Universität Hamburg

Fachbereich Informatik, Arbeitsbereich Softwaretechnik

Ein objektorientierter Interaktionstyp für die dreidimensionale Darstellung von dynamischen Statistiken unter Motif in C++

Studienarbeit

Andreas Kornstädt

betreut von Guido Gryczan

Inhaltsverzeichnis

Vorwort ... III

1 Einleitung ... 1

2 Kelly-Grids ... 3

3 Dynamische Statistiken ... 5

4 Objektorientierte Entwicklung mit der Werkzeug-Material-Metapher ... 11

4.1 Anforderungsermittlung ... 12

4.2 Entwurf ... 12

4.3 Implementation ... 15

4.3.1 Die FIAK-Bibliothek ... 16

4.3.2 Die IATMotif-Bibliothek ... 17

5 Entwicklung des Interaktionstyps ... 19

5.1 Anforderungen ... 19

5.2 Entwurf ... 22

5.2.1 Material ... 24

5.2.2 Werkzeug (FK) ... 27

5.2.3 Motif-Aspekt ... 30

5.2.4 Werkzeug (IAK) ... 31

5.3 Implementation ... 35

5.3.1 Material ... 36

5.3.2 Werkzeug (FK) ... 39

5.3.3 Aspekt ... 39

5.3.4 Werkzeug (IAK) ... 40

5.4 Bewertung ... 40

6 Zusammenfassung und Ausblick ... 47

Anhänge ... V

Zeichensatzkonventionen ... V

Graphische Konventionen ... V

Namenskonventionen ... V

Im Programm V

Im Betriebssystem..... VI

Literaturverzeichnis VI

Vorwort

Die Anzahl der Unzulänglichkeiten in dieser Studienarbeit bezüglich der Kernpunkte der am Arbeitsbereich Softwaretechnik gelehrten Inhalte, insbesondere der Dokumenttypen in frühen Stadien des Entwicklungsprozesses und der evolutionären Vorgehensweise veranlassen mich zu diesem Vorwort.

Diese Arbeit entstand in mehrerlei Hinsicht unter nicht optimalen Bedingungen:

- Der zukünftige Benutzer war nach den einleitenden Besprechungen kaum noch direkt persönlich erreichbar, was ein evolutionäres Vorgehen nahezu ausschloß.
- Zum Zeitpunkt der einleitenden Besprechungen hatte ich die Dokumenttypen des Entwicklungsprozesses noch nicht wirklich kennengelernt, so daß eine Erstellung von Szenarios, Systemvisionen und Prototypen nicht erfolgen konnte. Eine Rekonstruktion wäre unehrlich und wenig aussagekräftig gewesen.

Aufgrund dieser Tatsachen ist hauptsächlich die Beschreibung der technische Umsetzung der Werkzeug-Material-Metapher, nämlich Trennung in Werkzeug (mit FK, IAK und IATs) und Material, sowie eine ausführliche Bewertung der IATMotif-Bibliothek von softwaretechnischem Interesse.

Trotzdem oder gerade deswegen hat diese Studienarbeit ihr Ziel erreicht, das im Lernprozeß besteht. Ohne die hier erkannten Fehler und Mißstände wären die Vorbereitungen auf meine Diplomarbeit gewiß wesentlich ungünstiger verlaufen.

1 Einleitung

Die "Psychologie der Persönlichen Konstrukte" ist ein Teilgebiet der Psychologie. Ein wichtiges Verfahren zur Analyse der psychischen Struktur von Versuchspersonen stellt das Repertory- oder Kelly-Grid-Verfahren dar. Zu bestimmten Dingen (Elementen) müssen Bewertungen bezüglich bestimmter Eigenschaften (Konstrukte) vorgenommen werden, so daß ein n-dimensionaler Ergebnisvektorraum aufgespannt wird. Die Auswertung dieser Daten gestaltet sich aufgrund der resultierenden Unübersichtlichkeit ohne Rechnerunterstützung als sehr zeitaufwendig.

Eine Möglichkeit der Unterstützung sind dynamische Statistiken, mit denen vor allem dreidimensionale Daten intuitiv auf graphische Weise analysiert werden können. Im Gegensatz zu starren Tabellen kann der Auswerter in Echtzeit die Darstellung der Daten so verändern, daß eine fachlich aussagekräftige Analyse möglich ist.

Gemäß dem aktuellen Stand der Softwaretechnik bieten sich objektorientierte Modellierungsmethoden zur Realisierung von Anwendungssystemen an, da sie es erlauben, die sematische Lücke zwischen fachlichem und technischem Entwurf verhältnismäßig einfach zu überbrücken. Einen Leitfaden zum Vorgehen bietet hierbei die am Arbeitsbereich Softwaretechnik verwendete Werkzeug-Material-Metapher, die die bruchlose Modellierung -vor allem von Einzelbüroarbeitsplätzen- über alle Projektstadien hinweg unterstützt

Gegenstand dieser Arbeit ist die Entwicklung eines Dynamische-Statistik-Werkzeugs zur Analyse von Kelly-Grids. Anhand dieses Beispiels wird untersucht, inwiefern die Werkzeug-Material-Metapher sich auf ein Problem, das nicht unmittelbar aus dem Bereich der Büroarbeit kommt, anwenden läßt. Des weiteren wird festgestellt, ob die zur Unterstützung der Implementation vorhandene Interaktionstypenbibliothek gemäß den entstehenden Erfordernissen erweiterbar ist. Schließlich wird beurteilt, ob das Dynamische-Statistik-Werkzeug in Form eines separaten Interaktionstyps in die Bibliothek integriert werden kann.

In Kapitel 2 wird zunächst die Kelly-Grid-Methode geschildert und die zu deren Interpretation relevanten graphischen Methoden erwähnt. In Kapitel 3 werden daraufhin dynamische Statistiken als allgemeine Datenanalyse-Hilfsmittel eingeführt, die sich auch zur Kelly-Grid-Analyse eignen. Anschließend wird in Kapitel 4 die Werkzeug-Material-Methode über alle Projektstadien hinweg beschrieben. Kapitel 5 enthält die eigentliche Entwicklung des Kelly-Grid-Analyse-Werkzeugs anhand der zuvor beschriebenen Methode. Hier werden die Erfahrungen im Umgang mit der Interaktionstypenbibliothek geschildert. Kapitel 6 beinhaltet eine kritische Würdigung des Erreichten.

2 Kelly-Grids

Laut der Theorie der "Psychologie der Persönlichen Konstrukte"[CatSch93], die 1955 von G.A. Kelly eingeführt wurde, konstruiert sich jeder Mensch ein Weltbild, das auf bestimmten individuellen Hypothesen fußt. Je nach Lebenssituation werden diese Weltbildstützen entweder verstärkt oder abgeschwächt. Äußere Eindrücke und insbesondere andere Personen werden anhand dieses Wertesystems eingeordnet.

Um also einen Einblick in die Psyche einer Person zu erhalten, wäre es eine große Hilfe, wenn man ein Verfahren besäße, mit dessen Hilfe man das aus Hypothesen und anhand derselben klassifizierten Personen bestehende Weltbild ermitteln könnte. Genau solch ein Verfahren hat Kelly entwickelt. Der Nutzen einer Analyse kann entweder in generellen psychologischen und philosophischen Betrachtungen liegen, sowie auch in ganz konkreten therapeutischen Bemühungen.

In der Fachsprache heißen die Hypothesen *Konstrukte*, die Personen *Elemente* und das Weltbild aus sämtlichen Konstrukten und Elementen Repertoire. Dieses *Repertory* kann matrixartig in einem *Repertory-Grid* dargestellt werden, welches Kelly zu Ehren auch *Kelly-Grid* genannt wird. Die analysierte Person wird *Versuchsperson* (Vp) genannt.

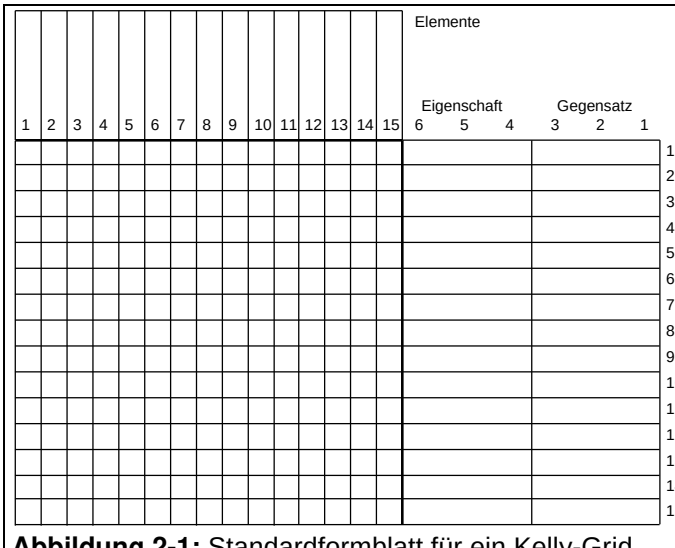


Abbildung 2-1: Standardformblatt für ein Kelly-Grid

Quelle: [Sch93, S.27]

Das Verfahren zur Gewinnung des Grids ist beliebig. Es existiert jedoch ein Standardverfahren[Sch93], das besonders für die Evaluierung des sozialen Umfelds von Vps anhand von mehrstufigen Einstufungskategorien (*Ratings*) geeignet ist. Zunächst wird in einer ersten Stufe eine Liste von für die Vp relevanten Personen (Elementen) erstellt (zu der auch das "Ich" und das "ideale Ich" zählen), anhand derer in einem zweiten Schritt die Konstrukte ermittelt werden. Hierbei werden der Vp vom *Untersucher*[1] z.B. drei der ausgewählten relevanten Personen genannt, von denen sie zwei gegenüber der verbleibenden Person anhand einer bestimmten Eigenschaft abgrenzen soll. Z.B. können Fritz und Erna sich dadurch von Willi unterscheiden, daß sie sich beide spontan verhalten, während Willi stets nur geplantes tut. Daraus läßt sich das erste Konstrukt (spontan) und sein Gegenpol (*Kontrastpol*) (geordnet) ableiten.[2] So wird fortgefahren, bis genauso viele Konstrukte vorhanden sind, wie Elemente. Anschließend soll

[1] Zur Erhöhung der Lesbarkeit werden ausschließlich männliche Personenbezeichnungen verwendet. Weibliche Personen fühlen sich bitte ebenso angesprochen.

[2] Der Kontrastpol zu "spontan" könnte ebensogut "langweilig", "unentschlossen" oder "erstarrt" lauten. Die Wahl des Kontrastpols läßt weitere Schlüsse über das Weltbild der Vp zu.

die Vp angeben, inwiefern jedes Konstrukt oder dessen Kontrastpol auf jede Person zutrifft, so daß eine vollständig ausgefüllte Matrix entsteht.

Ziel der Analyse ist es stets, die Ähnlichkeit bestimmter Punkte zueinander festzustellen. In gewissen Fällen kann es z.B. dazu kommen, daß sich Konstrukte in gegenüberliegenden Gruppen häufen und somit "Schwarz-Weiß-Pole" des Weltbilds der Vp darstellen. Solche Extremkonstellationen können Hinweise auf psychische Störungen geben.

Die Analyse der Grids kann auf unterschiedlichste Arten erfolgen, für deren Diskussion auf [Rae93] verwiesen wird. Ein Hauptverfahren zur Darstellung des Zusammenhangs zwischen Elementen und Konstrukten ist die *Hauptkomponentenanalyse* (Principal Components Analysis, PCA). Der Inhalt der Matrix wird dabei als Vektorfeld interpretiert und somit auf einen n-dimensionalen Vektorraum (n>2) projiziert. Die diesen Raum aufspannenden Vektoren werden als Hauptachsen bezeichnet. In diesem Raum dicht beieinander liegende Punkte besitzen eine hohe Ähnlichkeit, weit entfernte Punkte sind je nach Lage bezüglich des betrachteten Punkts entweder invariant (rechtwinklige Lage) oder entgegengesetzt (nahezu 180 Grad Opposition).

Um eine Ähnlichkeitsanalyse so einfach wie möglich durchführen zu können, wären die folgenden Hilfsmittel sehr geeignet:

- Eine Darstellung der Ortsvektoren zweier Punkte zwecks Winkelmessung.
- Eine Markierung von Gruppen ähnlicher Punkte zur besseren Hervorhebung (z.B. durch Farbe oder Form)
- Ein automatisches Auffinden der naheliegendsten x Punkte.
- Eine Summenvektorbildung von ähnlichen Punkten zur Reduzierung der Anzahl von Elementen und Konstrukten.
- Eine Kontrastierung zweier Elemente, bei der die den Unterschied ausmachenden Konstrukte hervorgehoben werden.

Die größte Hürde vor der Anwendung solcher Hilfsmittel stellt die geeignete Darstellung des mehrdimensionalen Raums dar, so daß sich einige Verfahren (z.B. *Biplot*) auf eine zweidimensionale Darstellung beschränken, die dadurch natürlich an Aussagekraft nicht unerheblich einbüßen.

3 Dynamische Statistiken

Dynamische Statistiken[3] erleichtern die Auswertung von drei- oder höherdimensionalen statistischen Daten. Sie unterstützen den Analysator beim Identifizieren und Hervorheben bestimmter Daten, so daß Trends, Gruppen, Ausreißer und andere Strukturmerkmale leichter und schneller erkennbar werden als mit herkömmlichen Methoden.

Bei statischen nicht-graphischen Methoden wie z.B. Tabellen sind zwar alle Daten gleichzeitig sichtbar, nicht aber "erkennbar". Dies liegt an der abstrakten numerischen Darstellung, die das Gehirn nicht so intuitiv verarbeiten kann wie ein Bild.

	X	Y
Ich	-1.998	3.606
Vater	2.861	-0.020
Mutter	0.722	0.449
IdArzt	5.556	0.787
Freund	-0.139	-4.104
Therap	0.499	1.899
Ideal	5.556	0.787
IchAndere	-1.348	2.466
kompromißbereit	2.657	3.086
von.sich.ueberzeugt	-2.263	-3.509
arrogant	-2.377	-1.751
viel.Menschenkenntnis	3.259	1.175
stur	-3.710	1.605
locker	2.669	-2.025
ausgeglichen	3.520	-0.609
unsicher	-3.845	2.764
engstirnig	-2.657	-3.086
kann.Fehler.zugeben	2.263	3.509
nicht.ueberheblich	2.377	1.751
wenig.Menschenkenntnis	-3.259	-1.175
freundlich	3.710	-1.605
gehemmt	-2.669	2.025
unausgeglichen	-3.520	0.609
kann.auf.Leute.zugehen	3.845	-2.764

Abbildung 3-1: Tabelle

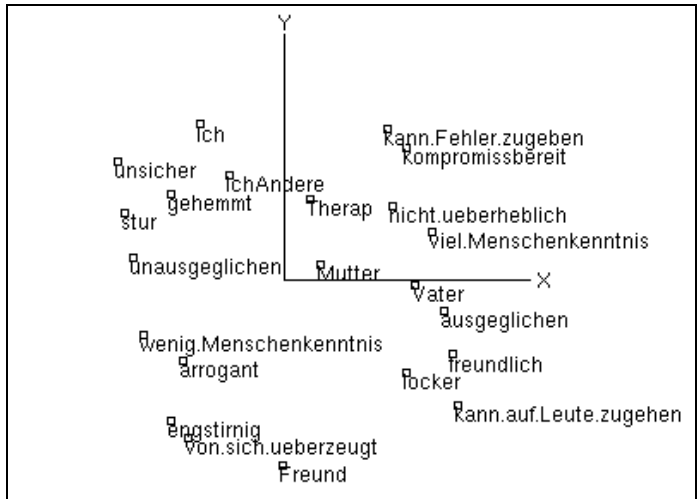


Abbildung 3-2: Diagramm (Daten wie in Abb. 3-1).

Die räumliche Anordnung der Daten läßt sich auf einen Blick feststellen. Extremwerte (z.B. "Freund mit minimalem Y-Wert) und Gruppen (z.B. "engstirnig" und "von sich überzeugt") sind intuitiv erkennbar. Diese Informationen lassen sich auch aus der Tabelle gewinnen, sind dort aber aufgrund der räumlichen Anordnung der Werte "zwischen den Zahlen versteckt". Die Daten stehen dort anstelle der Informationen im Vordergrund.

Bei statischen graphischen Methoden (z.B. Diagrammen) muß sich der Analysator vor Beginn der Auswertung für eine ihm interessant erscheinende Kombination von zwei Variablen entscheiden, die er dann untersucht, um dann je nach Ergebnis eine weitere Kombination zu betrachten. Um eine abschließende Aussage treffen zu können, muß er also eine größere Menge von zweidimensionalen Darstellungen zunächst auswerten und dann vergleichen. Dementgegen erlauben dynamische Methoden es dem Analysator, die Gesamtheit der Daten auf einen Blick zu erfassen und in Echtzeit zu manipulieren.

Die folgenden Tätigkeiten bilden den Hauptteil der Arbeit eines Analysators[DonDonGas88]:

---

[3] Die Bezeichnung in der englischsprachigen Literatur lautet "dynamic graphics for statistics". Der Autor zieht "dynamische Statistiken" der länglichen direkten Übersetzung vor, um die Lesbarkeit zu erhöhen. Nichtsdestotrotz sollte im Auge behalten werden, daß nicht die statistischen Daten selbst, sondern deren Darstellung dynamisch ist.

- **Das Auffinden von Ausreißern**, d.h. von Elementen, die sich weit außerhalb der Bereiche befinden, in denen der Großteil der Elemente zu finden ist. Ausreißer können von Meßfehlern oder Irrtümern bei der Zusammenstellung der Datenmenge herrühren. In seltenen Fällen können sie auch tatsächlich Bestandteile der gültigen Datenmenge sein und bilden dann herausragend wichtige Punkte derselben.
- **Die Markierung von Untermengen.** Untermengen vereinen Elemente, deren Werte bezüglich einer Variable identisch sind, bzw. in einem bestimmten Bereich liegen. Durch diese Markierung läßt sich herausfinden, ob eine Abhängigkeit zwischen bestimmten Variablen existiert, z.B. ob alle Elemente, bei denen A kleiner als fünf ist auch einen C-Wert kleiner als fünf haben.

Daraus ergeben sich die folgenden Forderungen, die an dynamische Statistiken gestellt werden[BecCleWil88]:

- **Skalierung.** Um bestimmte Eigenschaften der Daten hervorzuheben, soll die Größe der Darstellung sowie das Verhältnis von Variablen zueinander (drei X-Achsen-Längeneinheiten entsprechen einer Y-Achsen-Längeneinheit) modifizierbar sein. Starke Unterschiede in der Skalierung heben Extremwerte hervor.
- **Identifikation.** Wenn der Analysator ein Element mit der Maus auswählt, soll dessen Bezeichnung (*Label*) erscheinen.
- **Lokalisierung.** Wenn der Analysator die Bezeichnung eines Elements eingibt, soll es in der Darstellung hervorgehoben werden.
- **Löschen.** Wenn ein Ausreißer die Darstellung unnötigerweise zusammenquetscht, soll er durch den Analysator löschbar sein, so daß die gesamte Darstellung günstiger skaliert werden kann.
- **In Relation zueinander bringen.** Es soll für den Analysator erkennbar sein, ob eine Abhängigkeit zwischen Variablen besteht.

Es gibt drei hauptsächlich verfolgte Ansätze zur Darstellung der höheren Dimensionen bei dynamischen Statistiken, die jeweils mit bestimmten Vor- und Nachteilen behaftet sind:

Alternagraphics

Alternagraphics bestehen aus einer Matrix von Graphen, in denen je die Abhängigkeiten zwischen zwei Variablen dargestellt werden. Die restlichen Variablen der Elemente werden vernachlässigt. Auf der Hauptdiagonalen werden die Variablenbezeichnungen sowie deren Maximalwerte aufgetragen. Redundanz entsteht dadurch, daß die Graphen in "linken unteren" Dreiecksmatrix "rechts oben" gespiegelt noch einmal auftauchen. Für n Variablen ergibt sich also eine $n_x n$ Matrix.

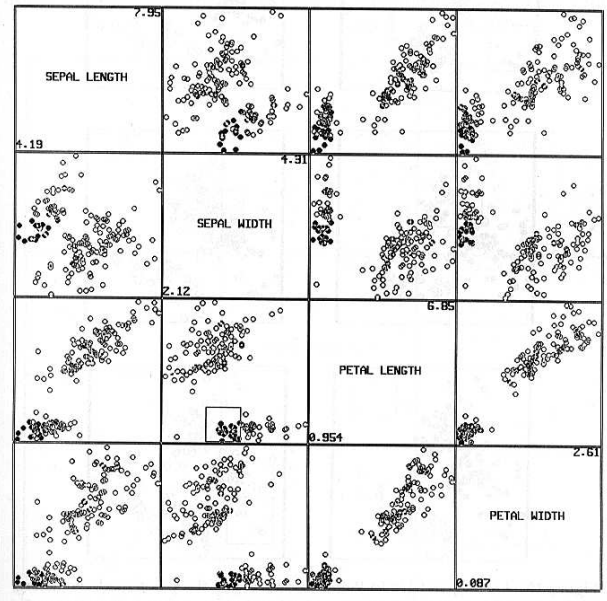


Abbildung 3-3: Alternagraphics mit Brush in (2,3)

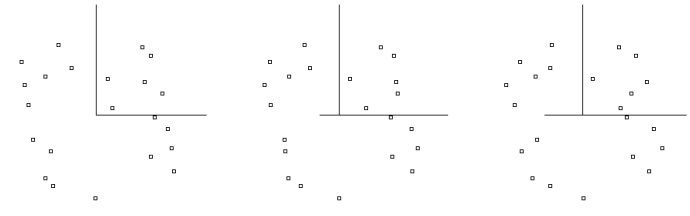
In allen Darstellungen liegen die in (2,3) hervorgehobenen Elemente (schwarz markiert) in einem eng umgrenzten Bereich. Es besteht also eine Abhängigkeit bezüglich der Abmessungen von Kelchblättern (sepals) und Blüttenblättern (petals)

Quelle: [BecCleWil88, S.15]

Selektion von Elementen erfolgt mit einem Rechteck, das als *Brush* bezeichnet wird und dessen Größe variierbar ist. Mit diesem Brush fährt der Analysator über einen Graphen und je nach gewähltem Modus werden die Elemente innerhalb des Rechtecks permanent oder transient einer bestimmten Operation unterzogen. Dies kann Markieren, Löschen oder Identifizieren sein. In Relation gebracht werden Elemente in verschiedenen Ansichten dadurch, daß Operationen, die im Graph mit dem Brush ausgeführt werden, sich sofort auf alle anderen Graphen auswirken.

Rotation

Durch eine schnelle Abfolge von Bildern auf dem Bildschirm aus leicht unterschiedlichen Perspektiven läßt sich der Eindruck räumlicher Tiefe hervorrufen, obwohl die Bilder auf ein nur zweidimensionales Medium projiziert werden. Der Analysator sieht auf seinem Bildschirm ein dreiachsiges Koordinatensystem mit den Elementen als Punktwolken darin. Der dreidimensionale Effekt entsteht nur bei Rotation. Da diese Wirkung sich mit bildlichen Mitteln im Rahmen dieser Studienarbeit nicht darstellen läßt, wird dringend empfohlen, sich die Rotation mit dem in Kapitel 5 beschriebenen Werkzeug zu veranschaulichen.

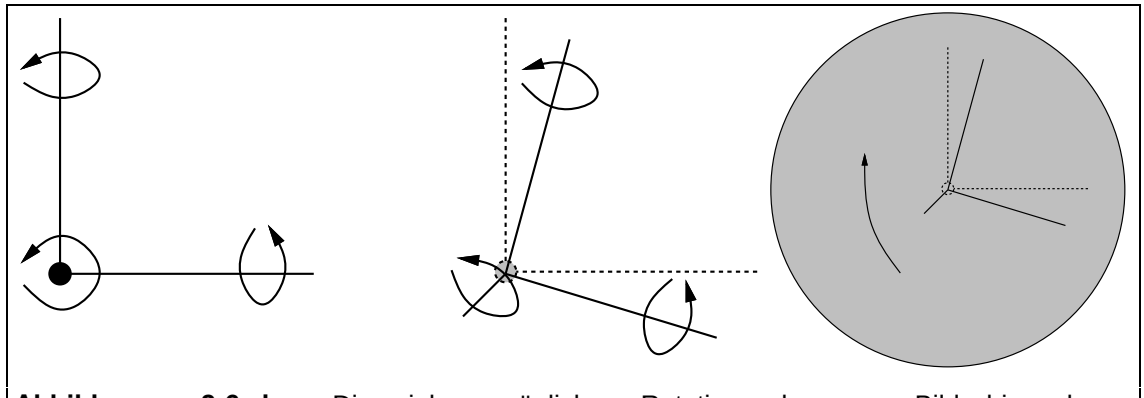


Abbildungsfolge 3-5a,b,c: Rotation um die Y-Achse.

Der dreidimensionale Effekt stellt sich ein, wenn die Bilder in schneller Folge auf dem Bildschirm dargestellt werden.

Ohne Bewegung können die Daten bearbeitet werden wie bei Alternagraphics, d.h. sie können mit einem Brush permanent oder transient selektiert, markiert, identifiziert oder gelöscht werden. In Relation werden die Elemente durch die Rotation gebracht, bei der ersichtlich wird, ob die, anhand von zwei Dimensionen markierten, Elemente auch bezüglich ihrer dritten Dimension eine Gruppe bilden.

Die Rotation kann wahlweise um eine der eigentlichen Datenachsen, eine der Bildschirmachsen oder eine frei vom Analysator gewählte Achse erfolgen, die er z.B. mit einer Mausbewegung festgelegt hat. Die letztgenannte Methode ermöglicht die intuitivste Steuerung des Graphen, da man sich den Graph leicht als einen Ball vorstellen kann, der in der Mitte fixiert ist, und den man durch Ziehen mit der Maus in Rotation versetzen kann wie einen Globus mit einer frei wählbaren Achse.



Abbildungen 3-6a,b,c: Die sieben möglichen Rotationsachsen. **a:** Bildschirmachsen, **b:** Datenachsen, **c:** frei gewählte Achse

Bei mehr als drei Dimensionen ist Rotation nur dann einsetzbar, wenn die fehlenden Dimensionen auf eine andere Weise angedeutet werden können, z.B. durch Farben. Die hierfür gewählten Unterscheidungsmerkmale kommen dann allerdings nicht mehr für die Markierung von Gruppen in Betracht.

Stereoskopische Betrachtung

Eine Verbesserung der Rotationsmethode kann erreicht werden, wenn stereoskopische Sehhilfen verwendet werden, um den dreidimensionalen Eindruck der Daten auch im Stillstand aufrecht zu erhalten [BecCleWei88]. Dies kann entweder mit einer Rot-Grün-Brille oder einer Brille, deren Gläser synchron zum Bildschirm abwechselnd verdunkelt werden, um nur das "linke" oder "rechte" Bild aufzunehmen, geschehen. Die zweite Methode ist zwar aufwendiger, hat aber den Vorteil, daß das volle Farbspektrum verwendet werden kann.

Ein Vergleich zwischen Alternagraphics und den Rotationsmethoden ergibt, daß Alternagraphics bei mehr als drei Dimensionen übersichtlicher sind, dafür aber die Rotationsmethoden optisch leichter erfaßbar sind. Der Nachteil der nicht-stereoskopischen Rotationsmethode, daß Rotation gleichzeitig dazu benutzt wird, die Betrachtungsposition zu verändern *und* die dritte Dimension entstehen zu lassen, muß in Kauf genommen werden, wenn aufwendige stereoskopische Sehhilfen nicht zur Verfügung stehen.

In Kapitel 5 wird die nichtstereoskopische Rotationsmethode verwendet, da bei der Kelly-Grid-Analyse zunächst nur dreisimensionale Daten anfallen werden. Dort werden aus dem im vorigen Kapitel geschilderten Kelly-Grid-Verfahren und der in diesem Kapitel beschriebenen Rotationsmethode die Anforderungen an ein Kelly-Grid-Analyse-Werkzeug auf Rotationsbasis zusammengetragen und anhand dieser das Werkzeug entwickelt. Die Entwicklung erfolgt objektorientiert gemäß des Leitbildes von Werkzeug und Material, das im folgenden Kapitel 4 vorgestellt wird.

4 Objektorientierte Entwicklung mit der Werkzeug-Material-Metapher

Dieses Kapitel beschreibt den Entwurf von objektorientierten Systemen nach der Werkzeug-Material-Metapher. Zunächst wird Objektorientierung an sich vorgestellt, die dem Entwickler aber kein Vorgehensmodell zum Entwurf an die Hand gibt. Dies leistet die im Anschluß beschriebene Werkzeug-Material-Metapher.

Objektorientierung

Objektorientierte Entwurfstechniken und Programmiersprachen erlauben es dem Entwickler, fachliche Systeme "natürlich" zu modellieren, ohne Phänomene der Anwendungswelt "künstlich" in Daten und Funktionen trennen zu müssen. Objekten der Realität bezüglich der Anwendung werden *Objekte* im Modell gegenübergestellt, die in den für das Modell relevanten Eigenschaften mit denen der Realität übereinstimmen. Die Eigenschaften lassen sich in Daten (*Exemplarvariablen*) und Funktionen (*Methoden*) unterteilen, wobei die Exemplarvariablen nur über die Methoden zugreifbar sind (*Kapselung*). Objekte mit ähnlichen Ausprägungen lassen sich zu *Klassen* zusammenfassen. Klassen dienen als Muster zur Erzeugung von Objekten, die alle eine identische Struktur (Methoden und Exemplarvariablen), aber einen anderen Zustand (*Variablenbelegung*) haben. Das Verhältnis von Klassen untereinander kann durch *Vererbungsbeziehungen* beschrieben werden. Spezialisierung, d.h. die Hinzunahme bzw. Änderung von Eigenschaften erfolgt durch die Bildung von *Unterklassen*, Verallgemeinerung, d.h. Weglassen von speziellen Eigenschaften, durch *Oberklassen*. Die Isolierung von Klassen und der Aufbau einer Klassenhierarchie sind Hauptaufgaben der objektorientierten Modellierung.

Die Werkzeug-Material-Metapher

Die Art der Modellierung kann durch die Entwurfs- oder Programmiersprache selbst nicht vorgegeben werden. Dazu bedarf es eines bestimmten Leitbilds, das je nach dem Anwendungskontext gewählt werden muß. Ein sehr mächtiges Leitbild der objektorientierten ist das von Werkzeugen und Materialien, das von Züllighoven [KilGryZül93] und anderen entwickelt worden ist. Sie ist auf interaktive Ein-Benutzer-Bürosysteme zugeschnitten und propagiert die Modellierung der dort vorhandenen Objekte entweder als *Materialien* oder als *Werkzeuge*. Materialien verhalten sich passiv und können von Werkzeugen bearbeitet werden, wobei durchaus keine 1:1-Beziehung gegeben sein muß. Ein Material kann ebenso von mehreren Werkzeugen bearbeitet werden, wie ein Werkzeug mehrere Materialien bearbeiten kann. Jedes Material wird von verschiedenen Werkzeugen unter einem anderen *Aspekt* bearbeitet. Für ein Betrachtungswerkzeug ist z.B. der Inhalt eines Dokuments ausschlaggebend, während für ein Ordnerwerkzeug nur der Titel von Interesse sein dürfte.

Die Werkzeug-Material-Metapher unterstützt die Entwicklung in in allen Projektstadien gleichermaßen. In den folgenden Unterkapiteln werden die Vorgehensweisen und Dokumente beschrieben, die zur Analyse des Anwendungsbereichs, zum Entwurf und zur Implementation des Systems verwendet werden. Diese Trennung bedeutet jedoch nicht, daß diese Stadien linear wie im Wasserfallmodell durchlaufen werden. Der

Entwicklungsprozeß ist vielmehr evolutionär und alle Dokumente können zu jedem Zeitpunkt geändert werden, um z.B. Änderungen im Anwendungsbereich gerecht zu werden. Die Unterteilung dient lediglich der Betonung der Hauptaktivitäten in den jeweiligen Projektstadien.

Aus der Beschreibung der Metapher bezüglich der verschiedenen Stadien geht eine ihrer wesentlichen Stärken hervor: Sie ist nicht nur isoliert für die Analyse *oder* den Entwurf *oder* die Implementation anwendbar, sondern erlaubt die bruchlose Modellierung eines Anwendungsbereichs durch Werkzeuge und Materialien, die *sowohl* eine fachlich *als auch* eine technische Bedeutung haben. Damit setzt sie sich von anderen weniger weitgreifenden Methoden ab.

4.1 Anforderungsermittlung

Die Analyse des Anwendungsbereichs erfolgt mit dem zukünftigen Benutzer des zu erstellenden Systems möglichst an seinem derzeitigen Arbeitsplatz. Da Benutzer und Entwickler aus verschiedenen Umfeldern kommen, muß eine einheitliche Projektsprache entwickelt werden, in der Fachbegriffe aus beiden Welten für beide verständlich verwendet werden. Diese Begriffe werden in einem *Glossar* zusammengefaßt, das für den gesamten Projektverlauf als Grundlage und Nachschlagewerk dient. Das Glossar entsteht nicht isoliert sondern während der Erfassung von für den Benuzter typischen Arbeitsabläufen in *Szenarios*. Aus ihnen wird der übliche Umgang mit den am Arbeitsplatz vorhandenen Gegenständen ersichtlich, wobei sich oft bereits eine Unterteilung in Materialien und Werkzeuge erkennen läßt.

4.2 Entwurf

Auf der Grundlage der Glossare und der Szenarios können nun mit Hilfe von *Systemvisionen* Arbeitssituationen mit dem zukünftigen System beschrieben werden. Systemvisionen sind imaginäre Szenarios. Sie enthalten bereits eine Liste von möglichen Softwarewerkzeugen und Materialien, die zur Ausführung der geplanten Tätigkeiten notwendig sind. Bildschirmskizzen dieser Softwarewerkzeuge sollten den Systemvisionen hinzugefügt werden.

Durch die Herauslösung der Werkzeuge ergibt sich eine Unterteilung des Systems in Werzeuge, Materialen und Aspekte. Diese werden im folgenden charakterisiert.

Materialien werden niemals von sich aus aktiv, sondern können nur durch sondierende oder manipulierende Methoden von einem Werkzeug aus unter einen bestimmten Aspekt bearbeitet werden. Materialien dürfen keinerlei Annahmen über ihre zukünftige Verwendung treffen, da dies eine zu große Einschränkung für noch zu entwickelnde Werkzeuge darstellt. Auf Materialien kann niemals direkt zugegriffen werden, obwohl durch ein sehr gut gestaltetes Werkzeug dieser Eindruck entstehen kann.

Werkzeuge erlauben dem Benutzer die aktive Bearbeitung von Materialien. Sie bieten ihm stets einen Blick auf das Material an sich und dessen momentanen Zustand. Zusätzlich werden die Bearbeitungsmöglichkeiten

angezeigt. Ein Werkzeug kann für Routineaufgaben eigene Zustände besitzen, die aber für den Benutzer nicht unsichtbar sein dürfen.

Die Zuordnung von Methoden zu Werkzeugen oder Materialien ist nicht immer einfach und hängt von den jeweiligen Erfordernissen ab. So könnte sich z.B. bei der Modellierung eines Ordners die Frage ergeben, ob die Methode zur Bereitstellung des Inhaltsverzeichnisses dem Material (dem Ordner) oder einem Werkzeug (z.B. einem Ordnerbrowser) zugeordnet werden soll. Als Leitlinie kann hierbei gelten, daß Zustandsänderungen, die von jedem bearbeitenden Werkzeug vorgenommen werden müssen, dem Material zuzuordnen sind und solche, die nur von einem bestimmten Werkzeug ausgeführt werden müssen, bei dem jeweiligen Werkzeug belassen werden sollten. Im Falle des Ordners hieße dies wahrscheinlich, das Inhaltsverzeichnis dem Ordner selbst beizulegen, denn die meisten Werkzeuge würden sich vermutlich mit dessen Inhalt auseinandersetzen. Letztendlich ist die Entscheidung also keine technische, sondern hängt von den konkreten Umgangsformen ab.

Wiederverwendung von möglichst großen Teilen eines Systems ist eine Hauptmotivation für die Entwicklung objektorientierter Sprachen gewesen. Auch innerhalb der Werkzeug-Material-Metapher nimmt sie einen wesentlichen Platz ein. Werkzeuge sollten so allgemein gehalten werden, daß sie sich zum alleinigen Einsatz wie auch zum Einsatz als *Subwerkzeug* eines übergeordneten Werkzeugs eignen. Ein Notizeditor sollte beispielsweise sowohl isoliert als auch als Subwerkzeug eines Hypertextannotationswerkzeugs verwendbar sein.

Während Materialien weitestgehend plattformunabhängig sind, da ihre Darstellung ja von den Werkzeugen übernommen wird, so sind letztere dadurch eng an eine bestimmte Oberflächen- und Handhabungsnorm gebunden. Um die eigentliche Funktionalität des Werkzeugs von der jeweiligen Systemumgebung zu entkoppeln, werden Werkzeuge in *Funktions-* und *Interaktionskomponente* unterteilt. Die Funktionskomponente (FK) verkörpert die eigentlichen Eigenschaften des Werkzeugs mit dem Aufruf der sondierenden und manipulierenden Materialmethoden. Die Interaktionskomponente (IAK) regelt, wie der Benutzer diese Eigenschaften nutzen kann.

Der Vorteil dieser Trennung liegt in der Tatsache, daß im Falle der Verwendung des Werkzeugs in einem anderen Umfeld lediglich die Interaktionskomponente ausgewechselt werden muß. So kann es beispielsweise Interaktionskomponenten für Textterminals, verschiedene Fenstersysteme und verschiedene Handhabungsformen geben. Die Interaktionskomponente übernimmt die Darstellung jedoch nicht selbst, sondern delegiert sie an *Interaktionstypen* (IATs). Ein Interaktionstyp ist die Abstraktion einer elementaren einheitlichen Interaktionsform mit einem Softwarewerkzeug. Dies ist z.B. die Auswahl eines Punktes aus einer Liste von Optionen, das Ausfüllen eines Formulars, die Eingabe eines Textes oder das Öffnen einer Datei. Diese Abläufe sind unabhängig von der jeweiligen Präsentationsform immer identisch. Im Falle einer Portierung sind also bei gleichbleibender Handhabung nur die Interaktionstypen anzupassen.

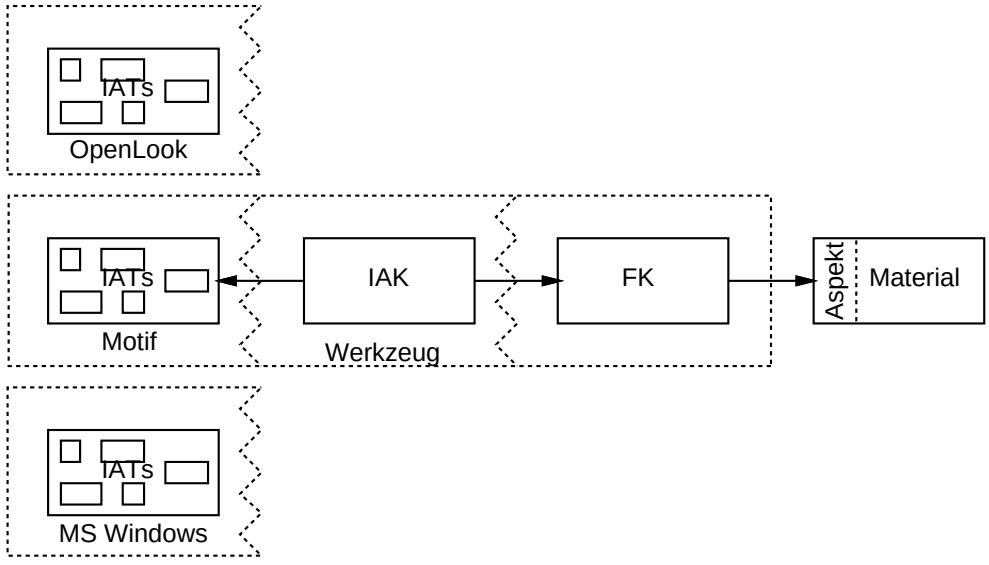


Abbildung 4.2-1 Werkzeugstruktur mit Auswechselungsmöglichkeiten am Beispiel der IATs[4]

Interaktionstypen sind Subwerkzeugen recht ähnlich. Der Unterschied besteht darin, ob zur Ausführung Kontextwissen nötig ist oder nicht. Im ersten Fall handelt es sich um ein Subwerkzeug, im zweiten um einen IAT. Die Eingabe eines Datums als Interaktionsform kann auf beiderlei Art durchgeführt werden. Mit einem Texteingabe-IAT, der sein Ergebnis dem Werkzeug übergibt, welches es auf Plausibilität überprüft, oder mit einem Subwerkzeug, das über Eingabezeilen- Bestätigungs- und Abbruch-IAT verfügt und eine eigenständige Überprüfung vornimmt, bevor das Ergebnis an das übergeordnete Werkzeug weitergereicht wird.

[4] Graphische Konventionen: Siehe Anhang

Die Kommunikation zwischen den vier Systemkomponenten IAT, IAK, FK und Material erfolgt über einen Beobachtermechanismus. Er trennt Beobachtete, die jede eigene Veränderung an einen übergeordneten Beobachter melden, von dem sie nur die Methode zur Ablieferung ihrer Meldungen kennen, und Beobachtende, die die komplette Methoden-Schnittstelle der von ihnen beobachteten Objekte kennen. Mit Hilfe diese Konzepts ist eine größtmögliche Entkoppelung der einzelnen Komponenten möglich, da von den Beobachteten keine Annahmen über ihre Beobachter getroffen werden können. Zusätzliche Attraktivität gewinnt es dadurch, daß ein Beobachteter gleichzeitig Beobachter weiterer untergeordneter Objekte sein kann, deren Meldungen er zunächst selbst prüft, bevor er sie ggf. nach oben weitermeldet. Im Rahmen der Werkzeug-Material-Metapher ist der oberste Beobachter eines Werkzeugs die Haupt-IAK. Sie empfängt Meldungen von den IATs, Sub-IAKs und der FK, die wiederum von Sub-FKs und den Materialien benachrichtigt wird.

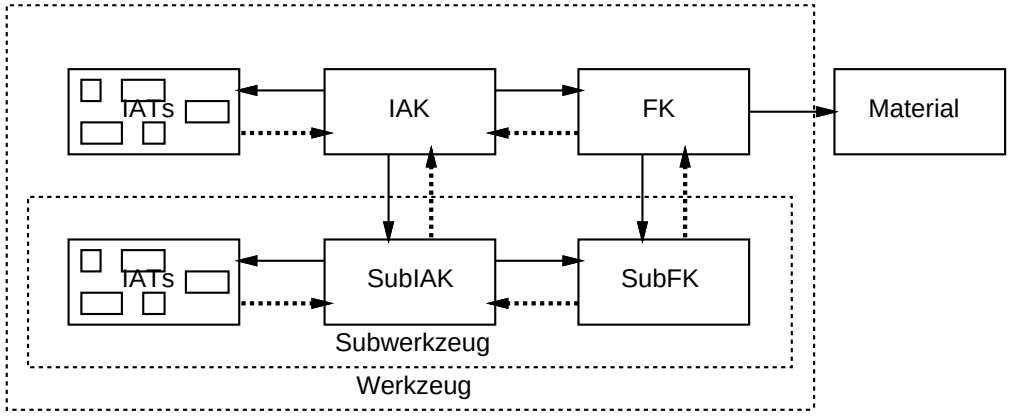


Abbildung 4.2-2: Werkzeug-Meldewege

4.3 Implementation

Für die Implementation stehen am Arbeitsbereich Softwaretechnik die FIAK- und die IATMotif-Bibliothek zur Verfügung, die im wesentlichen von Dirk Riehle entwickelt worden sind. Beiden gemein ist die Verwendung des Beobachterkonzepts, dessen Implementation zunächst erläutert werden soll.

Die Klasse `tNotifier`[5] stellt die abstrakte Oberklasse für Beobachter dar. "Abstrakt" bedeutet, das mindestens eine Methode von der erbenden Klasse, die den Beobachtermechanismus verwenden möchte, redefiniert werden muß. Die wesentlichen Methoden sind `Notify( tEvent, tNotifier* )`, in der Meldungen in Form von Ereignissen (`tEvents`) von beobachteten Klassen (`tNotifier*`) ausgewertet werden, und `Tell( tEvent )`, das ein Ereignis an alle übergeordneten Beobachter schickt. Während sich hinter `Tell()` lediglich der Aufruf von `Notify()` der beobachtenden Klasse verbirgt (`void Tell( tEvent ) { pNotifier->Notify( tEvent, this ); }`), handelt es sich bei `Notify()` um ein großes `switch()`, bei dem je nach Ereignis und Meldendem entweder eigene Maßnahmen getroffen werden oder die Meldung an eventuell vorhandene übergeordnete Beobachter weitergegeben wird. Sämtliche IAT-, IAK- und FK-Klassen besitzen `tNotifier` als gemeinsame Oberklasse.

[5] Namenskonventionen: Siehe Anhang

4.3.1 Die FIAK-Bibliothek

Die FIAK-Bibliothek [Rie93c] stellt einen Anwendungsrahmen für Werkzeuge zur Verfügung. Dies geschieht auf mehreren aufeinander aufbauenden Ebenen, die, von oben[6] beginnend, je nach Werkzeugtyp benutzt werden können oder nicht.

Die oberste Ebene besteht aus den Klassen `tIAKBase` und `tFKBase`. Ihre alleinige Funktionalität besteht darin, neue Subwerkzeuge geordnet zu erzeugen. Dazu sendet ein Objekt der Klasse `tFKBase` das Ereignis `cEventFKInit` an seine IAK, die daraufhin ihre Methode `CreateSubIAK()` aufruft. Des weiteren wird in diesen Klassen zum ersten Mal ein doppelter Konstruktorbaum eingeführt. Dies erscheint sinnvoll, da beim Aufruf des Standardkonstruktors das System noch nicht komplett funktionstüchtig ist, und somit nur eingeschränkt Anweisungen zur Initialisierung ausgeführt werden können. Während im Standardkonstruktor, von der höchsten Oberklasse beginnend, nur die eigenen Exemplarvariablen auf `NULL` initialisiert werden, so erfolgen die komplexeren Operationen im darauffolgenden `Init()`-Baum, der ebenfalls von der Wurzel bis zur auslösenden untersten Klasse durchlaufen wird. `Init()` ruft also zunächst die `Init()`-Methode seiner direkten Oberklasse auf, bevor es sich an die eigenen Variablen macht.

Die Klassen `tIAKMain` und `tFKMain` stellen eine Spezialisierung von `tIAKBase` und `tFKBase` dar. Sie bilden einen Rahmen für Werkzeuge, die eine Menüleiste mit Datei-, Editier- und Hilfemenü besitzen. Dazu bedient sich die IAK einer Sub-IAK namens `tIAKMenus`, die die einzelnen Menüs verwaltet. Die FK besitzt die Methode `DoMenuCommand( long, tString )`, die von der IAK benutzt wird, um Methoden der FK zentral anzusprechen, Methoden zum Laden und Speichern von Zeichenketten, und einen einfachen Datenverwalter (`tIsManager`), der das bearbeitete Material der FK in Zeichenkettenform zur Verfügung stellt. Der Datenverwalter besitzt Methoden zum Einfügen, Entfernen und Abholen des Materials sowie einen Status, der über den Änderungsstand des Materials Auskunft gibt.

Eine typische Meldefolge sieht wie folgt aus: Die Menü-IAK meldet die Auswahl eines Menüpunkts an die Haupt-IAK, die daraufhin mit Hilfe von `DoMenuCommand()` eine entsprechende Methode der HauptFK aufruft, die sich gegebenenfalls einer Methode des Datenverwalters bedient. Die FK meldet je nach Ergebnis eventuell ein Ereignis an die IAK (z.B. "Materialzustand hat sich verändert"), woraufhin diese erneut die FK aufruft, um sich mit einer aktuellen Materialrepräsentation zu versorgen, die sie dann mit Hilfe von IATs darstellt. Hervorzuheben ist hierbei, daß die FK nicht davon ausgehen darf, daß die IAK nach dem Melden des Ereignisses "Material hat sich verändert"(`cEventFKUpdate`) auch tatsächlich eine neue Repräsentation anfordert. Ereignisse dienen ***nur*** der Information, nicht der Übermittlung von Anforderungen. Diese Verknüpfung würde die weitere Verwendung der Klasse über Gebühr stark einschränken und die Befehlshierarchie aufweichen.

Über die beschriebenen Komponenten hinaus gibt es noch eine Toolbox-IAK (`tIAKToolbox`), die eine Werkzeugleiste als Subwerkzeug einer Haupt-IAK darstellt und eine Klasse `tEnvironment`, die die Schnittstelle zum System darstellt, in die das Werkzeug eingebettet ist. Für die nähere Erläuterung dieser

---

[6] Entgegen der sprachlichen Konvention, die grundlegendes mit unten assoziiert, wird hier analog zur objektorientierten Diktion verfahren, in der grundlegende Oberklassen oben stehen.

Klassen wird auf [Rie93a] verwiesen, da sie keine konzeptionellen Erweiterungen beinhalten. Gemein ist ihnen und `tIAKMain`, `tIAKMenus`, `tFKMain` und `tIsManager` jedoch, daß zur Erweiterung lediglich neue Unterklassen gebildet werden müssen, die erweiterte Eigenschaften aufweisen und die sich in die `Init()`-Konstruktorhierarchie einklinken.

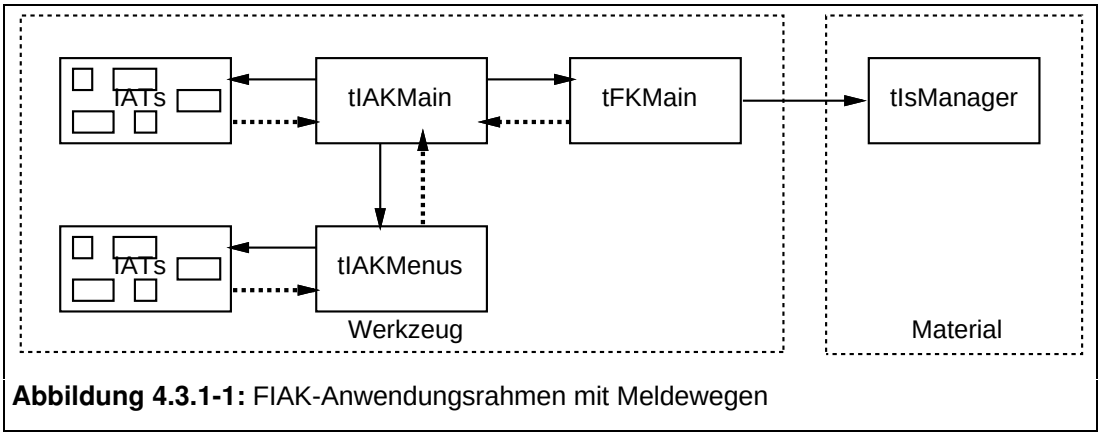


Abbildung 4.3.1-1: FIAK-Anwendungsrahmen mit Meldewegen

4.3.2 Die IATMotif-Bibliothek

Die IATMotif-Bibliothek[Rie93a] bietet eine Sammlung von IAT-Implementationen für die Verwendung unter dem Fenstersystems XWindows. Die gemeinsame abstrakte Oberklasse aller IATs ist `tIATBase`, die von `tNotifier` (und `tStatus`) erbt und die Methoden `Show()` und `Hide()` besitzt, um einen IAT anzuzeigen bzw. nicht darzustellen. Jeder IAT definiert einen Satz von spezifischen Ereignissen, die er bei Bedarf (z.B. Auswahl durch den Benutzer) an die beobachtende IAK meldet. Die eigentliche Systemanbindung erfolgt über den `tIATMeta`, der XWindows-Ereignisse abfängt und sie an die zuständigen IATs weiterleitet. Die Arbeit des Meta-IATs wird von der Klasse `tEnvironment` der FIAK-Bibliothek gestartet und beendet.

IATs können und müssen geschachtelt werden, um eine sinnvolle Bildschirmdarstellung zu erreichen. So gibt es neben IATs zur Darstellung auch reine Verwaltungs-IATs, die die Anordnung untergeordneter IATs regeln. IATGruppen umfassen Fenster-IATs, Menü-IATs und Einfach-IATs wie Knöpfe, Eingabezeilen und Textfelder. Weiterhin gibt es komplexe IATs wie Knopfsammlungen, Listenbrowser und Dateilade- und Speicherrequester. Verwaltungs-IATs sind Bretter, Rahmen, Gitter und Popups, in denen untergeordnete IATs angebracht werden können. Der `tIATCustom` stellt eine Zeichenfläche zur Verfügung, auf dem Objekte aus der Hierarchie der graphischen Objekte dargestellt werden können. Diese Hierarchie, die aus `tDrawables` und `tDraggables` besteht umfaßt Linien, Ellipsen, Rechtecke und vieles mehr.

Während mit Hilfe von IATs eine Hierarchie derselben erzeugt werden kann, so bedarf es noch der Spezifikation in einer eigenen Ressourcendatei, die von XWindows bei der Programmausführung interpretiert wird, um das exakte Erscheinungsbild auf dem Bildschirm festzulegen. Auf diese Weise sind zum Beispiel die Größe, Nachbarschaftsverhältnis, Beschriftung und Farbe der IATs manipulierbar, die für die Funktionalität natürlich irrelevant sind.

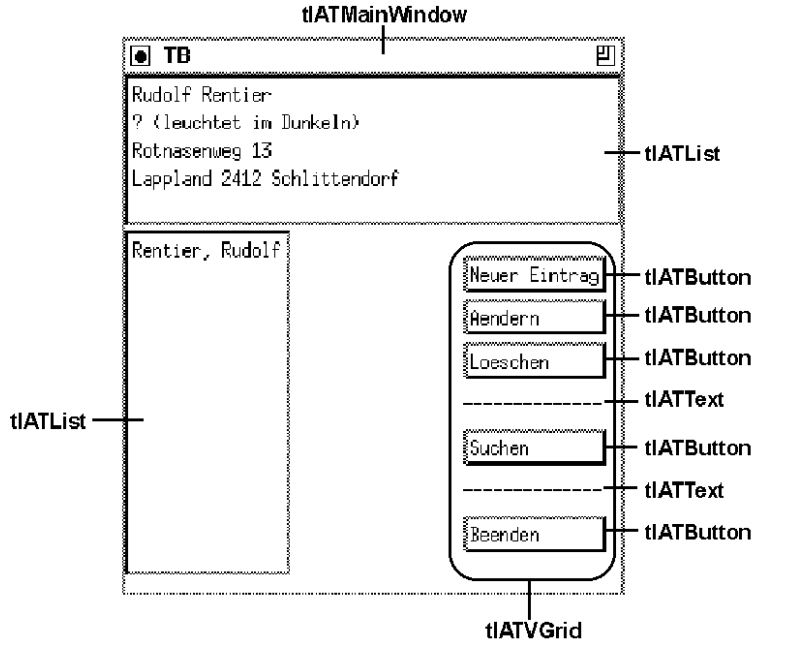


Abbildung 4.3.2-1: Typisches Werkzeug (privates Telefonbuch) mit markierten IATs

5 Entwicklung des Interaktionstyps

Nachdem in den Kapiteln 2 bis 4 die theoretischen Grundlagen für die Entwicklung eines Kelly-Grid-Analyse-Werkzeugs gelegt worden sind, geht es in diesem Kapitel um die konkrete Erstellung dieses Werkzeugs. In Kapitel 2 wurden Kelly-Grids als Hilfsmittel zur Analyse der persönlichen Konstrukte von Versuchspersonen eingeführt. Die anfallenden Analysedaten bilden eine dreidimensionale Matrix. Kapitel 3 schilderte dynamische Statistiken als Möglichkeit zur dreidimensionalen, interaktiven Betrachtung von Statistiken unter Ausnutzung eines Rotationsverfahrens. Eine Anwendung von dynamischen Statistiken auf Kelly-Grid-Analysedaten in Form eines Kelly-Grid-Analyse-Werkzeugs auf der Basis von dynamischen Statistiken mittels Rotation erschien wünschenswert. Kapitel 4 beschrieb die Werkzeug-Material-Metapher, die es unter Verwendung des Leitbilds von Werkzeugen und Materialien ermöglicht, Anwendungssysteme bruchlos fachlich und technisch evolutionär zu entwickeln. Anhand dieser Metapher wird in diesem Kapitel das in Kapitel 3 geforderte Werkzeug entwickelt. Die Entwicklung erfolgt unter der Annahme, daß das Werkzeug in Form eines Interaktionstyps (IATSpin) in die IATMotif-Bibliothek eingegliedert werden wird.

Die Unterkapitel befassen sich analog zu Kapitel 4 mit den Hauptaktivitäten in den Projektstadien, in denen die Analyse des Anwendungsbereichs, der Klassenentwurf und die Implementation im Vordergrund stehen. Wie bereits im Vorwort erwähnt, erfolgte die Entwicklung aufgrund kommunikativer Engpässe entgegen dem in Kapitel 4 geforderten evolutionären Vorgehen eher linear mit Zyklen nur im technischen Bereich. Das letzte Unterkapitel enthält eine Betrachtung des entwickelten Werkzeugs und gliedert die nicht erreichten Ziele gemäß Ursachen in der IATMotif-Bibliothek, mangelnder Zeit und noch unvollständiger Analyse.

5.1 Anforderungen

Die Anforderungen an das zu erstellende Werkzeug lassen sich in einen fachlichen und in einen organisatorischen Teil gliedern. Im fachlichen Teil geht es um die konkrete Auskleidung der Forderungen aus den Kapiteln 2 und 3, im organisatorischen Teil um deren Einbettung in die vorhandene Systemumgebung.

Fachliche Anforderungen

Da die in Kapitel 2 genannten Anforderungen an ein vollständiges Kelly-Grid-Analyse-Werkzeug den Rahmen einer Studienarbeit zu sprengen drohen, beschränkt sich der Autor auf ein allgemeines Dynamische-Statistik-Werkzeug, das jedoch einen Grundstock für ein spezielles Grid-Analyse-Werkzeug darstellt und im Rahmen z.B. einer weiteren Studienarbeit leicht dazu ausgebaut werden könnte.

Die in den Kapiteln 2 und 3 geschilderten Anforderungen dienen bereits als Szenarios und sollen hier nur stichwortartig wiederholt werden. Für dynamische Statistiken werden die folgenden Funktionen gefordert:

- Skalierung der Darstellung, um bestimmte Eigenschaften der Daten hervorzuheben.
- Identifikation von Datenelementen durch Anzeige des Labels bei Anwahl mit der Maus.
- Lokalisierung von Elementen anhand ihres Namens.
- Rotation, um Abhängigkeiten zwischen Variablen zu erkennen.

Löschen von Elementen ist nicht gefordert, da nach der Transformation der Kelly-Grid-Daten keine Ausreißer zu erwarten sind.

Über diese allgemeinen Anforderungen hinaus sind folgende Funktionen für eine Kelly-Grid-Analyse sinnvoll, die in Gesprächen mit dem zukünftigen Benutzer ermittelt wurden:

- Zusätzliche textuelle Listen, anhand derer Datenelemente und Gruppen ausgewählt werden können.
- Hilfskonstrukte, um den dreidimensionalen Eindruck zu verstärken.
- Zwischenspeicherung von Rotationsbildern, um einen direkten Vergleich zu ermöglichen.
- Eine einfache Methode, um die Darstellung von ausgewählten Datenmengen zu verändern.
- Linien als zusätzliche Datenelemente, um die Ähnlichkeit bestimmter Datenpunkte zu verdeutlichen.

Bis auf den letzten Punkt wären alle zusätzlichen Forderungen auch für den allgemeinen Fall interessant.

Wie bei jedem Werkzeug sollte es ebenfalls Funktionen zum

- Laden des Materials
- Speichern des Materials
- Anzeigen von kontextsensitiven Hilfen
- Verlassen des Werkzeugs

geben.

Je nach Wichtigkeit und Einordnung in den Analyseablauf werden diese Punkte nun bestimmten Oberflächenkomponenten des zukünftigen Werkzeugs zugeordnet. Oberflächenkomponenten können Menüs, Auswahlleisten und Requester sein. Jede von ihnen hat Stärken und Schwächen:

- Auswahlleisten (Toolboxen) sind mit Piktogrammen versehen und wirken in großen Mengen unübersichtlich. Aktionen können nicht parametrisiert werden. Sie können schnell angewählt werden.
- Menüpunkte können ausführlich beschriftet sein und sind somit kaum doppeldeutig. Ihre Anwahl nimmt Zeit zum Navigieren innerhalb der Menühierarchie in Anspruch.
- Requester erlauben eine qualifizierte Spezifikation von Aktionen und folgen der Auswahl von Menüpunkten. Sie unterbrechen die direkte Manipulation des Materials.

Gemäß der Bearbeitungsreihenfolge bei der Analyse (Identifikation → Selektion → Manipulation) ergeben sich bestimmte Anforderungsprofile an die einzelnen Bearbeitungsschritte:

- Identifikation erfolgt durch Rotation und Anwahl mit der Maus. Nicht auf jede Identifikation folgt notwendigerweise eine Selektion.

  Da dieser Schritt der häufigste ist, sollten die Funktionen, die zu seiner Durchführung notwendig sind, einer Auswahlleiste zugeordnet werden.

- Selektion kann nach räumlichen und logischen Gesichtspunkten erfolgen. Erstere kann mit der Maus in der Statistik erfolgen, letztere bedarf einer qualifizierten Spezifikation.

  Räumliche Selektion sollte direkt über eine Auswahlleiste (Selektionsmodus ein/aus), logische Selektion (anhand des Namens) über Menüs und/oder Requester erfolgen.

- Manipulation verändert mindestens einen Parameter der Daten. Dieser muß geeignet spezifiziert werden.

  Bei nur einem Parameter genügt eine (möglichst textuelle) Auswahlleiste oder ein Menüpunkt. Bei zwei oder mehr Parametern ist ein Requester erforderlich

Die Benutzungsoberfläche des Werkzeugs sollte daher wie folgt aussehen:

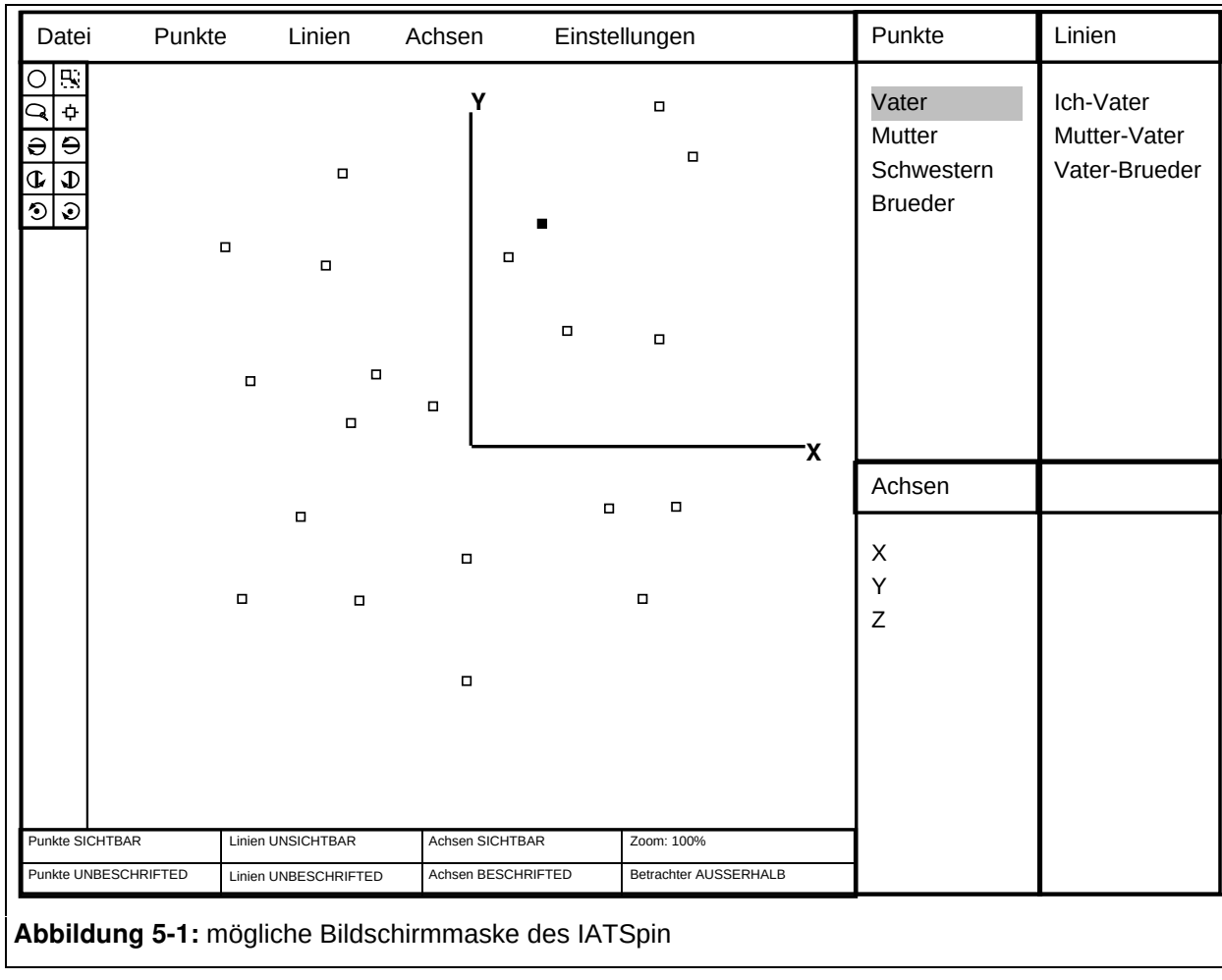


Abbildung 5-1: mögliche Bildschirmmaske des IATSpin

- Ein Hauptfenster, in dem die Statistik in ihrem momentanen Zustand sichtbar ist und manipuliert werden kann.
- Eine Auswahlleiste zur Identifikation und Selektion von Datenelementen.
- Eine Auswahlleiste zur einparametrigen Manipulation der Statistik, die gleichzeitig als Zustandsanzeiger dient.

- Eine Menüleiste mit Menüs für das Laden und Speichern des Materials sowie das Beenden des Werkzeugs, für die Selektion von Datenelementen anhand des Namens, für die Manipulation der Daten und für die Änderung der Werkzeugeinstellungen.

Organisatorische Anforderungen

Das soeben entworfene Werkzeug soll als Interaktionstyp für dynamische Statistiken (IATSpin) in die IATMotif-Bibliothek eingebunden werden und somit einen Baustein für komplexere Werkzeuge bilden.

Die darzustellenden Daten sollen von einem ebenfalls noch zu entwickelnden Kelly-Grid-Eingabewerkzeug [Fah94] geliefert werden.

Wie bereits in der Einleitung zu Kapitel 5 angedeutet wurde, konnten weder alle fachlichen noch alle organisatorischen Anforderungen erfüllt werden. Die vielschichtige Ursachen werden in Kapitel 5.4 ausführlich geschildert. Trotzalledem wurde das meiste unverändert realisiert, so daß der prinzipielle Erfolg bezüglich der Erstellung des Werkzeugs außer Frage steht.

5.2 Entwurf

Die Architektur des IATSpin baut auf dem in Kapitel 4.3.1 geschilderten Anwendungsrahmen der FIAK-Bibliothek auf. Die dort enthaltenen Klassen werden gemäß den Anforderungen spezialisiert. Die IAK erhält entsprechend der Systemvision eine Menüleiste, eine Auswahlleiste für Aktionen und eine kombinierte Anzeige- und Auswahlleiste für den Werkzeug- und Materialzustand. Zur Eingabe und Auswahl von Parametern zur Modifikation des Materials dienen Requester. Die FK wird um die eigentliche Werkzeugfunktionalität erweitert. Das Material erbt von `tIsManager` und enthält die zu bearbeitende dynamische Statistik in der Form von Punkten, Linien und Achsen. Zur Gewährleistung der Systemunabhängigkeit des Werkzeugs sind die Motif-spezifischen[7] Eigenschaften von FK und Material in Aspektklassen gekapselt.

[7] Motif ist eine mögliche Oberflächengestaltungsart unter XWindows.

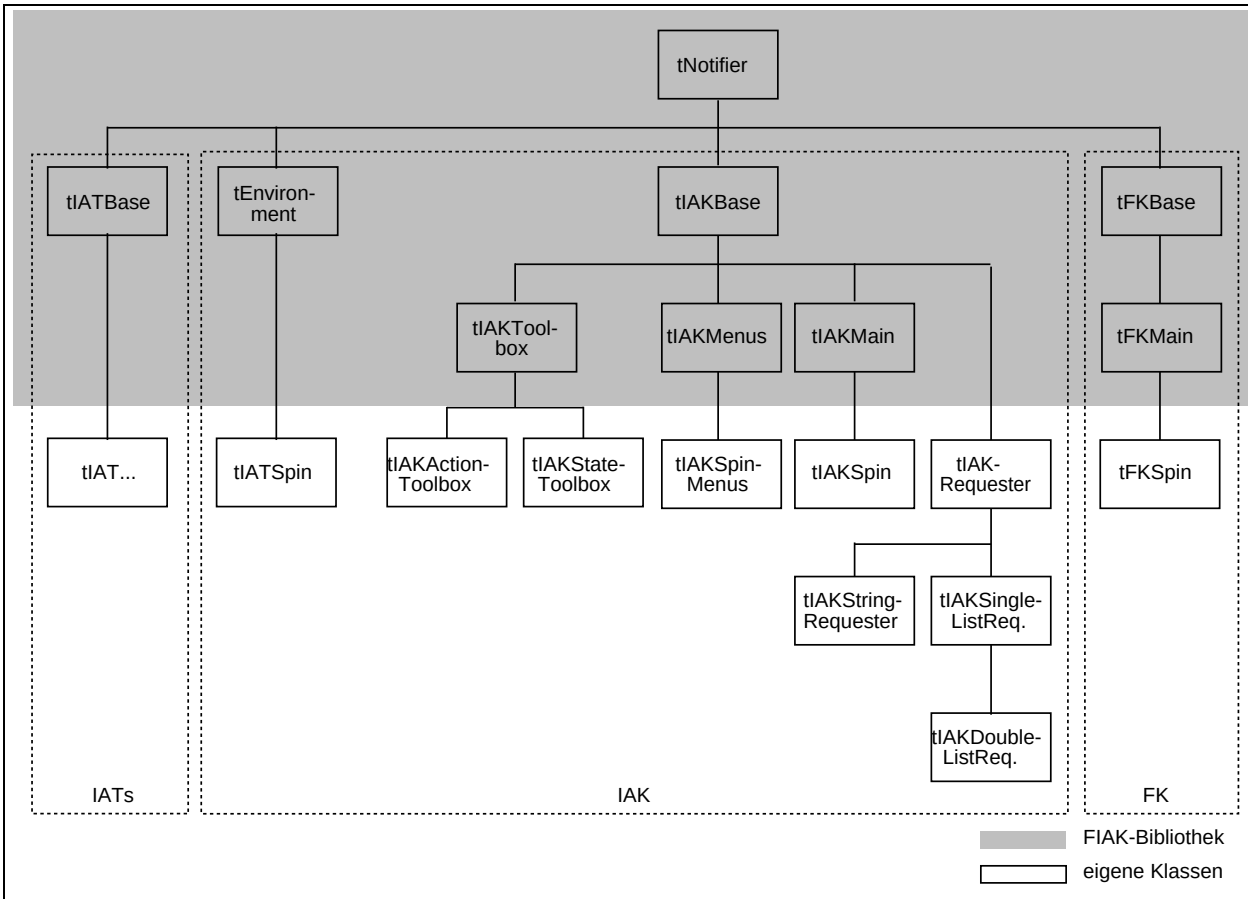


Abbildung 5.2-1: Statische Architektur des IATSpin (ohne Material und Aspekte)

Schnittstellen

Die Entwurfsdetails der einzelnen Komponenten werden in den folgenden Unterkapiteln beschrieben. Es gibt jedoch komponentenübergreifende Gesichtspunkte bezüglich der Schnittstellen, die hier kurz diskutiert werden sollen.

Der Aufruf von FK-Methoden von der IAK aus erfolgt, wie bereits erwähnt, hauptsächlich über die Methode `DoMenuCommand()`. Die Schnittstelle dieser Methode besitzt nur ein, die gewählte Aktion spezifizierendes, Kommando (`long`) und eine Zeichenkette zur Übermittlung von Daten. Da aber auch komplexe Kommandos zu realisieren sind (z.B. "Alle selektierten Punkte blau färben"), bedarf es eines Hilfsmittels, mit dem komplexe Parameter auf IAK-Seite eindeutig auf eine Zeichenkette abgebildet und auf FK-Seite wiedergewonnen werden können. Dies leistet die Klasse `tEncoder`. Sie wird von IAK und FK benutzt, um ihre Kommandos zu ver- bzw. entschlüsseln.

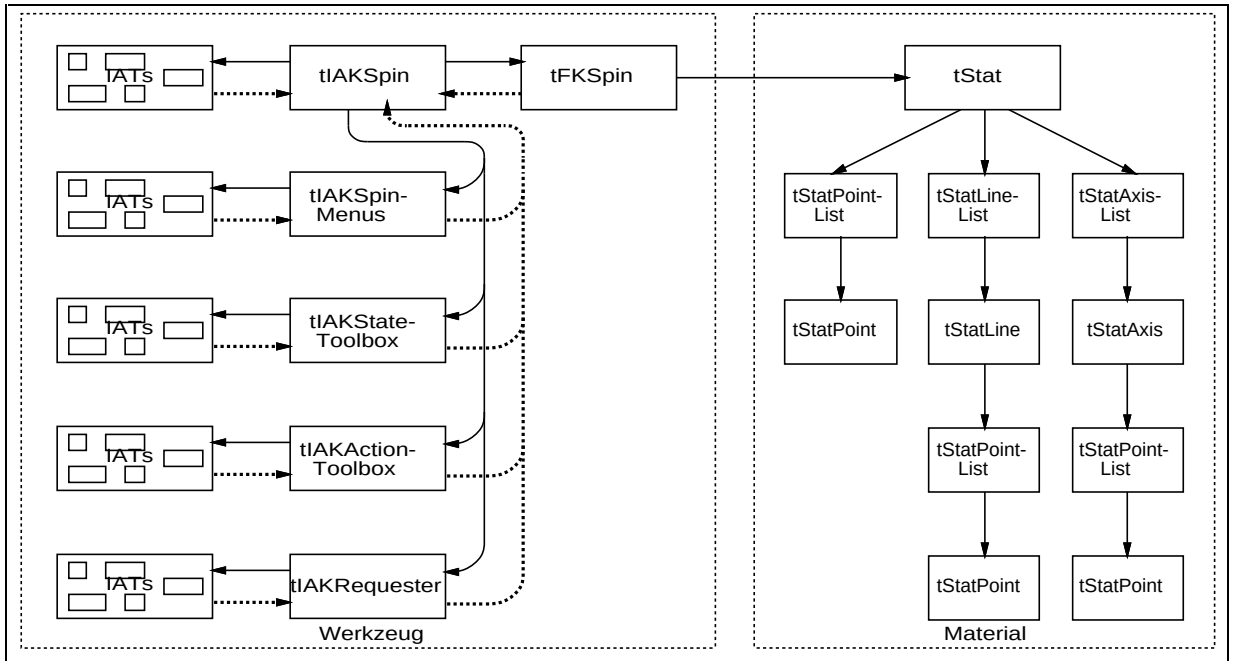


Abbildung 5.2-1: Dynamische Architektur des IATSpin mit Material und Meldewegen

Die Trennung von FK und Material ist im Rahmen dieser Arbeit nicht-trivial. Wie auch beim Modellieren eines Ordners und eines ordnerbearbeitendem Werkzeugs stellt sich die Frage, ob z.B. das Inhaltsverzeichnis vom Ordner selbst oder vom Werkzeug bereitgestellt werden soll. Die Entscheidung wurde gemäß den in [GryZül94] angegebenen Kriterien wie folgt getroffen:

- Das aktuelle Element des Materials wird vom Werkzeug verwaltet.
- Eigenschaften wie Selektionsgenauigkeit und Rotationsgeschwindigkeit sind Werkzeugeinstellungen.
- Methoden zum unqualifizierten[8] Traversieren des Materials à la `getMore()` gehören zum Material.
- Methoden zum qualifizierten[9] Traversieren und Manipulieren des Materials sind Teile des Werkzeugs.
- Grundlegende Eigenschaften dynamischer Statistiken wie Größe, Zoomfaktor und Rotationszustand sind Bestandteile des Materials.

5.2.1 Material

Allgemeines

Entscheidend für die Struktur des Materials ist die Realisierung der Rotation. Sie kann entweder als Methode des jeweiligen Materialbestandteils[10] (Punkt, Linie, Achse) oder als Methode des Materials (dynamische

[8] unter "unqualifiziert" versteht der Autor eine Vorgehensweise, bei der der Benutzer keinen Einfluß auf die Eigenschaften des ihm gelieferten Materialbestandteils hat. Alle Materialbestandteile werden als homogen angesehen.

[9] unter "qualifiziert" versteht der Autor eine Vorgehensweise, bei der der Benutzer Einfluß auf die Eigenschaften des ihm gelieferten Materialbestandteils nehmen kann. Die Materialbestandteile erscheinen also nicht als homogene Masse.

[10] Die Teile, aus denen sich eine Statistik zusammensetzt, werden von nun an "Bestandteile" genannt. Das kürzere "Elemente" wurde wegen Doppelbelegung in der Anwendungswelt ("Elemente" und "Konstrukte") nicht gewählt.

Statistik) angesehen werden. Je nach Entscheidung ergeben sich unterschiedliche Klassenstrukturen. Im ersten Fall müßte jeder Bestandteil Rotationsmittelpunkt und -winkel erfahren, um die Rotation ausführen zu können. Im zweiten Fall entfiele die Rotationsmethode der Bestandteile, da die Rotation zentral von der Statistik ausgeführt würde. Im Sinne einer natürlichen Modellierung wurde die zweite Variante gewählt, da "rotiere Dich" im Gegensatz zu Methoden wie "färbe Dich" und "mache Dich unsichtbar" keine eigentliche Eigenschaft z.B. eines Punktes ist, sondern nur im Kontext mit einer Statistik sinnvoll ist. Da die Bestandteile jedoch möglichst wenig über ihren Verwendungskontext wissen sollen, wurde die Rotationsmethode in die eigentliche Statistik verlagert.

Jeder Bestandteil erhält drei Koordinatensätze, um Rotation und Darstellung zu realisieren. Ein dreidimensionaler Satz für die ursprüngliche Position, der für Abfragen des Benutzers und zum Wiederherstellen des Urzustandes nach längeren Sitzungen dient. Ein weiterer dreidimensionaler Satz für die aktuelle Position, die durch mehrere Rotationen aus der ursprünglichen Position hervorgegangen ist. Abschließend ein zweidimensionaler Satz für die Darstellungsposition auf dem Bildschirm, der sich direkt aus der aktuellen Position berechnen läßt. Natürlich reichten auch nur die ursprünglichen Koordinaten aus, wenn man bereit wäre, für jede erneute Darstellung sämtliche Positionen neu zu berechnen. Da dies aber bei großen Statistiken ineffizient erscheint, wurde der beschriebene Weg gewählt.

Die dynamische Statistik besteht also aus Punkten (`tStatPoint`), Linien (`tStatLine`) und Achsen (`tStatAxis`). Diese besitzen einen Bezeichner und Koordinatensätze, über die sie eindeutig zugreifbar sind. Darüber hinaus hat jeder Bestandteil eine gewisse Darstellungsgröße (dünn, normal, fett) und eine Farbe. Er selbst oder sein Namensschild (Label) können sichtbar oder unsichtbar sowie selektiert oder nicht selektiert sein. Linien können eine bestimmte Strichlierung besitzen. Achsen können sichtbar oder unsichtbar skaliert sein.

Aufgrund der teilweise überlappenden Eigenschaften bietet sich die Schaffung einer gemeinsamen Oberklasse `tStatObject` an, die Bezeichner, Größe, Farbe, Sichtbarkeit, Sichtbarkeit des Labels und Selektionszustand enthält. Die Koordinaten werden dort nicht abgelegt, da es sich bei Punkten um Einzelkoordinaten, bei Linien und Achsen aber um Listen von Koordinaten handelt. Da diese aber einheitlich handhabbar sein sollen, wird in `tStatObject` bereits ein Satz von Traversierungsfunktionen abstrakt deklariert. Da das tatsächliche Löschen von Elementen aus der Statistik nicht vorgesehen ist, reichen die folgenden Methoden, die das unidirektionale Traversieren ermöglichen, aus:

```
void goToFirstStatPoint( void );
void goToNextStatPoint( void );
BOOL isLastStatPoint( void );
tStatPoint *getStatPoint( void );
```

Bei Linien und Achsen werden sich dahinter tatsächliche Listenoperationen verbergen, während bei Punkten eine Liste nur simuliert werden wird.

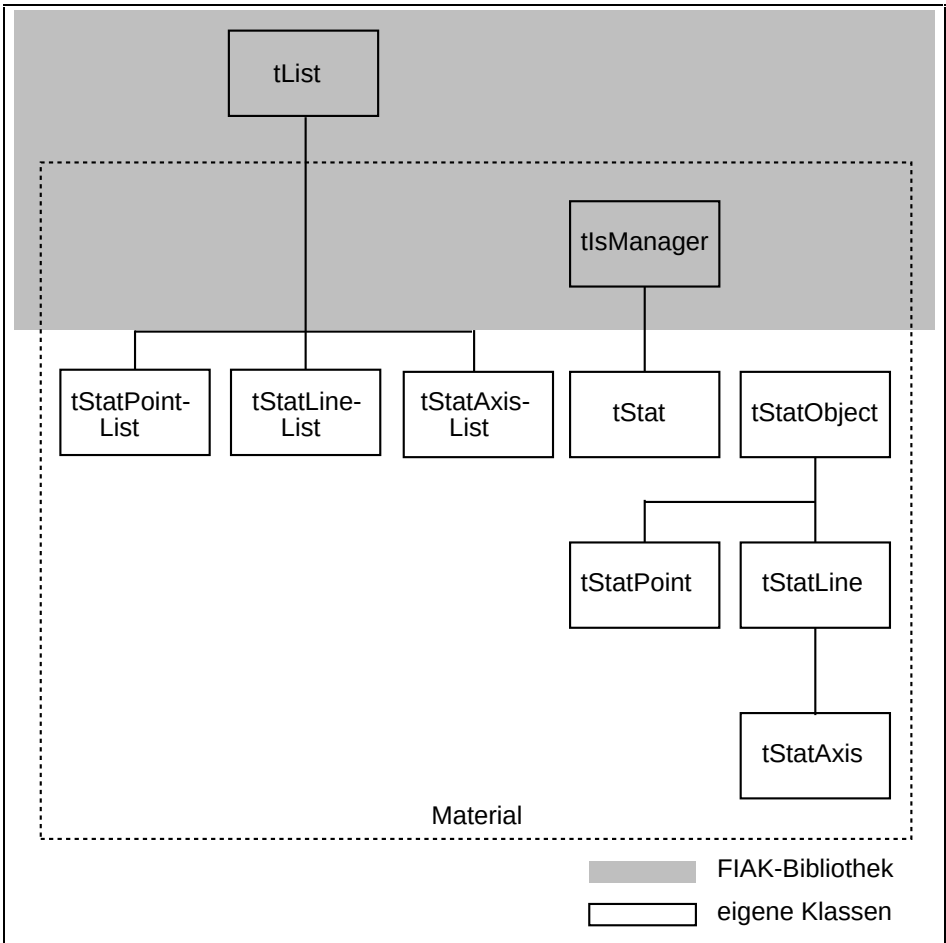


Abbildung 5.2.1-1: Statische Architektur des Materials. Die dynamische Architektur ist in Abbildung 5.2-1 dargestellt.

Punkte werden mit der Klasse `tStatPoint` modelliert, die von `tStatObject` erbt und zusätzlich die Eigenschaft einer Punktform besitzt. Anhand dieser Punktform (z.B. Kreis, Rechteck, Sternchen) können disjunkte Gruppen von Punkten (z.B. Elemente und Konstrukte) unterschieden werden. Die Traversierungsfunktionen simulieren lediglich eine Liste von Punkten, indem nach `goToNextStatPoint()` `isLastStatPoint()` nur noch `TRUE` zurückliefert, woraufhin jede listenabfragende Schleife abbricht. Nur durch Rücksetzen mit `goToFirstStatPoint()` kann der Ausgangszustand wieder erreicht werden.

Linien (`tStatLine`) erben ebenfalls von `tStatObject` und enthalten eine Liste von `tStatPoints` zur Festlegung der sie beschreibenden Raumkurve. Von den enthaltenen `tStatPoints` werden nur die Koordinaten, nicht aber die restlichen Eigenschaften genutzt. Die Punkte können mit den beschriebenen Methoden traversiert werden, um z.B. die Koordinaten neu zu bestimmen. Linien besitzen eine Strichlierungsart (durchgezogen, strichliert, punktiert) und eine Linienbreite (dünn, mittelbreit, breit).

Achsen (`tStatAxis`) sind spezielle Linien und erben demzufolge von `tStatLine`. Einschränkend sind allerdings nur zwei formbestimmende Punkte erlaubt, nämlich der Beginn im Ursprung und das Achsenende. Achsen besitzen eine Skalierung (linear oder dekadisch logarithmisch), die sichtbar oder unsichtbar sein kann. Je nach Skalierungsart werden Punkte und Linien entlang einer Achse unterschiedlich auf den Bildschirm projiziert.

Die dynamische Statistik (`tStat`) ist eine Spezialisierung des Datenverwalters `tIsManager` aus der FIAK-Bibliothek. Sie besitzt demzufolge Methoden zum Einfügen und Auslesen des Materials in Zeichenkettenform, sowie eine Methode zum Löschen desselben. Die Methoden zum Anzeigen von Änderungen des Materials spielen keine Rolle, da zum momentanen Zeitpunkt lediglich die Darstellung des Materials, nicht aber das Material an sich (die ursprünglichen Koordinatensätze) vom Benutzer geändert wird.

Die Statistik enthält ihre Bestandteile in Form von Listen, die nach dem Muster von `tStatObjects` traversiert werden können.

`tStat` besitzt sondierende Methoden, die Auskunft über die Ausdehnung der enthaltenen Daten geben. So können die minimalen (am weitesten hinten links unten) und maximalen (am weitesten vorne rechts oben) Punkte und Linien in ursprünglichen und aktuellen Koordinaten, sowie der maximale Abstand vom Ursprung ermittelt werden, was für benutzende Werkzeuge jeder Art für die optimale (d.h. die Fläche am besten ausnutzende) Darstellung sehr nützlich ist.

Die modifizierenden Methoden beziehen sich ausschließlich auf das gesamte Material und erlauben eine Änderung der Betrachtungsart durch Rotation, Zoom und Wechsel der Position des fiktiven Beobachters. Speziellere Methoden befinden sich in der FK des Werkzeugs.

5.2.2 Werkzeug (FK)

Die FK des Werkzeugs erbt von der Klasse `tFKMain` aus der FIAK-Bibliothek und enthält somit Standardmethoden zum Laden und Speichern des Materials. Da `tStat` eine Unterklasse des Datenverwalters `tIsManager` ist, müssen diese Methoden nicht redefiniert werden. Die FK enthält des weiteren einen Zeiger auf den aktuell ausgewählten Bestandteil (*Cursor*) sowie Variablen, die die momentanen Werkzeugeinstellungen (Rotationsgeschwindigkeit und Selektionsgenauigkeit) enthalten. Die FK-Methoden lassen sich gemäß des Szenarios aus Kapitel 3 in vier Gruppen unterteilen, die im folgenden detailliert beschrieben werden. Es sind Methoden zur Lokalisierung der gewünschten Daten, deren Selektion und anschließende Manipulation sowie einige Werkzeugeinstellungsmethoden.

Zuvor muß jedoch eine zentrale Methode angesprochen werden, derer sich fast alle selektierenden und manipulierende Methoden bedienen.

Diese Methode heißt `search()` und ermöglicht das Auffinden von Bestandteilen der Statistik nach exakt vorgegebenen Kriterien. Sie liefert anderen Methoden geeignetes Material, das diese dann je nach Aufgabe weiter bearbeiten können. Die Steuerung erfolgt über zwei Parameter, durch die eine Vorauswahl bezüglich eines bestimmten Selektionszustandes und einer bestimmten Objektart durchgeführt wird. Ein dritter Parameter gibt an, ob die Suche von vorne beginnen oder bei dem aktuell ausgewählten Objekt fortgesetzt werden soll.

```
BOOL search( int selectionRange = SR_ALL,
             int objectRange = OR_ANY,
             int searchMode = SM_NEW );
```

In der obigen Standardkonfiguration liefert `search()` den ersten Bestandteil des Materials (Punkt, Linie oder Achse). Durch Wechsel des dritten Parameters auf `SM_CONTINUE` kann so die gesamte Statistik traversiert werden. Das gleiche kann auch für z.B. "alle Punkte" oder "alle selektierten Bestandteile" erreicht werden, wenn die ersten beiden Parameter dementsprechend modifiziert werden. `search()` setzt im Erfolgsfall den Cursor um und liefert `TRUE` zurück. Andernfalls bleibt der Cursor unverändert und der Rückgabewert ist `FALSE`. Eine Entkoppelung von `search()` und einer sondierenden Methode zur Feststellung des Erfolgs (z.B. "`BOOL newObjectFound()`") wäre zwar möglich gewesen, aufgrund der engen Koppelung aber unnötig umständlich, denn nach jedem Aufruf von `search()` müßte natürlich überprüft werden, ob das Ergebnis denn überhaupt gültig ist.

Die Methoden zur Lokalisierung von bestimmten Bestandteilen der Statistik werden 1:1 auf die entsprechenden Methoden des Materials abgebildet, nämlich Rotation, Zoom und Wechsel der Position des fiktiven Beobachters. Zur Erleichterung des Auffindens kann auch die Ausdehnung der Statistik verändert (vor allem vergrößert) und der Rotationsmittelpunkt so versetzt werden, daß eine aussagekräftigere Rotationsstruktur entsteht.

Die Methoden zur Selektion zerfallen in zwei Teile. Der erste Teil ermöglicht die Selektion über die Eingabe einer Zeichenkette und besteht nur aus der Methode `select()` mit der Signatur `select( tString *name )`. Nach dem Aufruf haben alle Bestandteile mit dem angegebenen Namen den Selektionszustand "selektiert". Der zweite Teil erlaubt die Selektion über eine direkte Angabe der Koordinaten (die später von der IAK geliefert werden sollen, z.B. durch Auswahl mit der Maus). Der Analysator kann einen Punkt einzeln auswählen (`select( tPoint2D displayPosition )`), alle Bestandteile innerhalb eines Rechtecks (`select( tPoint2D displayPositionUpperLeft, displayPositionLowerRight )`) oder innerhalb eines freien Linienzugs (*Lasso*) (`select( tListPoint2D displayPositionLasso)`).

Alle Methoden haben einen vergleichbaren Aufbau und bedienen sich einer speziellen `search()`-Variation, z.B. `searchName()`, `searchDisplayPosition()`, usw. . Diese Variationen verfügen über die in `search()` vorhandenen Parametern hinaus über eine bestimmte gewünschte Materialeigenschaft (z.B. einen Namen) und einen Parameter, der angibt, ob der gefundene Bestandteil auch selektiert werden soll. Die Selektionsmethoden rufen also diese privaten Variationen solange in einer Schleife auf, bis alle in Frage kommenden Bestandteile abgearbeitet sind.

```
void tFKGrid::select( tString *name )
{
 BOOL found;
 BOOL selected = FALSE;

 unselect( SR_ALL );
 found = searchName( name, SR_ALL, OR_POINT, SM_NEW, SC_SELECT );
 selected = found;
 while( found == TRUE )
  {
   found = searchName( name, SR_ALL, OR_POINT, SM_CONTINUE, SC_SELECT
);
  }
 if( selected == TRUE )
  {
   Tell( cEventFKUpdate );
  }
}
```

Die Notwendigkeit zur Einführung dieser `search()`-Variationen als Zwischenstufe zwischen `search()` und `select()` ergibt sich durch denkbare Sonderfälle, bei denen Bestandteile zwar lokalisiert, nicht aber selektiert werden sollen. Der einzige dieser Sonderfälle, der realisiert wurde, ist die Methode `setCursor()`. Mit ihrer Hilfe wird zwar der mittels einer Bildschirmposition spezifizierte Bestandteil zum aktuellen Bestandteil, auf dem der Cursor steht, jedoch nicht selektiert. Der Nutzen dieser Methode besteht darin, daß der spezifizierte Bestandteil zu Darstellungszwecken temporär hervorgehoben werden kann. Der einzige Unterschied zu `select( tPoint2D displayPosition )` besteht darin, daß in der zentralen Schleife `searchDisplayPosition()` nicht mit dem Parameter `SC_SELECT` (SelectionConsequence: select), sondern mit `SC_CURSOR` aufgerufen wird.

Die Methoden zur Manipulation von selektierten Bestandteilen erlauben die Veränderung sämtlicher in Kapitel 5.2.1 geschilderten Eigenschaften der Bestandteile der Statistik, wie Name, Farbe und Sichtbarkeit. Sie sind alle nach dem gleichen Schema aufgebaut, das wiederum dem der `select()`-Methoden sehr ähnlich ist. Mit `search()` werden in einer Schleife alle in Frage kommenden Bestandteile durchlaufen und dann je nach Methode unterschiedlich manipuliert. Die ursprüngliche Verwendung dieser Methoden schloß sich an eine Selektion an, was durch den minimalen Aufruf von z.B. `setDisplayColour( int displayColour )` erreicht wird. Um aber auch Aufrufe ohne vorhergehende Selektion wie z.B. "färbe alle Linien blau" oder "mache alle selektierten Bestandteile unsichtbar" zu ermöglichen, wurde die Signatur um zwei Parameter erweitert, die eine Spezifikation nach dem Selektionszustand und der Bestandteilart erlaubt. Sie sind defaultmäßig auf `SR_SELECTED` (SelectionRange: selected) und `OR_ANY` (ObjectRange: any) eingestellt.

```
void tFKGrid::setDisplayColour( int displayColour,
                                int selectionRange,
                                int objectRange )
{
 BOOL success = TRUE;
 int searchMode = SM_NEW;

 while ( success == TRUE )
  {
   success = search( selectionRange, objectRange, searchMode );
   searchMode = SM_CONTINUE;
   if( success == TRUE )
    {
     pCurrentGridObject->setDisplayColour( displayColour );
    }
  }
}
```

Die Werkzeugeinstellungsmethoden betreffen nicht die Statistik direkt, sondern betreffen die Art, in der sie vom Werkzeug manipuliert wird. Eingestellt werden können die Rotationsgeschwindigkeit und die Selektionsgenauigkeit, d.h. die Marge, um die ein Bestandteil mit seinen Koordinaten von den in einer Selektionsmethode spezifizierten Koordinaten abweichen darf, um trotzdem noch selektiert zu werden. Dies ist notwendig, um Ungenauigkeiten bei der Verwendung von Fließkommazahlen zu vermeiden.

Außerdem kann eine bestimmte Rotationsposition gespeichert werden, um vom Analysator zu einem ihm geeignet scheinenden Zeitpunkt (z.B. zum Vergleich) wieder abgerufen werden zu können.

5.2.3 Motif-Aspekt

Der Entwurf von Material und FK des Werkzeugs ist bisher vollkommen systemunabhängig. Es wurden weder Annahmen über eine bestimmte Handhabungsmetapher noch über eine bestimmte Darstellungsweise getroffen. Beim Entwurf der IAK des Werkzeugs stellen sich nun diese Fragen. Während die IAK eigentlich nur die Handhabung und ihre IATs erst die konkrete Darstellung regeln, so muß doch die IAK schon eine geeignete Repräsentation des Materials besitzen, die sie an ihre IATs weitergeben kann. Da das Material bisher keine Informationen zur Darstellung enthält, ergeben sich zwei Möglichkeiten, der IAK diese zur Verfügung zu stellen:

- Die IAK ermittelt mit sondierenden Methoden die Struktur des Materials und baut eine für sie geeignete Parallelstruktur auf. Bei jeder Aktion des Benutzers muß dann allerdings dessen Auswahl aus der Parallelstruktur in die tatsächliche Materialstruktur rückübersetzt werden, damit der FK eindeutige Anweisungen übermittelt werden können.
- Das Material erhält eine zusätzliche Eigenschaft, die ihm die Verwaltung einer Darstellung für eine bestimmte Oberfläche erlaubt, ohne daß sich die sonstige Schnittstelle ändert. Die FK reicht die Darstellung des Materials ohne jegliche Modifikation auf Wunsch an die IAK weiter, die diese wiederum an ihre IATs weiterleitet. Es entsteht keine Dualität.

Um bei der zweiten Methode, die im weiteren verfolgt werden wird, das Material nicht unnötig aufzublähen, bedient man sich eines Aspekts, der die Darstellung von Materialien unter Motif regelt. Dadurch muß nicht jedes Material die Darstellungen für alle denkbaren Oberflächen verwalten, sondern je nach Bedarf werden Aspektklassen für das jeweils benutzte Fenstersystem entworfen. Diese Verwendung des Aspekts weicht etwas von der in Kapitel 4 geschilderten Verwendung ab, bei der ein Aspekt - gleich einem Passepartout - die Klassenschnittstelle einschränkte. Hier dient er als eine Erweiterung. An der grundlegenden Funktion ändert dies aber nichts, da nach wie vor der Aspekt eine spezifische Betrachtung des Materials erlaubt.

Um die Statistik also unter Motif darstellen zu können, muß sie und somit auch alle ihre Bestandteile ein Äquivalent enthalten, das die Darstellungsinformation enthält. Diese parallele Materialstruktur entsteht durch Erben von der bisherigen Struktur und Hinzunahme der Eigenschaft, unter Motif darstellbar zu sein. Dazu muß jeder Bestandteil ein graphisches Objekt enthalten, wie es in [Rie93b] beschrieben wird. Jede Materialmethode muß dahingehend erweitert werden, daß zusätzlich zu der bisherigen Abarbeitungsfolge auch dieses graphische Objekt entsprechend modifiziert wird.

```
void tMotifGridObject::setLabel( tString *name )
{
 tGridObject::setLabel( name );
 pDisplayObject->SetLabel( *name );
}
```

Jeder Bestandteil erhält eine Methode `getDisplay()`, die das graphische Objekt nach außen liefert. `tMotifStat` enthält eine Liste graphischer Objekte, die durch Traversieren sämtlicher enthaltener Bestandteile gewonnen wird und die mit `getDisplay()` der FK zur Verfügung gestellt wird. `tFKMotifSpin` erhält nur eine neue Methode, nämlich `getDisplay()`, die die von `tMotifStat` erhaltene Liste an die IAK durchreicht. Weitere Modifikationen sind nicht nötig. Diese wenigen Änderungen zeigen deutlich die Mächtigkeit der Werkzeug-Material-Metapher.

5.2.4 Werkzeug (IAK)

Die IAK stellt eine Spezialisierung der `tIAKMain` aus der FIAK-Bibliothek dar. Aus Benutzersicht bedeutet das, daß ihm ein Hauptfenster mit einer Menüleiste zur Verfügung steht. Aus softwaretechnischer Sicht hat dies zur Folge, daß die Grundstruktur aus `Init()`-Methoden und der Benachrichtigung der FK über `DoMenuCommand()` beibehalten wird. Zur Darstellung der Statistik dient ein `tIATCustom`, dem selbige in Form von graphischen Objekten zur Verfügung gestellt wird. Es gibt somit drei Gruppen von Methoden, die im Laufe diese Kapitels behandelt werden:

Die die Oberflächengestalt bestimmenden `Init()`-Methoden und deren Erweiterungen, die die Steuerung betreffende zentrale `Notify()`-Methode und Hilfsmethoden zur Darstellung des Materials.

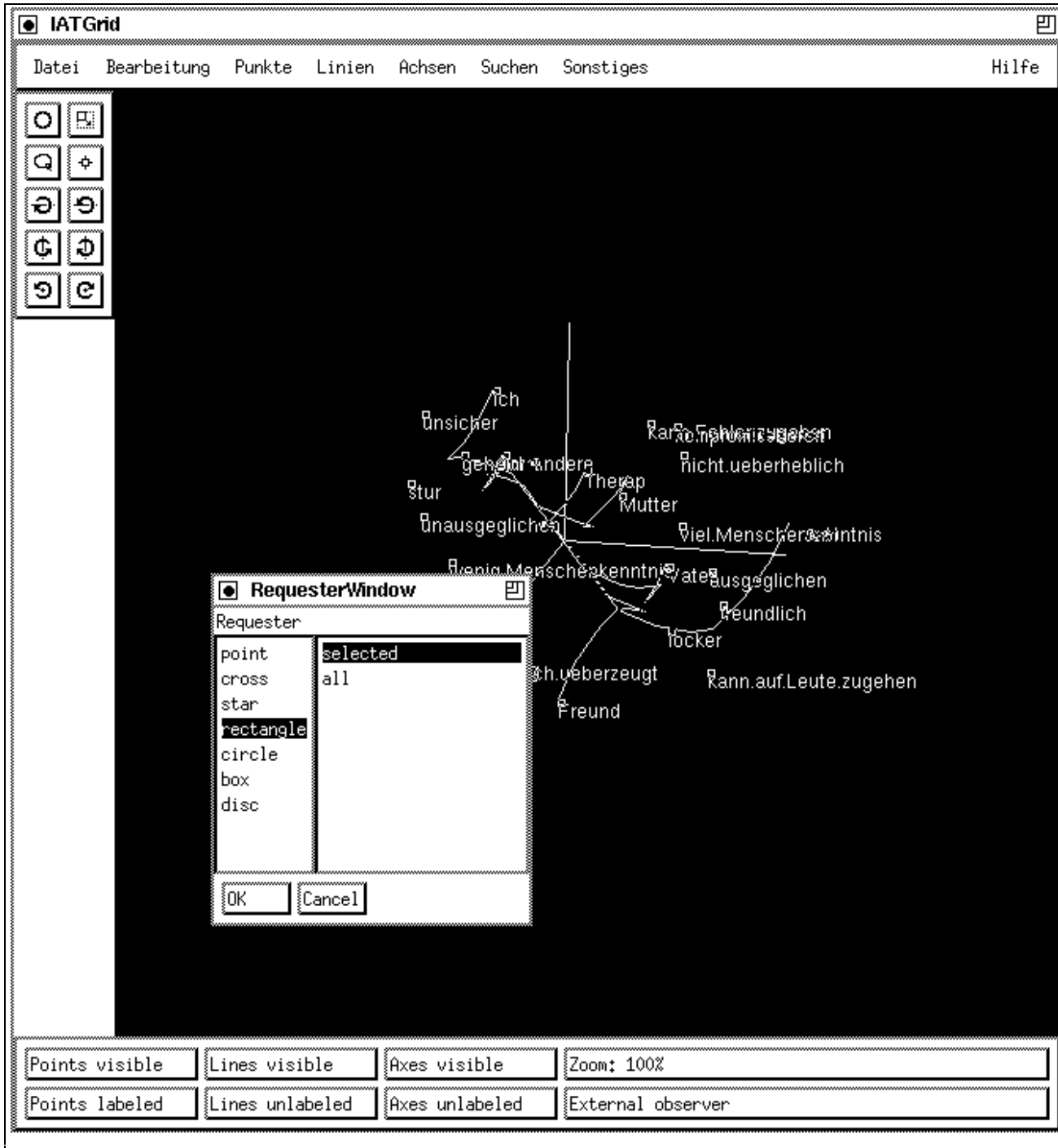


Abbildung 5.2.4-1: Die IAK des IATSpin

Oberflächengestaltung

Gemäß der Systemvision setzt sich die Oberfläche der IAK aus Menüs, Toolboxen und Requestern zusammen.

Menüs. Die Lokalisierung und Selektion von Bestandteilen erfolgt über die Toolbox am linken Rand, die Manipulation über die Menüs "Punkte", "Linien" und "Achsen" sowie über die Toolbox im unteren Bereich. Die Menüs sind so aufgebaut, daß zunächst Menüpunkte zur Manipulation der spezifischen Eigenschaften aufgeführt werden, denen dann ein Menüpunkt zur Selektion aller Bestandteile des jeweiligen Typs folgt. Die

Werkzeugeinstellungs- und sonst nicht zuzuordnende Methoden werden über das "Spezial"-Menü aufgerufen. Ein "Suchen"-Menü erlaubt das Suchen von Bestandteilen anhand ihres Namens.

Der Auswahl nicht bereits eindeutiger Menüpunkte (z.B. "selektiere alle Punkte") folgt das Aufblenden eines adäquaten Requesters, mit dessen Hilfe die gewünschte Aktion spezifiziert werden kann. So folgt zum Beispiel dem Menüpunkt "färbe Punkte" ein Requester zur Auswahl der Farbe und zur Festlegung der betroffenen Punkte (alle Punkte oder selektierte Punkte).

Die Menüs werden in der Klasse `tIAKSpinMenus` definiert, die von `tIAKMenus` erbt. Die `Init()`-Methode baut nach Aufruf des Oberklassen-`Init()`s die eigene Menüstruktur mit Hilfe der speziellen Methoden `InitPoints()`, `InitLines()`, `InitAxes()`, `InitSearch()` und `InitSpecial()` auf.

Toolboxen. Die Aktions-Toolbox auf der linken Seite ist ebenso wie die Zustands-Toolbox am unteren Rand eine Unterklasse von `tIAKToolbox`. In den `Init()`-Methoden werden lediglich horizontale bzw. vertikale Box-IATs angelegt. Dies geschieht gemäß einer von der Haupt-IAK gelieferten Liste von Zeichenketten, die die einzelnen Toolbox-Einträge enthält.

Requester. Die in der FIAK-Bibliothek vorhandenen Requester sind nicht gut für die Bildung von Unterklassen geeignet, da sie je nach Typ direkt auf spezielle Motifrequester (z.B. "Warnung" und "Information") aufbauen. Zur Realisierung von Requestern zum Zweck der Spezialisierung einer Aktion mußte daher eine eigene Requesterklassenhierarchie geschaffen werden, die auf der abstrakten Oberklasse `tIAKRequester`[11] fußt.

Die Spezialisierungen von `tIAKRequester` besitzen eine oder zwei Auswahllisten, aus denen je ein Element ausgewählt werden kann bzw. ein Eingabefeld für Zeichenketten. Am unteren Rand befinden sich zwei Knöpfe zum Bestätigen oder Verwerfen der Eingabe. Die Requester werden bei Bedarf dynamisch von der Methode `InitRequester()` erzeugt, die als Parameter einen anwendungsspezifischen Requestertyp in Form einer Konstante sowie den Requesternamen erhält. In Abhängigkeit von der Konstante werden die Requesterinhalte (Listen oder Texte) erstellt. Jeder Requester bekommt bei der Initialisierung auch das Ereignis mitgeteilt, mit dem bei positiver Auswahl die FK benachrichtigt wird. Die Haupt-IAK kann also ohne Ansehen des Requestertyps dessen Rückgabewerte auslesen und "ungesehen" an die FK durchreichen, ohne weitere Fallunterscheidungen treffen zu müssen.

Steuerung

Die `Notify()`-Methode der IAK ist die zentrale Steuerung der gesamten Anwendung. Hier wird auf Aktionen des Benutzers reagiert und dementsprechend Methoden der FK aufgerufen. Die Struktur dieser

---

[11] Unter Berücksichtigung der Definition in Kapitel 4.2 müßte diese Klasse eigentlich `t`**`IAT`**`Requester` heißen, da es sich lediglich um eine Aggregation von weiteren IATs handelt, die keine weitere Funktionalität beinhaltet. Allerdings müßte dann auch die bereits in der FIAK-Bibliothek enthaltene `tIAKToolbox` überdacht werden, da im Gegensatz zu `tIAKMenus`, außer der ohnehin von der Haupt-IAK gelieferten Liste von Tools, keine weiteren Funktionen übernommen werden. Bis zur Klärung dieser Fragen in Kapitel 5.4 wird die Bezeichnung `tIAKRequester` in Anlehnung an `tIAKToolbox` beibehalten.

Methode wird von einer großen Fallunterscheidung bestimmt, in der zunächst nach Ereignistypen (z.B. `cEventMENUPointColour`) und dann ggf. nach weiteren Kriterien entschieden wird (z.B. zunächst `cEventIAKStateTool` und dann nach der dort ausgewählten Aktion). Elementare Funktionen, die keiner Spezifizierung durch einen Requester bedürfen, werden sofort an die FK weitergeleitet.

```
case cEventMENULineSelectAll:
 {
  pFKMain -> DoMenuCommand ( cFKMENULineSelectAll, "" );
  break;
 }
```

Andernfalls jedoch ändert sich der Zustand der IAK. Diese Änderung kann das Aufblenden eines Requesters sein, was die bisherige Handhabungsart ändert, oder Änderung der Reaktion der IAK auf einen Mausklick im Materialanzeige-IAT. Je nach Auswahl in der Toolbox am linken Rand kann ein Klick Selektion, Versetzen des Cursors (mit Anzeige des aktuellen Labels) oder freie Rotation bedeuten. Diese Zustandsänderungen werden zunächst gespeichert und nach Abarbeitung der zentralen Fallunterscheidung ausgewertet.

```
case cEventMENUAxisStyle:
 {
  StateOfExecution = AXISLINESTYLE_REQUESTER;
  break;
 }
```

Nach der Auswertung der Benutzeraktion kann also zustandsabhängig noch eine weitere Aktion erfolgen, die ggf. solange weiter ausgeführt wird, bis der Benutzer in einen anderen Zustand wechseln läßt.

```
case ROTATE_X_POSITIVE:
 {
  pFKGrid->rotateAbsolute( pFKGrid->getRotationSpeed(), 0, 0 );
  break;
 }
```

Materialdarstellung

Jede Änderung des Materialzustands hat in der FK das Melden des Ereignisses `cEventFKUpdate` zur Folge. Eine Aktualisierung kann auch im Falle einer Überdeckung des Hauptfensters durch andere Anwendungen notwendig sein. Die Aktualisierung wird von `updateDisplay()` vorgenommen. Über `getDisplay()` wird die aktuelle Materialrepräsentation von der FK erfragt, und die erhalten Liste von graphischen Objekten elementweise dem `tIATCustom` zur Abbildung übergeben.

```
 pIATGridDisplayObjects = pFKGrid->getDisplay();
 pIATGridDisplayObjects->Head();
       for(       i=0;       i<pIATGridDisplayObjects->Num();       i++,
pIATGridDisplayObjects->Next() )
  {
                pIATGridDisplay->ShowObject(      (tGFXDraggable      *)
pIATGridDisplayObjects->Get() );
                 pIATGridDisplay->ShowLabel(      (tGFXDraggable      *)
pIATGridDisplayObjects->Get() );
  }
```

5.3 Implementation

Die Realisierung eines Entwurfs auf einem bestimmten System in einer konkreten Entwicklungsumgebung verlangt sowohl Konkretisierungen als auch Modifikationen. Klassen müssen implementiert und Dateien zugewiesen werden. Nicht in der Umgebung Vorhandenes muß entweder selbst erstellt oder umgangen werden. Das Gesicht der Anwendung darf sich dadurch geringfügig ändern, die Funktionalität nicht.

Wie in Kapitel 5.2 werden zunächst allgemeine Gesichtspunkte behandelt und dann die Auswirkungen auf Material, FK, Aspekt und IAK.

Die zu dieser Studienarbeit entstandene Implementation hat einen Umfang von ca. 9200 Programmzeilen. Davon entfallen 2650 auf das Material, 2350 auf die FK, 1000 auf Aspektklassen und 3000 auf die IAK. Die restlichen 200 Zeilen entfallen auf die Anwendungsklasse und das Hauptprogramm. Für die Implementation von Testklassen fielen zusätzliche 8200 Zeilen an. Die Erstellung dauerte, wäre sie an einem Stück erfolgt, ungefähr sechs Wochen à 40 Stunden. Zu jeder Klasse existieren zwei Dateien, deren Rumpfname identisch mit dem jeweiligen Klassennamen ohne vorangehendes "`t`" ist. In der Headerdatei mit der Erweiterung "`.H`" befindet sich sie Klassendefinition. Die Implementation ist in der Datei mit der Erweiterung "`.C`". Material und FK befinden sich im Verzeichnis "`FK`", die IAK mit allen Sub-IAKs ist im Verzeichnis "`IAK`". Die Anwendungsklasse selbst, das Hauptprogramm und die Ressourcen sind im Hauptverzeichnis zu finden.

Die verwendeten Konstanten, die als Parameter für materialverändernde Methoden dienen, sind in "`StatDisplay.H`" innerhalb von Aufzählungstypen definiert. Für jede Materialausprägung besteht ein solcher Typ. Der Name jedes Elements besteht aus einer zweibuchstabigen Typkennung, gefolgt von der eigentlichen Elementausprägung. Zwei weitere Elemente enthalten das "kleinste" bzw. "größte" Element (letztendlich werden die Ausprägungen vom Compiler in aufsteigender Folge auf Ganzzahlen abgebildet), so daß für Gültigkeitsüberprüfungen auch ohne konkrete Kenntnis der möglichen Materialausprägungen eine Abfrage nach dem Muster "`if( ( ... >= MIN_DISPLAYPOINTFORM ) && ( ... <= MAX_DISPLAYPOINTFORM ) )`" ausreicht.

```
enum DisplayPointForms
{
 PF_POINT,
 PF_CROSS,
 PF_STAR,
 PF_RECTANGLE,
 PF_CIRCLE,
 PF_BOX,
 PF_DISC,
 MIN_DISPLAYPOINTFORM = PF_POINT,
 MAX_DISPLAYPOINTFORM = PF_DISC
};
```

5.3.1 Material

Sämtliche Konkretisierungen beziehen sich auf die gesamte Statistik `tStat`. Die einzelnen Materialbestandteile (Punkte, Linien und Achsen) wurden unverändert implementiert.

Die Rotation wurde gemäß dem Standardverfahren für Rotation um eine beliebige Raumachse realisiert, wie sie z.B. in [HeaBak86] beschrieben wird. Für Rotation um die Bildschirmachsen[12] dient eine einfachere Formel, bei der zwei Rotationsschritte entfallen, die bei beliebiger Rotation notwendig sind, um die Statistik entlang einer der Bildschirmachsen auszurichten.

Zur Vereinfachung der Matrizenmultiplikation und -addition wurde die Klasse `tMatrix` entworfen. Im Sinne dieser Studienarbeit nicht relevante Eigenschaften von Matrizen (Inversion, Determinanten- und Gradbestimmung) wurden nicht implementiert.

Zur Beschleunigung der Rotation wurden die notwendigen Matrizenmultiplikationen auf ein Minimum beschränkt und die verwendeten trigonometrischen Funktionswerte von sinus und cosinus in einer Tabelle abgelegt. Da Rotationen in mindestens Ein-Grad-Schritten erfolgen, genügt eine Tabelle mit 360 Einträgen. Die Phasenverschiebung von 90 Grad, um z.B. die Berechnung von sin(180) auf cos(90) zurückzuführen, wurde nicht ausgenutzt. Als nächsten Schritt hätten noch mehrere Symmetrieeigenschaften ausnutzen werden können, um letztendlich nur einen Viertelkreis von 90 Grad tabellieren zu müssen. Da aber Geschwindigkeit im Vordergrund stand, wurde auf jede zusätzliche Berechnung verzichtet.

Die Hilfsfunktionen `min()`, `max()` und `sqr()`, die standardmäßig nicht vorhanden sind, wurden zusammen mit den tabellierten trigonometrischen Funktionen `sint()` und `cost()` zur Klasse `tBasicRoutines` zusammengefaßt, die von `tStat` und `tFKGrid` benutzt wird.

---

[12] Siehe Abbildung 3-6a, Bildschirmachsen

A	B	C	
-1.998	3.606	-0.687	Ich
2.861	-0.020	1.608	Vater
0.722	0.449	-3.326	Mutter
5.556	0.787	-0.176	IdArzt
-0.139	-4.104	-0.199	Freund
0.499	1.899	0.161	Therap
5.556	0.787	-0.176	Ideal
-1.348	2.466	1.280	IchAndere
2.657	3.086	-0.319	kompromissbereit
-2.263	-3.509	-1.178	von.sich.ueberzeugt
-2.377	-1.751	2.652	arrogant
3.259	1.175	2.049	viel.Menschenkenntnis
-3.710	1.605	1.287	stur
2.669	-2.025	0.188	locker
3.520	-0.609	-0.575	ausgeglichen
-3.845	2.764	-1.068	unsicher
-2.657	-3.086	0.319	engstirnig
2.263	3.509	1.178	kann.Fehler.zugeben
2.377	1.751	-2.652	nicht.ueberheblich
-3.259	-1.175	-2.049	wenig.Menschenkenntnis
3.710	-1.605	-1.287	freundlich
-2.669	2.025	-0.188	gehemmt
-3.520	0.609	0.575	unausgeglichen
3.845	-2.764	1.068	kann.auf.Leute.zugehen
0	0	0	Ich0
-0.6102	0.6327	-0.0943	Ich1
-1.4894	1.4964	-0.6300	Ich2
-1.9136	1.7471	-0.3070	Ich3
-2.7183	1.9373	-0.0186	Ich4
-3.0909	1.7538	-0.5327	Ich5
-2.8192	2.0273	-1.1982	Ich6
-2.5155	2.5094	-1.2781	Ich7
-1.9981	3.6059	-0.6869	Ich8
0	0	0	Vater0
-0.3051	0.3164	-0.0471	Vater1
0.5741	-0.5473	0.4886	Vater2
1.4224	-1.0488	-0.1575	Vater3
1.8248	-1.1439	-0.3017	Vater4
2.5700	-0.7768	0.7265	Vater5
2.2982	-1.0503	1.3919	Vater6
2.6019	-0.5682	1.3120	Vater7
2.8606	-0.0200	1.6076	Vater8

Abbildung 5.3.1-1: MacSpin-Dateiformat

Die externe Materialrepräsentation, wie sie von `GetMaterial()` geliefert und von `InsMaterial()` akzeptiert wird, besteht aus einer formatierten Zeichenkette. Gelesen wird das nur grob strukturierte MacSpin-Format, sowie ein eigenes blockorientiertes Format. In Backus-Naur-Form[13] stellen sie sich wie folgt dar[14] :

```
IATSPIN
AXES
3
A
B
C
ELEMENTS
8
      -1.998    3.606     -0.687    Ich
       2.861    -0.020     1.608    Vater
       0.722     0.449    -3.326    Mutter
       5.556     0.787    -0.176    IdArzt
      -0.139    -4.104    -0.199    Freund
       0.499     1.899     0.161    Therap
       5.556     0.787    -0.176    Ideal
      -1.348     2.466     1.280    IchAndere
CONSTRUCTS
16
       2.657     3.086    -0.319    kompromissbereit
      -2.263    -3.509    -1.178    von.sich.ueberzeugt
      -2.377    -1.751     2.652    arrogant
       3.259     1.175     2.049    viel.Menschenkenntnis
      -3.710     1.605     1.287    stur
       2.669    -2.025     0.188    locker
       3.520    -0.609    -0.575    ausgeglichen
      -3.845     2.764    -1.068    unsicher
      -2.657    -3.086     0.319    engstirnig
       2.263     3.509     1.178    kann.Fehler.zugeben
       2.377     1.751    -2.652    nicht.ueberheblich
      -3.259    -1.175    -2.049    wenig.Menschenkenntnis
       3.710    -1.605    -1.287    freundlich
      -2.669     2.025    -0.188    gehemmt
      -3.520     0.609     0.575    unausgeglichen
       3.845    -2.764     1.068    kann.auf.Leute.zugehen
LINES
2
Ich
9
       0         0         0
      -0.6102    0.6327   -0.0943
      -1.4894    1.4964   -0.6300
      -1.9136    1.7471   -0.3070
      -2.7183    1.9373   -0.0186
      -3.0909    1.7538   -0.5327
      -2.8192    2.0273   -1.1982
      -2.5155    2.5094   -1.2781
      -1.9981    3.6059   -0.6869
Vater
9
       0         0         0
      -0.3051    0.3164   -0.0471
       0.5741   -0.5473    0.4886
       1.4224   -1.0488   -0.1575
       1.8248   -1.1439   -0.3017
       2.5700   -0.7768    0.7265
       2.2982   -1.0503    1.3919
       2.6019   -0.5682    1.3120
       2.8606   -0.0200    1.6076
```

Abbildung 5.3.1-2: IATSpin-Dateiformat

---

[13] Backus-Naur-Form: `""` Terminal

`{}` beliebig oft, inklusive 0mal

`[]` beliebig oft, mindestens einmal

`+` Konkatenation

`|` Selektion

[14] Betrachtet wird jeweils der dreidimensionale Fall. MacSpin kann auch höherdimensionale Daten verarbeiten.

```
MacSpin-Datei := Kopfzeile + { Element }
Kopfzeile := { Achsenname }3
Element := Punktkoordinaten + Name
Punktkoordinaten := X-Koord + Y-Koord + Z-Koord
Name := Punktname | Linienpunktname
Punktname := { "A".."z" }
Linienpunktname := Punktname + { Ziffer }
Koord := { Ziffer } + "." + { Ziffer }
Ziffer := "0".."9"
```

Linien werden also wie Punkte dargestellt, nur daß ihnen Nummern folgen, die die Position innerhalb der Linie angeben.

```
IATSpin-Datei := Header + { Block }
Header := "IATSPIN"
Block := Punktblock | Linienblock | Achsenblock
Punktblock := ( "POINTS" | "CONSTRUCTS" | "ELEMENTS" ) + Blocklänge +
{ Punkt }Blocklänge
Linienblock := "LINES" + Blocklänge + { Linie }Blocklänge
Achsenblock := "AXES" + "3" + { Name }3
Punkt := Punktkoordinaten + Name
Punktkoordinaten := X-Koord + Y-Koord + Z-Koord
Linie := Name + Punktanzahl + { Punktkoordinaten }Punktanzahl
Name := { "A".."z" | Ziffer }
Koord := { Ziffer } + "." + { Ziffer }
Ziffer := "0".."9"
```

Polymorphie der Materialbestandteile wird erst in der FK, aber noch nicht im Material, realisiert. Während in der `tFKSpin` Operationen ausschließlich auf `tStatObjects` ausgeführt werden, so besitzt `tStat` noch drei disjunkte Listen für Punkte, Linien und Achsen. Der Grund dafür liegt in der notwendigen Unterscheidbarkeit dieser Bestandteile. Würde `tStat` nur eine einzige Liste von `tStatObjects` verwalten, so wären FK-Methodenaufrufe wie "mache alle Linien unsichtbar" nur noch ausführbar, wenn die einzelnen Bestandteile eine explizite Typinformation enthielten. Dies ist jedoch das genaue Gegenteil dessen, was Polymorphie erreichen will, denn in diesem Fall müßte wieder eine Fallunterscheidung in der FK eingeführt werden, die nach Bestandteiltyp unterscheidet. Um dies zu vermeiden, wurden die drei Listen als implizite Typunterscheidung beibehalten und sämtliche Zugriffe auf diese in der `search()`-Methode von `tFKSpin` gekapselt. `search()` liefert ein beliebiges `tStatObject` als aktuellen Bestandteil, doch hat der Benutzer die Möglichkeit, dessen Typ über die übergebenen Parameter einzugrenzen[15] .

Zur Verringerung des zusätzlichen Implementationsaufwandes bei Einführung von Aspektklassen wurden die Methoden `makeStatPoint()`, `makeStatLine()` und `makeStatAxis()` eingeführt. Sie werden von `InsMaterial()` aufgerufen, um neue Bestandteile einzufügen. Da bei zukünftigen Aspektklassen (z.B. `tMotifStat`) keine `tStatObjects` (sondern z.B. `tMotifStatObjects`) eingefügt werden müssen, das

[15] Siehe Kapitel 5.2.2

Dateiformat sich aber nicht ändert, muß nun nicht `InsMaterial()` mit all seine Korrektheitsüberprüfungen sondern nur `makeStatPoint()`, `makeStatLine()` und `makeStatAxis()` neu implementiert werden.

5.3.2 Werkzeug (FK)

Die verwendeten Konstanten, die als Parameter für FK-Methoden wie `search()` und `DoMenuCommand()` dienen, sind in "`FKGrid.H`" nach dem Muster von "`StatDisplay.H`" als Aufzählungstypen definiert.

Polymorphe Materialmethodenaufrufe, wie sie im Kapitel 5.3.1 beschrieben sind, bringen eine Verkürzung des Programmtexts um 55% mit sich. Aus dreifachen Fallunterscheidung werden einfache polymorphe Aufrufe. Die Datei "`FKSpin.C`" konnte dadurch von 120K bei gleicher Funktionalität auf 55K reduziert werden.

5.3.3 Aspekt

Die Erweiterung der Materialien unter dem Motif-Aspekt besteht in der Hinzunahme eines graphischen Objekts, das die gesamte Darstellungsinformation enthält. Diese graphischen Objekte sind in "`Symbol.H`" definiert und in "`Symbol.C`" implementiert.

Die Zuordnung von graphischen Objekten zu Materialbestandteilen sieht wie folgt aus: Punkte werden auf Bitmaps, Linien auf sogenannte Polylinien mit mehreren Stützpunkten und Achsen auf einfache Linien abgebildet. Diese Abbildung erfolgt in Klassen, die bereits den jeweiligen Materialbestandteilen entsprechen (`tDisplayLine`, `tDisplayAxis`) , wobei Punkte noch weiter nach den möglichen Punktformen unterteilt sind (`tDisplayPoint_Cross`, `tDisplayPoint_Star`, usw.).

Um bei einem Unsichtbarmachen nicht das graphische Objekt selbst zu löschen, das dann bei der Sichtbarmachung wieder neu erzeugt werden müßte, enthält jedes `tDisplayObject` ein graphisches Objekt für die sichtbare (`pVisibleDisplayObject`) sowie für die unsichtbare (`pInvisibleDisplayObject`) Darstellung. Im Falle der Unsichtbarkeit wird einfach der interne Verweis vom `pVisibleDisplayObject` auf das `pInvisibleDisplayObject` umgelenkt. Das `pInvisibleDisplayObject` enthält immer eine leere Bitmap. Modifikationen des Materials beziehen sich immer auf das `pVisibleDisplayObject`.

Mehrfachvererbung wird dadurch erforderlich, daß z.B. eine `tMotifStatAxis` sowohl von `tStatAxis` als auch von `tMotifStatLine` erbt. Um die Exemplarvariablen dabei nicht zu verdoppeln (ein Variablensatz je direkter Oberklasse) muß bei der Klassendefinition dem Vererbungskonstrukt das Schlüsselwort `virtual` vorangestellt werden. Ebenso muß der Konstruktor der gemeinsamen Basisklasse parameterlos sein[16] .

---

[16] Diese Voraussetzung ist wenig dokumentiert. Die Fehlermeldung des Compilers ist von geringer Aussagekraft:

```
internal <<AT&T C++ Translator 2.1.03 08/31/90>> error:
int  argument missing for tStatObject::~tStatObject()
```

Die Lösung fand sich in [Lip91, S.363].

5.3.4 Werkzeug (IAK)

Der `tIATCustom`, der zur Darstellung des Materials dient, zeichnet jedes übergebene graphische Objekt bei Erhalt sofort neu. Bei Anwendungen mit einer relativ geringen Anzahl von graphischen Objekten wie z.B. dem Aufgabennetzeditor in [Rie93a] hat dies kaum sichtbare Folgen. Beim IATSpin mit seinen vielen Bestandteilen zeigt sich jedoch ein starkes Flackern, welches einen 3D-Effekt bei Rotationen fast vollständig verhindert. Die Geschwindigkeit der Rotation leidet ebenfalls beträchtlich unter der dauernden Neuzeichnung des `tIATCustom`s nach jedem neu eingefügten graphischen Objekt.

Als Abhilfe wurde ein Double-Buffering-Custom-IAT entwickelt, bei dem graphische Objekte zunächst verdeckt in eine unsichtbare Bitmap gezeichnet werden und dann nach Fertigstellung der Darstellung mit einem Schlag in die sichtbare Bitmap kopiert werden. Diese Idee kam dem Autor beim Betrachten von `xasteroids` [Goe90] und der anschließenden Analyse des Programmtexts.

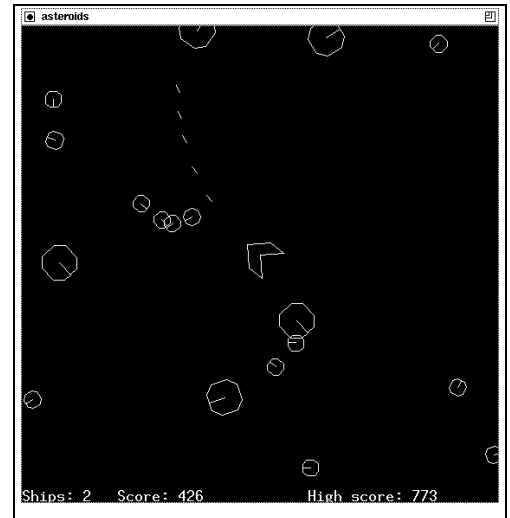


Abbildung 5.3.4-1: xasteroids

Die zusätzlichen Custom-IAT-Methoden heißen `SetBuffered()` und `FlushBuffer()`. Zum Einschalten der unsichtbaren Zeichen-Bitmap muß die Methode `SetBuffered()` mit dem Parameter `TRUE` aufgerufen werden. Bei `FALSE` verhält der IAT sich wie gewohnt. Mit `FlushBuffer()` wird die unsichtbare in die sichtbare Bitmap kopiert und die unsichtbare Bitmap für weitere Zeichenoperationen gelöscht.

In `tIAKSpin::updateDisplay()` muß also nach der Darstellung der Statistik `pIATGridDisplay->FlushBuffer()` aufgerufen werden.

5.4 Bewertung

Nach Abschluß der Implementation bleiben noch zahlreiche Punkte, die in den Anforderungen vorhanden sind, nicht realisiert. Dies liegt an drei verschiedenen Kategorien von Ursachen, die im weiteren Verlauf ausgeführt werden. Darüber hinaus sind einige der getroffene Entwurfsentscheidungen durchaus kritisch diskutierbar.

Nicht realisierte Funktionen aufgrund von Nichtenthaltensein benötigter Funktionen in der IATMotif-Bibliothek:

- **Listen am rechten Rand der IAK, die eine textuelle Repräsentation des Materials enthalten.** Listen mit Rollbalken sind nicht verläßlich einsetzbar. Eine Markierung entsprechend des Selektionszustandes oder der jeweiligen Punktform ist nicht möglich.

- **Intuitive Anzeige des aktuellen Werkzeugzustands.** Eine Hervorhebung von Elementen der Toolbox am linken Rand dergestalt, daß der aktuelle Modus zu erkennen ist (z.B. Rahmen um "Rechteck-Selektion"), ist nicht möglich. Eine Ersatzlösung wurde in Hoffnung auf baldige Erweiterung der Bibliothek nicht implementiert.

- **Anhaltende Rotation durch Dauerdrücken der Rotationsknöpfe.** Der dem `tIATBox` zugrundeliegende `tIATButton` meldet nur einen `cEventIATSelect`, nicht aber einen `cEventIATDeselect`, so daß das Loslassen des Knopfes nicht registriert werden kann. Dies führt zu einer sehr unbequemen Handhabung, bei der der Knopf in schneller Folge solange gedrückt werden muß, wie die Rotation aufrecht erhalten werden soll.
- **Flackerloser Wechsel der Beschriftung der Status-Toolbox.** Eine Umbenennung eines Knopfes der Status-Toolbox kann nur über eine Neuerzeugung der Toolbox erfolgen, da ein gezielter Namenswechsel nicht unterstützt wird. Dadurch wird der gesamte untere Bereich der IAK kurzfristig graphisch neu aufgebaut, was optisch sehr störend und zeitaufwendig ist. Um unnötige Aufrufe zu verhindern, wurde die Methode `updateStateToolbox()` eingeführt, die nach jeder Änderung einer Einstellung direkt aufgerufen wird. Dieses ist sehr unschön, da die Statustoolbox eigentlich alle Veränderungen ungefiltert mitgeteilt bekommen sollte, und daraus die für sie relevanten Ereignisse eigenständig heraussuchen sollte.
- **Dateirequester mit Filter für Dateinamen.** Eine Filterung der Dateinamen, die im Requester auftauchen (z.B. nur "`*.spin`"-Dateien), wäre wünschenswert, wird aber vom `tIATFile` nicht unterstützt.
- **Farbige graphische Objekte.** Es sind nur Graustufen in 5%-Schritten vorgesehen, auf die die Farben innerhalb von `tMotifStatObject` als Zwischenlösung abgebildet werden (schwarz $\rightarrow$ 100%, rot $\rightarrow$ 80%, bis weiß $\rightarrow$ 0%). Bei Bitmaps, die für die Repräsentation von Punkten verwendet werden, haben auch Grautöne keinerlei Ausprägung. Bitmaps können nur normal oder invertiert dargestellt werden.
- **Verschiedene Linienarten.** Linien können nur normal oder invertiert durchgezogen gezeichnet werden. Strichlierungen sind nicht vorgesehen.
- **Kontinuierlicher Zeichenmodus im `tIATCustom`.** Pixel werden im `tIATCustom` im XOR-Modus gesetzt, das heißt, daß das Zeichnen eines Punktes an eine Position, an der sich bereits ein Punkt befunden hat, die Auslöschung dieses Punktes bedeutet. Dies hat zur Folge, daß z.B. Achsen bei der Rotation kurzfristig verschwinden, wenn sie sich überlagern, was auf den Benutzer irritierend wirkt. Dieses Verhalten stammt noch aus der Zeit des Nicht-Double-Buffering-`tIATCustoms`, als graphische Objekte nur durch erneutes Zeichnen im XOR-Modus gelöscht werden konnten. Für den Double-Buffering-`tIATCustom` wäre Zeichnen im OR-Modus erstrebenswert.
- **Reaktion des `tIATCustoms` auf Größenveränderung des Hauptfensters.** Der `tIATCustom` paßt sich bei entsprechender Spezifikation in den Ressourcen zwar bei einem "Resize" in der Form den neuen

Ausmaßen an, teilt diese aber nicht seinem Inhalt mit. Somit kann die Darstellung des Materials nicht adäquat erfolgen.

Nicht realisierte Funktionen aufgrund programmiertechnischer Ökonomie:

Angesichts der beliebig erweiterbaren Liste von Verbesserungen und Erweiterungen mußte ein Schlußstrich gezogen werden. Den folgenden Punkten liegen keine prinzipiellen Probleme zugrunde und könnten wahrscheinlich zum Teil in kurzer Zeit implementiert werden.

- **Benutzerführung in einheitlicher Sprache.** Menüs und Knöpfe können leicht über die Ressourcen mit Direktiven wie `"*Name.labelString: angezeigter Name"` umbenannt werden. Die in Auswahllisten und Toolboxen verwendeten Namen müssen jedoch im Programmtext spezifiziert werden. Hierzu bietet sich eine Unterscheidung mit Präprozessoranweisungen à la "`#ifdef GERMAN`" an.
- **Stets sichtbarer Materialname.** Es ist im Werkzeug nicht zu erkennen, welche Datei gerade bearbeitet wird. Das Material könnte seinen Namen z.B. in die Liste der graphischen Objekte einreihen, die es dem Werkzeug zu Darstellungszwecken zur Verfügung stellt.
- **Suche und Selektion nach Namen.** Für diese Operationen müßte ein komplexer Requester entworfen werden, der neben der Eingabe des Namens eine Einschränkung der Suche auf bestimmte Materialbestandteile (Punkte, Linien, Achsen; selektierte, alle) erlaubt.
- **Brushing.** Ein Brush kann über Rechteck-Selektion realisiert werden. Zusätzlich müßte ein Brush frei definiert werden können.
- **Informationsfenster.** Um die Werte, die `tFKSpin::getInfo()` zurückliefert, darzustellen, wäre ein Requester notwendig, der nur einen unmodifizierbaren Text sowie eine "OK"-Knopf besitzt.
- **Bildschirmausdruck.** Wenn die Zeichenoperationen innerhalb des `tIATCustoms` nicht auf einen `tCanvas`, sondern einen `tPSCanvas`, wie er in der IATMotif-Bibliothek sogar vorhanden ist, abgebildet würden, müßte ein Ausdruck möglich sein.
- **Konvertierung von Dateiformaten.** Beim Speichern des Materials wird dieselbe Datei wie beim Laden (evtl. unter anderem Namen) zurückgeschrieben. `tStat::GetMaterial()` liefert also lediglich das mit `tStat::InsMaterial()` eingefügte Material zurück.

Nicht realisierte Funktionen aufgrund noch nicht vollständiger Analyse:

Im Gegensatz zu den vorhergehenden Punkten sind die folgenden bisher nur angedacht oder theoretisch noch nicht vollständig durchdrungen. Daß auch solche darunter sind, die in den Anforderungen auftauchen, liegt daran, daß ihre Umsetzung sich im Laufe der Arbeit als wesentlich komplizierter herausgestellt hat, als dies zu Beginn anzunehmen war.

- **Zwischenspeicherung von Rotationsbildern** wäre mit der zunächst vorgesehenen Realisierung der Rotation unproblematisch gewesen. Jede Position sollte ursprünglich nur über drei Rotationswinkel angegeben werden können (Rotation um absolute X-, Y- und Z-Achse). Diese Winkel sind jedoch nicht dazu geeignet, eine eindeutige Position anzugeben, da Matrizenmultiplikationen, mit denen die Rotation realisiert wird, nicht kommutativ sind. Es müßte also eine Reihenfolge der Rotationsschritte festgelegt werden (z.B. erst X, dann Y, dann Z). Rotationen um eine andere als die erste Achse wären mathematisch sehr aufwendig, so daß das Verfahren mit dem zusätzlichen Koordinatensatz in den einzelnen Materialbestandteilen gewählt wurde[17] . Die Klasse `tStat` enthält nun keine unmittelbaren Informationen mehr über ihren Rotationszustand. Demzufolge kann er auch nicht gespeichert werden.

  Ein Lösungsweg, bei dem nicht die gesamte Statistik zu Speicherungszwecken dupliziert werden müßte, wäre eventuell die Einführung einer Rotationsgeschichtenmatrix. Diese Matrix enthielte zunächst die Identitätsmatrix und würde dann mit jeder folgenden Rotationsmatrix multipliziert. Um also zum momentanen Rotationsstand zu gelangen, müßten alle ursprünglichen Koordinatensätze nur mit dieser Matrix multipliziert werden. Diese Matrix könnte platzsparend gespeichert werden.

- **Die zufriedenstellende Planetariumsbetrachtung** ist mit der bisherigen Methode, die lediglich die Punkte mit einer negativen Z-Koordinate darstellt, nicht gelungen. Eine spezielle Projektionsformel, die die gesamte Darstellungsfläche ausnutzt, muß gefunden werden.

- **Unterschiedlich skalierbare Achsen** stellen zusätzliche Anforderungen an die Projektionsformel. Gemäß der jeweiligen Skalierungsart muß jede Koordinate ein weiteres Mal umgerechnet werden. Wie sich die Darstellung beim Umstellen der Skalierungsart verhalten soll (automatische / manuelle Neuoptimierung) und wo diese Information abgelegt werden soll (Änderung der ursprünglichen Datenkoordinaten?) ist ungeklärt.

- **Die Selektion von Bestandteilen der Statistik mit einem Lasso** erfordert eine komplexe Berechnung, um festzustellen, welche Bestandteile innerhalb des Lassos sind. Denkbar erscheint ein Verfahren in Anlehnung an das Zeichnen von Polygonen auf Rastergeräten. Anstatt die Polygonfläche jedoch zu füllen, müßten in einer Schleife alle Punkte im Inneren durchlaufen und selektiert werden. Da die Punktkoordinaten in Fließkommazahlen abgelegt werden, müßte die Selektionsgenauigkeit soweit heruntergesetzt werden, daß garantiert alle Bestandteile erfaßt würden.

- **Das Zoomen eines beliebigen Bereichs** läßt sich auf zweierlei Arten umsetzen. Entweder wird der gewählte Bereich vergrößert und zum neuen Rotationsmittelpunkt, oder aber der alte Mittelpunkt bleibt erhalten. Die erste Methode verbindet einen Betrachtungspositionswechsel unzulässigerweise mit einer Materialmanipulation. Bei der zweiten Methode huschen die Punkte beim Rotieren (zu?) schnell vorüber. Hier muß anforderungsgemäß abgewogen werden.

- **Das Löschen von Bestandteilen der Statistik** ist in der Systemvision nicht vorgesehen. Um jedoch die in Kapitel 3 beschriebene volle Funktionalität einer dynamischen Statistik zu erreichen, müßte auch das

[17] Siehe Kapitel 5.2.1

Löschen realisiert werden. Dazu müßten, bei `tStatObject` beginnend, die Traversierungsmethoden[18] erweitert werden.

- **Das Einfügen von Hilfskonstrukten zur Verbesserung des räumlichen Eindrucks** wirft die Frage auf, wo und wie diese Konstrukte abgelegt werden sollen (im Material oder in der FK) und wie sie wieder entfernt werden sollen (unsichtbar machen oder löschen[19]).
- **Ein Hilfesystem**, wie es bereits durch das "Hilfe"-Menü der `tIAKMain` andeutet ist, fehlt bisher vollkommen. Eine sinnvolle Einbindung in die FIAK-Bibliothek und somit auch in die Werkzeug-Material-Metapher scheint dem Autor notwendig zu sein.

Abgesehen von den bisher erwähnten Punkten, die noch nicht umgesetzt worden sind, soll im weiteren Verlauf diskutiert werden, ob die bereits getroffenen Entwurfsentscheidungen adäquat gewesen sind. Da sich im Prinzip natürlich alles anders machen ließe, beschränkt sich der Autor hier auf drei Punkte: Die Art, in der Linien ihre Raumkurve verwalten, die Unterscheidung zwischen einer IAK- oder einem IAT-Requester und die Frage, ob Requester überhaupt sinnvoll sind.

Linien verwalten die sie definierende Raumkurve in Form von `tStatPoints`. Von diesen werden jedoch nur die Koordinatensätze tatsächlich genutzt. Die weiteren Punkteigenschaften (Name, Form, Farbe, Darstellungsstärke, Selektionszustand, Sichtbarkeitszustand, Sichtbarkeitszustand des Labels) liegen brach. Diese Konstruktion hat ihre Ursache in einer unklaren Systemvision. Lange Zeit war es nicht klar, ob die Einzelpunkte einer Linie (wie bei MacSpin) auch individuell existenzberechtigt sind, oder ob sie nur ein Hilfskonstrukt als Bestandteil einer Linie darstellen. Erst spät stellte es sich heraus, daß die MacSpin-Art der Liniendarstellung nur eine Behelfslösung unter MacSpin-Bedingungen war. Die einzelnen Linienpunkte besitzen keine fachliche Aussagekraft und sind somit als Individuen uninteressant.

Zum jetzigen Erkenntniszeitpunkt könnten sie durch neu zu schaffende `tStatLinePoints` ersetzt werden, die nur aus Koordinatensätzen bestehen.

IAK oder IAT Requester? In Kapitel 4.2 wird klar zwischen Subwerkzeug (also IAK) und IAT unterschieden. Ein Subwerkzeug besitzt Kontextwissen über die Informationen, die es verarbeitet (und kann somit bei Bedarf eine Plausibilitätskontrolle durchführen), ein IAT nicht. Demzufolge sind Requester als IATs einzuordnen. Die in der FIAK-Bibliothek vorhandenen Requester sind zwar IATs, jedoch zur Spezialisierung ungeeignet, da sie direkt auf Motif-Requester abgebildet werden. Eine unabhängige Requesterhierarchie, wie in dieser Arbeit entwickelt, sollte sie mittelfristig ersetzen.

An der Bezeichnung `tIATRequester` sollte festgehalten werden. Dementgegen sollte in Erwägung gezogen werden, ob es sich bei anderen IAKs nicht tatsächlich um IATs handelt. Der `tIAKToolbox` besitzt z.B. keine eigene Funktionalität, die über die des `tIATBox` hinausgeht.

---

[18] Siehe Kapitel 5.2.1

[19] Siehe letzter Punkt

Requester kontra Untermenüs. Ein Requester unterbricht kurzzeitig die direkte Manipulation des Materials. Wünschenswert wäre eine Veränderungsmöglichkeit "direkt am Material". Durch die zusätzliche Einschränkung der Darstellungsmöglichkeiten innerhalb von Requestern[20] muß der Benutzer sich auch noch auf mehrere Repräsentationsformen von Materialeigenschaften einlassen. So werden z.B. Punktformen in Requestern textuell beschrieben.

Über Untermenüs ließen sich solche Einstellungen in Verbindung mit der `"*Menüpunktname.labelPixmap: Bilddateiname"`-Direktive für Ressourcen sehr ansprechend und intuitiv erfaßbar gestalten.

Aufgrund des relativ hohen Änderungsaufwandes, den eine solche Umstellung mit sich zieht, wurde auf diese Verbesserung verzichtet. Die Menü-IAKs müßten dahingehend erweitert werden, daß im Falle der Auswahl nicht nur z.B. `cEventMENUPointForm` sondern `cEventMENUPointFormRectangle` gemeldet würde.

[20] Siehe "Nicht realisierte Punkte" zu Beginn diese Kapitels

6 Zusammenfassung und Ausblick

Gegenstand dieser Arbeit ist die Erstellung eines Interaktionstyps für dynamischen Statistiken. Als konkretes Anwendungsbeispiel diente die Auswertung von Kelly-Grids. In Kapitel 2 wurde das Fachkonzept der Kelly-Grids erläutert. In Kapitel 3 wurden dynamische Grafiken für Statistiken abstrakt eingeführt und die Rotationsmethode als am anschaulichsten ausgewählt. In Kapitel 4 wurden Grundlagen der objektorientierten Entwurfs im allgemeinen und die Werkzeug-Material-Metapher im speziellen vorgestellt. Letztere diente in Kapitel 5 als Leitbild zur Erstellung des Interaktionstyps. Aufgrund mangelnder Kommunikation erfolgte die Entwicklung nicht evolutionär sondern nahezu zyklenfrei. Wegen unzureichender Kenntnis der Dokumenttypen des objektorientierten Entwicklungsprozesses, wie er in [KilGryZül93] beschrieben wird, sind die Anforderungen nur unscharf spezifiziert worden. Die Tragfähigkeit der Werkzeug-Material-Metapher konnte daher nur im technischen Entwurf untersucht werden.

Die Umsetzung der ursprünglichen Anforderungen ist weitgehend gelungen. Die Strukturierung des Materials und der FK sind übersichtlich und leicht erweiterbar. Gleiches gilt für die IAK, ihre Kommunikation mit der FK und die IATs.

Ersteres läßt sich auf die Mächtigkeit von objektorientierten Entwurfsmethoden im allgemeinen, letzteres auf die Werkzeug-Material-Metapher im speziellen zurückführen. Beide erlauben nach einer gewissen Einarbeitungszeit, deren Länge nicht unterschätzt werden sollte, eine überschaubare Entwicklung von Anwendungssystemen. Der Autor kann die allgemeine Erfahrung bestätigen, daß objektorientiert durchgeführte Projekte eine längere Vorbereitungszeit erfordern, dafür aber Änderungen zu späteren Zeitpunkten wesentlich erleichtern.

Trotz dieser softwaretechnischen Erfolge ist das System bisher nicht für die ernsthafte Analysetätigkeit einsetzbar. Dies liegt vor allem an der unzureichenden Unterstützung spezieller Interaktionsformen durch die IATMotif-Bibliothek. Während sie von ihrer Struktur her vorbildlich das vorhandene Fenstersystem im objektorientierten Sinne kapselt, so mangelt es ihr doch erheblich an selbst elementarsten IATs. Der Autor empfiehlt dringend eine Vervollständigung. Bei einem eventuell geplanten Wechsel auf eine andere Bibliothek zu Darstellungszwecken sollte auf jeden Fall das die IATMotif-Biliothek durchziehende Beobachterkonzept beibehalten werden, durch das die Anbindung an das Werkzeug erfolgt. Es hat sich im Laufe dieser Arbeit als sehr mächtig herausgestellt.

Eine Einbindung des "IAT"Spins in die IATMotif-Bibliothek ist bisher noch nicht erfolgt und dürfte sich als sehr problematisch herausstellen. Aufgrund der beachtlichen Komplexität[21] und des Rückgriffs auf Kontextwissen[22] scheint eine Einordnung nur als Subwerkzeug sinnvoll. Die IATMotif-Bibliothek sollte weiterhin aus möglichst elementaren IATs bestehen.

---

[21] Der bisher größte IAT der Bibliothek besitzt einen Umfang von 550 Zeilen. Der "IAT"Spin besitzt 24000 Zeilen.

[22] Der "IAT"Spin interpretiert z.B. die ihm übergebene Materialzeichenkette und stellt sie dementsprechend dar. Nur mit Kontextwissen kann eine Transformation zwischen verschiedenen textuellen Repräsentationen erfolgen.

Eine Integration mit verwandten Studienarbeiten [Fah94] steht noch aus. Außer der Bekanntgabe des Dateiformats erfolgte keinerlei Abstimmung. Bei weiterem Interesse am Themenkomplex Kelly-Grids ergäben sich hier Themen für weitere Arbeiten.

Im Falle der weiteren Vergabe von Studienarbeiten, bei denen Anwendungssysteme für Fachbereichexterne erstellt werden sollen, sollte ein Augenmerk auf die Ansprechbarkeit der jeweiligen Partner gerichtet werden. Monatliche persönliche Konsultationen erscheinen dem Autor hierfür das absolute Minimum zu sein, um unnötigen Komplikationen vorzubeugen[23] .

Besonders am Herzen liegt dem Autor der Hinweis, daß es sich beim IATSpin nicht um ein System handelt, mit dem ausschließlich Psychologen arbeiten können. Mit minimalen Erweiterungen können alle in Kapitel 3 genannten Eigenschaften von dynamischen Statistiken realisiert werden. Dieses universelle Hilfsmittel verdient größere Beachtung und rechtfertigt weitere Entwicklungen um den Noch-IAT IATSpin.

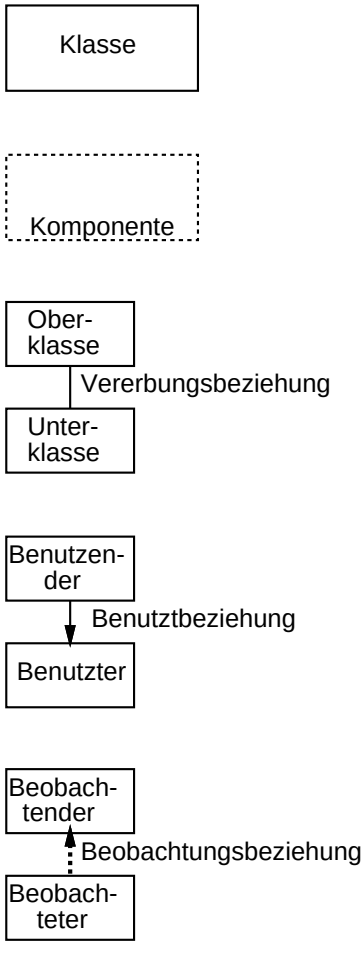
[23] Siehe Abschnitt "Linien" in Kapitel 5.4

Anhänge

Zeichensatzkonventionen

`Courier`	Klassennamen, Bezeichner und Programmfragmente
italics	Fachbegriffe aus Anwender- und Entwicklerwelt bei ihrer ersten Nennung

Graphische Konventionen



Namenskonventionen

Im Programm

- Englisch als Implementationssprache (kürzere Bezeichner als im Deutschen möglich)
- frühestmögliche Großschreibung bei zusammengesetzten Worten, wenn Abgrenzung zu anderen Worten wichtig ist (z.B. `Visibility`**`S`**`tate` und `Selection`**`S`**`tate` (zwei Bezeichner mit Ende auf "`State`") vs. `Zoom`**`f`**`actor` und `Tool`**`b`**`ox` (zwar zusammengesetzt, aber keine weiteren Bezeichner im Projekt, die mit `Zoom` oder `Tool` beginnen)

- Klassennamen großgeschrieben und mit vorhergehendem "t" (`tStat`)
- Zeigernamen mit vorhergehendem "p" (`pCurrentStatObject`)
- Exemplarvariablen großgeschrieben (`PointForm`)
- sonstige Variablen kleingeschrieben (`maxPoint`)
- Methodennamen kleingeschrieben mit vorangehendem Verb (`setOrigin()`)

Im Betriebssystem

- Klassendefinitionsdateiname := Klassenname ohne Anfangs-"t" + ".H" (`MotifGridObject.H`)
- Klassenimplementationsdateiname := Klassenname ohne Anfangs-"t" + ".C" (`MotifGridObject.C`)
- Testklassendateiname := Klassenname ohne Anfangs-"t" + "Test.C" (`MotifGridObjectTest.C`)

Literaturverzeichnis

1. [BecCleWei88] Becker, R.A., Cleveland, W.S., Weil, G., *The Use of Brushing and Rotation for Data Analysis*, in: Cleveland, W.S., McGill, M.E., Hrsg., *Dynamic Graphics for Statistics*, Belmont, Calif., 1988, S.247-275
2. [BecCleWil88] Becker, R.A., Cleveland, W.S., Wilks, A.R., *Dynamic Graphics for Data Analysis*, in: Cleveland, W.S., McGill, M.E., Hrsg., *Dynamic Graphics for Statistics*, Belmont, Calif., 1988, S.1-72
3. [CatSch93] Catina, A., Scheer, J.W., *Psychologie der Persönlichen Konstrukte und Repertory Grid-Technik*, in: Catina, A., Scheer, J.W. (Hrsg.), *Einführung in die Repertory Grid-Technik. Band1: Grundlagen und Methoden*, Bern, 1993, S.8-23
4. [DonDonGas88] Donoho, A.W., Donoho, D.L., Gasko, M., *MACSPIN: Dynamic Graphics on a Desktop Computer*, in: Cleveland, W.S., McGill, M.E., Hrsg., *Dynamic Graphics for Statistics*, Belmont, Calif., 1988, S.331-351
5. [Fah94] Fahrenholtz, D., *Entwurf und Implementation einer Benutzerschnittstelle für computerunterstützte Repertory Grids unter C++ und OSF/Motif*, Studienarbeit, Hamburg, 1994
6. [Goe90] Goetz, P., *xasteroids*, Buffalo, New York, 1990
7. [GryZül94] Gryczan, G., Züllighoven, H., *Objektorientierte Konstruktion von Software-Werkzeugen*, zur Veröffentlichung vorbereitet
8. [HeaBak86] Hearn, D., Baker, M.P., *Computer Graphics*, London, 1986
9. [KilGryZül93] Kilbert, K., Gryczan, G., Züllighoven, H., *Objektorientierte Anwendungsentwicklung*, Braunschweig, 1993
10. [Lip91] Lippman, S., *C++Primer*, 2.Auflage, Reading, Mass., 1991

11. [Rae93] Raeithel, A., *Auswertungsmethoden für Repertory Grids*, in: Catina, A., Scheer, J.W. (Hrsg.), *Einführung in die Repertory Grid-Technik. Band1: Grundlagen und Methoden*, Bern, 1993, S.41-67

12. [Rie93a] Riehle, D., *Objektorientierte Architektur nach der Werkzeug-Material Metapher am Beispiel eines grafischen Aufgabennetzeditors*, Studienarbeit, Hamburg, 1993

13. [Rie93b] Riehle, D., *Interaktionstypen Dokumentation zur IATMotif v1.0 Bibliothek*, Hamburg, 1993

14. [Rie93c] Riehle, D., *Dokumentation zur FIAK v1.1 Bibliothek*, Hamburg, 1993

15. [Sch93] Scheer, J.W., *Planung und Durchführung von Repertory Grid-Untersuchungen*, in: Catina, A., Scheer, J.W. (Hrsg.), *Einführung in die Repertory Grid-Technik. Band1: Grundlagen und Methoden*, Bern, 1993, S.24-40