# Interaktionsformen für objektorientierte, reaktive Softwaresysteme unter Berücksichtigung der Werkzeug-Material-Metapher

**Studienarbeit am**
**Arbeitsbereich Softwaretechnik**
**im Fachbereich Informatik**
**Universität Hamburg**

**Autor: Thorsten Sturm**
**Betreuer: Dr. Guido Gryczan**

# Inhaltsverzeichnis

# 1 Einleitung

Ein Schwerpunkt der Arbeit am Arbeitsbereich Softwaretechnik (SWT) des Fachbereichs Informatik an der Universität Hamburg ist die Entwicklung einer durchgängigen Methodik zur Entwicklung objektorientierter Software für reaktive Softwaresysteme.[1] Unter reaktiven Softwaresystemen werden dabei Systeme verstanden, die auf Eingaben des Benutzers warten und reagieren. Die Eingaben bestimmen dabei, wann welche Komponente des Systems in welcher Form aktiviert wird. Nachdem die Eingabe des Benutzers abgearbeitet wurde, wartet das System auf die nächste Eingabe durch den Benutzer (siehe auch [KGZ94] ). Als Leitbild für die Softwareentwicklung kommt dabei das Bild des Arbeitsplatzes für qualifizierte menschliche Arbeit zur Anwendung. Der daraus entwickelten Methodik folgend, wurden verschiedene objektorientierte Klassenbibliotheken entwickelt, die ihre Verwendung unterstützen sollen. Hierzu wird, auf Basis einer vorgegebenen Entwicklungsmetapher, eine bestimmte Struktur der Software festgelegt. Nach den Hauptbestandteilen der resultierenden Softwaresysteme wird diese Methodik Werkzeug-Automat-Material (WAM) genannt.

Die Problematik bei der Entwicklung reaktiver Softwaresysteme liegt in der Unvorhersagbarkeit von Ereignissen. Der Benutzer teilt der Anwendung mit, was er als nächstes von ihr erwartet. Die daraus resultierende Reaktion der Anwendung wird wiederum dem Benutzer zur Kenntnis gebracht. Diese Form des Informationaustausches wird meist ***Interaktion***[2] genannt. Da moderne, reaktive Softwaresysteme kaum noch ohne die Verwendung von graphischen Benutzungsoberflächen auskommen, gilt bei der Entwicklung und Wartung derartiger Softwaresysteme der zugehörigen Benutzungsschnittstelle zunehmende Aufmerksamkeit.

Auch WAM trägt diesem Punkt Rechnung. Kapitel 2 zeigt jedoch, daß die bisherige Einbettung der Benutzungsschnittstelle eine einschränkend enge Kopplung zwischen den anwendungsspezifischen und den technischen Bestandteilen der Interaktionen vorgibt. Diese Kopplung kann zu erhöhtem Arbeitsaufwand bei der Entwicklung und Wartung des Softwaresystems führen.

Diese Arbeit geht der Fragestellung nach, wie eine stärkere Entkopplung der technischen und anwendungsspezifischen Bestandteile der Interaktion erreicht werden kann. Dazu wird mit den ***Interaktionsformen*** ein neues Konzept zur Trennung zwischen dem anwendungsspezifischen und dem technischen Teil der Interaktion eingeführt. Besondere Beachtung finden dabei die Möglichkeiten unterschiedlich starker Entkopplung. Die Kapitel 3 und 4 zeigen, wie dieses Konzept in die schon bestehende Struktur eingefügt werden kann. Dabei werden die möglichen Interaktionen auf ihre Abstrahierbarkeit untersucht und die unterschiedlichen Entkopplungsansätze diskutiert. Die Kapitel 5 und 6 beschäftigen sich abschließend mit einer prototypischen Realisierung von Interaktionsformen auf Basis des empfohlenen Entkopplungsansatzes. Die Realisierung findet in Form einer objektorientierten C++-Klassenbibliothek statt, die mit den bereits bestehenden Klassenbibliotheken kooperiert.

[1] Einzelne Mitglieder des Lehrpersonals haben dieses Thema schon an der TU Berlin verfolgt und ihre damaligen Ergebnisse mit nach Hamburg gebracht.

[2] Die Verwendung des Begriffs soll keine Beziehungen zu anderen Forschungsfeldern, insbesondere im Bereich der Sprach- oder Kommunikationswissenschaft, knüpfen. Er wird hier in seiner einfachsten Form verwendet.

## 1.1 Verwendete Notationen

Die verwendeten Notationen für Klassen- und Objektdiagramme sind der Object Modeling Technique (OMT) [RBP+91] entlehnt. An diesen Notationen werden allerdings leichte Modifikationen, wie sie in [GHJV95] beschrieben sind, vorgenommen. Nachfolgend werden die verwendeten Notationen kurz beschrieben.

### 1.1.1 Klassendiagramm

Eine Klasse wird als Rechteck dargestellt. Der Klassenname steht in Fettschrift im oberen Bereich des Rechtecks. Handelt es sich um eine abstrakte Klasse, wird der Name kursiv dargestellt. Die wichtigen Methoden stehen, durch eine Trennlinie abgeteilt, unterhalb des Klassennamens. Abstrakte Methoden werden, wie abstrakte Klassen, durch kursiv dargestellte Namen gekennzeichnet. Darunter, ebenfalls durch eine Linie getrennt, erscheinen die Instanzvariablen der Klasse. Typinformationen zu den Methoden oder Instanzvariablen sind optional. Gemäß der C++-Konvention werden die Typinformationen vor den Namen der Methode, der Instanzvariablen oder des Parameters gestellt. Kursiv dargestellte Typinformationen zeigen, daß der Typ oder die Methode abstrakt sind.

Vererbungsbeziehungen zwischen Klassen werden durch eine Dreiecksverbindung, d.h. eine Linie, die von einem Dreieck unterbrochen wird, dargestellt. Die Spitze des Dreiecks zeigt dabei auf die Oberklasse. Eine sogenannte Besitzt-Beziehung, d.h. eine Exemplar einer Klasse ist Bestandteil einer Exemplar einer anderen Klasse, wird durch einen Pfeil mit einem Diamant-Symbol am Anfang symbolisiert. Der Pfeil ist dabei auf die Klasse gerichtet, deren Exemplar besessen wird. Eine Benutzt-Beziehung, d.h. eine Klasse referenziert eine andere Klasse, wird hingegen nur durch einen Pfeil dargestellt. Hier zeigt der Pfeil auf die Klasse, die referenziert wird. Als Erweiterung zur OMT-Notation kann zusätzlich dargestellt werden, daß Klassen Exemplare anderer Klassen erzeugen können. Symbol hierfür ist ein Pfeil mit gestrichelter Linie. Der Pfeil zeigt auf die Klassen, von der ein Exemplar erzeugt werden kann.

Können mehrere Exemplare benutzt, besessen oder instanziiert werden, so wird dies durch einen ausgefüllten Kreis vor der Pfeilspitze deutlich gemacht.

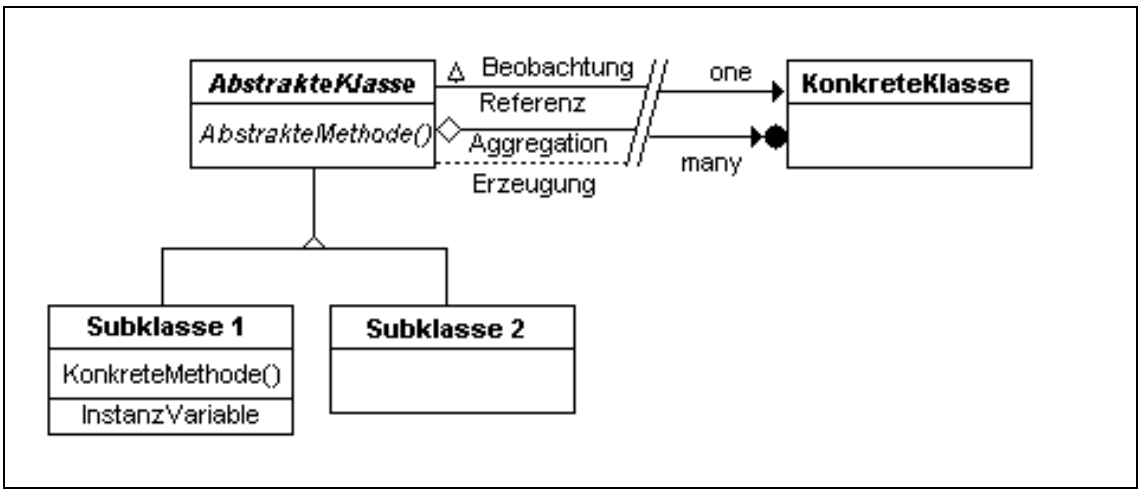


**Abbildung 1. Notation für Klassendiagramme nach [GHJV95]**

Als zusätzliche Erweiterung zur OMT-Notation macht eine stilisierte Glocke über dem Beziehungspfeil eine Beobachtungsbeziehung [GHJV95] deutlich. Die Klasse am Beginn des Pfeils beobachtet dabei die Klasse am Ende des Pfeils.

### 1.1.2 Objektdiagramm

Ein Objektdiagramm zeigt ausschließlich Exemplare und kann als solches nur als Momentaufnahme verstanden werden. Objekte werden dabei als Rechtecke mit abgerundeten Ecken dargestellt. Der Objektname steht dabei in Fettschrift im oberen Teil des Rechtecks. Darunter stehen, durch eine Linie vom Namen abgetrennt, die Referenzen auf andere Objekte. Diese Referenzen werden durch Pfeile zu anderen Objektsymbolen dargestellt. Weitere Darstellungsmöglichkeiten sind nicht vorgesehen.

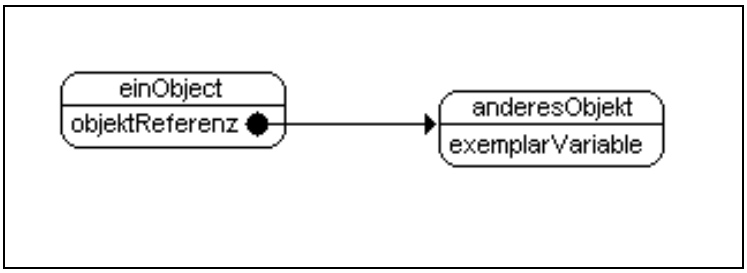


**Abbildung 2. Notationen für Objekt-Diagramme nach [GHJV95]**

### 1.1.3 Interaktionsdiagramm

Ein Interaktionsdiagramm zeigt den zeitlichen Verlauf einer Interaktion von verschiedenen Objekten. Der Zeitverlauf wird dabei entlang der senkrechten Achse verdeutlicht.

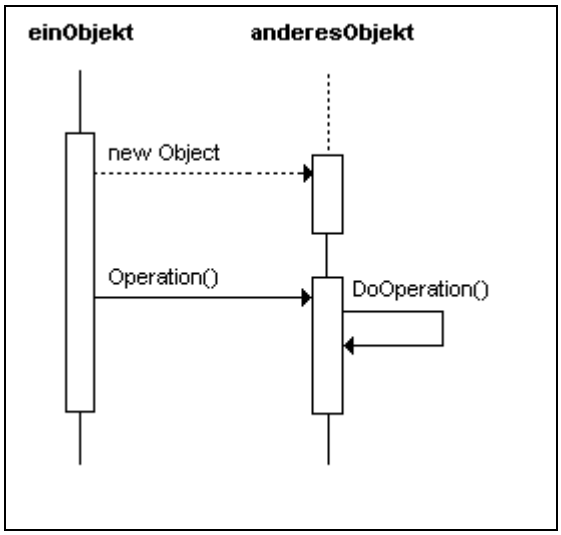


**Abbildung 3. Notation für Interaktionsdiagramme nach [GHJV95]**

Für jedes Objekt gibt es eine eigene Zeitachse, wobei der Verlauf der Zeit auf allen Achsen gleich dargestellt wird. Entlang einer Zeitachse werden die Phasen, in denen ein Objekt aktiv ist, durch Rechtecke dargestellt. Aktiv bedeutet in diesem Fall, daß innerhalb des Objektes eine Methode abgearbeitet wird. Der Aufruf von Methoden der einzelnen Objekte an anderen Objekten oder sich selbst wird durch Pfeile dargestellt.

# 2 Struktur einer Interaktion nach der Werkzeug-Material-Metapher

Die gesamte Arbeit bewegt sich im Kontext einer speziellen Vorgehensweise für die Entwicklung objektorientierter Software. Diese Vorgehensweise und deren Umgang mit Interaktionen zwischen Benutzer und Applikation bildet die Grundlage aller weiteren Diskussionen im Rahmen dieser Arbeit. Daher soll dieses Kapitel dazu dienen, den derzeitigen Stand darzustellen und zu diskutieren. Zunächst stelle ich kurz die Vorgehensweise vor. Anschließend betrachte ich, welchen Aufbau und Ablauf diese Vorgehensweise benutzt, um Interaktionen zwischen Benutzer und Applikation abzubilden. Dazu gehört auch eine kurze Erläuterung der Struktur des benutzten Fenstersystems Motif. Abschließend diskutiere ich, welchen Einschränkungen und Schwächen das derzeitige Konzept der Interaktionsabbildung unterliegt.

## 2.1 Die Werkzeug-Material-Metapher

Im Arbeitsbereich Softwaretechnik des Fachbereichs Informatik der Universität Hamburg wird für die objektorientierte Programmierung eine Vorgehensweise benutzt, die sich kurz mit den Begriffen Werkzeug, Automat und Material, kurz WAM, charakterisieren läßt. Ausgehend von einem Leitbild für die Entwicklung, dem Arbeitsplatz für qualifizierte menschliche Arbeit, in dem die Anwender als Experten für ihren Bereich angesehen und daher in die Entwicklung einbezogen werden, werden in einem mehrstufigen evolutionären Prozeß verschiedene Dokumente erstellt. Jedes Dokument stellt dabei den gegenwärtigen Wissenstand in einem Teilbereich des Entwicklungsprozesses dar. Die Form der Dokumente geht dabei von Prosatext für die Beschreibung elementarer Handlungsabläufe bis zu ablauffähigen Prototypen. Ziel dieser Vorgehensweise soll ein durchgängiger Prozeß von der Identifizierung des Anwendungsbereichs bis zum einsatzfähigen Softwaresystem sein.

Ein derart konstruiertes System besteht aus einer Kombination verschiedener Bestandteile. Gemäß der Entwicklungsmetapher besteht das System aus Werkzeugen, Automaten und Materialien. Werkzeuge bieten dem Anwender die Möglichkeit mit Materialien umzugehen. Die Form des Umgangs wird dabei von den Fähigkeiten des Werkzeugs bestimmt. Beispiele für Werkzeuge sind Browser zum Betrachten, Editoren zum Verändern und Auflister zum Auswählen von Materialien. Materialien hingegen sind diejenigen Dinge, die im Arbeitsablauf zum Arbeitsgegenstand werden. Materialien sind passiv, d.h. Aktionen mit oder an einem Material werden immer von einem Werkzeug durchgeführt. Ein Automat kapselt einen routinisierten Ablauf, der vom Anwender nur noch angestoßen werden muß und dann selbständig abläuft. Anpassungen an den jeweiligen Anwendungskontext lassen sich durch Veränderung der Voreinstellungen des Automaten durchführen. Eine ausführliche Beschreibung dieser Vorgehensweise findet sich in [KGZ94] und [Gryc96].

## 2.2 Architektur einer Interaktion in WAM

Eine Interaktion zwischen Anwender und einem nach WAM konstruierten Softwaresystem kann nur über die verfügbaren Werkzeuge des Softwaresystems stattfinden (siehe Kap. 2.1). Eine Betrachtung der Architektur einer Interaktion in einem nach WAM konstruierten System kann sich daher auf die Werkzeuge beschränken.

Werkzeuge werden nicht als monolithischer Block konstruiert. Statt dessen findet eine Trennung zwischen dem funktionsspezifischen, genannt Funktionskomponente oder FK,

und dem interaktionsspezifischen Teil, genannt Interaktionskomponente oder IAK, statt. Der funktionsspezifische Teil enthält dabei die Funktionalität des Werkzeugs sowie die Verbindung zu den benutzten Materialien. Der interaktionsspezifische Teil beschreibt die Darstellung des Werkzeugs auf der Arbeitsoberfläche und die Reaktionen auf Manipulationen durch den Benutzer. Beide Teile sind nicht fest miteinander verbunden. Die Verbindung zwischen funktionsspezifischem und interaktionsspezifischem Teil erfolgt über einen sogenannten Beobachtermechanismus [GHJV95]. In unserem Fall übernimmt der interaktionsspezifische Teil die Rolle des Beobachters und der funktionsspezifische Teil die Rolle des Beobachtbaren. Der funktionsspezifische Teil des Werkzeugs ist somit nicht unmittelbar an einer Interaktion beteiligt und daher in der folgenden Betrachtung auch nicht interessant.

Ein Werkzeug besteht aus einer **Funktionskomponente**, die die Funktionalität des Werkzeugs enthält, und einer oder mehrerer **Interaktionskomponenten**, die die Darstellung des Werkzeugs an der Benutzungsschnittstelle beschreibt.

Abbildung 4 zeigt, in welcher Verbindung ein Werkzeug bzw. seine Bestandteile mit der Benutzungsschnittstelle stehen. Die Interaktionskomponente beinhaltet demnach nicht die gesamte Behandlung der Interaktion mit dem Benutzer. Sie ist vielmehr nur ein Glied in einer Kette von Bestandteilen. Der Benutzer sieht nur seine Benutzungsoberfläche, hinter der ein bestimmtes Fenstersystem steht, z.B. Microsoft Windows® oder die Workplace Shell (OS/2). Ein Fenstersystem besteht dabei aus einer Sammlung von Objekten, die auf der Oberfläche sichtbar sind, den sogenannten Widgets oder Oberflächenobjekte. Zusätzlich muß das Fenstersystem über

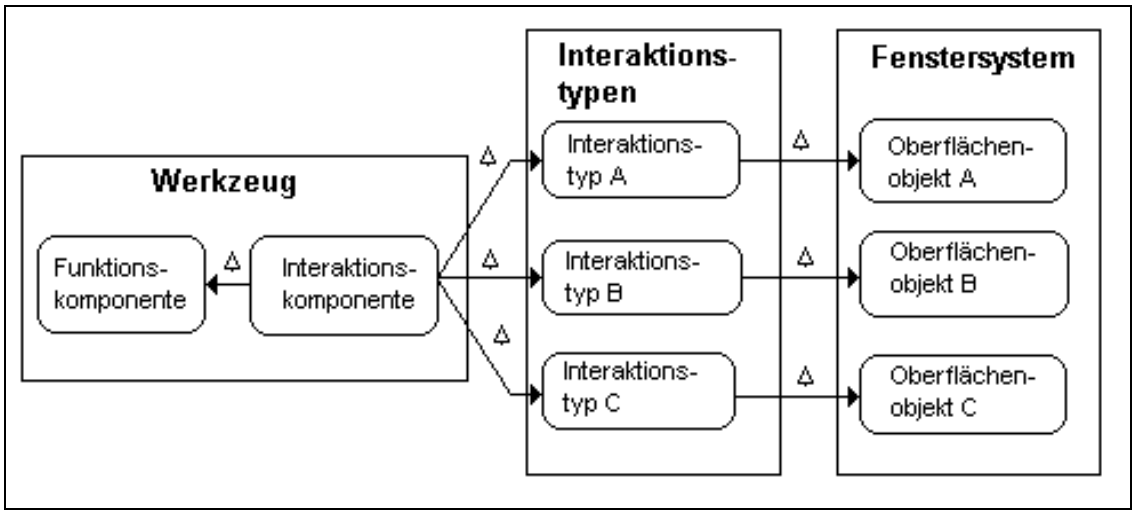


**Abbildung 4. Werkzeug-Konstruktion in WAM**

Möglichkeiten verfügen diese Objekte zu verwalten. Der Benutzer manipuliert auf seiner Oberfläche nun die Oberflächenobjekte oder deren Inhalte. Da das Fenstersystem kein Anwendungs- sondern ein Systemprogramm ist, muß es eine Schnittstelle bieten, daß Anwendungsprogramme unter Nutzung des Fenstersystems mit dem Benutzer in Kontakt treten können und von den Manipulationen des Benutzers erfahren. An dieser Stelle setzen Interaktionstypen-Bibliotheken an. Sie nutzen die Schnittstelle des Fenstersystems, um dessen Möglichkeiten in einer bestimmten Programmiersprache verfügbar zu machen. Jedes Oberflächenobjekt wird dabei von einem passenden

Interaktionstyp beobachtet. Die Interaktionskomponente beobachtet ihrerseits alle Interaktionstypen, die die Benutzungsschnittstelle des Werkzeugs ausmachen.

**Interaktionstypen** kapseln die Programmierschnittstelle des Fenstersystems. Jede Interaktionstypen-Klasse kapselt ein Oberflächenobjekt. Ein Interaktionstypen-Exemplar beobachtet das zugehörige Oberflächenobjekt der Benutzungsschnittstelle.[3]

Aufgrund der Manipulationen des Benutzers löst das Fenstersystem Ereignisse aus, die Art und Ort der Manipulation näher beschreiben. Betrifft ein Ereignis ein Oberflächenobjekt der Benutzungsschnittstelle des Werkzeugs, landet es bei dem beobachtenden Interaktionstyp. Dieser behandelt das eintreffende Ereignis und löst gegebenenfalls ein eigenes Ereignis aus. Die beobachtende Interaktionskomponente nimmt das Ereignis entgegen und bearbeitet es entsprechend des Anwendungskontextes.

## 2.3 Das Fenstersystem Motif

Unter dem Betriebssystem Unix, daß in seiner ursprünglichen Form ohne grafische Benutzungsoberfläche auskommen muß, wird das Fenstersystem durch eine, auf dem Unix-Kern aufbauende, Betriebssystemerweiterungen namens X11 realisiert. X11 stellt dabei die grundlegenden Funktionen für grafische Anwendungen zur Verfügung.

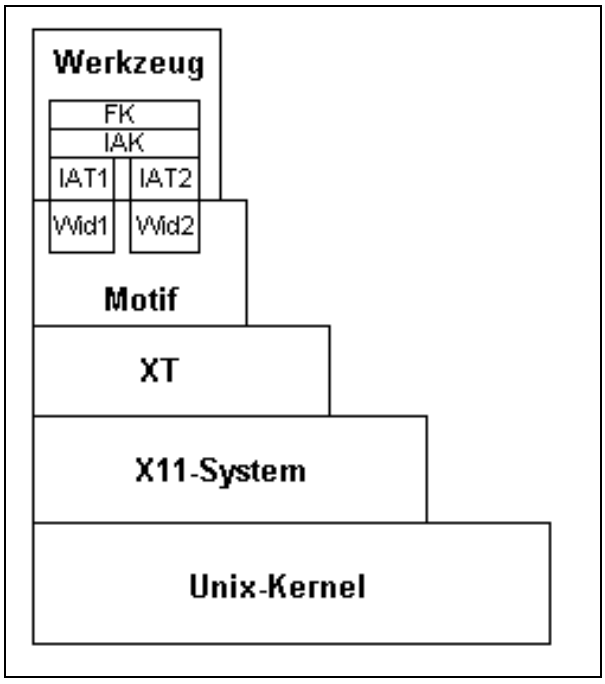


**Abbildung 5. Aufbau eines Motif-Programms**

Eine Grundmenge von Oberflächenobjekten, die auf den Funktionen, die X11 zur Verfügung stellt, basieren, bildet das sogenannte XT. Da die Oberflächenobjekte von XT nur mit viel Aufwand zu programmieren sind, gibt es eine Vielzahl von Bibliotheken, die eine vereinfachte Programmierschnittstelle zur Verfügung stellen. In den meisten Fällen erweitern diese Bibliotheken die Menge der verfügbaren

[3] Der Begriff Interaktionstyp verändert seine Bedeutung durch die Einführung von Interaktionsformen (siehe Kapitel 2.5.1).

Oberflächenobjekte und führen für alle Objekte ein spezifisches Aussehen ein. Eine dieser Bibliotheken ist Motif.

Im Kontext dieser Arbeit existiert eine Interaktionstypen-Bibliothek für die Verwendung von Motif. Die Bestandteile einer solchen Interaktionstypen-Bibliothek bilden meist die zur Verfügung stehenden Oberflächenobjekte und die Verwaltungskonstrukte des Fenstersystems nach.

Abbildung 5 zeigt die Hierarchie der einzelnen Bestandteile und die Einbettung der WAM-Architektur. Dabei wird offensichtlich, daß die Einflußmöglichkeiten des Benutzers von Motif bei den Interaktionstypen endet.

## 2.4 Ablauf einer Interaktion in WAM

Die Oberfläche eines Werkzeugs setzt sich aus der Kombination verschiedener Interaktionstypen zusammen. Da Interaktionstypen nur das Fenstersystem kapseln, haben sie kein anwendungsspezifisches Wissen. Daher werden die eventuell für die Anwendung interessanten Informationen nur durchgeschleust bzw. an die Gegebenheiten angepaßt und weitergeleitet. Am Ende der Kette steht nun die Interaktionskomponente. Sie kennt alle beteiligten Interaktionstypen und nimmt die von ihnen kommenden Informationen entgegen oder fordert von ihnen bestimmte Aktionen an der Oberfläche auszulösen. So erfährt sowohl das Werkzeug von den Manipulationen, die der Benutzer an der Oberfläche vorgenommen hat, als auch die Oberfläche von vorzunehmenden Veränderungen an der Darstellung, die durch die Reaktionen des Werkzeugs nötig geworden sind.

Ausgangspunkt der Interaktionen ist bei reaktiven Softwaresystemen stets eine Eingabe durch den Benutzer an einem der verfügbaren Werkzeuge. Unterscheiden muß man dabei allerdings zwischen den Eingaben, die eine Reaktion des Werkzeugs erforderlich machen, und den Eingaben, die ohne Reaktion des Werkzeugs innerhalb eines Oberflächenobjektes behandelt werden. Ein Beispiel für eine Eingabe, die ohne Reaktion des Werkzeugs behandelt wird, ist die Verschiebung des Fokus auf das Fensters, in dem das Werkzeug dargestellt wird. Der Fokus bestimmt das Fenster auf der Oberfläche, das die Eingaben durch den Benutzer entgegennimmt. Das Fenster zählt bereits zu den Oberflächenobjekten, die dem Werkzeug zugeordnet werden. Die Verschiebung des Fokus ermöglicht es aber dem Benutzer erst, mit dem Werkzeug zu interagieren. Für die Betrachtung des Ablaufs einer Interaktion sind nur die Eingaben interessant, die eine Reaktion des Werkzeugs erfordern.

Erfolgt eine entsprechende Eingabe durch den Benutzer, löst das betroffene Oberflächenobjekt eine Nachricht aus. Mit dem Oberflächenobjekt ist ein Interaktionstypen-Objekt verbunden, das die ausgelöste Nachricht entgegen nimmt. Wenn die Nachricht eine Reaktion des Werkzeugs, in dem sie ausgelöst wurde, erfordert, wird sie vom Interaktionstypen-Objekt weitergeleitet.

Alle Interaktionstypen-Objekte, die die Darstellung des Werkzeugs an der Benutzungsoberfläche ausmachen, werden von einer Interaktionskomponente (IAK) kontrolliert. Alle weitergeleiteten Nachrichten landen in dieser IAK. Dazu ruft sie an der zugehörigen Funktionskomponente (FK) die geeigneten Methoden auf, um auf die Nachricht zu reagieren. Führt die Reaktion auf die Nachricht zu Veränderungen, die sich

an der Benutzungsoberfläche widerspiegeln sollen, signalisiert die FK der IAK über eine weitere Nachricht, daß sich etwas geändert hat. Da die IAK aus der Nachricht nicht entnehmen kann, was sich geändert hat, ruft sie an der FK sondierende Methoden auf, um sich so über den derzeitigen Zustand des Werkzeugs zu informieren. Die Informationen werden über den Aufruf manipulierender Methoden an den betroffenen Interaktionstypen-Objekten an diese weitergeleitet. Anschließend werden die Interaktionstypen-Objekte aufgefordert sich mit dem aktualisierten Zustand neu auf der Benutzungsoberfläche zu zeichnen.

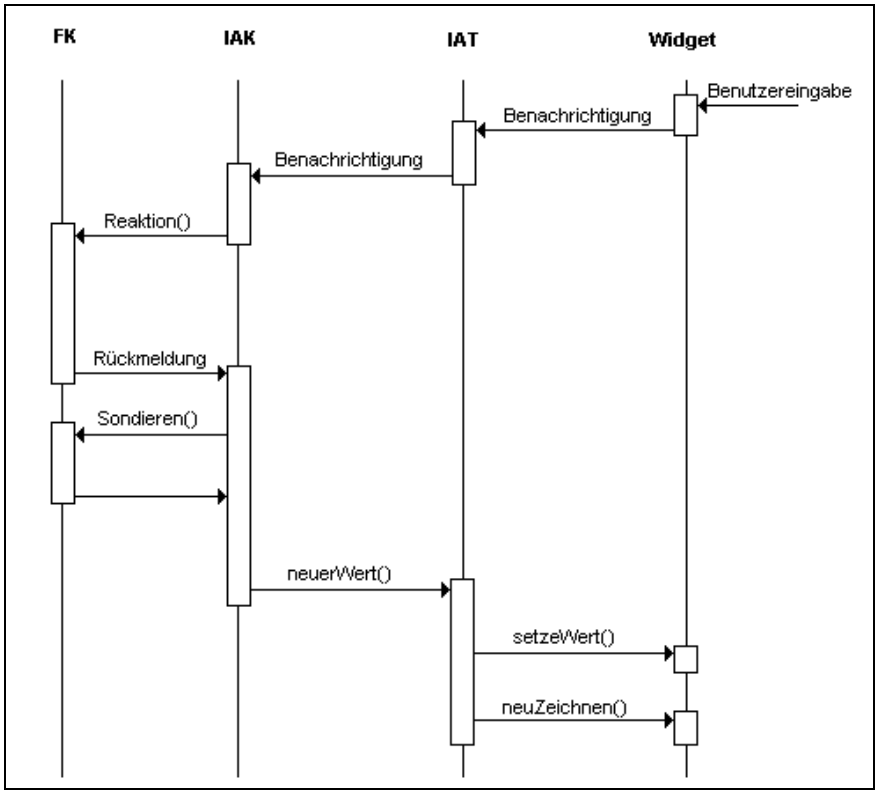


**Abbildung 6. Interaktionsdiagramm einer Benutzereingabe**

Analog zum Weg von der Benutzereingabe zu den Interaktionstypen-Objekten wird das erneute Zeichnen des Oberflächenobjektes über eine entsprechende Nachricht angefordert. In diesem Fall wird die Nachricht aber von den Interaktionstypen-Objekten an das Fenstersystem geschickt.

Abbildung 6 zeigt in vereinfachter Form, wie eine Interaktion, in diesem Fall eine Benutzereingabe, unter WAM abläuft. Vom Fenstersystem wird dabei eine Nachricht ausgelöst, die vom beobachtenden Interaktionstyp aufgenommen und interpretiert wird. Da es sich hier um eine Nachricht handelt, die weitere Aktionen des Werkzeugs erforderlich macht, setzt der Interaktionstyp die Nachricht so um, daß die Interaktionskomponente sie verarbeiten kann. Die Interaktionskomponente entscheidet, daß die zugehörige Funktionskomponente tätig werden muß. Dazu ruft die Interaktionskomponente an der Funktionskomponente eine geeignete Methode auf. Da die Arbeit der Funktionskomponente zu Veränderungen geführt hat, die eine Aktualisierung der Darstellung des Werkzeugs nötig machen, teilt sie dies der Interaktionskomponente über eine Nachricht mit. Die Nachricht signalisiert dabei lediglich, daß eine Veränderung stattgefunden hat. Die Details der Veränderung muß die

Interaktionskomponente über den Aufruf sondierender Methoden an der Funktionskomponente in Erfahrung bringen.

Im dargestellten Beispiel hat sich ein Wert geändert, der an der Benutzungsschnittstelle des Werkzeugs dargestellt wird. Die Interaktionskomponente versorgt daher den betroffenen Interaktionstyp mit dem aktuellen Wert, den dieser an das zugehörige Oberflächenobjekt weitergibt. Um den aktuellen Wert an der Benutzungsschnittstelle sichtbar zu machen, fordert der Interaktionstyp abschließend das Oberflächenobjekt noch dazu auf, sich neu zu zeichnen.

## 2.5 Schwächen des Konzeptes

Um geeignet auf die Ereignisse reagieren zu können, die von der Oberfläche ausgelöst werden, beobachtet die Interaktionskomponente jeden ihrer Interaktionstypen. Sie entscheidet für jedes Ereignis, ob und in welcher Weise das Programm reagiert. Gegebenenfalls ruft die Interaktionskomponente eine Methode an der Schnittstelle eines Interaktionstypen auf, um sich Informationen zu besorgen oder weitere Aktionen auszulösen. Damit enthält die Interaktionskomponente bereits Informationen darüber, aus welchen Bestandteilen sich die Benutzungsschnittstelle des Werkzeugs zusammensetzt.

Die direkte Verbindung zwischen Interaktionskomponente und Interaktionstypen beinhaltet dabei drei Designentscheidungen, die zu verschiedenen Zeitpunkten der Werkzeugentwicklung getroffen werden und unterschiedlicher Änderungshäufigkeit unterliegen.

- Die erste Entscheidung[4] betrifft die Einflußmöglichkeiten des Benutzers auf das Werkzeug. Sie legt aus fachlicher Sicht fest, welcher Form die Interaktion an dieser Stelle sein soll, z.B. soll eine Auswahl getroffen werden. Diese Festlegungen bezeichne ich im folgenden als **fachlichen Bestandteil der Interaktion**. Die Entscheidungen über die Einflußmöglichkeiten des Benutzers unterliegen während des Werkzeugentwurfs häufigen Änderungen, werden danach aber kaum noch in Frage gestellt.
- Die zweite Entscheidung betrifft den Entwurf der Benutzungsschnittstelle des Werkzeugs. Hier wird festgelegt, in welcher Form sich die Interaktion an der Werkzeugoberfläche präsentiert, z.B. stellt sich eine Auswahl als Listbox dar. Das Aussehen der Benutzungsschnittstelle ändert sich meist während des gesamten Entwicklungsvorgangs und teilweise auch noch darüber hinaus. Einfluß auf die Interaktionskomponente haben allerdings nur Änderungen, bei denen Teile der Benutzungsschnittstelle durch andere Oberflächenobjekte ersetzt werden. Die Festlegungen auf ausgewählte Oberflächenobjekte, die die Benutzungsschnittstelle bilden sollen, bezeichne ich im folgenden als **technischen Bestandteil der Interaktion**.
- Die dritte Entscheidung betrifft die Auswahl, welches Fenstersystem oder gar welches Betriebssytem für das Werkzeug verwendet werden soll bzw. in welchem Umfeld das Werkzeug laufen soll. Diese Entscheidung liegt oft schon vor Beginn des eigentlichen Entwicklungsvorgangs fest. Allerdings verstärken sich die Tendenzen, Werkzeuge in mehr als einer Umgebung laufen lassen zu können. Betrifft diese

[4] Die genannte Reihenfolge ist nur als Aufzählung zu verstehen. Eine zeitliche Folge soll damit nicht dargestellt werden.

Entscheidung auch die Wahl des Betriebssystems, ist zu beachten, daß für die Verwendbarkeit des Werkzeugs unter verschiedenen Betriebssystemen auch andere Bestandteile des Werkzeugs entscheidend sein können.

Die Interaktion zwischen Benutzer und Werkzeug besteht aus einem **fachlichen** und einem **technischen Teil**. Der fachliche Bestandteil legt die Einflußmöglichkeiten des Benutzers fest, während der technische Bestandteil aussagt, welche Oberflächenobjekte für die Darstellung der Interaktion benutzt werden.

Durch die Festlegung des Aussehens der Benutzungsschnittstelle und des zu benutzenden Fenstersystems innerhalb der Interaktionskomponente wird diese allerdings in ihrer Verwendbarkeit stärker als nötig eingeschränkt. Veränderungen an der Oberfläche, die über einfache Reorganisation der vorhandenen Oberflächenobjekte hinaus gehen, erzwingen Veränderungen am Code der Interaktionskomponente. Entsprechendes gilt für den Wechsel des benutzten Fenstersystems. Um der Funktionalität einer Interaktionskomponente, die Präsentation des Werkzeugs und die Bearbeitung von Ereignissen, die vom Benutzer ausgelöst werden, gerecht zu werden, sind diese Informationen nicht nötig.

## 2.6 Zusammenfassung

Zunächst habe ich in diesem Kapitel den derzeitigen Stand der Abbildung von Interaktionen zwischen Benutzer und Applikation in WAM dargestellt. Die abschließende Diskussion über die Einschränkungen und Schwächen dieser Struktur hat dann gezeigt, daß die enge Kopplung zwischen dem fachlichen und dem technischen Teil der Interaktion, wie sie in WAM benutzt wird, die flexible Verwendbarkeit der Applikation unter verschiedenen Fenstersystemen oder sich verändernden Oberflächen einschränkt. Um diese Einschränkungen zu umgehen, ist eine Veränderung der Abbildung von Interaktionen nötig. Eine Möglichkeit der Veränderung sind die sogenannten **Interaktionsformen**, die im folgenden Kapitel eingeführt werden.

# 3 Interaktionsformen

Kapitel 2 hat gezeigt, daß die derzeitige Abbildung von Interaktion zwischen Benutzer und Applikation Einschränkungen unterliegt. In diesem Kapitel führe ich mit den **Interaktionsformen** ein Konzept ein, daß diese Einschränkungen umgehen soll. Neben der Einführung des Begriffs bildet die Identifikation der Interaktionsformen den Hauptteil des Kapitels. Dazu untersuche ich jeweils eine Applikation mit grafischer Benutzungsoberfläche und ein Fenstersystem auf ihre Interaktionsmöglichkeiten. Anhand der Untersuchungsergebnisse identifiziere ich anschließend die Interaktionsformen.

## 3.1 Der Begriff Interaktionsform

Wie in Kap. 2.4 ausgeführt, beinhaltet die bisher in WAM benutzte Form der Konstruktion einige Einschränkungen im Hinblick auf die Flexibilität in der Verwendung des Werkzeugs. Gesucht wird daher eine Möglichkeit die Informationen über die Interaktion, die die Interaktionskomponente benötigt, auf ihren fachlichen Anteil, also Voraussetzungen und Ergebnisform, zu reduzieren.

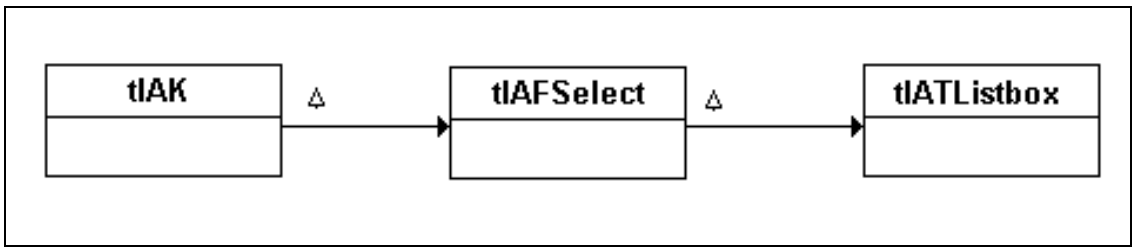


**Abbildung 7. Verbindung von IAK und IAT mit Interaktionsformen**

Eine derartige Abstraktion über die möglichen Interaktionen führt dazu, daß mehrere Interaktionstypen in der Lage sein können, die fachlichen Anforderungen an die Interaktion zu erfüllen. Daher muß die Abstraktion so einerseits angelegt sein, daß sie von Interaktionstypen unabhängig und die Verbindung zwischen Interaktionskomponente und Interaktionstyp möglichst locker und spät festlegbar ist. Andererseits darf sie auch kein Anwendungswissen enthalten, muß also kontextfrei sein. Da sich eine solche Abstraktion damit auseinandersetzt, die Form einer Interaktion zu identifizieren, nenne ich sie folgenden **Interaktionsform**. In den folgenden Kapiteln soll ein mögliches Konzept für die Nutzung von Interaktionsformen bei der Werkzeugkonstruktion nach der Werkzeug-Material-Metapher vorgestellt werden.

## 3.2 Identifizierung sinnvoller Interaktionsformen

Ein wichtiger Schritt zur Verwendung von Interaktionsformen ist die Identifizierung von sinnvollen und wiederverwendbaren Abstraktionen von den möglichen Interaktionen, die zwischen Benutzer und Anwendungsprogrammen vorkommen können. Ein möglicher Weg solche Abstraktionen zu identifizieren, wären arbeitspsychologische Untersuchungen über Mensch-Maschine-Interaktionen. Eine solche Untersuchung würde aber den Rahmen dieser Arbeit sprengen und wird daher nicht durchgeführt.

Eine mögliche Alternative ist die exemplarische Untersuchung bestehender Programme unter dem Gesichtspunkt, welche Oberflächenobjekte darin Verwendung finden und welche Interaktion mit dem Programm durch sie ausgelöst wird. Eine weitere

Alternative ist die Untersuchung der möglichen Interaktionen, wie sie über die überhaupt vorhandenen Oberflächenobjekte dem Benutzer von der Oberfläche vorgegeben werden. Eine Zusammenfassung der Gemeinsamkeiten in der Nutzung der verschiedenen Oberflächenobjekte liefert dabei nicht nur mögliche Abstraktionen, sondern gibt auch erste Hinweise darauf, welche Interaktionstypen später mit welcher Interaktionsform zusammenarbeiten müssen, um die Möglichkeiten der Oberfläche nicht unnötig einzuschränken. Um die Abstraktion auf eine möglichst breite Basis zu stellen und ausreichend Hinweise auf mögliche Interaktionsformen zu erhalten, werden im folgenden beide Untersuchungen durchgeführt und deren Ergebnisse zusammengefaßt.

### 3.2.1 Exemplarische Untersuchung bestehender Programme

Diese Untersuchung soll herausfinden, welche Oberflächenobjekte für welche Interaktionen benutzt werden. Dafür betrachte ich ein bestehendes Programm, das auf einer grafischen Benutzungsoberfläche basiert. Verwendet wird das Programm TextPad 2.3. Als Oberfläche wird Windows 95™ verwendet.

Das ausgewählte Programm ist ausschließlich als Beispiel für eine Applikation mit grafischer Benutzungsoberfläche zu verstehen. Es dient in diesem Zusammenhang lediglich als Grundlage für die Untersuchung. Aufbau und Struktur des Programms basieren nicht auf WAM und sind hier auch nicht Gegenstand der Untersuchung. Als Grundlage für die folgende Betrachtung dient ein Bildschirmfoto (Abbildung 8). Es zeigt ein TextPad-Fenster mit eingeschalteter Auswahl und dem Einstellungsdialog, positioniert auf die Einstellungen für Dateien. Im folgenden untersuche ich die einzelnen Bestandteile der gezeigten Fenster nach ihren Bedeutungen für die Interaktion zwischen Benutzer und Programm.

Wie bei jedem Programm, das auf einer grafischen Benutzungsoberfläche basiert, dient ein Fenster als oberstes Oberflächenobjekt. Innerhalb des Fensters gruppieren sich die einzelnen Oberflächenobjekte, die dem Benutzer überhaupt erst die Interaktion mit dem Programm ermöglichen. Auch Fenster können derartige Oberflächenobjekte sein. Man spricht dann von abhängigen oder Unterfenstern. Im Unterschied zu den anderen Oberflächenobjekten sind abhängige Fenster aber nicht zwangsläufig auf die Ausmaße des übergeordneten Fensters beschränkt. Sie können frei auf der Benutzungsoberfläche positioniert werden. Das höchstrangige Fenster einer Anwendung wird Hauptfenster genannt. In Abbildung 8 sind drei Fenster erkennbar. Die Fenster Auswahl und Einstellungen sind dabei vom Fenster TextPad abhängig. Aus der Sicht des Benutzers bilden die Fenster aber nur die Möglichkeiten der Positionierung und Größenänderung, die beide keine Auswirkungen auf den Programmablauf haben. Eine Interaktion mit dem Programm findet also nicht statt.

Nachfolgend betrachte ich zunächst die Interaktionsmöglichkeiten, die das Hauptfenster (TextPad) dem Benutzer bietet. Im oberen Teil des Hauptfensters befindet sich eine Menüleiste. Die Einzelteile der Menüleiste lassen sich durch anklicken auswählen und offenbaren ein vorher verborgenes Menü. Auch innerhalb dieser Menüs, die zum Teil kaskadiert, also mehrstufig, sind, lassen sich die einzelnen Teile durch anklicken auswählen, wobei die Auswahl eines Menüpunktes eine direkte Reaktion durch das Programm bewirkt. Der gesamte Menübereich dient also der Auswahl aus einer Menge möglicher Aktionen.

Unterhalb der Menüleiste sowie an ihrem rechten Rand stehen diverse Schaltflächen bereit. Ähnlich der Funktionalitäten, die über die Menüs erreichbar sind, kann der Benutzer hier direkte Reaktionen im Programm, an der Oberfläche oder im Betriebssystem auslösen. Auch hier findet also eine Auswahl statt, wobei die Einzelteile dem Auslösen von Kommandos gelten.

Direkt unter den Schaltflächen befindet sich der eigentliche Aktionsbereich des Programms, in dem mehrzeiliger Text angezeigt und editiert werden kann. Da dies die hauptsächliche Funktion des Programms ist, nimmt dieser Teil auch den bei weitem größten Platz im Fenster ein. Da der angezeigte Text nicht auf die Größe des Fensters beschränkt ist, stehen am unteren und rechten Rand jeweils eine Scrollbar zur Verfügung, mit der der sichtbare Textausschnitt verschoben werden kann.
Am unteren Rand des Hauptfensters befindet sich ein Bereich, in dem das Programm kontextsensitive Informationen anzeigt. Im allgemeinen wird dieser Bereich Statuszeile genannt. Sie bietet allerdings nur eine Kommunikation in eine Richtung. Dem Benutzer wird an dieser Stelle keine Interaktionsmöglichkeit geboten. Allerdings können die angezeigten Informationen Einfluß auf die weitere Interaktion des Benutzers mit dem Programm haben.

Als kleines, etwas abgesetztes Fenster (Auswahl) steht eine Liste mit den bereits geöffneten Dateien zur Verfügung. Aus dieser Liste kann mit einfachem Klick diejenige Datei ausgewählt werden, deren Inhalt im Anzeigebereich des Hauptfensters angezeigt werden soll. Auch hier haben wir es also mit einer Auswahl, diesmal aus einer Liste von Möglichkeiten, zu tun.

Das über dem Hauptfenster positionierte Fenster (Einstellungen) zeigt, neben den schon betrachteten Schaltflächen, im zentralen Bereich sogenannte Karteikarten. Diese

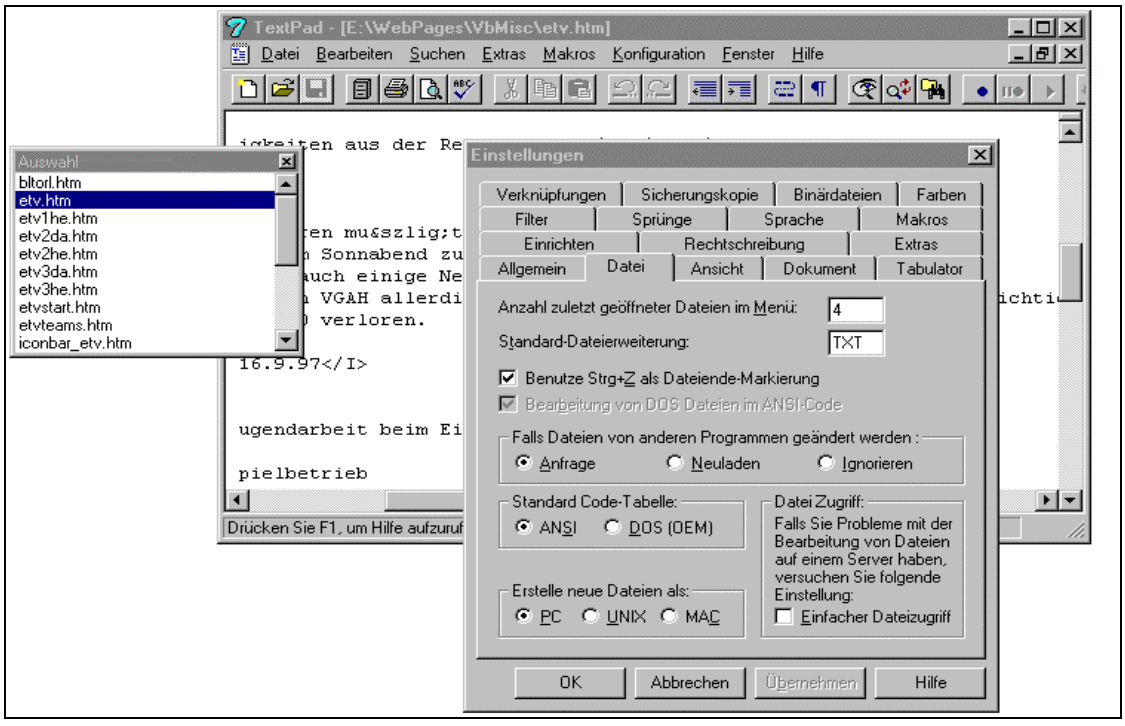


**Abbildung 8. Snapshot von TextPad 2.3**

gruppieren eine Menge von Oberflächenobjekten zu einer thematischen Einheit.

Karteikarten haben in ihrer Wirkung große Ähnlichkeit mit abhängigen Fenstern, besitzen jedoch keine eigenen Oberflächenobjekte für Fenstermanipulation. Diese werden vollständig in dem Fenster gehalten, das die Karteikarten enthält und wirken auf alle Karteikarten gleichermaßen, unabhängig davon, ob die Karteikarte im Vordergrund ist oder nicht. Alle Karteikarten eines Fensters haben die gleiche Größe. Zudem kann, im Gegensatz zu abhängigen Fenstern, immer nur eine Karteikarte zur Zeit im sichtbar sein. Die restlichen Karteikarten sind nur als sogenannte Karteikartenreiter repräsentiert. Diese Reiter wirken wie Schaltflächen. Ein Klick auf sie bringt die zugehörige Karteikarte zur Anzeige.

Die, in Abbildung 8 sichtbare, Karteikarte dient der Kontrolle des Verhaltens von TextPad bezüglich Dateien. Im oberen Teil befinden sich zwei einzeilige Eingabefelder. In diese Felder kann beliebiger Text eingegeben werden. Die Überprüfung des Feldinhalts auf Plausibilität findet erst in dem Moment statt, in dem die Karteikarte nicht mehr geschlossen oder in den Hintergrund gerückt wird. Die Eingabefelder können also Informationen in verschiedenen Formaten (z.B. numerisch oder alphanumerisch) aufnehmen. Unterhalb der Eingabefelder befinden sich zwei Checkboxen, von denen nur eine aktiv ist. Die Checkboxen sind entweder ein- oder ausgeschaltet. Sie repräsentieren eine Eigenschaft, die entweder aktiviert oder deaktiviert ist. Die Checkbox in der rechten unteren Ecke der Karteikarte besitzt die gleiche Wirkung auf eine weitere Eigenschaft. Der umgebende Rahmen dient nur der optischen Aufbereitung. Die Checkboxen ermöglichen dem Benutzer also eine binäre Entscheidung zu treffen.

Im mittleren und unteren, linken Bereich der Karteikarte befinden sich weitere Rahmen. Diese dienen allerdings einem anderen Zweck. Sie gruppieren die darin befindlichen Radioboxen zu sogenannten Radiogroups und werden daher Groupbox genannt. Dabei ist in jeder Radiogroup immer genau eine Radiobox ausgewählt. So kann sich der Benutzer beispielsweise entscheiden, ob eine von außen geänderte Datei sofort neu geladen wird, die Änderung ignoriert oder das weitere Vorgehen nachgefragt werden soll. Auch hier haben wir also eine Auswahl. Die Zahl der auswählbaren Möglichkeiten entspricht der Anzahl der Radioboxen in der Radiogroup.

Bisher noch nicht benannter Bestandteil der Oberfläche sind die Beschreibungen bzw. Beschriftungen, die an den einzelnen Oberflächenobjekten angebracht sind, um dem Benutzer zu ermöglichen, die Oberflächenobjekte zu identifizieren und eventuell Rückschlüsse auf die, mit ihnen verbundenen, Auswirkungen zu ziehen. Im allgemeinen werden solche Beschreibungen als Labels bezeichnet. Sie bieten dem Benutzer allerdings keine Interaktionsmöglichkeit, auch wenn es durchaus möglich ist, sie während des Programmablaufs zu ändern. Dies geht allerdings nur von seiten des Programms.

Die weiteren Karteikarten bieten keine zusätzlichen Oberflächenobjekte, so daß auf ihre Betrachtung verzichtet wird.

| Oberflächenobjekte | Funktionsbeschreibung |
|---|---|
| Fenster | Bildet den Rahmen für Oberflächenobjekte. Bietet dem Benutzer keine Interaktionsmöglichkeit mit dem Programm. |
| Rahmen | Optische Zusammenfassung von Oberflächenobjekten ohne Interaktionsmöglichkeit für den Benutzer. |

| | |
|---|---|
| Schaltfläche (Button) | Löst bei Betätigung den Ablauf einer festgelegten Handlung in Programm, Oberfläche oder Betriebssystem aus. |
| Menüleiste | Besondere Form von aneinandergereihten Schaltflächen, bei denen die Handlung im Öffnen eines Menüs besteht. Der Benutzer hat dabei keine Interaktionsmöglichkeit mit dem Programm. |
| Menü | Organisationsform für Menüpunkte. |
| Menüpunkt | Besondere Form von Schaltfläche, die nur in Menüs vorkommt. |
| Einzeiliges Eingabefeld | Ermöglicht dem Benutzer die Eingabe von Informationen, die vom Programm abgerufen und verändert werden können. Die Übernahme der Informationen erfolgt nicht automatisch. |
| Mehrzeiliges Eingabefeld | Ermöglicht die mehrzeilige Eingabe von Informationen. Funktionalität wie bei einzeiligen Eingabefeldern. |
| Scrollbars | Bietet dem Benutzer die Möglichkeit, den sichtbaren Ausschnitt eines Fensters, mehrzeiligen Eingabefeldes oder einer Listbox zu verändern. |
| Statuszeile | Bereich, in dem das Programm Informationen an den Benutzer weiterleiten kann, ohne daß dieser Interaktionsmöglichkeiten besitzt. |
| Listbox | Listenförmige Zusammenfassung von Auswahlmöglichkeiten. |
| Karteikartenreiter | Besondere Form von Schaltflächen, bei denen die Handlung darin besteht, die zugehörige Karteikarte in den Vordergrund zu holen. |
| Karteikarte | Fensterähnliche Organisationsform von Oberflächenobjekten. |
| Checkbox | Eingabefeld für eine binäre Eigenschaft. |
| Radiobox | Eingabefeld für eine Eigenschaft. Im Gegensatz zur Checkbox muß es sich nicht um eine binäre Entscheidung handeln. |
| Radiogroupbox | Organisationsform für zusammenhängende Radioboxen. Es muß immer eine der Radioboxen aus einer Gruppe ausgewählt sein. |
| Label | Beschriftung oder Beschreibung anderer Oberflächenobjekte. Ein Label ist nicht durch den Benutzer änderbar. |

**Tabelle 1. Vom Programm verwendete Oberflächenobjekte.**

Tabelle 1 faßt noch einmal die gefundenen Oberflächenobjekte zusammen und erläutert kurz ihre Funktion.

### 3.2.2 Untersuchung möglicher Interaktionen am Beispiel von Motif

Grafische Benutzungsoberflächen werden im Prinzip nach einem Bausteinsystem zusammengesetzt. Hierbei werden die vorhandenen Bausteine in unterschiedlichster Form kombiniert um den Anforderungen der darauf aufbauenden Programme gerecht zu werden. Teil dieser Bausteinsammlung sind die Oberflächenobjekte. Sie ermöglichen

dem Benutzer mit dem Programm Informationen auszutauschen. Diese Untersuchung soll für eine bestimmte Oberfläche die vorhandene Oberflächenobjekte unter dem Gesichtspunkt der möglichen Interaktion betrachten und zusammenfassen. Basis für die Untersuchung ist die Oberfläche Motif. Tabelle 2 zeigt die, unter Motif zur Verfügung stehenden, Oberflächenobjekte und ihre Funktion.

| Oberflächenobjekte | Funktionsbeschreibung |
|---|---|
| Window | Fungiert als Behälter für andere Oberflächenobjekte. Ermöglicht die Gruppierung von unterschiedlicher Oberflächenobjekte. |
| MainWindow | Window, das als Fenster auf der obersten Ebene dient. |
| Text | Einzeiliger, konstanter Text (Beschriftung). |
| Popup | |
| Button | Aktionsauslösende Schaltfläche. |
| Menuitem | Benannte Schaltfläche, die den Zugriff auf eine Aktionsauswahl ermöglicht. |
| Checkbutton / Checkbox | Schaltfläche, die eine binäre Auswahl repräsentiert. Ein Checkbutton ist entweder ausgewählt oder nicht. Mehrere Checkbutton können in einer Checkbox zusammengefaßt sein. |
| Radiobutton / Radiobox | Schaltfläche, die eine Auswahlmöglichkeit repräsentiert. Von einer zusammengefaßten Gruppe (Radiobox) von Radiobutton kann immer nur einer ausgewählt sein. |
| List | Ermöglicht gleichzeitige Darstellung einer Anzahl von Auswahlmöglichkeiten. |
| Menu | Aktionsauswahl. |
| Scrollbar | Ermöglicht bei Anzeigen, die über die Fenstergrenze hinaus gehen, den sichtbaren Ausschnitt zu verschieben. |
| Request | |
| Sedit | Einzeiliges Texteingabefeld. |
| Medit | Mehrzeiliges Texteingabefeld. |
| File | Standardisierter Dateidialog. |
| Hypertextbrowser/ -editor | Betrachter / Editor für Hypertext-Dokumente. |
| Frame | |

**Tabelle 2. Oberflächenobjekte in Motif**

Schon durch einen kurzen Blick auf die Funktionalität der einzelnen Oberflächenobjekte läßt sich zumindest eine Zweiteilung erkennen. So dienen die Oberflächenobjekte Window, MainWindow und Frame nicht unmittelbar der Interaktion mit dem Benutzer. Sie stellen vielmehr einen organisatorischen Rahmen für die Darstellung der eigentlichen Interaktionsbestandteile der Anwendung dar. Da ich mich an dieser Stelle mit der Interaktion zwischen Benutzer und Anwendung befasse, bleiben die oben genannten Oberflächenobjekte im weiteren außer Acht.

### 3.2.3 Identifizierung der Interaktionsformen

Ein Vergleich der beiden Untersuchungen ergibt einige Überschneidungen, obwohl sich die Untersuchungen über verschiedene Betriebs- und Fenstersysteme erstrecken (Windows95 bzw. Motif/Unix). Diese Überschneidungen geben erste Hinweise auf mögliche Abstraktionen. Des weiteren gibt es bei verschiedenen Oberflächenobjekten

große Übereinstimmungen in ihren Funktionsbeschreibungen. In den nachfolgenden Kapiteln fasse ich diese Übereinstimmungen und Überschneidungen zusammen und identifiziere darüber die einzelnen Interaktionsformen.

#### 3.2.3.1 Auswahl

Bei den Beschreibungen der Oberflächenobjekte erscheint auffällig oft der Begriff Auswahl. Tatsächlich ermöglichen die Oberflächenobjekte Checkbutton, Radiobutton, Listbox und Menu dem Benutzer eine Wahl zwischen verschiedenen Möglichkeiten zu treffen. Die Form der Auswahl unterscheidet sich dabei zwar von Objekt zu Objekt, die Anzahl der zur Verfügung stehenden Oberflächenobjekte läßt jedoch den Schluß zu, daß das Treffen einer Auswahl eine zentrale Rolle in der Interaktion zwischen Benutzer und Anwendung spielt. Die Beschreibung der untersuchten Anwendung bestätigt diese Annahme. An verschiedensten Stellen muß der Benutzer eine Wahl treffen. **Auswahl** identifiziere ich daher als eine Interaktionsform. Die Oberflächenobjekte Checkbox und Radiobox bzw. Radiogroupbox fallen nicht unter diese Interaktionsform, da sie nur der Gruppierung der, auf der Oberfläche sichtbaren, Check- bzw. Radiobuttons dienen.

#### 3.2.3.2 Aktion

Das direkte Auslösung einer Aktion wird von Oberflächenobjekten unmittelbar unterstützt, sei es unspezifiziert per Button oder mit festgelegter Semantik, wie beim Menubutton bzw. Menuitem. Auch innerhalb der Anwendung hat der Benutzer an verschiedenen Stellen die Möglichkeit, eine Aktion auszulösen. Da die Oberflächenobjekte kein Anwendungswissen haben, ist ihnen die auszulösende Aktion nicht bekannt. **Aktion** identifiziere ich daher ebenfalls als Interaktionsform.

#### 3.2.3.3 Eingabe

Die direkte Eingabe von Informationen als Text wird in Motif durch zwei Oberflächenobjekte unterstützt. SEdit (einzeilig) und MEdit (mehrzeilig) ermöglichen es dem Benutzer, in Form und Informationsgehalt nicht weiter spezifizierte Informationen einzugeben oder zu verändern. Der weitere Verwendungszweck der Informationen ist nicht bekannt. Innerhalb der Anwendung ermöglichen die verschiedenen Möglichkeiten für unspezifizierte Eingabe von Text dem Benutzer die Einflußnahme auf das Programm. Zudem ist die zentrale Aufgabe der untersuchten Anwendung die Anzeige und Manipulation von Dateiinhalten in textueller Form. Da nicht nur hier die meisten Informationen als Text oder grafisches Element vorliegen, handelt es sich hierbei um eine der zentralen Interaktionsformen. Ich nenne sie im folgenden **Eingabe**.

#### 3.2.3.4 Schieber

Insbesondere das Hauptfenster der untersuchten Anwendung macht sehr schnell klar, daß dem Benutzer häufig nur ein Teil der Informationen angezeigt wird. Der restliche Teil wird solange verborgen gehalten, bis der Benutzer die Anzeige fordert. Hierfür verwenden sowohl die Anwendung als auch Motif sogenannte Scrollbars. Ob es sich bei der Verwendung von Scrollbars wirklich um eine Interaktion des Benutzers mit dem Programm handelt, ist vom Programm abhängig. So ist es denkbar, daß die Anwendung den kompletten Dateiinhalt auf einmal einliest und nur den Teil anzeigt, der in das Hauptfenster hineinpaßt. Um den Rest angezeigt zu bekommen, verwendet der Benutzer zwar die Scrollbars, mit dem Programm interagiert er dabei allerdings nicht. Statt dessen verschiebt die Oberfläche lediglich den sichtbaren Bereich. Allerdings ist es auch

denkbar, daß das Programm immer nur so viele Informationen von der Datei anfordert, wie auf einmal im Hauptfenster darstellbar sind. Die Benutzung der Scrollbars führt dann dazu, daß das Programm nicht bekannte Informationen von der Datei anfordert. In diesem Fall ist es sicherlich berechtigt von einer Interaktion zwischen Benutzer und Programm zu sprechen. Da keine der bisher beschriebenen Interaktionsformen auf Scrollbars anwendbar ist, identifiziere ich hier eine weitere Interaktionsform, die ich im folgenden **Schieber** nenne.

#### 3.2.3.5 Betrachtung der verbliebenen Oberflächenobjekte

Bei der Benutzung des untersuchten Anwendungsprogramms fällt auf, daß ein Teil der Interaktionen sich nur unzureichend durch die bisher identifizierten Interaktionsformen beschreiben läßt. Dabei handelt es sich um kurze Anfragen der Anwendung an den Benutzer, die auch eine direkte Antwort erfordern. Beispiel hierfür ist die Frage der Anwendung nach einem Dateinamen. Diese Anfragen bestehen aus mehreren Oberflächenobjekten, die, für sich genommen, durch die bisher identifizierten Interaktionsformen abgebildet werden können. Allerdings bilden die einzelnen Oberflächenobjekte im Sinne der Interaktion eine logische Einheit. Zwar liefert auch die Betrachtung von Motif einen Hinweis für die Berechtigung einer derartigen Abstraktion durch die Existenz verschiedener derartiger Oberflächenobjekte (Popup, File, Request). Allerdings unterscheidet diese Gruppe von Oberflächenobjekten ein wesentlicher Punkt von den Eigenschaften der bisher identifizierten Interaktionsformen. Die bereits beschriebenen Interaktionsformen machen keine Annahmen über fachliche oder technische Voraussetzungen machen, sind also als kontextfrei anzusehen. Die Benutzung von Oberflächenobjekten aus der genannten Gruppe ist dagegen sehr wohl mit bestimmten Voraussetzungen verbunden. So macht die Verwendung des Oberflächenobjektes File nur Sinn, wenn es auf einer dateiorientierten Speicherungsform aufsetzt. Die Einführung einer weiteren Interaktionsform zur Abdeckung derartiger Oberflächenobjekte ist daher nicht möglich.

Ein Blick in die Tabellen 1 und 2 zeigt, daß sowohl die Anwendung als auch Motif noch eine größere Menge an Oberflächenobjekten benutzt bzw. zur Verfügung stellt. Eine genauere Betrachtung zeigt aber, daß diese Oberflächenobjekte sich vorwiegend mit der Organisation von anderen Oberflächenobjekten an der Oberfläche beschäftigen. Sie können daher zwar vom Benutzer manipuliert werden, haben nur Einfluß auf die Darstellung des Programms an der Oberfläche. Daher gibt es für sie auch keine Interaktionsform.

#### 3.2.3.6 Ergebnis der Identifizierung

Die Betrachtung der Oberflächenobjekte und ihrer Funktionsbeschreibungen hat dazu geführt, daß ich einen Satz von vier Interaktionsformen identifizieren konnte. Tabelle 3 faßt noch einmal die identifizierten Interaktionsformen und ihre Beschreibungen zusammen.

| Interaktionsform | Beschreibung |
|---|---|
| Auswahl | Auswahl ermöglicht die Selektion von einem oder mehreren Elementen aus einer mindestens zwei-elementigen Menge. |
| Aktion | Aktion löst in der Anwendung, mit der der Benutzer interagiert, direkt und bewußt eine festgelegte Aktivität |

| | |
|---|---|
| | aus. |
| Eingabe | Eingabe ermöglicht dem Benutzer Informationen in nicht spezifizierter Form mit der Anwendung auszutauschen. |
| Schieber | Schieber ermöglicht dem Benutzer den für ihn sichtbaren Ausschnitt der Anwendung bzw. der Informationen zu verschieben. |

**Tabelle 3. Interaktionsformen**

Diese Interaktionsformen bilden in den folgenden Kapiteln die Grundlage für die vorgeschlagene Einbettung einer Interaktionsformen-Bibliothek in bereits bestehende Klassenbibliotheken.

## 3.3 Zusammenfassung

In diesem Kapitel habe ich den Begriff der **Interaktionsform** eingeführt. Interaktionsformen sollen dazu dienen, die Interaktion zwischen Benutzer und Werkzeug in einen fachlichen und einen technischen Teil zu trennen. Diese Trennung soll durch den Einsatz von Interaktionsformen auch in der Struktur des Werkzeugs sichtbar werden.

Zur Identifizierung der Interaktionsformen habe ich eine zweiteilige Untersuchung durchgeführt. Im ersten Teil habe ich ein Anwendungsprogramm mit grafischer Benutzungsoberfläche auf seine Interaktionsbestandteile hin betrachtet. Auch wenn das ausgewählte Programm keinen Bezug zu WAM aufweist, konnte die Oberfläche hilfreiche Hinweise auf mögliche Interaktionsformen liefern. Im zweiten Teil habe ich das Fenstersystem Motif auf seine Unterstützung für Interaktionen hin untersucht. Ein Vergleich der Ergebnisse beider Untersuchungen hat dann zur Identifikation der Interaktionsformen geführt, wie sie in Tabelle 3 beschrieben sind.

Für die nachfolgenden Kapitel gehe ich davon aus, daß die Interaktionen zwischen Benutzer und Werkzeug über die Verwendung des identifizierten Satzes von Interaktionsformen realisiert werden sollen. Dazu zeige ich eine Möglichkeit, eine Klassenbibliothek für Interaktionsformen zu realisieren und für die Werkzeugkonstruktion zu verwenden.

# 4 Technische Umsetzung der Interaktionsform

In den bisherigen Kapiteln stand vorwiegend die Frage im Vordergrund, wie die Interaktionen zwischen Benutzer und Werkzeug bisher abgebildet werden. Mit den Interaktionsformen habe ich ein neues Konzept eingeführt, das die Veränderung dieser Abbildung ermöglicht. Nachdem ich im letzten Kapitel einen Satz Interaktionsformen identifiziert habe, beschäftige ich mit in diesem Kapitel mit der Einbettung der Interaktionsformen in die Werkzeugkonstruktion. Ausgehend von der bisherigen Verbindung zwischen Interaktionskomponente und Interaktionstypen stelle ich verschiedene Möglichkeiten vor, Interaktionskomponente und Interaktionstypen über die Verwendung von Interaktionsformen miteinander zu verbinden. Im Mittelpunkt der abschließenden Diskussion steht der erhöhte Aufwand, den der Werkzeugentwickler für die Verwendung von Interaktionsformen betreiben muß.

## 4.1 Bisherige Verbindung von Interaktionsform und Interaktionstyp

Die bei WAM bisher verwendete Aufteilung eines Programms in Funktionskomponente (FK), Interaktionskomponente (IAK) und Interaktionstypen (IAT) schreibt eine direkte Verbindung der Interaktionskomponente mit den verwendeten Interaktionstypen vor.

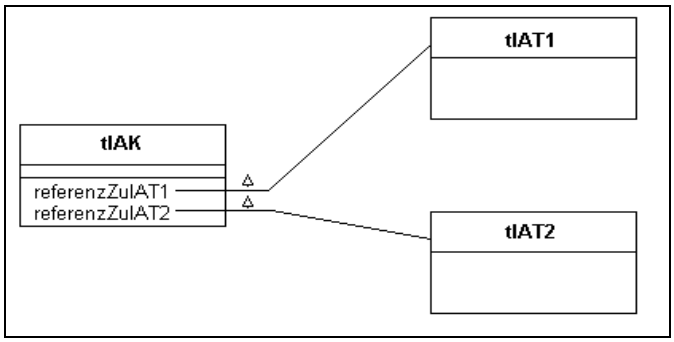


**Abbildung 9. Verbindung zwischen IAK und IAT**

Dadurch wird zwar nicht das Aussehen des Werkzeugs, wohl aber die von ihm benutzten Bestandteile festgelegt. Der Wechsel dieser Bestandteile hat direkten Einfluß auf die IAK, da ein neuer Interaktionstyp angesprochen werden muß. Dies erfordert immer Eingriffe am Code der IAK.

## 4.2 Struktur einer Interaktion mit Interaktionsformen

Betrachten wir noch einmal die Entscheidungen, die die bisherige Verbindung zwischen IAK und IAT beinhaltet (Kap. 2.5), so hat nur die Festlegung auf die Art der Interaktion Einfluß auf die Funktionalität des Werkzeugs. So muß auf eine Auswahl sicherlich anders reagiert werden als auf eine Texteingabe. Die Präsentation an der Oberfläche und die Umgebung, in der das Werkzeug später laufen soll, sind hingegen für seine Funktionalität unerheblich. Die neue Struktur soll dafür sorgen, daß die IAK von diesen, Entscheidungen nicht mehr beeinflußt wird. Anstelle der bisher verwendeten Interaktionstypen (s.o.) verbinden wir daher die IAK mit den Interaktionsformen, die ich in Kapitel 3.2.3 eingeführt habe. Damit wird in der IAK nur noch festgelegt, daß es sich bei der betroffenen Interaktion beispielsweise um eine Auswahl handelt. Wie sich diese Interaktion an der Oberfläche darstellt und welche Oberflächenbibliothek verwendet wird, ist der IAK nicht mehr bekannt. Abbildung 10 zeigt, wie eine Interaktion nach dieser einfachen Entkopplung von IAK und IAT aussieht.

Nachteil dieser einfachen Entkopplung ist, daß für jeden Interaktionstyp eine eigene Interaktionsformen-Klasse geschrieben werden müßte. Da es für Interaktionsformen verschiedene Repräsentationen geben kann, würden mehrere Klassen mit nahezu identischer Funktionalität entstehen. Dadurch wäre die geforderte Entkopplung ad absurdum geführt, da die Änderung der Repräsentation die Verwendung einer anderen IAF-Klasse implizieren würde. Daher ist es zusätzlich nötig, die Interaktionsformen von den Interaktionstypen zu entkoppeln.

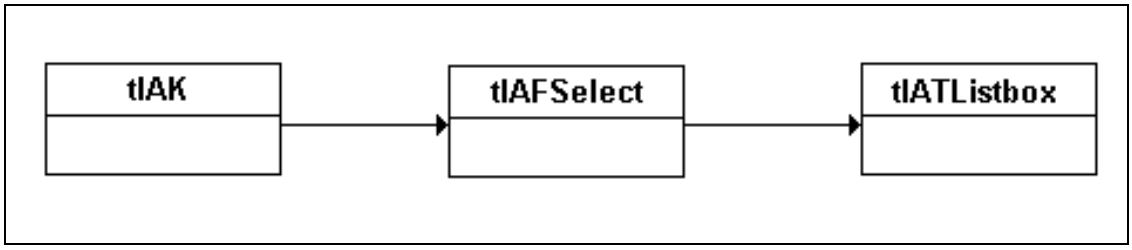


**Abbildung 10. Einstufige Entkopplung**

Die IAK benutzt dann die Abstraktion der Interaktionsform. Für jede Interaktionsform wird eine Minimalschnittstelle, die von Seiten der Oberfläche zu bedienen ist, festgelegt und von der Abstraktion benutzt. Von dieser Schnittstelle wird für jeden möglichen Interaktionstyp eine Klasse abgeleitet. Dies entspricht dem Brückenmuster (siehe [GHJV95]). Mit dem Brückenmuster soll eine Trennung von Schnittstellen und deren Implementation erreicht werden. Die einzelnen Implementationsklassen werden unter Verwendung von Methoden des zu benutzenden Interaktionstyps implementiert. Dies entspricht dem Adaptermuster, das dazu dient, Klassen mit nicht kompatiblen Schnittstellen zusammenarbeiten zu lassen [GHJV95]. Abbildung 11 zeigt schematisch, wie eine solche, doppelt entkoppelte Interaktion aussehen würde.

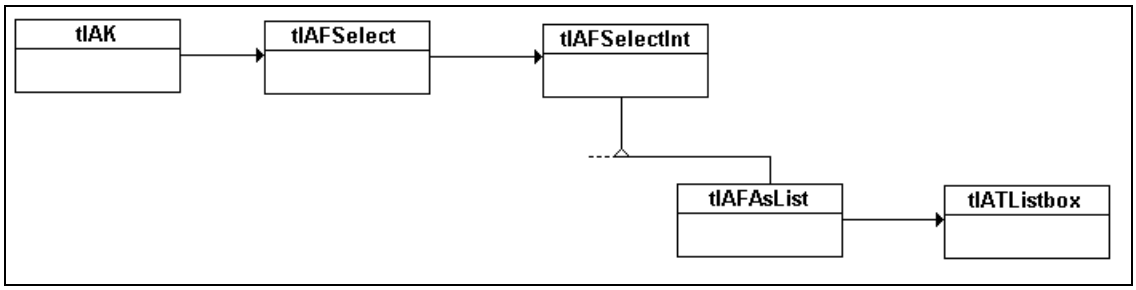


**Abbildung 11. Zweistufige Entkopplung**

Damit ist die IAK nun endgültig von der Repräsentation der Interaktion an der Oberfläche unabhängig. Auch die Interaktionsform ist nicht mehr von der Verwendung bestimmter Interaktionstypen abhängig, sondern kann über das Brückenmuster auswählen, welche Repräsentation tatsächlich Verwendung findet. Damit sind die Entscheidungen über die Interaktion und über ihre Repräsentation voneinander getrennt worden. Allerdings beinhaltet in dieser Version die Entscheidung über die gewählte Repräsentation immer noch die Entscheidung über die gewählte Umgebung. Zwar wäre es möglich, durch eine massive Vermehrung der Ableitungen von der Schnittstelle auch verschiedene Klassen für z.B. Listboxen in Motif und in Tcl/tk zu erzeugen. Aber die Aufnahme neuer Umgebungen würde Anpassungen an den Schnittstellenimplementationen nötig machen, die eigentlich nicht an diese Stelle gehören.

Daher führe ich eine weitere Entkopplungsstufe ein, mit der die Implementation der Schnittstellen von den technisch bedingten Unterschieden zwischen verschiedenen Interaktionstypen-Bibliotheken getrennt werden. Anstelle des bisher verwendeten Adaptermusters betrachte ich die Implementation der Schnittstellen als Abstraktion und verschiebe deren Implementation über eine erneute Verwendung des Brückenmusters in einen weiteren Klassenbaum. Die Basisklasse dieses Klassenbaums definiert die Schnittstelle, die in Ableitungen in Abhängigkeit von den zu nutzenden Interaktionstypen-Bibliotheken implementiert wird. Die Implementation findet dabei unter Verwendung von Methoden der Interaktionstypen statt und entspricht damit dem Adaptermuster. Abbildung 12 zeigt schematisch, wie eine derart dreifach entkoppelte Interaktion aussehen kann.

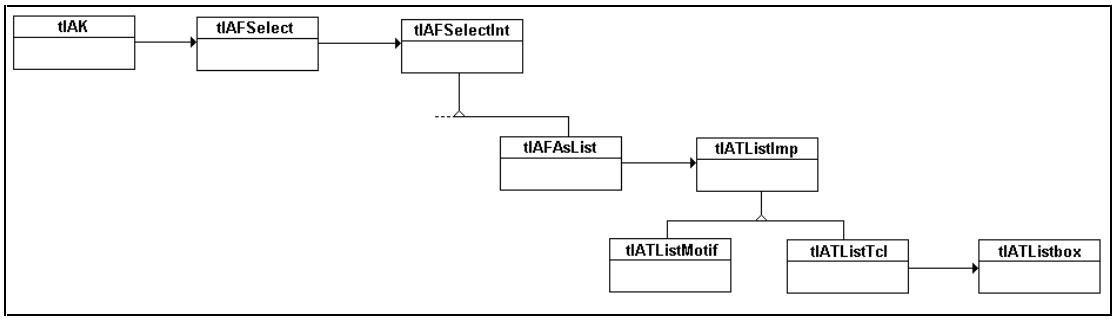


**Abbildung 12. Dreistufige Entkopplung**

Für den Programmablauf und damit für die IAK ist es jetzt völlig unerheblich geworden, mit welchen Oberflächenobjekten die Benutzungsschnittstelle realisiert wird und welches Fenstersystem dafür verwendet wird. Die Entwicklung der IAK und das Design einer geeigneten Oberfläche können stärker entkoppelt werden. Allerdings darf nicht übersehen werden, daß nach wie vor die Notwendigkeit besteht, im Rahmen des Designs der Anwendung die benötigten Interaktionsformen festzulegen und Änderungen sowohl in die Anwendungs- als auch in die Oberflächenentwicklung zu übernehmen.

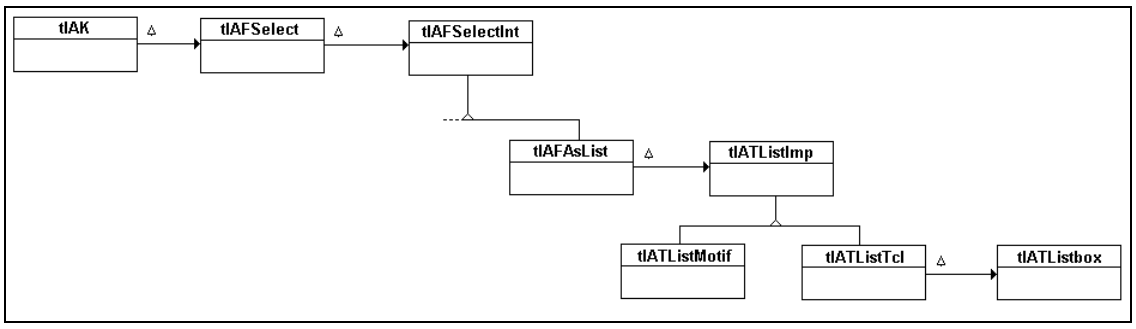


**Abbildung 13. Dreistufige Entkopplung mit Beobachtung**

In der bisherigen Betrachtung habe ich den Aspekt der Beobachtung von Objekten außer Acht gelassen. Das soll an dieser Stelle nachgeholt werden. Die direkte Verbindung von IAK und IAT hat einen durchgängigen Nachrichtenfluß vom Fenstersystem bis zur IAK zugelassen (siehe auch Abbildung 4 und Abbildung 9). Erst der ungehinderte Nachrichtenfluß ermöglicht es dem Werkzeug, eine Interaktion mit dem Benutzer einzugehen. Daher muß der Nachrichtenfluß auch bei Einführung der Interaktionsformen gewährleistet bleiben. Aus diesem Grund werden die Verbindungen zwischen den einzelnen Bestandteilen der Entkopplungsstufen über

Beobachtungsbeziehungen realisiert. Abbildung 13 zeigt die dadurch entstandene Klassenstruktur.

## 4.3 Diskussion des Konzeptes

Durch die Verwendung von Interaktionsformen sollen die Schwächen des bisherigen Konzepts der direkten Verbindung von IAK und IAT ausgeglichen werden. Sowohl die Austauschbarkeit von IATs ohne Auswirkungen auf die IAK als auch die Unabhängigkeit von Fenstersystemen sprechen für die Verwendung von Interaktionsformen.

Allerdings hat die Verwendung von Interaktionsformen nicht nur Vorteile. Der Aufwand, den der Entwickler bei der Programmierung von Werkzeugen treiben muß, wird durch die komplexen Entkopplungsmechanismen der Interaktionsformen erhöht.

Die Interaktionskomponente ist dabei den geringsten Änderungen unterworfen. Hier müssen die Interaktionstypen gegen die entsprechenden Interaktionsformen ausgetauscht werden. Weitere Veränderungen sind nicht nötig. Mit der Festlegung der Interaktionstypen in der Interaktionskomponente ist in der bisherigen Form der Interaktionsabbildung die Verbindung zwischen Werkzeug und Oberfläche abgeschlossen. Resourcen-Builder unterstützen den Entwickler in der Auswahl der Interaktionstypen, die die Benutzungsoberfläche des Werkzeugs bilden sollen.

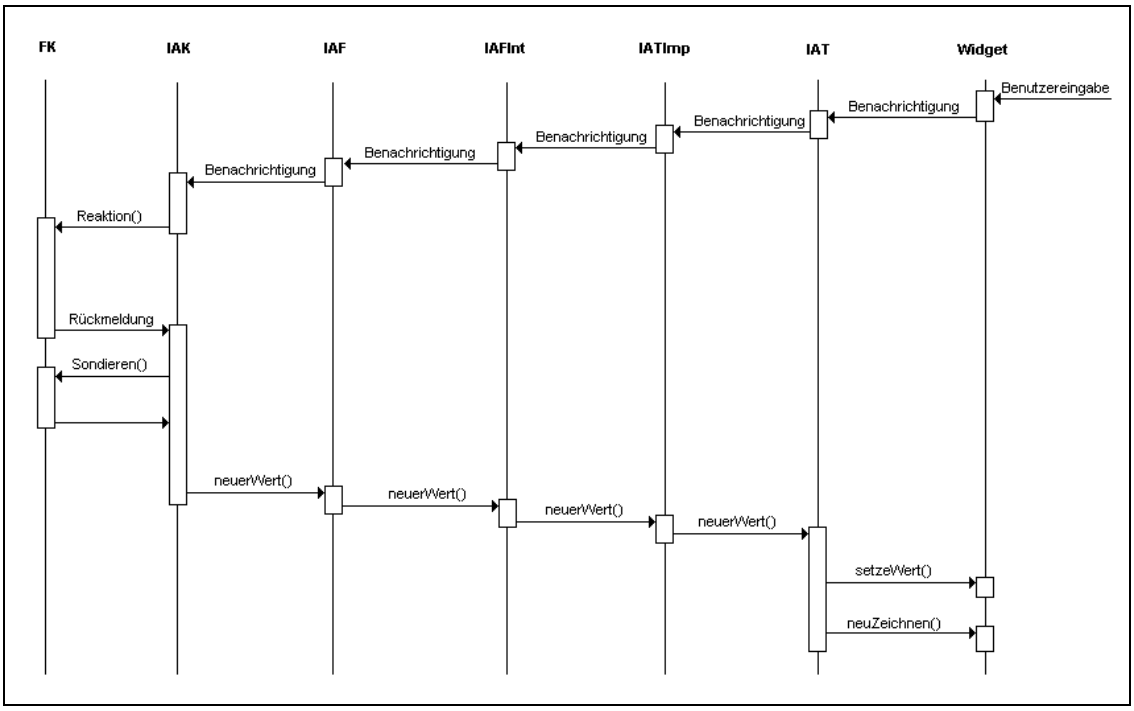


**Abbildung 14. Interaktionsdiagramm einer Benutzereingabe mit Interaktionsformen**

Bei der vorgeschlagenen Entkopplungsstruktur zwischen Interaktionskomponente und Oberfläche ist die Festlegung der verwendeten Interaktionsformen nur der erste Schritt. Für die vollständige Verbindung fehlen noch die Festlegungen über das verwendete Fenstersystem und den Interaktionstyp, der die Interaktionsform an der Oberfläche präsentieren soll. Entwicklungswerkzeuge, die den Entwickler bei diesen Entscheidungen unterstützen, existieren derzeit nicht. Daher sind die Festlegungen ganz

in die Hände des Entwicklers gelegt. Die vielfältigen Kombinations- und Fehlermöglichkeiten erschweren diese Arbeit.

Die Objektstruktur des Werkzeugs erfährt ebenfalls deutliche Veränderungen. Die Interaktionskomponente und die verwendeten Interaktionstypen finden sich sowohl in der bisherigen, als auch in der vorgeschlagenen Interaktionsabbildung. Die vorgeschlagene Entkopplung führt für jeden verwendeten Interaktionstyp fünf weitere Objekte ein (siehe Abbildung 12). Bei Werkzeugen mit komplexeren Oberflächen kommt es so schnell zu einer Vervielfachung der Objektanzahl. Dadurch sind Performance-Einbußen gegenüber der bisherigen Abbildung zu erwarten.

Abbildung 14 zeigt anhand des Beispiels aus Kapitel 2.4 noch einmal sehr deutlich, daß die Verwendung von Interaktionsformen die Komplexität des Werkzeugstruktur erhöht. Trotz der einfachen Grundstruktur der dargestellten Interaktion sind statt vorher vier nun 7 Objekte an ihrer Behandlung beteiligt. Die Aufgabe der IAF-Objekte (IAF, IAFInt und IATImp) besteht dabei in der Kapselung der jeweils folgenden Entkopplungsstufe. Dazu werden die eingehenden Nachrichten analysiert und durch eigene Nachrichten ersetzt. Ebenso werden die Methodenaufrufe für die nachfolgend aufgerufenen Objekte angepaßt.

## 4.4 Das Kommando-Muster als alternativer Kopplungsmechanismus

In den vorangegangenen Kapiteln habe ich eine mögliche Kopplung von Interaktionskomponenten mit den zugehörigen Interaktionstypen vorgestellt. Die Struktur der Kopplung macht keine Annahmen darüber, wie die einzelnen Verbindungen technisch realisiert werden. Das Interaktionsdiagramm aus Abbildung 14 hingegen sieht einen Reaktionsmechanismus auf Basis von Ereignissen als Verbindung vor. Dieser Mechanismus wird von mir auch für die Realisierung der Interaktionsformen verwendet.

Die Verwendung von Ereignissen für die Kopplung von Objekten ist allerdings nicht die einzige Möglichkeit. In [GHJV95] wird alternativ eine Verwendung des Kommando-Musters vorgeschlagen. Ein Kommando-Objekt enthält dabei die Information darüber, welches Objekt zu benachrichtigen ist und welche Methode für die Benachrichtigung aufgerufen werden soll. Dieses Kommando-Objekt wird dem benachrichtigenden Objekt bekannt gemacht. Damit wird eine Verbindung zwischen dem benachrichtigenden und dem benachrichtigten Objekt geschaffen, die in keines der beteiligten Objekte einprogrammiert werden muß.

Für die Verwendung von Interaktionsformen können Kommando-Objekte an zwei Stellen eingesetzt werden:

1. Bei der Kopplung von Interaktionstypen mit Interaktionsformen und
2. bei der Kopplung von Interaktionsformen mit der Interaktionskomponente.

Nicht eingesetzt werden können Kommando-Objekte bei der Kopplung von Interaktionstypen und den Oberflächenobjekten des verwendeten Fenstersystems. Da die Oberflächenobjekte meistens in Form von Bibliotheken vorliegen, ist man auf die Verwendung des vorgegebenen Reaktionsmechanismus angewiesen.

Abbildung 15 zeigt das Interaktionsdiagramm einer Benutzereingabe für die Verbindung zwischen Interaktionstyp und Interaktionsform. Bei der Initialisierung des Interaktionstyps wird diesem ein Kommando-Objekt zugewiesen. Das Kommando-Objekt enthält in der Methode execute() die Information, welche Methode bei der beobachtenden Interaktionsform aufgerufen werden soll. Bei einer Benutzereingabe ruft der Interaktionstyp beim Kommando-Objekt die Methode execute() auf, die die vorgegebene Methode bei dem vorgegebenen Interaktionsform-Objekt aufruft.

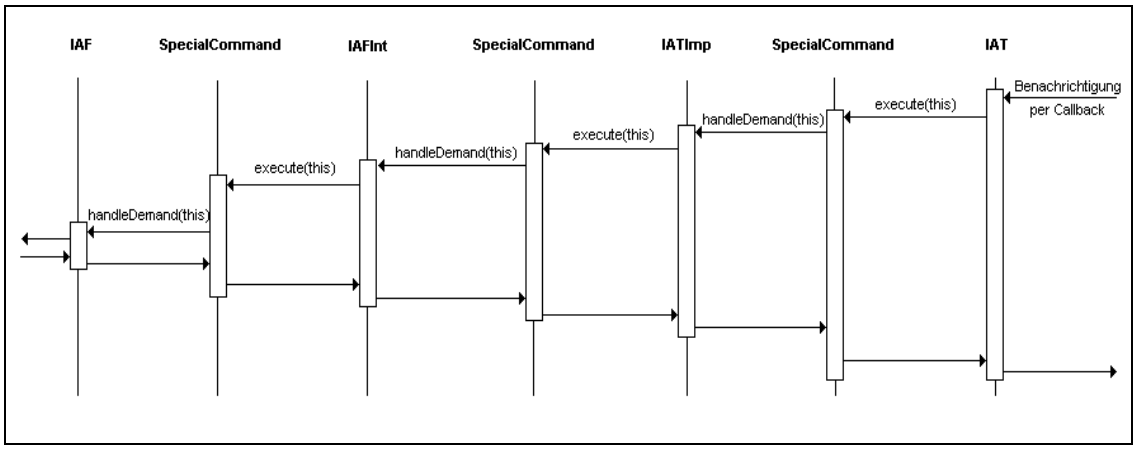


**Abbildung 15. Interaktionsdiagramm einer Benutzereingabe mit Kommando-Mechanismus**

In diesem Beispiel wird von den Kommando-Objekten immer die gleiche Methode aufgerufen. Alternativ wäre es daher sicher denkbar, bereits das erste Kommando-Objekt mit einem Aufruf auf das Interaktionsformen-Objekt zu versehen. Im Sinne der Trennung der einzelnen Entkopplungsschichten ist eine solche Vorgehensweise aber abzulehnen, da so doch wieder eine stärkere Verbindung etabliert wird.

Die Verwendung des Kommando-Musters zur losen Kopplung von Objekten ermöglicht eine durchgängige Verwendung von Objekten, während die Nutzung eines ereignisbasierten Reaktionsmechanismus eine Übertragung des Callback-Mechanismus prozeduraler Sprachen darstellt. Durch die Festlegung innerhalb des Kommando-Objektes, kann die Reaktion des beobachtenden Objekts einfach verändert werden. Dazu muß lediglich ein anderes Kommando-Objekt mit einer anderen Festlegung dem beoachteten Objekt übergeben werden.

Trotz dieser Vorteile verwende ich für die Realisierung der Interaktionsformen einen ereignisbasierten Reaktionsmechanismus. Einerseits würde die Verwendung von Kommando-Objekten die Zahl der benötigten Objekte noch weiter erhöhen, als dies sowieso schon der Fall ist. Da die meisten Fenstersysteme ebenfalls einen ereignisbasierten Reaktionsmechanismus verwenden, wird andererseits dieser Mechanismus durchgängig verwendet.

## 4.5 Zusammenfassung

In diesem Kapitel habe ich verschiedene Möglichkeiten vorgestellt, um mit dem Konzept der Interaktionsformen den technischen und den fachlichen Teil von Interaktionen voneinander zu entkoppeln. Die Interaktionsformen werden dabei zwischen Interaktionskomponente und Interaktionstyp eingefügt. Die Entkopplung

zwischen den beiden Teilen kann mit einer unterschiedlichen Zahl an Entkopplungsstufen durchgeführt werden.

Die abschließende Diskussion des eingeführten Konzeptes zeigt, daß die Verantwortung des Entwicklers durch die Entkopplung weiter ansteigt. Zudem nimmt die Zahl der beteiligten Objekte deutlich zu. Performanceeinbußen und erschwerte Fehlersuche sind die Folge.

Für die prototypische Realisierung einer Klassenbibliothek verwende ich die beschriebene dreistufige Entkopplung. Der Reaktionsmechanismus wird unter Verwendung von Ereignissen implementiert. Eine Beschreibung der Interaktionsformen-Klassen findet sich im folgenden Kapitel.

# 5 Realisierung einer Interaktionsformen-Bibliothek

Im vorangegangenen Kapitel habe ich die Möglichkeiten diskutiert, Interaktionsformen in die Werkzeugkonstruktion einzubetten. Die Klassen, die die Interaktionsformen repräsentieren, treten dabei nur einzeln auf und erscheinen als Blackbox. Den bisher fehlenden Zusammenhang der Interaktionsformen-Klassen in Form einer Klassenbibliothek stelle ich in diesem Kapitel her. Die Realisierung der Klassenbibliothek wird von mir nur in Teilen durchgeführt. Die realisierten Klassen beschreibe ich abschließend im Sinne einer Benutzerreferenz.

## 5.1 Die Klassenhierarchie

Da einige grundsätzlichen Entscheidungen, die die Struktur der Klassenbibliothek mitbestimmen, bereits in den vorangegangenen Kapiteln untersucht worden sind, soll an dieser Stelle nicht noch einmal auf den zugrunde liegenden Entscheidungsprozeß eingegangen werden. Für die grafische Darstellung der Klassenhierarchie wird die Baumstruktur gewählt. Auf einer Hierarchiestufe des entstehenden Baumes stehen die, in Kapitel 3.2.3 identifizierten, Interaktionsformen. Wie bereits in Kapitel 4 zu sehen ist, werden die Klassennamen zur besseren Unterscheidung mit dem Präfix tIAF versehen. Außerdem verwende ich für die Klassennamen Begriffe in englischer Sprache, da damit die Wiederverwendbarkeit der Klassen ausgedrückt werden soll. Zudem verwenden auch die bereits bestehenden Bibliotheken, in deren Umgebung sich die Interaktionsformen-Bibliothek einfügen soll, englische Klassennamen. Damit erhalte ich folgende Klassennamen: tIAFSelection, tIAFControl, tIAFEntry und tIAFScroll. Allerdings bilden diese Klassen zum Teil selbst Abstraktionen von der bereits abstrahierten Interaktion. So wird die Klasse tIAFSelection in Einfach- und Mehrfachauswahl (tIAFSingleSelect bzw. tIAFMultiSelect) spezialisiert. Von der Klasse tIAFEntry werden Klassen für ein- und mehrzeilige Eingaben abgeleitet (tIAFSingleLineEntry bzw. tIAFMultiLineEntry). Die so entstandenen Klassen bilden mit den nicht weiter spezialisierten Klassen die Blätter des Klassenbaums. Exemplare dieser Klassen werden direkt innerhalb der Interaktionskomponenten erzeugt und benutzt. Funktionaltät, die allen Klassen gemeinsam ist, wird in einer abstrakten

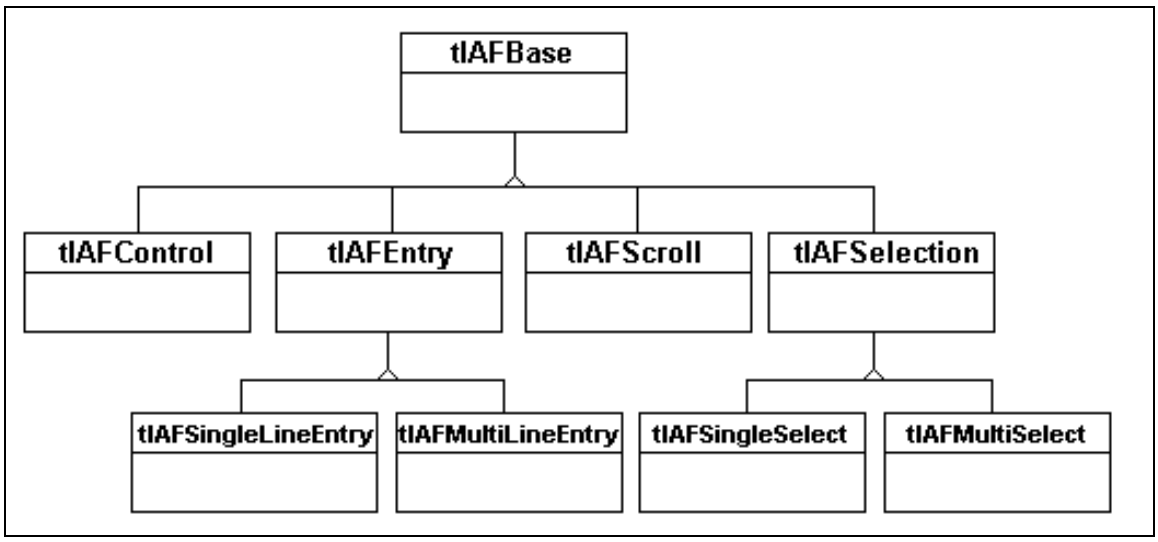


**Abbildung 16. Der Interaktionsformen-Klassenbaum**

Oberklasse für alle bereitgestellt. Da diese Klasse innerhalb des entstehenden Klassenbaums als Wurzelklasse fungiert, wird sie, analog zum Vorgehen in den bereits

bestehenden Bibliotheken, tIAFBase genannt. Abbildung 16 zeigt den so entstandenen Klassenbaum.

## 5.2 Umgangsformen der Interaktionsformen-Klassen

In diesem Kapitel werden die Schnittstellen der Interaktionsformen-Klassen tIAFBase, tIAFSelection, tIAFSingleSelect und tIAFMultiSelect beschrieben, die im Rahmen dieser Arbeit implementiert werden. Dieses Kapitel ist damit auch als Benutzerreferenz für diese Klassen zu verstehen.

### 5.2.1 tIAFBase

Die Klasse tIAFBase stellt die Grundmenge der Methoden dar, die von allen abgeleiteten Interaktionsformen-Klassen zur Verfügung gestellt werden müssen. Dies beinhaltet die Nutzung des Beobachter-Mechanismus, Methoden zu Erzeugung und Löschung von Verbindungen zu Interaktionstypen und die Initialisierung.

#### 5.2.1.1 isValidClause( tClause& )

Die Methode isValidClause() erhält einen Parameter vom Typ tClause. Dieser Parameter wird daraufhin überprüft, ob es sich die Beschreibung einer Interaktion handelt.

#### 5.2.1.2 Notify( tNotifier*, tEvent )

Die Methode Notify() beschreibt, wie die Klasse tIAFBase auf das Auftreten des Ereignisses tEvent reagiert. Damit finden die Interaktionsformen-Klassen Anschluß an den Beobachter-Mechanismus der anderen Bibliotheken. Diese Methode ist hier nur standardmäßig implementiert und muß von den Ableitungen von tIAFBase überschrieben werden.

#### 5.2.1.3 Init( tNotifier*, tIAFInterface*)

In der Methode Init() werden die Initialsierungen für ein neu erzeugtes Objekt vorgenommen. In der Klasse tIAFBase wird die Initialisierung für den Beobachter vorgenommen und die Referenz auf das Interface-Objekt zugewiesen, mit dem die Verbindung zu einem Interaktionstypen erfolgen soll.

#### 5.2.1.4 ConstructIAT( tIAFClause)

Die Methode ConstructIAT() erzeugt eine Verbindung von der Interaktionsform zu einem Interaktionstypen. Die Einzelheiten der Verbindung sind im Parameter tIAFClause festgelegt. Diese Methode ist hier leer implementiert und muß von allen Ableitungen von tIAFBase überschrieben werden.

#### 5.2.1.5 DeleteIAT( tIAFInterface* )

Die Methode DeleteIAT() zerstört die Verbindung zu einem Interaktionstypen. Die Verbindung wird über die Referenz auf eine Interface-Klasse identifiziert. Diese Methode ist hier leer implementiert und muß von allen Ableitungen von tIAFBase überschrieben werden.

### 5.2.2 tIAFSelection

Die Klasse tIAFSelection ist von der Klasse tIAFBase abgeleitet. Sie ist ebenfalls eine abstrakte Klasse, die die Grundfunktionalität der Interaktionsformen um spezielle Umgangsformen für beliebige Auswahlen erweitert. Dies beinhaltet Möglichkeiten, die Liste der auswählbaren Einträge zu manipulieren und festzustellen, ob ein bestimmter Eintrag ausgewählt ist. Als Einträge sind nur Zeichenketten vom Typ tString zugelassen.

Alle Einträge werden in einem Container vom Typ tList gehalten. Für eine genauere Beschreibung der Funktionalität von tList wird auf der Bibliothek ConLib verwiesen, die diesen Typ zur Verfügung stellt [Trau95].

#### 5.2.2.1 Init( tList <tString*>* )

Die Methode Init() initialisiert ein neu zu erzeugendes Objekt mit einer Liste aus Einträgen vom Typ tString. Diese Liste stellt die Auswahlmöglichkeiten dar, die zur Zeit der Initialisierung zur Verfügung stehen.

#### 5.2.2.2 GetSelectionList()

Die Methode GetSelectionList() liefert eine Referenz auf die Liste aller auswählbaren Einträge, unabhängig davon, ob sie ausgewählt sind.

#### 5.2.2.3 PutSelectionList( tList <tString*>* )

Die Methode PutSelectionList() ersetzt die bisher benutzte Liste der auswählbaren Einträge durch eine neue Liste. Die alte Liste wird dabei komplett ausgewechselt. Eine eventuell bestehende Auswahl auf Einträge aus der alten Liste ist anschließend nicht mehr gültig.

#### 5.2.2.4 AppendSelectionItem( tString* )

Die Methode AppendSelectionItem() erweitert die Liste der auswählbaren Einträge um einen weiteren Eintrag. Der neue Eintrag wird am Ende der Liste angefügt.

#### 5.2.2.5 InsertSelectionItem( tString*, int )

Die Methode InsertSelectionItem() erweitert die Liste der auswählbaren Einträge um einen weiteren Eintrag. Der neue Eintrag wird in der Liste an der angegebenen Position eingefügt. Sollte die angegebene Position über das Ende der Liste hinaus reichen, wird der Eintrag am Ende der Liste eingefügt.

#### 5.2.2.6 DeleteSelectionItem( int ) / DeleteSelectionItem( tString* )

Die Methode DeleteSelectionItem() entfernt eine Eintrag aus der Liste der auswählbaren Einträge. Der Eintrag wird über seine Position innerhalb der Liste oder die Referenz auf den Identifikationsstring identifiziert.

#### 5.2.2.7 ChangeSelectionItem( tString*, int ) / ChangeSelectionItem( tString* , tString* )

Die Methode ChangeSelectionItem() ersetzt einen Eintrag in der Liste der auswählbaren Einträge durch einen anderen Eintrag. Der erste Parameter gibt dabei den Eintrag an. Der zweite Parameter identifiziert den zu ersetzenden Eintrag. Die Identifikation kann dabei entweder über die Position des Eintrags in der Liste oder die Referenz auf den Identifikationsstring erfolgen.

#### 5.2.2.8 IsSelected( tString* )

Die Methode IsSelected() stellt fest, ob der übergebene Eintrag derzeit ausgewählt ist. Die Antwort wird als Boolean-Wert zurückgeliefert. Diese Methode ist hier nicht implementiert. Sie muß von jeder Ableitung überschrieben werden.

#### 5.2.2.9 UpdateSelectionList()

Die Methode UpdateSelectionList() wird verwendet, um Veränderungen an der Liste der auswählbaren Einträge bis zum betreffenden Oberflächenobjekt bekannt zu machen.

Diese Methode ist hier nicht implementiert. Sie muß von jeder Ableitung überschrieben werden.

### 5.2.3 tIAFSingleSelect

Die Klasse tIAFSingleSelect ist von der Klasse tIAFSelection abgeleitet. Sie erweitert die Funktionalität der Klasse tIAFSelection um Methoden für die Verwendung einer Einfach-Auswahl zur Verfügung. Die zusätzlichen Methoden ermöglichen die Handhabung des jeweils ausgewählten Eintrags.

#### 5.2.3.1 IsSelected( tString* )

Die Methode IsSelected() stellt fest, ob der angegebene Eintrag derzeit ausgewählt ist. Die Antwort wird als Boolean-Wert zurückgeliefert.

#### 5.2.3.2 GetSelectedItem()

Die Methode GetSelectedItem() liefert den derzeit ausgewählten Eintrag zurück.

#### 5.2.3.3 SetSelectedItem( tString* )

Die Methode SetSelectedItem() verändert die derzeitige Auswahl derart, daß der angegebene Eintrag als ausgewählt gilt. Die vorherige Auswahl ist nicht mehr gültig. Der betroffene Eintrag wird durch die Referenz auf seine Stringidentifikation identifiziert.

#### 5.2.3.4 UpdateSelectionList()

Die Methode UpdateSelectionList() gibt die aktuelle Liste der auswählbaren Einträge an das Interface weiter, das mit der Interaktionsform verbunden ist.

### 5.2.4 tIAFMultiSelect

Die Klasse tIAFMultiSelect ist von der Klasse tIAFSelection abgeleitet. Sie erweitert die Funktionalität der Klasse tIAFSelection um Methoden für die Verwendung einer Mehrfach-Auswahl zur Verfügung. Die zusätzlichen Methoden ermöglichen die Handhabung der jeweils ausgewählten Einträge.

#### 5.2.4.1 IsSelected( tString*)

Die Methode IsSelected() stellt fest, ob der angegebene Eintrag derzeit ausgewählt ist. Die Antwort wird als Boolean-Wert zurückgeliefert.

#### 5.2.4.2 GetSelectedList()

Die Methode GetSelectedList() liefert die Liste der ausgewählten Einträge. Diese Liste ist immer eine Teilmenge der Liste aller auswählbaren Einträge.

#### 5.2.4.3 SetSelectedList( tList <tString*>* )

Die Methode SetSelectedList() verändert die derzeitige Auswahl derart, daß die Einträge in der übergebenen Liste als ausgewählt gelten. Die vorherige Auswahl ist nicht mehr gültig. Die betroffenen Einträge werden durch die Referenzen auf ihre Stringidentifikation identifiziert.

#### 5.2.4.4 UpdateSelectionList()

Die Methode UpdateSelectionList() gibt die aktuelle Liste der auswählbaren Einträge an das Interface weiter, das mit der Interaktionsform verbunden ist.

# 6 Verwandte Arbeiten

Die Vorteile der Trennung der Anwendung von der Benutzungsoberfläche schon im Bereich des Anwendungsdesign hat bereits andere Arbeiten inspiriert, die zum Teil andere Ansätze oder andere Zielrichtungen verfolgen. Eine dieser Arbeiten möchte im folgenden noch kurz darstellen.

## 6.1 Abstraktion zum Ziele der Interaktionsmodalität [Gell96]

Diese Arbeit befaßt sich mit der Trennung von Softwareentwicklung und Interaktionsentwicklung. Beide Teile werden dabei zwar als getrennte Einheiten gesehen, die aber im Zuge der Entwicklung interaktiver Software integriert werden müssen. Unter Interaktionsmodalität wird dabei die Durchführung von Interaktion auf, vom Benutzer unterscheidbare, Art und Weise verstanden. Mögliche Unterscheidungen liegen dabei in den verwendeten Kommunikationskanälen (z.B. visuell, auditiv), den Darstellungsmedien (Text, Grafik) oder den Interaktionsmetaphern (z.B. Direktmanipulation, Konversation).

Neben verschiedenen Methodiken zu Design und Entwurf interaktiver Software wird im Rahmen dieser Arbeit auch eine Hierarchie von abstrahierenden Interaktionsklassen entwickelt, die die Integration von Interaktions- und Softwareentwicklung unterstützen sollen. Die Unterscheidungen der Interaktionsmöglichkeiten basieren auf der Art des Nachrichtenaustausches, den die instanziierten Objekte unterstützen.

Die Interaktionsformen versuchen hauptsächlich die Oberflächenobjekte vor der Interaktionskomponente zu verbergen und somit die Präsentation und die Implementation zu entkoppeln. Als Interaktionsbereich steht dabei stets eine grafische Benutzeroberfläche zur Verfügung, auf der alle Ein- und Ausgaben stattfinden. Alle identifizierten Abstraktionen gehen immer von dieser Grundvoraussetzung aus. Durch Interaktionsmodalität hingegen wird versucht, eine Abstraktion vom verwendeten Interaktionsbereich zu finden. Die daraus resultierende Hierarchie von Interaktionsklassen orientiert sich daher an den Fähigkeiten, die der Interaktionsbereich zum Austausch von Nachrichten zur Verfügung stellt. Ziel ist es, den Wechsel des Interaktionsbereichs soweit wie möglich zu kapseln.

Der Einsatzbereich von Interaktionsformen würde sich nach [Gell96] auf unimodale Systeme beschränken, d.h. auf Systeme, die nur einen Interaktionsbereich kennen. Die Abstraktion der Interaktionsformen geht auf solchen Systemen allerdings über die Abstraktion des Nachrichtenaustausches aus [Gell96] hinaus.

# 7 Offene Punkte und Probleme

Nachdem in den bisherigen Kapiteln die Vorzüge der Verwendung von Interaktionsformen ausführlich gewürdigt wurden, soll an dieser Stelle nicht verschwiegen werden, daß es bei ihrer Verwendung auch bisher nicht gelöste Probleme gibt.

In der bisherigen Form hat das Design einer Benutzungsoberfläche immer direkte Auswirkungen auf die IAK gehabt. Interaktionsformen lösen diesen direkten Bezug auf. Ihre Verwendung macht es jedoch notwendig, den Prozeß des Oberflächendesign den neuen Erfordernissen anzupassen und aufzuteilen. Im ersten Teil erfolgt die Analyse der gewünschten Interaktion und die Festlegung, welche Interaktionsformen an welcher Stelle in der IAK anzusprechen ist. Der zweite Teil besteht dann daraus, aus den Vorgaben des ersten Teils eine ergonomisch und funktionell angemessene Oberfläche zu bauen. An genau dieser Stelle entsteht das Problem. Es existiert kein Hilfsmittel, das den Designer der Oberfläche darin unterstützt, die Oberfläche gemäß den Vorgaben aufzubauen. Er muß also genau wissen, welche Oberflächenobjekte ihm für die Repräsentation einer bestimmten Interaktionsform zur Verfügung stehen.

Für das Design einer Benutzungsoberfläche stehen, je nach gewähltem Oberflächenmanager, verschiedene Designwerkzeuge, sogenannte Resource-Builder, zur Verfügung. Diese Werkzeuge erzeugen Beschreibungen der zu erzeugenden Oberfläche, die vom Compiler oder Interpreter dazu benutzt werden, die gewünschte Oberfläche zu erzeugen. Leider sind die so erzeugten Beschreibungen in der Regel zueinander inkompatibel und auf einen speziellen Compiler/Interpreter angepaßt. Eine Gemeinsamkeit besitzen jedoch alle Resource-Builder. Sie sind für die Verwendung von Interaktionsformen unbrauchbar.

Soll die Anwendung Interaktionsformen verwenden, wird die Verbindung zwischen IAK und Benutzungsoberfläche dynamisch mittels „später Erzeugung“ hergestellt. Dafür benötigen sie eine Beschreibung, wie die Verbindung aussehen soll, d.h. welcher, in der IAK benannten, Interaktionsform welches Oberflächenobjekt zugeordnet werden soll. Um sinnvoll und benutzerfreundlich mit Interaktionsformen arbeiten zu können, werden also Werkzeuge benötigt, die sowohl den Oberflächendesigner unterstützen, als auch eine Beschreibung der Oberfläche und ihrer Verbindung zur IAK erzeugen. Solche Werkzeuge existieren zum gegenwärtigen Zeitpunkt noch nicht. Die Beschreibung der Verbindung der IAK mit der Oberfläche erfolgt derzeit über eine Textdatei. Dieses Verfahren ist nur mühsam sondern auch ausgesprochen fehleranfällig. Viele Vorteile der Interaktionsformen, insbesondere die erleichterte Verwendung von Prototypen, sind daher derzeit nicht oder nur eingeschränkt nutzbar.

# 8 Literaturverzeichnis

[Gell96] Hans-Werner Gellersen, *Methodische Entwicklung flexibler interaktiver Software*, Shaker Verlag, Aachen, 1996.

[Gryc96] Guido Gryczan. *Prozessmuster zur Unterstützung kooperativer Tätigkeit*, Deutscher Universitäts-Verlag, Wiesbaden, 1996.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, 1995.

[KGZ94] Klaus Kilberth, Guido Gryczan, Heinz Züllighoven. *Objektorientierte Anwendungsentwicklung - Konzepte, Strategien, Erfahrungen*, Friedr. Vieweg, Braunschweig, 1993.

[Meye90] Bertrand Meyer. *Objektorientierte Softwareentwicklung*, Hanser, München, 1990.

[Oust94] John K. Ousterhout. *Tcl and the Tk toolkit*, Addison-Wesley, Reading, MA, 1994.

[RBP+91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorenson. *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[RiZü95] Dirk Riehle, Heinz Züllighoven. *A Pattern Language for Tool Construction and Intergration Based on the Tools and Materials Metaphor*, in J. Coplien, D. Schmidt: Pattern Languages of Program Design, Addison-Wesley, Reading, MA, 1995.

[RoWo96] Stefan Roock, Henning Wolf. *Konzeption und Implementierung eines Reaktionsmusters für objektorientierte Softwaresysteme*, Studienarbeit am FB Informatik, Hamburg, 1996.

[Trau95] Horst-Peter Traub, *Objektorientierte Behälterklassen - Konzepte, Entwurf und Implementation*, Diplomarbeit am FB Informatik, Hamburg, 1995.

Der Sourcecode der besprochenen Interaktionsformen-Klassen liegt als Archiv unter http://swt-www.informatik.uni-hamburg.de/~1sturm/public/iafsource.tar.gz.