondern für alle Objekte einer Klasse identisch sind. Im Unterschied zur Klasse beginnt der Name eines Objekts mit einem Kleinbuchstaben und ist unterstrichen. Zusätzlich wird häufig die Klasse des Objekts mit angegeben, welche nach dem Objektnamen erscheint und durch einen Doppelpunkt von diesem getrennt ist. In einer zweiten Rubrik können Attribute mit ihrem aktuellen Wert aufgelistet werden. Abbildung 4.5 zeigt verschiedene Möglichkeiten zur Darstellung eines Objekts.

Metaklassen (*metaclasses*) sind spezielle Klassen, deren Objekte wiederum Klassen sind. Im zugehörigen Diagramm steht dafür über dem Klassennamen `<<metaclass>>`.

Zur Strukturierung einer Klassenhierarchie werden **abstrakte Klassen** (*abstract classes*) formuliert. Diese erzeugen selbst keine Objekte, sondern stellen lediglich Attribute und Operationen bereit, die von ihren *Unterklassen* (s. Abschnitt 4.2.2.2) benötigt wer-

Abbildung 4.5: Verschiedene Darstellungsmöglichkeiten von Objekten

den. In Delphi wird dieses Konzept leider nur unzureichend unterstützt, da es keinen Mechanismus gibt, der verhindert, Objekte einer Klasse zu erzeugen. Im Diagramm wird eine abstrakte Klasse dadurch kenntlich gemacht, indem `{abstract}` unter den Klassennamen geschrieben wird.

Beispiel 4.3 *Eine abstrakte Klasse im zu erstellenden Programm ist die Klasse TReadData. Sie ist die* Oberklasse *zu den Klassen, die sich mit dem Einlesen der Daten aus einer Textdatei beschäftigen. Abbildung 4.6 zeigt den Aufbau der Klasse. Im folgenden werden nur die Elemente beschrieben, die nötig sind, um das Prinzip einer abstrakten Klasse zu begreifen. Etwas ausführlicher und zusammen mit ihren Unterklassen wird TReadData nochmals in Beispiel 4.5 erläutert. Ein Objekt der Klas-*

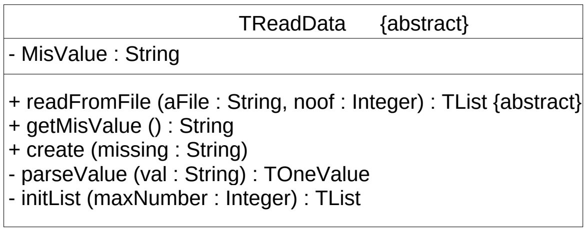


Abbildung 4.6: Die abstrakte Klasse TReadData

se TReadData selbst zu erstellen ist nicht sinnvoll, da es zwei Möglichkeiten gibt, wie eine Textdatei aussehen kann, die die Rohvariablen enthält (s. Kapitel 5.1.2). Dabei ändert sich das Einlesen der Daten jedoch grundlegend. Deshalb ist TReadData als abstrakt deklariert. Die Operation readFromFile*, die genau dieses Einlesen der Daten übernimmt, ist ebenfalls abstrakt, da die Unterklassen von TReadData die Operation zwar enthalten müssen, die Implementierung selbst hingegen in den einzelnen Klassen jedoch vollkommen anders ist. Die Operation* parseValue *liest eine Realisierung* *einer Variable in die in Beispiel 4.2 beschriebene Klasse ein. Da dies in den Unterklassen auf dieselbe Art und Weise geschieht, ist diese Operation schon in TReadData implementiert, also nicht abstrakt. Diese Kombination von abstrakten und schon implementierten Operationen ist typisch für abstrakte Klassen.*

Zudem kann in Delphi das Konzept der **Schnittstellenklassen** (*interfaces*) realisiert werden. Da dies aber nur auf sehr umständliche Art und Weise möglich ist, werden sie im Programm zwar nicht verwendet, der Vollständigkeit halber aber im folgenden beschrieben. Schnittstellenklassen sind abstrakte Klassen, die das externe Verhalten von Klassen spezifizieren. Sie enthalten die Beschreibung von Attributen und Operationen in abstrakter Form. Dies bedeutet, daß nur die Signatur einer Operation zur Verfügung gestellt wird, die Implementierung dazu aber nicht vorhanden ist. Klassen, welche die von einer Schnittstelle zur Verfügung gestellten Attribute und Operationen implementieren, stellen Umsetzungen dieser Schnittstelle dar. Da eine Klasse dabei mehrere Schnittstellen implementieren kann, sind Schnittstellen ein gutes Konzept in den objektorientierten Sprachen, die wie Delphi nur Einfachvererbung erlauben. Das Prinzip der Vererbung wird in Abschnitt 4.2.2.2 genauer erklärt. Eine Schnittstelle wird durch `<<interface>>` über dem Klassennamen gekennzeichnet.

Paket Ein Paket (*package*) ist eine Ansammlung von Bestandteilen des aufzustellenden Modells. Es dient zur Aufteilung des Gesamtmodells in überschaubare Einheiten. Pakete werden aufgrund logischer oder physikalischer Zusammenhänge von Elementen gebildet. Ein Paket kann selbst wieder Bestandteil eines anderen Pakets sein. Die Notation ist ein Aktenregister, in dem der Paketname steht. Zudem besteht die Möglichkeit, innerhalb eines Pakets dessen Elemente, also z.B. Klassen mitanzugeben. Ist dies der Fall, so wird der Paketname auf das Register geschrieben. Abbildung 4.7 zeigt exemplarisch die Darstellung eines Pakets.

Beispiel 4.4 *Das Gesamtmodell des Programms ist in die grundlegenden Pakete eingeteilt, die in Abbildung 4.8 zu sehen sind.*
Eine detaillierte Beschreibung der einzelnen Pakete findet sich in Abschnitt 5.2.1

4.2.2.2 Beziehungen

Generalisierung Unter einer Generalisierung (*generalization*) versteht man ein Abstraktionsprinzip zur hierarchischen Strukturierung von Modellen. Dabei werden die

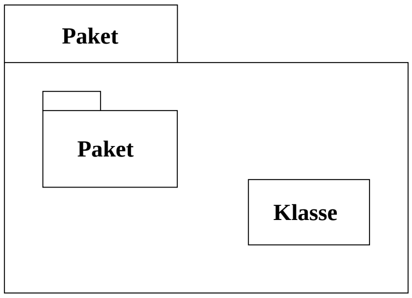


Abbildung 4.7: Beispiel zur Notation eines Pakets

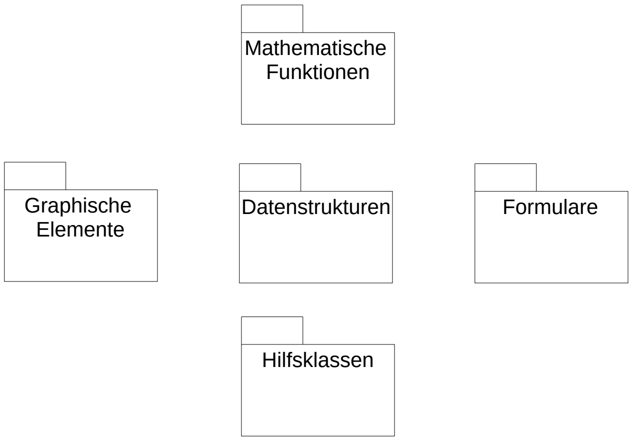


Abbildung 4.8: Elementare Pakete des Programms

Eigenschaften von Klassen gegliedert. Eigenschaften allgemeinerer Bedeutung werden allgemeineren Klassen, sogenannten **Oberklassen** (*superclasses*), zugeordnet. Spezielle Eigenschaften dagegen werden Klassen zugeordnet, die den allgemeineren Klassen untergeordnet sind. Diese Klassen nennt man **Unterklassen** (*subclasses*). Dabei kann eine Klasse Unterklasse einer Klasse und selbst wiederum Oberklasse von anderen Klassen sein. Derartige Beziehungen führen zu einer hierarchischen Struktur. Eigenschaften der Oberklasse werden an ihre Unterklassen weitergegeben, so daß die Unterklasse zu den von ihr selbst definierten Eigenschaften auch die Eigenschaften all ihrer Oberklassen enthält. Die Unterscheidung zwischen Ober– und Unterklassen erfolgt meist durch den sogenannten **Diskriminator** (*discriminator*). Dieser stellt das für die hierarchische Strukturierung maßgebliche Unterscheidungsmerkmal dar. Innerhalb der Klassenhierarchie werden Attribute und Operationen in die Klasse mitaufgenommen, in welcher

die ihnen zugeschriebene Semantik eine Eigenschaft genau dieser Klasse darstellt.
Ein mit der Generalisierung verwandter Begriff ist die **Vererbung** (*inheritance*). Dieser beschreibt den Umsetzungsmechanismus für Beziehungen zwischen Ober– und Unterklassen. Vererbung wird in der Literatur allerdings häufig synonym zur Generalisierung verwendet. Man spricht dabei von **Einfachvererbung** (*single inheritance*), wenn jede Klasse höchstens eine direkte Oberklasse besitzt. Besitzt eine Klasse mehrere direkte Oberklassen spricht man von **Mehrfachvererbung** (*multiple inheritance*). Dieses Prinzip kann allerdings zu einigen Problemen führen (s. (Oestereich 1997), S. 190), und wird deshalb von einigen Programmiersprachen, wie z.B. Java oder Delphi, nicht unterstützt und deswegen hier auch nicht näher erläutert.
Die Notation für Generalisierung ist ein nicht ausgefüllter Pfeil zwischen Ober– und Unterklasse, wobei die Pfeilspitze zur Oberklasse zeigt (—▷). Am Pfeil kann als Beschriftung der Diskriminator angebracht werden. Zudem sind auch andere Beschriftungen üblich, die aber in dieser Arbeit nicht benötigt werden. Genauere Angaben dazu findet man z.B. in Oestereich (1997, S. 187f).

Beispiel 4.5 *Als Beispiel zur Vererbung dient wiederum die Klasse TReadData, diesmal zusammen mit ihren Unterklassen TReadFormattedData und TReadUnformattedData. Abbildung 4.9 zeigt die Struktur der Vererbung dieser Klassen. Wie in Beispiel*

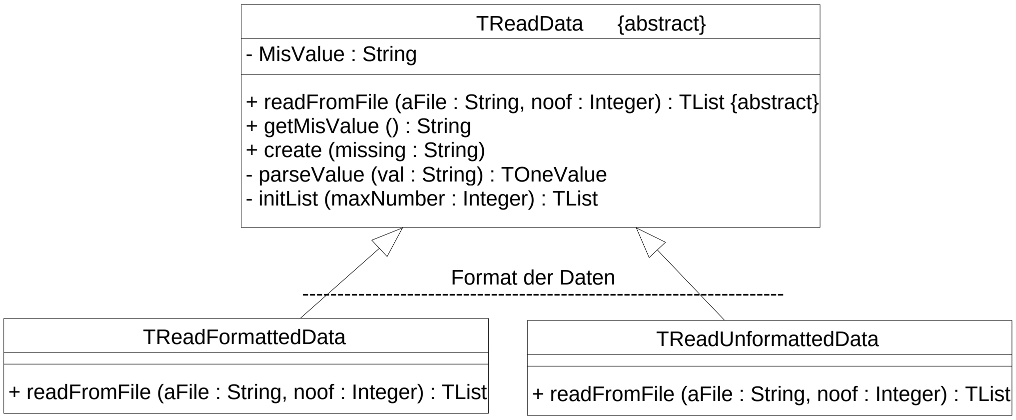


Abbildung 4.9: TReadData und Unterklassen

4.3 schon erwähnt, wird die Klasse TReadData in zwei Unterklassen aufgeteilt, da es zwei Möglichkeiten gibt, wie die Daten in einem Textfile vorliegen. Das Einlesen der Daten muß deswegen auf zwei verschiedene Weisen erfolgen, alle anderen Attribute *und Operationen können von der Oberklasse geerbt werden. Der Diskriminator ist in diesem Falle also das Format der Daten.*

Assoziation Eine Assoziation (*association*) dient der Kommunikation zwischen Objekten. Sie stellt eine Verbindung zwischen verschiedenen Klassen her. Eine konkrete Beziehung zwischen den Objekten dieser Klassen wird **Objektverbindung** (*link*) genannt. Eine Objektverbindung ist also ein Exemplar einer Assoziation.
Im Normalfall besteht eine Assoziation zwischen zwei voneinander verschiedenen Klassen. Es besteht aber auch die Möglichkeit, daß eine Klasse mit sich selbst in Verbindung steht, oder daß mehr als zwei Klassen miteinander in Verbindung stehen. Dies nennt man dann eine **reflexive Assoziation** (*reflexive association*) bzw. eine **n–äre Assoziation** (*n–ary association*). Bei einer reflexiven Assoziation wird davon ausgegangen, daß an der Beziehung zumindest verschiedene Objekte der Klasse beteiligt sind.
Im Diagramm ist die Notation für eine Assoziation eine Linie zwischen den beteiligten Klassen. Handelt es sich um eine n–äre Assoziation, so ist die Darstellung eine Raute, von der Linien zu allen beteiligten Klassen gezogen werden. Jede Assoziation sollte mit einem Namen versehen sein. Dieser Name ist kursiv zu setzen und muß ein Verb sein. Um eine Beziehung besser verstehen zu können, ist es möglich, neben den Namen ein ausgefülltes Dreieck zu setzen, dessen Spitze in die Leserichtung zeigt. An den beiden Enden der Assoziation kann man zusätzlich noch die **Multiplizität** (*multiplication*) angeben. Sie besagt, mit wie vielen Objekten der anderen Klasse ein Objekt der Klasse assoziiert sein kann. Sie wird entweder als einzelne Zahl oder als Wertebereich notiert. Das Zeichen `"*"` steht dabei für beliebig viele. Da die Definition der Multiplizität bei einer n–ären Assoziation schwierig ist, wird sie bei einer solchen Beziehung meist weggelassen. Ebenso kann an den Assoziationsenden noch die Rolle angegeben werden, die die Klasse in der vorliegenden Beziehung spielt. Eine Rolle beschreibt eine Assoziation aus der Sichtweise des Objekts. Eine Beschreibung von zusätzlichen Beschriftungen der Assoziationen, die in der vorliegenden Arbeit nicht benötigt werden, findet sich in Rational Software Corporation (1997b, S. 50ff). Abbildung 4.10 zeigt schematisch die Darstellung einer Assoziation bzw. einer ternären Assoziation.
Ein besondere Art von Assoziationen sind **gerichtete Assoziationen**. Bei diesen kann nur eine Klasse auf eine andere zugreifen und nicht umgekehrt. Im Diagramm wird diese Beziehung durch eine offene Pfeilspitze ($\rightarrow$) dargestellt, die in Richtung der Klasse zeigt, die von der anderen Klasse benutzt werden kann. Ansonsten gleicht die Darstellung der einer normalen Assoziation.

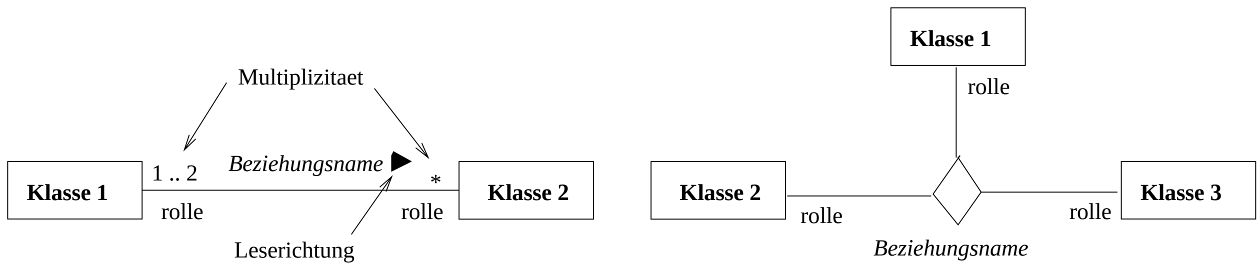


Abbildung 4.10: Notation einer Assoziation und einer ternären Assoziation

Beispiel 4.6 *Das Beispiel zur Assoziation beschäftigt sich wiederum mit dem Einlesen der Daten. Diese werden vom Hauptprogramm aus mittels einer der beiden Unterklassen von TReadData eingelesen (das Hauptprogramm ist in der Objektorientierung ebenfalls eine Klasse). Abbildung 4.11 zeigt diese Situation für das Einlesen von unformatierten Datensätzen. Das Hauptprogramm* MainForm *liest die Daten im Falle eines*

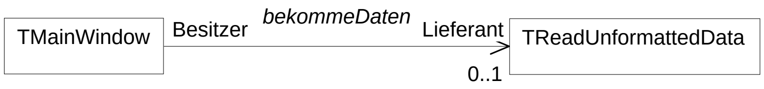


Abbildung 4.11: Assoziation für das Einlesen der Daten

unformatierten Datensatzes mittels der Klasse TReadUnformattedData ein. Da für das Einlesen der Daten ein *Objekt von TReadData ausreicht oder gar keines benötigt wird, wenn es sich um einen formatierten Datensatz handelt, ist die Multiplizität von TReadData in dieser Assoziation* [0..1]*. Außerdem handelt es sich um eine gerichtete Assoziation, da TReadData zum Einlesen der Daten nicht auf Mainform zugreifen muß.*

Aggregation Unter einer Aggregation (*aggregation*) versteht man die Zusammensetzung eines Objekts aus seinen Einzelteilen. Die Klasse, die die Teile enthält, wird **Aggregatsklasse** oder kurz **Aggregat** genannt. Die Aggregatsklasse hat die Aufgabe, die an diese Klasse gestellten Anforderungen an ihre Teile zu delegieren. Sie enthält dazu z.B. Operationen, welche die auszuführenden Aufgaben an ihre Teile weiterleiten. Im Gegensatz zur Assoziation haben hier die in Beziehung stehenden Klassen keine gleichberechtigte Rolle inne, sondern das Aggregat übernimmt die Verantwortung und die Führung.

Die Darstellung im Diagramm ist eine Linie mit einer Raute auf der Seite des Aggregats. Ansonten gibt es dieselben Beschriftungsmöglichkeiten wie bei Assoziationen.

Komposition Eine Komposition (*composition*) ist eine strenge Form der Aggregation. Bei ihr ist die Existenz der Teile vom Aggregat abhängig. Das heißt, bei der Erzeugung oder Zerstörung des Aggregats werden auch seine Teile erzeugt bzw. gelöscht. Die Darstellung der Kompositionsbeziehung ist dabei dieselbe wie die des Aggregats, nur daß die Raute in diesem Falle ausgefüllt ist.

Beispiel 4.7 *Jede untersuchte Variable wird im Programm mittels der Klasse TOneData modelliert. Diese Klasse enthält außer dem Namen, dem Skalenniveau und anderen Informationen zur Variable auch die einzelnen Realisierungen, die in der Klasse TOneValue gespeichert sind. Es besteht also eine Aggregation zwischen TOneData und TOneValue, wobei TOneData das Aggregat ist, da es mehrere Objekte von TOneValue besitzen kann. Da aber die einzelnen Realisierungen ohne die gesamte Variable sinnlos sind, muß diese Beziehung sogar als Komposition betrachtet werden. Abbildung 4.12 zeigt diesen Zusammenhang.*

Abbildung 4.12: Komposition zwischen TOneData und TOneValue

4.2.3 Verhaltensdiagramme

Verhaltensdiagramme (*behavior diagrams*) spiegeln die dynamische Struktur des Modells wider. Dabei gibt es verschiedene Arten von Diagrammen, die jeweils einen anderen Sachverhalt hervorheben. Diese werden in den folgenden Unterkapiteln beschrieben.

4.2.3.1 Interaktionsdiagramme

Ein **Interaktionsdiagramm** (*interaction diagram*) zeigt das Verhaltensmuster von Interaktionen zwischen Objekten. Innerhalb dieser Diagrammart wird zwischen

Sequenzdiagrammen (*sequence diagram*) und Kollaborationsdiagrammen (*collaboration diagram*) unterschieden. Sequenz– und Kollaborationsdiagramme zeigen dieselbe Informationen an, jedoch jeweils aus einer anderen Perspektive. Beim Sequenzdiagramm steht der zeitliche Verlauf der *Nachrichten* (s.u.) im Vordergrund. Das Kollaborationsdiagramm hebt dagegen die Zusammenarbeit der Objekte hervor, der zeitliche Verlauf wird hier nur angedeutet. Ein Kollaborationsdiagramm ist stets durch einfache Umformungen aus einem Sequenzdiagramm zu erhalten. Zudem können die wesentlichen Erläuterungen zum Programm besser mittels eines Sequenzdiagramms dargestellt werden. Aus diesen Gründen werden hier nur die Bestandteile dieser Diagrammart ausführlich beschrieben. Informationen zum Kollaborationsdiagramm findet man unter Rational Software Corporation (1997b, S. 88ff).

Ein Sequenzdiagramm besitzt zwei Dimensionen. An der vertikalen Linie wird die Zeit aufgetragen und an der horizontalen die beteiligten Objekte. Normalerweise erfolgt der zeitliche Ablauf von oben nach unten; zwischen den Objekten gibt es keine bestimmte Ordnung. Die Objekte werden durch gestrichelte senkrechte Linien dargestellt, wobei ganz oben das Objekt selbst in üblicher Darstellung aufgetragen wird.
Die Hauptinformation eines Sequenzdiagramms sind **Nachrichten** (*messages*), welche einen Mechanismus zur Kommunikation von Objekten bieten. Eine Nachricht übermittelt einem Objekt Information darüber, was von ihm ausgeführt werden soll. Das angesprochene Objekt selbst kann nach Erfüllung der Aufgabe, was meist mittels einer Operation geschieht, eine Antwort zurückschicken. Nachrichten werden als waagerechte Pfeile zwischen den Objekten gezeichnet, wobei die Pfeilspitze in Richtung des Empfängers gerichtet ist. Über dem Pfeil wird der Name der Nachricht folgendermaßen angebracht: `nachricht(argumente)`. Die mögliche Antwort wird entweder in Textform `antwort=nachricht()` oder als eigener Pfeil zurück zum Sender der Nachricht dargestellt.
Die gestrichelten Linien zur Objektdarstellung werden teilweise durch nichtausgefüllte senkrechte Balken überlagert, um den Steuerungsfokus zu kennzeichnen. Dieser gibt an, welches Objekt gerade aktiv ist, also die Programmkontrolle besitzt.
Auch das Erzeugen und Entfernen von Objekten im Speicher kann durch Sequenzdiagramme veranschaulicht werden. Die Objekterzeugung wird durch eine Nachricht dargestellt, die auf ein Objektsymbol (also nicht nur auf eine gestrichelte Linie) trifft. Die Entfernung eines Objekts wird ebenfalls durch eine Nachricht dargestellt, bei der am Ende des Steuerungsfokusses ein Kreuz zu sehen ist. Abbildung 4.13 zeigt die

Darstellung eines Nachrichtenaustausches zwischen zwei Objekten.

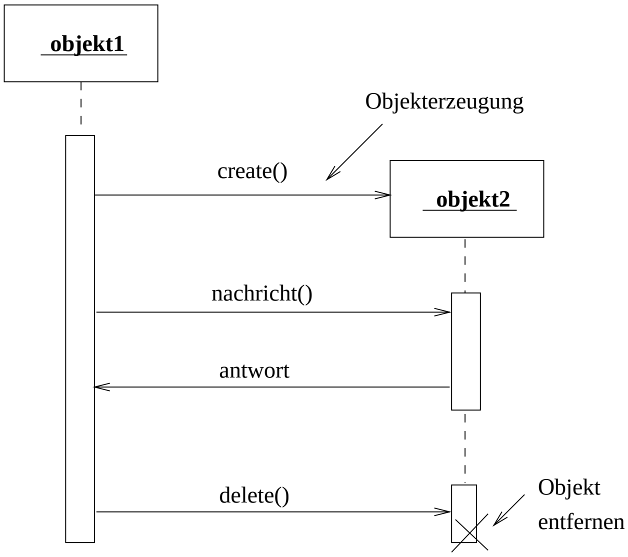


Abbildung 4.13: Sequenzdiagramm zwischen zwei Objekten

Beispiel 4.8 *Das Einlesen von Rohdaten ist ein einfaches Beispiel für ein Sequenzdiagramm, dessen Ablauf hier schematisch erläutert wird. Die Darstellung für ein formatfreies Einlesen von Daten findet sich in Abbildung 4.14.*
Das Objekt mainWindow ist ein Exemplar der Klasse TMainWindow, die im Programm die Hauptverwaltung übernimmt. Beim Einlesen von Daten wird zunächst ein Exemplar von TReadUnformattedData, nämlich readIn, mittels des Aufrufs von create(...) *angelegt. Im nächsten Schritt wird readIn durch die Operation* readFromFile(...) *dazu aufgefordert, Daten aus einer bestimmten Datei auszulesen. Als Antwort sendet readIn die Liste der Variablen, nämlich* wholeList *an mainWindow zurück. Damit ist die Aufgabe von readIn beendet, und das Exemplar wird mittels* Free *destruiert.*

4.2.3.2 Zustandsdiagramm

Ein Zustandsdiagramm (*statechart diagramm*) zeigt eine Folge von Zuständen, die ein Objekt während seiner Lebensdauer einnehmen kann. Dabei reagiert das Objekt jeweils auf von außen kommende Ereignisse. Ein Zustandsdiagramm ist also einer Klasse zugeordnet und spiegelt das interne Verhalten eines Objektes wider. Die Modellierung eines Zustandsdiagrammes ist nur nötig, wenn die Reihenfolge von Ereignissen für ein

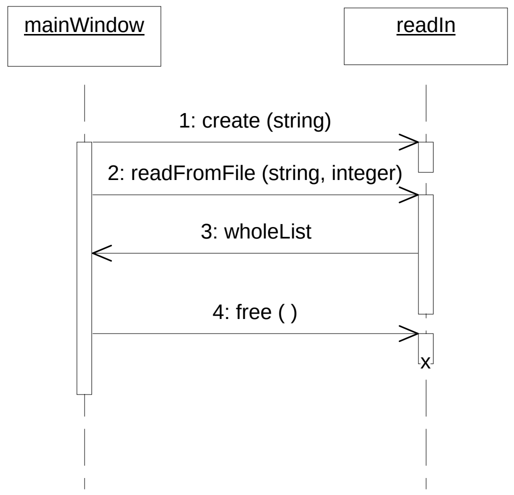


Abbildung 4.14: Sequenzdiagramm für das Einlesen von Daten

Objekt wichtig ist. Im Grunde genommen beschreibt ein Zustandsdiagramm einen deterministischen endlichen Automaten. Dies ist eine hypothetische Maschine, die sich zu jedem Zeitpunkt in einem Zustand befindet. Ein solcher Automat besitzt folgende Komponenten: Eine endliche, nicht–leere Menge von Zuständen, eine endliche, nicht–leere Menge von Ereignissen, Funktionen, die den Übergang von einem Zustand in den nächsten beschreiben, einen Anfangszustand und eine endliche Menge von Endzuständen. Im folgenden werden diese Elemente näher beschrieben und auf die Bestandteile eines Objektes übertragen.

Zustand Ein Zustand (*state*) stellt die Zusammenfassung einer Menge von möglichen Attributswerten dar, die die Objekte einer Klasse einnehmen können. Ein Objekt bleibt dabei für eine bestimmte Zeit in einem Zustand. Es wird nicht für jede Änderung der Attributswerte ein neuer Zustand erzeugt, sondern nur für Ereignisse, die das Verhalten des Objekts maßgeblich beeinflussen. Ein Zustand wird durch ein abgerundetes Rechteck dargestellt, das in bis zu drei Abteilungen unterteilt werden kann. In der obersten Abteilung steht der Name des Zustandes. Wird dieser nicht angegeben, so gilt der Zustand als anonym. In einem Zustandsdiagramm steht dabei jeder anonyme Zustand für einen anderen Zustand. Außerdem kann jeder Zustand eine Menge von Zustandsvariablen enthalten, welche in der mittleren Abteilung des Rechtecks notiert werden. Zustandsvariablen sind dabei Attribute der Klasse, zu der

der Zustand gehört. Um die Übersicht zu gewährleisten, werden nur solche Attribute dargestellt, die für den Zustand wichtig sind. In der untersten Abteilung können zusätzlich auch interne Aktionen oder Aktivitäten angegeben werden, die von Ereignissen ausgelöst werden, den Zustand des Objekts aber nicht ändern. Hierbei sind drei spezielle Auslöser vordefiniert: *entry* löst automatisch eine Aktion beim Eintritt in einen Zustand aus, *exit* löst automatisch eine Aktion beim Verlassen eines Zustands aus und *do* wird immer wieder ausgelöst, solange der Zustand nicht verlassen wird. Abbildung 4.15 zeigt Beispiele für die verschiedenen Notationsweisen von Zuständen.

Abbildung 4.15: Darstellungen von Zuständen

Eine besondere Art von Zuständen sind der Anfangszustand und die Endzustände. Ein Anfangszustand wird durch die Erzeugung eines Objekts erreicht und wird durch einen ausgefüllten Kreis dargestellt. Ein Endzustand stellt die Destruktion eines Objekts dar und durch einen ausgefüllten Kreis notiert, der von einem weiteren Kreis umrahmt wird. Beide Zustände sind in Abbildung 4.16 dargestellt.

Abbildung 4.16: Anfangs– und Endzustand in einem Zustandsdiagramm

Ereignis und Zustandsübergänge Ein Ereignis (*event*) ist ein Vorkommnis, das einen Zustandsübergang (*transition*) auslöst. Es gibt zwei Arten von Ereignissen: Entweder ist eine Bedingung erfüllt, oder ein Objekt erhält eine Nachricht, die auszuführen ist. Im Gegensatz zu Zuständen sind Ereignisse nicht an die Klasse gebunden, zu der das Zustandsdiagramm gehört, sondern werden meistens von anderen Objekten ausgelöst. Ein Zustandsübergang wird durch einen Pfeil von einem Zustand zum nächsten angezeigt, über dem das Ereignis näher beschrieben wird. Zudem gibt es auch Übergänge, die nicht durch ein bestimmtes Ereignis ausgelöst werden, sondern erfolgen, sobald die mit einem Zustand verbundenen Aktionen beendet sind. Solche Übergänge werden ϵ–Übergänge genannt und sind dadurch gekennzeichnet, daß sie nicht beschriftet sind.

Beispiel 4.9 *Die Klasse TReadUnformattedData, die für das Einlesen von Daten zuständig ist, hat ein äußerst einfaches Zustandsdiagramm, das in Abbildung 4.17 zu sehen ist.*

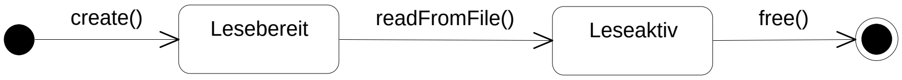


Abbildung 4.17: Zustandsdiagramm für TReadUnformattedData

Nach dem Kreieren eines Exemplars mittels create(), *befindet sich die Klasse im Zustand* Lesebereit. *Durch das Ereignis* readFromFile() *wird sie in den Zustand* Leseaktiv *versetzt. Von diesem Zustand aus führt das Ereignis* create() *zum Endzustand, also zur Destruktion eines Exemplars von TReadUnformattedData.*

Kapitel 5

Grundlegendes Design des Programms

Dieses Kapitel gibt einen groben Überblick über die zum jetzigen Zeitpunkt vorhandene Programmstruktur. Es werden mehrere Aspekte behandelt, die für verschiedene Leser unterschiedlich interessant sein können. In Abschnitt 5.1 wird die Benutzerschnittstelle des Programms erläutert, wobei auch das Format der vom Programm verwendeten Dateien beschrieben wird. Dieser Abschnitt ist also vor allem für Anwender des Programms gedacht. In Abschnitt 5.2 werden die wichtigsten im Programm modellierten Elemente vorgestellt. Es wird zunächst die Grobstruktur des Programms beschrieben und davon ausgehend auf die einzelnen Teilaspekte näher eingegangen. Dieser Abschnitt richtet sich daher an die Leser, die sich näher mit der Modellierung des Programms beschäftigen möchten. Abschließend werden in Abschnitt 5.3 offene Fragen vorgestellt, die bisher bei dem Entwurf des Programms noch keine Beachtung fanden. Es bleibt anzumerken, daß die Modellierung bzw. die Beschreibung des Graphen nicht Bestandteil dieses Kapitels ist, da dieser Problematik Kapitel 6 gewidmet ist.

5.1 Modellierung der Benutzerschnittstelle

Die Benutzerführung ist im Verlauf der Zeit ein äußerst wichtiges Kriterium beim Programmentwurf geworden. Gab es zu Beginn der Computerära nur äußerst spartanische Eingabemöglichkeiten für den Benutzer, so ist es heute wichtig, den Umgang mit einem Programm möglichst klar zu halten, um den Anwender die Benutzung komfortabel zu gestalten.

In Abschnitt 5.1.1 werden einige Anforderungen genannt, die an die Benutzerführung heutiger Programme gestellt werden, und deren Umsetzung im Programmentwurf erläutert. Im darauffolgenden Abschnitt werden die verschiedenen Dateiformate vorgestellt, die im Programm verwendet werden, wobei die beiden Arten von Quelldateien detailliert beschrieben werden. Abschnitt 5.1.3 gibt schließlich einen Überblick über den Aufbau und die Menüstruktur des Systems, kann dabei aber nicht ein Benutzerhandbuch ersetzen. Eine ausführliche Darstellung der Arbeitsschritte, die die Berechnung eines graphischen Modells vorbereiten, finden sich in Kapitel 7.

5.1.1 Anforderungen an die Benutzerschnittstelle

Schon seit einiger Zeit gibt es Bestrebungen, eine möglichst einheitliche Benutzerschnittstelle zu bieten, um den Umgang mit Programmen zu erleichtern. Eine Vorreiterrolle spielte dabei das Deutsche Institut für Normung, das schon 1988 eine Richtlinie zur Gestaltung von Benutzerdialogen herausgab, deren Inhalt und Umsetzungen im Programm im folgenden diskutiert werden (siehe (Deutsches Institut für Normung 1988)).

Aufgabenangemessenheit *Ein Dialog ist aufgabenangemessen, wenn er die Erledigung der Arbeitsaufgabe des Benutzers unterstützt, ohne ihn durch Eigenschaften des Dialogsystems unnötig zu belasten.*

Im Programm wird darauf geachtet, daß die verwendeten Dialoge einfach zu bedienen und nicht mit zu vielen Informationen überladen sind.

Selbstbeschreibungsfähigkeit *Ein Dialog ist selbstbeschreibungsfähig, wenn dem Benutzer auf Verlangen Einsatzzweck sowie Leistungsumfang des Dialogsystems erläutert werden können, wenn jeder einzelne Dialogschritt unmittelbar verständlich ist oder der Benutzer auf Verlangen dem jeweiligen Dialogschritt entsprechende Erläuterungen erhalten kann.*

Beim Entwurf des Systems wurde Wert darauf gelegt, daß in jedem Dialog zu jedem Thema Hilfe angefordert werden kann. Ein erfahrener Benutzer wird so vom Hilfesystem in seiner Arbeit nicht unnötig aufgehalten. Ein Anfänger hingegen kann zu jedem Zeitpunkt die angebotene Hilfe nutzen. Wegen der Komplexität der Erstellung graphischer Modelle ist zudem die Überlegung wert, ob nicht ein

Lernprogramm, wie es zum Beispiel in Word for Windows enthalten ist, mitgeliefert wird, in dem der Benutzer sich schrittweise anhand eines Beispiels durch das Programm durcharbeiten kann.

Steuerbarkeit *Ein Dialog ist steuerbar, wenn der Benutzer die Geschwindigkeit des Ablaufs sowie die Auswahl und Reihenfolge von Arbeitsmitteln oder Art und Umfang von Ein– und Ausgaben beeinflussen kann.*

Im Programmentwurf wird dieser Forderung Rechnung getragen, indem der Benutzer zum Beispiel die Anzahl der vom Programm automatisch ausgeführten Arbeitsschritte selbst bestimmen kann und sich soviel Information in der graphischen Darstellung des Modells ausgeben lassen kann, wie er für nötig hält (siehe Abschnitt 6.2.2). Zudem soll auch im Rahmen des Ergebnisprotokolls eingestellt werden, welche Information erscheint.

Erwartungskonformität *Ein Dialog ist erwartungskonform, wenn er den Erwartungen der Benutzer entspricht, die sie aus Erfahrungen mit bisherigen Arbeitsabläufen oder aus der Benutzerschulung mitbringen sowie den Erfahrungen, die sie sich während der Benutzung des Dialogsystems und im Umgang mit dem Benutzerhandbuch bilden.*

Es wurde darauf geachtet, die Menüstruktur so zu erstellen, daß sie bekannten Windows–Programmen ähnelt. Es gibt dazu einen sogenannten Styleguide, der von der Firma Microsoft herausgegeben wurde (Microsoft Corporation 1992). So ist die Hilfe–Menüleiste ganz rechts zu finden und auch die Fenster–Menüleiste enthält Punkte, wie sie in Windows–Programmen üblich sind. Außerdem entspricht die Menüstruktur Design–Vorgaben, wie sie in Shneiderman (1992, Kapitel 3), beschrieben werden. Als Beispiel soll hier die Anordung der Punkte im Menü „Analyse“ dienen, die eine zeitliche Reihenfolge widerspiegeln: Es müssen zuerst die Modellvariablen festgelegt, die Abhängigkeitskette erstellt, anschließend das Screening durchgeführt werden, bevor eine Berechnung des Graphen stattfinden kann. Im Menü wurde diese Struktur nachgebildet, indem die Punkte der Reihe nach von oben nach unten angeordnet sind, wie es in Abbildung 5.1 zu sehen ist.

Fehlerrobustheit *Ein Dialog ist fehlerrobust, wenn trotz erkennbar fehlerhafter Eingaben das beabsichtigte Arbeitsergebnis mit minimalem oder ohne Korrekturauf-*

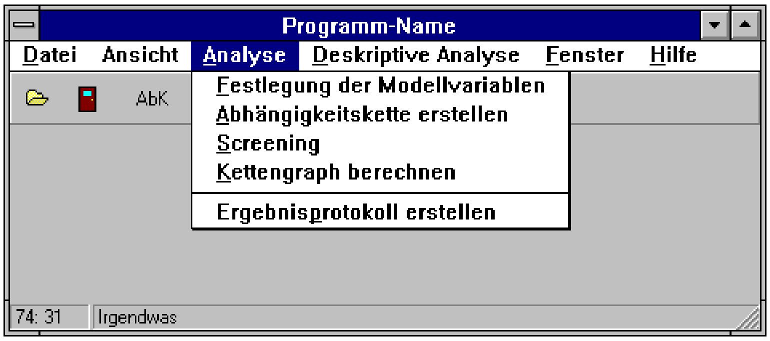


Abbildung 5.1: Aufbau des Menüs „Analyse"

wand erreicht wird. Dazu müssen dem Benutzer die Fehler zum Zwecke der Behebung verständlich gemacht werden.

Die Fehlerrobustheit ist ein äußerst wichtiger Punkt im Entwurf des Programms, da ein fehlendes Abfangen von Fehlern sehr schnell zu einer Verärgerung des Benutzers führt. Deswegen wird bei allen Dialogen versucht, die Eingaben möglichst rasch auf Fehler zu prüfen und dem Anwender die Fehlerursache möglichst genau mitzuteilen.

Zusätzlich zur Fehlerrobustheit wird im Programm auch dafür gesorgt, daß viele Fehler überhaupt nicht entstehen können. So ist es zum Beispiel nicht sinnvoll, einen Kettengraphen berechnen zu lassen, wenn die Abhängigkeitskette noch nicht erstellt wurde. Deswegen kann der Menüpunkt „Kettengraph berechnen" erst angewählt werden, wenn zuvor eine Abhängigkeitskette erstellt wurde. Auch in den Dialogen selbst wird so weit wie möglich darauf geachtet, Auswahllisten oder Schalter zu verwenden, um den Benutzer vor falschen Eingaben zu schützen. Dazu gehört auch, daß der Benutzer davon in Kenntnis gesetzt wird, wenn eine Aktion von ihm große Auswirkungen im Programm hat. Wird beispielsweise das Skalenniveau der Variablen mitten in der Graphberechnung geändert, führt dies zu einer kompletten Neuberechnung des Graphen. Sobald der Benutzer diese Aktion auslöst, erscheint daher ein Fenster mit einer Warnung, wie in Abbildung 5.2 dargestellt.

Abbildung 5.2: Warnung beim Ändern des Skalenniveaus

5.1.2 Datenstrukturen

Im Programm gibt es mehrere Arten von Dateien, die sich durch eine artspezifische Endung unterscheiden, um den Überblick über die verschiedenen Dateien zu bewahren. Die Endungen sind an bekannte Windows–Konventionen angelehnt. Zum einen sind Binär–Dateien vorhanden, die im Laufe der Programmausführung angelegt werden können und wichtige Datenstrukturen abspeichern. Dazu gehört die Sitzungsdatei, die die Daten zusammen mit relevanten Informationen enthält. Zu diesen Informationen gehört das Skalenniveau der Variablen, die Modellvariablen mit ihren Interaktionen und quadrierten Variablen und die Anordnung der Variablen in der Abhängigkeitskette. Sitzungsdateien besitzen die Endung `.sav`. Der Kettengraph wird in verschiedenen Stadien der Berechnung auf Wunsch in Graphikdateien abgelegt. Um die Möglichkeit zu bieten, den Kettengraphen in anderen Zeichenprogrammen weiterzuverarbeiten, sollte das abgespeicherte Dateiformat JPEG sein, da dies von Delphi unterstützt wird und zudem ein gängiges Dateiformat für Graphiken ist. Graphikdateien erhalten deshalb die Endung `.jpg`. Die andere Gruppe von Daten umfaßt Textdateien. Dazu gehört zum einen das Ergebnisprotokoll, das vom Benutzer gewünschte Informationen zum berechneten Modell enthält. Das Ergebnisprotokoll hat die Endung `.log`.
Andere Textdateien sind die Quelldateien, die die zu untersuchenden Daten enthalten. Diese können in zwei verschiedenen Formaten vorliegen, die in der Statistik üblich sind. Beiden Formaten ist gemeinsam, daß das Zeichen für fehlende Werte frei wählbar ist und nur numerische Werte enthalten sind. Bei reellen Zahlen ist darauf zu achten, daß als Trennzeichen zu den Dezimalstellen ein Punkt verwendet wird und kein Komma. Zudem dürfen in jeder Quelldatei nur die Ausprägungen der Variablen enthalten sein, da die Namen der Variablen im Programm vergeben werden. Sind in den Rohdaten fehlende Werte enthalten, wird zum jetzigen Zeitpunkt davon ausgegangen, daß diese zufällig aufgetreten sind. Dies bedeutet, daß die Beobachtungen, in denen fehlende

Werte auftreten, nicht aus der Analyse entfernt werden. Sollen die Beobachtungen entfernt werden, so muß der Benutzer vor Programmausführung ein sogenanntes geschlossenes File erstellen. Ein geschlossenes File ist eine Rohdaten–Datei, in denen keine fehlenden Werte vorkommen. Die beiden Formate unterscheiden sich prinzipiell dadurch, daß im einen Fall die Position der einzelnen Variablen fest vorgeschrieben ist, im folgenden formatgebundenes Einlesen genannt, und im anderen Fall die Variablen an verschiedenen Positionen stehen können, welche durch Trennzeichen getrennt werden, im folgenden als formatfreies Einlesen bezeichnet. Der Aufbau einer Datei für das formatfreie Einlesen ist in Tabelle 5.1 zu sehen.

$$x_{11} < \text{Trennzeichen} > x_{21} < \text{Trennzeichen} > \cdots < \text{Trennzeichen} > x_{p1}$$
$$x_{12} < \text{Trennzeichen} > x_{22} < \text{Trennzeichen} > \cdots < \text{Trennzeichen} > x_{p2}$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$x_{1n} < \text{Trennzeichen} > x_{2n} < \text{Trennzeichen} > \cdots < \text{Trennzeichen} > x_{pn}$$

Tabelle 5.1: Datenformat für formatfreies Einlesen

Dabei ist n die Anzahl der Ausprägungen und p die Anzahl der Variablen. Als Trennzeichen ist eine beliebige Anzahl von Leerzeichen oder Tabulaturzeichen erlaubt. Die einzelnen Werte müssen zudem nicht genau untereinander stehen. Ein Beispiel eines kleinen Datensatzes findet sich in Tabelle 5.2.

0.7	1	3	.
1.9	0	7	2.1
.	0	4	17.9
80.97	1	3	0.1

Tabelle 5.2: Beispiel für formatfreies Einlesen

Tabelle 5.3 zeigt Dateiformat für formatgebundenes Einlesen. Dabei ist n wieder die Anzahl der Ausprägungen und p die Anzahl der Variablen; k, l, m stehen für die Anzahl der Stellen, die jede Variable höchstens benötigt. Besitzt eine Variable eine unterschiedliche Anzahl von Stellen, so muß mit einer führenden `0`–Sequenz aufgefüllt werden. Zudem gilt: $x_{ij}^{(1)}, \ldots, x_{ij}^{(k)} = x_{ij}$ für $i \in \{1, \ldots, p\}$ und $j \in \{1, \ldots, n\}$.

$$\begin{array}{cccccccccc}
x_{11}^{(1)} & \cdots & x_{11}^{(k)} & x_{21}^{(1)} & \cdots & x_{21}^{(l)} & \cdots & x_{p1}^{(1)} & \cdots & x_{p1}^{(m)} \\
x_{12}^{(1)} & \cdots & x_{12}^{(k)} & x_{22}^{(1)} & \cdots & x_{22}^{(l)} & \cdots & x_{p2}^{(1)} & \cdots & x_{p2}^{(m)} \\
\multicolumn{10}{c}{\dotfill} \\
x_{1n}^{(1)} & \cdots & x_{1n}^{(k)} & x_{2n}^{(1)} & \cdots & x_{2n}^{(l)} & \cdots & x_{pn}^{(1)} & \cdots & x_{pn}^{(m)}
\end{array}$$

Tabelle 5.3: Datenformat für formatgebundenes Einlesen

Das vorhergehende Beispiel ist auf diesen Fall übertragen in Tabelle 5.4 dargestellt, wobei das Zeichen für fehlende Werte diesmal die `9` ist.

```
000.7130009
001.90702.1
000090417.9
80.971300.1
```

Tabelle 5.4: Beispiel für formatgebundenes Einlesen

5.1.3 Aufbau der Benutzerschnittstelle

Das Hauptfenster der Anwendung besteht aus einer Titelleiste, einer Menüzeile, einer Leiste mit Schaltflächen für häufig auszuführende Befehle und einer Statuszeile, die kurze Informationen zu den gerade ausgewählten Menüpunkten bietet. Die Arbeitsfläche des Hauptfensters bleibt leer und dient als Rahmen für die Formulare, die die eigentliche Information darstellen. Dies sind zum einen Dialoge, mit deren Hilfe der Benutzer Aktionen ausführen kann, und zum anderen Formulare, in denen der Graph in den verschiedenen Entwicklungsstadien dargestellt wird. Der Begriff Formular dient hierbei zur Verallgemeinerung von verschiedenen Arten von Fenstern, die im Laufe eines Programms auftreten können. Die Formulare des Graphen sind dabei als sogenannte MDI–Kindfenster (MDI steht für *multiple document interface*) konzipiert, was bedeutet, daß mehrere davon im Programm gleichzeitig geöffnet bleiben können. Ein weiterer Vorteil des MDI–Konzepts besteht darin, daß die Menüstruktur des Programms dynamisch verändert werden kann. So wird die Hauptmenüleiste bei der Anzeige eines Graphen um einige Punkte ergänzt, was in Kapitel 6.2.1 näher beschrieben wird. Eine Darstellung des Programmfensters ohne Graph ist in Abbildung 5.3 zu sehen.

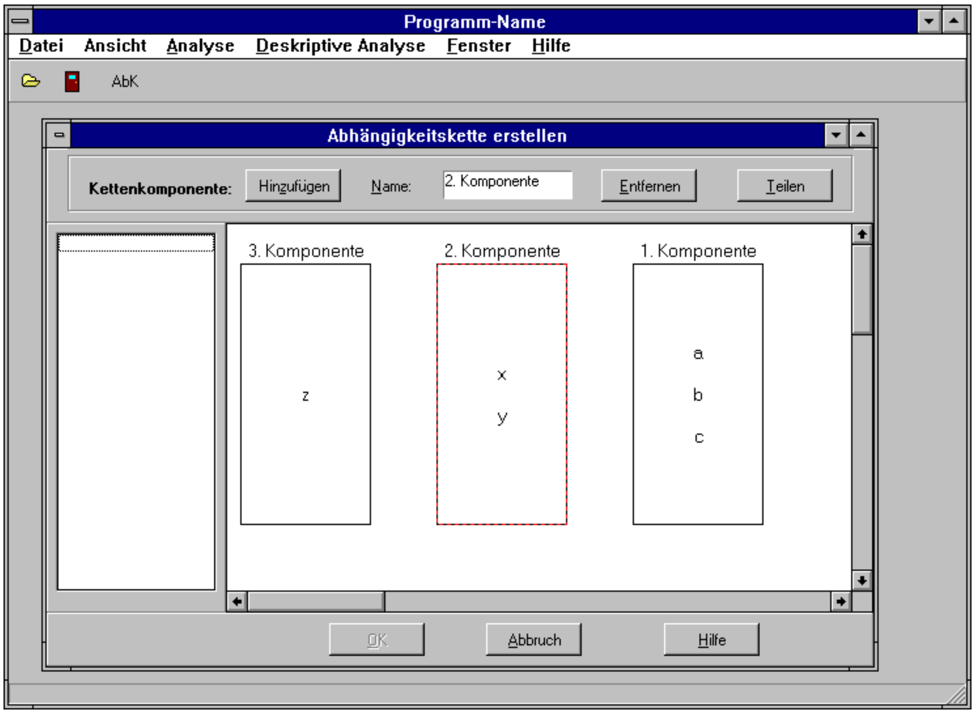


Abbildung 5.3: Hauptformular der Anwendung

Es folgt nun eine kurze Vorstellung der einzelnen Menüpunkte. Änderungen in der Menüstruktur, die sich bei der Anzeige eines Graphen ergeben, werden in Abschnitt 6.2.1 beschrieben. Im Menü „Datei“ findet man Punkte zum Speichern, Laden und Drucken von Dateien und zum Schließen der Anwendung. Mit Hilfe der Punkte im Menü „Ansicht“ können wichtige Datenstrukturen des Programms angezeigt werden, wie die Rohdaten selbst, das Skalenniveau und die Abhängigkeitskette. Das Menü „Analyse“ enthält diejenigen Punkte, die zur Erstellung eines Graphen angewählt werden können. Außerdem kann hier das Ergebnisprotokoll gestartet werden. Das Menü „Deskriptive Analyse“ enthält bisher nur einen Punkt: die Änderung des Skalenniveaus. Das Menü soll aber in Zukunft um Punkte ergänzt werden, die eine deskriptive Analyse der Variablen erlauben. Im Menü „Fenster“ können die verschiedenen Formulare des Programms angeordnet werden. Das Menü „Hilfe“ stellt die Hilfefunktion zum Programm zur Verfügung. Zusätzlich sind hier auch Informationen zur Programmversion abrufbar, wie es in Windows–Programmen üblich ist.

5.2 Modellierung wichtiger Programmelemente

Eine elementare Aufgabe beim Entwurf eines Softwaresystems ist es, ein grundlegendes Konzept zu finden, das in sich stimmig ist und leicht erweitert werden kann. Denn nur so ist es möglich, mit mehreren Personen zusammen an einem Software–Projekt zu arbeiten und das fertiggestellte Programm zu warten. Dieser Abschnitt stellt das Konzept für das zu entwickelnde System vor und gibt einen Einblick in die Implementierung wichtiger Elemente.

Zunächst wird die Einteilung des Programms in die grundlegenden Pakete beschrieben, die dann in den darauffolgenden Abschnitten zum Teil anhand von Beispielen genauer diskutiert werden. Hierbei werden auch interessante Details der Implementierung vorgestellt. Elemente, die sich mit der Erzeugung und Darstellung des graphischen Modells befassen, werden nicht in diesem Kapitel besprochen; ihnen ist Kapitel 6 gewidmet.

5.2.1 Allgemeiner Programmaufbau

Der Entwurf des Programms muß mehrere Dinge integrieren. Wie in jedem System ist es wichtig, eine gute Abstraktion der Daten zu finden, um den Umgang mit diesen innerhalb der Implementierung zu erleichtern. Im objektorientierten Paradigma bedeutet dies, geeignete Klassen zur Datenhaltung zu bilden. Zudem muß sichergestellt sein, daß die für die Erstellung eines graphischen Modells nötigen statistischen Verfahren einfach implementiert werden können. Ebenso spielt die Visualisierung des Graphen eine große Rolle, da gerade hierbei die Vorteile einer computergestützten Analyse zur Geltung kommen. Wichtig ist auch eine gute Online–Hilfe, die dem Benutzer sowohl Information zum statistischen Hintergrund als auch zur Bedienung des Programms liefert. Auf diese wird aber nicht näher eingegangen, da die Problematik einer Online–Hilfe eher in ihren Inhalten als in der technischen Realisierung zu sehen ist. Aus diesen Anforderungen heraus ist das Programm in die in Abbildung 5.4 dargestellten logischen Pakete eingeteilt.
Im Paket *Datenstrukturen* werden die im Programm betrachteten Daten systematisch zusammengefaßt. Die Elemente, die zur Visualisierung des Graphen benötigt werden, sind im Paket *Graphische Elemente* enthalten. Das Paket *Mathematische Funktionen* beinhaltet die statistischen Funktionen und Verfahren zur Erstellung des Graphen. Die einzelnen Formulare, die im Laufe des Programmes auftreten, werden im Paket *Formulare* gruppiert. Alle anderen Klassen, die sich nicht in eine der obigen Kategorien

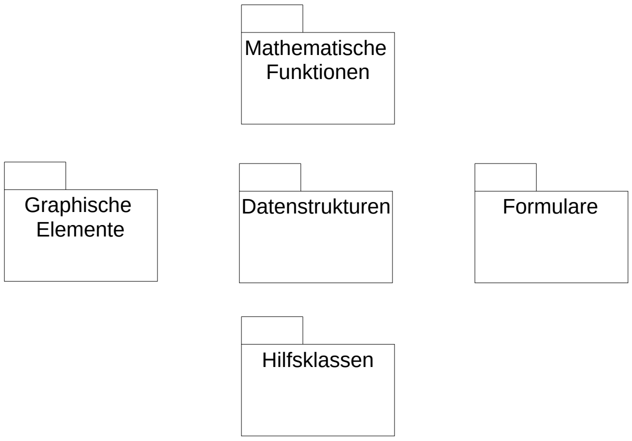


Abbildung 5.4: Grundlegende Pakete des Programms

einteilen lassen, sind im Paket *Hilfsklassen* zusammengefaßt. Es handelt sich dabei um Klassen, die entweder eine kleine Aufgabe im Programm zu erledigen haben oder das Zusammenspiel anderer Klassen unterstützen.
Tabelle 5.5 zeigt die Zuordnung der gebildeten logischen Einheiten zu den in Delphi vorhandenen physischen Units. Zusätzlich ist auch eine kurze Information zu finden, welche Aufgabe die einzelne Unit besitzt. Dabei ist anzumerken, daß das Paket *Formulare* in viele Units zerfällt, weil Delphi automatisch beim Erstellen eines Formulars eine neue Unit für dieses anlegt.

5.2.2 Paket Datenstrukturen

Die Modellierung der Daten ist in zwei Quelldateien aufgeteilt. Die erste Datei enthält die Klassenhierarchie, die die betrachteten Variablen und die dazugehörigen Effekte programmtechnisch umsetzt. In der anderen Datei wird die mathematische Beschreibung eines Kettenmodells gekapselt.
Diese beiden Dateien werden in den folgenden Abschnitten näher beschrieben.

5.2.2.1 Modellierung der Variablen

Jede Variable besitzt mehrere Ausprägungen, deren Modellierung als erstes untersucht wird. Eine Variablenausprägung wird im Programm durch die Klasse *TOneValue* umgesetzt, die in Abbildung 5.5 zu sehen ist.

Paket	**Unit**	**Aufgabe**
Datenstrukturen	Data	Haltung der Variablen
	Structures	Haltung des Kettengraphen
Graphische Elemente	GraphElements	Elemente eines Graphen
Mathem. Funktionen	Regressions	Regressionsverfahren
	Strategy	Selektionsstrategien
Formulare	DataScreening	Screening–Dialog
	Dependency	Erstellung der Abhängigkeitskette
	FileFormat	Abfrage nach Format der Rohdaten–Datei
	GraphWindow	Anzeige des Graphen
	MainForm	Hauptfenster
	ModelVar	Bestimmung der Modellvariablen
	NameVariables	Benennung der Variablen
	NewGraph	Dialog für die Einstellungen am Graphen
	Scales	Festlegung des Skalenniveaus
	ScrInfo	Information zum Screening
	SelectionDialog	Wahl der kritischen Werte der Sel.Strategie
Hilfsklassen	ReadInData	Einlesen der Daten
	Translators	Verbindung Modell – Graph

Tabelle 5.5: Pakete und Units

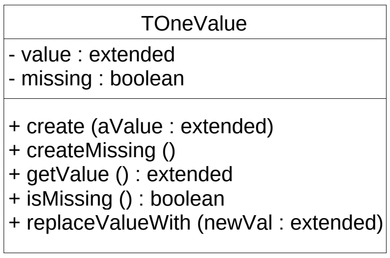


Abbildung 5.5: Klasse für die Modellierung einer Ausprägung

Da fehlende Werte im Programm berücksichtigt werden, kann eine Ausprägung entweder eine Zahl sein, oder sie ist nicht vorhanden. In der Klasse wird diese Tatsache so realisiert, indem bei echter Ausprägung `value` belegt ist und bei fehlender Ausprägung `missing` auf `True` gesetzt wird. Um Ausprägungen möglichst exakt im Programm umzusetzen, ist als deren Typ `Extended` gewählt, da dieser eine Fließkommazahl mit der höchstmöglichen Genauigkeit darstellt. Je nachdem, ob ein Wert vorliegt oder nicht, wird ein Element der Klassse durch die Konstruktoren `create` bzw. `createMissing` erzeugt. Zudem ist es auch möglich, den Wert einer Ausprägung zu ändern, was mit Hilfe der Operation `replaceValueWith` erfolgt.

Im Programm gibt es vier verschiedene Arten von Variablen. Für jeden Variablentyp wird eine Klasse gebildet, die jeweils Unterklassen der Klasse *TOneData* ist, in der die Gemeinsamkeiten aller Variablentypen beschrieben sind. Der Aufbau von *TOneData* ist in Abbildung 5.6 zu sehen.

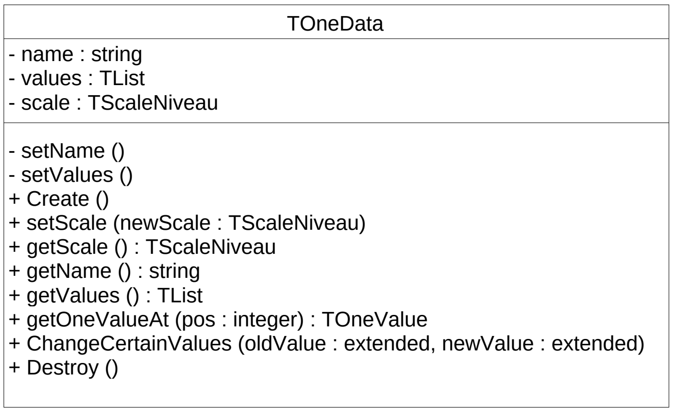


Abbildung 5.6: Aufbau der Klasse TOneData

Jede im Programm vorhandene Variable besitzt einen Namen, ein bestimmtes Skalenniveau und eine Liste von Ausprägungen, die im Programm als Attribute von `TOneData` realisiert werden. Das Attribut `name` ist eine beliebig lange Zeichenkette, `values` speichert die Ausprägungen in Form einer dynamischen Liste ab, und `scale` beschreibt das Skalnenniveau einer Variable. Der Typ des Skalenniveaus, `TScaleNiveau`, ist ein Aufzählungstyp, der in Abbildung 5.7 dargestellt ist. Eine Variable hat noch kein Skalenniveau zugewiesen bekommen, wenn `scale` den Wert `none` besitzt; sie ist diskret binär nominal, wenn `scale` den Wert `disBinNom` besitzt. Die anderen Abkürzungen ergeben sich entsprechend.
Die Hierarchie der Variablen ist in Abbildung 5.8 dargestellt. Ihr Aufbau wird im folgenden nur anhand ihrer Attribute beschrieben. Eine Beschreibung der jeweiligen Operationen erfolgt nicht, da die wesentlichen Eigenschaften einer Klasse schon durch

```
TScaleNiveau = (none, disBinNom, disBinOrd, disKatNom, disKatOrd, conNor,
conOth);
```

Abbildung 5.7: Der Aufzählungstyp TScaleNiveau

ihre Attribute gegeben sind.

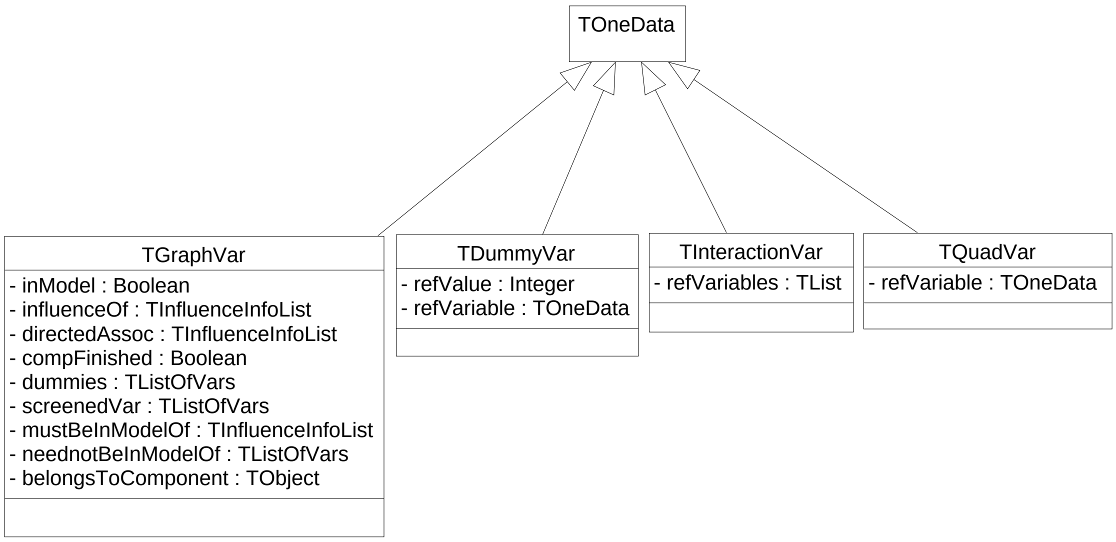


Abbildung 5.8: Hierarchie der Klassen, die die Variablen modellieren

Die beobachteten Variablen, die auch im Graph sichtbar sind, werden in der Klasse `TGraphVars` modelliert. Die Klasse ist dabei so konzipiert, daß sie die Variable sowohl als Zielvariable als auch als Einflußgröße widerspiegelt. Das Attribut `inModel` gibt an, ob die Variable im Modell untersucht werden soll. In den Attributen `influenceOf` und `directedAssoc` wird abgespeichert, welche anderen beobachteten Variablen die jeweilige Variable und in welchem Ausmaß sie diese beeinflußt. Ist eine Variable schon als Zielvariable betrachtet worden, so wird `compFinished` auf `True` gesetzt. Mit Hilfe des Attributs `screenedVars` werden diejenigen Effekte gespeichert, die im Screening mit der gegebenen Zielvariable auffällig sind. Fügt ein Benutzer Assoziationen hinzu oder entfernt gegebene Assoziationen, so wird dies in den Attributen `mustBeInModelOf` bzw. in `needNotBeInModelOf` festgehalten. Genauer werden hier die Zielvariablen gespeichert, bei denen die Assoziationen mit der gegebenen Variable als Einflußgröße vom Benutzer geändert werden. Das Attribut `belongsToComponent` enthält eine Referenz auf die Kettenkomponente, zu der die Variable gehört. Im Attribut `dummies` werden zugehörige Dummy–Variablen abgespeichert (s. unten). In der Klasse `TGraphVar` sind auch noch weitere Attribute denkbar, die einige deskriptive Merkmale der Variable beschreiben, wie z.B. den minimal und maximal angenommenen Wert, den Mittelwert und ähnliche Kenngrößen.

Zur Auswertung von diskreten multikategorialen Variablen ist es nötig, Dummy–Variablen anzulegen. Diese werden in der Klasse `TDummyVar` modelliert. Das Prinzip besteht darin, für jede Ausprägung eine neue Variable anzulegen. Sie erhält im Programm den Wert `1` an der Stelle, an der die Originalvariable die referenzierte Ausprägung besitzt und bei allen anderen Ausprägungen den Wert `0`. Eine Ausprägung ist linear abhängig von allen anderen Variablen, und zwar hat sie an den Stellen die Ausprägung `1`, an der alle anderen Variablen `0` sind, ansonsten ist sie `0`. Diese Ausprägung nennt man auch Referenzkategorie. Im Programm ist jeweils die größte Ausprägung einer Variable die Referenzkategorie. Die Klasse `TDummyVar` besitzt als Attribute den Wert der Originalvariable, an der die Dummy–Variable `1` ist, nämlich `refValue` und die dazugehörige Originalvariable, die in `refVariable` gespeichert wird. Hierbei ist zu bemerken, daß auch diskrete binäre Variablen intern auf die Werte `0` und `1` gesetzt werden, was aber in der Originalvariablen geschieht.

In der Klasse `TInteractionVar` werden Interaktionen und in der Klasse `TQuadVar` quadrierte Variablen modelliert, die im Screening auffällig sind oder im Laufe der Selektionsstrategie erzeugt werden. Zwar werden bis jetzt bei der Erstellung eines Graphen nur paarweise Interaktionen betrachtet; die dazugehörige Klasse ist aber so angelegt, daß sie theoretisch auch Interaktionen höherer Ordnung abspeichern kann. Dazu ist als Typ für das Attribut `refValues`, das die an der Interaktion beteiligten Variablen abspeichert, `TListOfVars` gewählt worden, was der Typ für eine beliebige Anzahl von Variablen ist.

5.2.2.2 Modellierung der Struktur des graphischen Modells

Bei der programmtechnischen Umsetzung eines graphischen Modells wird versucht, dessen allgemeinen Aufbau wiederzugeben. So werden die einzelnen Elemente eines graphischen Modells und ihre Zusammenhangsstruktur in Klassen und geeignete Assoziationen zwischen den Klassen umgesetzt. Abbildung 5.9 zeigt die so gefundene

Struktur.

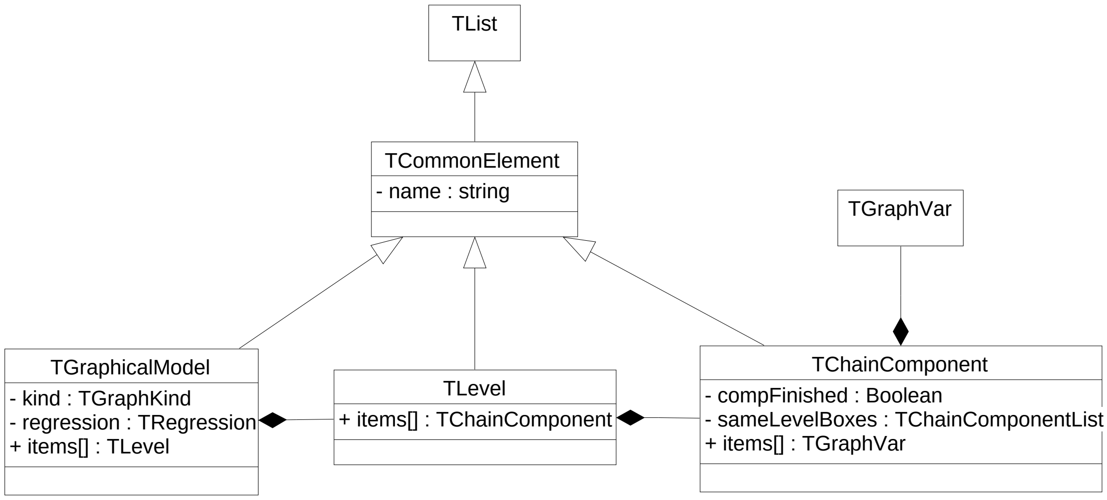


Abbildung 5.9: Modellierung eines graphischen Modells

Die Klasse `TGraphicalModel` ist die Abstraktion des gesamten graphischen Modells. Diese Klasse ist, wie all die anderen Klassen, die die Struktur des Modells widerspiegeln, eine Unterklasse zu `TCommonElement`. Diese besitzt als einziges Attribut `name`, das den Namen eines Elements des Graphen enthält. `TCommonElement` ist zudem von der in Delphi schon vorgebenen Klasse `TList` abgeleitet. Diese Klasse ist die Oberklasse aller dynamischen Listen. Sie wurde als Oberklasse gewählt, da jedes Element des Modells andere Elemente enthält, die in einer solchen Liste gespeichert werden.

Das Attribut `regression` der Klasse `TGraphical Model` beinhaltet die gewählte Regressionsart für ein vorhandenes Objekt der Klasse. Der dazugehörige Typ `TRegression` ist dabei die Oberklasse aller im Programm vorhandenen Regressionsarten. Da alle Unterklassen mit dem Typ der Oberklasse zuweisungskompatibel sind, ist es so möglich, jede beliebige Regression im graphischen Modell durchzuführen. Im Attribut `kind` wird vermerkt, ob es sich um einen Konzentrations– oder Kovarianzgraphen handelt. Der Typ dieses Attributs, `TGraphKind`, ist ein Aufzählungstyp, der die verschiedenen Arten von Graphen als Elemente besitzt. Abbildung 5.10 stellt den Aufbau von `TGraphKind` dar.
Die wichtigsten Bestandteile eines graphischen Modells sind aber die verschiedenen Stufen von Variablen, die in einer dynamischen Liste im Attribut `items` gespeichert werden. Die bestehende Ordnung auf den Stufen, also die Einteilung in primäre Ein-

```
TGraphKind = (covariance, concentration);
```

Abbildung 5.10: Aufzählungstyp TGraphKind

flußgrößen, Stufen von Intermediaten und primäre Zielgrößen, wird nicht extra in einem Attribut gespeichert, sondern ist durch den Index des jeweiligen Elements der Liste festgelegt. Die primären Einflußgrößen liegen in `items[0]` und die primären Zielgrößen in `items[Count-1]`, wobei `Count` die aktuelle Anzahl von Elementen in einer Liste beinhaltet. Da bei einer dynamischen Liste mit dem Index `0` angefangen wird, ist der höchste Index `Count-1`.

Die zugehörige Klasse der Stufen von Kettenkomponenten ist `TLevel`, die diese wiederum als eine dynamische Liste im Attribut `items` enthält. `TLevel` enthält dabei meist nur eine Kettenkomponente, nur im Fall von sogenannten "stacked boxes", also Kettenkoponenten, die sich auf einer Stufe befinden, sind mehrere Elemente in der Liste enthalten.

Die Klasse `TChainComponent` modelliert die Kettenkomponenten. Ihr Name ist im Attribut `name` festgehalten. Zusätzlich hält die Variable `compFinished` fest, ob die Assoziationsstruktur der Variablen innerhalb der Kettenkomponente berechnet ist. Im Attribut `sameLevelBoxes` sind Verweise auf eventuelle andere Kettenkomponenten festgehalten, die sich auf derselben Stufe befinden. Der zugehörige Typ ist wiederum eine dynamische Liste von Referenzen auf Kettenkomponenten. Das wichtigste Attribut ist auch hier `items`, mit dem die in einer Kettenkomponente enthaltenen Variablen identifiziert werden. Der Typ der Variablen ist `TGraphVar`, der schon in Abschnitt 5.2.2.1 vorgestellt wurde. Zu bemerken ist, daß für Assoziationen im graphischen Modell keine Klassen gebildet sind. Dies liegt darin begründet, daß schon die Variablen `TGraphValue` die dafür nötige Information als Attribute speichern (s. Abschnitt 5.2.2.1).

Abbildung 5.11 veranschaulicht den Zugriff auf alle Modellelemente anhand eines Programmausschnitts. Ein solcher Zugriff findet z.B. statt, wenn die Elemente des Modells in einen Graphen umgewandelt werden (s. Abschnitt 6.1.3).

5.2.3 Paket Graphische Elemente

Das Paket graphische Elemente befaßt sich mit der Darstellung der im Programm vorkommenden Graphen. Das sind die Abhängigkeitskette, der Graph des zugehörigen

```
procedure TGraphicalModel.pass;
var
  i,j,k:  Integer;
  aktLevel :  TLevel;
  aktComponent :  TChainComponent;
begin
  if Count = 0 then // kein Graph vorhanden
    exit;
  // Durchlaufen der Levels -- bei primären Einflußgrößen
  // anfangen
  for i :=0 to Count -1 do
  begin
    // Durchlauf der Kettenkomponenten eines Levels
    aktLevel := Items[i];
    for j:= 0 to aktLevel.Count -1 do
    begin
      // Durchlauf der Variablen einer Kettenkomponente
      aktComponent := aktLevel.Items[j];
      for k:=0 to aktComponent.Count -1 do
        ;
    end;
 end;
end;
```

Abbildung 5.11: Durchlauf durch ein graphisches Modell

graphischen Modells und bei entsprechender Wahl die aus den verschiedenen Regressionsschritten resultierenden Graphen bei der Modellanpassung. Da eine genaue Beschreibung der graphischen Elemente des Programms in Kapitel 6.1 erfolgt, wird hier nicht näher darauf eingegangen.

5.2.4 Paket Mathematische Funktionen

Das Paket der mathematischen Funktionen besteht bislang aus zwei Quelldateien, nämlich `Strategy` und `Regressions`. Hier sind noch zusätzliche Dateien denkbar, da die mathematischen Funktionen im vorliegenden Entwurf nur grob modelliert worden sind. So enthält `Regressions` die Hierarchie aller möglichen Regressionen, die das Programm bieten soll. Es ist dabei gedacht, daß es eine abstrakte Oberklasse `TRegression` gibt, von der die einzelnen tatsächlich vorhandenen Regressionen, wie etwa die CG–Regression abgeleitet werden. Die Grobstruktur einer Suchstrategie, nämlich der, die in Kapitel 3 vorgestellt wurde, ist hingegen in der Datei `Strategy` enthalten und wird im folgenden beschrieben. Hierbei ist zu beachten, daß dies nur eine denkbare Strategie zur Erstellung eines graphischen Modells ist. Es ist also bei der zukünftigen Implementierung des Programms wichtig, ebenfalls eine Klassenhierarchie für die verschiedenen Arten von Suchstrategien zu bilden.

Abbildung 5.12 gibt einen Überblick über die Elemente der Klasse `TSelectionStrategy`, die die Selektionsstrategie modelliert.

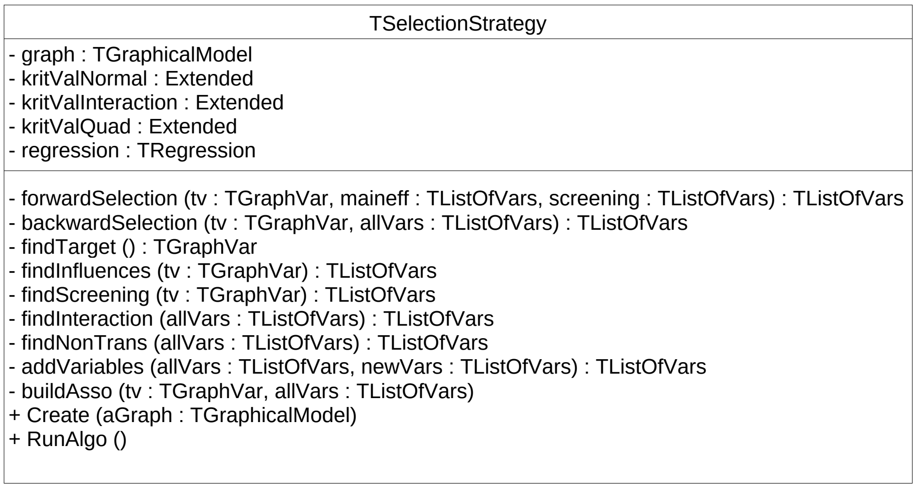


Abbildung 5.12: TSelectionStrategy

Als Attribute besitzt die Klasse das zu berechnende graphische Modell `graph`, die kritischen Werte der Selektionsstrategie für jeden Typ von Effekten `kritValNormal, kritValInteraction` und `kritValQuad` und die Art der durchzuführenden Regression `regression`.
Die meisten Operationen beziehen sich auf einzelne Teilschritte der Selektionsstrategie. Die Operation `forwardSelection` führt eine Vorwärtsselektion durch. Die Eingabeparameter sind die Zielvariable, die Haupteffekte und die im Screening auffälligen Effekte.

Das Resultat sind alle Einflußgrößen, die nach der Vorwärtsselektion im Modell sind. Sie werden in `TListOfVars` gespeichert (Information zu `TListOfVars` siehe Abschnitt 5.2.2.1). Mittels der Operation `backwardSelection` wird die Rückwärtsselektion im Programm modelliert. Als Eingabeparameter erhält diese Operation die Zielvariable und alle noch im Modell vorhandenen Einflußgrößen. Das Resultat sind wiederum die Einflußgrößen, die nach der Operation noch im Modell vorhanden sind. Mittels der Operation `findTarget` wird die neue Zielgröße bestimmt. Die Operationen `findInfluences` und `findScreening` geben zu der Zielvariablen, die als Eingabeparamter angegeben wird, die korrespondierenden Einflußvariablen bzw. die im Screening dazu auffälligen Variablen aus. Mittels der Operationen `findInteraction` und `findNonTrans` werden zu einer gegebenen Variablenliste alle paarweisen Interaktionen bzw. alle quadrierten Variablen gefunden. Die Kriterien, mittels derer Interaktionen bzw. quadrierte Effekte gebildet werden, sind in Schritt 4 und Schritt 5 der Selektionsstrategie zu finden (vgl. Abbildung 3.1). Durch die Operation `addVariables` werden neu gefundene Einflußgrößen in die Liste der bereits vorhandenen hinzugefügt. Die Operation `buildAsso` speichert die gefundenen Assoziationsstrukturen im graphischen Modell ab. Die bisher beschriebenen Operationen besitzen alle die Sichtbarkeitsstufe *private*, da sie von der Operation `runAlgo` aufgerufen werden, die die in Kapitel 3 beschriebene Selektionsstrategie umsetzt. Abbildung 5.13 zeigt die Zusammenarbeit der verschiedenen Operationen zur Selektionsstrategie und ihre Analogie zum abstrakten Selektionsalgorithmus anhand der Implementation der Operation `runAlgo`. Zudem sind auch Hinweise im Algorithmus enthalten, an welchen Punkten der Ablauf unterbrochen werden kann, um den bis dahin berechneten Graphen darzustellen. Das sind im wesentlichen alle Punkte, in denen ein Schritt im abstrakten Algorithmus abgearbeitet wurde. Schritt 2 bildet eine Ausnahme, da bei der Vorwärtsselektion nur Interaktionen und nichtlineare Transformationen hinzukommen, die im Graph nicht zu sehen sind.

5.2.5 Paket Formulare

Für jedes Fenster im Programm, das in Delphi allgemeiner als Formular bezeichnet wird, legt Delphi automatisch eine neue Quelldatei an. Die meisten Formulare des Programms sind Dialoge. Diese sind nur zur Ausführung einer Aktion sichtbar und verschwinden wieder, sobald der Dialog mittels `OK` oder `Abbruch` geschlossen wird. Um den Graph an verschiedenen Stellen der Berechnung gleichzeitig darstellen zu können,

```
Je nach Art des Graphen andere Schleifenkonstruktion.
Beim Konzentrationsgraphen werden alle Variablen einmal durchlaufen.
Beim Kovarianzgraphen werden alle Variablen zweimal durchlaufen.
begin
  newTarget := findTarget;
  newInfluences := findInfluences(newTarget);
  // Nachsehen, ob Variablen im Screening auffällig waren
  moreVariables:= findScreening(newTarget);
  if moreVariables<> Nil then // wenn ja, dann Vorwärtsselektion
  begin // Schritt 2 des Algorithmus -- Vorwärtsselektion
   newInfluences := forwardSelection(newTarget, newInfluences,moreVariables);
  end;
  // Schritt 3 des Algorithmus :  Rückwärtsselektion
  newInfluences := backwardSelection(newTarget,newInfluences);
  Graphausgabe möglich
  moreVariables := findInteraction(newInfluences); //suche Interaktionen
  if moreVariables<> Nil then // wenn Interaktionen vorhanden ...
  begin // ...führe Schritt 4 des Algorithmus aus
   newInfluences := addVariables(newInfluences,moreVariables);
   newInfluences := backwardSelection(newTarget,newInfluences);
  end;
  Graphausgabe möglich
  // suche Nichtlineare Transformation
  moreVariables := findNonTrans(newInfluences);
  if moreVariables<> Nil then // wenn Transformationen gefunden ...
  begin //...  führe Schritt 5 des Algorithmus aus
   newInfluences := addVariables(newInfluences,moreVariables);
   newInfluences := backwardSelection(newTarget,newInfluences);
  end;
  buildAsso(newTarget, newInfluences);// speichere gefundene Assoziationen
  Graphausgabe möglich
end;
```

Abbildung 5.13: Struktur des Algorithmus zur Selektionsstrategie sind die Formulare zur Graphdarstellung als sogenannte MDI–Kindfenster modelliert. MDI steht dabei für "multiple document interface". Die MDI–Schnittstelle wird immer dann genutzt, wenn mehrere Dokumente ähnlicher Struktur gleichzeitig angezeigt werden sollen. Im Falle des Programms ist ein Dokument ein graphisches Modell zu einem bestimmten Zeitpunkt der Berechnung. Die MDI–Schnittstelle vereinheitlicht die Handhabung mehrerer Dokumente. Kurz zusammengefaßt bietet MDI folgende Vorteile. Die MDI–Kindfenster befinden sich immer innerhalb des Hauptfensters und können dort beliebig angeordnet werden, was eine sehr gute Übersicht über alle Dokumente verschafft. Die Menüs und Schaltflächen, die ein MDI–Kindfenster besitzt, werden einfach in die Menü– bzw. Schaltflächenleiste des Hauptfensters eingeblendet. So wird im Programm der Menüpunkt `Datei` z.B. um den Menüpunkt `Drucke Graph` erweitert, wenn ein graphisches Modell sichtbar ist. Eine genaue Beschreibung des MDI–Konzepts und dessen Umsetzung in Delphi findet man in Warken (1997, S. 581ff).

5.2.6 Paket Hilfsklassen

Im Paket *Hilfsklassen* sind bis jetzt zwei Dateien vorhanden. Die eine beschäftigt sich mit der Übersetzung der Struktur eines graphischen Modells in einen Graphen und wird daher in Kapitel 6.1.3 genauer diskutiert. Die andere Datei, die das Einlesen der Daten aus einer Rohdatei zur Aufgabe hat, wird im folgenden näher beschrieben. Das Einlesen übernimmt die Klasse `TReadData` zusammen mit ihren Unterklassen. Einen Überblick über die Klassenhierarchie gibt Abbildung 5.14.

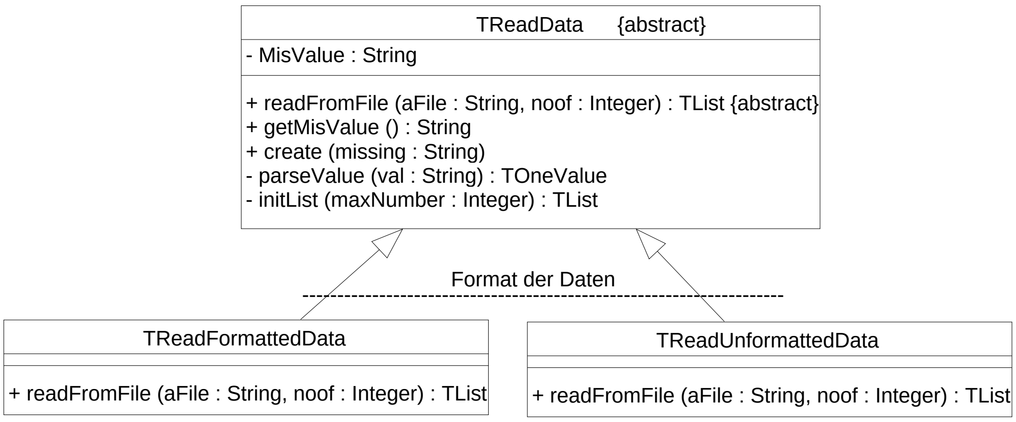


Abbildung 5.14: Hierarchie der Klassen, die Daten einlesen

Ziel der Modellierung der Hierarchie ist es, alle Gemeinsamkeiten von `TReadUnformattedData` und `TReadFormattedData` schon in der Oberklasse `TReadData` zu speichern. Sie besitzt als einziges Attribut das Zeichen für fehlende Werte `MisValue`, das in einem `String` gespeichert wird. Mit Hilfe der Operation `getMisValue` wird dieses Zeichen ausgelesen und im Konstruktor `Create` gesetzt. Die Operation `parseValue` hat die Aufgabe, einen aus der Textdatei gelesenen String in ein Objekt von `TOneValue`, also in eine Ausprägung umzuwandeln. Die Operation `initList` initiiert die dynamische Liste, die alle Variablen abspeichert.
Da sich die beiden Unterklassen nur in der Art des Einlesens unterscheiden, enthalten sie nur die dazu nötige Operation, nämlich `ReadFromFile`. Da diese Operation aber alle möglichen Unterklassen von `TReadData` besitzen müssen, wird sie abstrakt in der Oberklasse deklariert.

Im folgenden soll der Ablauf des Einlesens von Daten anhand eines abstrakten Algorithmus im Falle eines unformatierten Datensatzes erläutert werden. Insgesamt werden alle Variablen in einer dynamischen Liste gespeichert, wobei jede Variable selbst wiederum eine dynamische Liste besitzt, die alle Ausprägungen der Variablen speichert. Da jede Zeile in der Datei einer Ausprägung jeder Variablen entspricht, werden zeilenweise immer mehr Ausprägungen an die Liste einer jeden Variable angehängt. Diese Tatsache ist in Abbildung 5.15 schematisch dargestellt.
Abbildung 5.16 verdeutlicht die auszuführenden Schritte beim Einlesen einer Quelldatei, wobei die nötigen Anweisungen zum Abfangen von Fehlern, z.B. bei einer fehlgeschlagenen Dateiverbindung, außer acht gelassen werden.

5.3 Offene Fragen

Beim Entwurf des Programms sind einige Aspekte noch vernachlässigt worden. Im folgenden werden diese Aspekte und Vorschläge zu ihrer Bearbeitung aufgeführt.
Der wichtigste Punkt ist die Implementierung der nötigen Regressionen zur Berechnung des Graphen. Da diese zum Teil schon in anderen Programmen vorhanden sind, wäre es sinnvoll, sie über eine geeignete Schnittstelle einzulesen. Unter den Microsoft–Windows–Systemen wäre diese Schnittstelle eine sogenannte "dynamic link library", kurz DLL. Das Problem bei der Benutzung von DLLs liegt darin, daß diese nur Schnittstellen zu Funktionen und Variablen, aber keine zu Klassen bieten. Erweist sich diese Schnittstelle als zu umständlich, müssen die Regressionen „von Hand" implementiert werden.

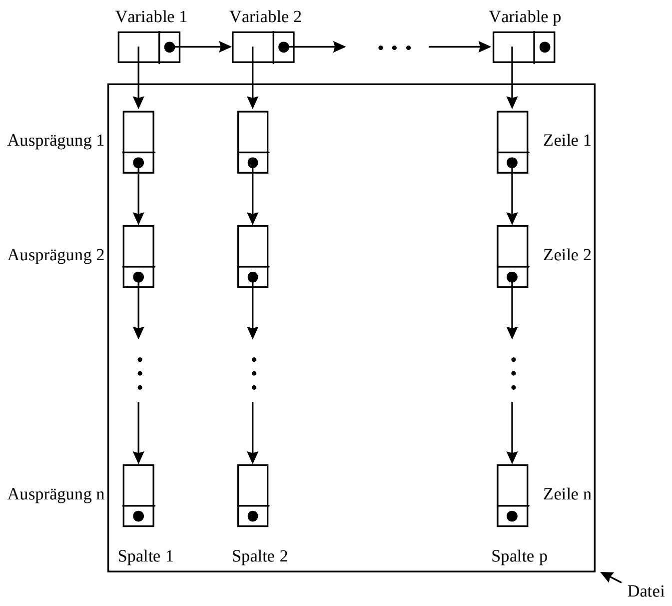


Abbildung 5.15: Zusammenhang Datei und Variablenabspeicherung

Ebenso muß das Ergebnisprotokoll in das Programm eingebaut werden. Da dieses eine Fülle von Informationen liefern kann, ist es hier wichtig, den Anwender in einem Dialogfeld auswählen zu lassen, welche Informationen er im Protokoll sehen möchte. Hierbei muß auch protokolliert werden, wenn die Berechnung aufgrund von Eingaben des Benutzers neu gestartet wird. Zudem soll es auch möglich sein, das Protokoll auszudrucken und zu speichern.
Eine weitere offene Frage ist das Abspeichern der im Programm schon ausgeführten Schritte mittels einer Sitzungsdatei, die dann bei einem erneuten Programmstart eingelesen werden kann. Ein Vorschlag ist, alle Objekte von Klassen in einer Binärdatei zu speichern, die dann einfach wieder in die Klassen eingelesen werden. Eine genauere Erläuterung einer solchen Verfahrensweise ist in Warken (1997, S. 389ff) zu finden.
Zudem ist noch offen, wie der Algorithmus der Selektionsstrategie unterbrochen werden kann, um den Graphen auszugeben. Denkbar wäre hier, daß man dem Algorithmus einem Prozeß zuweist, der angehalten wird, sobald der Graph dargestellt werden soll. Der Prozeß wird erst dann fortgesetzt, wenn der Benutzer mit der Berechnung des Graphen fortfahren möchte. Eine andere Lösungsmöglichkeit ist, daß man die Prozedur zur Berechnung des Algorithmus in kleinere Prozeduren aufteilt, an deren Ende jeweils der Graph nach Bedarf dargestellt werden kann. Bei dieser Möglichkeit müßten aber

Eingabe: `Name der Datei AName, Anzahl der beobachteten Variablen Noof`

Ausgabe: `Liste der Variablen mit ihren Ausprägungen`

Ablauf:
```
Lege alle Variablen an;
Stelle Verbindung zu Datei her und gehe an Dateianfang;
while nicht am Dateiende do
begin
  s:= nächste Zeile der Datei;
  i := 0; // Index der Variablenliste wird auf 0 gesetzt
  while s noch nicht leer ist do
  begin
   Entferne von links her solange Leerzeichen aus s, bis ein anderes
   Zeichen kommt;
   Solange nun kein Leerzeichen kommt, füge dieses Zeichen dem String
   aValue hinzu;
   Lösche die gelesenen Zeichen aus s;
   Wandle aValue mittels ParseValue in eine Ausprägung TOneValue um;
   Füge die so gewonnene Ausprägung der Liste der Variablen i hinzu;
   Gehe zur nächsten Variablen in der Liste, also i := i + 1;
  end;
end;
```

Abbildung 5.16: Einlesen von Daten

noch Informationen gespeichert werden, an welcher Stelle des Algorithmus man sich gerade befindet, damit die Berechnung richtig fortgesetzt wird.
Schließlich ist das Variablen–Screening noch nicht implementiert. Da hier ein zu einem interaktiven Streudiagramm äquivalentes Prinzip benutzt wird, könnte man das in Java implementierte Programm VisBenchmark nach Delphi portieren. Dazu müßte dieses Programm, das sich mit der Benutzung von interaktiven Streudiagrammen befaßt, geeignet an die Microsoft–Windows–Oberfläche angepaßt werden.

Kapitel 6

Modellierung des Graphen und dessen graphischer Darstellung

Die Visualisierung und interaktive Bedienung eines graphischen Modells sind die wichtigsten Komponenten des Programms, da gerade in diesem Aspekt die Vorteile einer Modellanpassung mittels eines Computers zum Tragen kommen. Um diesen Vorteil auch umfassend nützen zu können, müssen dem Benutzer Möglichkeiten zur Verfügung gestellt werden, die eine Modellanpassung „von Hand“ erweitern. Außerdem muß der Umgang mit den Graphen einfach gestaltet sein, um den Benutzer nicht unnötig zu belasten. Ebenso wichtig ist es, das Design der graphischen Elemente gründlich zu durchdenken, damit diese möglichst vielen Problemstellungen gewachsen sind. Das entspricht nicht nur dem objektorientierten Paradigma, sondern ist auch im besonderen innerhalb des Programms wichtig, da es verschiedene Arten von Graphen gibt, die durch verschiedene Methoden erzeugt werden.

Dieses Kapitel stellt die Konzeption der graphischen Elemente vor und erläutert Ideen zur Benutzerführung. In Abschnitt 6.1 wird die Modellierung der graphischen Elemente beschrieben und ein Einblick in die schon vorhandene Implementierung gegeben. Die Benutzerführung im Umgang mit graphischen Elementen wird in Abschnitt 6.2 erläutert. Die Diskussion der Benutzerführung beschränkt sich in diesem Abschnitt auf die Umgangweise mit den Graphen zu den verschiedenen Zeitpunkte der Modellanpassung, da der Umgang mit der Abhängigkeitskette ausführlich in Abschnitt 7.3 beschrieben wird.
Abschnitt 6.3 beschreibt noch offene Fragen im Zusammenhang mit der Visualisierung

der Graphen.

6.1 Modellierung der graphischen Elemente

Im Programm gibt es zwei Arten von Graphen. Bei einem der beiden Typen wird die Abhängigkeitskette dargestellt, während der andere das zugrundeliegende statistische Modell widerspiegelt. Letzterer spaltet sich wiederum in die Arten Konzentrationsgraph bzw. Kovarianzgraph auf. Sinn der Modellierung ist es nun, die Gemeinsamkeiten dieser Arten von Graphen herauszufinden und darauf aufbauend eine Klassenhierarchie zu erstellen. Zudem sollen die Elemente der graphischen Darstellung nicht mit dem zugrundeliegenden Model verzahnt sein, um so eine möglichst breite Zahl von Anpassungsverfahren in das Programm integrieren zu können.

Im folgenden wird zuerst auf grundlegende Elemente eingegangen, die bei jeder Art von Graphen auftreten. Anschließend wird die Modellierung der Abhängigkeitskette erläutert, deren Implementierung zudem anhand von Beispielen vorgestellt wird. In Abschnitt 6.1.3 werden die Elemente zur Visualisierung des graphischen Modells erklärt.

6.1.1 Grundlegende Klassen

Die Oberklasse zu allen Klassen, die graphische Elemente wie Pfeile, Knoten und Boxen modellieren, ist die Klasse `TElementOfGraph`. Sie kapselt alle Gemeinsamkeiten ihrer Unterklassen, die im folgenden vorgestellt werden. Ihr Aufbau ist in Abbildung 6.1 dargestellt.
Jedes graphische Element gehört zu einem bestimmten Graphen, der im Attribut `graph` gespeichert ist. Der Name des Elementes ist in `name` festgelegt. Die Position eines Elementes mittels virtueller Koordinaten ist in den Attributen `left, right, top` und `bottom` enthalten. Näheres zum Unterschied zwischen dem Konzept des virtuellen Bildschirms und der physikalischen Abbildungsfläche wird in Abschnitt 6.1.2 beschrieben. Das Attribut `marked` vermerkt, ob das Element vom Benutzer markiert wurde, und `hidden`, ob das Element gerade versteckt ist. Im Attribut `visualizationOf` wird festgehalten, welches Element des graphischen Modells vom vorliegendenden Element visualisiert wird. Die Attribute `Pen, Brush` und `Font` halten die Art des Stiftes, der vorhandenen Füllung und der zum Element gehörenden Schriftart fest.

Abbildung 6.1: Klasse TElementOfGraph

Die Operationen `getHMiddle` und `getVMiddle` liefern den horizontalen bzw. den vertikalen Mittelpunkt des Elements. Mittels der Operation `elementChanged` wird festgestellt, ob das Element verändert wurde. Wenn ja, wird danach ein erneutes Zeichnen des Elements erzwungen. Ein graphisches Element wird durch die Operation `create` erzeugt, der der zugehörige Graph und das Element umgebende Rechteck übergeben werden müssen. Hier ist zu bemerken, daß jedes graphische Element, sei es nun ein Kreis oder eine Linie, von Delphi durch ein unsichtbares umschließendes Rechteck bestimmt ist. Die Operationen `setMarked` und `isMarked` bzw. `setHidden` und `isHidden` setzen und erfragen die Zustände der Attribute `marked` bzw. `hidden`. Mittels der Operationen `setName` und `getName` wird der Name des Elements festgelegt und gelesen. Durch die Operation `getCoord` kann eine der Koordinaten des Elements erfragt werden. Die Operation `paint` zeichnet ein Element, und die Operation `paintMark` zeichnet einen Rahmen um das Element, wenn es markiert ist. Falls ein

Element andere graphische Elemente enthält, wie zum Beispiel eine Box Knoten enthält, so werden innerhalb der Methode `paint` die `paint`–Operationen der im Element enthaltenen graphischen Komponenten aufgerufen. Durch die Operation `invalidate` wird eine erneute Zeichnung eines Elementes erzwungen. Die Operation `invalidateRect` ist noch nicht implementiert. Sie bestimmt den Teil eines Objektes, der neu gezeichnet werden soll. Ob ein Punkt im vorliegenden Element liegt, wird mittels der Operation `pointInShape` überprüft. Zusätzlich wird auch noch die Art des graphischen Elementes, innerhalb dessen der Punkt liegt, mit ausgegeben. Die Operation `move` verschiebt ein Element um die in den Parametern angegebenen Werte. Die Operation `destroy` beendet den Lebenslauf eines Objekts dieser Klasse, d.h. es entfernt ein Onjekt dieser Klasse aus dem Speicher.

Eine andere wichtige Klasse ist die Klasse `TGraph`. Sie repräsentiert nicht selbst ein graphisches Element, sondern ist die Abstraktion eines Graphen. Außerdem stellt sie die Verbindung zu einer Zeichenfläche her. Abbildung 6.2 gibt ihren Aufbau wieder.

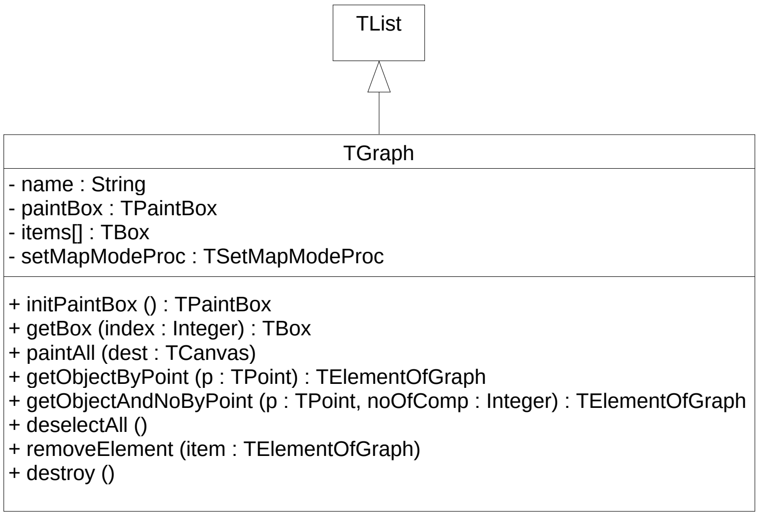


Abbildung 6.2: Klasse TGraph

Da `TGraph` zusammen mit ihren Unterklassen eine Menge von graphischen Elementen enthält, ist die Klasse von `TList` abgeleitet. Im Falle von `TGraph` sind die Elemente, die in `items` abgelegt werden, Boxen. Das Attribut `name` beinhaltet den Namen des Graphen. Im Attribut `paintBox` ist die Zeichenfläche, auf der der Graph zu sehen ist, festgelegt. Mittels des Attributs `setMapModeProc` wird ein Zeiger auf eine Methode erstellt, die die Verbindung des physikalischen zum virtuellen Bildschirm schafft. Ausführlicher wird doeser Aspekt in Abschnitt 6.1.2 behandelt.

Mittels der Operation `initPaintBox` wird die Zeichenfläche initialisiert. Die Operation `getBox` liefert eine bestimmte Box des Graphen. Die Elemente eines Graphen werden gezeichnet, indem die Operation `paintAll` die jeweiligen `paint`–Operationen der Elemente aufruft. Mittels der Operation `getObjectByPoint` wird ein Element des Graphen durch die Angabe eines Punktes bestimmt, der innerhalb dieses Elements liegt. Die Operation `getObjectAndNoByPoint` erfüllt dieselbe Aufgabe, liefert aber zusätzlich noch den Index des Elementes in der Liste. Durch `deselectAll` werden alle Markierungen im Graphen annuliert. Die Operation `removeElement` entfernt ein Element aus dem Graphen. Die Lebenszeit eines Objektes dieser Klasse wird mittels der Operation `destroy` beendet.

6.1.2 Vorstellung der Abhängigkeitskette

In diesem Abschnitt wird der Aufbau der Abhängigkeitskette erläutert. Dazu werden die für eine Abhängigkeitskette notwendigen graphischen Elemente beschrieben und zudem auf das Zusammenspiel mit dem Formular eingegangen, in dem die Abhängigkeitskette dargestellt ist. Bei Klassen, die auch innerhalb des Kettengraphen repräsentiert werden, werden nur die für den Aufbau der Abhängigkeitskette erforderlichen Elemente beschrieben.

6.1.2.1 Elemente der Abhängigkeitskette

Die elementaren Elemente einer Abhängigkeitskette sind die Kettenkomponenten und die Variablen, die in den Kettenkomponenten enthalten sind. Die dazu korrespondierenden graphischen Klassen sind `TBox` und `TNode`, die beide von `TElementOfGraph` abgeleitet sind. Im Unterschied zu einem Kettengraphen enthält hierbei eine Box keine Kanten und die Variablen werden nicht durch einen Kreis, sondern durch ihren Namen dargestellt. Ihr Aufbau ist in Abbildung 6.3 zu sehen.
Zusätzlich zu den von der Oberklasse geerbten Attributen und Operationen besitzt `TBox` folgende Elemente.

Im Attribut `nodes` werden die Knoten in einer Box mittels einer dynamischen Liste abgelegt. Die Operation `create` erzeugt ein neues Objekt dieser Klasse. Als Parameter muß der zugehörige Graph und die Position des Objektes angegeben werden. Obwohl diese Operation schon in der Oberklasse enthalten ist, wird sie in dieser Klasse

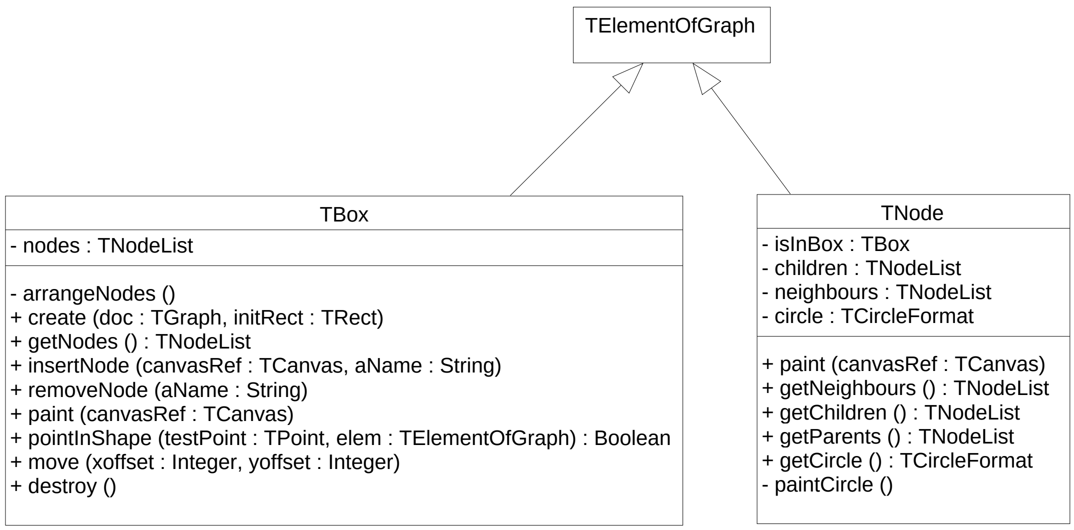


Abbildung 6.3: Klassen TBox und TNode

überschrieben, da zudem noch die Konstruktoren der in einer Box enthaltenen Knoten aufgerufen werden müssen. Die Operation `insertNode` fügt einen neuen Knoten in einer Box hinzu, und die Operation `removeNode` entfernt einen Knoten. Nach Ausführung dieser Operationen müssen die Knoten innerhalb der Box neu angeordnet werden, was mittels der Operation `arrangeNodes` geschieht. Diese richtet die Knoten zur Mitte der Box hin zentriert aus. Das Zusammenspiel dieser Operationen wird in Abschnitt 6.1.2.2 diskutiert. Die restlichen Operationen überschreiben wiederum die vorgegeben Operationen der Oberklasse. Der Unterschied liegt darin, daß die in der Box enthaltenen Knoten bei diesen Operationen mit berücksichtigt werden müssen. Da sie ansonsten analog zu den Operationen der Oberklasse sind, die in Abschnitt 6.1.1 erläutert sind, werden sie hier nicht näher diskutiert.

`TNode` erweitert `TElementOfGraph` um folgende Punkte. Innerhalb des Attributs `isInBox` wird die Box abgespeichert, in der der Knoten liegt. Die restlichen Attribute sind erst innerhalb der Darstellung eines Kettengraphen wichtig und werden deshalb mit den dazugehörigen Operationen in Abschnitt 6.15 beschrieben. Die Operation `paint` zeichnet im Falle einer Abhängigkeitskette den Namen der Variablen.

Die Beziehungen der Elemente einer Abhängigkeitskette sind in Abbildung 6.4 zusammengefaßt. Eine Abhängigkeitskette, also `TGraph`, besteht aus mehreren Elementen vom Typ `TBox`. Diese besitzen eine dymanische Liste von Knoten `TNodeList`, die wiederum eine beliebige Anzahl von Knoten `TNode` enthalten kann. Die Verwendung von `TNodeList` ist notwendig, da `TBox` im Gegensatz zu `TGraph` nicht von einer dynamischen Liste abgeleitet ist.

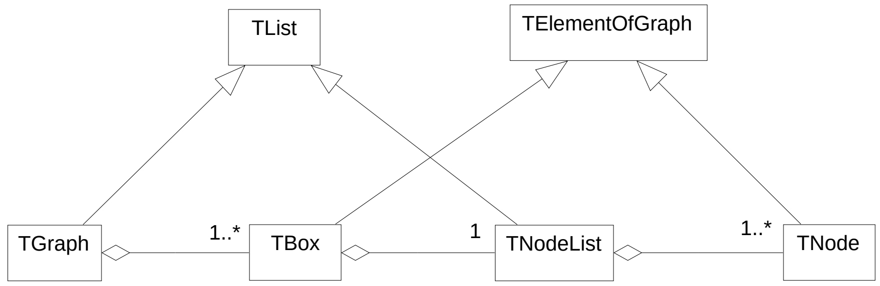


Abbildung 6.4: Struktur der Abhängigkeitskette

6.1.2.2 Zusammenspiel mit dem Formular

Die Abhängigkeitskette wird innerhalb eines Dialoges erstellt. Das zugehörige Formular wird von Delphi automatisch erzeut. Da hierbei immer eine Vielzahl von Attributen und Operationen angelegt wird, die für die Darstellung des Zusammenspiels vom Dialog und Graph nicht wichtig sind, werden hier nur interessante Elemente der Klasse `TDepWindow` beschrieben, die das Formular modelliert. Sie sind in Abbildung 6.5 zu sehen.

Im Attribut `objects` wird der Graph der Abhängigkeitskette abgespeichert und in `actObject` ein Element dieser Kette, das der Benutzer gerade angewählt hat. Die Attribute `lastLeftPos, lastTopPos, hoffset` und `voffset` sind für die Positionierung der Kettenkomponenten gedacht. Innerhalb des Attributs `numberOfVariables` wird die Anzahl der Variablen und innerhalb des Attributs `noOfComp` die Anzahl der Kettenkomponenten vermerkt. Das Attribut `lstVars` legt die Variablen zunächst in einer Liste ab. Von dort aus teilt der Benutzer diese dann in die jeweilige Komponente ein. Dieses Verfahren wird aus technischer Sicht weiter unten und in praktischer Hinsicht in Kapitel 7.3 diskutiert. Durch das Attribut `paintBox` wird die Zeichenfläche festgelegt, die wiederum mittels des Attributs `scrollBox` zu einer Zeichenfläche gemacht wird, in der ein Bewegen des sichtbaren Bildausschnittes, „Scrolling" genannt, möglich ist.

TDepWindow
- objects : TGraph - actObject : TElementOfGraph - lastLeftPos : Integer - lastTopPos : Integer - hoffset : Integer - voffset : Integer - numberOfVariables : Integer - noOfComp : Integer + scrollBox : TScrollBox + lstVars : TListBox + paintBox : TPaintBox
- MapMousePos (x : Integer, y : Integer) - setButtons () - unsetButtons () + formCreate (sender : TObject) + paintBoxPaint (sender : TObject) + paintBoxMouseDown (sender : TObject, button : TMouseButton, shift : TShiftState, x : Integer, y : Integer) + paintBoxMouseUp (sender : TObject, button : TMouseButton, shift : TShiftState, x : Integer, y : Integer) + lstVarsMouseDown (sender : TObject, button : TMouseButton, shift : TShiftState, x : Integer, y : Integer) + paintBoxDragOver (sender : TObject, source : TObject, x : Integer, y : Integer, state : TDragState, accept : Boolean) + paintBoxDragDrop (sender : TObject, source : TObject, x : Integer, y : Integer) + lstVarsDragOver (sender : TObject, source : TObject, x : Integer, y : Integer, state : TDragState, accept : Boolean) + lstVarsDragDrop (sender : TObject, source : TObject, x : Integer, y : Integer) + setMapMode (destCanvas : TCanvas, destControl : TControl) + makeInitialModel () : TGraphicalModel + btnAddClick (sender : TObject) + btnRemoveClick (sender : TObject) + edtNameExit (sender : TObject) + btnOkClick (sender : TObject)

Abbildung 6.5: Klasse TDepWindow – Formular der Abhängigkeitskette

Ein neuer Dialog wird mittels der Operation `formCreate` erzeugt. Das Ergebnis ist in Abbildung 6.6 zu sehen. Die Operationen `btnAddClick` und `btnRemoveClick` reagieren auf das Betätigen der Schaltflächen `Hinzufügen` und `Enfernen`. Hierbei wird eine neue Kettenkomponente auf der Zeichenfläche gezeichnet bzw. eine ausgewählte von der Zeichenfläche entfernt. Jeder Kettenkomponente kann im Editierfeld `Name` ein Name zugeordnet werden. Die Operation `edtNameExit` reagiert, sobald die Eingabe des Namens beendet wurde und ruft Operationen auf, die den angegebenen Namen in der Zeichenfläche darstellen. Je nachdem, ob eine Kettenkomponente gewählt ist oder nicht, sind andere Schaltflächen für den Benutzer zugänglich. Das Erlauben bzw. Verbieten von Betätigen von Schaltflächen ermöglichen die Operationen `setButtons` bzw. `unsetButtons`. Die Operation `paintBoxPaint` wird immer dann aufgerufen, wenn die Zeichenfläche neu gezeichnet werden muß. Ihre Aufgabe besteht hauptsächlich darin, die Zeichenprozedur von `objects` aufzurufen. Alle Operationen, in denen der Term `Mouse` im Namen enthalten ist, reagieren auf Mausereignisse. Hier werden Aktionen ausgeführt, die ein Element der Zeichenfläche markieren oder eine sogenannte "Drag–and–Drop"–Operation starten bzw. beenden. Näheres dazu und zum Prinzip des Drag–and–Drops wird weiter unten erläutert. Mittels der Operation `setMapMode` werden die Koordinaten des virtuellen Bildschirms bestimmt. Die Bedeutung und Realisation eines virtuellen Bildschirms wird ebenfalls weiter unten diskutiert. Die Operation `MapMousePos` nimmt eine Umrechnung der realen in die virtuellen Koordinaten vor. Dies ist wichtig, um herauszufinden, welchen Punkt des Graphen der Benutzer mit der Maus angewählt hat. Ist der Benutzer mit der Erstellung der Abhängigkeitskette fertig und betätigt die Schaltfläche `OK`, so wird die Operation `btnOkClick` aufgerufen. Ihre wesentliche Aufgabe besteht darin, mittels der Operation `makeInitialModel` aus der Zeichnung ein Objekt der Klasse `TGraphicalModel`, also ein graphisches Modell im Anfangszustand zu bilden.

Abbildung 6.6: Benutzersicht auf das Formular TDepWindow

Im folgenden werden einige Details der Implementierung erläutert, die das Zusammenspiel Dialog und Graph verdeutlichen. Zunächst muß beim Erstellen des Dialogs seine virtuelle Zeichenfläche mit der des Graphen verbunden werden. Abbildung 6.7 gibt einen Ausschnitt aus der Methode `formCreate` wieder, der dies regelt.

```
procedure TDepChain.FormCreate(Sender: TObject);
begin
  ...
  Objects:=TGraph.Create; // erzeuge Graph
  Objects.PaintBox:=PaintBox; // verbinde die Zeichenflächen
  Objects.SetMapModeProc:=SetMapMode; // verbinde die virtuellen Bildschirme
end;
```

Abbildung 6.7: Verbindung zwischen Formular und Graph herstellen

Bei zeichenintensiven Programmen ist es wichtig, Möglichkeiten zu besitzen, einen gegeben Bildschirmausschnitt zu vergrößern oder zu verkleinern. Dieses Prinzip wird oft als „Zooming" bezeichnet. Um dies zu ermöglichen, wird das Koordinatensystem des realen Bildschirms auf das eines virtuellen abgebildet. Im Programm erledigt dies die Operation `SetMapMode`, die immer vor einem Zeichenvorgang aufgerufen wird. Diese legt den Ursprung des virtuellen Bildschirms, die Achsenausrichtung und die Skalierung fest. Die Realisation ist dabei so ausgelegt, daß bei einer Größenänderung des sichtbaren Bildausschnittes nur zwei Konstanten geändert werden müssen, um das Bild um den gewünschten Faktor zu vergrößern bzw. zu verkleinern. Da die dabei aufzurufenden Funktionen von Delphi nicht unterstützt werden, ist dieses Vorgehen relativ mühsam programmtechnisch umzusetzen. So müssen innerhalb von `SetMapMode` mehrere Funktionen des "Application–Programmers–Interface" von Windows, kurz Windows–API genannt, aufgerufen werden. Ist dies aber erfolgt, so bringt diese Vorgehensweise noch einen anderen Vorteil. In der Windows–API sind auch Abbildungsmodus–Konstanten enthalten, die das reale Fenster auf bestimmte Längeneinheiten, wie z.B. Millimeter, abbilden. Dies kann beim Drucken des Graphen genutzt werden. Eine genaue Diskussion und Erläuterung dieser Vorgehensweise ist in Warken (1997, S. 562ff) zu finden.

Eine weitere wichtige Funktionalität des Dialogs betrifft das Hinzufügen einer neuen Kettenkomponente, was durch Betätigen der Schaltfläche `Hinzufügen` vom Benutzer erfolgt. Der abstrakte Ablauf der Methode `btnAddClick`, die durch dieses Ereignis aufgerufen wird, ist in Abbildung 6.8 zu sehen. Eine neue Komponente wird immer rechts an die schon vorhandenen Komponenten angefügt. Es können nicht mehr

Kettenkomponenten hinzugefügt werden, als Variablen vorhanden sind. Dabei ist zu beachten, daß eine neu hinzugefügte Komponente zunächst immer leer ist.

```
procedure TDepChain.btnAddClick(Sender:  TObject);
begin
  if Anzahl der Kettenkomponenten gleich Anzahl der Variablen then
    Warnung:  Anzahl der Kettenkomponenten wird größer als Anzahl der
    Variablen
  else
  begin
    Position der neuen Komponente berechnen und sie erzeugen;
    Komponente in die Liste objects hinzufügen;
    Positionierungsvariablen neu bestimmen;
  end;
  Zeichnen--Routine aufrufen;
end;
```

Abbildung 6.8: Hinzufügen einer Kettenkomponente

Beim Entfernen einer gewählten Kettenkomponente müssen mehrere Dinge berücksichtigt werden. Zum einen sind die in der Komponente vielleicht enthaltenen Variablen wieder in die Liste der noch nicht zugeordneten Variablen mit aufzunehmen. Zum anderen müssen alle Kettenkomponenten, die rechts von der zu entfernenden Komponente stehen, um die entsprechende Position nach links verschoben werden. Abbildung 6.9 zeigt den dafür nötigen Algorithmus.

Das Zusammenspiel zwischen dem Formular, der Abhängigkeitskette und den Elementen der Abhängigkeitskette ist besonders beim Verschieben von Variablen zu sehen. Im Programm wird diese Aufgabe durch eine sogenannte "Drag–and–Drop"–Operation realisiert. Das bedeutet, daß der Benutzer eine Variable anklickt, diese bei gedrückter Maustaste bewegt und sie dann in einem Element des Formulars durch das Loslassen der Maustaste ablegt. Im Programm kann eine Variable auf diese Weise in eine Kettenkomponente oder in die Liste der noch nicht zugewiesenen Variablen verschoben werden. Im folgenden werden die Erläuterungen zu diesem Vorgehen auf das Ablegen in einer Kettenkomponente beschränkt. Analoges gilt für den Fall des Verschiebens in

```
procedure TDepChain.btnRemoveClick(Sender:  TObject);
begin
  Listenindex der markierten Komponente erfragen;
  if Variablen in Komponente enthalten then
  begin
    Variablen in die Liste der noch nicht zugeordneten Variablen
    mitaufnehmen;
    OK--Schaltfläche unbenutzbar machen;
  end;
  Komponente aus Liste der Objekte löschen;
  Komponenten rechts von der entfernten nach links verschieben;
  Positionierungsvariablen neu bestimmen;
  Zeichnen--Routine aufrufen;
end;
```

Abbildung 6.9: Entfernen einer Kettenkomponente

die am linken Rand zu sehende Liste.
Abbildung 6.10 gibt die Zustände wieder, die die Zeichenfläche bei Mausoperationen einnehmen kann. In jedem Zustand wird eine dem Zustandsnamen entsprechende Operation der Klasse `TDepWindow` ausgelöst. So wird zum Beispiel im Zustand `OnDragDrop` die Operation `paintBoxDragDrop` ausgeführt.

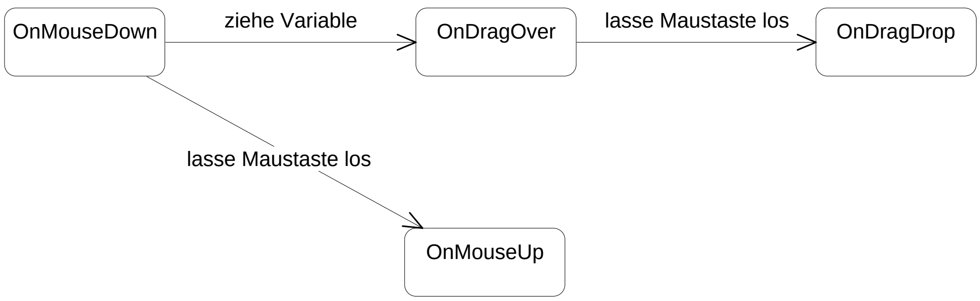


Abbildung 6.10: Zustandsdiagramm für die mausintensiven Operationen von TDep-Window

Die Operation zu `OnMouseDown` stellt zunächst fest, ob ein Objekt des Graphen, und wenn ja, welches Objekt der Benutzer mit der Maus angewählt hat. Ist es eine Variable, so geht `TDepWindow` in den Zustand `OnDragOver` über, ist es eine Kettenkomponente, so wird diese markiert und sofort der Zustand `OnMouseUp` eingenommen. In diesem Zustand wird die Markierung im Graphen eingezeichnet. Abbildung 6.11 veranschaulicht die Vorgehensweise bei der Bearbeitung des Ereignisses `OnMouseDown`.

```
Eingabe: Unter anderem: Koordinaten (x,y) der Mausposition

procedure TDepChain.PaintBoxMouseDown(...);
begin
  Alle Elemente des Graphen deselektieren;
  MapMousePos(x,y); // Koordinaten umrechnen
  Ausgewähltes Element anhand der Mausposition bestimmen;
  if ausgewähltes Objekt ist eine TBox then
  begin
    setze Attribut marked des ausgewählten Objekts auf True;
    SetButtons;
  end
  else if ausgewähltes Objekt ist TNode then
    starte Drag und Drop
  else // kein Element wurde ausgewählt
    UnsetButtons;
end;
```

Abbildung 6.11: Methode PaintBoxMouseDown

Im Zustand `DragOver` werden diejenigen Elemente des gesamten Formulars bestimmt, auf die die gewählten Variablen abgelegt werden können. Dies sind die Zeichenfläche und die Liste der noch nicht zugewiesenen Variablen. Dem Benutzer wird dies deutlich gemacht, indem bei allen anderen Elementen der Mauszeiger ein Symbol annimmt, das darauf hinweist, daß die Variable hier nicht abgelegt werden kann.
Der Zustand `OnDragDrop` wird erreicht, sobald der Benutzer die Maustaste über einem passenden Element losläßt. Wird die Maustaste bei einem anderen Element losgelassen, wird die Drag–and–Drop–Operation einfach ignoriert. In der zugehörigen Methode wird festgestellt, in welche Kettenkomponente der Benutzer die Variable gezogen hat. Danach wird die Variable in die passende Komponente eingefügt und aus dem Element

des Formulars, aus dem sie stammt, entfernt. Abbildung 6.12 zeigt den dafür nötigen Algorithmus.

```
Eingabe:  Unter anderem:  Koordinaten (x,y) der Mausposition

procedure TDepChain.PaintBoxDragDrop(...);
begin
  MapMousePos(x,y); // Koordinaten umrechnen
  Ziel der Operation bestimmen;
  if Ziel ist eine TBox then
  begin
    if Variable stammt aus Liste then
    begin
      Füge Variable in TBox ein;
      Lösche Variable aus Liste;
    end
    else if Variable stammt aus Kettenkomponente then
      Füge Variable in TBox ein;
      Lösche Variable aus anderer TBox;
    end;
    PaintBoxPaint(self);
  end;
end;
```

Abbildung 6.12: Methode PaintBoxDragDrop

Aus der obigen Beschreibung ist ersichtlich, daß bei einer Drag–and–Drop–Operation mehrere Klassen zusammenarbeiten müssen, um diese korrekt umzusetzen. Abbildung 6.13 verdeutlicht dieses Zusammenspiel anhand eines Sequenzdiagramms, das den Ablauf einer solchen Operation beschreibt, wenn eine Variable von einer Kettenkomponente in eine andere verschoben wird.

6.1.3 Vorstellung der Elemente zur graphischen Darstellung des graphischen Modells

Bei der Modellierung des zu einem graphischen Modell gehörenden Graphen müssen die in Abschnitt 6.1.2.1 beschriebenen Klassen um einige Elemente erweitert werden.

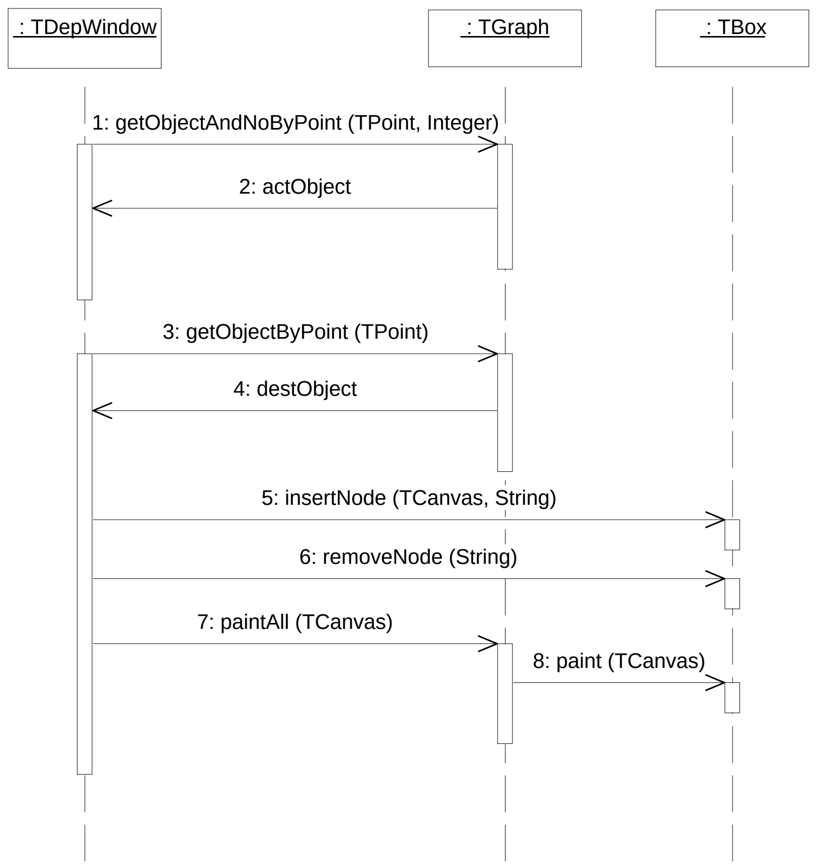


Abbildung 6.13: Sequenzdiagramm für eine Drag–and–Drop–Operation

Nachfolgend werden die wichtigsten Änderungen diskutiert.

In der Klasse `TNode` werden folgende Elemente erst bei der Darstellung des graphischen Modells benötigt. Das Attribut `circle` enthält die Art des Kreises, mit der die Variable dargestellt wird. Dieser ist bei diskreten Variablen ausgefüllt und bei stetigen nicht. In den Attributen `neighbours` und `children` werden die Knoten gespeichert, die Nachbarn oder Kinder von dem betrachteten Objekt sind. Auf das Vokabular von graphischen Modellen übertragen, sind dies genau die Variablen, die mit der gegebenen Variable beidseitig bzw. einseitig direkt assoziiert sind. Die zusätzlichen Operationen der Klasse sind dafür gedacht, die oben beschriebenen Attribute zu verarbeiten.

Alle anderen Klassen, die für eine graphische Darstellung des graphischen Modells gedacht sind, sind von schon beschriebenen Klassen abgeleitet. Sie sind in Abbildung 6.14 zusammen mit ihren Oberklassen abgebildet.

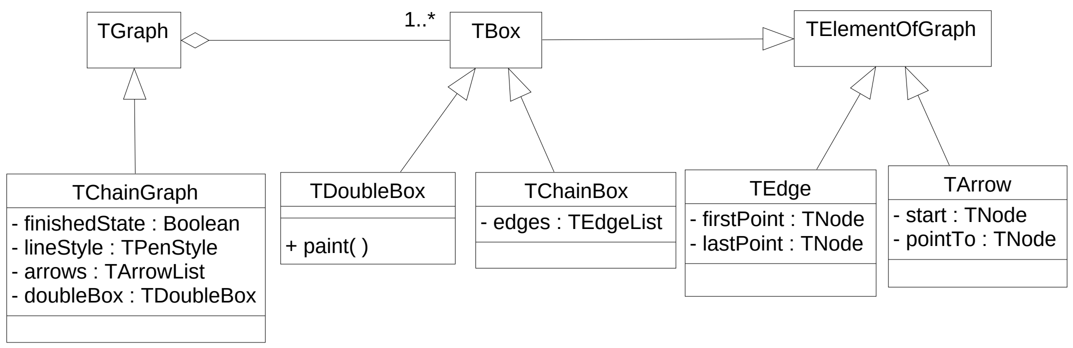


Abbildung 6.14: Klassen für die graphische Darstellung des graphischen Modells

Die Klasse `TChainGraph` ist wie ihre Oberklasse `TGraph` die Abstraktion eines Graphen. Im Attribut `lineStyle` wird vermerkt, ob die Linien im Graph gestrichelt oder als durchgezogene Linie dargestellt werden. Dies hängt von der Art des graphischen Modells ab. Bei einem Kovarianzgraph sind es gestrichelte Linien, beim Konzentrationsgraph durchgezogene. Zusätzlich zu einer Liste von Boxen muß bei einem Kettengraph auch eine Liste von Pfeilen gespeichert werden, die die einseitigen Assoziationen darstellen. Während der Modellgenerierung werden die Variablen, deren Assoziationsstruktur nicht von Interesse ist, in einer doppelt umrandeten Box dargestellt. Diese ist im Attribut `doubleBox` enthalten. Mittels des Attributs `finishedState` wird festgestellt, ob das graphische Modell schon fertig berechnet worden ist.
Die zugehörige Klasse von `doubleBox` ist `TDoubleBox`. Da sich diese von einer normalen `TBox` nur durch die Art der Darstellung unterscheidet, indem sie doppelt umrahmt ist, muß bei ihr als einzige Änderung die Operation `paint` überschrieben werden.
Die Boxen eines Kettengraphen besitzen im Unterschied zu normalen Boxen auch eine Menge von Kanten zwischen den Knoten. Dies wird in der Klasse `TChainBox` durch das Attribut `edges` festgelegt.
Die Klassen `TEdge` und `TArrow`, die die Linien und die Pfeile des Graphen modellieren, haben einen analogen Aufbau. Beide besitzen als Attribute die Endpunkte des Pfeils bzw. der Linie, die jeweils anders bezeichnet werden. Zudem ist die Art der Darstellung unterschiedlich.

Der Aufbau eines graphischen Modells mittels der vorgestellten Klassen ist in Abbildung 6.15 dargestellt. Ein Kettengraph `TChainGraph` besteht aus höchstens einer Liste von Pfeilen `TArrowList`, mehreren Kettenkomponenten `TChainBox` und höchstens einer doppelt umrandeten Box `TDoubleBox`. Eine Liste von Pfeilen besitzt mehrere Pfeile `TArrow`. Im Spezialfall eines ungerichteten Graphen, der nur aus einer Kettenkomponente besteht, besitzt `TChainGraph` keine Liste von Pfeilen. Ist das Modell fertig berechnet, so ist keine doppelt umrandete Box enthalten. `TChainBox` und `TDoubleBox` enthalten beide eine Liste von Knoten `TNodeList`, die wiederum mehrere Knoten `TNode` enthält. Zusätzlich besitzt `TChainBox` noch höchstens eine Liste von Kanten `TEdgeList`, die selbst wiederum Kanten `TEdge` enthält. Die Liste ist dann nicht vorhanden, wenn die Variablen einer Kettenkomponente nicht miteinander assoziiert sind.

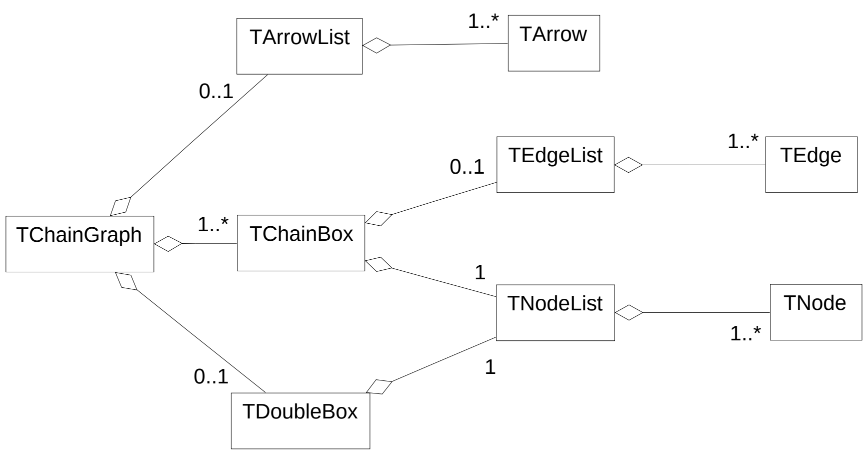


Abbildung 6.15: Struktur des Graphen

Die Übersetzung der Elemente des graphischen Modells in den Graphen und umgekehrt nimmt die Klasse `TTranslator` vor. Diese umfaßt Operationen, die entweder einzelne Elemente oder das ganze Konstrukt übersetzen. Die wichtigsten Zusammenhänge der Elemente sind in Tabelle 6.1 dargestellt.

Graphisches Modell	Graphische Darstellung
TGraphicalModel	TChainGraph
TGraphicalModel.kind	TChainGraph.lineStyle
TChainComponent	TChainBox
TGraphVar	TNode
TGraphVar.influenceOf	TArrowList
TGraphVar.directAssoc	TEdgeList
TGraphVar.scale	TNode.circle

Tabelle 6.1: Zusammenhang graphische Darstellung – graphisches Modell

6.2 Benutzerführung

Ein wichtiger Bestandteil bei der Modellanpassung im Programm ist die Möglichkeit, den Graphen in jedem Schritt darstellen zu können. Wie schon in Abschnitt 5.1.3 kurz angeschnitten, werden die einzelnen Darstellungen des Graphen während der Modellanpassung als MDI–Kindfenster realisiert. Die dabei auftretende erweiterte Menüstruktur wird in Abschnitt 6.2.1 vorgestellt. Zudem stellt sich die Frage, welche Möglichkeiten der Benutzer haben soll, die Darstellung des Graphen zu verändern und in die Berechnung einzugreifen. Diese grundsätzlich verschiedenen Eingriffsmöglichkeiten werden im Programm mittels zweier Modi realisiert, die in den Abschnitten 6.2.2 und 6.2.3 beschrieben werden. Im Abschnitt 6.2.4 wird die gewählte Realisierung der Benutzerführung diskutiert.

6.2.1 Menüstruktur

Das auch ohne Anzeige eines Graphen vorhandene Menü „Datei” wird um Punkte erweitert, mit denen der dargestellte Graph abgespeichert und gedruckt werden kann. Neu hinzugekommen sind die Menüs „Berechnung” und „Bearbeiten”. Das Menü „Berechnung” enthält einen Punkt, mit dem die Berechnung des Graphen fortgesetzt werden kann, und einen weiteren, der den zuletzt ausgeführten Berechnungsschritt rückgängig macht. Im Menü „Bearbeiten” sind diejenigen Punkte aufgeführt, mit denen in die Darstellung und automatische Berechnung des Graphen eingegriffen werden kann. Hier sind der „Informationsmodus” und der „Bearbeitungsmodus” enthalten, die beide in den folgenden Unterabschnitten ausführlich beschrieben werden. Zusätzlich gibt es in diesem Menü noch Punkte zum Vergrößern und Verkleinern des sichtbaren Bildausschnitts. Alle Menüpunkte des Menüs „Bearbeiten" sind auch in einer Schaltflächenleiste am oberen Fensterrand zu sehen. Ist ein Menüpunkt angewählt, so wird dies auch in der Schaltflächenleiste gekennzeichnet.

6.2.2 Informationsmodus

Im Informationsmodus kann der Benutzer die Darstellung des Graphen verändern, um zum einen eine übersichtlichere Darstellung des Graphen zu erhalten und zum anderen zusätzliche Informationen zum Modell zu bekommen. Dazu ist der Modus in mehrere Punkte unterteilt, die vom Benutzer im Menü oder in der Schaltflächenleiste ausgewählt werden können. Diese werden im folgenden näher erläutert.

Standard In diesem Menüpunkt bedeutet ein Anklicken von Kettenkomponenten oder Variablen ein Markieren dieser Elemente. Das kann dazu genutzt werden, mittels der entsprechenden Menüpunkte den sichtbaren Bereich des Bildes auf das gewählte Element hin zentriert zu vergrößern bzw. zu verkleinern. Außerdem kann der Name des markierten Elementes geändert werden. Ebenso ist es möglich, Kettenkomponenten und Variablen per gedrückter linker Maustaste zu verschieben. Hier werden die jeweiligen Linien und Pfeile mitverschoben, die mit den verschobenen Elementen direkt verbunden sind, allerdings mit der Einschränkung, daß die Elemente nur so weit verschoben werden dürfen, daß sie nicht das schon berechnete Modell verändern. Dies bedeutet, daß Variablen nur innerhalb einer Kettenkomponente verschoben werden dürfen, und Kettenkomponenten nur so weit, daß sie nicht die Position mit anderen Kettenkomponenten in der Abhängigkeitskette wechseln.

Verstecken Hier resultiert das Betätigen der Maustaste auf ein Element des Graphen im Verstecken desselben. Das wird im Graph symbolisch als Dreieck dargestellt. Werden hierbei Kettenkomponenten oder Variablen versteckt, so werden alle direkt damit verbundenen Pfeile und Linien ebenfalls nicht mehr angezeigt. Ein Klick auf ein verstecktes Element macht dieses wieder sichtbar.

Beschriften Wird in diesem Modus ein Element markiert, so kann der Benutzer zusätzliche Informationen zum gewählten Element angeben, die auch editiert werden können.

Farblich Hervorheben Ist dieser Menüpunkt gewählt, so erscheint ein Farbdialog, in dem eine Farbe ausgewählt werden kann. Wird in diesem Modus ein Element des Graphen angewählt, so wird dieses in der entsprechenden Farbe dargestellt.

Details Mittels dieses Menüpunktes werden zu einem angewählten Element Details in einem extra Fenster angegeben. Bei Variablen können dies zum Beispiel der minimal und maximal angenommene Wert und der Mittelwert sein, bei einer Linie oder einem Pfeil die Richtung des Zusammenhangs.

Hervorheben von Elementen Hiermit können verschiedene Zusammenhänge im Graphen farblich hervorgehoben werden, die zuvor in einem Dialog festgelegt werden müssen. Dabei kann es sich beispielsweise um interessierende Pfade oder auch bestimmte Unabhängigkeiten handeln. Ist die Art der Hervorhebung gewählt, so muß eine Variable angeklickt werden, auf die sich die Untersuchung beziehen soll.

6.2.3 Bearbeitungsmodus

In diesem Modus kann der Benutzer in die Berechnung des Modells eingreifen, indem er die Struktur des Graphen verändert. Er besitzt hier die Möglichkeit, Kanten hinzuzufügen oder zu löschen. In beiden Fällen wird bei einer darauffolgenden Weiterberechnung des Modells ein Hinweis gegeben, daß Teile der Modellanpassung nochmals auszuführen sind. Es ist zu bemerken, daß es hier nicht vorgesehen ist, Variablen in andere Kettenkomponenten zu verschieben, da dies zu große Änderungen nach sich ziehen würde. So müßte bereits das Daten–Screening neu durchgeführt werden. Will der Benutzer die Anordnung der Variablen ändern, so muß er die Abhängigkeitskette neu erstellen, und damit mit der Berechnung von vorne beginnen.

Im folgenden werden die Punkte des Bearbeitungsmodus genauer besprochen.

Kante hinzufügen Hiermit kann eine Kante zwischen zwei Variablen eingefügt werden, indem die beiden Variablen nacheinander angewählt werden. Diese Kante bleibt dann bei allen Berechnungen im Modell enthalten. Die so entstandene Kante wird im Graphen rot eingetragen, um sie von den anderen Kanten hervorzuheben.

Kante entfernen Mittels dieses Menüpunkts kann eine Kante aus der Berechnung entfernt werden. Dazu muß die zu entfernende Kante markiert werden. Die

Darstellung einer nicht vorhandenen Kante ist etwas schwierig. Um sie von normalen nichtvorhandenen Kanten zu unterscheiden, wird sie rot gepunktet in den Graphen eingetragen.

Originalzustand wiederherstellen Dieser Menüpunkt stellt den Zustand wieder her, der vorlag bevor Änderungen im Modell vorgenommen wurden.

6.2.4 Überlegungen zur Realisation der Benutzerführung

Die hier gewählte Realisation der Eingriffsmöglichkeiten in die graphische Darstellung des graphischen Modells bei der Modellgenerierung ist nur eine von vielen Möglichkeiten. So hätte man die verschiedenen Tasten der Maus mehr nutzen können, um einige der oben angesprochenen Punkte zu einem einzelnen zusammenzufassen. So könnte man zum Beispiel die Menüpunkte `Standard` und `Verbergen` so miteinander kombinieren, daß die linke Maustaste die Aktion zu `Standard` ausführt und die rechte die Aktion zu `Verbergen`. Außerdem könnte man auf die Aufteilung des den Bearbeitungsmodus in die Menüpunkte `Kante hinzufügen` und `Kante entfernen` verzichten, da die gewünschten Aktionen eindeutig durch die entsprechenden Mausoperationen festgelegt sind. So müssen bei `Kante Hinzufügen` Variablen und bei `Kante entfernen` vorhandene Kanten angeklickt werden. Beide Methoden hätten den Vorteil, daß bei verschiedenen Aktionen innerhalb der Bearbeitung des Graphen nicht immer ein anderer Punkt angewählt werden muß, so daß ein geübter Benutzer schneller zum Ergebnis kommt.

Die oben beschriebenen Methoden besitzen aber auch Nachteile, die durch die vorliegende Realisation behoben werden. Sie wurde aus folgenden Gründen gewählt. Durch die Anzeige des gewählten Menüpunktes in der Schaltflächenleiste wird dem Benutzer deutlich gemacht, welche Aktion er gerade ausführt. Das ist gerade im Bearbeitungsmodus wichtig, da man sich hier genau überlegen sollte, welche Veränderungen man vornimmt. Zudem trägt eine Mehrfachbelegung der Maus leicht zu einer Verwirrung des Benutzers bei, da sich dieser stets daran erinnern muß, welche Taste für welche Aktion steht. Ein weiterer Vorteil der gewählten Realisation liegt in der Erweiterbarkeit der Menüstruktur. So ist es hier einfach möglich, einen neuen Menüpunkt anzuhängen, falls eine neue Möglichkeit zur Bearbeitung des Graphen als sinnvoll betrachtet wird.

6.3 Offene Fragen

Die wichtigste noch zu erledigende Aufgabe besteht darin, die graphische Darstellung des graphischen Modells während der Modellanpassung zu implementieren. Die Grundstrukturen der dazu nötigen Klassen sind schon vorhanden; es fehlt noch die Implementierung der Methoden, die den Graphen visualisieren. Hier ist es sinnvoll, auf die schon bestehende Implementierung der Abhängigkeitskette aufzubauen, da die Visualisierung beider Graphen ähnlich ist. Die größte Schwierigkeit bei dieser Aufgabe liegt darin, eine möglichst übersichtliche Anordnung der graphischen Elemente zu gewährleisten. Dazu müssen zunächst Kriterien gefunden werden, die den Begriff „übersichtlich" beschreiben. So kann es bei dem Versuch, eine Darstellung mit möglichst wenigen Überschneidungen von Kanten zu finden, passieren, daß dadurch der Graph für den Betrachter nicht unbedingt an Übersichtlichkeit gewinnt. Diese Problematik wird im Rahmen des "Graph drawing"–Forschungsgebietes untersucht. Es ist also nötig, Algorithmen aus diesem Bereich im Programm zu implementieren.
Ebenso sind die in Abschnitt 6.2 beschriebenen Konzepte zur Bedienung des Graphen noch nicht implementiert. Hier kommt es teilweise zu einer Vernetzung mit dem Gebiet der Graphentheorie, da Konzepte wie benachbarte oder nichterreichbare Knoten umgesetzt werden müssen.

Da die dargestellten Graphen recht groß werden können, müssen Möglichkeiten zur Gewährleistung der Übersicht über den Graphen erwogen werden. Die Konzepte, die im Programm dafür schon vorhanden sind, sind das Bewegen und das Vergrößern bzw. Verkleinern des sichtbaren Bildausschnittes sowie die Möglichkeit, die Lage von Variablen und Kettenkomponenten zu ändern. Ein Vorschlag zur weiteren Verbesserung der Übersichtlichkeit wäre es, in einem kleinen Fenster den gesamten Graphen darzustellen. Innerhalb dieser Darstellung ist ein kleines Rechteck markiert, das den gerade aktuellen Bildausschnitt darstellt. Durch eine Verschiebung dieses Rechtecks mit der Maus wird auch der aktuell dargestellte Bildausschnitt verschoben.

Das Besondere bei der Visualisierung des Graphen eines graphischen Modells ist, daß nicht ein resultierender Graph am Ende der Modellgenerierung dargestellt werden muß, sondern Graphen zu verschiedenen Stufen der Modellgenerierung. Solche Probleme werden auch unter dem Begriff der diskreten Animation behandelt. Diskrete Animationen kommen beispielsweise auch bei der Darstellung von Indexstrukturen vor. Hier wird z.B. sukzessive ein Baum aufgebaut. Dabei ist es wünschenswert, den Aufbau

Schritt für Schritt verfolgen zu können. Anwendungen für diskrete Animationen sind in zahlreichen unterschiedlichen Gebieten zu finden. Aus diesem Grund scheint es sinnvoll zu sein, die Problematik auf eine möglichst hohe Abstraktionsebene zu bringen, um mehrere Gebiete damit behandeln zu können. Dies hätte zur Konsequenz, daß eine Programmiersprache realisiert werden müßte, mittels derer Graphanimationen leicht beschrieben werden können. Eine Übertragung auf das Programm wäre dann ein Anwendungsgebiet dieses Ansatzes. In der Forschung gibt es schon einige Ansätze, die sich mit der Erstellung von Programmiersprachen für die Visualisierung von Graphen beschäftigen. Die wichtigsten Vertreter sind die Systeme daVinci (Fröhlich und Werner 1994), Graphlet (Himsolt 1997) und ein System, das bei AT&T Bell Laboratories entwickelt wurde (North and Koutsofios 1998). Diesen drei Systemen ist gemeinsam, daß sie eine möglichst breite Palette von Graphen beschreiben wollen. Es gibt aber auch Systeme, die sich mit Graphen spezieller Art beschäftigen. Hier ist das System VCG zu nennen (Sander 1995). VCG steht für "Visualization of Compiler Graphs" und beschäftigt sich explizit mit der Visualisierung von Graphen, die beim Compiler–Bau erscheinen. Es ist hier also nötig, den Aufbau der im Gebiet der graphischen Modelle vorkommenden Graphen vernünftig zu definieren um herauszufinden, inwieweit die schon bestehenden Systeme auf die vorliegende Problematik angewandt werden können oder ob ein komplett neues System erstellt werden muß.

Kapitel 7

Vorbereitende Schritte zur Erstellung eines Graphen

Bis zur eigentlichen Erstellung eines graphischen Modells müssen vom Benutzer einige Schritte ausgeführt werden. Zunächst werden Daten eingelesen und mit einem Skalenniveau versehen. Danach erfolgt die Wahl der Modellvariablen und die Erstellung der Abhängigkeitskette. Als nächstes wird ein Variablen–Screening durchgeführt und zuletzt werden die Parameter zur Graph–Berechnung festgelegt. Bis auf das Variablen–Screening sind diese Schritte im derzeitigen Entwicklungsstadium des Programms realisiert. Da ein komplettes Benutzerhandbuch zur Zeit noch nicht sinnvoll ist, illustriert das vorliegende Kapitel anhand eines Beispiels die Abfolge dieser Schritte im Programm. Es handelt sich hierbei wieder um ein rein fiktives Beispiel, in dem die inhaltliche Bedeutung der betrachteten Variablen keine Rolle spielt. Um die aufgeführten Situationen zu verdeutlichen, wird bei jedem wichtigen Schritt ein Screenshot gezeigt.

Die Einteilung des Kapitels bildet die Struktur der auszuwählenden Menüpunkte im Programm nach, indem jeder Menüpunkt in einem Unterabschnitt behandelt wird. Zu beachten ist, daß ein Daten–Screening nicht durchgeführt wird, da dieser Aspekt des Programms noch nicht realisiert ist.

7.1 Einlesen der Daten

Die Rohdatei mit dem Namen `bsp6.dat` besteht aus sechs Variablen, die jeweils sieben Ausprägungen haben. Sie besitzt somit die in Tabelle 7.1 dargestellte Form.

1	7	1	4	10.7	67
2	5	-1	12	0.9	21
3	8	1	13	1.6	4.34
4	3	-1	15.7	34	.
2	5	.	0.6	234	21.07
1	.	-1	54.6	32.3	21.3
4	3	1	23.89	85.4	43.4

Tabelle 7.1: Rohdatei bsp6.dat

Wie den Werten zu entnehmen ist, handelt es sich bei den ersten drei Variablen um diskrete und bei den letzteren um stetige Variablen. Nach Auswahl des Menüpunktes `Datei - Daten einlesen` öffnet sich der in Abbildung 7.1 abgebildete Dialog, mit dessen Hilfe eine Datei auszuwählen ist, die die Rohdaten enthält. Da der Dialog einem Windows–Standard–Dialog entspricht, wird auf eine genaue Beschreibung der Funktionsweise des Dialogs verzichtet. In Falle des Beispiels wird die Datei `bsp6.dat` gewählt.

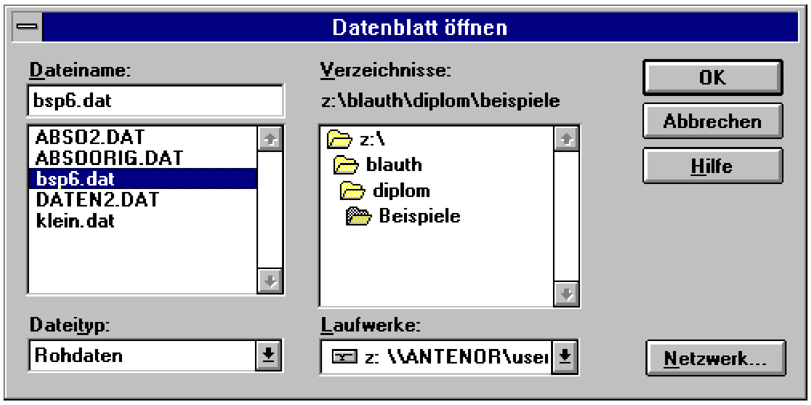


Abbildung 7.1: Öffnen eines Datenblatts

Als nächstes muß der genaue Aufbau der Datei mittels des in Abbildung 7.2 gezeigten Dialogs bestimmt werden. Im Optionsfeld `Dateiart` wird festgelgt, in welchem

Format die Daten abgespeichert sind (vgl. Abschnitt 5.1.2). Zudem wird die Anzahl der betrachteten Variablen in das Eingabefeld `Anzahl der beobachteten Variablen` eingetragen. Im Beispiel werden die entsprechenden Eingaben für formatfreies Einlesen und für `6` Variablen getätigt.

Abbildung 7.2: Bestimmung der Dateiart

Der folgende, in Abbildung 7.3 dargestellte Dialog beschäftigt sich mit dem Inhalt der Datei. Es werden die Namen der Variablen festgelegt und zudem das Zeichen für fehlende Werte eingetragen. Der erste Name steht dabei für die erste Spalte der Rohdaten, der zweite Name für die zweite, und so weiter. Da die Beispieldatei keine inhaltliche Bedeutung hat, sind die Variablen mit einzelnen Buchstaben belegt worden. Diese spiegeln das Skalenniveau der Variablen wider. Für diskrete Variablen wurden die Buchstaben `A, B, C` und für stetige Variablen die Buchstaben `X, Y, Z` gewählt. Als Zeichen für fehlende Werte wurde `'.'` angegeben, welches schon standardmäßig eingestellt ist.
Schließlich muß jeder Variable ein Skalenniveau zugewiesen werden, was mit Hilfe des in Abbildung 7.4 gezeigten Dialogs erfolgt. Dazu geht man folgendermaßen vor. Durch Anklicken wird eine Variable ausgewählt, für die das Skalenniveau festgelegt werden soll. Die einzelnen Punkte des Skalenniveaus sind in verschiedene Ebenen zusammengefaßt, die baumartig angeordnet sind, und zwar indem sie nach rechts hin das Skalenniveau der Variablen immer genauer beschreiben. So unterscheidet die erste Ebene zwischen diskreten und stetigen Variablen, die obere Hälfte der zweiten Ebene trennt binäre von multikategorialen diskreten Variablen und die dritte Ebene unterscheidet in den Teilbereichen jeweils zwischen nominalen und ordinalen Variablen. Um für eine

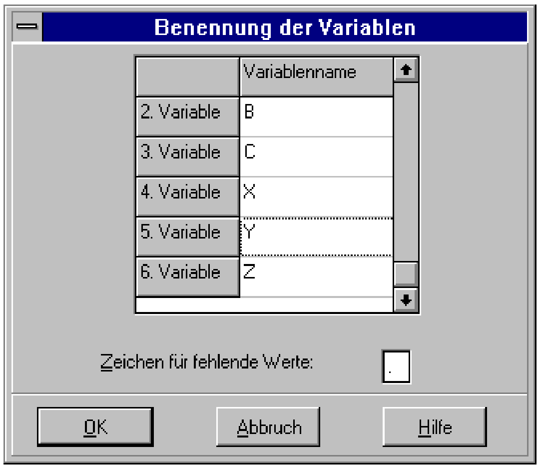


Abbildung 7.3: Benennung der Variablen

gewählte Variable das Skalenniveau exakt festzulegen, muß jeweils der am weitesten rechts stehende Punkt der Baumstruktur angeklickt werden. Ist dies der Fall, so wird durch ein Häkchen vor dem Variablennamen gekennzeichnet, daß die Variable fertig belegt wurde. Erst wenn alle Variablen ein Skalenniveau besitzen, kann die Schaltfläche `OK` betätigt werden, um den Dialog zu beenden. Die anderen Optionsfelder innerhalb der Belegung wurden deshalb in den Dialog mitaufgenommen, da man bei jedem Klick auf ein bestimmtes Niveau genauere Informationen dazu im linken Teil des Fensters erhält.
Tabelle 7.2 faßt die Skalennniveaus zusammen, die für das vorliegende Beispiel gewählt wurden.

Variable	**Skalenniveau**
A	diskret multinomial ordinal
B	diskret multinomial nominal
C	diskret binär nominal
X	stetig normalverteilt
Y	stetig normalverteilt
Z	stetig normalverteilt

Tabelle 7.2: Skalenniveaus der Beispielvariablen

Hiermit ist das Einlesen der Daten beendet. Es ist jetzt auch ein geeigneter Zeitpunkt,

Abbildung 7.4: Festlegung des Skalenniveaus

die Sitzung mittels des Menüpunkts `Datei - Sitzung speichern` abzuspeichern, um beim nächsten Mal nicht wieder alle Einstellungen vornehmen zu müssen.

7.2 Festlegung der Modellvariablen

Sind die Daten eingelesen, wird genauer bestimmt, welche Variablen der Rohdatendatei tatsächlich im Modell betrachtet werden sollen. Dazu wählt man den Menüpunkt `Analyse - Festlegung der Modellvariablen` an, der den in Abbildung 7.5 dargestellten Dialog erzeugt. Der Dialog wird folgendermaßen bedient. Durch Auswahl von ein oder mehreren Variablen und darauffolgendes Anklicken von `<` oder `>` werden Variablen aus der Reihe der Modellvariablen entfernt bzw. hinzugenommen. Will man derartiges für alle Variablen einer Liste ausführen, so genügt es, auf `<<` bzw. auf `>>` zu klicken, ohne vorher Variablen zu wählen.
Im Beispiel sollen alle Variablen der Rohdatei auch im Modell untersucht werden. Aus diesem Grund sind alle Variablen in der Liste `Modellvariablen` enthalten.

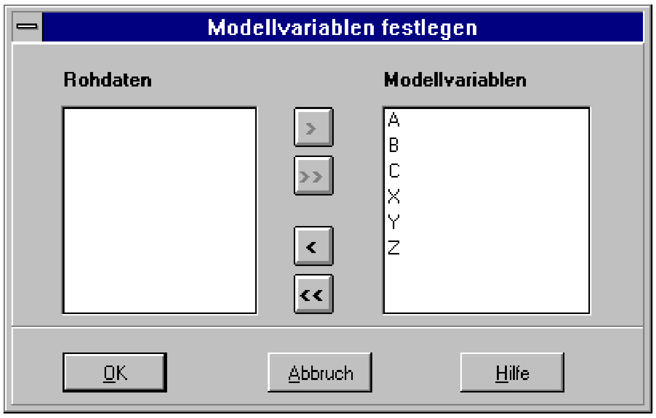


Abbildung 7.5: Festlegung der Modellvariablen

7.3 Erstellung der Abhängigkeitskette

Nach der Festlegung der Modellvariablen muß bei der Erstellung eines Kettengraphen die Abhängigkeitskette erzeugt werden. Im Programm wird dazu der Menüpunkt `Analyse - Abhängigkeitskette erstellen` aufgerufen. Die Bedienung des daraufhin erscheinenden Dialogs, der in Abbildung 7.6 dargestellt ist, ist folgendermaßen. Mittels der Schaltfläche `Einfügen` kann eine beliebige Anzahl von Kettenkomponenten erzeugt werden. Die zu betrachtenden Variablen, die anfangs in einer Liste zu finden sind, werden dann mittels "Drag–and–Drop" in die passende Komponente verschoben. Dies bedeutet, daß man eine Variable anklickt, sie mit gedrückter linker Maustaste in die passende Komponente bewegt und dort die Maustaste wieder losläßt. Durch das gleiche Prinzip können die Variablen auch innerhalb der verschiedenen Kettenkomponenten ausgetauscht werden. Durch ein Anklicken einer Kettenkomponente wird diese markiert, was mit einer roten Umrandung der Komponente angezeigt wird. Ist dies der Fall, so kann durch Anklicken der Schaltfläche `Entfernen` die Komponente aus der Abhängigkeitskette wieder entfernt werden. Sind Variablen in dieser Komponente enthalten, so werden sie in die Liste zurückgenommen. Außerdem werden die restlichen Komponenten neu angeordnet. Zudem kann eine Komponente im markierten Zustand beschriftet werden, indem einfach die Information im Editierfeld `Name` eingetragen wird. Es wird auch die Möglichkeit bestehen, eine markierte Kettenkomponente zu teilen, um so sogenannte "stacked boxes" zu bilden, was aber zum vorliegenden Zeitpunkt noch nicht realisiert ist. Sind alle Variablen auf die Kettenkomponenten verteilt, kann der

Dialog mittels der Schaltfläche `OK` geschlossen werden.

Abbildung 7.6: Erstellung der Abhängigkeitskette – Anfang

Im Beispiel wurden die Variablen auf drei Kettenkomponenten verteilt. Variable A ist dabei die einzige reine Zielgröße, die Variablen B und C stellen die einzige Stufe von Intermediaten dar und die Variablen X, Y und Z sind die primären Einflußgrößen. Das Ergebnis ist in Abbildung 7.7 zu sehen.

7.4 Erstellung des Kettengraphen

Der nächste auszuführende Schritt betrifft das Daten–Screening, bei dem festgestellt wird, welche Interaktionen bzw. nichtlinearen Transformationen im 2. Schritt des Selektionsalgorithmus zu berücksichtigen sind (vgl. Abschnitt 3.2). Da das Screening im Programm aber noch nicht realisiert ist und zudem nicht unbedingt durchgeführt werden muß, wird es im Beispiel übersprungen. Schließlich kann mit der eigentlichen Erstellung des Kettengraphen begonnen werden, indem der Menüpunkt `Analyse - Kettengraph berechnen` gewählt wird. Im daraufhin erscheinenden Dialog, der in Abbildung 7.8 dargestellt ist, kann der Benutzer zunächst allgemeine Richtlinien für die Erstellung des Graphen bestimmen. So kann jeweils über DropDown–

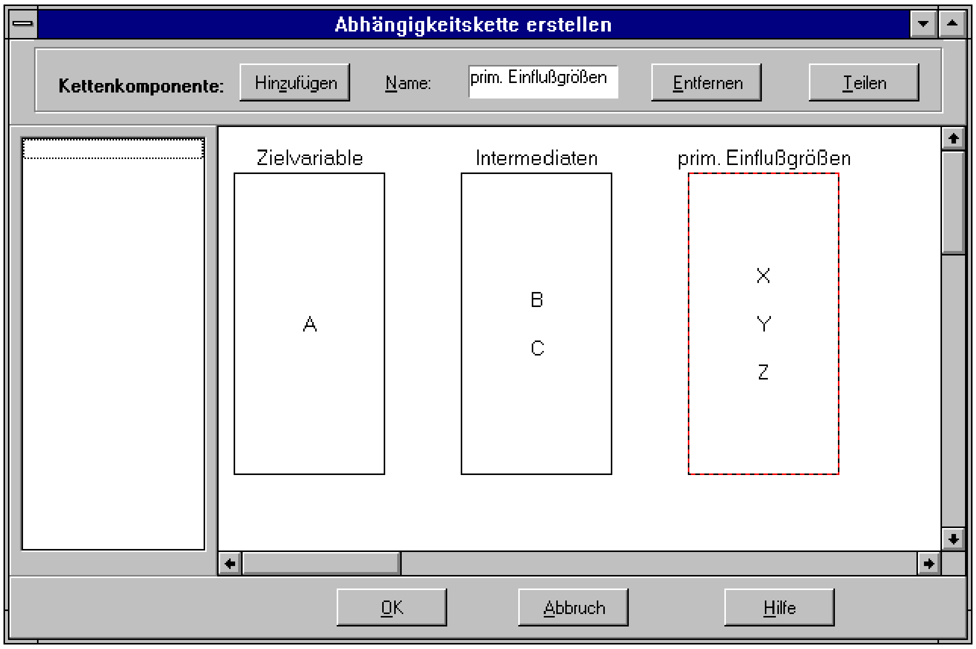


Abbildung 7.7: Erstellung der Abhängigkeitskette – Ende

Listen die Art des zu berechnenden Graphen, also Konzentrations– oder Kovarianzgraph, und die Regressionsart gewählt werden. Hierbei ist zu beachten, daß der Inhalt der DropDown–Liste für die Regression von der gewählten Graphenart abhängig ist, da nicht alle Regressionsarten für die beiden Arten von Graphen sinnvoll sind. Im unteren Bereich des Dialogs erfolgt die Festlegung der Anzahl der Schritte, die der Rechner automatisch ausführen soll, bevor der Benutzer eingreifen kann. In der DropDown–Liste `Berechnungsart` erfolgt eine grundlegende Unterscheidung, und zwar die, ob der Graph ganz automatisch berechnet werden soll, oder ob man nach einer gewissen Anzahl von Schritten anhalten und eventuell in die Berechnung eingreifen möchte. Wird hier der Punkt `schrittweise` gewählt, so erscheinen neue Dialogelemente, in denen man genauer festlegen kann, wieviele Schritte automatisch zu berechnen sind. In der DropDown–Liste `Schrittweite` erfolgt die Grobeinteilung, und im Eingabefeld `Anzahl` wird die gewünschte Anzahl der gewählten Schrittweite angegeben. Folgende Schrittweiten sind in der Liste einstellbar. Eine vollständige Berechnung einer gewissen Anzahl von Kettenkomponenten erfolgt durch die Wahl von `Kettenkomponente`. Durch die Einstellung `Variable` kann man eine bestimmte Anzahl von Variablen automatisch berechnen. Die feinste Einteilung ist diejenige, bei der man die Berechnung einer

Variable mitverfolgt. Hierzu muß als Schrittweite `Einzelschritt der Sel.` gewählt werden. In diesem Modus hält das Programm nach jedem Schritt der Selektionsstrategie an und zeigt das Ergebnis. Da hier die Angabe einer Anzahl nicht sinnvoll ist, verschwindet bei der Auswahl dieses Punktes die Eingabefläche `Anzahl`.
Im Beispiel wird ein Konzentrationsgraph mittels einer CG–Regression berechnet. Der Graph soll nach der Ausführung der Berechnung einer Kettenkomponente wieder angezeigt werden.

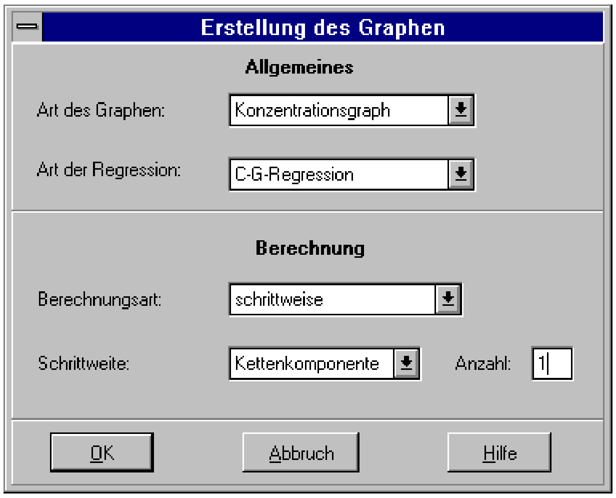


Abbildung 7.8: Dialog zur Erstellung eines Graphen

Nun müssen noch die kritischen Werte festgelegt werden, die der Selektionsstrategie zugrunde gelegt werden, anhand derer Variablen aus dem Modell entfernt bzw. in das Modell aufgenommen werden (vgl. Abschnitt 3.2), was in dem in Abbildung 7.9 dargestellten Dialog erfolgt. Dabei ist es möglich, daß für die verschiedenen Arten von Einflüssen verschiedene kritische Werte sinnvoll sind. Deshalb kann jede Effektart mittels verschiedener Eingabefelder ein eigener Wert zugewiesen werden. Um unerfahrene Benutzer nicht zu verwirren, sind in allen Eingabefeldern schon bestimmte Werte vorgegeben, die sich als sinnvoll erwiesen haben.
Im Beispiel wurden die Standardkriterien für die CG–Regression, nämlich jeweils der Wert 2, übernommen. Nach Abschluß dieses Dialogs beginnt die Berechnung des Graphen.

Abbildung 7.9: Dialog zur Festlegung der kritischen Werte für die Selektionsstrategie

Kapitel 8

Zusammenfassung und Ausblick

Graphische Modelle sind ein wirkungsvolles Instrument zur Analyse komplexer Zusammenhangsstrukturen. Problematisch ist derzeit jedoch, daß kaum Möglichkeiten zu einer praktikablen und anwenderfreundlichen Modellgenerierung bestehen. Diesem Dilemma zwischen praktischer Bedeutung und Nutzung graphischer Modelle kann durch die Bereitstellung geeigneter Software abgeholfen werden. Ein wichtiger Schritt dazu wurde in dieser Arbeit geleistet, in der eine umfassende Konzeptionalisierung eines solchen Systems erstellt und teilweise implementiert wurde. Die für die Erstellung eines Programms wesentlichen Elemente graphischer Modelle wurden dabei herausgearbeitet. Es zeigte sich, daß ein oberflächliches Verständnis graphischer Modelle für ein Design nicht ausreicht, sondern der Interdisziplinarität der Arbeit Rechnung getragen werden muß. Die enge Verzahnung der relevanten Bereiche zeigte sich z.B. darin, daß genaue Kenntnis darüber vorhanden sein mußte, zu welchen Zeitpunkten eine Unterbrechung der laufendenen Selektionsstrategie sinnvoll ist, um die zu diesem Zeitpunkt vorliegende graphische Darstellung auszugeben.
An der Zunahme der Publikationen und Forschungsprojekte im Bereich graphischer Modelle ist zu sehen, daß es sich um ein aktives und innovatives Forschungsgebiet handelt. So ist auch die Erstellung des Programms in ein Projekt eines Sonderforschungsbereichs der Deutschen Forschungsgesellschaft eingebettet. Beim Entwurf des Programms wurde deshalb darauf geachtet, ihn so flexibel wie möglich zu gestalten, um möglichst viele Forschungsergebnisse mit aufnehmen zu können. Um dies zu erreichen, wurden Methoden objektorientierter Analyse und Design verwendet und auch eine objektorientierte Sprache, nämlich Delphi, eingesetzt.
Zudem wurden Ideen für Eingriffsmöglichkeiten in die graphische Darstellung der Modellgenerierung gefunden, bei der sowohl zusätzliche Information zum Modell ange-

fordert als auch ein Eingriff in die Berechnung vorgenommen werden kann. Hierbei wurde ein Ansatz gewählt, der intuitiv zu bedienen ist und dabei aber den Anwender vor nicht mehr zu behebenden Fehlern schützt. Besondere Aufmerksamkeit wurde auch darauf verwendet zu erkennen, welche zusätzliche Information bei der Erstellung eines graphischen Modells im Graphen interessant ist. Ein Beispiel hierfür ist, die Variablen farblich hervorzuheben, die unabhängig von einer gegebenen Variablen sind.

Die Implementierung des Programms wurde, wie schon öfter erwähnt, mittels der Programmiersprache Delphi vorgenommen. Delphi ist eine objektorientierte Sprache, die auf Objekt–Pascal aufbaut. Das Besondere daran ist, daß im Programm vorkommende Formulare nicht programmiert werden müssen, sondern visuell erstellt werden können. Dies ist insbesonders bei der Entwicklung eines Prototypen von Vorteil, da der statische Aspekt der Anwendung schnell erzeugt werden kann und somit eine gute Grundlage für Diskussionen über den grundlegenden Aufbau des Programms geschaffen wird. Allerdings ist Delphi anzumerken, daß der objektorientierte Teil ein Aufsatz auf die Programmiersprache Pascal ist. So beherrscht Delphi nur das Konzept der Einfachvererbung, ohne aber als Ergänzung dazu das Konzept von Schnittstellenklassen vernünftig realisiert zu haben. Außerdem ist es nicht möglich, wie in anderen objektorientierten Programmiersprachen üblich, zwei Methoden, die sich in ihrer Signatur unterscheiden, mit gleichen Namen zu versehen.
Die Diagramme der UML–Notation wurden mit Hilfe der Software "Rational Rose C++ 4.0 Demoversion" erzeugt. Dieses Tool wurde von der Firma Rational entwickelt, die auch die UML erstellt hat. Das Programm hat primär die Aufgabe, aus Spezifikationen, die in UML–Notation eingegeben werden, Rümpfe von Klassen in C++ Notation zu erstellen. Da in dieser Arbeit nicht C++ als Programmiersprache eingesetzt wurde und Rational Rose nur noch in einer Java–Version vorliegt, beschränkte sich die Nutzung des Tools darauf, die Modellierung des Programms darzustellen. Hierbei zeigte sich, daß noch nicht alle Elemente, die in der UML–Spezifikation angegeben sind, in der derzeitigen Version enthalten sind. Zudem ist die Version noch instabil. Wenn in zukünftigen Versionen diese Fehler behoben sind, stellt Rational Rose sicherlich ein hilfreiches Werkzeug für die Analyse und das Design objektorientierter Software dar.

Damit steht eine Software zur Verfügung, die folgende Leistungsmerkmale besitzt. Die Selektionsstrategie für die Erstellung eines graphischen Modells kann komplett mittels eines Programms durchlaufen werden. Die Ergebnisse dieser Selektionsstrategie können auf Wunsch des Benutzers zu jedem Einzelschritt visualisiert werden. Bei der Darstellung der Visualisierung ist eine interaktive Modifikation des Graphen möglich, die so-

fern nötig automatisch zu einer teilweisen Neuberechnung des Modells führt. Gerade in der automatisierten Darstellung und in der Modifikationsmöglichkeit des Graphen zu jedem Zeitpunkt der Modellanpassung liegt der besondere Verdienst des Programms, da die umständliche manuelle Umsetzung textueller Auswertungsergebnisse entfällt. Die für die Selektionsstrategie nötigen Regressionen sind selbst bisher nicht implementiert. In diesem Zusammenhang wird noch nach geeigneten Anknüpfungspunkten zu schon bestehenden Programmen gesucht.
Bei der Entwicklung des Programms erwies sich die Kommunikation mit denen am Projekt graphischer Modelle beteiligten Mitarbeiterinnen des Instituts für Statistik als konstruktiv. Den größten Diskussionspunkt hierbei bildeten die verschiedenen Sichtweisen auf die Problematik, die Modellanpassung bei graphischen Modellen in ein Programm umzusetzen.

Allerdings bietet die umfassende Aufgabenstellung der Software–Entwicklung auf dem Gebiet der graphischen Modelle noch eine Reihe sinnvoller und wünschenswerter Erweiterungsmöglichkeiten, die über die noch zu implementierenden Teile des Programms (vgl. Abschnitt 5.3 und Abschnitt 6.3) hinausgehen. So werden in vielen statistischen Untersuchungen die erhobenen Daten in einer Datenbank abgespeichert, so daß es günstig wäre, im Programm eine Schnittstelle zum Einlesen der Daten aus einer Datenbank zu schaffen. Dies ist mittels der Sprache Delphi gut zu lösen, da die Anbindung an Datenbanken ein wichtiges Leistungsmerkmal der Sprache ist. Zum einen kann eine Vielzahl von bekannten Datenbanken, wie z.B. Oracle, Access oder Paradox, über die gleichen Befehle angesprochen werden, da in Delphi eine Schnittstelle vorhanden ist, die den Zugriff auf diese Datenbanken kapselt. Zum anderen sind die Elemente des Datenbankzugriffs in die visuelle Komponente von Delphi integriert, so daß Abfragen schon zur Entwurfszeit sichtbar gemacht werden können.
Als Nachteilig könnte sich erweisen, daß die Ausführung des Programms auf Rechner beschränkt ist, die als Betriebssystem eine 32 Bit Version von Microsoft–Windows, wie z.B. Windows 95 oder Windows NT, besitzen. Diese Art von Betriebssystem ist zwar weit verbreitet, aber gerade in der Forschung werden auch andere Betriebssysteme, wie z.B. UNIX, genutzt. Es wäre deshalb günstig, zumindest eine Verbindung zu Rechnern mit derartigen Betriebssystemen im Programm herstellen zu können, um Daten einzulesen, die auf diesen Rechnern abgespeichert sind. Als Lösungsansatz bietet sich hier an, die Daten mittels “File Transfer Protocol”, kurz FTP, auf den Rechner, zu übertragen, auf dem das Programm installiert ist. Dies läßt sich auch innerhalb des

Programms realisieren, da Delphi Klassen besitzt, mit denen eine FTP–Verbindung erstellt werden kann.
Wünschenswert wäre ebenfalls die Verfügbarkeit einer englischsprachigen Version, um eine universellere Nutzung des Programms zu ermöglichen. Dazu müssen alle im Programm erscheinenden Texte gesondert in einer sogenannten "Resource–Datei" gespeichert werden. Innerhalb des Programmes werden Verweise auf diese Datei erstellt, in denen festgelegt wird, an welcher Stelle der entsprechende Text erscheinen soll.
Zudem ist es denkbar, zusätzlich zur vorliegenden Bedienung über Menüs einen Kommandozeileninterpreter einzubauen, mit dem Befehle eingegeben werden könnnen. Damit ermöglicht man geübten Benutzern, mittels dieser Eingabeform schneller zu Resultaten zu kommen.
Für die Zukunft wird eine Kommunikation mit anderen Programmen angestrebt, die eine ähnliche Problematik bearbeiten. Ein solches Programm ist MIM von David Edwards, das bislang die Analyse ungerichteter graphischer Modelle ermöglicht, wobei eine Erweiterung auf allgemeinere Graphen geplant ist. Die wichtigste Dokumentation zu diesem Programm ist in (Edwards 1995) zu finden.
Eine interessante Aufgabe besteht zudem in der Untersuchung, wie in Abschnitt 6.3 schon angesprochen, inwiefern sich die Erzeugung des Graphen auf einer höheren Abstraktionsebene durchführen ließe. Hierbei ist es notwendig, eine Abstraktion zu finden, mit der auch andere Aufgabengebiete behandelt werden können. Dazu müßte das Gebiet gegenüber schon vorhandenen Ansätzen zu einer Sprache für Graphen abgegrenzt werden. Zudem muß das Schema der Graphen bei einem graphischen Modell genauer unter die Lupe genommen und mit Schemata anderer Grapharten verglichen werden.

Anhang A

Formale Definition wichtiger Begriffe aus der Statistik

A.1 Gemeinsame Verteilungen und Dichten

Die folgenden Definitionen basieren im wesentlichen auf Fahrmeir und Hamerle (1984, S. 19ff) und beziehen sich auf grundlegende Eigenschaften von Zufallsvariablen und deren Beziehungen untereinander. Dabei handelt es sich, anschaulich formuliert, bei einer Zufallsvariable X um eine reellwertige Variable, deren verschiedene Werte den Ausgang eines Zufallsvorgangs wiedergeben, also selbst zufällig sind. Zur formalen Definition einer Zufallsvariable sind maßtheoretische Begriffe notwendig, worauf an dieser Stelle verzichtet wird. Es sei noch ergänzend erwähnt, daß eine mehrdimensionale Zufallsvariable $\mathbf{X} = (X_1, \ldots, X_p)'$ als Zufallsvektor bezeichnet wird.

Definition A.1 ((gemeinsame) Verteilungsfunktion) *Sei* $\mathbf{X}$ *ein Zufallsvektor. Als (gemeinsame) Verteilungsfunktion* $F_{\mathbf{X}}(\mathbf{x})$ *des Zufallsvektors* $\mathbf{X}$ *bezeichnet man die Funktion*

$$F_{\mathbf{X}}(\mathbf{x}) = F_{X_1,\ldots,X_p}(x_1, \ldots, x_p) = P(X_1 \leq x_1, \ldots, X_p \leq x_p) = P(\mathbf{X} \leq \mathbf{x}).$$

Definition A.2 (Stetigkeit von X, (gemeinsame) Dichte) $\mathbf{X}$ *heißt stetig, wenn es eine Funktion* $f_{\mathbf{X}}(\mathbf{x}) = f_{X_1,\ldots,X_p}(x_1, \ldots, x_p) \geq 0$ *mit*

$$F_{\mathbf{X}}(\mathbf{x}) = F_{X_1,\ldots,X_p}(x_1, \ldots, x_p) = \int_{-\infty}^{x_p} \cdots \int_{-\infty}^{x_1} f_{X_1,\ldots,X_p}(u_1, \ldots, u_p) du_1 \ldots du_p = \int_{-\infty}^{\mathbf{x}} f_{\mathbf{X}}(\mathbf{u}) d\mathbf{u}$$

gibt. Die Funktion $f_{\mathbf{X}}$ *heißt (gemeinsame) Dichte von* $\mathbf{X}$.

Definition A.3 (Randverteilungsfunktion, Randdichte) *Ein $(p+q)$– dimensionaler Zufallsvektor* $\mathbf{Z}$ *sei partitioniert in* $\mathbf{Z}' = (\mathbf{X}', \mathbf{Y}')$, *wobei* $\mathbf{X}$ *und* $\mathbf{Y}$ *p– bzw. q–dimensional sind.*

$$F_{\mathbf{X}}(\mathbf{x}) = P(\mathbf{X} \leq \mathbf{x}) = F_{\mathbf{X},\mathbf{Y}}(x_1, \dots, x_p, \infty, \dots, \infty)$$

heißt dann Randverteilungsfunktion von $\mathbf{X}$. *Entsprechendes gilt für* $\mathbf{Y}$. *Eine zu* $F_{\mathbf{X}}$ *gehörige Dichte* $f_{\mathbf{X}}$ *heißt Randdichte von* $\mathbf{X}$, *und es ist*

$$f_{\mathbf{X}}(\mathbf{x}) = \int_{-\infty}^{\infty} f_{\mathbf{X},\mathbf{Y}}(\mathbf{x}, \mathbf{y}) d\mathbf{y}.$$

A.2 Unabhängigkeit von Zufallsvariablen

Die folgenden Definitionen und Sätze richten sich nach (Kuhnt 1994). Alle hier betrachteten Verteilungen besitzen eine streng positive Dichte.

Definition A.4 (Bedingte Dichte) *Gegeben seien zwei Zufallsvariablen* X_1, X_2, *für die* f_{X_1,X_2} *die Dichte ihrer gemeinsamen Verteilung* F_{X_1,X_2} *ist. Dann ist die bedingte Dichte* $f_{X_2|X_1=x_1}$ *von* X_2 *bei festem* $X_1 = x_1$ *gegeben durch*

$$f_{X_2|X_1=x_1}(x_2|x_1) = \frac{f_{X_1,X_2}(x_1, x_2)}{f_{X_1}(x_1)}.$$

Folgerung A.5 (Faktorisierung der gemeinsamen Dichte) *Die gemeinsame Dichte* $f_{X_1,\dots,X_p}$ *eines Zufallsvektors läßt sich folgendermaßen faktorisieren*

$$f_{X_1,\dots,X_p} = f_{X_p|X_{p-1},\dots,X_1} \cdot f_{X_{p-2}|X_{p-3},\dots,X_1} \cdot \dots \cdot f_{X_1}.$$

Definition A.6 ((Marginale) Unabhängigkeit von Zufallsvariablen) *Es seien* $X_1, \dots, X_p$ *reelle Zufallsvariablen mit den Dichten* $f_{X_1}, \dots, f_{X_p}$. *Dann sind* $X_1, \dots, X_p$ *genau dann (marginal) unabhängig, wenn*

$$f_{X_1,X_2,\dots,X_p}(x_1, \dots, x_p) = f_{X_1}(x_1) \cdot \dots \cdot f_{X_p}(x_p)$$

die gemeinsame Dichte der Zufallsvariablen ist.

(Dawid 1979) gibt mehrere äquivalente Beziehungen für die (marginale) Unabhängigkeit zweier Zufallsvariablen X_1 und X_2 mit Dichten $f_{X_1}, f_{X_2}, f_{X_1,X_2}$ an. Dabei sind g_{X_1} und g_{X_2} Funktionen, die nicht notwendigerweise die entsprechenden Randdichten sein müssen.

Satz A.7 *Die Zufallsvariablen X_1 und X_2 sind (marginal) unabhängig, wenn eine der nachfolgenden Gleichungen gilt:*

i) $f_{X_1,X_2} = f_{X_1} \cdot f_{X_2}$

ii) $f_{X_1|X_2} = f_{X_1}$

iii) $f_{X_1,X_2} = g_{X_1} \cdot g_{X_2}$

iv) $f_{X_1|X_2} = g_{X_1}$.

(Dawid 1979) führt als Bezeichnung für die (marginale) Unabhängigkeit das Symbol $\perp$ ein. Damit bezeichnet $X_1 \perp X_2$, daß X_1 und X_2 unabhängig sind.

Zudem finden sich in (Dawid 1979) auch für die bedingte Unabhängigkeit mehrere äquivalente Definitionen.

Definition A.8 (Bedingte Unabhängigkeit von Zufallsvariablen) *Gegeben seien drei Zufallsvariablen X_1, X_2, X_3. Die bisher verwendeten Bezeichnungen gelten weiterhin. Dann sind X_1 und X_2 bedingt unabhängig gegeben X_3, bezeichnet durch $X_1 \perp X_2 | X_3$, falls eine der folgenden äquivalenten Bedingungen für alle x_3 gilt:*

i) $f_{X_1,X_2|X_3} = f_{X_1|X_3} \cdot f_{X_2|X_3}$

ii) $f_{X_1,X_2|X_3} = g_{X_1|X_3} \cdot g_{X_2|X_3}$

iii) $f_{X_1|X_2,X_3} = f_{X_1|X_3}$

iv) $f_{X_1|X_2,X_3} = g_{X_1|X_3}$.

Anhang B

Wichtige Begriffe aus der Objektorientierung

ABSTRAKTE KLASSE Abstrakte Klassen (*abstract classes*) werden zur Strukturierung einer Klassenhierarchie formuliert. Sie erzeugen selbst keine *Objekte*, sondern stellen lediglich *Attribute* und *Operationen* bereit, die von ihren *Unterklassen* benötigt werden.

AGGREGAT Ein Aggregat (*aggregation class*) hat die Aufgabe, die an eine *Klasse* gestellten Anforderungen an ihre Teile zu delegieren. Sie enthält dazu z.B. *Operationen*, welche die auszuführenden Aufgaben an die Teile weiterleiten. Im Gegensatz zur *Assoziation* haben hier die in Beziehung stehenden Klassen keine gleichberechtigte Rolle inne, sondern das Aggregat übernimmt die Verantwortung und die Führung.

AGGREGATION Unter einer Aggregation (*aggregation*) versteht man die Zusammensetzung eines Konstrukts aus Einzelteilen. Die *Klasse*, die die Teile enthält, wird Aggregatsklasse oder kurz *Aggregat* genannt.

ASSOZIATION Eine Assoziation (*association*) dient der Kommunikation zwischen *Objekten*. Sie stellt eine Verbindung zwischen den verschiedenen beteiligten *Klassen* her. Eine konkrete Beziehung zwischen den Objekten dieser Klassen wird *Objektverbindung* genannt.

ATTRIBUT Ein Attribut (*attribute*) ist eine Dateneigenschaft, die für jedes *Objekt* einer *Klasse* vorliegt und für jedes Objekt einen individuellen Wert annimmt.

DISKRIMINATOR Unter einem Diskriminator (*discriminator*) versteht man ein Unterscheidungsmerkmal, das *Klassen* in *Ober–* und *Unterklassen* aufteilt und somit die Bildung einer Klassenhierarchie impliziert.

EINFACHVERERBUNG Von Einfachvererbung (*single inheritance*) spricht man, wenn eine *Klasse* höchstens eine *Oberklasse* besitzt.

GENERALISIERUNG Unter Generalisierung (*generalization*) versteht man ein Abstraktionsprinzip zur hierarchischen Strukturierung von Modellen. Es werden dabei die Eigenschaften von *Klassen* gegliedert. Eigenschaften allgemeinerer Bedeutung werden *Oberklassen*, spezielle Eigenschaften dagegen werden *Unterklassen* zugeordnet. Dabei kann eine Klasse sowohl Unterklasse einer Klasse und selbst wiederum Oberklasse von anderen Klassen sein, was zu einer hierarchischen Struktur führt. Die Eigenschaften der Oberklasse werden an ihre Unterklassen weitergegeben, so daß die Unterklasse die Eigenschaften all ihrer Oberklassen und die von ihr selbst definierten Eigenschaften enthält. Die Aufteilung in Ober– und Unterklassen erfolgt meist durch den *Diskriminator*.

KLASSE Eine Klasse (*class*) beschreibt eine Menge von *Objekten*, die ähnliche Struktur, Verhalten und Beziehungen besitzen. Die Definition einer Klasse erfolgt durch die Angabe ihrer *Attribute* und *Operationen*.

KOMPOSITION Die Komposition (*composition*) ist eine strenge Form der *Aggregation*. Bei ihr sind die Teile vom *Aggregat* existenzabhängig. Das heißt, daß bei der Erzeugung oder Zerstörung des Aggregats auch seine Teile erzeugt bzw. gelöscht werden.

MEHRFACHVERERBUNG Von Mehrfachvererbung (*multiple inheritance*) spricht man, wenn eine *Klasse* mehrere *Oberklassen* besitzen kann.

METAKLASSE Eine Metaklasse (*meta class*) ist eine *Klasse*, deren *Objekte* wiederum Klassen sind.

METHODE Eine Methode (*method*) ist die Implementierung einer *Operation*. In vielen Programmiersprachen wird allerdings zwischen Methode und Operation oft nicht unterschieden.

NACHRICHT Nachrichten sind ein Mechanismus zur Objektkommunikation. Eine Nachricht (*message*) bringt einem *Objekt* eine Information darüber, was für eine Aktivität von ihm gefordert wird.

OBERKLASSE Eine Oberklasse (*superclass*) enthält allgemeinere Eigenschaften, die jede ihrer *Unterklassen* besitzt. Die Eigenschaften werden an die Unterklassen weitergegeben.

OBJEKT Ein Objekt (*instance*) ist ein bestimmtes Exemplar einer *Klasse*, d.h. ein Objekt enthält eine Ausprägung aller *Attribute* einer Klasse und kann die in der Klasse definierten *Operationen* ausführen.

OBJEKTVERBINDUNG Eine Objektverbindung (*link*) ist eine konkrete Beziehung zwischen den *Objekten* von *Klassen*, die durch eine *Assoziation* verbunden sind. Eine Objektverbindung ist also ein Objekt einer Assoziation.

OPERATION Eine Operation (*operation*) ist eine Dienstleistung, die ein *Objekt* ausführen kann. Sie ist mit einer Funktion oder Prozedur im traditionellen imperativen Paradigma vergleichbar. Eine Operation wird durch ihre Signatur beschrieben, welche sich aus dem Operationsnamen, den Übergabeparametern und dem Rückgabetyp zusammensetzt. Operationen werden durch *Methoden* implementiert. In der Praxis wird der Begriff der Operation und der Methode häufig synonym verwendet.

SCHNITTSTELLENKLASSE Eine Schnittstellenklasse (*interface*) ist eine *abstrakte Klasse*, die das externe Verhalten von *Klassen* spezifiziert. Sie enthält die Beschreibung von *Attributen* und *Operationen* in abstrakter Form. Klassen, welche die von einer Schnittstelle zur Verfügung gestellten Attribute und Operationen implementieren, stellen Umsetzungen dieser Schnittstelle dar.

SICHTBARKEIT Die Sichtbarkeit (*visibility*) eines Elements einer *Klasse* gibt an, welche Klassen darauf zugreifen können. Hierbei gibt es drei Arten von Sichtbarkeitsstufen. Auf ein Element der Sichtbarkeitsstufe `public` kann jede Klasse zugreifen. `Protected` bedeutet, daß die Klasse selbst und ihre *Unterklassen* dieses Element ansprechen können. Die größte Einschränkung beim Zugriff auf Klassenelemente ist die Sichtbarkeitsstufe `private`. Besitzt ein Element diese Stufe, kann nur die Klasse selbst darauf zugreifen.

UNTERKLASSE Eine Unterklasse (*subclass*) ist einer oder mehreren *Oberklassen* zugeordnet. Sie enthält alle von ihren Oberklassen und die von ihr selbst definierten Eigenschaften.

VERERBUNG Die Vererbung (*inheritance*) beschreibt ein Programmiersprachenkonzept. Sie ist der Umsetzungsmechanismus für die Beziehungen zwischen *Ober–* und *Unterklassen*. Der Begriff der Vererbung wird in der Literatur häufig synonym zur *Generalisierung* verwendet.

Anhang C

UML Notation

C.1 Anwendungsfalldiagramm (use case diagram)

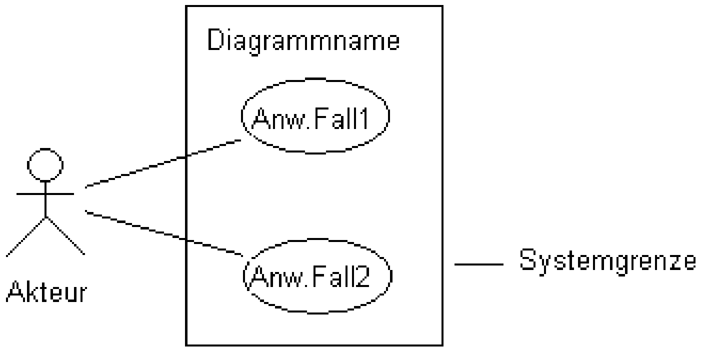


Abbildung C.1: Anwendungsfalldiagramm

C.2 Klassendiagramm (class diagram)

C.2.1 Basiselemente

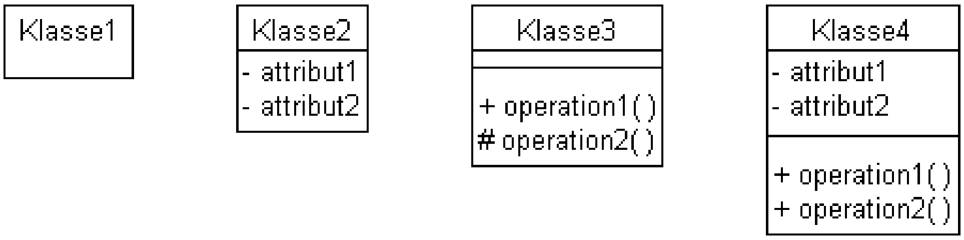


Abbildung C.2: Klassen

Abbildung C.3: Objekte

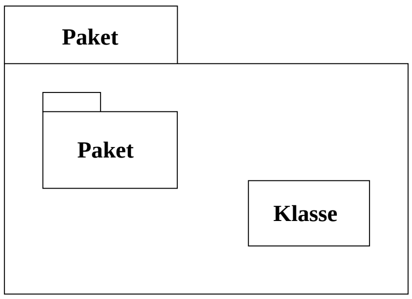


Abbildung C.4: Paket

C.2.2 Beziehungen

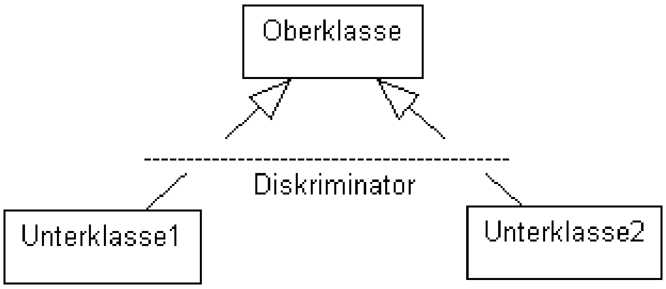


Abbildung C.5: Generalisierung

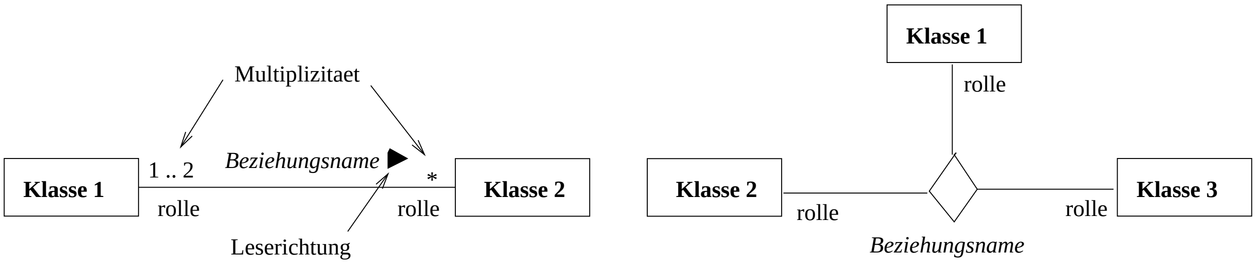


Abbildung C.6: Assoziation und ternäre Assoziation

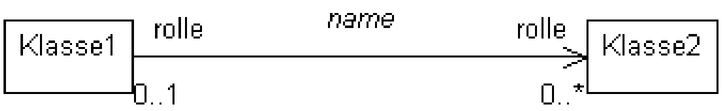


Abbildung C.7: gerichtete Assoziation

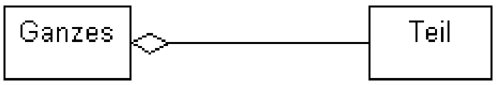


Abbildung C.8: Aggregation

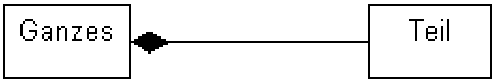


Abbildung C.9: Komposition

C.3 Verhaltensdiagramme (behavior diagrams)

C.3.1 Sequenzdiagramm (sequence diagram)

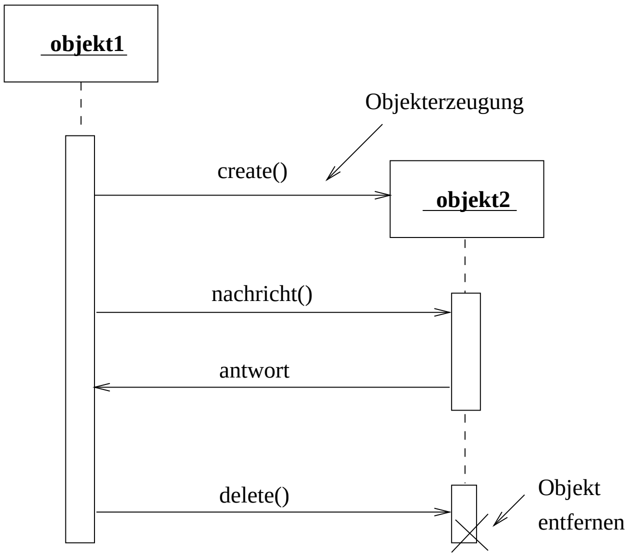


Abbildung C.10: Sequenzdiagramm

C.3.2 Zustandsdiagramm (statechart diagram)

Abbildung C.11: Zustandsdiagramm

Literatur

Booch, G. (1994). *Objektorientierte Analyse und Design: Mit praktischen Anwendungsbeispielen.* Addison-Wesley, Bonn.

Caputo, A. (1998). *Eine alternative Familie von Modellverteilungen für Kovarianz– und Konzentrationsgraphen.* Dissertation, Ludwig–Maximilians–Universität–München.

Cox, D. R. und N. Wermuth (1994). Tests of linearity, multivariate normality and the adequacy of linear scores. *Applied Statistics 94*, 347 – 355.

Cox, D. R. und N. Wermuth (1996). *Multivariate Depencencies – Models, Analysis and Interpretation.* Chapman and Hall, London.

Dawid, A. P. (1979). Conditional independence in statistical theory. *Journal of the Royal Statistical Society, Series B 41*, 1 – 31.

Deutsches Institut für Normung (1988). *DIN 66234 Teil 8: Bildschirmarbeitsplätze; Grundsätze ergonomischer Dialoggestaltung.* Deutsches Institut für Normung.

Edwards, D. (1995). *Introduction to Graphical Modelling.* Springer, New York.

Fahrmeir, L. und A. Hamerle (Hrsg.) (1984). *Multivariate statistische Verfahren.* de Gruyter, Berlin.

Fahrmeir, L., R. Künstler, I. Pigeot und G. Tutz (1997). *Statistik – Der Weg zur Datenanalyse.* Springer, Berlin Heidelberg New York.

Fröhlich, M. und M. Werner (1994). Demonstration of the interactive graph–visualization system da vinci. *Lecture Notes in Computer Science 894*, 266 – 269.

Heinicke, A. (1996). Anwendung und Grenzen graphischer Kettenmodelle. Diplomarbeit, Ludwig–Maximilians–Universität München.

Himsolt, M. (1997). The graphlet system (system demonstration). *Lecture Notes on Computer Science 1190*, 233 – 240.

Jacobsen, I., M. Christerson, P. Jonsson und G. Overgaard (1992). *Object-Oriented Software Engineering*. Addison-Wesley, New York.

Koehler, K. J. und J. T. Symanowski (1995). Constructing multivariate distributions with specific marginal distributions. *Journal of Multivariate Analysis 55*, 261 – 282.

Kuhnt, S. (1994). Graphische Modelle und ihre Anwendungen in der Statistik. Diplomarbeit, Universität Dortmund.

Lauritzen, S. L. (1996). *Graphical Models*. Clarendon Press, Oxford.

Lauritzen, S. L. und N. Wermuth (1989). Graphical models for associations between variables, some of which are qualitative and some quantitative. *The Annals of Statistics 17*, 31 – 57.

Microsoft Corporation (1992). *The Windows Interface; An Application Design Guide*. Microsoft Corporation.

North, S. and E. Koutsofios (1998). Applications of graph visualization. Technical report, AT&T Bell Laboratories, USA.

Oestereich, B. (1997). *Objektorientierte Softwareentwicklung mit der Unified Modeling Language* (3. Aufl.). R. Oldenbourg, München.

Pigeot, I. und R. Streit (1994). Applications of graphical chain models: basic idea and examples (with summary of plenary discussion). In H.-J. Schmidt (Hrsg.), *Proceedings of the International Seminar 'Problem Solving and Misconceptions in Chemistry and Physics', Dortmund, 1994*, 31 – 45.

Rational Software Corporation (1997a, September). *UML Summary - Version 1.1*. Rational Software Corporation.

Rational Software Corporation (1997b, September). *UML Notation Guide - Version 1.1*. Rational Software Corporation.

Rational Software Corporation (1997c, September). *UML Semantics - Version 1.1*. Rational Software Corporation.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy und W. Lorensen (1991). *Object-oriented Modeling and Design*. Prentice Hall, Englewood Cliffs.

Sander, G. (1995). Vcg – visualization of compiler graphs. Technical report, Universität des Saarlandes.

Shneiderman, B. (1992). *Designing the User Interface: Strategies for Effective Human–Computer Interaction* (2. Aufl.). Addison-Wesley, New York.

Vermunt, J. K. (1997). *Log–linear Models for Event Histories*. SAGE Publications, London.

Warken, E. (1997). *Delphi 3 – Entwicklung leistungsfähiger Anwendungen und Komponenten*. Addison–Wesley, Bonn.

Whittaker, J. (1991). *Graphical Models in Applied Multivariate Statistics*. John Wiley and Sons, Chichester.