Verwendung von GUI - Komponenten in Rahmenwerken nach dem Werkzeug und Material - Ansatz

Diplomarbeit
am
Fachbereich Informatik,
Arbeitsbereich Softwaretechnik,
der Universität Hamburg

Betreuer:
Prof. Dr. Heinz Züllighoven
Prof. Dr. Horst Oberquelle

Eike Steffen
Wilhelm–Cohrs–Weg 2a
21217 Seevetal

Sven Lammers
Geschwister–Scholl–Str. 7a
23795 Bad Segeberg

November 1999

Betreuung:

Prof. Dr. Heinz Züllighoven
Arbeitsbereich Softwaretechnik (SWT)
Fachbereich Informatik
Universität Hamburg
Vogt–Kölln–Straße 30

22527 Hamburg

Prof. Dr. Horst Oberquelle
Arbeitsbereich Angewandte und Sozialorientierte Informatik (ASI)
Fachbereich Informatik
Universität Hamburg
Vogt–Kölln–Straße 30

22527 Hamburg

Inhaltsverzeichnis

1 Einleitung

Fast jedes zur Zeit verfügbare Softwareprodukt versucht mittlerweile einem Benutzer den Zugang zu der Funktionalität dieses Produktes über eine grafische Benutzerschnittstelle[1] zu vereinfachen. Ursache hierfür ist die stetige Steigerung der zu verarbeitenden Menge an Informationen, sowie die zunehmende Komplexität der Aufgaben, die mit Hilfe von Software zu erledigen sind. Für die meisten Projekte, die mit der Realisierung von Software befaßt sind, gilt, daß die Implementierung und das Design der Oberfläche annähernd die Hälfte der zur Verfügung stehenden Zeit in Anspruch nimmt. Zu der Frage, warum die technische Realisierung von derlei Benutzerschnittstellen keine leichte Aufgabe ist, finden wir in einem Papier von Myers mit dem Titel UIMSs, Toolkits, Interface Builders[2] (siehe [Myers96] Kapitel 1):

> *These interfaces require that the programmer deal with elaborate graphics, multiple ways for giving the same command, multiple asynchronous input devices (usually a keyboard and a pointing device such as a mouse), a „mode free" interface where the user can give any command at virtually any time, and rapid „semantic feedback" where determining the appropriate response to user actions requires specialized information about the objects in the program.*

Bezogen auf die Gesamtheit einer grafischen Benutzerschnittstelle müssen also im wesentlichen drei Anforderungen realisiert werden. Zunächst ist hier die eigentliche Darstellung auf einem Grafikkontext zu nennen, gefolgt von dem Umgang mit den verschiedensten Eingabegeräten und schließlich die Benachrichtigung der die Funktionalität bereitstellenden Software über Ereignisse innerhalb der grafischen Oberfläche. Wie schwierig die Zusammenstellung einer Oberfläche ist, hängt naturgemäß von der Komplexität der zu erfüllenden Aufgabe ab. In diesem Zusammenhang erreicht insbesondere der Aspekt der Ergonomie einer Oberfläche eine immer größer werdende Bedeutung. Die Tatsache, daß derlei Oberflächen nicht mehr aus den heutigen Entwicklungen wegzudenken sind und die Anzahl der Gestaltungsmöglichkeiten einer Oberfläche stetig wächst hat uns dazu motiviert, diese Arbeit zu erstellen. Wir haben uns für diese Arbeit allerdings aus diesem umfangreichen Thema denjenigen Bestandteil herausgesucht, welcher der Frage nachgeht, wie die Elemente der Oberfläche an ein Werkzeug angebunden werden. Wir werden diese Frage aus der Blickrichtung des am Arbeitsbereich Softwaretechnik vertretenen Werkzeug, Automat und Material - Ansatz betrachten.

Vom technischen Standpunkt aus betrachtet, bestehen diese grafischen Benutzerschnittstellen aus einzelnen Elementen, den sogenannten Widgets, die in einem eigenen Darstellungsbereich, einem Fenster, angeordnet sind. Die Aufgabe eines Anwendungsentwicklers bei der Implementierung einer Oberfläche besteht nun darin, diese Widgets so in einen Zusammenhang zu bringen, daß dem Benutzer zum Beispiel die Bearbeitung einer fachlichen Aufgabe ermöglicht wird. Der angesprochene Zusammenhang bedeutet

---

[1] Häufig werden wir in der Folge auch das Kürzel GUI (Graphical User Interface) als Synonym in diesem Zusammenhang verwenden.

[2] UIMS ist die Kurzform für User Interface Management System.

auf der einen Seite, daß die Gestaltung der Oberfläche unter Berücksichtigung von ergonomischen Gesichtspunkten erfolgen muß und auf der anderen Seite, daß die einzelnen Widgets an die darunter liegende Software angebunden werden müssen, welche die eigentliche Funktionalität bereitstellt. Es ist leicht einzusehen, daß ein Widget immer in starker Abhängigkeit zu dem verwendeten Betriebssystem steht. So ist es zum Beispiel zu erklären, daß die unterschiedlichen Betriebssysteme auch jeweils ein individuelles „Look and Feel" aufweisen. In der Konsequenz kann dies für ein bereits bestehendes Softwareprodukt bedeuten, daß die Portierung auf ein anderes Betriebssystem die erneute Erstellung der gesamten grafischen Benutzerschnittstelle nach sich zieht. Ein weiterer Aspekt, der in diesem Zusammenhang berücksichtigt werden muß, ist die Tatsache, daß man heutzutage nicht von einer langfristigen Verfügbarkeit von GUI Bibliotheken ausgehen kann. Für die Anwendungsentwickler ergibt sich somit die Notwendigkeit, Möglichkeiten zu untersuchen, durch die ein möglichst einfacher Austausch der Oberfläche realisiert werden kann. Vor diesem Hintergrund der Austauschbarkeit von Bibliotheken, welche die Widgets zur Verfügung stellen, ist es nicht wünschenswert, diese technischen Aspekte der Oberfläche tief in der Software zu verwurzeln. Aus diesem Grunde sind in dem am Arbeitsbereich Softwaretechnik vertretenen Werkzeug- und Material- Ansatz[3], Konstruktionsprinzipien vorgesehen, welche die technischen Aspekte der die Oberfläche definierenden Widgets gegenüber einem Werkzeug verbergen. Die Betrachtung und die mögliche Erweiterung dieser Prinzipien stellt in der Folge den Schwerpunkt unserer Arbeit dar. Unser Hauptaugenmerk haben wir allerdings auf ein Konstruktionsprinzip für die Anbindung von Widgets gerichtet, die einen großen Funktionsumfang aufweisen. Zu dieser Art von Widgets gehören zum Beispiel Tabellen, Trees aber auch eigens für einen Anwendungskontext definierte Widgets mit einer entsprechenden Komplexität.

In den Kapiteln 3 und 4 werden wir die Grundlagen zu diesem Themenkreis erläutern. Hierbei beschäftigen wir uns im Kapitel 3 mit eher allgemein angelegten Themen, wo hingegen wir im vierten Kapitel auf die speziellen Belange der Oberflächenanbindung im Zusammenhang mit dem WAM Leitbild eingehen werden. Wir haben versucht, diesen Abschnitt knapp zu halten mit der Maßgabe nur die für das Verständnis unseres speziellen Kontextes interessanten Teile zu erläutern.

Das Kapitel 5 beschäftigt sich mit einem technischen Konstruktionsprinzip zur Einbindung eines Widgets für Tabellen. Wir haben uns an dieser Stelle für Tabellen entschieden, da sie zum einen sehr wichtig für die Strukturierung von großen Informationsmengen innerhalb von Oberflächen sind. Zum anderen weisen Tabellen eine üppige Funktionalität auf, anhand derer die Konsequenzen unserer Konstruktion deutlich zu erkennen sind. Eines unserer Ziele, die dieser Arbeit zugrunde liegen, besteht darin, eine Konstruktion vorzustellen, die auch für andere Widgets tragfähig ist, ohne daß sie übermäßig kompliziert, gemessen an der Handhabung durch einen Anwendungsentwickler, wird. Diesen Aspekt bearbeiten wir am Ende des fünften Kapitels, wo wir versuchen, unsere Konstruktion auf ein Treewidget zu übertragen. Sowohl das Tabellenwidget, als auch das Treewidget sind von uns implementiert und in das am Arbeitsbereich entstandene JWAM[4] Rahmenwerk integriert worden.

---

[3] WAM steht als Kurzform für das Leitbild Werkzeug, Automat, Material.

[4] JWAM steht als Abkürzung für das am Arbeitsbereich entwickelte Rahmenwerk. Das J zeigt an, daß dieses Rahmenwerk unter der Programmiersprache Java realisiert wurde und ist dem Kürzel für den dem Rahmenwerk zugrunde liegenden Ansatz vorangestellt.

Im Umgang mit Widgets, die einen großen Datenbestand repräsentieren (zum Beispiel Tabellen), erwartet der Benutzer die Möglichkeit, bestimmte Einstellungen an dem Widget hinsichtlich der Farbe oder des Stils vornehmen zu können. Diese Erwartung entsteht unserer Meinung nach aus der Notwendigkeit des Benutzers, sich relevante Merkmale aus einem großen Wertbestand herauszusuchen und diese individuell zu markieren. Diese Notwendigkeit findet man allerdings nicht bei den einfachen Widgets, wie zum Beispiel einem Textfeld. In dem zur Zeit vorliegenden JWAM - Rahmenwerk ist dies über das Konzept der Interaktionsformen nicht vorgesehen. Wir zeigen deshalb im sechsten Kapitel einen Weg auf, mit dem es möglich ist, Einstellungen (Farbe, Textstil, etc.) an den Widgets direkt vorzunehmen.

Betrachtet man große Widgets, wie Tabellen und Trees, so erkennt man, daß die technische Konstruktion immer undurchsichtiger wird. Gleichzeitig birgt jedes Widget in sich eine große Menge an Funktionalität, die auf die verschiedenen Aufgaben ausgerichtet ist. Als Folge der beiden vorgenannten Punkte wird es uns erschwert, anhand der Konstruktion Unterschiede oder Entsprechungen von verschiedenen Widgets heraus zu arbeiten. Dirk Riehle hat in diesem Zusammenhang einen Vorschlag gemacht, wie man das Rollenkonzept auf Entwurfsmuster übertragen kann. Dieses eröffnet ihm die Möglichkeit, auf einer abstrakten Ebene über die Bestandteile dieser Muster zu sprechen, obwohl sie zum Beispiel in einem Objekt zusammengefaßt wurden. Unter diesem Gesichtspunkt ergeben sich unserer Meinung nach auch im Hinblick auf die Diskussion und Beschreibung von Widgets Vorteile, so daß wir im siebten Kapitel versuchen möchten, das Rollenkonzept auf Widgets zu übertragen.

Abschließend möchten wir an dieser Stelle besonders Prof. Dr. Heinz Züllighoven für die konstruktive Unterstützung bei der Erstellung dieser Arbeit danken. Insbesondere gilt unser Dank auch Prof. Dr. Horst Oberquelle als Zweitbetreuer dieser Arbeit. Nicht zu vergessen sind Henning Wolf, Stefan Roock, Martin Lippert und die Mitglieder des JWAM Teams, die zum Gelingen unserer Arbeit durch lebhafte Diskussionen beigetragen haben.

2 Verwendete Notationen

In dieser Diplomarbeit werden wir verschiedene Arten von Diagrammen verwenden um Ideen und Zusammenhänge zu illustrieren. Wir benutzen in Anlehnung an die „Unified Modeling Language" (UML) zwei verschiedene Arten der formalen Notation in dieser Diplomarbeit:

1. *Klassendiagramme* zeigen Klassen, ihre Struktur und ihre statische Beziehung untereinander.

2. *Interaktionsdiagramme* verdeutlichen den Ablauf des Kontrollflusses zwischen den Objekten.

Im folgenden werden wir die beiden Diagrammtypen kurz vorstellen.

2.1 Klassendiagramme

Die Abbildung 2-1 zeigt die Notation für abstrakte und konkrete Klassen. Eine Klasse wird durch einen rechteckigen Kasten mit dem fettgedruckten Klassennamen oben im Kasten dargestellt.

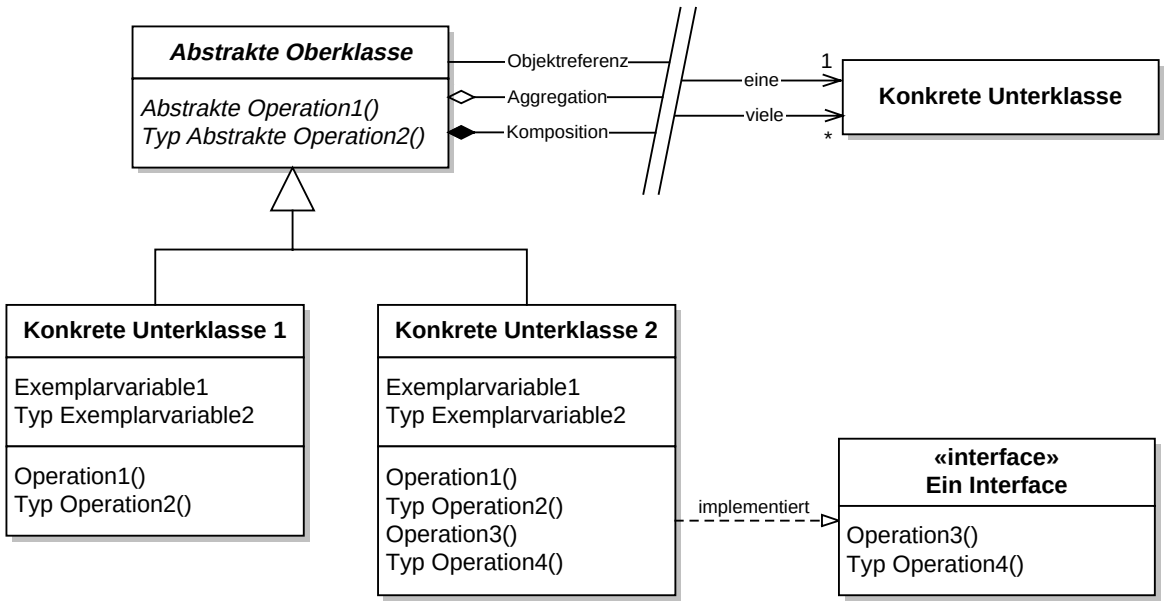

Abbildung 2-1: Ein Klassendiagramm

Wichtige Operationen und Exemplarvariablen erscheinen unterhalb des Klassennamen. Eine Schnittstellenbeschreibung[5] wird als rechteckiger Kasten mit dem Schlüsselwort «interface» oberhalb des Schnittstellennamen dargestellt.

[5] In Java werden die Schnittstellenbeschreibungen als Interfaces bezeichnet.

Wie bei den Klassen beinhaltet die Darstellung einer Schnittstellenbeschreibung einen fett geschriebenen Schnittstellennamen und die Operationen der Schnittstelle. Alle kursiven Bezeichnungen stellen abstrakte Klassen oder Operationen dar. Die unterschiedlichen Arten von Beziehungen zwischen Klassen werden durch verschiedene Linienarten dargestellt. Die Darstellung für Klassenvererbung ist das Dreieck, das die Unterklasse mit seiner Oberklasse verbindet. Eine Aggregationsbeziehung wird durch einen Linie mit Pfeilkopf und einem nicht ausgefüllten Diamanten am Ausgangspunkt dargestellt. Der Pfeil zeigt dabei auf die Klasse, die aggregiert wird. Wenn der Diamant schwarz ausgefüllt ist, dann steht dieses Linie für eine Kompositionsbeziehung. Eine einfache Pfeilverbindung ohne Diamant zeigt eine einfache Objektreferenz (Bekanntschaftsbeziehung) an. Wenn eine Zahl am Anfang der Linien oder am Pfeilkopf erscheint, so ist gemeint, daß zur Laufzeit mehrere Objekte referenziert oder aggregiert werden können. Ein Stern hingegen bedeutet, daß beliebig viele Referenzen möglich sind. Eine gestrichelte Linie mit einem nicht ausgefüllten Pfeilkopf verbindet eine Klasse mit einer Schnittstellenbeschreibung, das heißt die Klasse implementiert, bzw. erfüllt diese Schnittstellenbeschreibung.

In der als Beispiel dienenden Abbildung 2-1 erben die "Konkrete Unterklasse 1" und die "Konkrete Unterklasse 2" von der "Abstrakten Oberklasse". Zusätzlich implementiert die "Konkrete Unterklasse 2" die Operationen der Schnittstellenbeschreibung „Ein Interface".

2.2 Interaktionsdiagramme

Ein Interaktionsdiagramm zeigt die zeitliche Reihenfolge des Aufrufs von Methoden zwischen Objekten auf. Die Zeitachse verläuft bei einem Interaktionsdiagramm von oben nach unten. Eine senkrechte Linie zeigt die Lebensdauer eines Objektes an. Wenn das Objekt nicht vor Beginn der Zeitaufnahme des Diagramms erzeugt wird, erscheint der Kasten mit dem Objektnamen erst zum Zeitpunkt seiner Erzeugung.

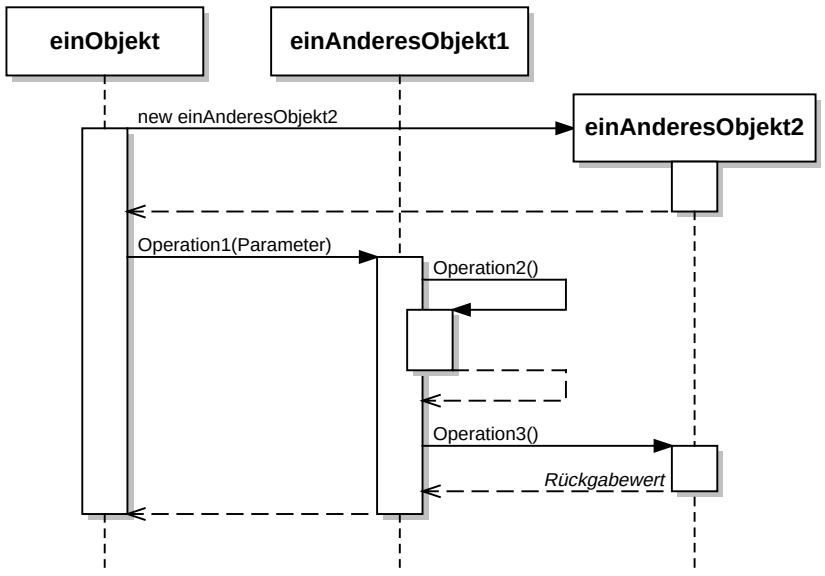

Abbildung 2-2: Notation eines Interaktionsdiagrammes

Ein vertikales Rechteck zeigt an, daß ein Objekt aktiv ist und Operationsaufrufe abarbeitet. Eine Operation kann weitere Operationen an anderen Objekten aufrufen. Diese Aufrufe werden durch eine horizontale Linie mit einem Pfeil dargestellt. Der Pfeil zeigt dabei auf das Objekt, an dem die Operation aufgerufen wird. Der Name der

aufgerufenen Operation wird oberhalb des Pfeils angegeben. Operationsaufrufe am Objekt selbst werden durch einen Pfeil dargestellt, der auf das Objekt selber zeigt. Falls ein Operationsaufruf einen Rückgabewert liefert, so wird die Rückgabe dies Wertes in Form einer gestrichelten Linie mit einem Pfeil von dem Objekt, das den Wert liefert, zum Objekt, das den Wert empfängt angezeigt. Der Typ des Rückgabewertes wird in kursiver Schrift oberhalb der Linie angegeben.

3 Grundlagen

Der Entwurf und die Programmierung von reaktiven Softwaresystemen mit einer interaktiven grafischen Benutzerschnittstelle (GUI) ist heutzutage eine komplexe und aufwendige Aufgabe geworden, da die Erwartungen, die an eine gute und intuitive Benutzungsschnittstelle gestellt werden, immer umfangreicher werden. Neben den Anforderungen aus der Softwareergonomie müssen moderne GUI's mit den unterschiedlichsten asynchronen Eingabegeräten (unter anderem Tastatur und Maus) und somit verschiedenen Wegen zur Eingabe ein und desselben Befehls sowie aufwendigen Grafiken umgehen können. Noch während der Ausführung von Operationen sollen sie ein sofortiges semantisches Feedback[6] bieten. Es ist zu erwarten, daß die Anforderungen noch weiter steigen werden, da es vermehrt zum Einsatz von multimedialen Techniken kommen wird und auch die Komplexität der Interaktion des Benutzers mit der grafischen Benutzerschnittstelle durch neue Eingabegeräte wie Sprach- und Gestenerkennung steigen wird.

Nachfolgend werden wir kurz die Eigenschaften von Toolkits beschreiben, die dem Anwendungsentwickler die für die Erstellung einer grafischen Oberfläche notwendige Funktionalität zur Verfügung stellen.

3.1 Überblick über die Verwendung von Widgets

Um den vielfältigen Anforderungen an eine grafische Benutzerschnittstelle gerecht werden zu können, ist es für einen Programmierer kaum möglich, ohne die Hilfe einer GUI Klassenbibliothek auszukommen. In Anlehnung an [Myers96] und [Blee99] verstehen wir unter einer solchen Klassenbibliothek, eine Ansammlung von wiederverwendbaren Oberflächenelementen (Widgets). Die in diesen Bibliotheken enthaltenen Widgets können in den unterschiedlichen Anwendungen verwendet werden. Die Abbildung 3-1 zeigt eine kleine Auswahl von Widgets aus der SWING Klassenbibliothek des „Java Development Kit[7] 1.2" von Sun. Diese Klassenbibliotheken werden auch GUI Toolkits genannt. Sie enthalten häufig auch zusätzliche Werkzeuge, welche die einfache und schnelle Erzeugung von Oberflächen gestatten, die sogenannten GUI Builder.

Für die Entwicklung und Implementierung von größeren Softwareprodukten ist die alleinige Verwendung eines GUI Toolkits nicht ausreichend, da der Umgang mit den zahlreichen Klassen eines Toolkits im Vorwege bereits einen großen Einarbeitungsaufwand erfordert und die Konstruktion der Widgets teilweise schwer durchschaubar ist. Deshalb sind die GUI Toolkits häufig Bestandteil eines Rahmenwerkes (Application Framework). Unter einem Rahmenwerk verstehen wir eine Sammlung von kooperierenden Klassen, welche die grundlegende Architektur einer Anwendung vorgeben und bereits den Kontrollfluß der Anwendung festlegen. Rahmenwerke dieser Art werden oft von den

---

[6] Feedback beschreibt im Kontext von Benutzungsoberflächen Rückkopplung mit Hilfe von optischen und / oder akustischen Signalen. Wird ausgehend von einer (lokalen) Aktion des Benutzers für die Rückkopplung Wissen aus der Funktionskomponente benötigt, sprechen wir von semantischem Feedback.

[7] Für das Java Development Kit werden wir in Folge auch die Kurzform JDK verwenden.

Betriebssystemherstellern angeboten, um die Entwicklung von Anwendungen für ihre Plattform zu unterstützen. Die „Microsoft Foundation Classes" [Micro93], kurz MFC, oder „MacApp" von Apple [Wilson90] sind bekannte Beispiele für solche Rahmenwerke.

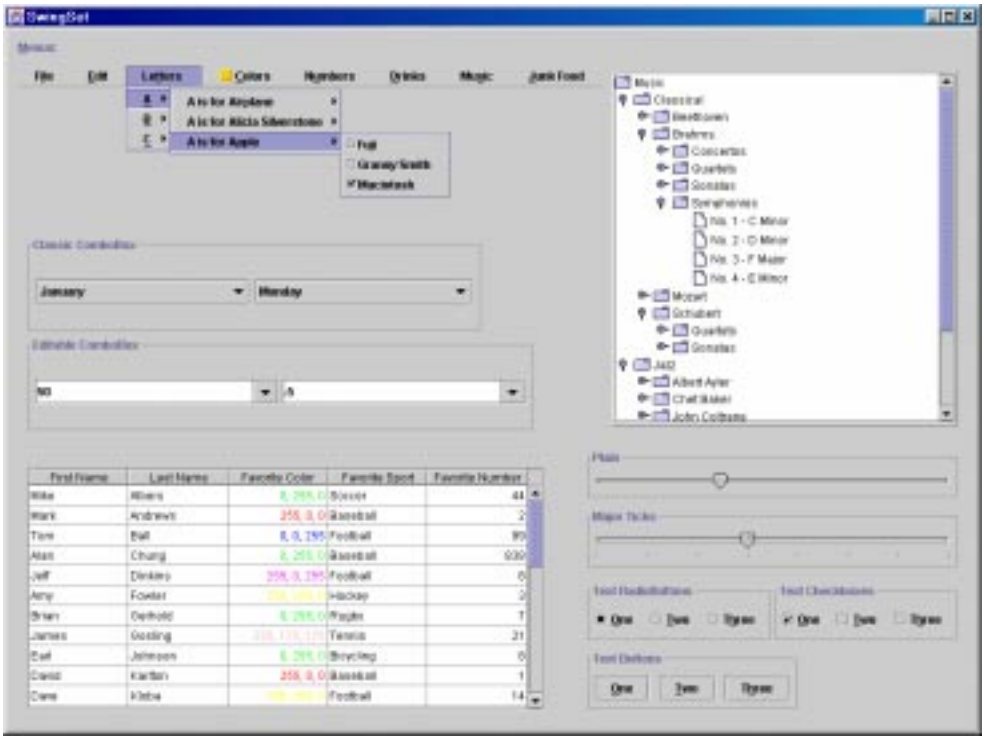

Abbildung 3-1: Swing - Widgets im Java „Look and Feel"

Der Einsatz von GUI Toolkits bietet eine Menge von Vorteilen[8]. Die Qualität der Benutzeroberfläche kann durch eine schnellere Implementation und ein schnelleres Prototyping gesteigert werden, da hierdurch ein rascherer Durchlauf des Autor - Kritiker-Zykluses erreicht wird. Unmittelbar damit verbunden ist somit die Möglichkeit, eine bessere Anpassung des GUI's bei gleichbleibendem Zeitaufwand zu erreichen. Zusätzlich lohnen sich auch längere Einarbeitungsphasen in ein spezielles Toolkit, sofern dieses in verschiedenen Applikationen mehrfach Verwendung findet. Normalerweise resultiert aus dieser Mehrfachverwendung eine bessere Ausnutzung der Möglichkeiten eines Toolkits. Die Verwendung eines Toolkits in mehreren Projekten hat somit auch zur Folge, daß verschiedene Applikationen eine konsistente und einheitliche Benutzungsoberfläche vorweisen. Die Unterstützung von Undo- und Hilfe- Funktionen aus dem Toolkit heraus, erleichtern dem Anwendungsentwickler die Bereitstellung dieser aus heutigen Softwareprodukten nicht mehr wegzudenkenden Funktionen. Erst durch den vermehrten Einsatz von GUI - Buildern für diese Toolkits ist es unter anderem auch für Interfacedesigner und Psychologen möglich geworden, mit keinen oder wenig Programmierkenntnissen Oberflächen zu erstellen.

Neben der möglichen Steigerung der Qualität einer Oberfläche besteht ein weiterer Vorteil im Hinblick auf den Einsatz von Toolkits darin, daß der so entstandene Code für die Benutzungsoberfläche einfacher und wirtschaftlicher zu erzeugen und zu warten ist. Hierbei ist immer die Qualität des verwendeten Toolkits ausschlaggebend. Mit Hilfe eines Toolkits kann die Spezifikation der Benutzungsschnittstelle einfacher dargestellt, validiert

[8] Siehe hierzu auch [Myers96].

und evaluiert werden. Auch muß in der Regel weniger Code geschrieben werden, da viele Funktionen durch das Toolkit bereits abgedeckt werden. Es erfolgt eine bessere Modularisierung der Applikation durch die Trennung der Klassen für die Oberflächenerstellung von den anderen Klassen einer Applikation. Dies ermöglicht Änderungen an den GUI - Klassen ohne größere Auswirkungen auf die Applikations - Klassen, sofern die Schnittstellen dabei stabil bleiben. Die Portierung auf andere Hardware- oder Softwareumgebungen wird dadurch vereinfacht, daß die Abhängigkeiten der Applikation von Ein- und Ausgabegeräten in dem GUI Toolkit isoliert sind. Da GUI Toolkits große Teile der Komplexität des Betriebssystems verbergen, benötigen Programmierer weniger Wissen bezüglich der Betriebssystem nahen Oberflächenprogrammierung.

Kritisch zur Verwendung von Rahmenwerken merken Züllighoven [Züll98] und Görtz [Görtz98] an, daß sowohl in Rahmenwerken als auch in GUI Toolkits die Vorgabe eines Kontrollflusses auch erhebliche Probleme verursachen kann. Der Anwendungsentwickler, der ein Rahmenwerk oder Teile davon verwendet, möchte dabei nicht die Architektur seiner Anwendung durch das Rahmenwerk bestimmen lassen. Demzufolge bleiben ihm nur die zwei folgenden Möglichkeiten. Entweder übernimmt er die Architektur des Rahmenwerks für seine Anwendung, oder er programmiert mit viel Mühe gegen das Rahmenwerk eine eigene Architektur. Ein weiteres Problem von GUI Toolkits besteht in ihrer Veränderlichkeit über die Zeit. Viele GUI Toolkits haben in der Geschichte ihrer Entwicklung wiederholt die Schnittstelle geändert, oder sie sind sogar ganz vom Markt verschwunden.

Um diese Probleme zu umgehen, wurde zum Beispiel im JWAM Rahmenwerk auf eine lose Koppelung zwischen der Interaktionskomponente eines Werkzeuges und dem verwendeten GUI Toolkit geachtet. Die lose Koppelung wird durch die Verwendung von Interaktionstypen oder Interaktionsformen anstelle der konkreten Widgets eines GUI Toolkits erreicht. In Kapitel 4 werden wir die beiden Ansätze im Detail diskutieren.

GUI Toolkits lassen sich wie Rahmenwerke in White - Box und Black - Box Toolkits einteilen (siehe auch [Weiß97]). Objektorientierte White - Box Toolkits werden durch Subklassenbildung und Überschreiben von Methoden an den konkreten Anwendungskontext angepaßt. Die Beziehungen und das Zusammenspiel der Klassen beziehungsweise Objekte ist durch das Toolkit bereits vorgegeben und muß nur durch anwendungsspezifisches Verhalten ergänzt werden. Die Verwendung von solchen Toolkits setzt jedoch voraus, daß der Anwendungsprogrammierer die interne Implementierung der Klassen und die Interaktion zwischen den Objekten zur Laufzeit verstanden hat.

Ein Beispiel für ein White - Box Toolkit in Java ist das „Abstract Windows Toolkit“[9] mit dem Event Modell des JDK 1.0. Das AWT beinhaltet vordefinierte grafische Komponenten, aus denen die Oberfläche eines Programmes zusammengesetzt werden kann. Die Komponenten lassen sich in zwei Gruppen einteilen. Zum einen die Container, die sich dadurch auszeichnen, daß sie andere Komponenten des AWT enthalten können und in der Regel keine Benutzerinteraktion entgegen nehmen. Durch die Tatsache, daß Container auch wieder Container enthalten können, kann eine Hierarchie aufgebaut beziehungsweise eine Gliederung der Oberflächenkomponenten vorgenommen werden. Die Widgets auf der anderen Seite werden direkt auf einem Grafikkontext dargestellt und nehmen die Benutzeraktionen entgegen. Damit ein Widget auf Benutzerinteraktionen reagieren kann, muß entweder die zentrale Event Methode `handleEvent(evt)` oder eine der folgenden Hilfsmethoden redefiniert werden.

[9] Häufig wird hier die Abkürzung AWT verwendet.

- `keyDown(evt, key)`
- `mouseMove(evt, x, y)`

In dem nachstehenden Interaktionsdiagramm ist der Ablauf bei der Betätigung eines Buttons dargestellt, wobei dieser Button über ein Panel in ein Fenster integriert wurde. Der boolesche Rückgabewert einer Event Methode gibt an, ob daß Ereignis vom Widget bereits bearbeitet worden ist. Falls dies nicht der Fall ist, wird das Event nach dem Entwurfsmuster der „Zuständigkeitskette" (siehe [GHJV96]) an die nächst höhere Komponente in der Oberflächenhierarchie weitergeleitet.

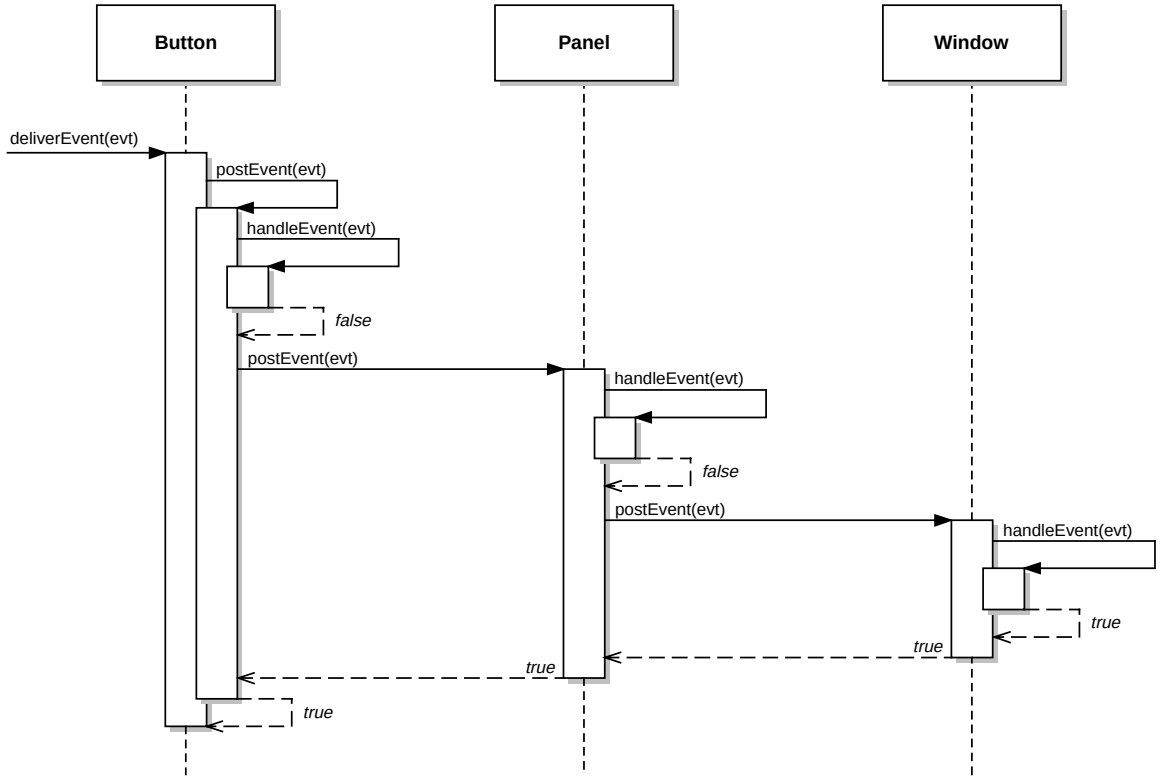

Abbildung 3-2: Event Loop des AWT

Die Default - Implementierung aller Event Methoden in den AWT Komponenten sieht so aus, daß immer der Wert `false` zurückgegeben wird. Demzufolge beachtet in der Standardimplementierung keine Komponente die eintreffenden Ereignisse, sondern leitet diese nur unbearbeitet weiter. Damit eine Applikation, die AWT Komponenten benutzt, auf Benutzerinteraktionen und damit auch auf Events reagieren kann, müssen eigene Komponenten von allen benutzten AWT Komponenten abgeleitet und die entsprechenden Event Methoden redefiniert werden. Alternativ würde es auch ausreichen, nur die Event Methoden des obersten Container zu redefinieren, da die unbearbeiteten Events in der Hierarchie bis zum obersten Container weitergeleitet werden. Dies hätte jedoch zur Folge, daß dieser Container die Events aller Widgets behandeln müßte. Die Implementierung ist entsprechend aufwendig, schlecht zu warten und damit fehleranfällig. Ein weiterer Nachteil ist die umständliche und überflüssige Weiterleitung von Events. Die Events werden über alle Hierarchiestufen weitergeleitet, obwohl sich nur die letzte Komponente in der Zuständigkeitskette dafür interessiert. Dieses Event Modell funktioniert nur bei kleinen Programmen gut. Bei größeren Programmen ist es jedoch zu inflexibel und aufwendig. Die Firma Sun empfiehlt aus diesem Grund, dieses Event Modell nicht mehr zu

benutzen. Die AWT Komponenten mit dem Event Modell des JDK 1.0 sollten deshalb nur noch für Internet - Anwendungen, die als Java Applet in einem Web Browser laufen und ein großes Publikum erreichen sollen, verwendet werden. Diese Entscheidung basiert darauf, daß alle Web Browser die Java Applets unterstützen, auf jeden Fall mit den AWT Komponenten des JDK 1.0 umgehen können.

Im Gegensatz zu den White - Box Toolkits sind die objektorientierten Black - Box Toolkits dadurch gekennzeichnet, daß die verschiedenen Klassen direkt instanziert und benutzt werden. Die Anpassung an den konkreten Anwendungskontext erfolgt durch die spezifische Parametrierung der erzeugten Objekte. Black - Box Toolkits haben den großen Vorteil, daß die Einarbeitungszeit und der Lernaufwand gering ist. Auf der anderen Seite haben sie jedoch gegenüber einem White - Box Toolkit den Nachteil der geringeren Flexibilität und Anpaßbarkeit.

Das AWT mit dem Event Modell des JDK 1.1 kann als Black - Box Toolkit verwendet werden. Allerdings kann auch diese Version des AWT, entsprechend der Beschreibung in den vorigen Absätzen, wie ein White - Box Toolkit erweitert werden. Die Anbindung der GUI Komponenten funktioniert bei dieser AWT Version nach dem Beobachtermuster. Bei dem Beobachtermuster kennt das Subjekt, das zu beobachtende Objekt, seine Beobachter nur unter einer bestimmten Schnittstelle und ruft bei Zustandsänderungen eine definierte Methode dieser Schnittstelle auf. Damit das Subjekt eine Liste von Beobachtern führen kann, muß es selber Methoden zum An- und Abmelden von Beobachtern anbieten. Da Java keinen Callback Mechanismus per Methodenzeiger wie in C oder C++ kennt, werden an den beobachtbaren GUI Komponenten sogenannte „Event - Listener" Objekte registriert. Eine GUI Komponente als Subjekt informiert ihre Event Listener über ein Ereignis durch den Aufruf einer festgelegten Methode und Übergabe eines speziellen Event Objektes. Eine Klasse wird zum Event Listener, wenn sie mindestens eines der elf vorhandenen Listener Interfaces implementiert. Jedes Listener Interface spezifiziert die Signatur einer Methode mit einem speziellen Event Objekt, welches im Ereignisfall gerufen wird. Das übergebene Event Objekt enthält dabei nähere Informationen über das eingetretene Ereignis.

Bei dieser Art der Implementierung des Beobachtermusters bieten sich Adapterklassen als Event Listener an, welche an den GUI Komponenten registriert werden und welche die entsprechenden Methoden in der Interaktionskomponente der Applikation aufrufen. Für große Oberflächen mit vielen Widgets werden entsprechend viele Adapterklassen benötigt. In Java bietet sich die Verwendung von inneren Klassen als Adapterklassen an.

Toolkits, welche für die Entwicklung von Oberflächen unter verschieden Betriebssystemen geeignet sind, werden virtuelle Toolkits genannt. Virtuelle Toolkits versuchen, die Unterschiede und Eigenheiten der verschiedenen Fenstersysteme der Betriebssysteme zu verbergen. Zu diesem Zweck stellen sie virtuelle Widgets zur Verfügung, die unter all denjenigen Fenstersystemen lauffähig sind, die vom virtuellen Toolkit unterstützt werden. Dabei werden meist auch das „Look and Feel" und die Styleguides des jeweiligen Betriebssystem eingehalten. Die im JDK 1.2 enthaltenen GUI Toolkits AWT und SWING sind virtuelle Toolkits, da sie auf jedem die Programmiersprache Java unterstützenden Betriebssystem lauffähig sind.

Bei der Implementierung eines virtuellen Toolkits bieten sich zwei Ansätze an. Der eine Ansatz geht von einem minimalen Satz von Widgets aus, welcher von allen Fenstersystemen unterstützt wird. Das virtuelle Toolkit kapselt dabei die konkreten GUI Toolkits der Fenstersysteme und bildet ein virtuelles Widget auf ein konkretes Widget der Zielplattform ab. Der Nachteil dieser Lösung ist, daß nur die Schnittmenge aller Widgets der ver-

schiedenen unterstützten Fenstersysteme vom virtuellen Toolkit abgedeckt werden kann und daß für jedes Fenstersystem eine spezielle Version des virtuellen Toolkits geschrieben und angepaßt werden muß. Das schon erwähnte AWT von Sun ist ein Beispiel für solch ein virtuelles Toolkit. Die AWT Widgets werden über Peer Objekte direkt an die Widgets des jeweiligen Fenstersystems angebunden. Demzufolge müssen für eine Systemplattform zusammen mit der entsprechenden Laufzeitumgebung auch passende, nur auf dieser Plattform laufende, Peer Objekte ausgeliefert werden. Diese Lösung hat den Vorteil, daß das „Look and Feel" der Zielplattform eingehalten werden kann, da die konkreten Widgets der Zielplattform benutzt werden.

Beim zweiten Ansatz werden die verschiedenen Widgets der unterstützten Fenstersysteme in dem virtuellen Toolkit nachimplementiert. Zu diesem Zweck werden primitive Zeichenoperationen des jeweiligen Fenstersystems benutzt. Diese Art von virtuellen Toolkits bieten zwar meist einen reichhaltigen Satz von Widgets, neigen jedoch dazu, die Styleguides der verschiedene Fenstersysteme zu verletzen, da sie unter anderem Funktionen und Widgets zur Verfügung stellen, die in den verschiedenen Fenstersystemen beziehungsweise Styleguides nicht immer vorgesehen sind. Durch die Nachimplementierung aller Widgets werden die Laufzeitbibliotheken entsprechender virtueller Toolkits meist sehr umfangreich und ihr Laufzeitverhalten ist schlechter als die Verwendung nativer Widgets. Das Swing Toolkit aus dem JDK 1.2 ist ein Beispiel für diese Art von virtuellen Toolkits.

3.1.1 Innerer Aufbau eines Toolkits am Beispiel MVC

In Bezug auf die Implementierung eines Toolkits gibt es vielfältige Möglichkeiten, die unter anderem von den Konzepten und Möglichkeiten der verwendeten Programmiersprache geprägt sind. In den letzten Jahren hat sich jedoch gezeigt, daß ein Entwurfsmuster besonders geeignet scheint und in der Folge auch häufig angewandt wird (siehe [Howa95], [SteLam97] und [WeiAsb98]). Dieses Entwurfsmuster ist unter dem Namen Model - View - Controller[10] bekannt. Das MVC Muster von Reenskaug [Reen95] beschreibt die Interaktion und die Aufgaben von den drei Objekten Model, View und Controller. MVC ist ein Muster, das sich sowohl bei der Anwendungsentwicklung im allgemeinen, als auch bei der Entwicklung von Widgets in GUI - Toolkits anwenden läßt. Im Einflußbereich der Programmiersprache Smalltalk ist MVC sehr stark verbreitet. Im folgenden beschreiben wir kurz das MVC Muster, da wir bei der Konstruktion unseres Tabellenwidgets im fünften Kapitel einen ähnlichen Ansatz verfolgen.

Das Modell ist für das Management der domänenspezifischen Information einer Anwendung beziehungsweise eines Widgets zuständig. Zu diesem Zweck stellt es Methoden zur Sondierung und Manipulation der im Modell gespeicherten Informationen zur Verfügung. Das Modell besitzt jedoch keinerlei Angaben zur grafischen Repräsentation der Informationen und nimmt auch keine Benutzeraktionen entgegen.

Die View hat die Aufgabe, die im Modell gespeicherten Informationen auf einem Ausgabemedium darzustellen. Zu diesem Zweck greift sie auf die sondierenden Methoden des Modells zurück. Eine View stellt in der Regel immer nur eine Sichtweise auf ein Modell zur Verfügung. Es ist daher durchaus möglich und üblich, daß für ein Modell mehrere Views existieren, die jeweils eine andere Sicht auf die im Modell hinterlegten Informationen bieten. Damit eine View immer den aktuellen Zustand ihres Modells reflektieren kann,

[10] Oftmals verwenden wir in diesem Zusammenhang die Kurzform MVC.

muß sie über Zustandsänderungen informiert werden. Zu diesem Zweck registriert sich die View nach dem Beobachtermuster als Beobachter an dem Modell. Das Modell kann nun bei auftretenden Zustandsänderungen alle am ihm registrierten Views über diese informieren.

Der Controller ist ein der View direkt zugeordnetes Objekt, welches die Interaktion mit dem Benutzer steuert und die aufgetreten Events in Methodenaufrufe umwandelt, die entweder an der View oder dem Modell aufgerufen werden. Nur der Controller darf die den Zustand des Modells verändernden Methoden aufrufen.

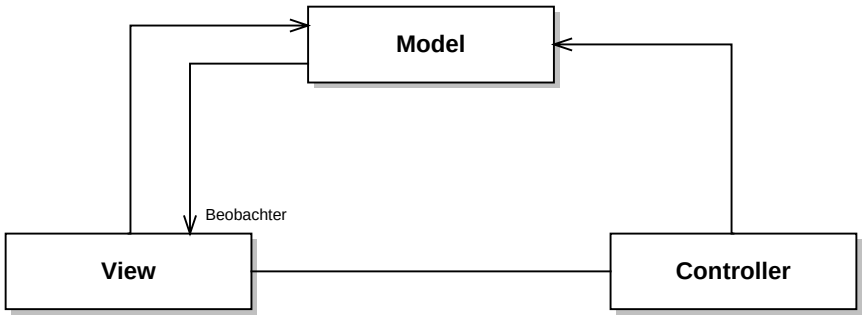

Abbildung 3-3: Klassendiagramm vom MVC - Muster

Die Bindung zwischen Controller und View ist meist sehr eng, da die möglichen Interaktionen, auf die der Controller reagieren muß, stark von der Repräsentation der Informationen abhängen. Ein Benutzer erwartet zum Beispiel von einer Liste einen anderen Umgang als von einem Eingabefeld. Technisch betrachtet ergeben sich an dieser Stelle auch Unterschiede in den zu verschickenden Events. Deshalb spricht man im MVC-Kontext auch von View / Controller Paaren. Das Klassendiagramm in der Abbildung 3-3 gibt die Beziehung von Modell, View und Controller wieder.

Es wurde lange Zeit eine Diskussion darüber geführt, ob das MVC - Muster besser nur mit zwei anstatt mit drei Objekten implementiert werden sollte. Im Falle von zwei Objekten werden die View und der Controller in einem Objekt zusammengefaßt. Reenskaug argumentiert in [Reen95] jedoch, daß diese Diskussion überflüssig sei, wenn man Modell, View und Controller nicht als Objekte, sondern als Rollen versteht. Dann ist es nämlich gleichgültig, ob eine konkrete Implementierung zwei oder drei Objekte für das Muster benötigt. Bei einer Implementierung mit zwei Objekten übernimmt ein Objekt dann zwei Rollen. Die Wahl der Implementation kann dann von anderen Aspekten geleitet werden. Im Swing Toolkit des JDK 1.2 vereinigen die Widgets die Rollen View und Controller in einem Objekt, das heißt es gibt in Swing nur Widgets und ihre Modelle.

3.2 Möglichkeiten der Kombination von Widgets

Im vorigen Abschnitt haben wir beschrieben, wie Widgets im allgemeinen dem Anwendungsentwickler zur Verfügung gestellt werden. Wir haben allerdings noch keine näheren Aussagen über die Eigenschaften von Widgets gemacht. Eine mögliche Eigenschaft von Widgets, wir meinen hier die Kombination von Widgets, möchten wir in der Folge näher vorstellen. Im Zuge dieser Vorstellung ist es unser Ziel zu motivieren, warum eine Kombination von Widgets vorteilhaft sein kann.

Innerhalb der Informatik ist das Thema der freien Kombinierbarkeit schon seit langem bekannt. So finden wir in [Züll94] zum Beispiel die Beschreibung des Prinzips der Orthogonalität in Bezug auf Operationen, die auf Datenstrukturen angewendet werden.

> *Bei der Auswahl der Operationen auf Datenstrukturen und Datentypen ist Orthogonalität ein softwaretechnisch relevantes Kriterium.*
>
> *Orthogonalität bedeutet bezogen auf eine Programmiersprache allgemein: Daß ein kleiner Satz elementarer Konstrukte einfach kombiniert werden kann, um daraus die Datenstrukturen aber auch Kontrollstrukturen aufzubauen. Dabei ist jede mögliche Kombination zulässig und sinnvoll.*

Sicherlich sind Widgets nicht ohne weiteres mit Datenstrukturen wie einem Kellerstapel zu vergleichen. Überträgt man jedoch den der Orthogonalität zugrunde liegenden allgemeinen Ansatz auf Widgets, so bedeutet dies, daß jede Kombination von Widgets möglich sein muß und dabei auch sinnvoll ist. Bildlich gesprochen werden also die Widgets selbst ineinander geschachtelt.

Eine andere Sicht auf die Kombination von Widgets erhalten wir, wenn wir ein Widget als eine spezielle Komponente betrachten. Diese Sicht auf Widgets ist unter anderem damit begründbar, daß die für ein bestimmtes Fenstersystem erzeugten Widgets ohne weiteres innerhalb dieser Systemgrenzen ausgetauscht werden können. Sichtbar wird diese Austauschbarkeit in dem sogenannten „Look and Feel" eines Systems. Betrachtet man also ein bestimmtes System, so stellt man ein einheitliches Aussehen der verwendeten Widgets fest, da alle auf dem gleichen Toolkit respektive Rahmenwerk basieren. Vor dem Hintergrund der aktuell geführten Komponentendiskussion schreibt Szyperski in [Szyp98, Seite 3] über die Verwendung von Komponenten.

> *One thing can be stated with certainty: components are for compositting. Composition enables prefabricated ‚things' to be reused by rearranging them in ever new composites.*

Eine mögliche Interpretation dieses Zitats geht in die gleiche Richtung, wie die oben angeführte Beschreibung des Prinzips der Orthogonalität. An dieser Stelle finden wir ebenfalls keine Einschränkung bezüglich der Kombination von Widgets. So sollte also die Kombination von beliebigen Komponenten untereinander möglich sein. Allerdings wird an dieser Stelle keine Aussage mehr hinsichtlich des Sinns einer solchen Kombination getroffen. Zusätzlich fordert Szyperski in [Szyp98, Seite 280] aber die Integration von Komponenten in ein Rahmenwerk, um die unabhängige Entwicklung von Komponenten zu ermöglichen.

> *Component frameworks are the most important step to lift component software off the ground. Most current emphasis has been on the construction of individual components and on the basic ‚wiring' support of components. It is thus highly unlikely that components developed independently under such conditions are able to cooperate usefully.*

Für uns bedeutet dies, daß der Entwurf von Widgets immer aufgrund der für ein Rahmenwerk definierten Prinzipien der Integration von Widgets erfolgen sollte, wenn eine Kooperation zwischen den Komponenten gewünscht ist. Bezogen auf den hier angesprochenen Themenkreis entsteht für uns der Eindruck, daß eine freie Kombination von

Widgets nur dann möglich ist, wenn dies losgelöst von den Details der konkreten Implementation erfolgen kann.

Die eigentliche Frage, die wir bis zu diesem Punkt aber außer Acht gelassen haben, lautet, ob die Möglichkeit der freien Kombination von Widgets überhaupt wünschenswert ist. Um diese Frage beantworten zu können, ist es notwendig zu klären, wann eine Kombination sinnvoll ist und wann nicht. So macht es sicherlich keinen Sinn, einen Button mit einem Textfeld zu einem Widget zu kombinieren, bei dem innerhalb eines Buttons eine Texteingabe möglich ist. Bei der Verwendung von Tabellen auf der anderen Seite ist es durchaus sinnvoll, die Darstellung der einzelnen Zellen über die Kombination eines Tabellenwidgets mit mehreren anderen Widgets zu definieren.

Abbildung 3-4: Zusammengesetztes Widget

Gleiches kann aber auch für spezielle Widgets eines Anwendungskontextes gelten, bei denen der zu bearbeitende Wert aus mehreren Teilen besteht. Ein beliebtes Beispiel hierfür ist die in der Abbildung 3-4 dargestellte Eingabe einer Bankleitzahl.

Dieses Widget besteht aus drei einzelnen Textfeldern, mit deren Hilfe die Bankleitzahl eingegeben werden kann. Zusätzlich existiert noch ein nicht editierbares Eingabefeld, in dem nach erfolgter Eingabe der Bankleitzahl der Klartext der Bankbezeichnung erscheint. Ohne weiteres kann man diese Eingabe auch über die Anordnung der einzelnen Widgets nachbilden. Eine Konsequenz dieser Nachbildung über mehrere einzelne Widgets besteht darin, daß die Abhängigkeiten der Felder untereinander in der Interaktionskomponente festgelegt werden müssen. Das heißt, obwohl alle Felder in einem direkten Zusammenhang stehen, werden sie auf der technischen Ebene als separate Elemente behandelt. Wird eine Bankleitzahl innerhalb des Werkzeuges aber als ein Wert angesehen, so muß die Interaktionskomponente zusätzlich diesen Wert aufbrechen und an die verschiedenen Widgets verteilen. Die Möglichkeit der Kombination von mehreren einzelnen Widgets zu einem zusammengehörigen Widget eröffnet uns also an dieser Stelle einen einfachen Weg, mit dessen Hilfe wir den in einem Anwendungskontext benötigten Fachwerten eine entsprechende Präsentation anbieten können. Da die Realisierung eines entsprechenden Umganges für einen Fachwert immer die Aufgabe eines Anwendungsentwicklers ist, entsteht hier kein zusätzlicher Aufwand. Dieser Umgang ist dann in der Konstruktion der Interaktionskomponente unmittelbar zu erkennen, da an dieser Stelle keine Abhängigkeiten für einen Wert innerhalb der Interaktionskomponente nachgebildet werden müssen. Wir können also an dieser Stelle die für eine Domäne spezifischen Fachwerte direkt an die für die Darstellung zuständigen Widgets weiterreichen. Hieraus ergibt sich ein weiterer Vorteil, der darin besteht, daß die Sicht auf einen Typ von Werten ohne größere Codeänderungen verändert werden kann, da sich nur die Präsentation des Wertes ändert nicht aber der Typ des darzustellenden Wertes.

Eine andere Form der Kombination von Widgets finden wir vor, wenn die Verwendung eines Widgets extrem flexibel erfolgen kann. Grundsätzlich werden wir diesen Aspekt nur bei Widgets vorfinden, deren Aufgaben darin bestehen, mehr als einen Typ von Werten anzuzeigen. Eine Tabelle ist ein gutes Beispiel für eine so geartete Verwendung. Das Design eines Tabellenwidgets erfolgt zu einem Zeitpunkt, an dem der Typ des Inhaltes dieses Widgets noch nicht bekannt ist. Dies steht im Gegensatz zu dem von uns gewählten Beispiel des Widgets für eine Bankleitzahl. Wollte man die fachlichen Aspekte des Umgangs mit einer speziellen Tabelle berücksichtigen, so hieße dies, für jeden Einsatzfall ein eigenes Tabellenwidget zu realisieren. Um diese Mehrarbeit zu verhindern, beschreiten die Hersteller von GUI Toolkits zur Zeit den Weg, daß die in einer Tabelle darzustellenden Werte auf einen Standarddatentyp abgebildet werden, der dann ohne große Probleme durch das Widget angezeigt werden kann. Dieser Kompromiß unterbindet aber die Möglichkeit, den speziellen Umgang eines Benutzers mit den Inhalten der Zellen einer Tabelle zu realisieren. Einen Vorschlag zur Lösung dieser Problematik haben wir in dem Kapitel 5 beschrieben.

Unserer Meinung nach ergeben sich somit zwei unterschiedliche Arten der Kombination von Widgets.

- Kombination von Widgets zur Designzeit (statisch)
- Kombination von Widgets zur Laufzeit (dynamisch)

Im Rückgriff auf die am Anfang dieses Abschnitts beschriebenen Bedeutungen von Orthogonalität und Kombination von Komponenten können wir nun auch die Frage nach dem Sinn der Kombination von Widgets beantworten. Grundsätzlich ist für uns die Kombination von Widgets sinnvoll und wünschenswert. Im Gegensatz zu der Definition von Orthogonalität gehen wir aber davon aus, daß nur ein geringer Teil der möglichen Kombinationen sinnvoll ist und unterstützt werden kann. Grundlage einer entsprechenden Kombinierbarkeit stellt unserer Ansicht nach aber immer eine sinnvolle Integration der Widgets in ein Rahmenwerk dar. In der Konsequenz bedeutet dies für uns, daß zum einen ein einheitliches Konstruktionsprinzip auf alle in einem Rahmenwerk vorhandenen Widgets angewendet wird und zum anderen, daß die Art der anwendbaren Kombination aus der jeweiligen Integration in das Rahmenwerk ablesbar sein muß, um Mißverständnissen vorzubeugen. Die Integration von Widgets in ein Rahmenwerk bedeutet nicht zwingend, daß ein elementarer Satz an Widgets neu entwickelt werden muß. Vielmehr bedeutet dies, daß das für eine Programmiersprache oder ein System ausgewählte GUI Toolkit in das eigene Rahmenwerk eingebettet wird. Technisch gesehen erfolgt dies über das Kapseln der in einem speziellen Toolkit vorhandenen Widgets, wodurch ein Bindeglied zwischen dem eigenen Rahmenwerk und einem möglicherweise fremden GUI Toolkit geschaffen wird. Die Notwendigkeit der Kombination von verschiedenen Widgets ist im Bereich der Anwendungsentwicklung nicht als eine Ausnahme anzusehen. Alleine die Berücksichtigung der für eine Domäne spezifischen Fachwerte führt zu der Überlegung wie diese Fachwerte im Rahmen einer grafischen Oberfläche sinnvoll dargestellt werden können.

Wie wir bereits erwähnt haben, spielt die Integration der Widgets in ein Rahmenwerk eine entscheidende Rolle. Aus diesem Grund werden wir im folgenden kurz beschreiben, an welcher Stelle die angesprochenen Komponenten in einem Rahmenwerk angesiedelt sind. Ausgangspunkt hierfür ist die Zuordnung der Schichten eines Rahmenwerkes zu den unterschiedlichen Kontexten der Softwareentwicklung, die in der nachstehenden Abbildung gezeigt werden. Diese Abbildung basiert auf den von Züllighoven in [Züll98] identifizierten Dimensionen. Jede der in Abbildung 3-5 benannten Achsen stellt einen Kontext der Softwareentwicklung dar und findet seine Ausprägung in den Schichten eines

Rahmenwerkes. Dem Technikkontext werden hierbei die Systembasis, der Technologiebereich und das Objekt - Metamodell zugeordnet.

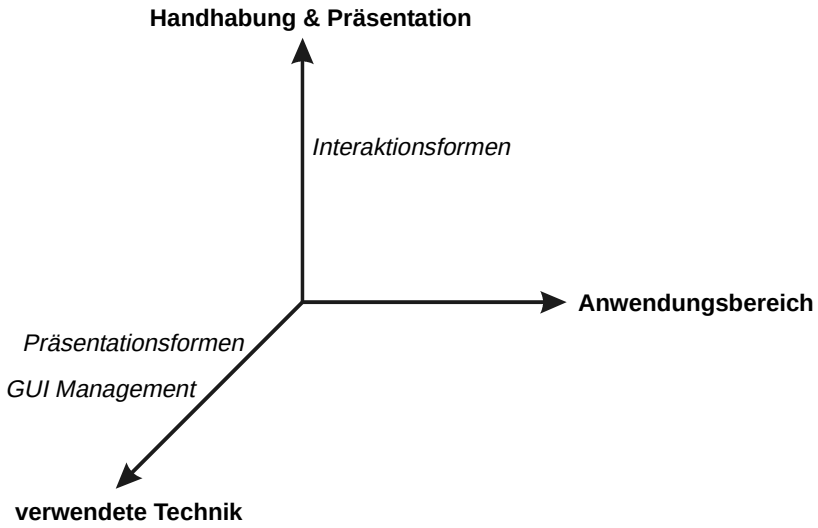

Abbildung 3-5: Widgets in den Kontexten der Softwareentwicklung

Zu den von der sogenannten Systembasisschicht angebotenen Dienstleistungen gehört unter anderem auch die Kapselung des jeweiligen Fenstersystems. Die Einordnung der Widgets erfolgt deshalb auf der Achse, welche die verwendete Technik repräsentiert. Genauer gesagt, erwarten wir, daß die Widgets eines Fenstersystems in der Systembasisschicht angelegt sind. In der Abbildung 3-5 verwenden wir den Ausdruck der Präsentationsformen, welche die konkreten Widgets eines Fenstersystems kapseln. Das Prinzip der Präsentationsformen werden wir in Kapitel 4.3 genauer erläutern. Für jedes vorhandene Widget existiert mindestens eine mögliche Umgangsform, die über das ebenfalls in Kapitel 4.3 vorgestellte Konzept der Interaktionsformen realisiert werden. Diese Interaktionsformen sind Bestandteil der Handhabungs- und Präsentationsschicht, in deren Rahmen die Art der Anbindung von Werkzeugen an das Fenstersystem festgelegt wird. Die in der Systembasis enthaltenen Widgets haben ihren Ursprung in dem jeweils verwendeten Fenstersystem respektive des verwendeten GUI Toolkits und sind somit auch nicht unbedingt auf einen Anwendungsbereich zugeschnitten. Werden von einem Anwendungsentwickler neue oder andere Widgets für einen bestimmten Anwendungskontext benötigt, so ist er aufgefordert, diese eigenständig zu integrieren. Diese Integration erfolgt dann aber nicht in der Systembasisschicht, sondern in dem sogenannten Gegenstandsbereich. Ursächlich ist hierfür, daß diese speziellen Widgets keinen allgemein gültigen Charakter haben. Der angesprochene Gegenstandsbereich ist im Rahmen der Abbildung 3-5 auf der als Anwendungsbereich bezeichneten Achse anzusiedeln. Die für einen Anwendungsbereich benötigten Fachwerte sind ebenfalls dem Gegenstandsbereich zuzuordnen. Im Rückgriff auf unser Beispiel des Bankleitzahlenwidgets wäre also sowohl der Fachwert Bankleitzahl, als auch das dazugehörige Widget Bestandteil des Gegenstandsbereiches. Sofern das Bankleitzahlenwidget nicht durch Kombination vorhandener Widgets entstanden ist, bedeutet dies, daß auch in dem Gegenstandsbereich eine Kapselung des Fenstersystems vorgenommen wird. Diese Kapselung betrifft dann genau diejenigen Elemente, die nicht durch das in der Systemschicht gekapselte GUI Toolkit bereitgestellt werden können. Dies kann dadurch vermieden werden, wenn sich zum Beispiel das Bankleitzahlenwidgets sich auf die bereits bestehenden Widgets abstützt. Sicherlich werden nicht alle Präsentationen für Fachwerte durch die Kombination von bestehenden Widgets realisierbar sein. Bei einem großen Teil

der Fachwerte eines Anwendungsbereiches wird es jedoch möglich sein, die Bearbeitung mit Hilfe von alphanumerischen Eingabefeldern zu realisieren.

Ein weiterer Punkt der im Zusammenhang mit der Kombination von Widgets positiv auffällt, betrifft den möglichen Austausch eines GUI Toolkits. Der Grund hierfür kann zum Beispiel in der Ablösung der ursprünglich gewählten Bibliothek durch eine neue Version liegen, wobei es zu Inkompatibilitäten mit der Vorgängerversion kommen kann. Andererseits ist alleine mit der Portierung eines Rahmenwerkes auf eine andere Programmiersprache ein solcher Austausch notwendig. Bei einem solchen Übergang wäre es von Nutzen, wenn ein großer Teil der von einem Anwendungsentwickler selbst erstellten Widgets auf Widgets basieren, die bereits elementarer Bestandteil der verwendeten GUI Bibliothek sind. Diese durch Kombination entstandenen Widgets können als eine Art Container angesehen werden, in denen mit Hilfe von eindeutigen Bezeichnern auf die zu verwendenden Widgets verwiesen wird. Bei einem Umstieg ist die erneute Integration der in einem Toolkit enthaltenen Widgets sicherlich unumgänglich. Für zusammengesetzte Widgets muß dann im schlechtesten Fall der diesem Widget zugrunde liegende Container an die neue Umgebung angepaßt werden. Bei einer konsequenten Umsetzung ist hier mit einer deutlichen Arbeitserleichterung zu rechnen.

Da wir der Meinung sind, daß die statische Kombination von Widgets ein wertvolles Instrument sein kann, wollen wir die hierbei bestehenden technischen Problemstellungen aufführen. Als zentrale Problemstellungen sehen wir folgendes:

- Konzeption und Realisierung eines allgemeinen Widget - Containers.
- Modellierung eines Layouts, welches die Anordnung von Widgets erlaubt.
- Definition von Methoden, über die der Umgang mit einem konkreten Fachwert erfolgt.

Der Widget - Container spielt in diesem Fall eine zentrale Rolle, da er die Grundlage für alle durch Kombination entstandenen Widgets darstellt. In dieser Klasse müssen zunächst die Grundlagen der Darstellung gelegt werden. Bezogen auf die Programmiersprache Java bietet sich in diesem Fall ein sogenanntes Panel an, in das beliebige AWT - Komponenten integriert werden können. Über dieses Panel wird dann die Darstellung der einzelnen Widgets koordiniert. Diese Panels eignen sich sehr gut für die Einbettung in die Oberfläche eines Werkzeuges, da sie selbst Elemente des GUI Toolkits darstellen und zumeist benutzt werden, um ein Fenster in mehrere logische Einheiten zu unterteilen. Ein großer Teil der hier benötigten Methoden kann also in dieser Klasse ohne zusätzliche Leistungen eines Anwendungsentwicklers implementiert werden und wird den erbenden Widgets als eine fertige Leistung angeboten. Zudem kann somit auch eine möglichst reibungsfreie Anbindung der Widgets an das Fenstersystem gewährleistet werden.

Bevor der eigentliche Vorgang der Darstellung von Widgets auf dem Grafikkontext erfolgt, muß aber definiert werden, wie die enthaltenen Widgets angeordnet werden sollen und welche Größe für dieses Widget anzunehmen ist. Hierbei ist zu beachten, daß der Anwendungsentwickler das Aussehen der Oberfläche bereits über eine andere Ressource[11] definiert hat. Insbesondere für die Ausmaße eines Widgets ist hier auf eine adäquate Behandlung zu achten. Eine Möglichkeit, die Anordnung von mehreren Widgets zu koordinieren besteht darin, das Panel als eine Matrix zu verstehen, so daß die Position

[11] Die Oberfläche kann zum Beispiel über eine eigene Klasse definiert werden, die dann durch das GUI Management zur Darstellung gebracht wird. Erzeugt werden diese Klassen in der Regel durch die GUI Builder eines Toolkits.

eines Widgets eindeutig über die Zeile und Spalte bestimmbar ist. Zusammen mit der vorzugebenden Ausdehnung eines jeden einzelnen Widgets kann hieraus die Positionierung abgeleitet und mit den Mitteln des Toolkits umgesetzt werden. Für das angesprochene Panel kann man zum Beispiel auf das `FlowLayout` des JDK zurückgreifen.

Damit letztlich auch die gewünschten Werte dargestellt und bearbeitet werden können, ist es unumgänglich, Methoden zu implementieren, die den entsprechenden Wert entgegen nehmen und das Auslesen des Wertes durch das Werkzeug erlauben. Innerhalb des Widgets muß dann durch die konkrete Implementierung festgelegt werden, ob und wie dieser Wert auf ein oder mehrere Widgets aufgeteilt werden soll. Die Schnittstelle eines solchen Widgets ist immer auf einen Typ zugeschnitten, so daß wir immer auch die notwendige Typsicherheit gewährleisten können. Sofern über solch ein Widget auch die Bearbeitung / Eingabe von Fachwerten durchgeführt wird, ist eine Besonderheit im Bezug auf Fachwerte zu beachten. Die Erzeugung eines Fachwertes ist immer die Aufgabe der sogenannten Fachwertfabrik, die zudem auch die Gültigkeit dieser Werte bestimmt. Wird also von einem Benutzer ein Wert eingegeben, so ist diese Fabrik damit zu beauftragen, zu prüfen, ob diese Eingabe auch ein gültiger Fachwert ist. In diesem Zusammenhang ist zu klären, wie dieser Ablauf konstruktiv zu lösen ist und welcher Werkzeugteil für diese Anfrage verantwortlich ist.

Bei den von uns beobachteten Möglichkeiten der Kombination von Widgets gehen wir davon aus, daß die Kombination zur Laufzeit die größeren Anforderungen bezogen auf ein Konstruktionsprinzip stellt. Die von uns in dieser Arbeit im Kapitel 5 beschriebene Tabellenkonstruktion fällt bereits in diese Kategorie, so daß wir an dieser Stelle auf technische Details möglicher Implementierungen verzichtet haben. Auch wenn die Erstellung von Widgets für Fachwerte einen wichtigen Aspekt der Spezialisierung von Fachwerten darstellen, werden wir im weiteren Verlauf dieser Arbeit nicht mehr explizit auf diese Art der Kombination eingehen. Allerdings bietet die im Kapitel 5 aufgezeigte Konstruktion bereits einen Lösungsansatz, um auch eine Kombination von Widgets zur Designzeit zu realisieren.

Die Möglichkeit eines Anwendungsentwicklers die Zahl der in einem Rahmenwerk enthaltenen Widgets durch Kombination zu erweitern, beeinflußt die Übersichtlichkeit der Anordnung innerhalb des Rahmenwerkes. Aus diesem Grunde möchten wir in dem folgenden Abschnitt eine Möglichkeit vorstellen, wie wir bestimmte Typen von Widgets identifizieren können.

3.3 Eine mögliche Klassifikation von Widgets

Wie wir bereits erwähnt haben, ist es unserer Meinung nach sinnvoll, einem Anwendungsentwickler die Kombination von in einem Rahmenwerk enthaltenen Widgets zugänglich zu machen. Alleine durch die Möglichkeit, das hierdurch einem Rahmenwerk neue Widgets hinzugefügt werden können, ist es aber auch sinnvoll, eine Klassifikation von Widgets einzuführen, welche die Übersicht innerhalb der Struktur des Rahmenwerkes bewahrt. Diese Klassifikation soll es ermöglichen, Rückschlüsse darauf zu ziehen, wie dieses Widget von einem Werkzeug verwendet werden kann, da sich die Elemente einer Gruppe den selben Konstruktionsprinzipien bedienen. Dieser Abschnitt befaßt sich noch nicht mit den weitergehenden Ansätze für die Integration von Widgets in Rahmenwerken, die bereits im Rahmen des WAM Leitbildes entwickelt worden sind. Die Beschreibung und Erläuterung dieser Konzepte ist Bestandteil des vierten Kapitels.

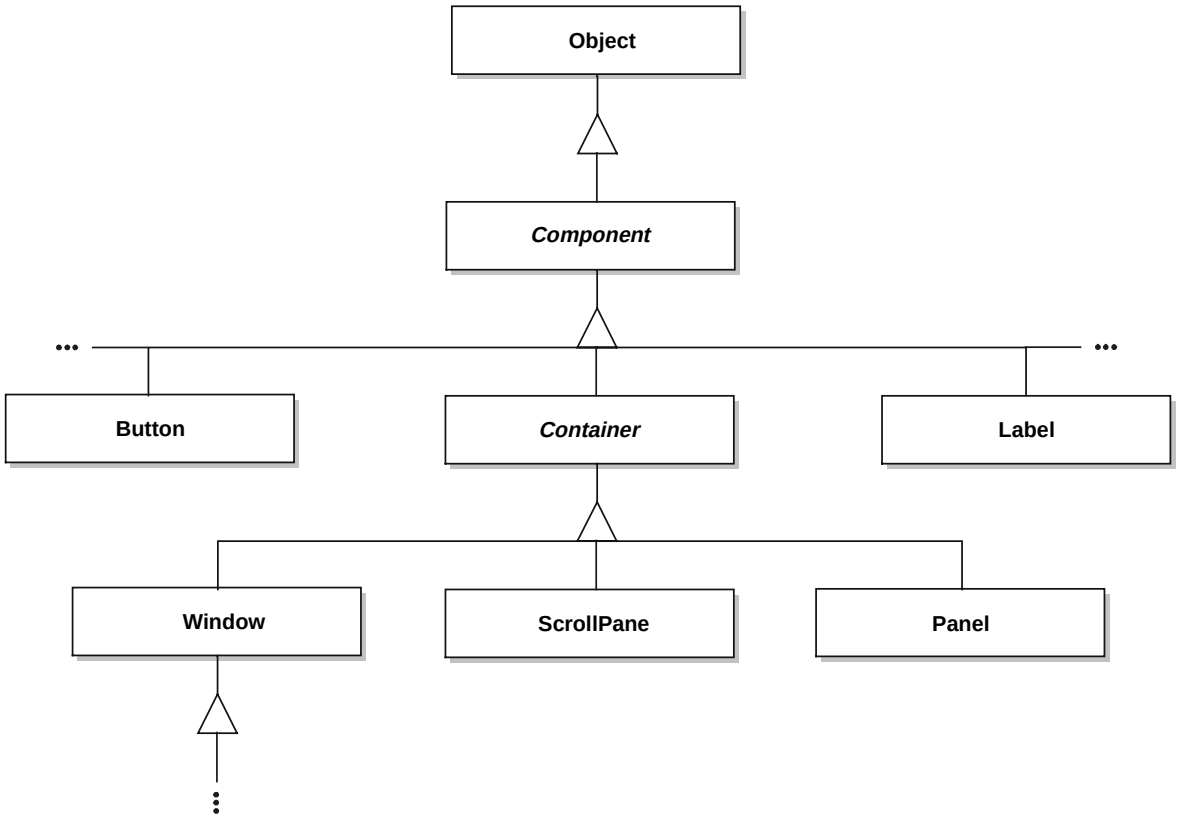

Abbildung 3-6: Ausschnitt aus der AWT Hierarchie

Eine erste Klassifikation von Widgets finden wir bereits in den GUI Toolkits selbst. In diesen Toolkits wird eine Zuordnung aufgrund der technischen Abhängigkeiten getroffen, die sich in der Klassenhierarchie ausdrücken. Die Abbildung 3-6 zeigt einen Ausschnitt aus der Klassenhierarchie des AWT des JDK 1.1 gemäß [Flan98]. Innerhalb des JDK's wird bereits zwischen rein der Darstellung dienenden Widgets und Widgets, die eine Containerfunktionalität aufweisen, unterschieden. Das klassische Beispiel für solch einen Container ist das Fenster. Auf der technischen Ebene werden die einzelnen Widgets in diesem Container zusammengefaßt und hierarchisch in einem Baum angeordnet. Der Container übernimmt in diesem Fall die Verwaltung der Widgets und in Teilbereichen die Bearbeitung von Benutzereingaben.

Ein anderes Beispiel für einen Container stellen die sogenannten Panels dar. Ein Panel kann als eine einfache Darstellungsfläche angesehen werden, auf der eine gewisse Anzahl von grafischen Elementen dargestellt werden kann. Im Inneren eines Panels werden die einzelnen Bestandteile in Analogie zu einem Fenster in einem Baum angeordnet. Ein Panel übernimmt aber im Gegensatz zu einem Fenster keine Bearbeitung von Benutzeraktionen. In diesem Fall sind die enthaltenen Widgets alleine für die vollständige Bearbeitung der interessierenden Benutzeraktionen verantwortlich. Allerdings können Panels niemals für sich alleine angezeigt werden, sondern sind als ein Bestandteil eines Fensters in die dortige Hierarchie eingeordnet.

Solange die Benutzung von Widgets durch einen Anwendungsentwickler innerhalb eines Rahmenwerkes nur auf die über ein Toolkit mitgelieferten Widgets beschränkt, ist diese Einordnung völlig ausreichend. Sofern wir aber einem Anwendungsentwickler zusätzlich die Möglichkeit der Kombination von Widgets zur Verfügung stellen, müssen wir in der Lage sein, die verschiedenen Typen zu unterscheiden. Hierbei ist zusätzlich zu beachten, daß für die Einbettung von Widgets in Rahmenwerken nach dem WAM Leitbild der

intendierte Umgang[12] mit einem Widget von wesentlichem Interesse ist. Für unser Verständnis können wir also Widgets klassifizieren, indem wir jedem Widget eine Anzahl von möglichen Umgangsformen zuordnen. Eine Beschreibung dieser Art Widgets zu klassifizieren finden wir bereits in [Blee99]. In dieser Arbeit wird allerdings der Ursprung eines Widgets nicht berücksichtigt. Das heißt die Autoren beschreiben in dieser Arbeit die Anbindung eines beliebigen Widgets über ein für alle Widgets zu verwendendes Entwurfsmuster. Sowie wir die verschiedenen Möglichkeiten der Kombination von Widgets berücksichtigen, ergeben sich auf der konstruktiven Seite verschiedene Varianten des verwendeten Musters. In diesen Varianten wird das verwendete Basismuster um Bestandteile erweitert, die vornehmlich in einer unterschiedlichen Benutzung durch einen Anwendungsentwickler resultieren. Um diesen Unterschieden gerecht werden zu können, schlagen wir vor, die Art von Widgets zusätzlich wie folgt zu differenzieren.

Widget Handhabung	statische Kombination	dynamische Kombination	kann elementare Widgets enthalten
elementar	-	-	-
zusammengesetzt	✓	-	✓
komplex[13]	✓	✓	✓

Tabelle 3-1: Handhabung und Kombination von Widgets

In der ersten Spalte sind die Bezeichnungen aufgeführt, anhand derer wir auf die technische Benutzung eines Widgets schließen wollen. Die folgenden beiden Spalten geben an, welche Art der Kombination einem Widget zugrunde liegt.

Unter elementaren Widgets verstehen wir diejenigen Elemente, die im Rahmen der Einbettung eines Toolkits in ein Rahmenwerk verfügbar werden. Widgets, die in diese Klasse einzuordnen sind zeichnen sich zusätzlich dadurch aus, daß sie eine fest definierte Darstellung für die Basistypen anbieten. Ein Widget dieser Klasse ist in der Lage, eine Präsentation für genau einen Typ zu realisieren. Die Zuordnung dieses Typs zu einem Widget erfolgt bereits zum Zeitpunkt der Konstruktion und ist somit als statisch anzusehen. Eine Ausnahme in dieser Hinsicht stellen Buttons dar, deren Aufgabe einzig darin besteht, eine Aktivierung durch einen Benutzer zu melden. Dennoch sind Buttons ebenfalls dieser Klasse zuzuordnen. Für die Anbindung von Elementen dieser Klasse an ein Werkzeug bedient sich ein Anwendungsentwickler immer des für diesen Zweck vorgesehenen Basismusters.

Ein Ausschnitt aus den im JWAM Rahmenwerk vorhandenen Widgets soll hier als Beispiel dienen.

---

[12] Im Rahmen der Beschreibung der Anbindung von Oberflächen nach dem WAM Ansatz, werden wir den Begriff des Umganges über die sogenannten Interaktionsformen ausführlich erläutern.

[13] Bedeutung (Auszug aus dem Duden): a) vielschichtig, b) zusammenhängend, c) [vieles] umfassend. In unserem Kontext verbinden wir diesen Ausdruck mit einer vielschichtigen Handhabung.

Widgetname	Ursprung	Umgang
pfJButton	SWING	Activator
pfJList	SWING	1FromNSelection Activator
pfJIntegerField	SWING	FillInNumber
pfJTextField	SWING	FillInText

Tabelle 3-2: Beispiele für Widgets mit elementarer Handhabung

Die Klasse der zusammengesetzten Widgets ist aus der Anforderung heraus entstanden, daß für einen Anwendungskontext spezifische Werte zur Darstellung gebracht werden sollen. In dem vorangegangenen Abschnitt haben wir in diesem Zusammenhang das Beispiel einer Bankleitzahl angeführt. Naturgemäß werden diese Widgets benutzt, um jeweils einen bestimmten Fachwert darzustellen. Auf dieser Ebene unterscheiden sich die Elemente dieser Klasse nicht von den von uns als elementar bezeichneten Widgets. Insbesondere betrifft dies auch die Eigenschaft dieser Widgets, das der Typ der darzustellenden Werte bereits zur Designzeit durch einen Anwendungsentwickler festgelegt werden muß. Der Unterschied liegt für uns vielmehr darin, daß ein oder mehrere elementare Widgets hier zu einem neuen Widget zusammengefaßt werden. Fachwerte können wiederum aufgeteilt in einfache und zusammengesetzte Fachwerte, wobei die zusammengesetzten Fachwerte aus einer Vielzahl an Fachwerten bestehen können. Ein beliebtes Beispiel ist in diesem Zusammenhang die Modellierung eines Kunden. Der Kunde einer Firma hat zum Beispiel eine Kundennummer, die einer durch die Firma definierten Syntax unterliegt. Diese Kundennummer ist also für sich schon ein zu berücksichtigender Fachwert des Anwendungsbereiches. Um solch einen zusammengesetzten Fachwert adäquat darstellen zu können, mu0 bereits für jeden in ihm enthaltenen Fachwert eine geeignete Präsentation existieren. Zusammenfassend kann man somit sagen, daß in einem Widget dieser Klasse neben elementaren Widgets auch Präsentationsformen für Fachwerte enthalten sein können. Wobei an dieser Stelle generell anzumerken bleibt, daß auch ein entsprechender Umgang für ein solches Widget definiert werden muß. Bei der Anbindung von Elementen dieser Klasse an ein Werkzeug findet ebenfalls das Basismuster Anwendung. In dieser Klasse finden wir also genau diejenigen Widgets, welche die Fachwerte einer Domäne darstellen. Für den Bankenkontext sind zum Beispiel die folgenden Widgets denkbar.

Widgetname	enthält	Umgang
pfRoutingCode	pfJTextField	FillInRoutingCode
pfCustomer	pfJTextField pfJNumber	FillInCustomer

Tabelle 3-3: Beispiele für Widgets mit zusammengesetzter Handhabung

Die beiden bisher beschriebenen Klassen enthalten ausschließlich Widgets, bei denen immer bekannt ist, von welchem Typ der darzustellende Wert ist. Um nun diejenigen Widgets einordnen zu können, bei denen der Typ der darzustellenden Werte von dem jeweiligen Anwendungskontext abhängt, haben wir die Klasse der komplexen Widgets

eingeführt. Sowie der Typ der darzustellenden Werte erst zum Zeitpunkt der Verwendung bestimmt werden kann, bedeutet dies, daß wir in diesem Fall von einem dynamischen Typ des Widgets ausgehen müssen. Komplex bezieht sich in diesem Fall auf die Eigenschaft, daß die für die Darstellung von Werten zu verwendenden Widgets durch einen Anwendungsentwickler zur Laufzeit bestimmt werden können. Die in dieser Klasse enthaltenen Elemente können dabei durchaus Bestandteil von in ein Rahmenwerk eingebetteten Toolkits sein. So gehören zum Beispiel Tabellen und Trees mittlerweile zum festen Repertoire eines GUI Toolkits. Damit haben wir auch gleichzeitig die beiden bekanntesten Vertreter dieser Klasse benannt. Um eine fachlich motivierte und trotzdem flexible Handhabung derartiger Widgets realisieren zu können, ist unserer Meinung nach eine andere Art der Handhabung durch einen Anwendungsentwickler nötig, die diese Klasse deutlich von den beiden anderen abhebt. Eine mögliche Benutzung dieser Widgets haben wir im Rahmen eines Vorschlages zur Konstruktion von Tabellen im fünften Kapitel beschrieben.

Widgetname	enthält	Umgang
`pfTable`	elementare und/oder zusammenges. Widgets	FillInTable
`pfTree`	elementare und/oder zusammenges. Widgets	FillInTree

Tabelle 3-4: Beispiele für Widgets mit komplexer Handhabung

Zusammenfassend ist zu sagen, daß sich dieses Konzept nicht nur als Richtlinie für die technische Implementierung eines Rahmenwerkes eignet, sondern als Bestandteil der Dokumentation eines Rahmenwerkes auch dem Anwendungsentwickler Aufschluß über die beabsichtigte Verwendung von Widgets gibt.

4 Oberflächen aus der Sicht des WAM-Leitbildes

Nachdem wir einen Überblick über den allgemeinen Einsatz von Widgets in grafischen Benutzeroberflächen geschaffen haben, möchten wir nun auf unseren spezifischen softwaretechnischen Hintergrund überleiten. Gemäß unserer Aufgabenstellung wollen wir dieses Thema aus der Sicht von Rahmenwerken, die nach dem WAM Ansatz konstruiert wurden, betrachten. Deshalb werden wir in diesem Kapitel Zielsetzungen und Möglichkeiten der Anbindung von GUI - Oberflächen an Softwarewerkzeuge diskutieren.

4.1 Ziele der Anbindung von grafischen Oberflächen nach WAM

Anwendungsorientierte Softwareprogramme im WAM - Kontext sind häufig durch das Leitbild eines *Arbeitsplatzes für qualifizierte und eigenverantwortliche Tätigkeiten* geprägt. Den Anwendern solcher Software wird neben ihrem Fachwissen auch eine gewisse Eigenständigkeit und Eigeninitiative bei der Erledigungen ihrer Arbeit am Rechner zugestanden. Dieses eigenverantwortliche Handeln stellt besondere Anforderungen an die Konstruktion der Software. Die Interaktion des Anwenders mit dem System erfolgt reaktiv, das heißt der Anwender führt die Arbeitsschritte in einer ihm angemessen erscheinenden Reihenfolge aus. Die Software gibt im Idealfall keine explizite Reihenfolge vor und reagiert nur auf die Eingaben des Anwenders.

Anwendungen nach dem WAM Leitbild werden unter anderem nach den Entwurfsmetaphern Werkzeug, Material, Automat und Arbeitsumgebung entworfen. Mit Hilfe der Softwarewerkzeuge erledigt der Anwender seine Tätigkeiten und Arbeitsabläufe. Diese Werkzeuge erlauben ihm dabei einen interaktiven Umgang mit den Materialien der Arbeitsumgebung. Im folgenden wollen wir die Konstruktion eines Softwarewerkzeuges kurz vorstellen, um die verschiedenen Möglichkeiten einer Anbindung eines GUI Toolkits besser beurteilen zu können. Eine umfassende Einführung in die Konstruktion von Anwendungssoftware nach dem WAM Ansatz findet sich in [Züll98].

Das Entwurfsmuster für ein Werkzeug besteht aus einer Aufteilung in einen interaktiven und einen funktionalen Teil. Diese beiden Teile sollten softwaretechnisch weitgehend unabhängig voneinander sein, damit sie getrennt voneinander entwickelt und gewartet werden können. Insbesondere sollen Änderungen am interaktiven Teil des Werkzeuges möglichst keine Anpassungen am funktionalen Teil erforderlich machen.

Der als Funktionskomponente bezeichnete funktionale Teil eines Werkzeuges realisiert die fachliche Funktionalität des Werkzeuges. Mit seiner Hilfe können die Aspekte der Materialien sondiert und bearbeitet werden. Die Funktionskomponente verwaltet dabei den Bearbeitungszustand der Materialien in einem Werkzeuggedächtnis. Dieses Werkzeuggedächtnis verändert sich in Abhängigkeit von dem Materialzustand und den Aktivitäten des Benutzers. Die Funktionskomponente macht dabei keine Annahmen über die Handhabung und Präsentation des Werkzeuges auf einer GUI Oberfläche. Sie soll möglichst vollständig von der grafischen Benutzungsschnittstelle entkoppelt sein.

Auf der anderen Seite realisiert der interaktive Teil, der auch als Interaktionskomponente bekannt ist, die Handhabung und Präsentation des Werkzeuges auf der grafischen Benutzungsschnittstelle. Die Interaktionskomponente nimmt dabei alle Systemereignisse,

die durch Interaktionen des Benutzers mit der Benutzungsschnittstelle ausgelöst werden, entgegen und interpretiert sie. Ausschließlich auf die Präsentation bezogene Ereignisse bearbeitet sie selber, fachlich relevante Ereignisse werden zur weiteren Bearbeitung an die Funktionskomponente weitergeleitet. Um die Unabhängigkeit der Interaktionskomponente von der Funktionskomponente zu gewährleisten, macht die Interaktionskomponente keinerlei Annahmen über die Auswirkungen der fachlichen Ereignisse, die sie weitergeleitet hat. Dies bedeutet, daß die Interaktionskomponente die Darstellung eines Materials nicht von sich aus ändert, sondern immer zuerst sondierende Funktionen der Funktionskomponente konsultiert.

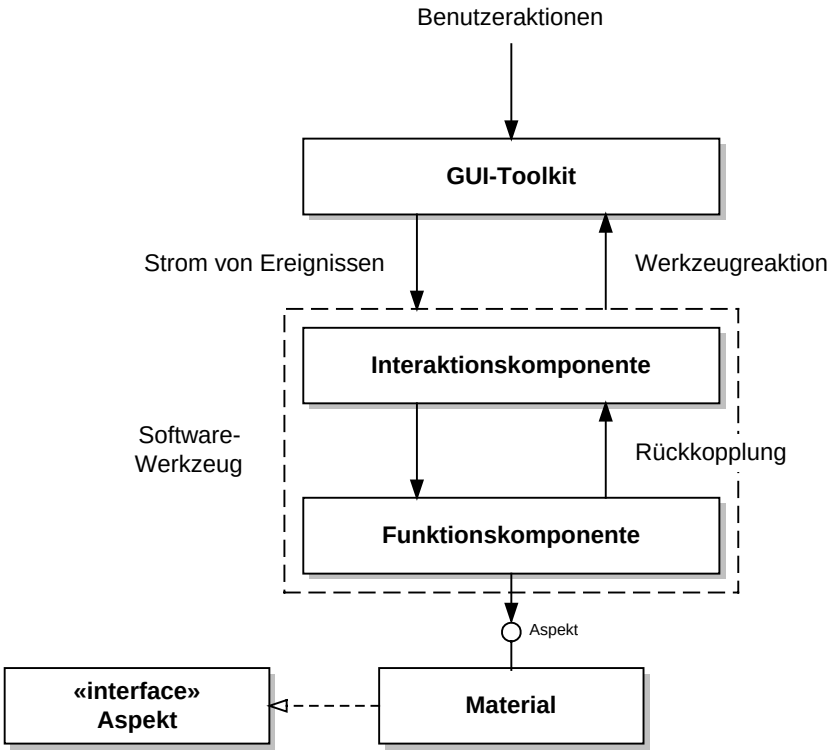

Abbildung 4-1: Werkzeugkonstruktion nach WAM

Die Interaktionskomponente und die Funktionskomponente dürfen aufgrund der geforderten Unabhängigkeit nur lose miteinander gekoppelt sein. Die Interaktionskomponente kennt die volle Schnittstelle der Funktionskomponente. Die Funktionskomponente hingegen kennt die Schnittstelle der Interaktionskomponente nicht. Für sie ist die Interaktionskomponente nur ein Beobachter, der über das Beobachtermuster aus [GHJV96] angebunden ist. Diese Art der losen Koppelung, bei der eine Komponente eine andere unter ihrer vollen Schnittstelle kennt und in der umgekehrten Richtung nur eine lose Beobachterbeziehung besteht, nennen Rook und Wolf in [RooWol97] ein Reaktionsmuster. Die Abbildung 4-1 gibt schematisch den beschriebenen Aufbau eines Softwarewerkzeuges nach dem WAM Ansatz wieder.

Innerhalb der Interaktionskomponente wird die eigentliche Anbindung des Werkzeuges an die GUI Oberfläche realisiert, wobei die Anbindung kann dabei auf verschiedene Art und Weise erfolgen kann. Um die verschiedenen Lösungen besser bewerten zu können, möchten wir kurz Anforderungen aufzeigen, die an die Anbindung eines GUI gestellt werden sollten.

Die Anbindung der Interaktionskomponente an die konkreten Widgets eines Betriebssystems erfolgt sinnvollerweise über ein GUI Toolkit[14]. Die Verwendung eines GUI Toolkits sollte die Konstruktion der Interaktionskomponente möglichst nicht beeinflussen. Wie wir bereits in Kapitel 3.1 dargelegt haben, sollte durch das GUI Toolkit insbesondere nicht der Kontrollfluß für die Interaktionskomponente vorgegeben werden. Der Kontrollfluß wird jedoch häufig durch den Benachrichtigungsmechanismus für Systemereignisse, den das GUI Toolkit vorgibt, stark beeinflußt. Der vorgegebene Benachrichtigungsmechanismus sollte deshalb möglichst wenig Einfluß auf die Konstruktion der Interaktionskomponente haben. Eine Anbindung zum Beispiel über Callback Operationen ist keine gute Lösung, da die gerufenen Methoden der Interaktionskomponente besondere Signaturen haben müssen. Generell sollte die Interaktionskomponente unabhängig von den Eigenschaften und der Schnittstelle des GUI Toolkits sein, denn dann haben Änderungen an der Schnittstelle des GUI Toolkits keine Auswirkungen auf die Interaktionskomponente. Nur auf diese Weise ist ein problemloser Austausch des GUI Toolkits oder die Verwendung von mehreren Toolkits gewährleistet.

Die Widgets sollten aus Sicht der Interaktionskomponente kontextfrei und nebeneffektfrei betrieben werden. Zu diesem Zweck sollten Sie mit den Interaktionskomponenten ausschließlich Informationen auf Basis von Wertsemantik austauschen. Neben den Basistypen der verwendeten Programmiersprache kommen dafür insbesondere die Fachwerte einer Domäne in Frage.

Um ein zügiges Arbeiten und ein sofortiges semantisches Feedback der Anwendung auf Aktionen des Benutzers zu gewährleisten, ist auch auf eine effiziente Implementierung der Anbindung der Widgets an die Interaktionskomponente zu achten. Besonders Anwendungen, die den WAM Ansatz berücksichtigen und in Java programmiert wurden, neigen aufgrund der besonderen Gegebenheiten der Programmiersprache Java und den vielen Abstraktionskonzepten in WAM dazu, einen etwas langsamen und behäbigen Bildschirmaufbau zu haben. Wie groß die Auswirkungen verschiedener Arten von GUI Anbindung in Java sein können, zeigen die Untersuchungen in [Half98]. Der Artikel dokumentiert unter anderem Unterschiede von bis zu 600 Prozent beim Öffnen eines Dialogfensters und bis zu 1200 Prozent beim Füllen eines Listenfeldes.

Der Aufbau und die Entwicklung von guten Benutzungsschnittstellen ist meist ein langwieriger Prozeß. In vielen Sitzungen und Interviews mit den zukünftigen Anwendern ergeben sich oft größere Änderungen in der Handhabung eines Softwarewerkzeuges. Erst nach mehreren Rückkoppelungszyklen kristallisiert sich die endgültige GUI Oberfläche heraus. Die Verwendung von GUI Buildern kann diesen Prozeß erheblich vereinfachen und verkürzen. Bei der Auswahl eines GUI Toolkits ist deshalb darauf zu achten, daß ein GUI Builder für dieses Toolkit existiert und daß die mit dem GUI Builder erstellten und abgespeicherten Oberflächen auch einfach an die Werkzeuge angebunden werden können.

Für einen flexiblen und ergonomisch guten Oberflächenentwurf sollten alle Möglichkeiten, die ein GUI - Tookit bietet, ausgenutzt werden können. Auch sollten die Styleguides der Fenstersysteme eingehalten werden können, damit sich die Softwarewerkzeuge gut in das gesamte System integrieren und die Oberfläche dem Benutzer vertraut erscheint.

[14] Siehe auch Kapitel 2.1.

Zusammenfassend müssen folgende Punkte bei einer GUI Anbindung beachtet werden:

- Möglichst keine Abhängigkeiten bei der Konstruktion zwischen der Interaktionskomponente und dem verwendeten GUI Toolkit.
- Kontextfreie und nebeneffektfreie Verwendung von Widgets.
- Austausch von Informationen mit dem GUI Toolkit auf Basis von Wertsemantik.
- Effiziente Anbindung des GUI Toolkits für eine schnelle Reaktionen der GUI Oberfläche auf Aktionen des Benutzers.
- Die Einbindung von GUI Buildern muß möglich sein.
- Alle Möglichkeiten des GUI Toolkits sollten nutzbar sein.
- Die Styleguides des verwendeten Fenstersystems sollten eingehalten werden.

Die in dieser Zielsetzung genannten Aspekte stehen teilweise im Widerspruch zueinander. Im Rahmen der Realisierung der Anbindung eines Fenstersystems an Werkzeuge muß dieses Spannungsfeld berücksichtigt werden. So steht zum Beispiel die Ausnutzung aller Möglichkeiten eines GUI Toolkits dem Ziel entgegen, möglichst wenig Abhängigkeiten zu dem konkret verwendeten Fenstersystem einzugehen. Desweiteren bleibt zu erwähnen, daß die Einhaltung der verschiedenen Styleguides der Fenstersysteme immer die Entscheidung der Anwendungsentwickler ist. Von daher ist der letztgenannte Punkt eher in dem Sinne einer Option zu verstehen.

Es wurden im Rahmen des WAM Leitbildes bisher drei verschiedene Arten der Anbindung von grafischen Benutzerschnittstellen über die Interaktionskomponente realisiert. Neben der direkten Verwendung der Widgets eines GUI Toolkits wurde versucht, über zusätzliche Abstraktionsebenen die Abhängigkeiten von einem konkreten GUI Toolkit zu minimieren. In den folgenden zwei Kapiteln werden wir diese beiden Ansätze näher vorstellen. Die direkte Verwendung von Widgets werden wir an dieser Stelle nicht weiter diskutieren, da sie nur wenige der oben genannten Forderungen erfüllt. In Kapitel 6 werden wir diesen Ansatz allerdings im Rahmen der Manipulation von Widgeteigenschaften noch einmal aufgreifen.

4.2 Interaktionstypen

Die direkte Verwendung eines GUI Toolkits durch die Interaktionskomponente eines Werkzeuges bietet viele Vorteile, wobei allerdings die Nachteile überwiegen. Ein schwerwiegender Nachteil besteht darin, daß die Interaktionskomponente direkt von Veränderungen am GUI Toolkit abhängig ist. Da sich GUI Toolkits den immer komplexer und aufwendiger werdenden Oberflächen der Fenstersysteme anpassen, ergeben sich des öfteren Erweiterungen und Anpassungen an der Schnittstelle eines Toolkits. Diese Änderungen können unter Umständen zu einem umfangreichen Anpassungsbedarf in allen Interaktionskomponenten einer Anwendung führen. GUI Toolkits machen auch immer Annahmen darüber, wie der Kontrollfluß in der Anwendung aussehen soll und wie die Benachrichtigung der Anwendung über Systemereignisse erfolgen soll. Diese Annahmen können im Widerspruch zu den Vorstellungen des Anwendungsentwickler stehen und so die Anwendungsentwicklung komplizierter und / oder aufwendiger gestalten. Um diese Nachteile zu vermeiden und die Unabhängigkeit der Interaktionskomponente von verschiedenen GUI Toolkits gewährleisten zu können, wurden Interaktionstypen zur Kapselung der Handhabung der verschiedenen Widgets eines GUI - Toolkits eingeführt.

Nach [Züll98, Seite 266] werden Interaktionstypen wie folgt definiert:

> ***Interaktionstyp (IAT):***
>
> *Ein Interaktionstyp ist eine abstrakte Form der Handhabung eines Werkzeugs. Er kapselt die konkrete Präsentation und den Interaktionsmechanismus, den die grafischen Bausteine eines User Interface Toolkit anbieten. Dabei wandelt ein IAT Systemereignisse in Programmereignisse um.*
>
> *Ein IAT präsentiert und liefert nur Fachwerte. Er hat also keine globalen Effekte.*
>
> *Ein IAT wird von einer Interaktionskomponente verwendet und macht sie damit unabhängig vom gewählten Toolkit.*

An der Definition eines Interaktionstypen fällt sofort auf, daß die Abstraktion vom verwendeten GUI Toolkit nicht auf einer technischen Ebene geschieht, sondern auf einer Ebene, welche die fachliche Handhabung und Präsentation eines Widgets betrachtet. Gängige Toolkits verwenden zum Bau ihrer Widgets oft die Implementationsvererbung. Bei der StarView Klassenbibliothek (siehe [Star93]) erbt ein CheckBox Widget zum Beispiel von der Oberklasse Window, weil Teile eines CheckBox Widgets ähnlich wie ein Fenster implementiert sind. Der Vererbungsbaum ist rein technisch motiviert und ohne Kenntnisse über die interne Implementation nicht einsichtig. Interaktionstypen hingegen sind am fachlichen Umgang orientiert. Sie kapseln den Umgang und die Präsentation eines Widgets. Die Abbildung 4-2 zeigt dies an einem Ausschnitt aus dem Vererbungsbaum der Interaktionstypen einer C++ Referenzimplementierung.

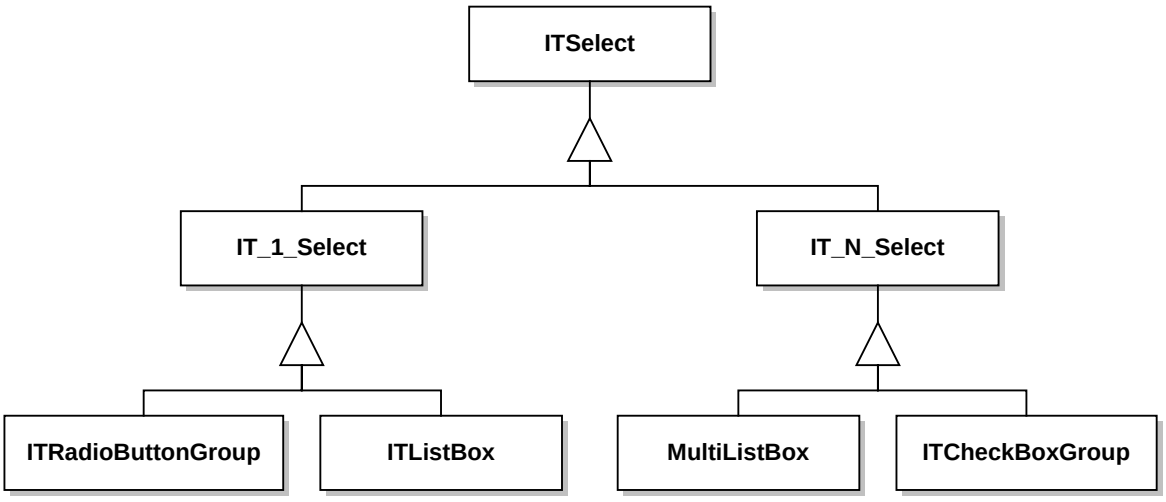

Abbildung 4-2: Ausschnitt aus dem Vererbungsbaum der Interaktionstypen

Interaktionstypen machen keine Annahmen über die Interaktionskomponente und tauschen mit dieser nur Fachwerte oder Basistypen aus. So können sie in jeder Anwendung kontextunabhängig verwendet werden. Dies betrifft besonders den Benachrichtigungsmechanismus für Systemereignisse. Die Interaktionstypen werden nur lose über das Befehlsmuster (siehe auch [GHJV96]) an die Interaktionskomponente gekoppelt. Die Interaktionstypen verschicken an die Interaktionskomponenten zur Benachrichtigung über die verschiedenen Systemereignisse Befehlsobjekte, wobei für jedes Ereignis eine eigene Befehlsklasse existiert. Die Befehlsobjekte können dabei auch nähere Informa-

tionen über das ausgelöste Systemereignis enthalten. Auf diese Weise erfolgt eine effiziente Benachrichtigung der Interaktionskomponente über Aktionen auf der grafischen Oberfläche.

Die Verwendung der Interaktionstypen durch die Interaktionskomponente erfolgt unter der Kenntnis der gesamten Schnittstelle. Die Interaktionskomponente erzeugt zunächst die benötigten Interaktionstypen. Mit Hilfe der Angaben aus einer Ressourcen Datei, die unter anderem auch mit einem GUI Builder erzeugt werden kann, stellen die Interaktionstypen ihre optischen Eigenschaften und sonstigen Einstellungen ein. Zu diesem Zweck bekommen die Interaktionstypen von der Interaktionskomponente bei ihrer Erzeugung jeweils einen eindeutigen Ressourcen Identifikator zugewiesen. Durch diesen Identifikator können die Interaktionstypen ihre Beschreibung innerhalb der Ressourcen Datei wiederfinden. Nach der Initialisierung der Interaktionstypen erzeugt sich die Interaktionskomponente für alle Systemereignisse, über die sie informiert werden möchte, passende Befehlsobjekte. Diesen Befehlsobjekten übergibt sie die nötigen Parameter und Verweise für den Aufruf von Operationen an ihr selbst. Diese Operationen sollen dann nach der Auslösung des entsprechenden Systemereignis gerufen werden. Diese Befehlsobjekte werden dann an den entsprechenden Interaktionstypen registriert. Ein Interaktionstyp aktiviert beim Eintritt eines passenden Systemereignisses das bei ihm registrierte Befehlsobjekt und versorgt es mit näheren Angaben zum Ereignis. Das Befehlsobjekt reagiert darauf mit dem Aufruf der entsprechenden Operation an der Interaktionskomponente. Die Interaktionskomponente hat nun die Möglichkeit, in der gerufenen Operation die Angaben zum Ereignis am Befehlsobjekt zu sondieren und auf das Ereignis zu reagieren.

Die Koppelung über das Befehlsmuster ist der Schlüssel zur Unabhängigkeit der Interaktionskomponente von den Interaktionstypen und dem Kontrollfluß im GUI Toolkit. Die Interaktionstypen aktivieren ausschließlich Befehlsobjekte. Die Interaktionskomponente hat somit die volle Kontrolle darüber, wie mit Systemereignissen umgegangen wird, da die Befehlsobjekte von der Interaktionskomponente selbst erzeugt und registriert werden.

Mit Hilfe der Interaktionstypen ist es relativ gut gelungen, fast alle Forderungen, welche wir in Kapitel 4.1 an eine GUI Anbindung gestellt haben, zu erfüllen. Dennoch können sich bei der Umsetzung und Anbindung eines konkreten GUI Toolkits über Interaktionstypen Probleme ergeben. Jeder Interaktionstyp kapselt in der Regel genau ein Widget, so daß Interaktionstypen nur die Schnittmenge aller gängigen, in den GUI Toolkits enthaltenen, Widgets abdecken können. Es ist so kaum möglich, die volle Funktionalität eines GUI Toolkits auszunutzen. Auch lassen sich so die verschiedenen Styleguides der unterschiedlichen Fenstersysteme kaum einhalten. Auf der technischen Seite ergeben sich bei der Implementierung der Interaktionstypen ebenfalls Probleme, da die GUI Toolkits vielerlei Vorgaben über den Kontrollfluß und ihre Benutzung machen. Diese Vorgaben müssen in den Interaktionstypen gekapselt und vor der Interaktionskomponente verborgen werden. Dies kann unter Umständen nur durch Verwendung von aufwendigen Adapter- und Brückenmustern erfolgen.

4.3 Interaktionsformen

Wie wir bereits anhand der Interaktionstypen gezeigt haben, ist eine möglichst umfassende Abstraktion vom verwendeten GUI Toolkit von Vorteil. Da Interaktionstypen die Handhabung und gleichzeitig auch die Präsentation von GUI Elementen in einem Objekt kapseln, repräsentiert ein Interaktionstyp in der Regel genau ein Widget des GUI Toolkits. Dies schränkt die Flexibilität beim Oberflächendesign für ein Werkzeug ein. Denn es steht

nur eine Obermenge von Widgets, die in den meisten GUI Toolkits vorkommen, zur Verfügung. Auch führen Veränderungen an der Werkzeugoberfläche, die häufig im Rahmen des Autor - Kritiker - Zyklus vorgenommen werden, nicht selten zu Änderungen in der Auswahl der in der Interaktionskomponente verwendeten Interaktionstypen. Dies bedeutet, daß oft nach einem Redesign der Werkzeugoberfläche auch die Interaktionskomponente geändert werden muß.

Um noch besser vom verwendeten GUI Toolkit und den in der Werkzeugoberfläche verwendeten Widgets zu abstrahieren, wird mit Hilfe von Interaktionsformen versucht, die Handhabung von der Präsentation zu trennen. Der Werkzeugentwickler soll beim Entwurf der Interaktionskomponente nur die Handhabung, im Sinne der Interaktion mit einem Benutzer, festlegen, ohne dabei auf die Präsentation des Werkzeuges auf dem Desktop achten zu müssen. Erst zu einem späteren Zeitpunkt wird eine passende Präsentation in Form von Widgets gewählt. Die Präsentation kann sogar im gewissen Umfang jederzeit geändert werden, ohne daß die Interaktionskomponente angepaßt werden muß. In [Blee99, Seite 11] wird eine Interaktionsform wie folgt definiert:

> ***Interaktionsform:***
>
> *Eine Interaktionsform definiert die abstrakte Handhabung eines Werkzeuges. Ein Werkzeug gebraucht Interaktionsformen, um die mögliche Interaktion mit dem Werkzeug zu beschreiben.*
>
> *Eine Interaktionsform repräsentiert eine fachliche Umgangsform mit einem Werkzeug und hat keine Seiteneffekte, die sich auf das Werkzeug auswirken.*
>
> *Interaktionsformen werden benutzt, um das Werkzeug mit einer konkreten Handhabung durch ein GUI Toolkit zu verbinden.*
>
> *Interaktionsformen machen ein Werkzeug unabhängig von der technischen Präsentation durch das Toolkit, die beliebig ausgetauscht werden kann.*

Der Werkzeugentwickler orientiert sich bei der Werkzeugkonstruktion jetzt nicht mehr an der Vorstellung der konkreten Werkzeugoberfläche, sondern er entwirft ausschließlich die Funktionalität und Handhabung des Werkzeuges. Dies geschieht durch die Auswahl und Verwendung passender Interaktionsformen. Die Werkzeugoberfläche mit den konkreten Widgets wird erst später zur Laufzeit der Anwendung dynamisch erzeugt oder geladen und an das Werkzeug gebunden.

Innerhalb des am Arbeitsbereiches entstandenen JWAM Rahmenwerkes hat sich der in der Tabelle 4-1 gezeigte Satz an Interaktionsformen bewährt. An diesem Satz von Interaktionsformen wird schon die starke Abstraktion von den konkreten Widgets der Toolkits deutlich. Die Anzahl der identifizierten Interaktionsformen ist um einiges kleiner als die Anzahl der zum Beispiel im Swing Toolkit enthaltenen Widgets.

Die Interaktionsform `ifTextDisplay` fällt dabei ein wenig aus dem Rahmen, denn sie dient nur zur Anzeige eines Textes durch das Werkzeug auf seiner Oberfläche. Streng genommen ist dies gemäß der Definition keine Interaktionsform, da der Benutzer des Werkzeuges den Text nicht bearbeiten kann und somit keine Interaktion mit dem Werkzeug stattfindet. Im Rahmen des sechsten Kapitels werden wir eine diesbezüglich eine Lösung aufzeigen, die mit Hilfe von zusammengesetzten Interaktionsformen solche der reinen Darstellung dienenden Widgets sauber integriert.

Die Widgets der Oberfläche werden nur noch unter dem Aspekt ihrer Handhabung betrachtet. Dies stellt eine höhere Abstraktion zwischen dem Werkzeug und dem verwendeten GUI Toolkit dar. Im Gegensatz zu den Interaktionstypen spielt die Präsentation für die Interaktionsformen keine Rolle mehr. Die Interaktionsformen definieren nur noch die Handhabung und kapseln somit keine konkreten Widgets mehr. Vielmehr stellen die Interaktionsformen „virtuelle Widgets" dar (siehe auch Kapitel 3.1). Da die konkrete Präsentation nicht mehr durch die Interaktionskomponente vorgegeben wird, ist sie unter Einschränkungen austauschbar. So können Widgets, welche die gleiche Handhabung realisieren, ohne Änderungen an der Interaktionskomponente untereinander ausgetauscht werden.

Interaktionsform	Funktionalität
`if1fromNSelection`	Einen Wert aus N Werten auswählen.
`ifActivator`	Eine Aktion auslösen.
`ifDrag`	Einen Objekt wegziehen.
`ifDrop`	Ein Objekt fallenlassen.
`ifFillIn`	Oberklasse aller Interaktionsformen, die eine Eingabe entgegen nehmen.
`ifFillInBoolean`	Eingabe eines boolschen Wertes.
`ifFillInFloatNumber`	Eingabe einer Gleitkommazahl.
`ifFillInNumber`	Eingabe einer ganzen Zahl.
`ifFillInString`	Eingabe eines Strings
`ifObject`	Oberklasse aller Interaktionsformen. Interaktion erlauben und verbieten.
`ifTextDisplay`	Einen Text darstellen ohne Interaktion mit dem Benutzer.

Tabelle 4-1: Interaktionsformen im JWAM Rahmenwerk

Wenn die Interaktionsformen keine Widgets mehr kapseln, stellt sich die Frage, wie die Anbindung an die in der Werkzeugoberfläche verwendeten Widgets erfolgen soll. Um die Möglichkeit zu haben, die Werkzeugoberfläche außerhalb des Werkzeuges zu erstellen und spätere Änderungen ohne Auswirkungen auf die Werkzeugkonstruktion vornehmen zu können, werden die eigentlichen Widgets durch Präsentationsformen gekapselt.

Präsentationsformen werden in [Blee99, Seite 12] wie folgt definiert:

> ***Präsentationsform:***
>
> *Eine Präsentationsform realisiert in der Benutzungsschnittstelle eines Werkzeuges die konkrete Präsentation und Handhabung einer fachlichen Umgangsform, die durch eine Interaktionsform vorgegeben wird.*
>
> *Präsentationsformen kapseln die verschiedenen Widgets eines GUI Toolkits und erfüllen dabei ein Protokoll, welches die Koppelung mit den Interaktionsformen erlaubt.*
>
> *Für jedes Widget existiert eine spezielle Präsentationsform, die vorgibt, mit welcher Interaktionsform dieses Widget verbunden werden kann.*
>
> *Präsentationsformen werden außerhalb eines Werkzeuges verwaltet und mit den Interaktionsformen innerhalb des Werkzeuges verbunden.*

Durch die Verwendung der Präsentationsformen erfolgt eine weitere Abstraktion von dem verwendeten GUI Toolkit. Bei einem Austausch des GUI Toolkits müssen lediglich diejenigen Klassen, welche die Präsentationsformen implementieren, geändert werden. Zusätzlich müssen die Oberflächen mit den neuen Widgets nochmals erzeugt werden. Auf die eigentliche Werkzeugkonstruktion hat der Austausch des GUI Toolkits nun keinen Einfluß mehr.

Da Präsentationsformen ähnlich wie Interaktionstypen die konkrete Handhabung und gleichzeitig die Präsentation eines Widgets kapseln, und Widgets verschiedene Arten von Handhabung ermöglichen, ist es nicht ausgeschlossen, daß ein Widget mehrere Präsentationsformen besitzt. Ein Beispiel für so ein Widget ist das Listenfeld. Es präsentiert sich zwar immer gleich, jedoch kann es unterschiedlich gehandhabt werden. Die Auswahl eines Wertes aus einer Menge von vorgebenden Werten geschieht durch einen einfachen Mausklick. Durch eine Doppelklick auf einen Wert kann jedoch eine Aktion mit diesem Wert ausgelöst werden. Ein Listenfeld hat somit zwei unterschiedliche Handhabungen und kann dementsprechend an zwei verschiedene Interaktionsformen gebunden werden.

Die Koppelung einer Interaktionsform mit einer Präsentationsform erfolgt nach einem Reaktionsmuster. Eine Interaktionsform kennt immer die zu ihr passende Präsentationsform und kann sie deshalb direkt unter ihrer vollen Schnittstelle ansprechen. Die Präsentationsform hingegen wird nur lose über das Befehlsmuster an die Interaktionsformen gebunden. Auf diese Weise ist der kontextfreie Einsatz der Präsentationsformen gegeben. Dies ist wichtig für Widgets, die mehrere Präsentationsformen besitzen. Denn erst zur Laufzeit des Werkzeuges wird entschieden, an welche konkrete Interaktionsform eine Präsentationsform gebunden wird. Die Abbildung 4-3 gibt eine mögliche Implementationsvariante der Einbettung einer Interaktionsform in ihren Kontext wieder.

Die Verwendung der Interaktionsformen erfolgt ähnlich wie die der Interaktionstypen. Die Interaktionskomponente erzeugt sich zunächst die passenden Interaktionsformen. Durch die Übergabe eines GUI - Kontext Objektes und eines Identifikators können sich die Interaktionsformen selbständig mit den zugehörigen Präsentationsformen verbinden. Eine Interaktionsform fragt bei ihrer Initialisierung den ihr bekannten GUI Kontext nach einer Referenz auf eine Präsentationsform mit dem übergebenen Identifikator. An der nun be-

kannten Präsentationsform meldet sich die Interaktionsform nach dem Befehlsmuster für Systemereignisse an.

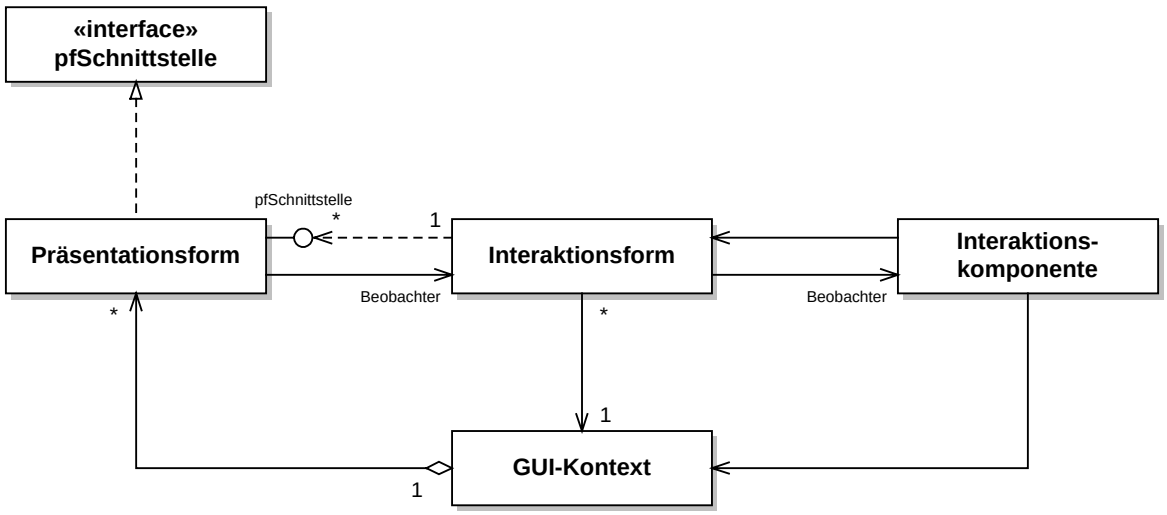

Abbildung 4-3: Klassendiagramm der Einbettung einer Interaktionsform

Voraussetzung für dieses Verfahren ist, daß der Entwickler bei der Erstellung der GUI Oberfläche allen Widgets einen eindeutigen Identifikator geben hat. Dadurch, daß die Präsentationsformen über ein separates GUI - Kontext Objekt verwaltet werden, kann die Anbindung der Oberfläche sehr flexibel gehandhabt werden. Je nach Implementierung des GUI Kontextes können vom GUI Builder gespeicherte Oberflächen eingelesen werden, Oberflächen selbst programmiert werden oder dynamisch zur Laufzeit anhand von Parametern erzeugt werden. Die Interaktionskomponente meldet anschließend wie bei den Interaktionstypen ihre vorher erzeugten und parametrisierten Befehlsobjekte bei den Interaktionsformen an. Der Austausch von Informationen erfolgt dabei ausschließlich nach dem Prinzip der Wertsemantik.

Durch die Einführung der Interaktions- und Präsentationsformen ist es sehr gut gelungen, Werkzeuge von dem verwendeten GUI Toolkit zu entkoppeln. Da ein Werkzeug nur noch die Handhabung beschreibt, ist die konkrete Präsentation für das Werkzeug unerheblich geworden. Dies ist ein großer Vorteil für die Portierung einer Anwendung auf andere Betriebs- oder Fenstersysteme. Die Anpassung erfolgt nur noch durch einen Austausch beziehungsweise durch eine einmalige Neuimplementierung der Präsentationsformen. So wäre es zum Beispiel bezüglich der Oberfläche denkbar, nur durch Verwendung geeigneter Präsentationsformen, welche die Funktionalität eines Proxy Objektes aufweisen, eine Stand-Alone - Anwendung in eine Web Anwendung für das Internet umzuwandeln. Die Styleguides eines Fenstersystems können so gut wie möglich eingehalten werden. Durch die Einführung eines GUI Kontextes ist der Entwickler auch sehr flexibel in der Art und Weise wie er seine Oberflächen erstellt und einbindet. Im JWAM Rahmenwerks sind die Interaktionsformen so flexibel implementiert, daß sie sich bei mehr als einer Präsentationsform anmelden können, wenn diese die gleiche Handhabung realisieren. So kann ein Ereignis wie zum Beispiel „Anwendung schließen" entweder durch einen Button oder einen Menüeintrag ausgelöst werden. Dieses beiden Präsentationsformen unterstützen die Handhabung „Aktivieren" und werden beide an die Interaktionsform `ifActivator` gebunden. Für das Werkzeug ist es unerheblich, durch welches Widget das Ereignis ausgelöst wird. Es ist nur an der Handlung des Benutzers selbst interessiert und nicht daran, durch wen die Handlung ausgelöst worden ist. Für

nähere Details zu der Implementierung von Interaktions- und Präsentationsformen in Java im JWAM Rahmenwerks empfehlen wir [Lipp97a].

Durch die hohe Abstraktion vom konkreten GUI Toolkit ergeben sich aber auch Probleme bei der Werkzeugkonstruktion. Die volle Funktionalität der Widgets kann nicht ausgenutzt werden, da die Interaktionsformen nur einen kleinen Satz von elementaren Handlungen abdecken und die Interaktionskomponente die verwendeten Widgets nicht mehr kennt. Über einen GUI Builder können zwar alle erforderlichen Einstellungen vorgenommen werden, jedoch können diese Einstellungen zur Laufzeit nicht mehr durch die Interaktionskomponente sondiert und verändert werden. Durch die zusätzlichen Abstraktionsschichten leidet auch die Performance der Oberfläche, da wesentlich mehr Objekte an der Oberflächendarstellung beteiligt sind als bei einer direkten Verwendung der Widgets. Durch die geschickte Verwendung des Rollenmusters läßt sich jedoch die Anzahl der Objekte verringern, weshalb wir diesen Aspekt in unserem siebten Kapitel aufnehmen werden.

Bis jetzt sind nur relativ einfache Widgets über Interaktionsformen angebunden worden, wobei einfach sich in diesem Zusammenhang auf Widgets bezieht, die entweder eine Aktion auslösen oder die Eingabe und Auswahl von Basiswerten ermöglichen. Die Handhabung solcher Art von Widgets läßt sich verhältnismäßig einfach beschreiben und in eine Methodenschnittstelle für eine Interaktionsform umsetzen. Für Widgets, die einen komplexeren Umgang ermöglichen wie zum Beispiel Tabellen- oder Treewidgets, gab es bisher noch keine ausreichenden Ansätze, wie sich die Handhabung solcher Widgets als Interaktionsform modellieren läßt.

5 Konstruktionsvorschlag für Tabellen

In den beiden vorangegangenen Kapiteln haben wir die Unterschiede in den Ansätzen verschiedener GUI Toolkits heraus gearbeitet, sowie deren Auswirkungen auf die Konstruktion von Software dargestellt. In der Folge haben wir dann diejenigen Konstruktionsprinzipien vorgestellt und diskutiert, mit denen wir innerhalb des WAM Leitbildes die Anbindung zwischen grafischen Oberflächen beziehungsweise deren Elementen und den zugeordneten Werkzeugen vornehmen.

Sicherlich ist es leicht einzusehen, daß uns Widgets, deren möglicher Inhalt von vorne herein bestimmbar ist, keinerlei Probleme bei der fachlichen und technischen Konstruktion bereiten. Zu diesen Widgets gehören beispielsweise Eingabefelder oder auch Listboxen. Wie verfahren wir aber mit Tabellen oder Tree's, bei denen der Typ des darzustellenden Inhaltes zum Zeitpunkt der Widgetkonstruktion noch nicht feststeht? Um diese Frage zu klären, werden wir uns nun mit einer Art von Widgets befassen, die in Abhängigkeit von ihrer Verwendung und ihres Inhaltes eine unterschiedliche Interaktion mit dem Benutzer erfordern. Stellvertretend für diese Gruppe von Widgets werden wir uns an dieser Stelle mit der Konstruktion eines Tabellenwidgets beschäftigen. In diesem Zusammenhang ist zu beachten, daß Tabellen eine beträchtliche Größe, bezogen auf die Anzahl der darzustellenden Information, erreichen können. Der Aspekt der Größe hat uns dazu bewogen, innerhalb dieser Konstruktion den Bereich der Vorhaltung von Werten für das eigentliche Widget besonders zu berücksichtigen.

Dieses Kapitel gliedert sich in drei Teile, wobei die ersten beiden Teile der Zielsetzung und Beschreibung der technischen Realisierung dienen, während der dritte Teil darstellt, in wie weit diese Konstruktion auf andere Widgets übertragbar ist.

5.1 Ziele der Konstruktion

Um die Ziele der Konstruktion bestimmen zu können, haben wir uns zunächst mit dem Einsatzbereich von Tabellen beschäftigt, um die intendierte Benutzung ableiten zu können. Grundsätzlich finden sich überall dort Beispiele für den Einsatz von Tabellen, wo große Datenbestände (zum Beispiel Kundendatenbanken) gefiltert und weiterverarbeitet werden müssen. Eine unmittelbar aus diesem Einsatzkontext hervorgehende Anforderung besteht also darin, große Mengen an Informationen übersichtlich darzustellen. Tabellen bieten beachtliche Möglichkeiten, das Ziel der Strukturierung einer Menge von Informationen zu erreichen. So können innerhalb einer Tabelle Werte unterschiedlicher Typen angezeigt und bearbeitet werden. Vom fachlichen Standpunkt aus betrachtet, kann man also den Inhalt einer Tabelle in einen entsprechend vom Anwendungskontext motivierten Zusammenhang stellen. Um die angesprochenen Möglichkeiten zu illustrieren, möchten wir stellvertretend für den Einsatz von Tabellen in grafischen Oberflächen die Software GTx der Firma Siemens anführen. Diese Software hat die Aufgabe, alle vertrieblichen Aufgaben der Innen- und Außendienste zu unterstützen. Ein großer Teil dieser Aufgaben wird durch die Angebotserstellung und die Auftragsbearbeitung eingenommen. Innerhalb der Angebotserstellung gilt es, individuelle Leistungsverzeichnisse zu erstellen, die den Umfang der Arbeiten bestimmen und gleichzeitig abgrenzen. In Abbildung 5-1 ist der Aufbau eines solchen Leistungsverzeichnis innerhalb der GTx - Oberfläche dargestellt. Für die sogenannte Leistungsübersicht hat man sich für den Einsatz von Tabellen entschieden, wobei die Inhalte der einzelnen Spalten bestimmten Kategorien (Positionsnummer, Arti-

kelnummer, Kurztext, etc.) zugeordnet sind. Innerhalb dieser Tabelle werden nur die Eckdaten einer Position dargestellt.

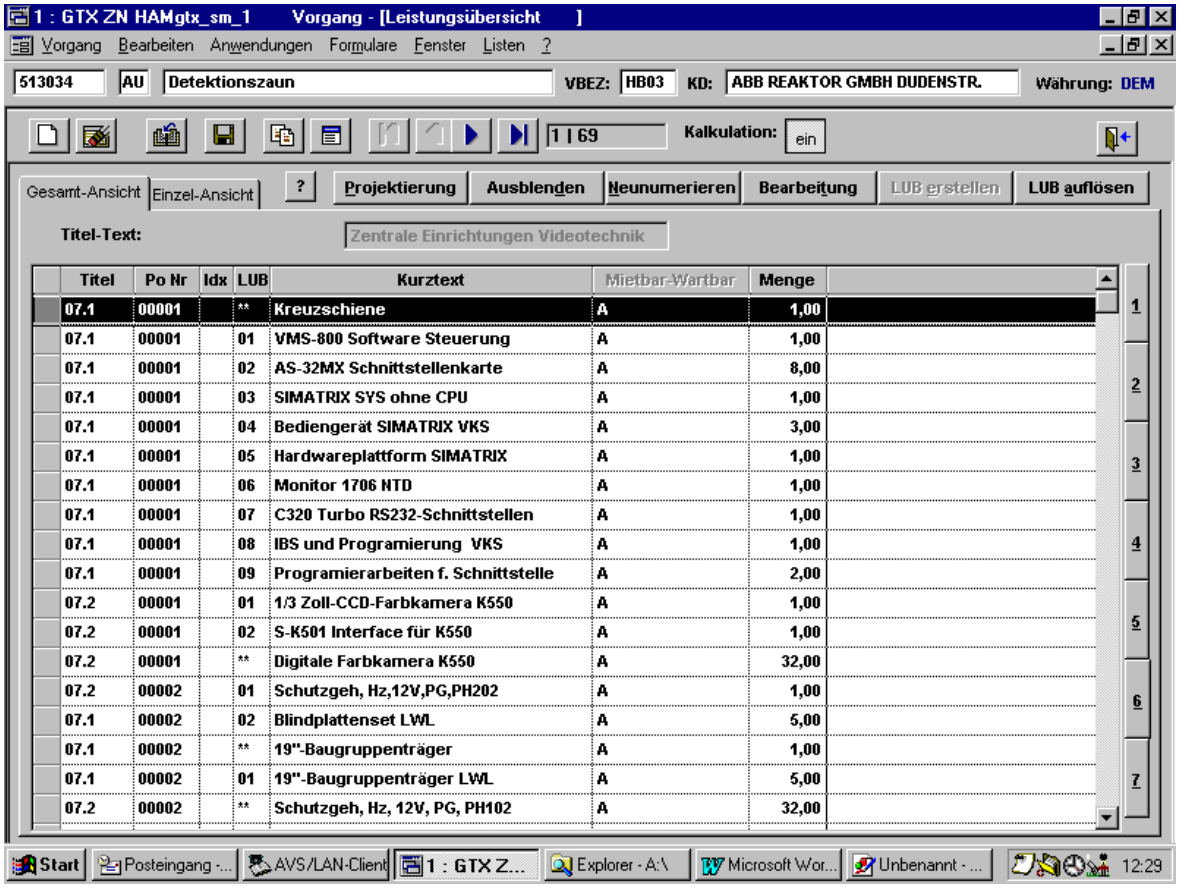

Abbildung 5-1: Leistungsverzeichnisse in GTx

Die für die Kalkulation notwendigen Details werden dem Benutzer über ein weiteres Werkzeug, der sogenannten Einzelansicht, zur Verfügung gestellt. Je nach Aufgabenstellung besitzen diese Leistungsverzeichnisse – Tabellen – eine beträchtliche Größe. Gängige Leistungsverzeichnisse kommen leicht über eine Anzahl von 100 Einzelpositionen.

Das vorgenannte Beispiel skizziert nur einen der vielfältigen Einsatzkontexte für die Präsentation von Materialien in Form von Tabellen. Dieses Beispiel ist von uns mit Bedacht gewählt worden, da in diesem Fall die Aufgabe des Widgets ausschließlich darin besteht, den logischen Aufbau der Tabelle in einer Oberfläche abzubilden und einen minimalen Satz an Funktionen zur Bearbeitung der Einträge anzubieten. Um deutlich zu machen, welchen Grad der Komplexität wir in dieser Arbeit im Zusammenhang mit Tabellen behandeln, sind die folgenden Fragen von Interesse.

- Wie sind die darzustellenden Werte strukturiert?
- Welche Funktionen sind durch das Widget anzubieten?

Die erste Frage befaßt sich mit dem logischen Aufbau der in einer Tabelle darzustellenden Werte. Unter dem logischen Aufbau einer Tabelle verstehen wir diejenige Struktur, welche im Rahmen der Modellierung durch einen Entwickler entstanden ist und zur Laufzeit die darzustellenden Werte enthält.

Im Bezug auf die Typisierung dieser Struktur können wir zwischen

- geordneten Listen gleichartiger Tupel,
- geordneten Listen potentiell unterschiedlicher Tupel und
- Wertemengen, die in Zeilen und Spalten angeordnet werden können

unterscheiden. Die Reihenfolge der einzelnen Möglichkeiten drückt gleichzeitig die ansteigende Komplexität aus. Geordnete Listen gleichartiger Tupel sind am leichtesten zu handhaben. Im Umgang mit diesen Listen können wir davon ausgehen, daß ein Tupel auf eine Zeile abgebildet werden kann. Erreichen können wir dieses durch die Nutzung des dieser Liste zugrunde liegenden Ordnungskriteriums. Zudem bestimmt die in diesem Fall für jedes Tupel identische Anzahl der in ihm enthaltenen Werte die Zahl der benötigten Spalten. Sowie wir unterschiedliche Tupel zulassen, können wir zwar immer noch ein Tupel einer Zeile zuordnen, wobei aber jede Zeile unterschiedlich aufgebaut werden kann. Bei beliebigen Wertemengen schließlich muß eine Abbildung definiert werden, die jeden darzustellenden Wert auf eine Zelle der Tabelle abbildet.

Die Frage nach der Funktionalität dieses Widgets umfaßt den Umgang eines Benutzers mit einer Tabelle. Als Beispiel können wir hier die bekannte Tabellenkalkulation Microsoft Excel anführen. Ein Benutzer kann mit den dort verwendeten Tabellen eine Vielzahl von Aufgaben, die von der Textverarbeitung bis zu einer Adressenverarbeitung reichen, erledigen, die mit den Aufgaben einer Tabellenkalkulation im eigentlichen Sinne nichts gemein haben. An diesem Beispiel wird deutlich wie weitreichend und universell Tabellen benutzt werden können. Dieser universelle Einsatz schafft uns allerdings Probleme, wenn wir für solche Widgets einen funktionalen Charakter fordern.

Wir haben uns in unserer Arbeit dafür entschieden, Tabellen als die Darstellung von geordneten Listen gleichartiger Tupel anzusehen. Auf der Seite der Funktionalität dieses Widgets halten wir die notwendige Funktionalität vor, um die Werte eines Tupels gezielt bearbeiten zu können. Wir wollen mit unserer Konstruktion somit auch nicht die Problematik einer kompletten Tabellenkalkulation erschlagen.

Ausgehend von dem am Anfang dieses Kapitels angeführten Beispiels möchten wir nun unser Verständnis von Tabellen darstellen. Zunächst werden wir unsere Sicht auf die äußere Struktur einer Tabelle vorstellen und im Anschluß daran, den fachlichen Umgang eines Benutzers mit Tabellen definieren. Mit dem Ziel, einen leicht handhabbaren und übersichtlichen Konstruktionsansatz für Tabellen zu erhalten, sind diese Definitionen vereinfachend zu verstehen.

> *Eine Tabelle besteht aus einer geordneten Liste von gleichartigen Tupeln. Hierbei repräsentiert ein Tupel eine Zeile in der Tabelle. Für alle Zeilen gemeinsam wird eine Unterteilung in mehrere Spalten vorgenommen, die dem Aufbau der Tupel entsprechen. Jede Zelle läßt sich eindeutig genau einer Zeile und genau einer Spalte zuordnen, wodurch die Abbildung auf einen Wert innerhalb eines Tupels gegeben ist.*

Aus dieser Definition der Struktur von Tabellen lassen sich nun bereits erste Einblicke für den Umgang eines Benutzers mit Tabellen ableiten. Hierbei müssen wir zunächst berücksichtigen, daß sich der Wert einer Tabelle aus den Inhalten der einzelnen Zellen zusammensetzt. Der Umgang mit einer Zelle innerhalb einer Tabelle erfolgt entsprechend des ihr zugewiesenen Typs. Somit unterscheidet sich ein Tabellen - Widget von anderen Widgets, wie zum Beispiel einem Textfeld, dadurch, daß für eine Gruppe von Zellen der

jeweilige Umgang realisiert werden muß. Der Wert einer Tabelle insgesamt wird schließlich dadurch bestimmt, daß die einzelnen Zellen mit Inhalten gefüllt werden. Im Zusammenhang mit grafischen Oberflächen erfolgt dieses durch das Editieren der einzelnen Zellen durch einen Benutzer. Im weiteren können wir nicht davon ausgehen, daß die Spalten einer Tabelle voneinander unabhängig sind. Vielmehr müssen wir sogar die Reihenfolge der Spalten als fachlich motiviert betrachten, das heißt das Ausfüllen einer Zelle kann innerhalb der aktuell bearbeiteten Zeile die Bearbeitung einer weiteren Zelle erfordern. In der Konsequenz bedeutet dies, daß anhand eines gültigen Wertes einer Zelle unter Umständen nicht die gesamte Zeile validiert werden kann. Anders ausgedrückt, muß man im Hinblick auf Tabellen davon ausgehen, daß der Handlungsabschluß nicht auf die Tabelle als Ganzes definiert werden kann, sondern sich immer auf die aktuelle Zeile bezieht (siehe auch [Blee99], Seite 18). Bei einfachen Eingabewidgets gibt es meisten einen expliziten Handlungsabschluß. Er wird in der Regel durch das Verlassen des Eingabefeldes ausgelöst. Dies gilt zwar auch für ein Tabellenwidget, doch zusätzlich gibt es noch einen impliziten Handlungsabschluß. Nach dem Editieren einer Zelle wird oft der Editiervorgang mit einer Nachbarzelle vorgesetzt. Aufgrund der möglichen Abhängigkeiten zwischen den Zellen einer Zeile muß das Werkzeug auch über diesen implizite Handlungsabschluß informiert werden. Nur so ist es für das Werkzeug möglich, die Konsistenz einer Zeile und damit der ganzen Tabelle zu erhalten. In Ergänzung zur Struktur einer Tabelle definieren wir nun den fachlichen Umgang eines Benutzers mit Tabellen.

> *Der Benutzer kann den Wert von Zellen auslesen, neu setzen und markieren. Die konkrete Ausführung der vorgenannten Tätigkeiten kann in Abhängigkeit von dem der Zelle zugewiesenen Typ differieren.*

Mit dieser Definition des Umganges vertreten wir eine sehr pragmatische Sicht, die zum Beispiel auch auf den Einsatz von relationalen Algebren verzichtet. Zusätzlich zur Definition der Struktur auf der einen und dem fachlichen Umgang mit Tabellen auf der anderen Seite, gilt es, die Anforderungen an ein derartiges Widget noch von anderen Seiten zu untersuchen.

Wie eingangs bereits erwähnt, kann die Wertemenge für eine Tabelle eine beträchtliche Größe annehmen. Hinsichtlich der Anzahl an darzustellenden Werten unterscheiden wir zwischen Wertemengen, deren Umfang bekannt ist und Mengen, deren Umfang sich nicht von vorne herein explizit bestimmen läßt. Ein Beispiel für die letztgenannte Situation kann man im Zusammenhang mit Datenbanken antreffen, bei denen die Anzahl der darzustellenden Informationen zum Beispiel von dem Ergebnis einer Filterabfrage abhängig ist. Es erscheint uns in diesem Zusammenhang vielversprechend, ein entsprechendes Prinzip im Widget zu verankern, so daß nicht für jede einzelne Zelle eine gesonderte Anfrage an das Werkzeug gestellt werden muß. Vielmehr sollten an dieser Stelle auch Möglichkeiten zum Caching von Werten vorhanden sein. Insbesondere im Zusammenhang mit Datenbanken als Datenquellen besitzt dieser Punkt eine große Bedeutung. Es gilt also für uns einen Weg zu finden, mit dem wir sowohl den Kommunikationsaufwand zwischen Werkzeug und Widget minimal halten können, als auch die Wertemenge im Widget mit der maßgeblichen Wertemenge des Werkzeuges konsistent halten zu können.

Bei einem Darstellungsmittel mit der angedeuteten Vielfalt taucht immer wieder die Frage auf, ob es sich bei solch einer Konstruktion noch um ein Widget oder bereits um ein Sub-Werkzeug handelt. So kann man sich ohne weiteres ein Werkzeug vorstellen, dessen Oberfläche aus einem Auflister und einer gruppierten Anzahl von Textfeldern besteht (siehe Abbildung 5-2). Innerhalb des Auflisters wird eine Liste derjenigen Elemente angezeigt, die mit Hilfe der Textfelder bearbeitet werden können. Somit haben wir auf der

Oberfläche einen Auflister mit dem Editor eines Materials kombiniert. Wird nun ein Element innerhalb des Auflisters selektiert, so wird dieses gleichzeitig in den Editor geladen.

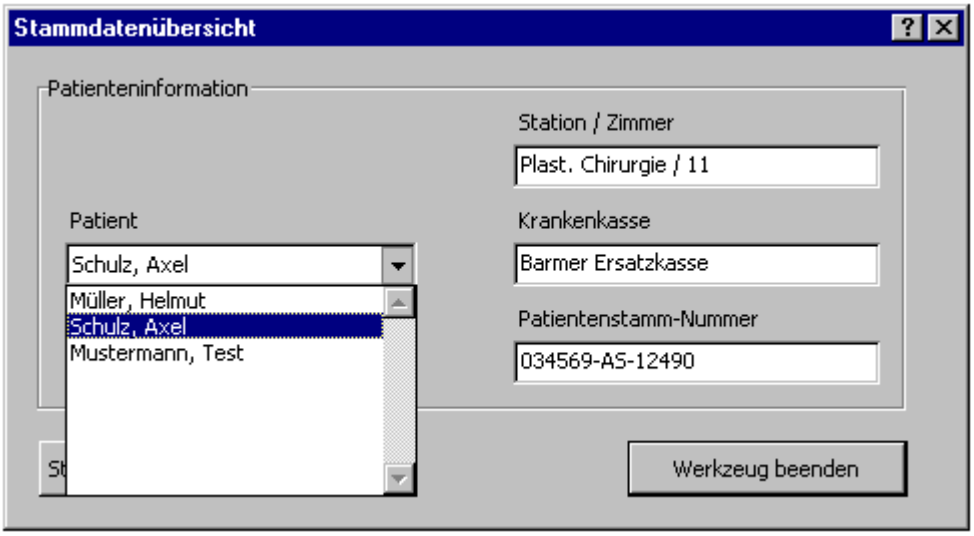

Abbildung 5-2: Die „Werkzeugform" einer Tabelle

Mit Hilfe einer Tabelle sind wir aber in der Lage, diese beiden Aufgaben zusammenzufassen. In diesem Fall bestünde die Zeile einer Tabelle aus der Bezeichnung, die vorher in dem Auflister angezeigt wurde und der dem Material zugehörigen Einträge. Die Anzeige und die Bearbeitung von Materialien kann also in einer Tabelle kombiniert werden. Um die Frage, ob es sich in diesem Fall um ein Sub - Werkzeug handelt oder nicht, klären zu können, müssen wir die folgenden drei Aspekte betrachten:

- Kontextunabhängigkeit
- Funktionaler Charakter
- Austausch von Werten

Für ein typisches Sub - Werkzeug könnten wir alle vorgenannten Punkte negativ beantworten. Vorerst gehen wir jedoch davon aus, daß wir eine Tabelle unter den Gesichtspunkten eines Widgets behandeln und konstruieren können.

Wir betrachten ein Tabellen - Widget als kontextunabhängig, da die Präsentation des Widgets nicht direkt von Zuständen in einem Werkzeug abhängig ist. Vielmehr soll die Darstellung der Tabelle auf direkt am Widget vorgenommenen Einstellungen basieren. Die Tabelle selbst wird sich also unabhängig davon, ob sie für Leistungsverzeichnisse oder für Lehrerkarten benutzt wird, präsentieren. Die Handhabung eines Widgets zeichnet sich durch einen funktionalen Charakter aus der Sicht des Werkzeuges aus. Die Interaktionskomponente übergibt einen Wert an das Widget und erhält für den Fall der Bearbeitung durch einen Benutzer einen anderen Wert des selben Typs zurück. Der funktionale Charakter drückst sich also durch die Möglichkeit aus, eine Abbildung zwischen den Eingabe- und den Rückgabewerten zu definieren. Während der Bearbeitung durch einen Benutzer erhält die Interaktionskomponente keinen Zugriff auf andere, interne Zustände des Widgets. Um in einer Tabelle, die unserer Definition entspricht, die gewünschten Werte anzeigen zu können, wird zwischen dem Widget und der Interaktionskomponente eine geordnete Liste von gleichartigen Tupeln ausgetauscht. Grundvoraussetzung hierfür ist, daß das Widget in der Lage ist, mit den für die Elemente eines Tupels benötigten

Präsentationsformen umgehen kann. Diese Diskussion stützt sich immer auf die von uns vereinfachten Definitionen der Struktur einer Tabelle und dem möglichen Umgang eines Benutzers mit einer Tabelle. So sind wir zum Beispiel nicht in der Lage auf der Basis dieser Definitionen eine Tabellenkalkulation adäquat zu definieren. Tabellenkalkulationen sind aber unzweifelhaft ein häufiger Anwendungsfall für derartige Widgets. Sofern wir unsere Definitionen auf diesen Bereich erweiterten, wäre zumindest auch der funktionale Charakter nicht mehr gegeben, da es keine Möglichkeit gibt, von vorne herein zu bestimmen, welche Aufgaben ein Benutzer mit diesem Widget durchführt. In diesem Zusammenhang können wir dann auch nicht mehr von einer gleichbleibenden Struktur der Tabelle ausgehen, da zum Beispiel Funktionen existieren, die es erlauben, einen sich über mehrere Spalten erstreckenden Bereich zu einer Zelle zusammen zu fassen. Spätestens an dieser Stelle haben wir ein Problem, diese Funktionalität auf einer formalen Ebene zu definieren. Wenn wir also im Rahmen dieser Arbeit von Tabellen sprechen, dann meinen wir immer die Präsentation von geordneten Listen gleichartiger Tupel in einem Raster aus Zeilen und Spalten.

Innerhalb des JWAM Rahmenwerks sind zur Zeit noch keine Möglichkeiten vorgesehen, Widgets wie zum Beispiel Tabellen oder Trees unter Berücksichtigung der Trennung von Interaktion und Präsentation anzubinden. Die einzige Möglichkeit, solchen Widgets habhaft werden zu können, besteht darin, eine direkte Anbindung der SWING Komponenten an die Interaktionskomponente vorzunehmen. In diesem Fall gilt es jedoch zu beachten, daß diese Anbindung unter einer technisch motivierten Schnittstelle vorgenommen wird. Das heißt, daß die Struktur dieses Widgets gegenüber der Interaktionskomponente nicht verborgen wird. Wir sind der Meinung, daß hinsichtlich eines fachlich motivierten Umganges mit Widgets dieser Art ebenfalls diese Form der Abstraktion eingesetzt werden sollte. Wir werden exemplarisch Tabellen in das JWAM Rahmenwerk integrieren und die Rahmenbedingungen hierfür aufzeigen.

Abschließend formulieren wir noch einmal in Stichpunkten die Ziele für unsere Konstruktion eines Tabellenwidgets:

- Erfüllung unserer Definition der Struktur von Tabellen
- Interaktion mit einem Benutzer gemäß unserer Definition des fachlichen Umgangs
- Möglichst wenig Kommunikation mit dem Werkzeug
- Konsistente Datenhaltung zwischen Werkzeug und Widget
- Abgrenzung zu einem Sub - Werkzeug
- Integration in das JWAM Rahmenwerk

5.2 Technische Realisierung eines Tabellenwidgets

Aus der Zielvorstellung heraus, Tabellen in das JWAM – Rahmenwerk zu integrieren, ergeben sich für uns sofort einige Rahmenbedingungen in Bezug auf die von uns angestrebte Konstruktion. Obwohl wir in Abschnitt 4.2 bereits die Vor- und Nachteile von Interaktionsformen und Interaktionstypen diskutiert haben, können wir nur einen Teil der Ergebnisse an dieser Stelle umsetzen. Dem Anwendungsentwickler werden zur Zeit über das JWAM Rahmenwerk bereits Widgets angeboten, die über das Prinzip der Interaktionsformen an die Werkzeuge angebunden werden. Für die Integration eines zusätzlichen Widgets, wie einer Tabelle, werden wir die gleichen Prinzipien verwenden, die bereits für

die vorhandenen Widgets implementiert wurden, um einen konzeptionellen Bruch innerhalb des Rahmenwerkes zu vermeiden. Letztendlich bedeutet dies, daß wir den Umgang eines Benutzers mit einer Tabelle und deren Darstellung über ein Paar, bestehend aus Interaktionsform und Präsentationsform, umsetzen werden.

Um den Aufbau unseres Widgets deutlich machen zu können, werden wir zunächst die technischen Konzepte erläutern, die notwendig sind um große Mengen an Werten darstellen zu können um danach die einzelnen Bestandteile des Widgets näher zu erläutern. Im weiteren werden wir dann das Zusammenspiel der verschiedenen Teile mit Hilfe von Interaktionsdiagrammen verdeutlichen. Abschließend werden wir dann auf Änderungen an den vorhandenen Widgets im JWAM Rahmenwerk eingehen, die notwendig sind, um unsere Tabelle integrieren zu können.

5.2.1 Darstellung großer Wertemengen

Die Überschrift dieses Abschnittes drückt bereits eine Problematik aus, die im Bereich von GUI Komponenten besondere Beachtung verdient. Wir sprechen hier von großen Wertemengen, ohne die eigentliche Ausdehnung genauer zu fassen. Es ist uns aber in vielen Fällen aber nicht möglich, die Anzahl der darzustellenden Werte genau zu fassen. Die Verwaltung von Kundenstammdaten kann zum Beispiel mehrere hundert Einträge oder aber auch mehrere tausend Einträge umfassen. Diese Unschärfe zieht sich dann bis in die Oberfläche durch. So wäre es eine unzulässige Einschränkung, für eine Listbox, in der alle Kundennamen enthalten sein sollen, eine maximal darstellbare Zahl von Einträgen festzulegen. Für Widgets, an die diese Anforderung gestellt wird, ist es also notwendig, zu klären, auf welche Art und Weise sie mit den darzustellenden Werten versorgt werden können. Das Ziel dieses Abschnittes liegt darin begründet, die zur Verfügung stehenden Alternativen aufzuzeigen und zu erläutern. Eine Bewertung der einzelnen Alternativen und die Entscheidung für eines der vorgestellten Konzepte ist Bestandteil des unmittelbar folgenden Abschnittes. Für die Klärung dieses Sachverhaltes sehen wir die hier vorgestellten fünf Alternativen.

Sofern man für derlei Widgets einen fachlichen Behälter vorsieht, der die darzustellenden Fachwerte enthält, können wir so den Austausch der darzustellenden Information zwischen Interaktionskomponente und Widget realisieren. Allerdings verlieren wir mit dieser Konstruktion den Aspekt des Umganges auf Basis der Wertsemantik zwischen Interaktionskomponente und Interaktionsform. Eine Grundvoraussetzung besteht hierbei darin, daß der Behälter komplett mit allen Werten gefüllt werden muß. Als Folge hiervon ergibt sich, daß der Umgang mit Massendaten zur Laufzeit Probleme hinsichtlich der Performanz hervorrufen kann. Im weiteren ist hier besonders auf eine effiziente Verwaltung der einzelnen Werte, die in dem Behälter enthalten sind, zu achten.

Eine weitere Möglichkeit besteht in unseren Augen darin, einen zusammengesetzten Fachwert zu modellieren. In diesem Fall wird die in einem Widget darzustellende Information in einem speziellen Fachwert untergebracht, der zwischen der Interaktionskomponente und der Interaktionsform ausgetauscht wird. Dieser spezielle Fachwert muß in der Lage sein, unabhängig vom konkreten Anwendungskontext, beliebige andere Fachwerte zu enthalten. Sowie dieser Fachwert diese Möglichkeit nicht besitzt, würde dies die individuelle Entwicklung eines Fachwertes für jedes, in einem bestimmten Kontext eingesetztes, Widget nach sich ziehen. Grundsätzlich hat das Prinzip des Austausches von Werten zwischen Widget und Werkzeug starke Auswirkungen auf die Implementierung. Die Nachbildung von Wertsemantik in objektorientierten Sprachen ist meist nur mit hohem Entwicklungsaufwand und hohen Laufzeitkosten möglich. Toleriert man den

erhöhten Aufwand für die Entwicklung, so bliebe das Thema der Laufzeitkosten, welche sich letztlich im Zusammenhang mit größeren Wertemengen negativ auf die Antwortzeiten der Oberfläche auswirken.

Im Zusammenhang mit Widgets, deren Einsatz nicht auf die Darstellung und Bearbeitung eines bestimmten Typs beschränkt ist, stellt sich die Frage, ob nicht eine reine Parametrisierung des Widgets ausreichend ist. In diesem Fall bestimmt der Entwickler der Oberfläche bereits zur Designzeit die Typen der darzustellenden Werte. Sowie nun ein Wert durch das Widget dargestellt werden soll, fordert das Widget selbst den entsprechenden Wert über die Interaktionsform bei der Interaktionskomponente ab. Diese Anforderung können wir technisch auf der Basis eines Request-Mechanismus realisieren. Somit existiert bei diesem Ansatz nicht die Notwendigkeit, Werte für das Widget vorzuhalten. Nachteilig hingegen ist der zu erwartende hohe Kommunikationsaufwand zwischen allen beteiligten Komponenten, der wiederum im Hinblick auf die Anzahl der darzustellenden Inhalte zu höheren Antwortzeiten führen kann.

Die vorstehenden Lösungsansätze gehen von den beiden extremen Fällen aus, daß entweder ein Objekt existiert, welches alle darzustellenden Werte beinhaltet oder das Widget selbst die entsprechenden Komponenten anfragen muß. Ein Prinzip das zwischen diesen beiden Extremen vermittelt, stellt unsere vierte Alternative dar. Hierbei wird ein Modell dazu benutzt, die Präsentationsform mit den darzustellenden Werten zu versorgen. Das Modell selbst verfährt gegenüber der Interaktionskomponente nach dem „Hollywood" Prinzip („Don't call us, we call you"), das heißt für den Fall daß Werte angezeigt werden sollen, die zur Zeit noch nicht im Modell bekannt sind, besteht die Aufgabe des Modells darin, diese zu besorgen. Unmittelbar mit dem Einsatz eines Modells dieser Art ist eine Veränderung des Kontrollflusses zwischen Oberfläche und Interaktionskomponente verbunden.

Die fünfte Alternative stellt das Konzept der Streams dar. In den vielen Programmiersprachen sind die Ein- und Ausgaberoutinen als Streams implementiert, worin der Bekanntheitsgrad dieses Konzeptes begründet liegen dürfte. Unser Verständnis basiert dabei auf der Sichtweise von Holyer in [Hol93, Seite 56]:

> *A stream is a list which is thought of as having its members produced one by one as required – for instance, a list of characters typed on the keyboard during the running of an interactive program, or an infinite list which ‚unwinds' as needed.*

Grundsätzlich ist ein Stream also eine Liste deren Dimension, im Sinne der Anzahl der in ihm enthaltenen Elemente, nicht begrenzt ist. Zusätzlich gehen wir davon aus, daß ein solcher Stream immer typisiert ist, das heißt es ist eine Aussage über den Typ seines Inhaltes möglich. Bezogen auf den Austausch von Werten mit dem Widget kann hierüber auch das Prinzip der Wertsemantik aufrecht erhalten werden. Ein weiterer interessanter Punkt in diesem Konzept besteht darin, daß über einen Index gezielt auf die Elemente eines Streams zugegriffen werden kann. Für die Versorgung eines Widgets mit darzustellenden Werten mit Hilfe dieses Konzeptes, wird also eine Verbindung zwischen der Interaktionskomponente und dem Widget etabliert. Für jedes so anzusprechende Widget ist von der Interaktionskomponente ein eigener Stream zu unterhalten. Streams stellen also ein dynamisches Konzept dar, in welche die darzustellenden Werte eingetragen werden, sowie diese der Interaktionskomponente zur Verfügung stehen. Letztlich enthält ein Stream zur Laufzeit alle darzustellenden Werte. Das Widget greift dann über entsprechende Methoden auf diesen Stream zu, mit dem Ziel sich die benötigten Werte zu beschaffen. Für den Fall, daß sich zur Laufzeit ein Wert ohne lokale Beeinflussung

geändert hat (verteilte Systeme), muß ein Mechanismus vorgesehen werden, der dem Widget mitteilt, daß sich an einer Stelle im Stream etwas geändert hat. Das Widget kann somit sinnvoll darauf reagieren, indem es den entsprechenden Wert erneut zur Anzeige bringt. Für die Rückgabe von Werten nach der Bearbeitung durch einen Benutzer bieten sich zwei Wege an. Zum einen kann man einen separaten Ausgabestream vorsehen, wobei hier zu beachten ist, das einem Wert im Ausgabestream ein Wert im Eingabestream zugeordnet werden muß, da die Interaktionskomponente ansonsten nicht in der Lage ist, dieses Ergebnis entsprechend zu behandeln. Zum anderen besteht die Möglichkeit, die durch einen Benutzer veränderten Elemente in den bereits bestehenden Stream zurück zu schreiben und die Interaktionskomponente dann über ein Command zu benachrichtigen, welches Element sich geändert hat.

In diesem Abschnitt sind von uns Wege beschrieben worden, mit denen Widgets mit darzustellenden Werten versorgt werden können. Insbesondere unter der Annahme, daß die Anzahl der darzustellenden Werte nicht von vorne herein bekannt ist. Welches der Konzepte wir für unsere Konstruktion einer Tabelle gewählt haben, beschreiben wir in dem folgenden Abschnitt. Die Auswahl eines solchen Konzeptes für ein bestimmtes Widget sollte aber immer berücksichtigen, für welchen Einsatz es gedacht ist. So ist die Implementierung eines Stream Konzeptes für einen Button zwar möglich aber dem Einsatzkontext nicht angepasst.

5.2.2 Vorhaltung der Werte im Widget

In diesem Abschnitt werden wir darstellen, für welches der angesprochenen Konzepte wir uns entschieden haben. Die zur Zeit in das JWAM - Rahmenwerk integrierten Interaktionsformen tauschen mit der Interaktionskomponente Werte auf der Basis von Wertsemantik aus. Im Sinne dieser Konstruktion sollte auch der Umgang zwischen einer Interaktionskomponente und einer Interaktionsform für Tabellen realisiert sein. Wenn wir also diesen funktionalen Charakter aufrecht erhalten wollen, müssen wir auf jeden Fall eine entsprechende Funktion definieren können.

Eine Anbindung der Tabelle über den von uns angesprochenen reinen Request - Mechanismus kommt für uns nicht in Betracht, da der zu erwartende Kommunikationsaufwand nicht unerheblich ist und das System zur Laufzeit zusätzlich belastet.

Zwischen der Interaktionskomponente und der eigentlichen grafischen Oberfläche werden nur Werte ausgetauscht. Hierzu zählen zum einen die Standardtypen wie Integer und String oder auch eigene Fachwerte (zum Beispiel Kontonummer). Deshalb scheidet in unseren Augen auch der Einsatz eines fachlichen Behälters aus, für den in [Züll98, Seite 190] folgende Dienstleistungen ausgemacht werden:

> *In Behältern können Materialien aufbewahrt, gesammelt und angeordnet werden.*
>
> *Ein Behälter kann die aufbewahrten Materialien verwalten und über die Sammlung fachliche Auskunft geben.*
>
> *Materialien können mit ihren Behältern an verschiedene Orte transportiert werden. Damit bieten Behälter den Ansatz zur Kooperation und Koordination.*

Die Modellierung eines Tabellenfachwertes scheint hier der gangbare Weg zu sein. Eine Auswirkung dieser Konstruktion besteht darin, daß die Tabellenfachwerte nicht verändert

werden können. Änderungen an einer einzigen Zelle haben gemäß dem Prinzip der Wertsemantik zur Folge, daß ein neuer Fachwert erzeugt werden muß. Tabellen werden aber häufig genau dann eingesetzt, wenn es gilt, große Mengen an Werten strukturiert anzuzeigen. Hieraus ergeben sich bei der Bearbeitung durch einen Benutzer hinsichtlich des Verhaltens zur Laufzeit deutliche Nachteile, die in den Möglichkeiten der Realisierung von Wertsemantik in objektorientierten Sprachen begründet sind. Die Performanz von Oberflächen ist aber eines der zentralen Probleme, so daß wir versuchen müssen, diese Probleme zu umgehen. Aus diesem Grund haben wir uns auch gegen den Einsatz eines speziellen Fachwerts für Tabellen ausgesprochen.

Erich Gamma in seinem Vortrag über Universal Light Clients Widgets für Tree's und Tabellen als datenzentriert bezeichnet. Datenzentriert bedeutet für ihn in diesem Zusammenhang, daß ein Modell die benötigten Werte für das jeweilige Widget bereit stellt. Diese Konstruktion erscheint besonders unter Berücksichtigung der mit ULC angestrebten Verteilung als sinnvoll. Das unter ULC eingesetzte Modell realisiert mit der eigentlichen Applikation ein Protokoll, so daß sich in dem Modell immer die gerade benötigten aktuellen Werte befinden. Wir haben uns ebenfalls dazu entschlossen, die Vorhaltung von Werten für eine Tabelle über ein Modell vorzunehmen.

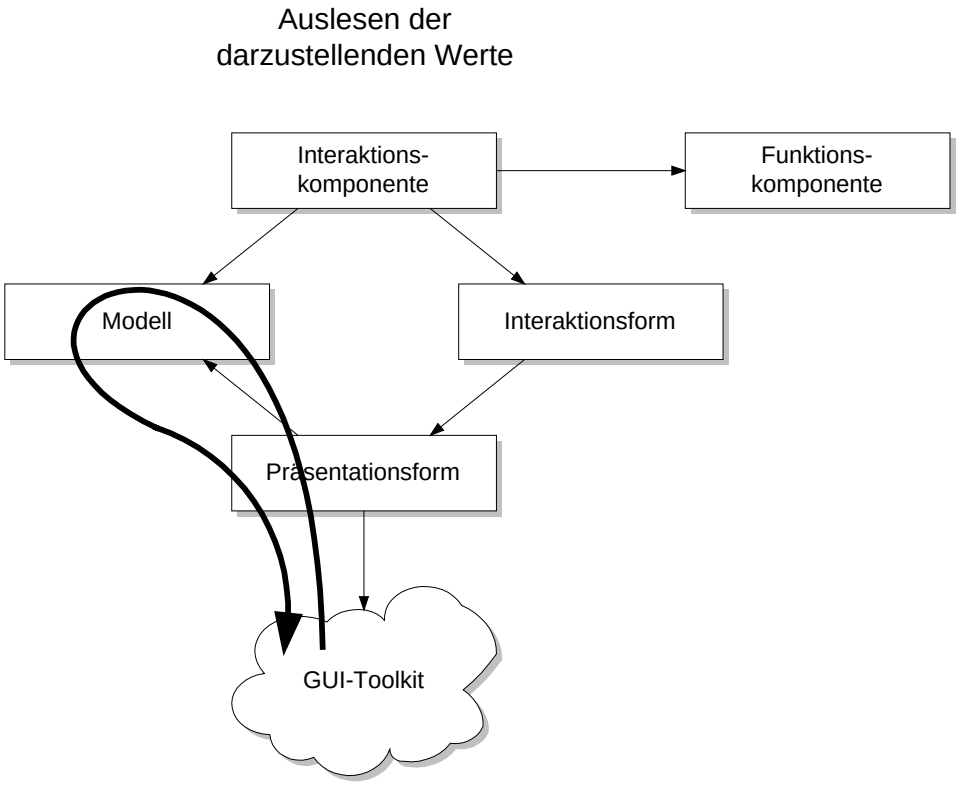

Abbildung 5-3: Änderungen des Kontrollflusses (Teil 1)

Nach wie vor verfolgen wir, bezogen auf die Inhalte der einzelnen Zellen, das Prinzip der Wertsemantik. Die Interaktionskomponente stellt der Interaktionsform ein Modell zur Verfügung, in das gemäß den Änderungen eines Benutzers von der Präsentationsform angepasst werden kann. Im Gegensatz zu der Konstruktion eines Tabellenfachwertes, wird mit Hilfe des Modells der Inhalt der Tabelle nicht als ein einziger Wert repräsentiert. Das von uns eingesetzte Modell dient also als Quelle für diejenigen Werte, die innerhalb der Tabelle dargestellt werden sollen. Aus der technischen Sicht betrachtet, stellt unser Modell ein Interface dar. Diese Art der Realisierung eröffnet uns zwei verschiedene Wege,

mit denen das Modell auf benötigte Werte zugreifen kann. Entweder enthält das Modell selbst die darzustellenden Werte oder es besitzt Informationen darüber, mit Hilfe welcher Methoden der Interaktionskomponente diese besorgt werden können. Die Entscheidung darüber, welche Möglichkeit hierfür genutzt wird, liegt bei dem Anwendungsentwickler. Zusätzlich ermöglicht dieses Prinzip den Einbau von Caching Algorithmen, die zum Beispiel im Zusammenhang mit Datenbanken einzelne Werte oder Gruppen von Werten auf Verdacht laden können. Anfallende Prüfungen des Tabelleninhaltes auf Konsistenz sind nicht Aufgabe des Modells, sondern wie bisher Aufgabe der Interaktions- und Funktionskomponente. Dies betrifft insbesondere die Validierung von Zellinhalten, die wiederum vom Inhalt anderer Zellen derselben Zeile abhängig sind. Um dieses realisieren zu können, bedienen wir uns eines Requests, welcher an die Interaktionskomponente weitergeleitet wird. In diesem Requests ist die gesamte von einem Benutzer veränderte Zeile enthalten, wodurch wieder das Prinzip eines Tupels widergespiegelt wird.

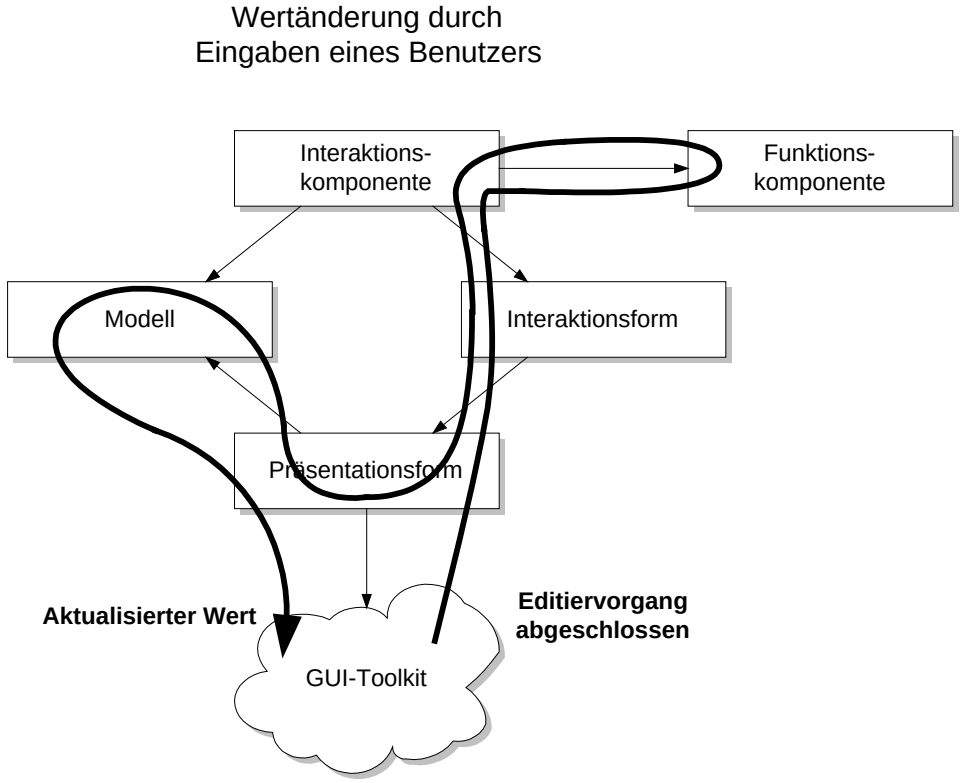

Abbildung 5-4: Änderungen des Kontrollflusses (Teil 2)

Die Schnittstelle des als Interface realisierten Modells stellt sich für die Präsentationsform wie folgt dar:

- `getColumnCount()`
- `getRowCount()`
- `getValue(row, column)`
- `setValue(row, column)`
- `canEditCell(row, column)`

Mit Hilfe der ersten beiden Methoden wird die Dimension der Tabelle bestimmt. Die Tatsache, daß die weiteren Methodenaufrufe mit dem Index der Zeile und der Spalte parametrisiert werden, beruht auf der Anforderung des Widgets, einzelne Zellen darstellen und bearbeiten zu können. In Richtung des Werkzeuges können wir jedoch immer noch von einer geordneten Liste von Tupeln ausgehen, die in dem Modell enthalten ist. Die Methode `getValue` schließlich liefert den in der über die Zeile und die Spalte bestimmten Zelle eingetragenen Wert zurück. Über die sondierende Methode `canEditCell` wird bestimmt, ob die entsprechende Zelle editiert werden darf oder nicht. Aufgrund der Tatsache, daß diese sondierende Methode eine Eigenschaft der darstellbaren Wertemenge widerspiegelt, haben wir diese Methode innerhalb des die Werte vorhaltenden Modells untergebracht.

Wie wir bereits angedeutet haben, ändert sich durch den Einsatz von Modellen der bisher bekannte Kontrollfluß zwischen der Interaktionskomponente und der Oberfläche. Mit den Abbildungen 5-3 und 5-4 möchten wir das Verhalten des Kontrollflusses für die Anzeige und das Bearbeiten von Zellinhalten illustrieren. Um die Tabelle mit Werten zu füllen, greift die Präsentationsform auf das Modell zurück. In diesem Beispiel gehen wir davon aus, daß das Modell die entsprechenden Werte enthält, so daß keine Anfragen an die Interaktionskomponente über das Modell notwendig sind. Für den Fall, daß der Benutzer Inhalte der Tabelle ändert oder neu einfügt, wird der entsprechende Wert über einen Request an die Interaktionskomponente weitergeleitet. Die Prüfung dieser Werte bleibt also die Aufgabe der Interaktions- beziehungsweise der Funktionskomponente. Nachdem die Interaktionskomponente die Prüfung durchgeführt hat, kann der Request akzeptiert oder abgelehnt werden. Für den Fall, daß die Interaktionskomponente den Request akzeptiert hat, wird das Modell durch das Widget entsprechend der Änderungen des Benutzers angepaßt.

Sofern wir das Modell als eine Liste von Tupeln verstehen, können wir das Modell als den Parameter einer Funktion verstehen. Als Rückgabewert erhalten wir dann eine durch die Funktion veränderte Liste von Tupeln. In unserem Fall stellt die Tabelle diese Funktion dar, die als Eingabe ein Modell entgegennimmt. Die Rückgabe der Tabelle an das Werkzeug besteht aber aus pragmatischen Gründen nur aus dem der geänderten Zeile zugrunde liegenden Tupel. Insgesamt haben wir also eine Abbildung einer Liste von Tupeln auf ein einzelnes Tupel geschaffen.

Die genauen Details des Ablaufes werden wir im Abschnitt 5.2.7 anhand von Interaktionsdiagrammen erläutern.

5.2.3 Das Layout Modell

Das im vorigen Abschnitt beschriebene Modell beinhaltet keinerlei Informationen darüber, wie letztlich die einzelnen Werte beziehungsweise wie die Tabelle selbst in der jeweiligen Oberfläche dargestellt werden soll. Die Art der Darstellung der verschiedenen Typen innerhalb einer Tabelle wird bereits zum Zeitpunkt der Erstellung der Oberfläche über die Festlegung der zu verwendenden Präsentationsform bestimmt. Damit gilt es für uns zu klären, wie die Darstellung der Tabelle selbst zu definieren ist. Wir sind der Meinung, daß im Zusammenhang mit Tabellen zwischen denjenigen Einstellungen unterschieden werden muß, welche die statische Einbettung von Tabellen in die entsprechende Oberfläche selbst beschreiben und Einstellungen, die zur Laufzeit variieren können.

Zu Einstellungen, welche die Einbettung der Tabelle in eine Oberfläche definieren, gehören zunächst die Größe und die Position, sowie der Name des Widgets. Offenbar

müssen derlei Parameter zur Designzeit der Oberfläche entschieden und eingetragen werden. Zusätzlich zu den vorgenannten Parametern findet zum Zeitpunkt der Erstellung der Oberfläche die Festlegung der zur Darstellung der unterschiedlichen Inhalte benötigten Präsentationsformen statt.

Zu den zur Laufzeit variablen Größen gehören die Spaltenbreite und –höhe, sowie die zu verwendenden Spaltenüberschriften. Sicherlich kann man sich noch weitere Parameter vorstellen, die durch einfache Erweiterungen dieses Modells versorgt werden können. Auch bei dem hier verwendeten Modell, existiert für die Präsentationsform die Möglichkeit, die Eintragungen in diesem Modell direkt zu manipulieren. Ursache hierfür ist die Art und Weise, mit der Einstellungen zum Beispiel an der Spaltenbreite durch den Benutzer durchgeführt werden. Die gängigen Oberflächen unterstützen hier den Benutzer mit dem Prinzip der direkten Manipulation, das heißt die Auswirkungen der Benutzeraktionen werden sofort dargestellt. Vom technischen Standpunkt aus betrachtet ergibt sich die Konsequenz, diese Änderungen möglichst direkt bekannt zu machen, damit die Darstellung der Oberfläche noch performant erfolgen kann. Parallel dazu wird allerdings ein Command versendet, daß die Interaktionskomponente darüber informiert, daß sich der Inhalt des Layout Modells geändert hat. Die Kommunikation mit der Präsentationsform wird über die im folgenden erläuterte Schnittstelle des Modell Interfaces realisiert.

- `canChangeColumnWidth (pos)`
- `getColumnWidth (pos)`
- `setColumnWidth (pos, width)`
- `getRowHeight ()`
- `setRowHeight (height)`

Aufgrund der Tatsache, daß der Präsentationsform ein schreibender Zugriff auf das Modell gewährt wird, müssen wir dem Eigentümer des Modells die Möglichkeit geben, Änderungen an den Werten zu verhindern. Dies geschieht mit Hilfe der Methode `canChangeColumnWidth`, mit der individuell für jede Spalte entschieden werden kann, ob eine Änderung der Breite erwünscht ist oder nicht. Für jede Aktualisierung der Oberfläche, die in Abhängigkeit von dem verwendeten Toolkit sogar durch eine Mausbewegung ausgelöst werden kann, benötigt die Präsentationsform die Spaltenbreite und die Zeilenhöhe, die sie sich über die vorstehenden Methoden beschaffen kann.

- `hasColumnTitle ()`
- `canEditColumnTitle (pos)`
- `getColumnTitle (pos)`
- `setColumnTitle (pos, title)`

Zusätzlich werden von der Präsentationsform noch Informationen bezüglich des für eine Spalte zu verwendenden Titels benötigt. Mit Hilfe der ersten beiden Methoden kann die Präsentationsform sondieren , ob ein Titel vorhanden ist. Analog zu den Zeilenhöhen und den Spaltenbreiten existiert eine `set`- und eine `get`-Methode, welche die Bearbeitung des Spaltentitels ermöglichen.

5.2.4 Interaktions- und Präsentationsstrategie

In den beiden vorangegangenen Unterkapiteln haben wir beschrieben, wie in unserer Konstruktion die benötigten Werte und die für das Layout einer Tabelle notwendigen Einstellungen vorgehalten werden. Im Anschluß hieran möchten wir das von uns gewählte Konzept vorstellen, mit dem wir zum einen der Zuordnung von bestimmten Formen der Interaktion zu einzelnen Spalten, als auch der Koordination der unterschiedlichen Modelle gerecht werden wollen.

Wir gehen hierbei davon aus, daß der Entwickler den Verwendungskontext der konkreten Tabelle kennt. Somit ist er in der Lage, diejenigen Interaktionsformen zu bestimmen, die den korrekten Umgang mit den Zellen einer Spalte in Abhängigkeit von einem entsprechenden Kontext ermöglichen. Gemäß unserer Sicht auf Tabellen, besteht die Möglichkeit, allen Spalten unterschiedliche Darstellungen zuzuweisen. Unmittelbar damit verbunden ist die eben angesprochene Zuordnung der Interaktionsformen zu den Spalten. Das heißt, in Abhängigkeit von der aktuell verwendeten Präsentationsform muß die korrespondierende Interaktionsform zur Laufzeit angekoppelt werden, um auf diesem Wege die notwendigen Einstellungen vornehmen zu können. Im Gegensatz zu anderen Widgets, wie zum Beispiel einem Textfeld, muß für eine konkrete Tabelle somit immer ein an dem entsprechenden Einsatzkontext orientierter Umgang definiert werden. Die eigentliche Funktionalität der Tabelle wird durch diese individuelle Ausrichtung jedoch nicht beeinflußt. Vor dem Hintergrund dieser Problemstellung haben wir uns bei der Konstruktion für ein Strategiemuster entschieden. In [GHJV96, Seite 333] finden wir in Bezug auf den Zweck dieses Verhaltensmusters:

> *Definiere eine Familie von Algorithmen, kapsele jeden einzelnen und mache sie austauschbar. Das Strategiemuster ermöglicht es, den Algorithmus unabhängig von ihn nutzenden Klienten zu variieren.*
>
> *Verwende das Strategiemuster, wenn sich viele verwandte Klassen nur in ihrem Verhalten unterscheiden. Strategieobjekte bieten eine Möglichkeit, eine Klasse mit einer von mehreren möglichen Verhaltensweisen zu konfigurieren.*

Für uns bedeutet dies, daß wir eine Möglichkeit haben, den Umgang des Benutzers mit einer Tabelle in Abhängigkeit von dem speziellen Einsatzbereich zu definieren. Verantwortlich für diese Anpassung ist der Anwendungsentwickler. Um diesen Aspekt zu realisieren, haben wir ein weiteres Interface eingeführt, welches von uns als Interaktionsstrategie bezeichnet wird. Der Anwendungsentwickler muß nun also ein Strategieobjekt zur Verfügung stellen, welches dieses Interface implementiert und damit den speziellen Umgang eines Benutzers in dem jeweiligen Kontext mit einer Tabelle koordiniert. Dieses Objekt wird der Interaktionsform zum Zeitpunkt der Erzeugung durch die Interaktionskomponente übergeben.

Die für eine Tabelle definierte Strategie hat jedoch nur dann Auswirkungen, wenn sie für das eigentliche Widget verfügbar ist. Deshalb reicht die Interaktionsform dieses Objekt an die Präsentationsform weiter. Da der Umgang des Benutzers mit den einzelnen Zellen innerhalb der Tabelle ebenfalls über die lose Koppelung zwischen Interaktionsformen und einer Präsentationsform realisiert werden soll, müssen wir hierbei darauf achten, keinen konzeptuellen Bruch zu begehen. Desweiteren kennt das Tabellenwidget nur die Präsentationsformen, die für die Darstellung der Tabelle benötigt werden und keine Interaktionsformen. Auf der Seite des Werkzeuges kennen wir wiederum nur Interaktions-

formen und keine Präsentationsformen. Unsere Aufgabe besteht nun darin, eine Verbindung zwischen Interaktion und Präsentation auf der Ebene der Strategie herzustellen. Zur Lösung dieses Problems haben wir folgenden Ansatz gewählt. Innerhalb der Interaktionsform für eine Tabelle wird ein Strategieadapter erzeugt, der das eigentliche Strategieobjekt einhüllt, bevor es an die `pfTable` weitergegeben wird. Diesen Adapter werden wir im weiteren auch als Präsentationsstrategie bezeichnen. Die Präsentationsform wiederum sorgt unmittelbar nach der Entgegennahme dieses Strategieadapters dafür, daß die an ihr registrierten Paare, bestehend aus Widgetnamen und Klassenpfad, an den Adapter weitergeleitet werden. Innerhalb dieses Adapters werden dann die eigentlichen Widgets erzeugt und verwaltet. Dieser Strategieadapter hat nun die Möglichkeit, über das in ihm gekapselte Strategieobjekt zu bestimmen, welche Präsentationsform und welche Interaktionsformen für welche Spalte einzusetzen sind. Zu diesem Zweck bedient sich der Adapter folgender Methoden des Strategieobjektes:

- `getPFName (id)`
- `getIFs (columnId)`

Die erste Methode dient dazu, zu bestimmen, welche Präsentationsform für eine Spalte, die durch ihren Identifikator eindeutig bezeichnet ist, einzusetzen ist. Als Ergebnis liefert dieser Methodenaufruf einen Klartext für das einzusetzende Widget (zum Beispiel „Frachtkosten"). Der hier zurückgelieferte Name muß mit einem der direkt an der `pfTable` registrierten (siehe 4.2.5) Namen übereinstimmen. Im Unterschied hierzu liefert die Methode `getIFs` ein Array zurück, in dem bereits Objekte der in diesem Fall zu verwendenden Interaktionsformen enthalten sind. Innerhalb des Adapters wird dann nach dem Aufruf dieser beiden Methoden die lose Koppelung der Interaktionsformen mit den Präsentationsformen initiiert. Aufgrund der Tatsache, daß alle für den Aufbau der Verbindung zwischen Interaktionsformen und Präsentationsform benötigten Methoden gleichlautend sind, ist der Adapter in der Lage, diese Koppelung vorzunehmen. Dieses gilt jedoch nicht für das Setzen und Holen der Werte in beziehungsweise aus den einzelnen Zellen mit Hilfe der hinterlegten Interaktionsformen. Um die unterschiedlichen Arten zu kompensieren, mit denen die Werte gesetzt oder aus den Zellen geholt werden, muß ein Strategieobjekt die folgenden Methoden implementieren.

- `customizeCell (ifs[], value, columnId, isSelected)`
- `getCellValue (ifs[], columnId)`

Eine weitere Möglichkeit, die sich durch den Einsatz einer Interaktionsstrategie eröffnet, betrifft die Auswahl der darzustellenden Werte, die in dem zugeordneten Modell enthalten sind. Teilweise kann eine übersichtliche Darstellung einer großen Menge von Werten nur durch das Ausblenden von Teilen dieser Wertemenge erreicht werden. In der Folge bedeutet dies aber auch, daß die Anzahl der auf dem Bildschirm dargestellten Spalten eventuell kleiner ist, als diejenige Anzahl an Spalten, die tatsächlich in dem Modell, welches für die Vorhaltung der Werte zuständig ist, vorhanden sind. Mit Hilfe der Interaktionsstrategie wird die Abbildung der auf dem Bildschirm dargestellten Spalten auf die „Spalten" des Datenmodells definiert. Diese Abbildung wird alleine dadurch notwendig, daß durch einfaches Vertauschen der Reihenfolge von Spalten durch den Benutzer nicht mehr von der Position der Spalten auf dem Bildschirm auf die Position innerhalb des Modells geschlossen werden kann.

Die Interaktionsstrategie bietet aus diesem Grund zwei komplementäre Methoden an, mit denen diese Abbildung definiert wird.

- `getColumnId (pos)`
- `getDataModelColumn (id)`

Über die der Präsentationsform bekannte Position der Spalte kann der eindeutige Identifikator abgefragt werden. Erst über diesen Identifikator ist es möglich, die Position der zu benutzenden Werte innerhalb des Modells zu bestimmen. Die Konsequenz dieses Entwurfes besteht letztendlich darin, daß nicht nur der Umgang des Benutzers mit einer Tabelle variabel gestaltet wird, sondern auch die für die Zuordnung der Spalten zu den Elementen des Modells notwendige Logik individuell bestimmbar ist.

5.2.5 Interaktionsform

Werkzeuge, die mit Hilfe des JWAM Rahmenwerkes entwickelt werden, binden die Widgets der Oberfläche über Interaktions- und Präsentationsformen an die Interaktionskomponente an. Und da eines unserer Ziele bezüglich der Konstruktion einer Tabelle darin besteht, diese Tabelle in dieses Rahmenwerk zu integrieren, werden wir an dieser Stelle die hierfür benötigte Interaktionsform vorstellen.

Interaktionsformen zeichnen sich insgesamt dadurch aus, daß sie unabhängig von der Darstellung auf der Oberfläche über einen Umgang eines Benutzers abstrahieren. So existieren zum Beispiel Interaktionsformen, welche die Eingabe einer Zeichenfolge oder eines ganzzahligen Wertes gegenüber einem Werkzeug vertreten. Eine direkte Bindung zwischen der Darstellung und der intendierten Benutzung erfolgt nicht. Die für Tabellen benötigte Interaktionsform unterscheidet sich nun von den bisher im Rahmenwerk vorhandenen. Begründet liegt dies darin, daß die Benutzung einer Tabelle in direkter Abhängigkeit von den in ihr enthaltenen Werten zu sehen ist (siehe auch 4.1). Abstrahieren wir nun über die Benutzung einer Tabelle, so stellen wir fest, daß eine Gemeinsamkeit von Tabellen, unabhängig von deren Verwendungskontext, darin besteht, daß die einzelnen Zellen mit Werten versehen werden. Allerdings müssen nicht alle Zellen einen Wert enthalten. Ebenso müssen mögliche Abhängigkeiten zwischen unterschiedlichen Zellen einer Tabelle berücksichtigt werden. Bezogen auf die vorgenannten Beobachtungen bedeutet dieses für uns, daß für eine Tabelle der Umgang über eine Interaktionsform `ifFillIn` realisiert werden kann. Die neue Interaktionsform für Tabellen werden wir vorerst `ifFillInTable` nennen.

Da innerhalb von Tabellen die Zusammenhänge zwischen einzelnen Zellen einer Zeile oder eines Bereiches eine besondere Bedeutung haben, spielt die Selektion in diesem Fall eine besondere Rolle. Häufig werden die für eine Tabelle definierten Funktionen auf einen durch den Benutzer selektierten Bereich angewendet. Die Möglichkeit der Selektion von Zellen innerhalb der Tabelle stellt allerdings einen Umgang dar, der über die Möglichkeiten einer reinen Interaktionsform `ifFillIn` hinaus geht. Die Frage, die in diesem Zusammenhang von uns geklärt werden muß ist, ob die Selektion eventuell eine eigene Interaktionsform darstellt oder zum Leistungsumfang der neu geschaffenen Interaktionsform für Tabellen gehört.

Sofern man sich für die Erstellung einer eigenen Interaktionsform entscheidet, ist es hilfreich, die den Wert enthaltenden Felder als eine zweidimensionale Matrix anzusehen. Textfelder zum Beispiel können so als einfache Matrizen mit genau einer Zeile und genau einer Spalte angesehen werden. Bei Selektionen, die durch den Benutzer ausgelöst

wurden, erhält man dann zum einen die Koordinate des Feldes in dem eine Selektion vorgenommen wurde und zum anderen den tatsächlich innerhalb des Feldes ausgewählten Bereich. Somit ist man unabhängig von dem verwendeten Widget in der Lage, auch Selektionen für die Werkzeuge verfügbar zu machen.

Für die hier vorgestellte Konstruktion einer Tabelle beziehen wir eine andere Position im Hinblick auf die Selektion von Zellen. Es scheint sich für uns so zu verhalten, daß die Art und Weise der Selektion ein echtes Spezifika für dieses Widget darstellt. In den meisten gängigen Widgets wird die Selektion als Editierhilfe benutzt und hat im weiteren keinerlei fachliche Bedeutung. Ausnahmen stellen in diesem Sinne Widgets dar, bei denen zum Beispiel aus einer Liste ein oder mehrere Elemente gewählt werden können. Hier wird die Selektion in einer Art und Weise eingesetzt, die fachliche Bedeutung für die zuständigen Werkzeuge besitzen und aus diesem Grund auch über Änderungen der Selektion informiert werden. Für uns geht die Selektion in einer Tabelle ebenfalls über eine Editierhilfe hinaus. Aus diesem Grund haben wir uns dazu entschlossen, die Selektion innerhalb der Tabelle zu implementieren und nicht als eigene Interaktionsform zu realisieren. Somit geben wir dem Anwendungsentwickler über die Schnittstelle der Interaktionsform die Möglichkeit, die verschiedenen Arten der Selektion einzustellen.

Allgemein unterteilen wir die Einstellung der Selektion in die Art und den Bereich. Mit Hilfe der Art der Selektion bestimmt der Entwickler, ob Selektion generell erlaubt ist und ob jeweils nur eine Auswahl oder mehrere getroffen werden können. Für diese Auswahl haben wir die folgenden Konstanten eingeführt:

- `NO_SELECTION`
  Generell ist in dieser Tabelle keine Selektion möglich.
- `SINGLE_SELECTION`
  Nur eine einzelne Zelle, Zeile oder Spalte kann ausgewählt werden.
- `SINGLE_RANGE_SELECTION`
  Ein einziger zusammenhängender Bereich aus Zellen, Zeilen oder Spalten kann ausgewählt werden.
- `MULTIPLE_RANGE_SELECTION`
  Mehrere zusammenhängende Bereiche aus Zellen, Zeilen oder Spalten können ausgewählt werden.

In Ergänzung hierzu definiert der Bereich die bestimmende Komponente der Selektion. Im einzelnen können dies Spalten, Zeilen und Zellen sein. Um diesen Bereich genau fassen zu können, sind von uns analog zu der Selektionsart Konstanten eingeführt worden:

- `CELL_SELECTION`
- `ROW_SELECTION`
- `COLUMN_SELECTION`

Über die Kombination der Methodenaufrufe, die diese Einstellungen realisieren, ist man in der Lage, die gewünschte Form der Selektion zu bestimmen. In Abhängigkeit von der fachlichen Bedeutung der Selektion im Anwendungskontext kann die Form der Selektion zur Laufzeit variieren und ist aus diesem Grund jederzeit durch die Interaktionskomponente veränderbar.

Über die von uns erstellte Interaktionsform `ifFillInTable` ergeben sich für einen Anwendungsentwickler keine großen Unterschiede im Vergleich zu den bereits vorhan-

denen Interaktionsformen (zum Beispiel `ifFillInText`) bei der Anbindung eines Tabellen Widgets an ein Werkzeug. Im Zuge der Erzeugung der Interaktionsform zur Laufzeit, bekommt diese über ihren zugehörigen Kontext die passende Präsentationsform geliefert, so daß die lose Koppelung der Interaktionsform mit der Präsentationsform sichergestellt werden kann. Die Interaktionskomponente kann sich dann bei der Interaktionsform für die sie interessierenden Commands anmelden. Zu diesem Zweck werden drei verschiedene Commands zur Verfügung gestellt:

- `TableChange`
  Nachdem der `TableRowDataChange` Request von der Interaktionskomponente akzeptiert worden ist, wird die Änderung des Modells veranlaßt. Sowie diese Änderung erfolgreich durchgeführt worden ist, wird der Interaktionskomponente dieses über dieses Command angezeigt.
- `TableSelectionChange`
  Dieses Command wird versendet, sofern sich die Selektion innerhalb der Tabelle geändert hat. Insbesondere für den Fall, daß eine Selektion Auswirkungen auf andere Teile der Oberfläche hat, wird dieses Command eingesetzt.
- `TableLayoutChange`
  Der Benutzer ist in der Lage, auf das Layout einer Tabelle Einfluß zu nehmen (zum Beispiel das Vertauschen von Spalten). Sofern die Präsentation einer Tabelle eine fachliche Bedeutung für das Werkzeug besitzt, besteht die Möglichkeit, mit Hilfe dieses Commands über die Änderungen informiert zu werden.

Zusätzlich wurde der folgende Request realisiert:

- `TableRowDataChange`
  Dieser Request wird an die Interaktionskomponente gestellt, bevor der Inhalt des Modells verändert wird. Somit kann zum Beispiel eine Konsistenzprüfung durch die Interaktionskomponente vor der Änderung durch die Präsentationsform erfolgen.

Die ersten Schritte bei der Verwendung unserer Interaktionsform unterscheiden sich also nicht von den für die bereits bekannten Interaktionsformen benötigten:

- Erzeugen der Interaktionsform durch die Interaktionskomponente
- Anbindung der Interaktionsform an die Präsentationsform (lose Koppelung)
- Interaktionskomponente meldet sich für die sie interessierenden Commands an

Was zu diesem Zeitpunkt noch aussteht, ist das Setzen der Werte. Bisher waren wir es gewohnt, den gewünschten Wert direkt über die entsprechende `set` - Methode der Interaktionsform zu setzen. Grundsätzlich möchten wir diesen Weg auch beibehalten, mit dem Unterschied, daß in dem Fall der Tabelle ein Modell übergeben wird, in dem eine Menge von Werten enthalten sein kann. Um also den Inhalt einer Tabelle festzulegen, übergibt die Interaktionskomponente ein `tableDataModel` (siehe Abschnitt 5.2.2) an die entsprechende Interaktionsform `ifFillInTable`. Der innere Aufbau des für die Tabelle einzusetzenden Modells erfolgt unserer Ansicht nach immer in Abhängigkeit von dem Einsatz der Tabelle. Handelt es sich bei dem Einsatz um ein gestalterisches Hilfsmittel, bei dem Werte aus einem oder mehreren Materialien dargestellt werden sollen, so besitzt die Interaktionskomponente Wissen darüber, wie die Tabelle aufgebaut werden soll. Demzufolge ist sie in der Lage, ein entsprechendes Modell mit den darzustellenden Werten zu versorgen. Im Fall der Anbindung von Datenbanken sollte unserer Meinung nach die Funktionskomponente für den Aufbau des Modells zuständig

sein, und dieses dann über die Interaktionskomponente der `ifFillInTable` bekannt machen. Mit dieser Art der Konstruktion können wir Kommunikation zwischen der Funktions- und der Interaktionskomponente vereinfachen. Es müssen also nicht alle Werte einzeln an die Interaktionskomponente durchgereicht werden, sondern es wird nur das entsprechende Modell weitergereicht.

Wird ein Wert, der in einer Tabelle dargestellt wird, innerhalb der Funktionskomponente verändert, so wird die entsprechende Interaktionsform über dieses Ereignis informiert, so daß sie in der Lage ist, die Aktualisierung der Tabelle zu veranlassen. Erreicht wird diese Aktualisierung dadurch, daß die Interaktionskomponente das Modell an der Interaktionsform `ifFillInTable` neu setzt. Verändert ein Benutzer den Inhalt der Tabelle, so wird die Interaktionskomponente über ein Request von der `ifFillInTable` benachrichtigt. In diesem Request ist das Tupel enthalten, welches durch den Benutzer verändert wurde. Die Überprüfung des beziehungsweise der Werte und die Veränderung des der Tabelle zugrunde liegenden Modells fällt in den Aufgabenbereich der Interaktions- und der Funktionskomponente.

5.2.6 Präsentationsform

In den vorigen Abschnitten haben wir diejenigen Elemente der Konstruktion vorgestellt, zu denen eine Interaktionskomponente eine direkte Beziehung aufbaut. Insgesamt betrifft dies die beiden unterschiedlichen Modelle für die Vorhaltung von Werten und Informationen bezüglich des Tabellenlayouts, die Interaktionsform für Tabellen und die Interaktionsstrategie. Allen vorgenannten Elementen gemeinsam ist die Tatsache, daß ihre Erzeugung zur Laufzeit Aufgabe der Interaktionskomponente ist. Sowohl die Modelle als auch die Interaktionsstrategien werden dann über den Aufruf der entsprechenden Methoden der Interaktionsform `ifFillInTable` gesetzt. Bis zu diesem Zeitpunkt haben wir allerdings die Tabelle noch nicht auf einem Grafikkontext dargestellt. Innerhalb der Werkzeugkonstruktion entfällt diese Aufgabe auf die Präsentationsformen. Im folgenden werden wir nun die Präsentationsform für Tabellen – `pfTable` – vorstellen.

Ausgangspunkt für die Erstellung einer solchen Präsentationsform ist immer die Definition eines sogenannten `pfInterface`. Innerhalb dieses Interfaces wird die Methodenschnittstelle definiert, mit deren Hilfe die Interaktionsform auf die Präsentationsform zugreifen kann. Die Implementierung dieser Schnittstelle erfolgt in einer Klasse, die entweder ein Widget aus einem bestehenden Toolkit kapselt oder das benötigte Widget selbst definiert. In unserem Fall haben wir uns mit Blick auf das JWAM Rahmenwerk dazu entschlossen, das in das SWING Toolkit integrierte `JTable` Widget zu kapseln.

Grundsätzlich liegt die Problematik bei der Darstellung einer Tabelle darin, den möglicherweise unterschiedlichen Typen der Spalten gerecht zu werden. Die Typisierung der Spalten kann nicht nur Auswirkungen auf die Interaktion des Benutzers haben, sondern kann auch die Art der Darstellung beeinflussen. Zusätzlich besteht auch häufig die Möglichkeit, unterschiedliche Darstellungen für einen Wert bei gleichbleibender Interaktion anzubieten. Um dieser Typisierung auch in der Darstellung gerecht werden zu können, benutzt die `pfTable` auf den jeweiligen Typ passende Präsentationsformen. Diese innerhalb der `pfTable` verwendeten Präsentationsformen sind nach außen nicht sichtbar oder bekannt. Das Wissen darüber, welche Präsentationsform für welche Zelle verwendet werden soll, kann unserer Meinung nicht in der `pfTable` vorliegen. Dieses Wissen sollte vielmehr in der für eine spezielle Tabelle zu verwendenden Interaktionsstrategie liegen. Somit bleibt nur zu klären, wie der Tabelle insgesamt die Namen der

Präsentationsformen bekannt gemacht werden, die zur Laufzeit Verwendung finden können. Letztlich sind wir zu dem Entschluß gekommen, zum Zeitpunkt der Layouterstellung in den Eigenschaften der Tabelle die Bestimmung der zugrunde liegenden Präsentationsformen vorzunehmen. Aus diesem Grund haben wir innerhalb der Präsentationsform `pfTable` eine Methode `registerColumnPF` vorgesehen, die es erlaubt, ein Paar bestehend aus einem Namen (Beispiel: „Betrag") und einem Klassenpfad (Beispiel: "de...swingimplementation. pfJIntegerField") in der Tabelle zu registrieren. Die `pfTable` verwaltet die für eine Tabelle registrierten Paare. Die Erzeugung der eigentlichen Widgets erfolgt allerdings erst in der Präsentationsstrategie anhand des spezifizierten Klassenpfades.

Auf der technischen Ebene existieren Unterschiede in der Benutzung von Widgets, die Auswirkungen auf die Anzahl der benötigten Objekte haben. Auf der einen Seite werden sie zur reinen Darstellung des jeweiligen Wertes benutzt und auf der anderen Seite werden sie zur Bearbeitung des dargestellten Wertes benötigt. Ursächlich hierfür ist anzusehen, daß die Darstellung eines Wertes zum Zeitpunkt der Bearbeitung anderen Ansprüchen gerecht werden muß. Als Beispiel hierfür kann die Eingabe eines Datums gelten. Für die Anzeige eines Datums wird das landesübliche Format – 12.08.1999 – verwendet. Sowie ein Benutzer aber in dieses Feld klickt, wird ein Kalenderblatt eingeblendet, welches dem Benutzer die Auswahl eines Datums ohne den Einsatz einer Tastatur ermöglicht. Die meisten GUI - Toolkits reagieren auf diese Anforderung in dem sie für ein Element der Oberfläche in sogenannte Render- und Editor - Widgets unterscheiden. In gängigen Oberflächen kann man aus diesem Grund zu einem Oberflächenelement zwei verschiedene Objekte finden, welche die Darstellung und die Bearbeitung unter sich aufteilen. Wollte man innerhalb einer Tabelle diese 1:2 (1 Zelle : 2 Objekte) Bindung nachvollziehen, so führte dies unweigerlich zu einer Vielzahl von Objekten, deren Verwaltung alleine die Antwortzeiten beträchtlich beeinflussen würde. Die optimale Lösung bestünde darin, für jede eingetragene Klasse genau ein Objekt zu erzeugen und dieses dann zur Laufzeit dynamisch an die entsprechende „Zelle" zu binden. An dieser Stelle unterliegen wir allerdings einer technischen Restriktion durch SWING. Diese äußert sich darin, daß für die Darstellung und für das Editieren nicht das identische Objekt verwendet werden kann. Letztlich bedeutet dies für unsere konkrete Implementation, daß wir von jeder einzusetzenden Präsentationsform zwei Objekte benötigen. Insgesamt erzeugen wir also für jede an der `pfTable` registrierte Präsentationsform ein Objekt für die Darstellung und ein Objekt für die Bearbeitung der Werte. Zur Laufzeit werden diese Objekte gemäß der aktuellen Anforderung dynamisch an die jeweiligen Zellen gebunden. Die Bindung der Präsentationsformen an die einzelnen Zellen ist hierbei von der Typisierung der die Zellen enthaltenen Spalten abhängig.

Da die `pfTable` die konkreten Methodenschnittstellen der zu verwendenden Präsentationsformen nicht kennt, ist es ihr nicht möglich, direkt an den Präsentationsformen Einstellungen vorzunehmen. Aus diesem Grund müssen wir einen Weg schaffen, auf dem die Präsentationsformen initialisiert und konfiguriert werden können. Insbesondere gehört hierzu das Setzen und das Auslesen der Werte an den konkreten Präsentationsformen. Um dieses Problem lösen zu können, greifen wir auf die im Abschnitt 5.2.4 beschriebene Präsentationsstrategie zurück. Die `pfTable` stellt dieser Strategie die einzustellende Präsentationsform zusammen mit dem einzutragenden Wert zur Verfügung. Die weitere Konfiguration ist dann die Aufgabe der Präsentationsstrategie.

5.2.7 Interaktionsdiagramme und Zusammenspiel der Komponenten

Da bei der Beschreibung der einzelnen Bestandteile einer Konstruktion das Zusammenwirken dieser nur teilweise erläutert werden kann, möchten wir diesen Aspekt an dieser Stelle vertiefend aufgreifen. Zuerst werden wir auf die Zusammenhänge und Benutzt-Beziehungen innerhalb unserer Tabellenkonstruktion eingehen und dieses mit Hilfe von Klassendiagrammen illustrieren. Im Anschluß hieran folgt die Beschreibung der dynamischen Abläufe innerhalb unserer Tabellenkonstruktion. Hierzu werden wir exemplarisch den Vorgang des Darstellens einer Zelle und der Änderung eines Wertes innerhalb einer Zelle mit Hilfe von Interaktionsdiagrammen erläutern.

Eine Vorbedingung für die Anbindung eines Tabellenwidgets an ein Werkzeug stellt das Vorhandensein eines zueinander passenden Paares aus Interaktionsform und Präsentationsform dar. Diese Bedingung stellt für einen mit dem JWAM Rahmenwerk befaßten Anwendungsentwickler keine Neuigkeit dar, sondern steht vielmehr für das der Anbindung von grafischen Benutzungsschnittstellen zugrunde liegende Konzept. Eine Neuerung dagegen stellt die Anforderung an den Entwickler dar, diesem Widget ein Modell zur Verfügung zu stellen, in dem die darzustellenden Werte enthalten sind.

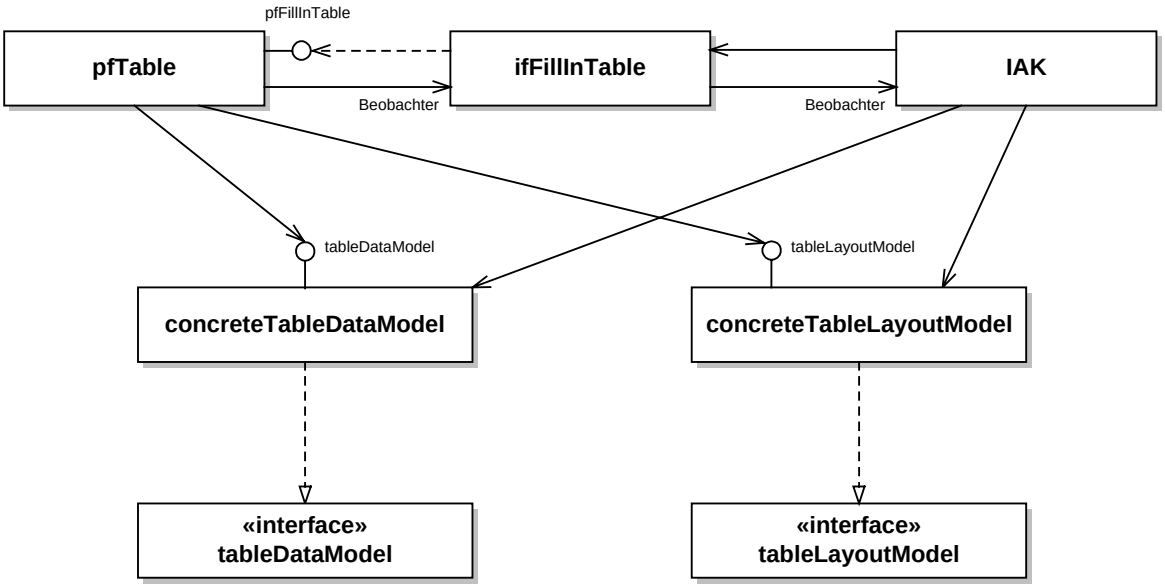

Abbildung 5-5: Einbindung der Modelle

Zur Verfügung stellen bedeutet in diesem Fall, daß er dafür Sorge tragen muß, eine Klasse zu implementieren, die das `tableDataModel` Interface berücksichtigt. Vom technischen Standpunkt aus betrachtet ist die Realisierung sowohl in einer separaten Klasse denkbar, als auch in einer bereits existierenden Klasse des betroffenen Werkzeuges. Hinsichtlich der Übersichtlichkeit der Konstruktion empfehlen wir die Implementierung des Modells in einer separaten Klasse. Parallel dazu hat der Entwickler die Möglichkeit, ein Layout Modell für diese Tabelle zu hinterlegen. Im Unterschied zu dem `tableDataModel` ist die Bereitstellung eines `tableLayoutModel` aber keine Pflicht. Für diesen Fall existiert innerhalb des Widgets ein Standard Modell über das zum Beispiel die Spaltenbreiten eingestellt werden können. Der Einsatz eines eigenen Layout Modells ist in unseren Augen immer dann sinnvoll, wenn die Präsentation der Tabelle innerhalb der Oberfläche zu einem späteren Zeitpunkt restauriert werden muß.

Die Änderungen eines Benutzers an der Präsentation der Tabelle können dann zu diesem Zweck über das `tableLayoutModel` persistent gemacht werden. Aus dem in Abbildung 5-5 dargestellten Klassendiagramm geht vorerst die Verbindung zwischen dem Werkzeug und einer Tabelle hervor.

Die zwischen der Interaktionskomponente und der Interaktionsform, sowie zwischen der `ifFillInTable` und der Präsentationsform etablierten Benutzt- und Benachrichtigt-Beziehungen unterscheiden sich nicht von denen, wie wir sie bei allen Widgets vorfinden, deren Anbindung über das Konzept der Interaktions- und Präsentationsformen erfolgt. Hinzu kommen nun die reinen Benutzt - Beziehungen zu den beiden unterschiedlichen Modelltypen. Zum Zeitpunkt der Initialisierung des Widgets setzt die Interaktionskomponente Referenzen auf die zu verwendenden Modelle an der Interaktionsform. Diese wiederum gibt die Referenzen weiter an das Widget, so daß auch die `pfTable` über die dafür vorgesehenen Schnittstellen auf die Modelle zugreifen kann. Änderungen an Eintragungen in der Tabelle werden der Interaktionskomponente über ein Request mitgeteilt. Die Entscheidung, ob diese Änderung letztlich durchgeführt werden kann und soll, liegt auf der Seite des Werkzeuges. Ein weiterer Punkt, der aus der Abbildung 5-5 deutlich wird, ist der, daß die Kommunikation des Widgets mit dem Werkzeug einzig über die Interaktionsform abgewickelt wird.

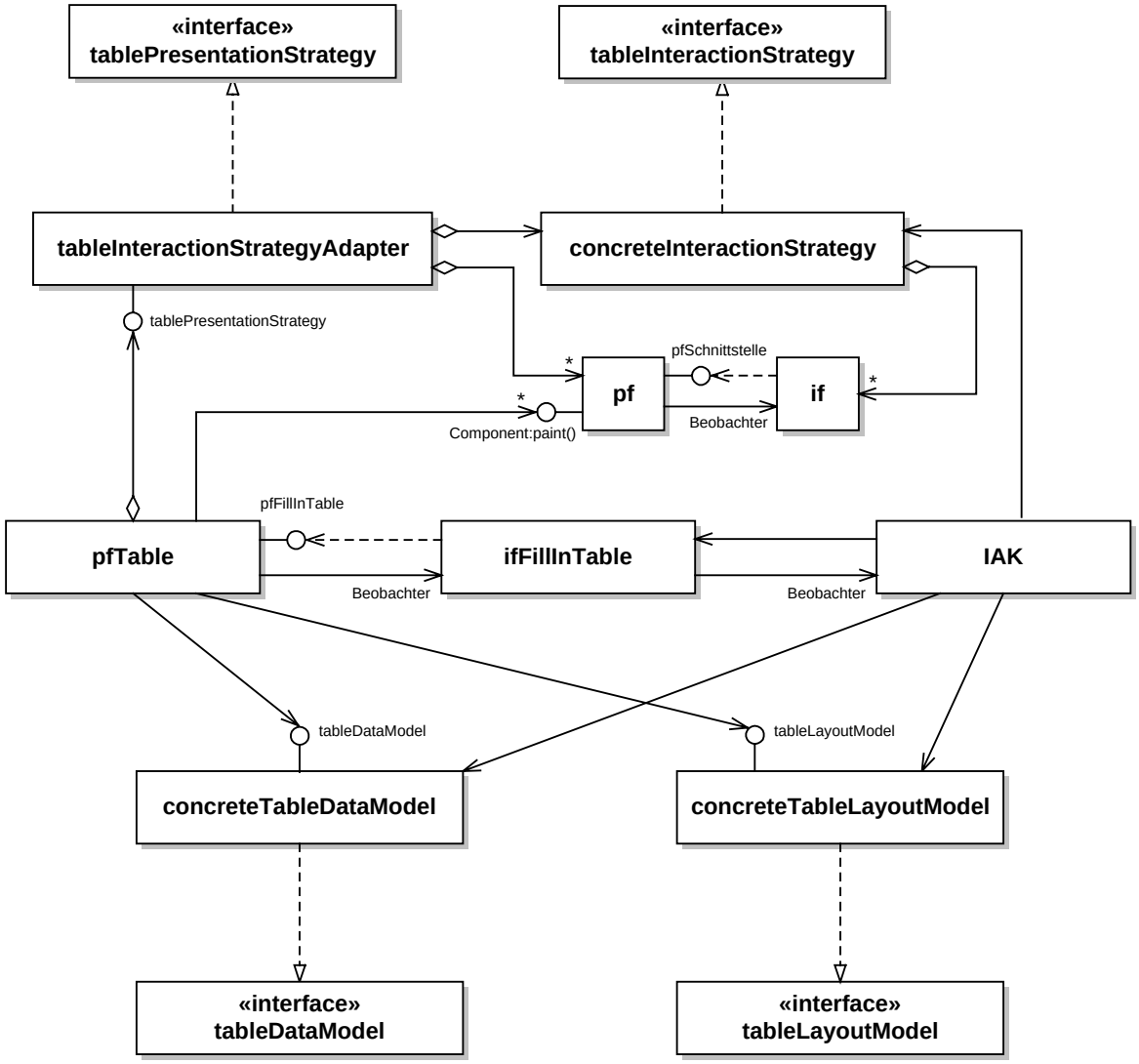

Abbildung 5-6: Tabellenkonstruktion

Die Konstruktion wird komplettiert durch die Interaktions- und Präsentationsstrategie, mit Hilfe derer der Umgang mit einer speziellen Tabelle definiert wird. Die Einbindung dieser beiden Strategien ist in der Abbildung 5-6 dargestellt. Die Schnittstellendefinition beider Strategien erfolgt in unserer Konstruktion über ein eigenes Interface. Wo hingegen aber die Interaktionsstrategie von dem Anwendungsentwickler in Abhängigkeit von der intendierten Verwendung der Tabelle gestaltet werden muß, existiert für die Präsentationsstrategie bereits eine Implementation, die in der Lage ist, dynamisch die Verbindungen zwischen den für den Umgang mit einer Zelle benötigten Interaktionsformen und der zugeordneten Präsentationsform herzustellen. Aus der Abbildung 5-6 ist auch zu ersehen, wie die Aufgaben hinsichtlich der Verwaltung der zu benutzenden Präsentations- und Interaktionsformen verteilt sind. Der von uns bereits implementierte `tableInteractionStrategyAdapter` ist in der Lage eine beliebige Anzahl von Präsentationsformen zu verwalten. Dieser Adapter aggregiert wiederum die durch einen Anwendungsentwickler über die Interaktionsform zur Verfügung gestellte `concreteInteractionStrategy`. Die Aufgabe des Adapters besteht nun, wie bereits in Abschnitt 5.2.4 erwähnt, darin, der Präsentationsform ein fertig initialisiertes Widget zurückzugeben, welches für die Darstellung oder das Editieren verwendet werden soll.

Um den konkreten Ablauf innerhalb unserer Konstruktion zur Laufzeit zu verdeutlichen, möchten wir nun exemplarisch den Kontrollfluß für einen Paint- und einen Edit - Vorgang anhand von Interaktionsdiagrammen darstellen. Ausgangspunkt für die Betrachtung dieser Vorgänge ist immer die Präsentationsform `pfTable`, da diese die Kapsel um das spezielle Tabellen - Widget eines Toolkits, in unserem Fall die `JTable` aus dem SWING-Rahmenwerk, bildet. Wir werden diesen Teil mit der Folge an Methodenaufrufen beginnen, die notwendig sind, um eine Zelle auf einem Grafikkontext darstellen zu können. Zur näheren Illustration ist diese Abfolge in Abbildung 5-7 dargestellt.

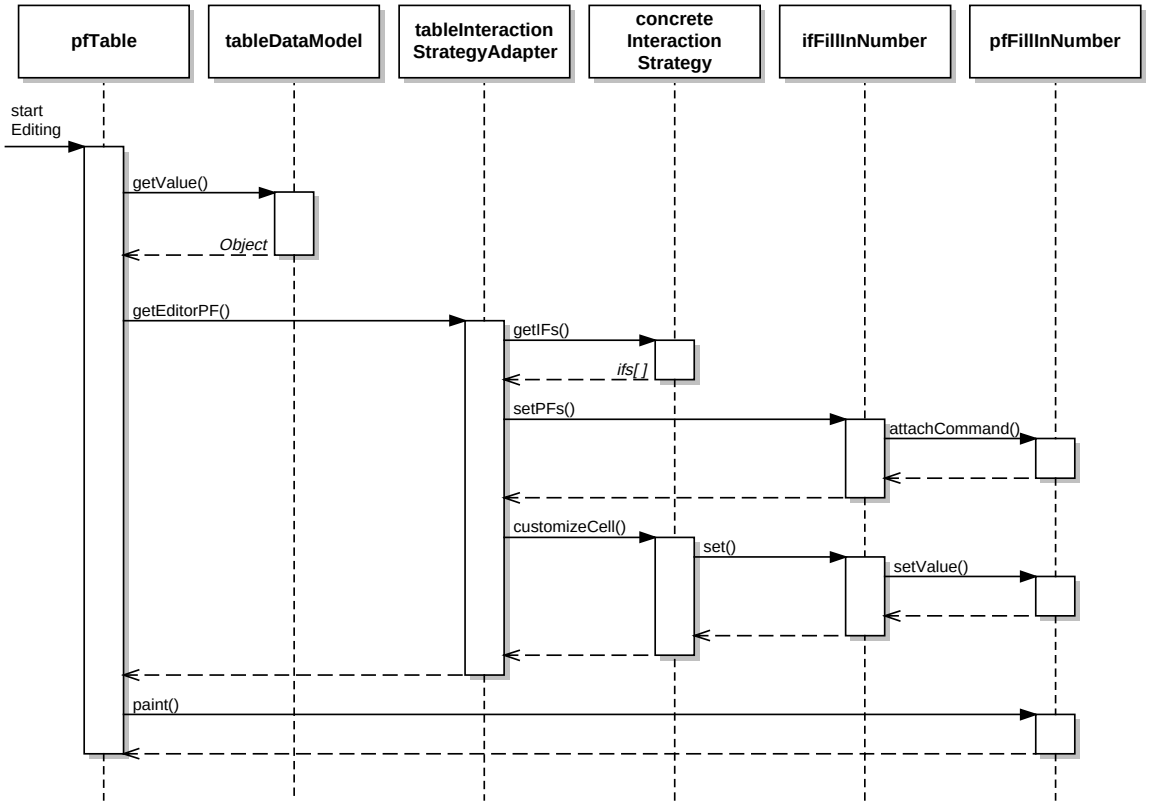

Abbildung 5-7: Darstellen einer Zelle

Sowie die `pfTable` die Aufforderung bekommt, eine Zelle darzustellen (paint), fordert sie über den `tableInteractionStrategyAdapter` eine fertig eingestellte Widget Komponente an, die sie dann auf ihrem Grafikkontext darstellen kann. Der `tableInteractionStrategyAdapter` seinerseits holt sich aus der `concrete-InteractionStrategy`, die der Anwendungsentwickler an der `ifFillInTable` gesetzt hat, die Liste der für diese Zelle zu verwendenden Interaktionsformen. Zusätzlich fragt der Adapter an der Strategie, den Namen der für diese Zelle zu verwendenden Präsentationsform ab, anhand dessen der Adapter das eigentliche Objekt identifizieren kann. Danach wird die Präsentationsform mit der Interaktionsform, in diesem Fall eine `ifFillInNumber`, verbunden. Nachdem diese Verbindung hergestellt wurde, können an der Präsentationsform über die entsprechende Interaktionsform die notwendigen Einstellungen vorgenommen werden. Letztlich gibt der `tableInteraction-StrategyAdapter` die so fertig konfigurierte Präsentationsform an die `pfTable` zurück, die dann die weiteren Schritte der Darstellung direkt mit der Komponente durchführen kann.

Der Bearbeitungsvorgang an einer Zelle gliedert sich in die Abschnitte des `StartEditing` Events und des `StopEditing` Events. Aus diesem Grund haben wir für die Erläuterung eines Editiervorgangs zwei Interaktionsdiagramme verwendet. Abbildung 5-8 repräsentiert die Vorgänge nach Auslösung eines `StartEditing` Events, während in der Abbildung 5-9 auf das `StopEditing` Event Bezug genommen wird.

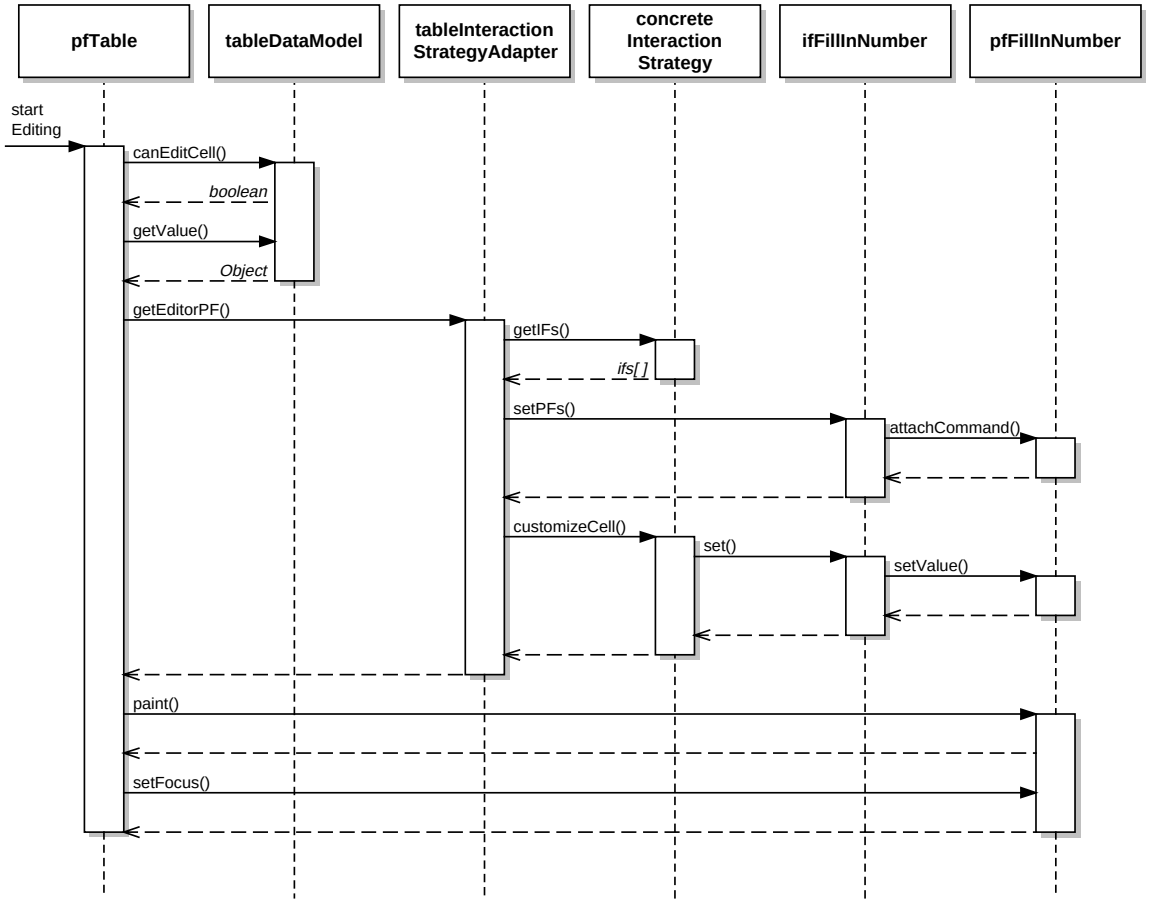

Abbildung 5-8: Beginn eines Editiervorganges

Nachdem das `StartEditing` Event ausgelöst wurde, stellt die `pfTable` mit Hilfe des `tableDataModel` fest, ob für die angefragte Zelle ein Editiervorgang erlaubt ist. Sofern dies der Fall ist, bestimmt sie zum einen den aktuellen Wert der Zelle und zum anderen die für die Darstellung zu verwendende Präsentationsform. Danach fordert die `pfTable` über die Methode `getEditorPF` die Strategie der Tabelle auf, die entsprechenden Einstellungen an der Präsentationsform für die Zelle vorzunehmen. Der nachfolgende Vorgang entspricht dann dem Prinzip mit dem sich auch eine Interaktionskomponente über die Interaktionsformen an die Elemente der Oberfläche anbindet. Analog zur Anzeige einer Zelle erhält die `pfTable` nun eine fertig parametrisierte Präsentationsform, die auf dem Grafikkontext dargestellt werden kann.

Das `StopEditing` Event, welches die Beendigung der Bearbeitung einer Zelle ausdrückt, wird immer dann ausgelöst, wenn die aktuelle Zelle den Focus verliert oder der Benutzer durch die Betätigung der Return - Taste das Ende seiner Bearbeitung bekannt gibt. Sowie dieses Event der `pfTable` bekannt gemacht wird, holt sie sich den aktuellen Wert mit Hilfe des `tableInteractionStrategyAdapters` aus der Präsentationsform. Um dieses zu erreichen, wird der Methodenaufruf bis zur Interaktionsform, die für den Umgang mit dieser Zelle zuständig ist, durchgereicht.

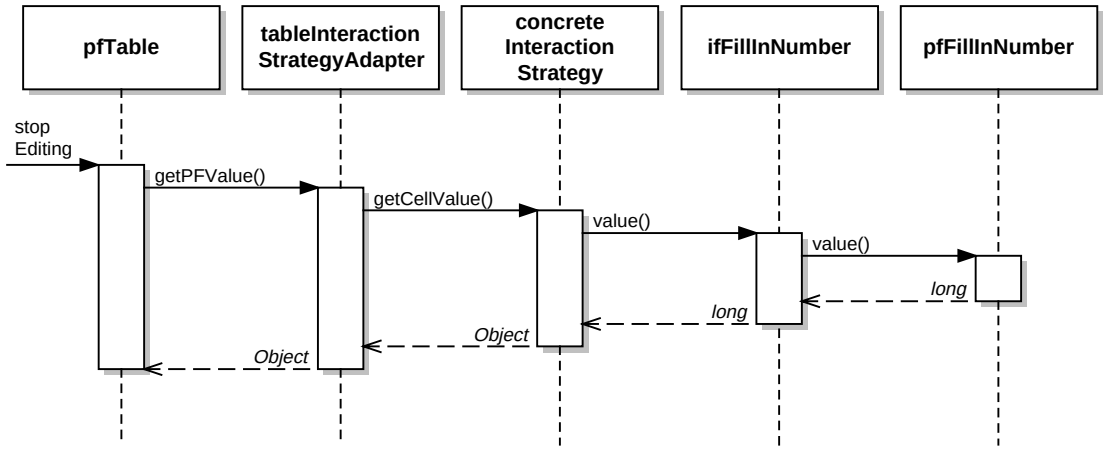

Abbildung 5-9: Ende eines Editiervorganges

5.2.8 JWAM - Integration

Eines unserer Ziele bestand darin, diese Konstruktion in das JWAM - Rahmenwerk einzubinden. In Teilen der vorangegangen Konstruktionsbeschreibung finden sich bereits deutliche Hinweise auf Rahmenbedingungen, die durch diese Einbindung Auswirkungen auf die Konstruktion hatten. Wir möchten an dieser Stelle vertiefen, welche Bestandteile der Konstruktion unabhängig von einem gewählten Toolkit sind und an welchen Stellen starke Abhängigkeiten bestehen.

Ausgehend von der Interaktionskomponente finden wir zunächst die Interaktionsform `ifFillInTable`. Die Aufgabe von Interaktionsformen allgemein besteht darin, ein spezielles Widget gegenüber dem Werkzeug nur über eine entsprechende Umgangsform bekannt zu machen. Insbesondere besitzt selbst die Interaktionsform nur die Möglichkeit, auf ihr bekannte Commands, die von der Präsentationsform an sie gesandt werden, zu reagieren. Unbenommen hiervon besitzt die Interaktionsform die Möglichkeit, mit Hilfe der ihr bekannten Schnittstelle des `pfInterfaces`, Werte an der Präsentationsform zu

setzen oder auszulesen. Hieraus folgt unmittelbar, daß in der Interaktionsform kein Wissen über das letztlich verwendete Toolkit existiert. Wollte man also innerhalb des Rahmenwerkes auf ein anderes GUI - Toolkit umstellen, so würde dies keine Änderung an der Interaktionsform nach sich ziehen. Gleiches gilt auch für die beiden von uns benutzten Modelltypen. Im Hinblick auf die Modelle ist immer zu beachten, daß die benötigte Methodenschnittstelle über Interfaces definiert wird. Der Anwendungsentwickler kann und sollte die Implementierung dieser Modellinterfaces unabhängig von dem verwendeten GUI - Toolkit vornehmen. In Analogie zu den eben angesprochenen Modellen erfolgt die Implementierung der Interaktionsstrategie ebenfalls durch den Anwendungsentwickler, wobei die Definition der Schnittstelle von uns über ein entsprechendes Interface vorgenommen wurde.

Diejenigen Teile der Konstruktion, die in einer starken Abhängigkeit zu dem verwendeten GUI Toolkit stehen, sind naturgemäß die Präsentationsformen. In der nachstehenden Grafik ist ein Ausschnitt der von uns gewählten Konstruktion zu sehen. Hinter den grau hinterlegten Elementen verbergen sich diejenigen Klassen, die in Abhängigkeit von dem gewählten Toolkit, in unserem speziellen Fall SWING, stehen. Zuerst einmal ist hier die `pfTable` selbst zu benennen. Diese Klasse erbt von der Klasse `JTable`, ein SWING Widget, und implementiert das `pfFillInTable` Interface.

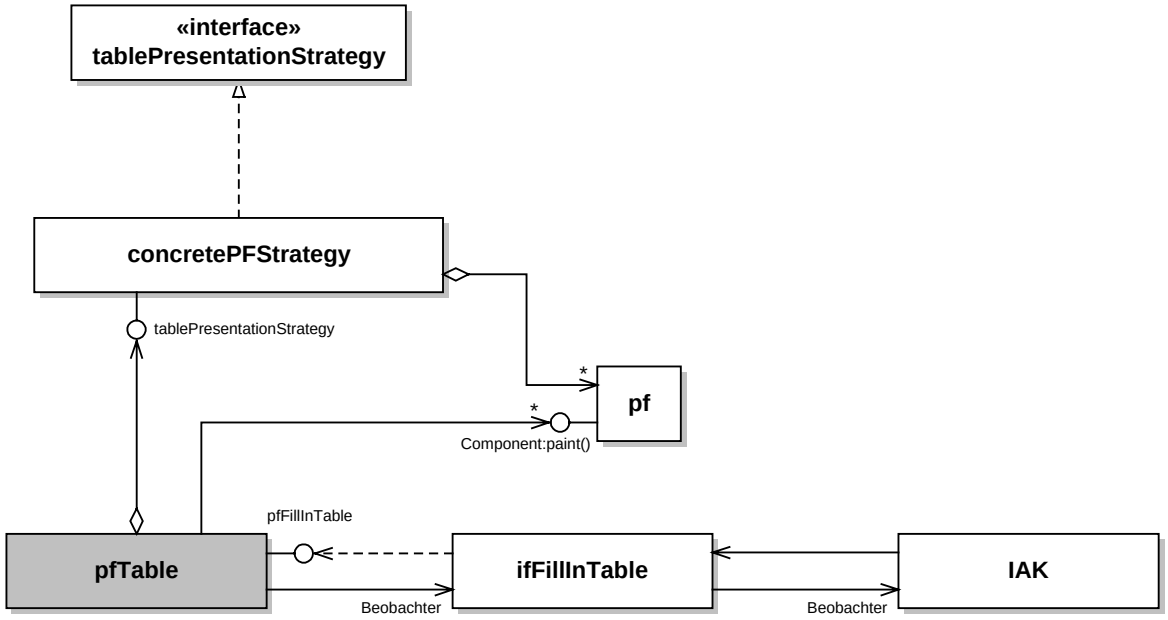

Abbildung 5-10: Abhängigkeiten der Konstruktion

Hinsichtlich des Interfaces können wir davon ausgehen, daß solange keine Änderungen an der Definition der benötigten Methoden vorgenommen werden, auch keine Änderungen an der entsprechenden Interaktionsform notwendig sind. Auf der Seite des SWING Toolkits können wir aber von solch einer Stabilität noch nicht sprechen. Unsere Erfahrungen gehen vielmehr dahin, daß die Schnittstellendefinitionen sehr häufig verändert werden. Für uns folgt daraus unmittelbar, daß Änderungen an der Präsentationsform nicht nur bei einem Wechsel des Toolkits notwendig werden, sondern auch bei der Verwendung einer neuen SWING Version.

Unsere Entscheidung, die `JTable` zu verwenden hatte für uns auch zur Folge, die Konstruktion dieses Widgets selbst zu berücksichtigen. Da wir grundsätzlich den Umgang

mit dem `tableDataModel` und dem `tableLayoutModel` unabhängig von dem konkreten Widget aufgebaut haben, sind die innerhalb der `JTable` verwendeten Modelle nicht mit den von uns konzipierten Modellen kompatibel. Um dieses Problem zu lösen, haben wir einen zusätzlichen Adapter benötigt, der nur durch die `pfTable` verwendet wird. Dieser Adapter ermöglicht der `JTable` den Zugriff auf unser `tableDataModel`. Des weiteren verlangt die `JTable` nach einem sogenannten `DefaultCellRenderer` und einem `DefaultCellEditor`. Die Aufgabe dieser beiden Objekte besteht darin, eine auf einem Grafikkontext darstellbare Komponente zurück zu liefern. In der von uns vorgeschlagenen Konstruktion entscheidet die Präsentationsstrategie zusammen mit der Interaktionsstrategie über das für eine Zelle konkret einzusetzende Widget. Die Präsentationsstrategie basiert im Hinblick auf die Art der Verbindung von Interaktionsformen und Präsentationsformen auf den Konzepten, die innerhalb des JWAM Kontextes auch für andere Widgets eingesetzt werden. Die direkte Einbindung unserer Strategie in die `JTable` hätte der Implementierung eines SWING Interfaces bedurft. Um an dieser Stelle eine Vermischung der durchaus unterschiedlichen Konzepte zu vermeiden, haben wir uns erneut eines Adapters bedient, mit dessen Hilfe wir unsere Präsentationsstrategie sowohl als `DefaultCellRenderer`, als auch als `DefaultCellEditor` verwenden können.

5.2.9 Einbettung in GUI - Builder

Eines der Ziele bei der Anbindung von Oberflächen an Werkzeuge nach WAM ist die Möglichkeit, GUI - Builder zur einfacheren und schnellen Erzeugung von Werkzeugoberflächen verwenden zu können. Im folgenden werden wir kurz skizzieren, welche Möglichkeiten im JWAM - Framework bestehen, die von GUI - Buildern erstellten Oberflächen in eigene Anwendungen einzubinden und welche Auswirkungen dies auf die Widgetkonstruktion hat.

Im Abschnitt 4.3 haben wir bereits gezeigt, daß die Präsentationsformen im JWAM-Framework über einen GUI - Kontext Objekt verwaltet werden. Es wurde bisher jedoch noch nicht deutlich gemacht, wie die Oberflächen aus GUI - Buildern zum GUI - Kontext gelangen. Folgende Möglichkeiten bieten sich an:

- Einlesen der vom GUI - Builder erzeugten Ressourcen - Datei.
- Einlesen von persistent gespeicherten Widgets beziehungsweise Oberflächen.
- Einbinden von automatisch generiertem Quellcode.

In den nachfolgenden Abschnitten werden wie einzelnen Möglichkeiten näher erläutern.

5.2.9.1 Ressourcen - Dateien

GUI - Builder bieten zumeist die Möglichkeit, die von Ihnen erzeugten GUI - Oberflächen in Form von Ressourcen - Dateien zu speichern. Eine Ressourcen - Datei ist eine textbasierte Beschreibung der Oberfläche mit Ihren Widgets. Je nach Format der Ressourcen - Datei folgt diese Datei einer festgelegten Grammatik, die genau vorgibt, wie die Bestandteile und Eigenschaften einer Oberflächen mit Textsymbolen beschrieben werden. Die mit einem GUI - Builder oder auch von Hand erzeugten Ressourcen - Dateien können vom GUI - Kontext eingelesen werden. Dies setzt aber voraus, daß im GUI - Kontext ein Interpreter für die Grammatik der Ressourcen - Datei implementiert ist. Dies ist der größte Nachteil dieses Ansatzes, denn Interpreter für Grammatiken sind in der Regel sehr aufwendig zu implementieren und damit auch sehr fehleranfällig. Da sich bis

heute im Java - Umfeld noch keine einheitliches Format für eine Ressourcen - Datei durchgesetzt hat und jeder GUI - Builder sein eigenes Format mit sich bringt, erscheint dieser Ansatz nicht sehr empfehlenswert.

Die Einbettung von selbsterstellten Widgets in die GUI - Builder ist problematisch, da ein GUI - Builder den Umgang und die Schnittstelle von selbsterstellten Widgets nicht kennt. In interpretierten Sprachen wie Smalltalk und Java gibt es jedoch die Möglichkeit, daß die Klassen eines neuen Widgets durch den GUI - Builder inspiziert werden können. Durch eine Standardisierung der Schnittstelle von Widgets und die Verwendung des Java Komponentenmodells JavaBeans kann erreicht werden, daß ein GUI - Builder mit beliebigen ihm unbekannten Widgets umgehen kann. Da alle AWT- und Swing - Widgets bereits als JavaBeans implementiert sind, und unser Tabellenwidget von der Swing-Klasse `JTable` erbt, kann das Tabellenwidget in allen GUI - Buildern, die JavaBeans unterstützen, verwendet werden.

5.2.9.2 Persistente Oberflächen

Neben der Abspeicherung einer Oberflächen in Form einer Ressourcen - Datei besteht auch die Möglichkeit, die Objekte der Oberfläche direkt in Form von persistenten Objekten zu speichern. Der GUI - Kontext liest dann die Oberflächenobjekte einfach aus einer Datei oder einer Datenbank wieder ein. Dieser Ansatz hat den Vorteil, daß der GUI-Kontext keine Interpretation vornehmen muß und das er keine Widgets selber erzeugen und konfigurieren muß.

Die Anwendung dieses relativ einfachen Verfahrens setzt voraus, daß der GUI - Builder in der Lage ist, Oberflächenobjekte persistent im richtigen Format zu speichern. Da in der aktuellen Java - Version ein sogenannter „Seriallisierungsmechanismus" zum persistenten Speichern von Objekten bereits vorhanden ist und einige GUI - Builder diesen bereits unterstützen, scheint dies ein gangbarer und einfacher Weg zu sein. Aber auch dieses Verfahren stützt sich auf das Komponentenmodell JavaBeans, da der GUI - Builder mit den selbsterstellten Widgets umgehen können muß. Martin Lippert zeigt in [Lipp97a] wie man GUI - Builder, die zwar das Komponentenmodell aber nicht die Seriallisierung unterstützen, um die gewünschte Funktionalität erweitern kann.

5.2.9.3 Einbindung von Quellcode

Die dritte und letzte Möglichkeit, die wir erläutern möchten, ist die Erstellung und Einbindung von Quellcode. Einige GUI - Builder bieten die Möglichkeit, Java - Quellcode zu generieren, der übersetzt und ausgeführt werden kann und so die Oberfläche erzeugt. Der generierte Quellcode sollte jedoch wegen der schon beschriebenen Nachteile nicht direkt in die Interaktionskomponente verwendetet werden. Eine bessere Lösung ist es, eine Subklasse vom GUI - Kontext erben zu lassen und in ihr den Quellcode einzubauen. Diese Lösung hat gegenüber den anderen beiden Ansätzen den Vorteil, daß der Quellcode nachträglich noch per Hand editiert werden kann, auch wenn dies manchmal angesichts des maschinell erzeugtem Code problematisch erscheint. Dieser Vorteil solle nicht unterschätzt werden, da so spezielle Änderungen, die vom GUI - Builder nicht unterstützt werden möglich sind. Quellcode ist außerdem die stabilste Form der Speicherung von Oberflächen. GUI - Builder und Ressourcen - Datei Formate ändern sich sicher häufiger. Ein Nachteil von maschinell generierten Code ist, daß er meistens nicht wieder vom selben oder anderen GUI - Builder eingelesen werden kann. Es sollte daher immer zusätzlich eine Speicherung der Oberfläche in einer Form erfolgen, die ein nachträgliches Einlesen und Bearbeiten der Oberfläche ermöglicht.

5.2.10 Erfüllung der Zielsetzung

Sicherlich kann dieser Abschnitt nur eine subjektive Einschätzung betreffend der Erfüllung der von uns für diese Konstruktion gesetzten Ziele widerspiegeln. Dennoch möchten wir an dieser Stelle die Ziele aufgreifen, die am Ausgangspunkt unserer Konstruktion gestanden haben.

Der schwierigste Punkt bei dieser Betrachtung, besteht darin den Unterschied zwischen diesem Widget und einem Sub - Werkzeug herauszuarbeiten. Schwierig wird diese Aufgabe dann, wenn das betrachtete Widget eine beträchtliche Funktionalität aufweist. Setzt man innerhalb einer Oberfläche ein Textfeld oder eine Listbox ein, so ist unabhängig von dem Verwendungskontext von vorne herein klar, was wir hier zum Beispiel als Wert zu erwarten haben. Bei einer Tabelle sieht das unserer Meinung dadurch anders aus, da man die Verwendung der Tabelle an einen bestimmten Kontext anpassen muß. Es existieren allerdings auch Einsatzfälle, bei denen zum Beispiel nur einfache Zeichenketten angezeigt und editiert werden. Sofern man aber die fachlichen Rahmenbedingungen für ein darzustellendes Material berücksichtigen möchte, erfolgt eine Spezialisierung der Tabelle für einen fest definierten Einsatzkontext. Betrachten wir noch einmal die von uns beschriebene Konstruktion, so findet man innerhalb der Tabelle annähernd eine Konstellation vor, die zumindest an die uns bekannten Werkzeuge erinnert. Die vom Anwendungsentwickler definierte konkrete Interaktionsstrategie repräsentiert in diesem Zusammenhang eine Interaktionskomponente, die mit Hilfe der Interaktionsformen benötigte Verbindungen zu den Präsentationsformen aufbaut. Im weiteren werden dem Widget mit den Modellen Quellen für Werte und Einstellungen benannt. Da diese Modelle nur wissen, wo sie die benötigten Werte bekommen können, ist es unserer Ansicht nach erlaubt, von einem Teil einer funktionalen Komponente zu sprechen. Zusammengenommen finden wir also ein Werkzeug vor, was im Inneren des Widgets verborgen ist. Unbenommen bleiben hierbei die Möglichkeiten, zur Laufzeit die beteiligten Objekte wie Strategie und Modell auszutauschen. Eine ähnliche Diskussion ist bereits in unserer Studienarbeit [SteLam97] aufgetaucht, bei der es darum ging, ein Widget für die Bearbeitung von Netzen in eine Smalltalk Entwicklungsumgebung einzubetten. Es ist auf der anderen Seite recht einfach, gegen ein Sub - Werkzeug zu argumentieren. Zum einen werden nämlich innerhalb des eigentlichen Widgets nur Aufgaben übernommen, die einen Bezug zum Grafik Kontext haben. Der Anwendungsentwickler auf der anderen Seite ist in der Lage, eine Tabelle entsprechend der intendierten Benutzung zu parametrisieren. Die Tabelle als solche ist damit wiederum kontextunabhängig und beschränkt sich auf den Austausch von Werten mit einem Werkzeug über ein Modell. Die Möglichkeit für beide Seiten zu argumentieren macht es für uns schwierig, eine abschließende Meinung zu bilden. Letztlich gehen wir davon aus, daß, sofern alle Möglichkeiten der Interaktionsstrategie und der Modelle genutzt werden, man von einem Sub - Werkzeug sprechen kann, obwohl wir mit unserer Konstruktion die im Teil 4.1 gesetzten Teilziele erfüllt haben.

Die Unklarheit, in die uns unsere Konstruktion in Bezug auf die Abgrenzung zu einem Sub-Werkzeug geführt hat, verschafft uns an anderen Stellen allerdings eine große Sicherheit. So finden wir sowohl unsere Sicht auf die Struktur einer Tabelle, als auch die Möglichkeit der Definition des fachlichen Umganges mit einer Zelle wieder. Dadurch, daß die Realisierung eines bestimmten Umgangs mit einer Zelle im Inneren des Widgets erfolgt, erreichen wir die größtmögliche Kontrolle über das Verschicken von Commands an das Werkzeug. Als Folge hiervon, waren wir in der Lage vier Commands zu definieren, die einem Werkzeug alle notwendigen Informationen über den aktuellen Zustand der Tabelle liefern können.

In einer der ersten Diskussionen bezüglich der Konstruktion von Tabellen wurden wir darauf hingewiesen, daß der Einsatz von Modellen nicht nur positive Erscheinungen hat. Das einfache ändern eines Wertes innerhalb eines Modells, kann zu einer Inkonsistenz mit dem von der Funktionskomponente bearbeiteten Material führen, sofern keine Synchronisation zwischen diesen beiden Teilen erfolgt. Ohne weiteres ist es in diesem Zusammenhang dann auch möglich, die Interaktionskomponente zu umgehen, was das innerhalb des WAM - Leitbildes aufgebaute Konstruktionsprinzip aufweichen würde. Um diese Gefahr zu umgehen, haben wir die Sicht auf ein die Werte enthaltenes Modell der Art modifiziert, daß Änderungen an Werten immer über die Interaktionskomponente an die Funktionskomponente weitergeleitet werden. Das direkte Schreiben von Werten in das Modell von der Seite der Präsentationsform ist nicht möglich (siehe 4.2.1). Durch diese Änderungen des internen Kontrollflusses können wir zu jeder Zeit gewährleisten, daß die Änderung eines Wertes in der Tabelle auch immer die Änderung eines Materials zur Folge hat. Zudem besteht so die Möglichkeit, gezielt auf ungültige Eingaben des Benutzers zu reagieren.

Diese Konstruktion ist erfolgreich in das JWAM Rahmenwerk unter Berücksichtigung der vorhandenen Prinzipien eingebunden worden. Henning Wolf und Stefan Roock haben mittlerweile erste Erfahrungen im Umgang mit diesen Tabellen im SaffranAB Projekt gesammelt.

5.3 Übertragbarkeit der Konstruktion

In der Einleitung zu diesem Kapitel haben wir gesagt, daß wir das Interesse auf den Umgang mit Widgets legen wollen, bei denen der Inhalt nicht zum Zeitpunkt der Widgetkonstruktion vorhersagbar ist. Die Tabelle galt für uns als ein Beispiel für Widgets auf die diese Eigenschaft zutrifft. Eine Frage, die sich für uns nach der theoretischen und praktischen Realisierung eines Widgets für Tabellen stellt, besteht darin, ob diese Konstruktion tatsächlich auch auf andere Widgets anwendbar ist. Mit dem Ziel, diese Frage zu klären, haben wir uns dazu entschlossen, ein weiteres Widget in das JWAM Rahmenwerk zu integrieren.

In diesem Fall haben wir uns für ein Treewidget entschieden, da es eine ähnlich mächtige Funktionalität aufweist und ebenso in der Lage sein muß, mit einer großen Anzahl von Werten umzugehen. Ein typisches Beispiel für den Einsatz von Bäumen in grafischen Benutzerschnittstellen sind alle Formen von Dateimanagern, die das jeweilige Dateisystem hierarchisch innerhalb einer Oberfläche bedienbar machen sollen.

Exemplarisch führen wir hier den Explorer des Betriebssystems Windows NT 4.0 an. Dieser stellt auf der rechten Seite, das hierarchisch aufgebaute Dateisystem dar. Zudem existiert die Möglichkeit den einzelnen Knoten eine individuelle Icon zuzuordnen. Vielmehr sehen wir an dieser Stelle eine Zuordnung der Einträge in dem Baum zu einem bestimmten Typ, der wiederum durch ein definiertes Icon angezeigt wird. So haben wir auf der einen Seite die einfachen Ordner, repräsentiert durch das Symbol eines Aktendeckels, und auf der anderen Seite zum Beispiel die sogenannten Recycled - Ordner, die für den Papierkorb stehen, der gelöschte Objekte enthält. In unserem speziellen Beispiel in Abbildung 5-11 sehen wir zusätzlich die Kombination eines Baumes mit einer Tabelle in einem Werkzeug. Hier wird also in Abhängigkeit von der Auswahl eines Knotens durch den Benutzer eine Tabelle angezeigt, die dieser Auswahl über eine Anwendungslogik zugeordnet wurde. Auch der Umgang mit Einträgen innerhalb des Baumes kann variieren. So ist es bei den den physikalischen Laufwerken zugeordneten Einträgen dem Benutzer möglich, den Namen direkt zu editieren. Eine Ausnahme hierbei stellen die Wechseldaten-

träger wie ZIP- oder CD-ROM - Laufwerk dar. Bei diesen Geräten ist solch eine Änderung nicht möglich.

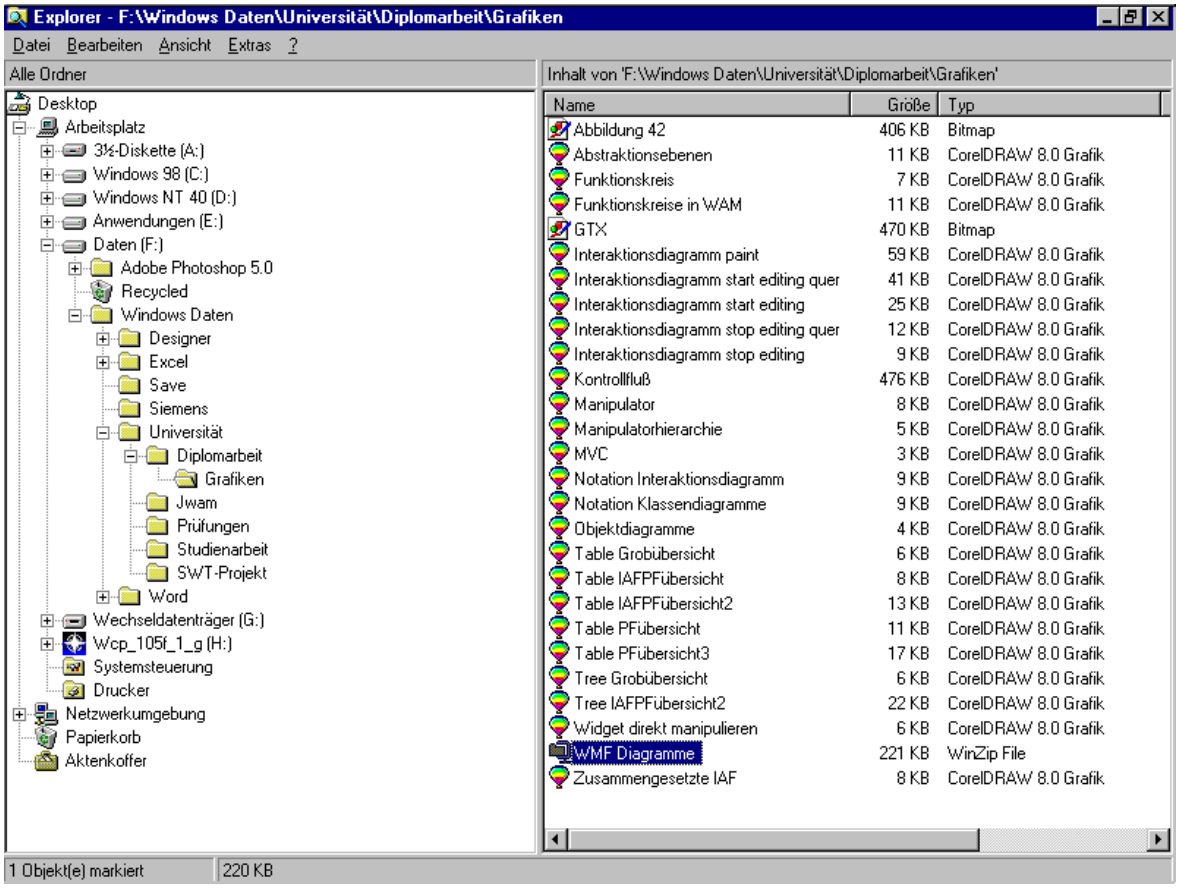

Abbildung 5-11: Windows NT 4.0 Explorer

5.3.1 Konstruktion eines Treewidgets

Nachdem wir bis jetzt nur anhand eines Beispiels die Möglichkeiten von Treewidgets erörtert haben, möchten wir, ähnlich wie in dem Abschnitt über Tabellen, sowohl die Struktur, als auch den Umgang mit Bäumen definieren. Zu der Struktur von Bäumen finden wir bei Hopcroft und Ullman:

> *Ein Baum (genau genommen ein geordneter gerichteter Baum) ist ein gerichteter Graph mit folgenden Eigenschaften:*
>
> 1. *Es gibt einen Knoten – den man als Wurzel bezeichnet – der keine Vorgänger besitzt und von dem aus ein Pfad zu jedem Knoten existiert.*
> 2. *Jeder Knoten außer der Wurzel hat genau einen Vorgänger.*
> 3. *Die Nachfolger eines Knotens sind von „links nach rechts" geordnet.*

Wie auch schon unser einfaches Beispiel eines Dateimanagers zeigt, können wir nicht davon ausgehen, daß der Umgang mit allen Elementen eines Baumes einheitlich ist. Im

Hinblick auf den Umgang eines Benutzers mit einem Baum fordern wir deshalb in Anlehnung an unsere Definition des Umganges mit den Zellen einer Tabelle, daß der konkrete Umgang mit einem Knoten in Abhängigkeit von einem zugeordneten Typ erfolgt. Somit können wir auch bei Treewidgets zum Zeitpunkt der Konstruktion nicht vorhersehen, welche Art von Werten dargestellt werden sollen.

Über die Tatsache, daß wir es hier auf der einen Seite mit einem Widget zu tun haben, bei dem der Typ der Elemente sehr stark von dem Einsatzkontext abhängt und auf der anderen Seite eine große Zahl an Einträgen zu erwarten ist, sind wir zu der Überzeugung gelangt, daß die prinzipielle Konstruktion der Tabelle auch für Trees einsetzbar ist. Die so entstandene Konstruktion möchten wir nun kurz vorstellen.

Anfangen möchten wir wieder mit der Einbindung der notwendigen Modelle in die Konstruktion. Allen beteiligten Komponenten liegt die entsprechende in den Abschnitten 5.2.2 – 5.2.7 erläuterte Beschreibung zu Grunde. Ein erster Ausschnitt aus der Realisierung des Treewidgets ist in der folgenden Abbildung zu sehen.

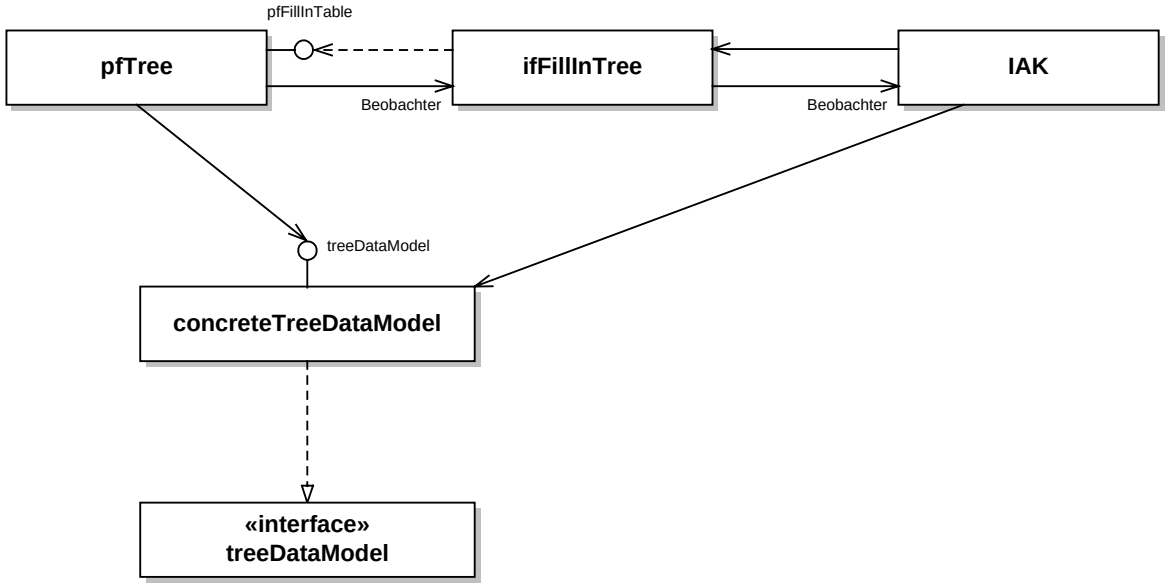

Abbildung 5-12: Erster Einstieg in die Tree - Konstruktion

Bezogen auf das Beispiel des Dateimanagers ist es möglich, direkt die Elemente eines Baumes zu editieren, ohne den Umweg über ein spezielles Werkzeug zu gehen. Für uns bedeutet dies, daß der Umgang eines Benutzers über die reine Auswahl eines oder mehrerer Elemente hinausgeht. Deswegen haben wir für die Anbindung eines Treewidgets an ein Werkzeug die Interaktionsform `ifFillInTree` implementiert. Diese Kategorisierung ist an dieser Stelle nur eine erste Annahme, da unser Hauptaugenmerk in diesem Fall auf die eigentliche Konstruktion gerichtet war und nicht auf die Erarbeitung des Umganges.

Auch hier haben wir ein entsprechendes Modell (siehe auch 4.2.1) eingesetzt, über das die einzelnen Werte zur Anzeige gelangen. An dieses Modell werden nun aber andere Anforderungen gestellt, als an das von der Tabelle bekannte `tableDataModel`. Dies liegt unter anderem darin begründet, daß wir bei Bäumen eine hierarchische Anordnung der Werte vorfinden, wohingegen bei Tabellen die Werte in einer zweidimensionale Matrix organisiert sind. Im Hinblick auf den Zugriff auf die einzelnen Elemente existieren fundamentale Unterschiede, die von uns bei der Definition eines Modells berücksichtigt

werden müssen. Um diesem Unterschied gerecht werden zu können, haben wir in das `treeDataModel` ein fachlich motiviertes Konzept bezüglich des Umganges mit Bäumen integriert. Diese Schnittstelle arbeitet auf den uns bekannten Begriffen, wie zum Beispiel Knoten, Wurzel oder Vorgänger. Weiter ist es auch möglich, ein Element eines Baumes über den eindeutigen Pfad in diesem Baum zu bestimmen. Bei der Implementation dieser Schnittstelle in einem `concreteTreeDataModel` ist als Konsequenz mit mehr Aufwand bei der Umsetzung zu rechnen, als bei dem für eine vergleichbare Funktion eingesetztem Modell einer Tabelle. Wir sind allerdings der Ansicht, daß die eingesetzten Modelle auch den Aufbau der Wertemenge darstellen sollten. Ist ein Wert dieser Menge über eine Matrix eindeutig zu identifizieren, so soll dies aus dem Modell genauso deutlich werden, wie die Identifikation eines Wertes über einen Pfad bei hierarchisch geordneten Werten. Die Komplexität eines Modells und der damit verbundene Aufwand bei der Implementierung steht in unmittelbarer Abhängigkeit von der Organisation der Wertemenge.

Das Layout eines solchen Widgets ist schon durch die anfänglich angeführte Definition der Baumstruktur vorherbestimmt. Aus diesem Grunde haben wir hier auf den Einsatz eines Layout Modells verzichtet. Da das Layout Modell insgesamt auch nur einen optionalen Charakter besitzt, haben wir an dieser Stelle keinen Widerspruch, zu den Annahmen, die wir in dem Abschnitt 5.2.2 getroffen haben.

Zusätzlich zu den bereits angesprochenen Bestandteilen Interaktionsform und Modell wird auch bei diesem Widget eine Interaktionsstrategie und eine Präsentationsstrategie benötigt. Es sei an dieser Stelle noch einmal betont, daß der Anwendungsentwickler ausschließlich für die Implementierung des die Interaktionsstrategie definierenden Interfaces zuständig ist. Die Präsentationsstrategie wird schon zum Zeitpunkt der Widgetkonstruktion geschrieben und muß im Normalfall nicht mehr angepaßt werden. Die Anzahl der für die Interaktionsstrategie eines Trees benötigten Methoden reduziert sich deutlich gegenüber der bei einer Tabelle benötigten Anzahl. Letztlich finden sich hier nur noch drei Methoden wieder:

- `getIFs (type)`
- `customizeNode (ifs[],value, type, isSelected)`
- `getNodeValue (ifs[], type)`

Der Unterschied zu den in einem `tableInteractionStrategy` Interface definierten Methoden besteht darin, daß wir an dieser Stelle versucht haben, die zu verwendenden Interaktionsformen über einen dem Element zugeordneten Typ zu bestimmen. Dieser Ansatz erschien uns insbesondere dadurch sinnvoll, da bei diesem Widget die Elemente keiner übergeordneten Gruppe, wie zum Beispiel einer Spalte, zugeordnet sind. Die anderen beiden Methoden dienen der Parametrisierung und dem Auslesen der Werte eines Widgets. Wir haben hier nur die Namen der Methoden entsprechend an den Kontext angepaßt. Die Funktionalität entspricht der, die wir bereits von den Tabellen her kennen.

Identisch zu dem Vorgehen bei Tabellen werden die für die Darstellung der Elemente eines Baumes über die Präsentationsform registriert. Über die Präsentationsform `pfTree` wird die SWING Klasse `Jtree` gekapselt. Für die Organisation des Ablaufes im Inneren der Konstruktion konnten wir auf die gleichen Prinzipien zurück greifen, die wir bereits bei Tabellen angewandt haben. Dies gilt insbesondere für die interne Verbindung von Interaktionsformen und Präsentationsform für ein darzustellendes Element eines Baumes.

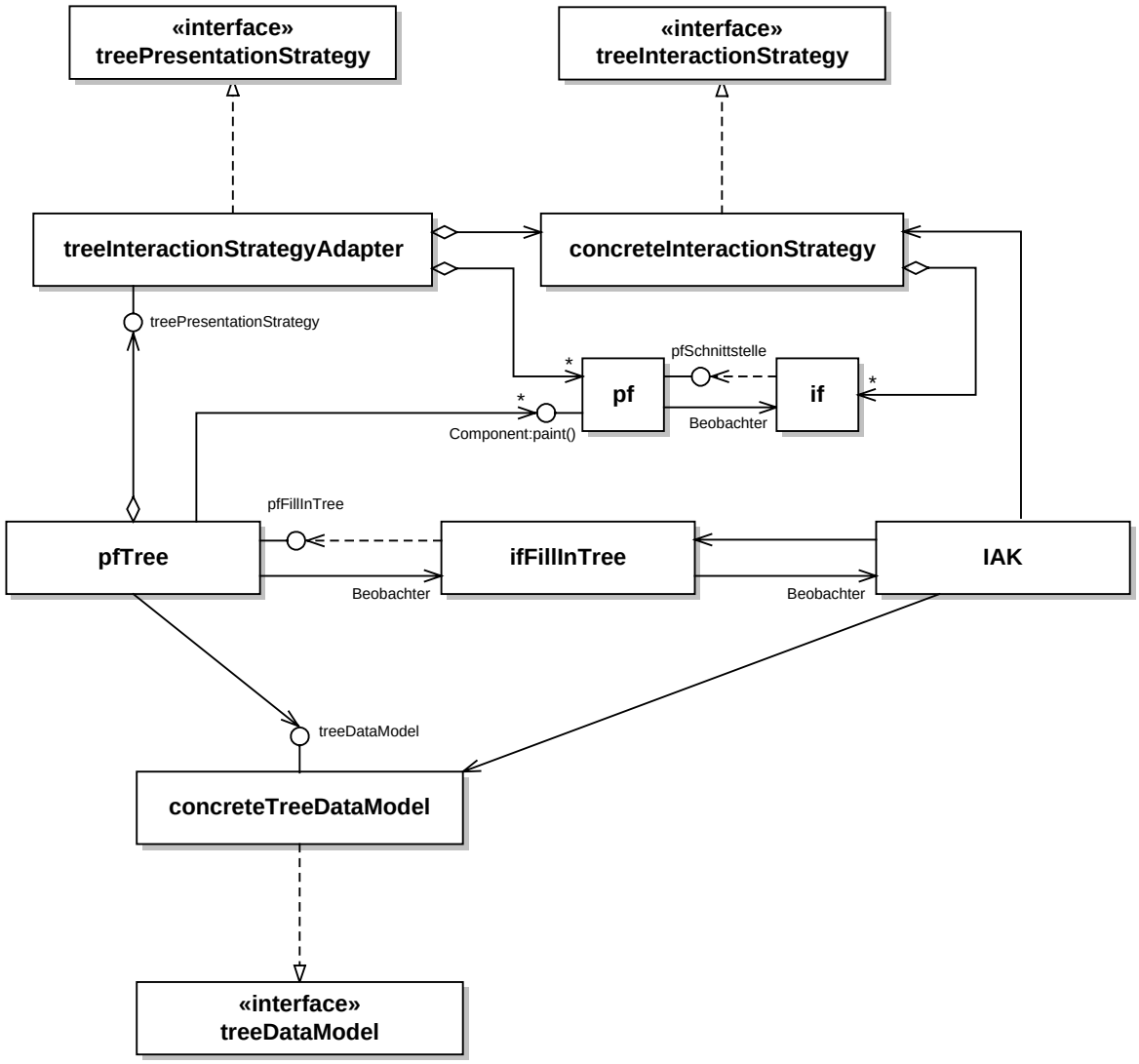

Abbildung 5-13: Klassendiagramm der Tree - Konstruktion

Bei der Realisierung des Treewidgets sind wir in zwei Schritten vorgegangen. Zuerst haben wir unsere eigene Präsentationsform geschaffen, welche die Funktionalität des `Jtree` aus SWING erbt. Innerhalb der Präsentationsform haben wir dann die notwendige Schnittstelle zu unserer Interaktionsform geschaffen. Dieses Vorgehen entspricht dem aus der Integration beliebiger Widgets in das JWAM Rahmenwerk bekannten Prinzip. Zusätzlich haben wir dann in einem zweiten Schritt die notwendigen Interfaces für das Modell und die Interaktions- und Präsentationsstrategie angepaßt.

Die Struktur des so entstandenen Widgets ist unserer Meinung nach als identisch zu der Struktur anzusehen, die wir für Tabellen vorgeschlagen haben. Es ist uns hierbei wichtig, zu betonen, daß wir diese Struktur nicht auf ein Treewidget angewendet zu haben, ohne den Umgang eines Benutzers mit einem Tree zu berücksichtigen. Daß eine solche Übertragung von Konstruktionsprinzipien nicht ohne Änderungen zu bewältigen ist, zeigt die Notwendigkeit, die einzelnen Schnittstellen entsprechend der fachlich motivierten Verwendung mit Hilfe der Interfaces anzupassen. Diese Änderungen betreffen allerdings in keiner Weise die Verbindungen der einzelnen Bestandteile untereinander, so daß wir davon ausgehen können, daß diese Konstruktion durchaus auf andere Widgets übertragbar ist.

5.3.2 Muster für Widgets

Im Rahmen der Entwicklung ist die Anbindung der Widgets über das Prinzip der Interaktions- und Präsentationsformen ein fester Bestandteil geworden. Nach wie vor ist es jedoch möglich, die Widgets der Oberfläche auch auf konventionelle, sprich direkte, Weise an die Interaktionskomponente anzubinden. Aus der im Rahmen dieser Arbeit gewonnenen Erfahrung im Umgang mit GUI's unter WAM, ziehen wir den Schluß, daß die Vorteile des Einsatzes einer separaten Abstraktionsebene gegenüber einer direkten Anbindung von Widgets überwiegen. Wir beziehen uns hier auf die von uns im Kapitel 4.2 geführte Diskussion über die Vor- und Nachteile einer solchen Konstruktion.

Der Einsatz einer solchen Abstraktionsebene bedeutet aber nicht, daß die Art der Widgets nicht berücksichtigt werden muß. Auf der Seite des fachlichen Umganges wird dieses sicherlich mit Hilfe der unterschiedlichen Interaktionsformen berücksichtigt. Auf der anderen Seite existieren aber auch für Widgets Rahmenbedingungen, die Berücksichtigung finden müssen. Hierzu zählt zum Beispiel die Größe der durch ein Widget darzustellenden Wertemenge, oder aber die Eigenschaft, daß der Umgang des Benutzers mit einem Widget in Abhängigkeit von den dargestellten Werten variieren kann. Generell denken wir, daß wir das grundlegende Prinzip der Interaktions- und Präsentationsformen als ein Entwurfsmuster ansehen können mit dessen Hilfe die Anbindung von Werkzeugen an grafische Benutzerschnittstellen beschrieben wird. In [GHJV96, Seite 4] finden wir hierzu

> *Die Bestimmung dessen, was ein Muster ist und was nicht, hängt von der jeweiligen Perspektive ab. Was aus einer Perspektive als Muster erscheint, stellt aus einer anderen Perspektive betrachtet einen primitiven Baustein dar.*
>
> *Die Entwurfsmuster in diesem Buch sind Beschreibungen zusammenarbeitender Objekte und Klassen, die maßgeschneidert sind, um ein allgemeines Entwurfsproblem in einem bestimmten Kontext zu lösen.*

Aus der Problemstellung heraus betrachtet, daß ein Werkzeug mit einem Widget kommunizieren soll, ohne eine Abhängigkeit von dem gewählten Toolkit einzugehen, steht uns mit dem Prinzip der Interaktions- und Präsentationsformen ein geeignetes Entwurfsmuster zur Verfügung. Wir möchten nun dieses Muster um optionale Teile erweitern, die einen geeigneten Umgang mit den verschiedensten Widgets ermöglichen. Es handelt sich insgesamt um zwei zusätzliche Teile, die optional in dieses Konzept eingebaut werden können. Bei dem ersten Bestandteil handelt es sich um die Einbettung der Idee eines Modells für die Werte und einem Modell für das Layout des Widgets. Der zweite zusätzliche Bestandteil betrifft Widgets, bei denen während der Konstruktion noch keine Aussagen über den Typ der darzustellenden Werte gemacht werden kann.

Da diese Zusätze aus den Erfahrungen stammen, die wir während der Konstruktion von Tabellen und Trees gemacht haben, ist es nicht weiter verwunderlich, daß sich die in der Abbildung 5-14 dargestellte Struktur nicht von der in Kapitel 5.2 vorgestellten Struktur unterscheidet.

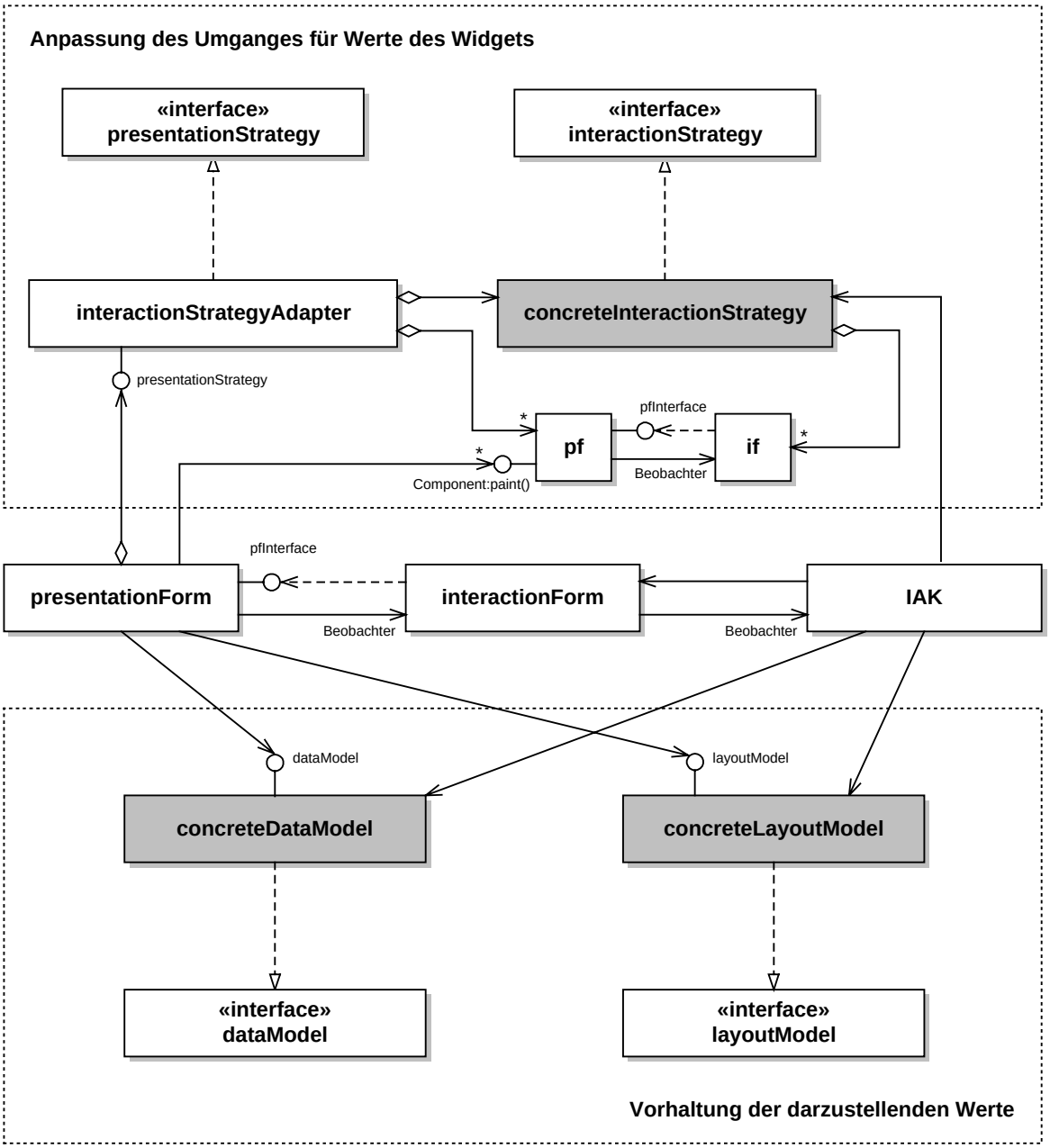

Abbildung 5-14: Ein Muster für Widgets

Das Grundmuster besteht nach wie vor in der losen Koppelung einer Interaktionskomponente und einer Interaktionsform, die den Umgang mit einem Widget repräsentiert. Diese ist wiederum mit einer Präsentationsform verbunden, welche die eigentliche Darstellung des Widgets auf einem Grafikkontext übernimmt. Für die Einführung eines neuen Widgets innerhalb eines Rahmenwerkes müssen alle Klassen, mit Ausnahme der grau hinterlegten, implementiert werden. Die grau hinterlegten Klassen, müssen bei der Verwendung dieses Musters durch einen Anwendungsentwickler konkretisiert werden.

Abschließend werden wir die einzelnen Teilnehmer an diesem Muster mit ihren Aufgaben kurz darstellen und auf die Konsequenzen dieser Konstruktion eingehen.

- `interactionForm`
  - definiert den Umgang eines Benutzers mit einem Widget (Bsp: `ifActivator`)

- `presentationForm`
  - kapselt ein Widget eines beliebigen Toolkits, um die Koppelung mit einer Interaktionsform zu realisieren (Beispiel: `pfListBox`)
- `dataModel` (Interface)
  - definiert eine fachlich motivierte Schnittstelle, mit deren Hilfe die Präsentationsform lesend und schreibend auf Werte zugreifen kann (siehe auch Kapitel 5.2.2)
- `layoutModel` (Interface)
  - definiert eine Schnittstelle, über welche die Präsentationsform Einstellungen des Benutzers in einem konkreten Objekt hinterlegen kann, mit der Möglichkeit diese Einstellungen jederzeit restaurieren zu können (siehe auch Kapitel 5.2.3)
- `interactionStrategy` (Interface)
  - definiert Methoden, die für das Widget notwendig sind, um den Umgang mit in ihm enthaltenen Werten realisieren zu können (siehe auch Kapitel 5.2.4)
- `presentationStrategy` (Interface)
  - definiert Methoden, die für die Einbettung der Strategie in das Widget eines Toolkits notwendig sind (siehe auch Kapitel 5.2.4)
- `interactionStrategyAdapter`
  - implementiert die Methoden der `presentationStrategy` und ist für den dynamischen Aufbau der Verbindungen zwischen Interaktionsformen und Präsentationsform, die für die Darstellung eines Wertes benötigt werden zuständig (siehe auch Kapitel 5.2.4)

Der Einsatz eines solchen Musters zieht neben den im vorwege erwähnten Vorteilen auch negative Konsequenzen nach sich. Aus der Sicht eines Anwendungsentwicklers ergibt sich ein komplexeres Bild bezüglich der Anbindung von einzelnen Widgets an ein Werkzeug bedingt durch den Mix an beigestellten und beizustellenden Komponenten. Zudem fördert die zusätzliche Einbindung von Objekten zur Laufzeit nicht die Performanz der Oberfläche. Diese spielt aber im Hinblick auf die Akzeptanz eines Softwareproduktes eine besondere Rolle.

6 Erweiterungen des IAF / PF Musters

Im Rahmen dieses Kapitels wollen wir eine mögliche Erweiterung des Interaktionsform und Präsentationsform Musters um Konzepte zum Zugriff und zur Manipulation von Widgeteigenschaften vorstellen. Bei diesen Eigenschaften handelt es sich um Aspekte wie zum Beispiel Farbe oder Größe. Im zweiten Teil des Kapitels werden wir den vorgestellten Ansatz weiterentwickeln zu dem Konzept der zusammengesetzten Interaktionsformen.

6.1 Zugriff auf Widgeteigenschaften

Bei der Entwicklung einer Benutzungsoberfläche gibt es für den Anwendungsentwickler einen großen Spielraum. Es stehen ihm eine Vielzahl von Widgets zur Verfügung, die er beliebig anordnen und mehr oder weniger funktional kombinieren kann. Neben der reinen Funktionalität sollten Benutzungsoberflächen auch nach softwareergonomischen Gesichtspunkten betrachtet werden, da angestellte Untersuchungen, unter anderem in [Shnei98], zeigen, daß sich durch einen guten Oberflächenentwurf die Arbeitsgeschwindigkeit steigern läßt und sich die Fehlerraten mindern lassen.

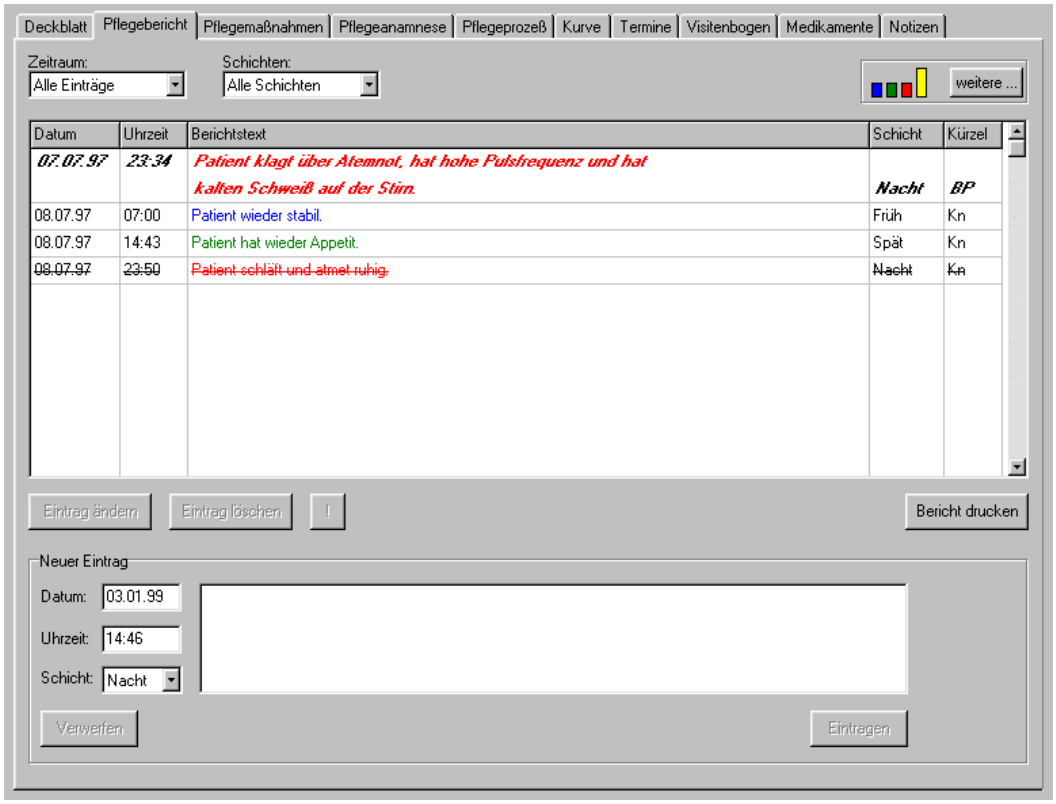

Abbildung 6-1: Pflegebericht im HIPPO - Prototyp

Bei der Entwicklung einer Benutzungsoberfläche unterscheiden wir zwischen dem statischen Entwurf und der Dynamik zur Laufzeit. Der statische Entwurf wird vom Entwickler zur Designzeit der Oberfläche bestimmt. Hierzu zählt die Auswahl der verwendeten Widgets, ihre Position und Größe sowie die Strukturierung der Widgets und der

nicht interaktiven Elemente wie zum Beispiel Rahmen und Linien. Der statische Entwurf ändert sich in der Regel zur Laufzeit nicht.

Das Verhalten und die Dynamik der Oberfläche zur Laufzeit hingegen wird auf der einen Seite durch den Anwendungsentwickler über die Gestaltung und Programmierung der Interaktionskomponenten und auf der anderen Seite durch die Handlungen eines Benutzers bestimmt. Durch Verwendung von verschiedenen Arten von Prototypen (siehe [KLSZ92]), insbesondere durch Verwendung von funktionalen Prototypen, kann das Verhalten, die Dynamik und die Funktionalität der Benutzungsoberfläche von allen beteiligten Gruppen evaluiert werden.

Softwareergonomisch gute Benutzungsoberflächen zeichnen sich unter anderem dadurch aus, daß die Informationen gut strukturiert sind und besonders wichtige Informationen hervorgehoben sind. Zudem finden wir in [Grand95] die Forderung, daß diese Systeme in der Lage sein sollen, dem Benutzer ein sofortiges optisches und / oder akustisches Feedback auf seine Aktionen zu geben. Diese Eigenschaften zählen zur Dynamik einer Oberfläche. Am Beispiel eines Prototypen zur Unterstützung von Pflegepersonal in einer Klinik, der im Rahmen des HIPPO - Projektes[15] an der Universität Hamburg und in Zusammenarbeit mit dem UKE Hamburg entwickelt wurde, möchten wir ein Beispiel für die Strukturierung und Hervorhebung von Informationen geben. Die Abbildung 6-1 zeigt einen Ausschnitt der grafischen Benutzerschnittstelle des Prototypen. Der Ausschnitt zeigt den Pflegebericht, welcher für jeden Patienten von der Pflegekraft geführt werden muß. Zum besseren Verständnis über den Umgang mit einem Pflegebericht muß man wissen, daß die Arbeitszeit im untersuchten Krankenhaus in drei verschiedene Schichten eingeteilt wird. Jede Schicht verwendet beim Ausfüllen des Pflegeberichtes eine eigene Farbe, die in der Abbildung 6-1 aus technischen Gründen leider nur als unterschiedliche Grauschattierungen zu erkennen sind. Die Korrektur eines Eintrages erfolgt durch ein sauberes Durchstreichen des falschen Eintrages, so daß die ursprüngliche Eintragung lesbar bleibt. Der korrigierte Eintrag wird in einer neuen Zeile eingetragen. Hierbei besteht die Möglichkeit, wichtige Einträge besonders hervorzuheben. Im vorliegenden Beispiel wurde versucht, den Umgang mit dem real existierenden Pflegebericht auf den Pflegebericht im Computer zu übertragen.

Der Anwendungsentwickler kann in diesem Beispiel zwar die grobe Struktur zur Designzeit festlegen, zur Laufzeit müssen jedoch viele Parameter wie Farbe und Schrifteigenschaften des dargestellten Pflegeberichtes geändert werden. Im JWAM Kontext stellt diese Anforderung aufgrund der Trennung von Interaktion und Präsentation ein Problem dar. Durch die hohe Abstraktion vom konkreten GUI Toolkit bei der Werkzeugentwicklung ist eine direkte Manipulation[16] der Eigenschaften eines Widgets zur Laufzeit nahezu ausgeschlossen.

Innerhalb des JWAM Rahmenwerkes gibt es zwei Abstraktionsebenen zwischen dem konkret verwendeten GUI Toolkit und der eigentlichen Interaktionskomponente im Werkzeug. In Abbildung 6-2 sind diese zwei Abstraktionsebenen grafisch dargestellt.

Um die Abhängigkeit von einem konkreten GUI Toolkit zu minimieren und den Austausch des GUI Toolkits zu vereinfachen, wurde eine erste Abstraktionsschicht mit Hilfe von

---

[15] HIPPO steht im Rahmen dieses Projektes für: Hilfe in Pflegeplanung und Organisation

[16] Mit dem Begriff „direkte Manipulation" ist in diesem Zusammenhang nicht das Manipulieren von Widgets durch den Benutzer und ein sofortiges semantisches Feedback gemeint, wie es zum Beispiel bei Drag and Drop Operationen Anwendung findet.

Präsentationsformen eingeführt. Präsentationsformen kapseln die konkreten Widgets entweder über den Einsatz eines Adaptermuster oder durch Unterklassenbildung. Die Schnittstelle dieser Präsentationsformen werden durch spezielle PF - Interfaces vorgegeben, wobei es immer möglich ist, daß ein Widget mehrere dieser PF - Interfaces implementiert. Die zweite Abstraktionsschicht wird durch die Interaktionsformen repräsentiert. Die Präsentationsformen werden nicht direkt durch die Interaktionskomponente der Werkzeuge angesprochen, sondern über den Umweg einer Interaktionsform benutzt. Interaktionsformen stellen eine Abstraktion über die Handhabung und den Umgang mit Widgets dar. Ihre Schnittstelle ist rein vom fachlichen Umgang geprägt und völlig unabhängig vom konkret verwendeten GUI Toolkit (siehe auch [Görtz98] und [Lipp97a]). An der Schnittstelle einer Interaktionsform finden sich deshalb auch keine Methoden zur Manipulation der grafischen Repräsentation eines Widgets wie zum Beispiel der Position, Größe, Farbe oder Schriftart. Dies hat unter anderem den Vorteil, daß bei einem Austausch des GUI Toolkits die Interaktionsformen und damit auch die Interaktionskomponenten der Werkzeuge nicht angepaßt werden müssen.

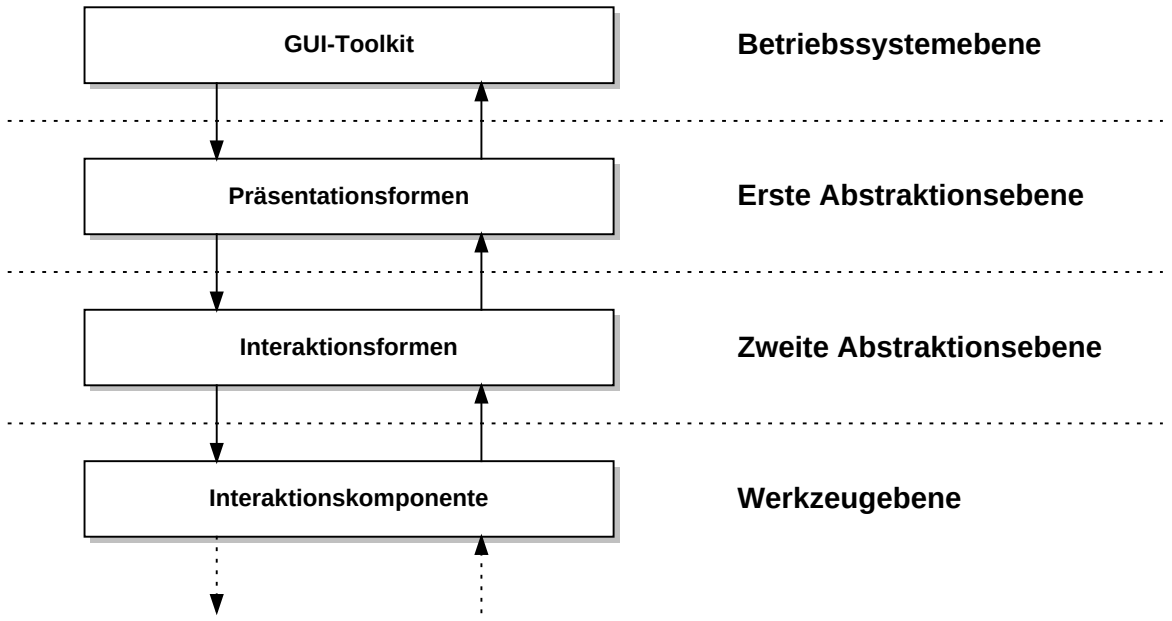

Abbildung 6-2: GUI Abstraktionsebenen in JWAM

Die oben genannten Vorgaben des Prototypen und andere softwareergonomische Forderungen lassen sich mit den im JWAM Rahmenwerk vorgegebenen Konzepten momentan nicht umsetzen. In der Folge werden wir diesbezüglich einige Lösungsansätze vorstellen und auf eine Einsetzbarkeit überprüfen.

Eine sehr einfache Lösung dieses Problems besteht darin, die Schnittstellen der Interaktionsformen und entsprechend die der Präsentationsformen um Methoden zur Manipulation von Farbe, Font, Stil, usw. zu erweitern. Dieses kann entweder generell im Rahmenwerk vorgesehen werden oder nach Bedarf durch Unterklassenbildung in jedem Anwendungsprogramm einzeln realisiert werden. Technisch betrachtet stellt diese Lösung zwar einen gangbaren Weg dar, jedoch unterlaufen diese Schnittstellenänderungen die zweite Abstraktionsebene. Änderungen an optischen Merkmalen der Präsentationsformen stellen nur in wenigen Fällen eine Interaktion mit dem Benutzer dar und sollten deshalb in den Schnittstellen der Interaktionsformen nicht auftauchen. Eine weitere Einschränkung ist, daß die Erweiterung der Schnittstelle einer Interaktionsform nicht immer problemlos

möglich ist. Dieses beruht auf der Tatsache, daß es in einigen Fällen durchaus sinnvoll ist, mehrere Präsentationsformen an eine Interaktionsform zu binden (siehe dazu auch [Görtz98]). Die Erweiterung einer Schnittstelle um diese spezifischen Eigenschaften bedeutet also, daß alle an die betreffende Interaktionsform gebundenen Präsentationsformen diese Eigenschaften auch unterstützen müssen. Dies ist jedoch nicht immer gegeben. In vielen Anwendungen werden Iconleisten verwendet, die einen Schnellzugriff auf häufig verwendete Befehle erlauben. Für die Realisierung in dem JWAM Kontext würde man zu diesem Zweck in der Interaktionskomponente eines Werkzeuges die Interaktionsform `ifActivator` verwenden, die den Umgang des „Aktivieren" kapselt. Diese Interaktionsform wird über den GUI Kontext, dessen Aufgabe in der Verwaltung aller Widgets besteht, gleichzeitig an einen Menüeintrag und an einen Button mit einem Icon gebunden. Die generelle Erweiterung der der Interaktionsform `ifActivator` zu Grunde liegenden Schnittstelle um eine Methode zur Änderung der Schriftattribute ist in diesem Fall weder möglich noch fachlich sinnvoll, da der Iconbutton keinen Text enthält und somit auch keine Schriftattribute unterstützt. Die Schnittstellenerweiterung der Interaktionsformen kann für uns deshalb keine geeignete Lösung darstellen.

Eine weitere Alternative besteht darin, die Interaktionskomponente direkt auf die konkreten Widgets zugreifen zu lassen. Die Abbildung 6-3 erläutert diesen Sachverhalt näher. Die Interaktionskomponente läßt sich zunächst vom GUI Kontext eine Referenz auf das konkret verwendete Widget Objekt des GUI Toolkits geben. Anhand der erhaltenen Referenz kann die Interaktionskomponente die volle Schnittstelle des Widgets sehen und nutzen. Die Interaktionskomponente ist nun in der Lage die gewünschten Änderungen direkt am Widget, unter Umgehung der Interaktionsform, vorzunehmen.

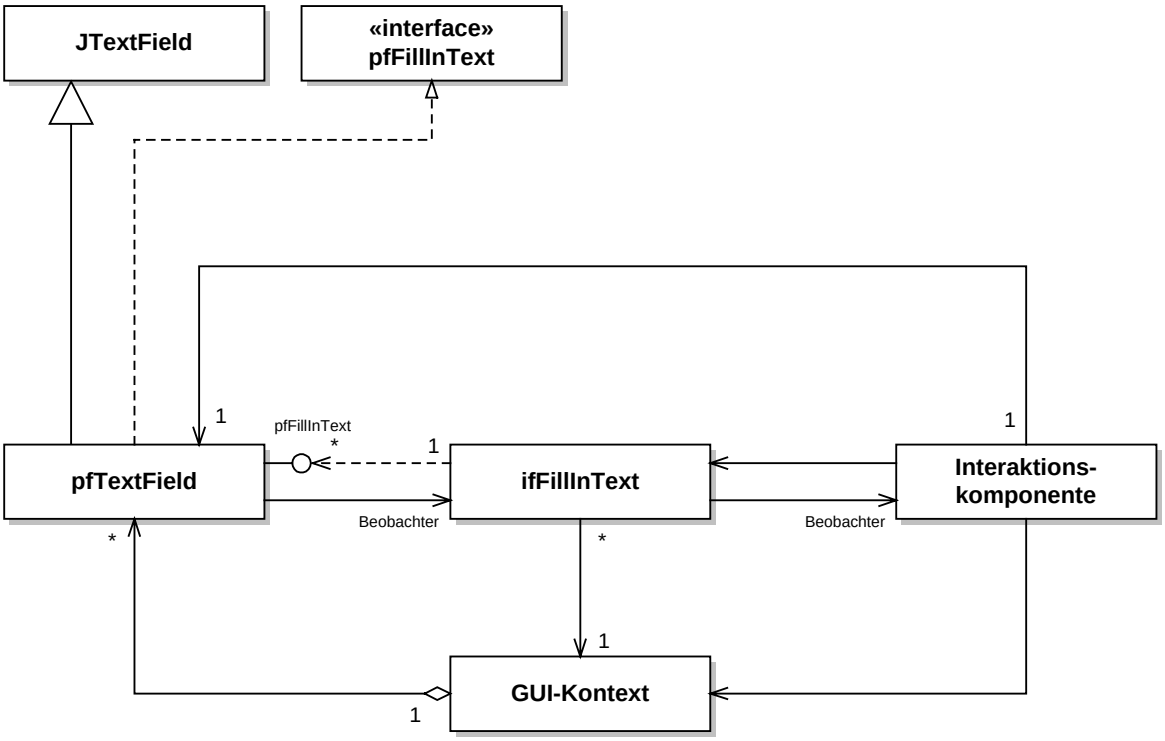

Abbildung 6-3: Direkter Zugriff der IAK auf eine Präsentationsform

Diese Lösung hat den Vorteil, daß alle Eigenschaften und Möglichkeiten der Widgets eines GUI Toolkits potentiell genutzt werden können. Die Nachteile einer solchen Konstruktion wiegen jedoch schwer. Durch die direkte Verwendung der Widgets werden die Vorteile

der beiden Abstraktionsschichten aufgegeben. Bei einem Austausch oder Update des Toolkits ändern sich oft die Methodenschnittstellen der Widgets. Dies hat Änderungen an allen Interaktionskomponenten, welche die Widgets direkt verwenden, zur Folge. Neben den veränderten Schnittstellen benutzen unterschiedliche Toolkits meistens auch eigene Datentypen und Hilfsklassen[17]. Als direkte Folge können wir an dieser Stelle einen zusätzlichen Änderungs- und Einarbeitungsaufwand für den Entwickler ausmachen. Insgesamt betrachtet, ist dieser Lösungsansatz unserer Meinung nach nur für kleine Anwendungen geeignet, bei denen der Anpassungsaufwand bei einem Austausch des Toolkits in einem überschaubaren Rahmen bleibt und sich die Einführung aufwendigerer Konzepte nicht lohnt. Für große Anwendungen erscheint uns der mögliche Anpassungsaufwand jedoch nicht tragbar.

Wir möchten im folgenden Abschnitt nun eine geeignete Erweiterung des Interaktionsform / Präsentationsform Musters darstellen, mit der wir in der Lage sind, den Zugriff auf die optischen Eigenschaften der Benutzungsschnittstelle sinnvoll zu ermöglichen. Wie wir bereits schon dargelegt haben, paßt die Manipulation von Widgeteigenschaften durch ein Werkzeug eigentlich nicht in das Abstraktionsschema von Interaktion und Präsentation. Dies liegt darin begründet, daß die Veränderung von Widgeteigenschaften eine einseitige Handlung von der Werkzeugseite aus darstellt und keine Interaktion zwischen dem Benutzer und dem Werkzeug im allgemeinen ist. Interaktionsformen lassen sich deshalb in der jetzigen Form nicht um Möglichkeiten zur Manipulation von Widgeteigenschaften erweitern, ohne einen Bruch in der zweiten Abstraktionsebene hervorzurufen.

Wir schlagen deshalb die Erweiterung des Musters um Manipulatoren vor. Ein Manipulator kapselt dabei einen technischen Umgang der Interaktionskomponente mit einem Widget. Über Manipulatoren kann die Interaktionskomponente Einfluß auf die optische Präsentation von Widgets nehmen. Neben der Hierarchie der Interaktionsformen für den fachlichen Umgang existiert somit eine Hierarchie von Manipulatoren, von der wir in Abbildung 6-4 einen möglichen Vererbungsbaum vorstellen.

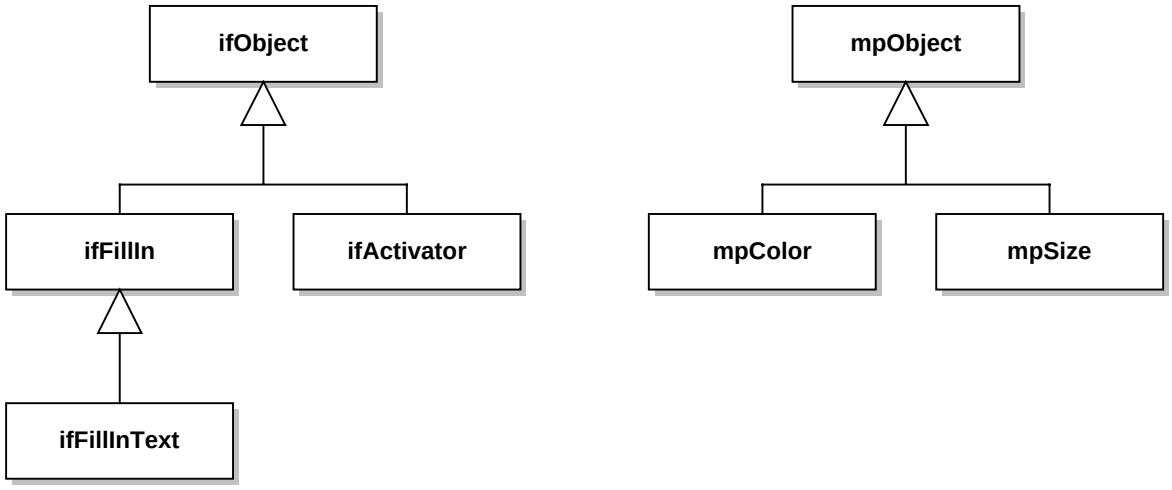

Abbildung 6-4: Parallele Vererbungshierarchie der Manipulatoren

[17] Eines der bekanntesten Beispiele in diesem Zusammenhang finden wir im Bereich der Farbkodierung. Um eine geeignete Umsetzung der Farben vornehmen zu können, setzen die gängigen Toolkits ein eigenes Farbmodell ein, welches dann über eigene Datentypen repräsentiert wird.

Der Umgang eines Anwendungsentwicklers mit Manipulatoren weist starke Ähnlichkeit zu dem Umgang mit Interaktionsformen auf. Über die Interaktionskomponente wird zunächst eine Instanz von dem gewünschten Manipulator erzeugt. Bei der Initialisierung des Manipulators wird dann, entsprechend dem Vorgang bei einer Interaktionsform, der GUI Kontext und der Name der Präsentationsform mit übergeben. Über den GUI Kontext kann der Manipulator sich anhand des Namens eine Referenz auf die entsprechende Präsentationsform geben lassen. Eine Registrierung des Manipulators als Beobachter an der Präsentationsform ist nur in seltenen Fällen notwendig, da der Benutzer in der Regel keine Möglichkeit hat, Einfluß auf Widgeteigenschaften zu nehmen. Sinn macht an dieser Stelle zum Beispiel eine Registrierung für Größenänderungen. Wenn sich die Fenstergröße ändert und ein Manipulator für Objektmaße über diese Änderung benachrichtigt wird, kann er die Interaktionsform darüber informieren, und diese kann dann das Layout der Widgets optimal an die neue Fenstergröße anpassen. Die nachfolgende Abbildung 6-5 illustriert die um Manipulatoren erweiterte Klassenstruktur des Interaktionsform / Präsentationsform Musters.

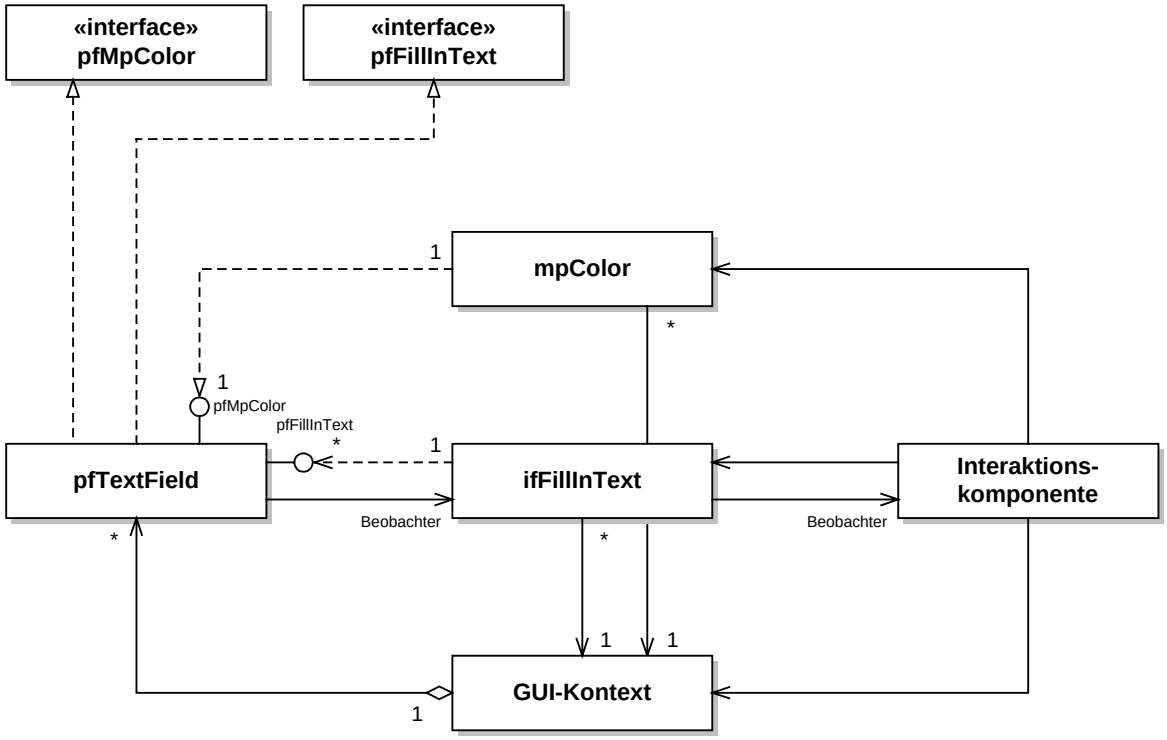

Abbildung 6-5: Klassendiagramm mit Manipulator

Damit die lose Koppelung einer Interaktionsform mit einer Präsentationsform überhaupt möglich wird, muß die Interaktionsform Kenntnisse über die Präsentationsform besitzen. Deshalb existiert für jede Interaktionsform ein Interface mit der Schnittstellenbeschreibung einer Präsentationsform. Wenn eine konkrete Präsentationsform beziehungsweise Widget diese Schnittstelle implementiert, kann die Interaktionsform sie benutzen und mit der Präsentationsform interagieren. Manipulatoren sollen aus den weiter oben schon genannten Nachteilen nicht direkt mit den Widgets eines Toolkits umgehen. Für Manipulatoren bietet sich deshalb auch die Anbindung über Interfaces an die konkreten Widgets an. Es gibt parallel zum Vererbungsbaum mit den Interfaces der Präsentationsformen auch einen Vererbungsbaum mit den Interfaces für Manipulatoren. Ein Widget implementiert dann nicht nur ein oder mehrere Präsentationsformeninterfaces sondern auch Manipulatorinterfaces.

Damit die Zuordnung von einem Manipulator zu einer Präsentationsform unabhängig von der Zuordnung einer Interaktionsform zu einer Präsentationsform vorgenommen werden kann, muß jedes Manipulatorinterface eine Methode zur Abfrage eines Manipulatornamens zur Verfügung stellen. Diese Unabhängigkeit ist notwendig, da verschiedene Widgets zwar den gleichen Umgang implementieren können, aber nicht unbedingt die gleichen Eigenschaften unterstützen.

Um das Konzept der Manipulatoren in das JWAM Rahmenwerk zu integrieren sind nur geringe Änderungen vorzunehmen. Neben dem Entwurf und der Implementierung der neuen Interfaces und Manipulatoren muß nur die den GUI Kontext repräsentierende Klasse angepaßt werden. Die Suchfunktion dieser Klasse, welche Widgets mit bestimmten Eigenschaften in den Beschreibungen der Oberflächen sucht und Referenzen auf diese zurück liefert, muß um die notwendige Funktionalität zum Umgang und Suchen der neuen Interfaces erweitert werden. Darüber hinaus sind keine weiteren Änderungen notwendig.

Der große Vorteil dieser Lösung ist die problemlose Integration in das bestehende Rahmenwerk ohne die Vorteile der beiden Abstraktionsebenen aufgeben zu müssen. Die erste Abstraktion vom konkret verwendeten GUI Toolkit bleibt weiterhin bestehen, da nur zusätzliche Präsentationsinterfaces hinzugefügt wurden. Bei einem Austausch des GUI Toolkits müssen nur wie bisher die Widgets per Vererbung oder Adaptermuster an die Präsentationsinterfaces des Rahmenwerkes angepaßt werden. Da auch die zweite Abstraktionsebene nicht umgangen wurde, müssen in der Regel an den Interaktionskomponenten der Werkzeuge keine Änderungen vorgenommen werden. Dies ist ein großer Vorzug gegenüber den anderen beiden Lösungen, was sich insbesondere bei größeren Anwendungen auswirkt, die eine große Anzahl von Werkzeugen implementiert haben.

Mit dem Konzept der Manipulatoren ist es nun auch möglich, reine Layout- und Präsentationselemente, wie zum Beispiel Labels, Images oder Linien, der Benutzungsschnittstelle zu unterstützen. Die Interaktionskomponente konnte bisher auf diese GUI Elemente keinen Einfluß nehmen, da sie keine Interaktionen mit dem Benutzer realisieren und deshalb nicht über Interaktionsformen erreichbar waren.

Probleme bei einem Austausch des GUI Toolkits treten nur in solchen Fällen auf, bei denen das neue Widget ein Manipulatorinterface nicht mehr unterstützen kann und es gleichzeitig kein anderes Widget gibt, welches die gleiche Interaktion realisiert und das Manipulatorinterface unterstützt. In der Konsequenz muß hier entweder die Interaktionskomponente so angepaßt werden, daß der betreffende Manipulator nicht mehr verwendet wird oder die Benutzungsoberfläche muß so geändert werden, daß ein anderes oder zusätzliches Widget die gewünschte Anzeige übernimmt. Dies wird jedoch nicht immer möglich sein.

Da die Manipulatoren genau wie die Präsentationsformen vom konkreten Widget abstrahieren, tritt das Problem der Granularität der Abstraktion auf. Auf der einen Seite sollen möglichst viele Eigenschaften der Widgets manipulierbar sein. Dies führt zu einer feinen Granularität, die eine große Zahl von Manipulatorinterfaces zur Folge hat. Je feiner jedoch die Granularität wird, desto schwächer wird die Abstraktion vom konkreten GUI Toolkit. Um möglichst viele GUI Toolkits unterstützen zu können und die Portierung der Präsentationsformen zu erleichtern, ist jedoch ein hoher Grad der Abstraktion sinnvoll. Erreichen können wir dieses nur durch eine kleine Anzahl von Manipulatorinterfaces, die dann nur eine Obermenge der Eigenschaften von Widgets möglichst vieler GUI Toolkits abbilden.

Hier gilt es letztlich, einen sinnvollen Kompromiß zu finden, was ohne weiteres möglich sein sollte, da für semantisches Feedback und dynamisches Oberflächenverhalten zur Laufzeit unserer Meinung nach folgender Satz von Eigenschaften ausreichen könnte:

Manipulator - Interface	Widgeteigenschaften
`mpDimension`	Größe und Position
`mpColor`	Hintergrundfarbe, Vordergrundfarbe und Selektionsfarbe
`mpFont`	Fontname, Fontgröße und Stil
`mpText`	Überschriften und Labeltexte
`mpImage`	Bitmapbild

Tabelle 6-1: Elementare Manipulatoren

Die Praxis muß zeigen, ob der oben genannte Satz von Manipulatoren ausreichend und angemessen ist. Falls auf besondere Eigenschaften eines Widgets, die nicht durch den vorhandenen Satz von Manipulatoren abgedeckt werden, zugegriffen werden muß, können einer Anwendung problemlos eigene Manipulatoren hinzugefügt werden. Diese Art der Erweiterung hat den Vorteil, daß bei einem Austausch des GUI Toolkits zusätzlicher Aufwand ausschließlich auf der Ebene der Präsentationsformen anfällt und nicht alle Interaktionskomponenten der Werkzeuge betrifft.

6.2 Zusammengesetzte Interaktionsformen

Die Verwendung von Interaktionsformen und Präsentationsformen innerhalb des JWAM Rahmenwerks hat sich bei der Umstellung der Widgets von AWT nach Swing bewährt. Ohne größere Probleme konnten die vorhandenen Präsentationsformen umgestellt werden, und es mußten kaum Änderungen an den Werkzeugen vorgenommen werden. Mit Einführung von Swing stehen nun auch komplexere Widgets wie Tabellen und Trees zur Verfügung. In dem fünften Kapitel haben wir gezeigt, wie diese beiden neuen Widgets in das bestehende Rahmenwerk integriert werden konnten. Diese neuen Widgets bieten jedoch im Gegensatz zu den einfacheren Eingabewidgets ein vielfältiges Spektrum an Möglichkeiten der Interaktion. Es scheint sich so zu verhalten, daß das bisherige Interaktions / Präsentations Konzept den neuen Möglichkeiten nicht gerecht werden kann. Am Beispiel der Interaktionsform „1 aus N Auswahl" wollen wir dies näher erklären. Die dieser Interaktionsform eigene Schnittstelle[18] haben wir in der folgenden Tabelle 6-2 aufgeführt. Bei Durchsicht der Methoden fällt auf, daß sie sich in zwei Gruppen einteilen lassen. Die ersten fünf Methoden bis einschließlich `attachSelectCommand` dienen zur Sondierung und Beobachtung der Benutzerinteraktionen. Die letzten vier Methoden dienen zum Setzen der Auswahl durch das Werkzeug.

[18] Aus Gründen der Übersichtlichkeit haben wir an dieser Stelle nur einen Ausschnitt der Schnittstelle aufgeführt.

Schon bei der Entwicklung des Konzeptes der Manipulatoren im vorigen Abschnitt haben wir dargelegt, daß sich der Kontrollfluß bei den Manipulatoren von dem Kontrollfluß in einer Interaktionsform unterscheidet. Bei einer Interaktionsform gibt es zwei unterschiedlich gerichtete Kontrollflüsse. Der eine Kontrollfluß läuft wie bei den Manipulatoren von der Interaktionsform zum Widget, und der andere Kontrollfluß läuft vom Benutzer über das Widget zur Interaktionsform. Die letzten vier Methoden der Auflistung realisieren dabei den Kontrollfluß vom Werkzeug zum Benutzter und die ersten fünf Methoden den umgekehrten Weg.

Methode	Beschreibung
`isSelected ()`	Ist ein Eintrag selektiert?
`selectionIndex ()`	Position des selektierten Eintrages?
`selection ()`	Selektierten Eintrag zurückgeben.
`elementCount ()`	Anzahl der Einträge zurückgeben.
`attachSelectCommand (cmd)`	Command für Änderungen anmelden.
`select (index)`	Selektieren eines Eintrages.
`add (item)`	Einen Eintrag zur Auswahl hinzufügen.
`remove (index)`	Eine Eintrag aus der Auswahl entfernen.
`removeAll ()`	Alle Einträge entfernen.

Tabelle 6-2: Auszug aus der Schnittstelle der Interaktionsform `if1fromNSelection`

Für die Realisierung einer Auswahl eines Elementes aus N zur Verfügung stehenden Elementen, bietet uns Swing mehrere Widgets :

- `JList`
- `JComboBox`
- `JMenu` / `JMenuBar` / `JPopupMenu`
- `JRadioButtons` mit einer `ButtonGroup`
- `JTable`
- `JTree`

Diese sechs Widgets könnten mit der Interaktionsform „1 aus N Auswahl" interagieren, wenn von ihnen Unterklassen abgeleitet werden, die das Präsentationsinterface `pf1fromNSelection` implementieren. Doch dies ist leider nicht immer zufriedenstellend möglich. Eine Tabelle läßt sich beispielsweise hervorragend für eine 1 aus N Auswahl nutzen, obwohl sie normalerweise eine N x M Matrix darstellt. Die in der Abbildung 6-6 dargestellte Bearbeitung der Druckerwarteschlange unter Windows 98 ist ein schönes Beispiel für eine so geartete Verwendung einer Tabelle.

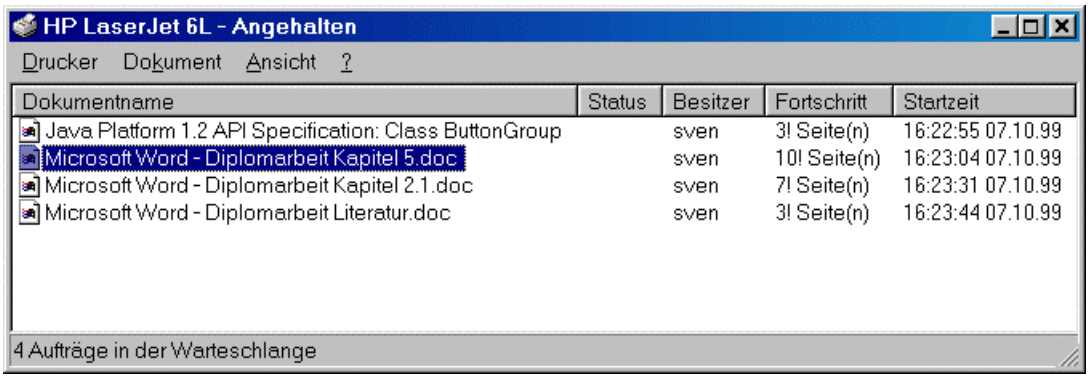

Abbildung 6-6: Darstellung der Druckerwarteschlange unter Windows 98

Obwohl die Tabelle mehrere Spalten hat, kann immer nur ein Dokumentenname zur Zeit ausgewählt und bearbeitet werden. Die weiteren Spalten enthalten zusätzliche Informationen über das Dokument und den Druckstatus. Diese Informationen dienen zur Vereinfachung der Auswahl und erhöhen die Benutzerfreundlichkeit der Auswahl. Betrachtet man jedoch die bereits dargestellte Schnittstelle der Interaktionsform zur 1 aus N Auswahl, so ist die Verwendung einer Tabelle im JWAM Rahmenwerk mit einer 1 aus N Auswahl momentan kaum sinnvoll möglich. Es ist über die einfache `add` Methode nicht möglich, mehr als einen String zu übergeben und so Zeilen mit mehreren Spalten aufzubauen. Es wäre in diesem Fall zwar vorstellbar, ein String mit Trennzeichen zu übergeben, wobei die Trennzeichen die einzelnen Spalten im String markieren. Dies setzt aber Kenntnisse in der Interaktionskomponente über das konkret verwendete Widget voraus, da zum Beispiel ein einfaches Listen Widget mit diesen Trennzeichen nichts anfangen könnte. Da der große Vorteil des Interaktions / Präsentations Musters die hohe Abstraktion von GUI Toolkits ist, sollten Interaktionskomponenten möglichst keine Kenntnisse und Abhängigkeiten von den konkreten Widgets haben. Aus diesem Grund würde die Verwendung eines Strings mit Trennzeichen keine gute Lösung sein.

Es ist auch vorstellbar, einen Tree zur Auswahl zu verwenden. Ein Tree bietet viele Möglichkeiten zur übersichtlichen Strukturierung der Auswahl und kann so die Navigation beschleunigen und vereinfachen. Was bei der Tabelle noch ansatzweise mit einem codierten String vorstellbar ist, ist bei einem Tree fast unmöglich. Ein Tree laßt sich nicht mehr sinnvoll über eine `add` Methode mit einem String als Parameter aufbauen. Ähnliches gilt für ein Auswahlmenü, das Icons, Tastatur - Short-Cuts und baumartige Untermenüs enthalten kann, die sich wiederum kontextsensitiv verhalten sollen und deshalb oft zur Laufzeit dynamisch aufgebaut, aktiviert, deaktiviert oder verändert werden müssen.

Ein weiteres Problemfeld stellen statische Radiobuttons dar. Die Anzahl der Radiobuttons und die Texte der Auswahl werden in der Regel zur Designzeit statisch in der Benutzungsschnittstelle abgelegt, da es momentan keine Möglichkeit im JWAM Rahmenwerk gibt, dynamisch neue Widgets der Benutzungsoberfläche hinzuzufügen oder zu entfernen. Wenn nun Radiobuttons zur 1 aus N Auswahl benutzt werden, kann das Werkzeug keinen Einfluß mehr auf die Auswahl nehmen. Die Methoden zum Verändern der Auswahl sind überflüssig, und es ist nicht definiert, was bei einem Aufruf dieser Methoden passiert. In solch einem Fall würde auch der Austausch der Radiobuttons Änderungen an der Interaktionskomponente nach sich ziehen.

Als Fazit können wir festhalten, daß mehrere Widgets eine 1 aus N Auswahl benutzerfreundlich realisieren könnten. Jedoch ist ihre technische Handhabung von der Werkzeug-

seite unterschiedlich und über die bisher vorhandene Schnittstelle der Interaktionsform nicht abbildbar.

Ein sehr einfacher Kompromiß wäre es, mehrere Interaktionsformen für die 1 aus N Auswahl zu schreiben, die jeweils ein Widget oder eine Gruppe von Widgets abdecken. Dies würde jedoch zu einer Vielzahl von spezialisierten Interaktionsformen führen, die den Aufwand für eine Portierung auf ein anderes GUI Toolkit stark erhöhen würden. Dieser Ansatz stellt für uns keine ernsthafte Lösung dar, weil die Interaktionsformen dann stark von den konkreten Widgets abhängig wären und es für einige Fälle auf eine 1:1 Beziehung zwischen Interaktionsform und Widget hinausläuft. Die Abstraktionsebenen[19] zwischen dem GUI Toolkit und dem Werkzeug wären faktisch aufgehoben und ein Austausch des GUI Toolkits oder sogar nur ein einfaches Redesign der Oberfläche würde zu Änderungen an den Interaktionskomponenten führen.

Wir möchten deshalb einen anderen Ansatz vorstellen. Zu diesem Zweck haben wir uns näher mit den biologischen Grundlagen von Interaktion auseinander gesetzt. Keil - Slawik diskutiert in [Keil90] die biologischen Grundlagen von Interaktion und die daraus resultierenden Konsequenzen für die Gestaltung interaktiver Systeme. Er führt dazu die Untersuchungen des Biologen Jakob von Uexküll an, der mit dem in Abbildung 6-7 dargestellten Funktionskreis gezeigt hat, daß es für ein Organismus nicht nur eine (Um-) Welt gibt, sondern eine Vielzahl verschiedener Umwelten.

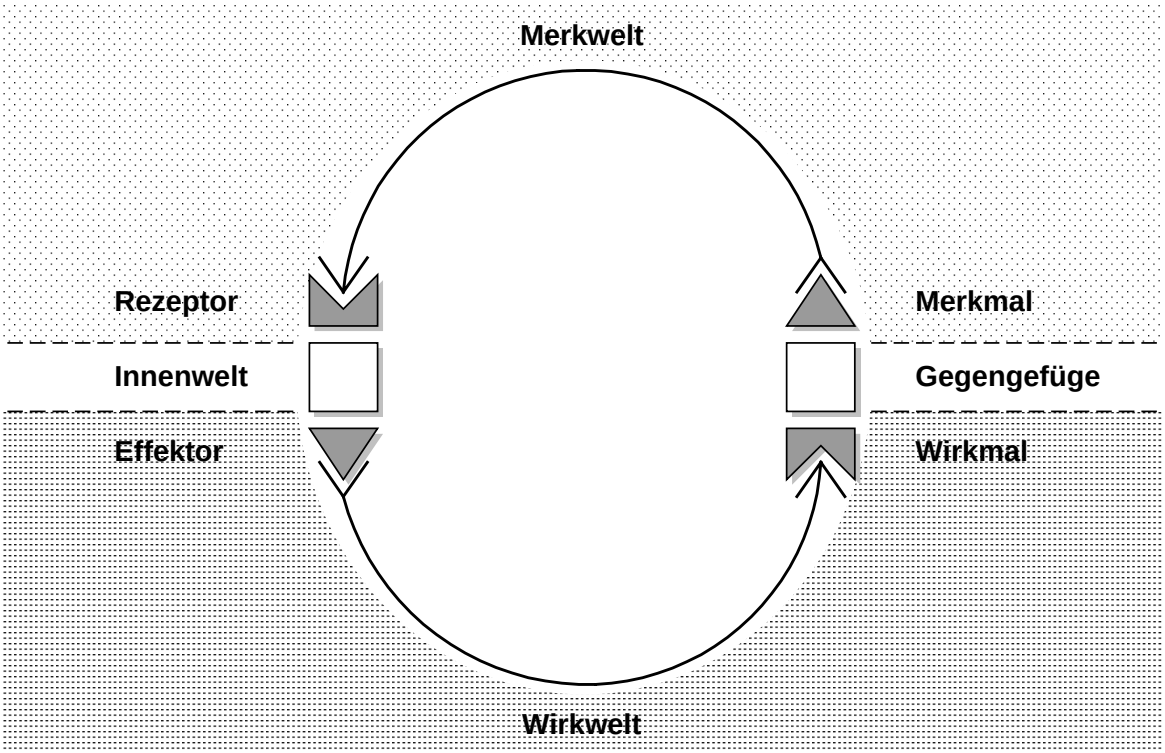

Abbildung 6-7: Funktionskreis nach Uexküll

Die Umwelten unterscheiden sich, je nachdem über welche Merkzeichen und welche Wirkzeichen der Organismus verfügt. Ein Organismus nimmt nur diejenigen Reize wahr, für die er ein Merkzeichen besitzt, und er kann nur insoweit auf Objekte Einfluß nehmen,

[19] Siehe auch Kapitel 5.1

wie er über Wirkzeichen verfügt. Objekte hingegen besitzen Merkmale und Wirkmale. Uexküll versteht unter Wirkmalen diejenigen Eigenschaften von Objekten, die es erlauben, sie auf eine bestimmte Art und Weise zu benutzen. Es besteht dabei eine enge Verbundenheit zwischen Wirkmalen und Merkmalen die er folgendermaßen erklärt:

> *Da die Eigenschaften eines Objekts, die als Merkmal oder Wirkmal fungieren, über den Bau des Objekts (Gegengefüge) miteinander verbunden sind, verändern die Wirkmale die Merkmale, indem jetzt andere Merkmale hervorgehoben oder ignoriert werden beziehungsweise mit einer anderen Bedeutung wahrgenommen werden. Wirkmale und Merkmale verschmelzen zu einer Einheit.*

Wenn ein handelndes Subjekt zu einem Objekt in Beziehung tritt, prägt es diesem Objekt seine Merk- und Wirkzeichen auf, so daß diese verschmolzen werden mit den Merk- und Wirkmalen des Objektes und es dadurch für das handelnde Subjekt neue Eigenschaften erhält.

Wenn man diese Erkenntnisse auf die Benutzung von Interaktionsformen durch Werkzeuge überträgt, so besitzen Interaktionsformen auch Merkmale und Wirkmale. Die Merkmale lassen sich über sondierende Methodenaufrufe in Erfahrung bringen, und zustandsverändernde Methodenaufrufe[20] stellen die Wirkmale dar. Die „Nutzung der Wirkmale" hat über die Widgets auch einen rückkoppelnden Einfluß auf die Merkmale der Interaktionsform. Die handelnden Subjekte, hier die Werkzeuge, haben durch ihre unterschiedlichen Ausprägungen verschiedene Merkzeichen und Wirkzeichen. Durch Benutzung der Interaktionsformen haben sie eine eigene Sicht auf die Benutzungsschnittstelle, die ihre Umwelt repräsentiert. Diese Überlegungen lassen sich auch auf die Beziehung zwischen Interaktionsform und Präsentationsform übertragen. Die Interaktionsformen sind ebenfalls handelnde Subjekte, die mit den Präsentationsformen, welche einen Teil ihrer Umwelt bilden, umgehen. Die Präsentationsformen haben dabei ihre eigenen Merkmale und Wirkmale, die sich unter Einfluß ihrer Umwelt, hier Interaktionsformen und Benutzer, verändern. Bezogen auf die Werkzeugkonstruktion lassen sich mindestens drei Funktionskreise ausmachen, welche untereinander rückgekoppelt sind. Die Abbildung 6-8 illustriert diese Rückkoppelungskreise. Durch die beiden linken gekoppelten Funktionskreise wird deutlich, warum die Interaktionskomponente eines Werkzeuges nicht die Sicht auf die Präsentationsformen hat, die nötig wäre, um die oben genannten Probleme zu umgehen. Die Interaktionsformen verschmelzen die Merkmale und Wirkmale der zugehörigen Präsentationsformen zu einer neuen Sicht auf die Benutzungsschnittstelle mit neuen Eigenschaften. Diese festen Sichten auf die Benutzungsschnittstelle wurden von den Entwicklern bei der Realisierung des Rahmenwerks fest vorgegeben, indem die Interaktionsformen bestimmte Merkmale und Wirkmale der Präsentationsformen zusammenfügen und unter einer neuen Schnittstelle zur Verfügung stellen. Dadurch aber kann die Interaktionskomponente nicht ihren Satz von Merkzeichen und Wirkzeichen vollständig anwenden und kombinieren.

[20] Die Interaktionsformen sind normalerweise zustandslos. Nur die angebundenen Präsentationsformen speichern Zustandsänderungen.

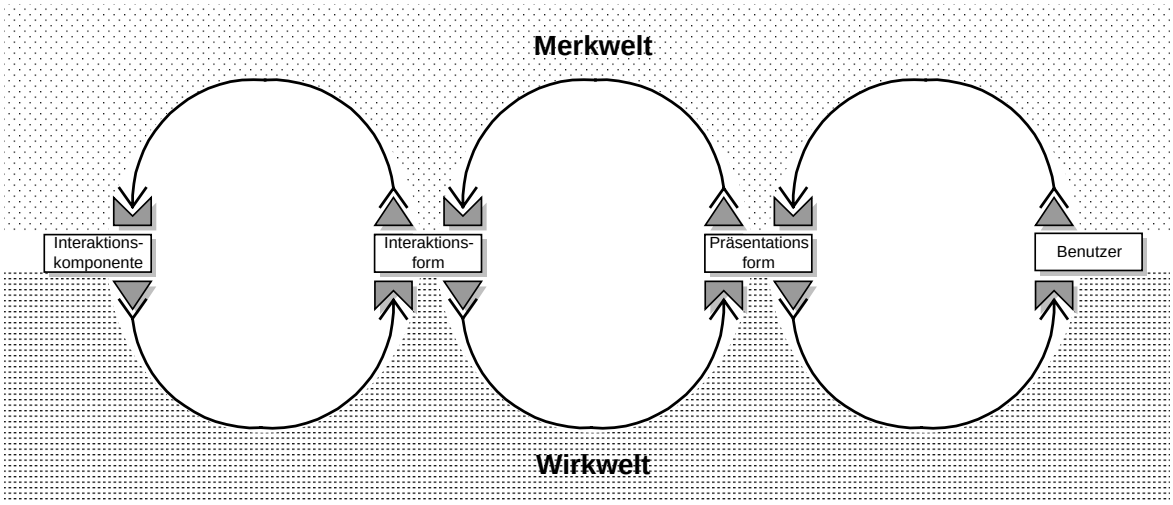

Abbildung 6-8: Funktionskreise im WAM Kontext

Keil - Slawik zieht für die Konstruktion von interaktiver Software folgende Konsequenz:

> *Eine Konsequenz, die sich daraus für die Gestaltung interaktiver Systeme ableiten läßt, ist die Tatsache, daß man durch keine Gestaltungsmaßnahme und keine Vorschrift oder Anweisung die Macht hat, das, was der Benutzer tut oder tun soll, annähernd vollständig vorherzubestimmen beziehungsweise zu kontrollieren. Wohl kann man Möglichkeiten eröffnen und durch eine geschickte Gestaltung ein Gegengefüge bereitstellen, daß es ihm erlaubt, seine Wirk- und Merkmale auszuprägen. Aber ebenso, wie man nicht auf jedem Gegenstand gleich gut sitzen kann, das heißt ihm das entsprechende Wirkmal aufprägen kann, sind mache physische Anordnungen besser geeignet, als Wirk- und Merkmal zu fungieren, als andere.*

Übertragen auf das Muster der Interaktions- und Präsentationsform stellen sich für uns nun zwei Lösungsmöglichkeiten dar. Entweder wird die Abstraktion mit Hilfe der Interaktionsformen aufgegeben, das heißt die Werkzeuge benutzen wieder den von uns in Kapitel 4.2 diskutierten Ansatz der Interaktionstypen mit allen Vor- und Nachteilen, oder es wird versucht, die feste Kombination von Merkmalen und Wirkmalen, welche die Interaktionsformen zusammenfassen, aufzugeben.

Wir möchten deshalb einen Konstruktionsvorschlag für zusammengesetzte Interaktionsformen machen. Mit Hilfe von zusammengesetzten Interaktionsformen soll es für ein Werkzeug möglich sein, die Handhabung der Wirkmale und Merkmale einer Präsentationsform nahezu beliebig zu kombinieren. Weiterhin soll dieses Konzept das bisherige Interaktionsform / Präsentationsform Muster erweitern, wobei die bereits existierenden Interaktionsformen wenn möglich erhalten bleiben.

Wie bereits dargelegt, lassen sich die Schnittstellen der Interaktionsformen in einen sondierenden Teil inklusive Beobachteranmeldung und einen Teil zum Setzen von Werten aufteilen. Mit dem sondierenden Teil lassen sich die Merkmale einer Interaktion bestimmen, und die Methoden zum Setzen von Werten fungieren als Wirkmale der Interaktion. Jede Interaktionsform läßt sich so in zwei Hälften aufteilen, eine Wirkmal - Interaktionsform und eine Merkmal - Interaktionsform. Zusammengesetzte Interaktionsformen werden aus diesen zwei Hälften in der Interaktionskomponente eines Werkzeuges

zusammengesetzt. Genaugenommen benutzt die Interaktionskomponente in unserem Entwurf nur die beiden Hälften. Neben einer Benutzung der entsprechenden Interaktionsanteile sind auch Lösungen denkbar, die auf Adapterklassen oder Core - Reflection aufsetzen. Aufgrund der immer noch verbesserungsbedürftigen Performance von Java (siehe [Half98] und [Kopp98]) und der besseren Handhabung sollte unserer Meinung nach auf solche Konstrukte zur Erzeugung der zusammengesetzten Interaktionsformen verzichtet werden. Die Abbildung 6-9 erläutert die Klassenstruktur des von uns favorisierten Entwurfs.

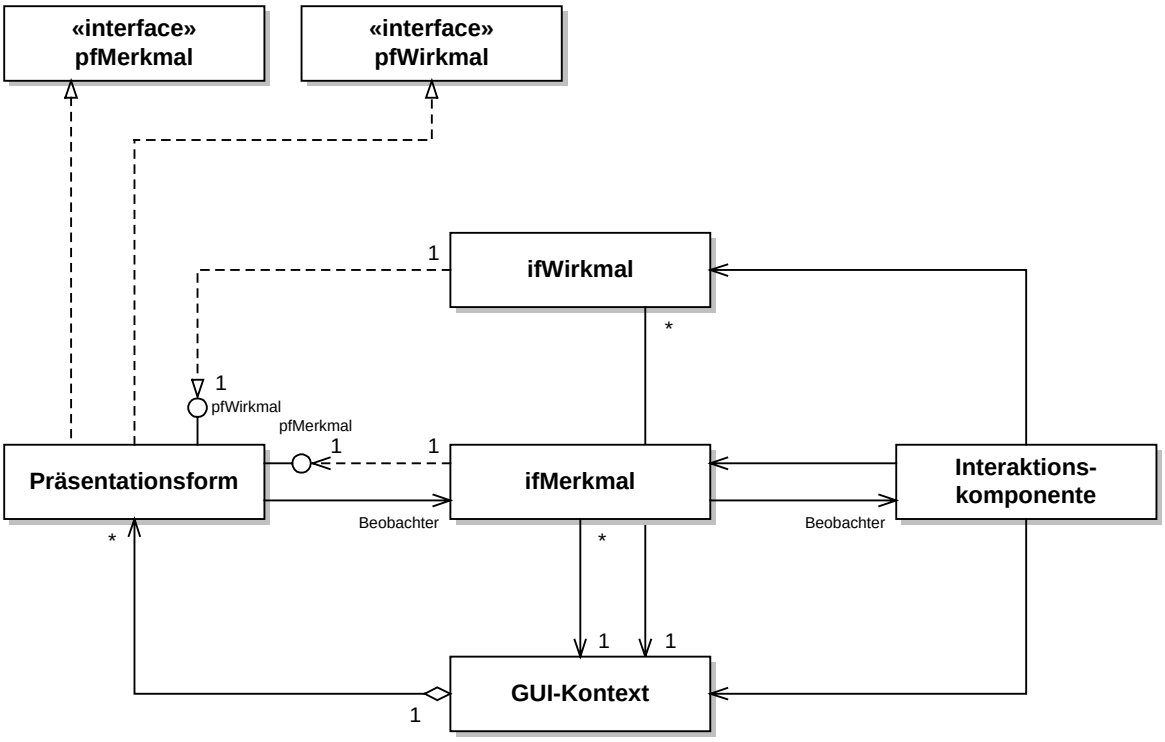

Abbildung 6-9: Klassenstruktur von zusammengesetzten Interaktionsformen

Dieser Entwurf paßt sich nahtlos in das bestehende Interaktionsform / Präsentationsform Muster ein. Die neuen Anteile einer Interaktionsform können zusätzlich zu den bestehenden Interaktionsformen eingeführt werden oder sie ganz ersetzen. Es bietet sich an, die bestehenden Interaktionsformen in ihren Wirkmal- und ihren Merkmal - Anteil aufzuteilen. Auch die Interfaces der Präsentationsformen müssen analog zu den Interaktionsformen in Interfaces für Wirkmale und Interfaces für Merkmale aufgeteilt werden. Eine Präsentationsform implementiert dann mehrere verschiedene Wirk- und Merkmalinterfaces. So würde zum Beispiel ein Tabellenwidget unter anderem auch das Merkmalinterface für die 1 aus N Auswahl implementieren, da es sich, wie schon beschrieben, auch gut für diesen Zweck benutzen läßt.

Mit Hilfe dieses Ansatzes ist es nun möglich eine Tabelle für eine 1 aus N Auswahl zu nutzen. Die Interaktionskomponente verwendet dazu ein `tableDataModel` mit einer Wirkmal - Interaktionsform für tabellarische Präsentationsformen und eine Merkmal-Interaktionsform für die 1 aus N Auswahl. Wobei die Merkmal - Interaktionsform nur die sondierenden Methoden der jetzigen Interaktionsform 1 aus N Auswahl enthält.

In diesen Ansatz lassen sich auch die Manipulatoren integrieren, da die Manipulatoren nur eine technische Interaktion zwischen dem Werkzeug und einem Widget realisieren nicht aber eine Interaktion mit dem Benutzer. Sie können durch eine Wirkmal - Interaktions-

form ersetzt werden. In diesem Fall nutzt die Interaktionskomponente keine Merkmal-Interaktionsform, da von außen keine Änderungen an dem Widget durch den Benutzer möglich sind. Die schon erwähnten Vor- und Nachteile des Manipulator - Konzeptes gelten aufgrund der ähnlichen Konstruktion auch für zusammengesetzte Interaktionsformen. Durch die Aufteilung der Interaktionsformen kann es je nach Implementierung des Interaktionsform / Präsentationsform Musters zu einer Verdoppelung der Objektanzahl für die GUI Anbindung im Werkzeug kommen. Dies läßt sich jedoch durch Anwendung von Rollenkonzepten und geschickte Programmierung vermeiden. Im nachfolgenden Kaptitel 6 werden wir näher darauf eingehen. Diese Art der Aufteilung von Interaktionsformen ist bisher noch nicht in das JWAM Rahmenwerk integriert worden. Prototypen und Referenz-implementierungen müssen deshalb in Zukunft zeigen, ob dieses Konzept gut handhabbar ist und sich im Einsatz bewährt.

7 Rollenkonzept und Widgets

Mit diesem Kapitel wollen wir unsere technische Diskussion bezogen auf die Verwendung von GUI Komponenten in Rahmenwerken nach dem Werkzeug- und Material - Ansatz abschließen. Diesen Abschluß wollen wir finden, indem wir eine andere Sicht auf das von uns im Abschnitt 5.3.2 vorgestellte Entwurfsmuster anbieten. Zur weiteren Motivation dieses Abschnittes führen wir an dieser Stelle ein Zitat aus [Rieh97c] an.

> *I present patterns using role diagrams, because role diagrams can be more easily composed than class diagrams. They are more abstract though, and therefore only complement the original class diagram based description. I hope that pattern descriptions based on role diagrams let us explain frameworks more easily as a composition of applied patterns than class diagram descriptions let us do.*

Die Tatsache, daß wir dem Entwurfsmuster der Interaktions- und Präsentationsformen weitere Objekte mit eigenen Aufgaben hinzugefügt haben, fördert nicht die Übersichtlichkeit der Konstruktion. Um das Verständnis der Anbindung von Widgets an ein Werkzeug zu fördern, werden wir nun in Anlehnung an Riehle das Entwurfsmuster der Interaktions- und Präsentationsformen auf der Basis von Rollen beschreiben. Dabei haben wir als einen einführenden Teil die Beschreibung des MVC Musters kurz aufgegriffen, bevor wir auf das angesprochene Muster überleiten.

Das MVC Muster wird im allgemeinen als ein zusammengesetztes Muster bezeichnet und nicht in eine der Basiskategorien eingeordnet, da es selbst mehrere unterschiedliche Entwurfsmuster enthält. Wir werden hier nicht den Aufbau und die Vorzüge dieses Musters selbst erläutern, da wir dieses Muster bereits im Abschnitt 3.1.1 vorgestellt haben. Die Abbildung 7-1 zeigt nun aus der Blickrichtung der zu unterscheidenden Rollen das Zusammenspiel innerhalb dieser Konstruktion. Ein solches Rollendiagramm bildet häufig die Basis für die Entwicklung der Abhängigkeiten zwischen den einzelnen Rollen.

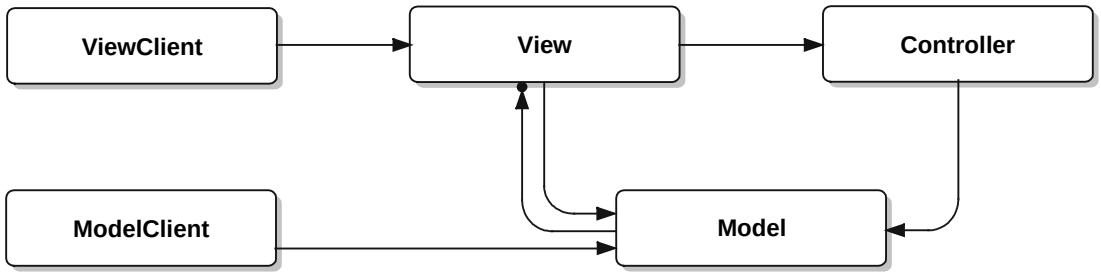

Abbildung 7-1: Rollendiagramm des MVC Musters[21]

An diesem einfachen Beispiel können wir bereits erkennen, daß das Rollendiagramm nicht unmittelbar von der gewählten technischen Implementierung beeinflußt wird. Diese Tatsache erleichtert es einem Entwickler, Zusammenhänge in einem Rahmenwerk oder in einer bestimmten Konstruktion zu erkennen. Durch die Vielzahl an existierenden

[21] Ohne Änderungen aus [Rieh97c] übernommen.

Möglichkeiten, ein solches Konzept umzusetzen, sind hier teilweise deutliche Unterschiede zwischen dem Klassendiagramm und dem Rollendiagramm eines Musters zu erwarten. Da es sich bei dieser Darstellung nicht um eine exklusive Wahl handelt, fordert Riehle, daß ein Rollendiagramm als Ergänzung zu einem Klassendiagramm verstanden werden soll. So kann ein Entwickler sich über eine abstrakte Ebene, dem Rollendiagramm, einen leichteren Zugang zu der tatsächlichen technischen Realisierung, dem Klassendiagramm, verschaffen. Komplettiert wird dieses System durch eine Matrix, welche die Abhängigkeiten der Rollen untereinander aufzeigt. Im Rahmen des hier als Einführung gedachten Beispiels ist in Tabelle 7-1 diese Matrix für das MVC Muster dargestellt.

	ModelClient$_{MVC}$	**Model**$_{MVC}$	**ViewClient**$_{MVC}$	**View**$_{MVC}$	**Controller**$_{MVC}$
ModelClient$_{MVC}$	■		▒		
Model$_{MVC}$		■			
ViewClient$_{MVC}$	▒		■		
View$_{MVC}$				■	▒
Controller$_{MVC}$				▒	■

Tabelle 7-1: Rollenbeziehungen im MVC Muster[22]

Die für ein bestimmtes Muster identifizierten Rollen sind in der die Rollenbeziehungen darstellenden Matrix symmetrisch angeordnet. Dabei wird dem Namen der Rolle als Suffix das Kürzel des Musters hinzugefügt. Schwarz hinterlegte Felder innerhalb der Matrix bedeuten, daß eine Rolle A immer auch eine Rolle B impliziert. Bezogen auf unser Beispiel bedeutet dies, daß sich in der Matrix für diesen Fall ausschließlich die Diagonale ergibt. Sofern sich durch die Einnahme einer Rolle A durch ein Objekt die Einnahme anderer Rollen verbietet, sind diese Felder weiß hinterlegt. Innerhalb des MVC Musters bedeutet dies zum Beispiel, daß die Rollen der View und des Models nicht miteinander vereinbar sind. Die dritte und letzte Gruppe in dieser Art von Matrix wird durch Rollen repräsentiert, bei denen eine Zusammenlegung denkbar aber nicht zwingend notwendig ist.

Auch bei dem IAF / PF - Muster, dem wir uns nun zuwenden, können wir von einem zusammengesetzten Muster sprechen. So werden auf der Ebene der technischen Konstruktion zum Beispiel die folgenden Muster verwendet:

- Adaptermuster. Das aus einem GUI Toolkit stammende Widget wird von uns mit Hilfe einer Adapterklasse, der Präsentationsform, gekapselt, so daß nach außen hin nur eine innerhalb des Rahmenwerkes definierte Schnittstelle sichtbar ist. Desweiteren verwenden wir intern auch Adapterklassen, die eine Bindung zwischen einer Interaktionsstrategie und einer Präsentationsform ermöglichen.

[22] Ohne Änderungen aus [Rieh97c] übernommen.

- Strategiemuster. Wie wir in Kapitel 5 bereits beschrieben haben, ist es bei einigen Widgets nicht möglich, den Typ des Inhaltes von vorne herein festzustellen. Um trotzdem einen fachlich motivierten Umgang mit dem Inhalt solcher Widgets realisieren zu können, setzen wir Strategiemuster ein, über deren Implementierung das Verhalten definiert wird.
- Reaktionsmuster. Da innerhalb der Konstruktion auf eine lose Koppelung der Komponenten Wert gelegt wird, kommen diese Muster an den jeweiligen Schnittstellen der einzelnen Komponenten zum Einsatz.

Der nächste Schritt in der Beschreibung dieses Musters besteht für uns nun darin, die einzelnen Rollen zu identifizieren und deren Zusammenspiel darzulegen. Diese Zusammenhänge haben wir in der nachstehenden Abbildung 7-2 aufgezeigt.

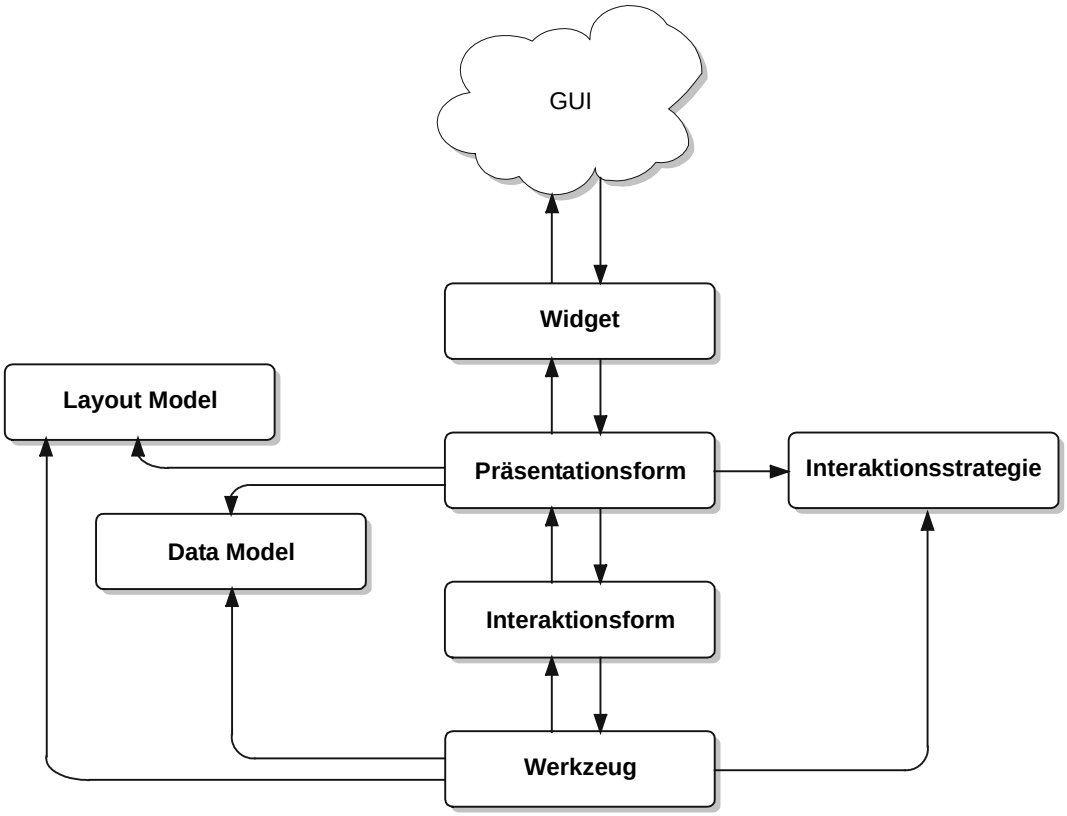

Abbildung 7-2: Rollendiagramm im IAF / PF Muster

Die Zentrale Rolle innerhalb dieses Musters wird unserer Meinung nach durch die Präsentationsform definiert. So besteht ihre Aufgabe zum einen darin, Ereignisse aus dem Bereich des GUI umzusetzen und über die Interaktionsform an das Werkzeug weiterzuleiten und zum anderen muß sie den über die Interaktionsform definierten Umgang in technische Aufrufe am Widget umsetzen. Je nachdem, aus welcher Blickrichtung wir eine Präsentationsform betrachten, ergibt sich somit ein unterschiedlicher Umgang.

Betrachten wir die in der Tabelle 7-2 dargestellten Abhängigkeiten der einzelnen Rollen untereinander, so können wir zwei größere Cluster ausmachen. Die erste Gruppe betrifft die von uns eingesetzten Modelle, wobei wir nach wie vor die Begriffsbildung aus dem fünften Kapitel für ein Modell zu Grunde legen. Bei einer Implementierung ist es durchaus möglich, das `dataModel` und das `layoutModel` zusammen zu fassen. Im Zusammenhang mit der Anbindung einer Datenbank kann es durchaus sinnvoll sein, in

einem Modell direkt die entsprechenden Methoden in der Funktionskomponente aufzurufen, die für die Lieferung der gewünschten Werte zuständig sind. Wichtig ist an dieser Stelle, daß keine Wege eingebaut werden, die den eigentlichen Kontrollfluß in einem Werkzeug umgehen. Generell ist es unserer Meinung nach aufgrund der angestrebten Übersichtlichkeit der Konstruktion aber empfehlenswert, diese Modelle getrennt zu implementieren.

	DataModel$_{IAF/PF}$	**LayoutModel**$_{IAF/PF}$	**Widget**$_{IAF/PF}$	**Präsentationsform**$_{IAF/PF}$	**Interaktionsform**$_{IAF/PF}$	**Interaktionsstrategie**$_{IAF/PF}$	**Werkzeug**$_{IAF/PF}$
DataModel$_{IAF/PF}$	■	▒					
LayoutModel$_{IAF/PF}$	▒	■					
Widget$_{IAF/PF}$			■	▒	▒		
Präsentationsform$_{IAF/PF}$			▒	■	▒		
Interaktionsform$_{IAF/PF}$			▒	▒	■		
Interaktionsstrategie$_{IAF/PF}$						■	
Werkzeug$_{IAF/PF}$							■

Tabelle 7-2: Rollenbeziehungen im IAF / PF Muster

Bei der Betrachtung des zweiten Clusters stellen wir fest, daß wir in der Lage sind, in einem Objekt die Aufgaben des Widgets, der Präsentationsform und der Interaktionsform unterzubringen. Die Zusammenlegung des Widgets mit der Präsentationsform ist hierbei keine neue Idee, sondern fußt auf den Prinzipien der Realisierung des IAF / PF - Musters. Die Interaktionsform auf der anderen Seite repräsentiert ein zustandsloses Konzept und definiert den Umgang eines Benutzers mit einem Widget. Solange wir bei einer Realisierung berücksichtigen, daß eine Interaktionskomponente nur über den Umgang eines Benutzers an die Präsentationsform gebunden wird, können wir ohne weiteres die Präsentations- und die Interaktionsform zusammenfassen. Allerdings verlieren wir bei einer derartigen Konzeption die Möglichkeit, mehrere Präsentationsformen an eine Interaktionsform zu binden. Da diese Möglichkeit aber nur in wenigen Fällen Anwendung findet, stellt dieser Verlust keinen entscheidenden Nachteil dar.

Abschließend fällt bei den eben beschriebenen Gruppen auf, daß sich deutliche Ähnlichkeiten zu der Struktur des MVC Musters ergeben. So finden wir auch in dem IAF / PF Muster eine Sorte von Komponenten, welche die Aufgabe haben, die darzustellenden Werte zur Verfügung zu stellen. Zudem haben wir mit der Präsentationsform einen Bestandteil, der größtenteils die gleichen Aufgaben übernimmt, wie eine View. Und

schließlich haben wir auch noch die Möglichkeit, der Präsentationsform eine Strategie zur Seite zu stellen, die Teile der Interaktion mit dem Benutzer realisiert, einem Controller nicht unähnlich. Die Unterschiede liegen vornehmlich in der Berücksichtigung des Umganges eines Benutzers mit einem Widget durch das IAF / PF Muster.

Insgesamt betrachtet entsteht für uns der Eindruck, daß eine Beschreibung des IAF / PF Musters mit Hilfe von Rollen vorteilhaft ist. Zum einen erhalten wir mit dem Rollendiagramm und der die Rollenbeziehungen beschreibenden Matrix wertvolle Bestandteile für die Dokumentation eines Rahmenwerkes. Zum anderen ermöglicht uns diese Sichtweise einen tieferen Einblick in mögliche Konstruktionen dieses Musters.

8 Zusammenfassung und Ausblick

In den vorangegangenen Kapiteln haben wir uns mit Möglichkeiten der Einbettung von Toolkits, welche die Erstellung und Benutzung einer grafischen Benutzerschnittstelle ermöglichen, in Rahmenwerke nach dem WAM Ansatz beschäftigt. Im Rahmen unserer Diplomarbeit haben wir am Beispiel von Tabellen und Trees eine Implementierung vorgeschlagen und im Zuge der Weiterentwicklung des am Arbeitsbereich entstandenen JWAM Rahmenwerkes auch realisiert. Aus der Situation heraus, daß zum einen einem Anwendungsentwickler alle Möglichkeiten eines gewählten Toolkits zur Verfügung gestellt werden sollten und zum anderen dabei ein softwaretechnisch vertretbarer Weg eingehalten werden muß, ordnen wir diesem Thema eine besondere Bedeutung auf dem Gebiet der Entwicklung und Pflege von Rahmenwerken zu. Insbesondere die verstärkte Berücksichtigung von ergonomischen Gesichtspunkten im Bereich des Designs von Oberflächen und die Integration von multimedialen Techniken, machen es für einen Anwendungsentwickler unabdingbar, alle Möglichkeiten eines Toolkits auszunutzen. Verstärkend kommt an dieser Stelle hinzu, daß kaum ein größeres Softwareprojekt ohne ein entsprechendes Anwendungsrahmenwerk realisiert wird. Aus diesem Grund haben wir die Kapitel 6 und 7 dazu benutzt, Erweiterungen des Konzeptes der Interaktions- und Präsentationsformen darzulegen, mit deren Hilfe wir auf eine geeignete Art und Weise mit beliebigen Widgets umgehen können.

Wir möchten diesen Abschnitt dazu nutzen, auf aktuelle Entwicklungen und mögliche Erweiterungen unserer Implementation vor dem Hintergrund des JWAM Rahmenwerkes hinzuweisen. Eine der wichtigsten Neuerungen basiert auf einer aktuellen Entwicklung für die kommende JWAM Version 1.4. Bereits in dieser Version hat man sich dazu entschlossen, die Interaktionsformen mit den Präsentationsformen zusammen zu legen. Diese Entwicklung entspricht unserer Empfehlung, die wir im Rahmen des siebten Kapitels vorgestellt haben. Da die von uns vorgestellte Konstruktion für Tabellen und Trees auf der Version 1.3 des JWAM Rahmenwerkes basiert, sind von uns die notwendigen Änderungen an der Implementation nachgeführt worden. Im wesentlichen haben diese Änderungen zu dem Wegfall der Präsentationsstrategie geführt. Die Grundlagen der Benutzung dieser Widgets durch einen Anwendungsentwickler haben sich allerdings nicht verändert. Aufgrund der Tatsache, daß wir unsere Implementation auf die Version 1.3 abgestützt haben und der zeitlichen Entwicklung haben wir darauf verzichtet, beide Versionen in dieser Arbeit vorzustellen.

Eine noch ausstehende Erweiterung der von uns implementierten Widgets betrifft den Umgang eines Benutzers über die Drag and Drop Technik, die von Martin Lippert in seiner Diplomarbeit [Lipp99] ausführlich vorgestellt worden ist. Diese Technik ist mittlerweile nicht mehr aus der Funktionalität von Oberflächen wegzudenken. Um also das Prinzip für die Einbindung von Tabellen und Trees abzurunden, ist es notwendig, diese Funktionalität in den beiden Widgets zu integrieren. Diese Erweiterungen sollten Hand in Hand gehen mit der eingehenden Überprüfung der technischen Benutzung der beiden Widgets, um unnötigen Mehraufwand vermeiden zu können. In diesem Zusammenhang bleiben auch die Erfahrungen abzuwarten, die Henning Wolf und Stefan Roock zur Zeit im Umgang mit Tabellen und Trees im Rahmen der Realisierung einer Auftragsbearbeitungssoftware sammeln.

Innerhalb des sechsten Kapitels behandeln wir Fragestellungen, bei denen es sich um Möglichkeiten handelt, die das Konzept der Interaktions- und Präsentationsformen

erweitern. Sofern diese Erweiterungen Berücksichtigung finden sollten, schlagen wir vor, die im Anschluß angeführte Reihenfolge der Realisierung zu wählen.

- Implementierung der elementaren Manipulatoren.
- Umsetzung des Prinzips der zusammengesetzten Interaktion.

Das Prinzip der Manipulatoren erinnert von der Seite der Konstruktion her sehr an die Interaktions- und Präsentationsformen. Da in Bezug auf diese Konstrukte bereits sehr viel Erfahrung bezüglich der technischen Implementierung gesammelt wurde, bietet sich dieser Aspekt für eine kurzfristige Realisierung an. Man wird an dieser Stelle auch sehr schnell feststellen können, welche Eigenschaften der Widgets von Werkzeugen manipuliert werden sollen, so daß eine schnelle Abschätzung des hierfür zu betreibenden Aufwandes möglich wird. Wir denken, daß mit dieser Erweiterung eine gute Möglichkeit besteht, die Möglichkeiten eines für ein Rahmenwerk gewählten Toolkits auszunutzen. Der im Kapitel 6.2 beschriebene Aufbau einer Struktur von Merk- und Wirkmalen, dient in erster Linie dazu, den Umgang mit den uns zur Verfügung stehenden Widgets besser realisieren zu können. Wenn wir die Sicht auf Widgets über einen spezifischen Umgang abstrahieren, erhalten wir als Resultat immer eine Gruppe von Widgets, auf die dieser Umgang zutrifft. In dieser Arbeit haben wir gezeigt, daß hiermit aber auch die Möglichkeit verloren geht, auf die Eigenheiten eines einzigen Widgets einzugehen. Aus diesem Grund erachten wir es als sinnvoll, auch diesen Ansatz zu verfolgen und die Handhabung über den Einsatz in einem Rahmenwerk zu überprüfen.

Wir teilen die von Dirk Riehle in [Rieh97c] vertretene Ansicht, daß das Verständnis der Zusammensetzung eines Rahmenwerkes gesteigert werden kann, wenn die angewandten Muster über das Konzept der Rolle zugänglich gemacht werden. Da wir denken, daß das Konzept der Interaktions- und Präsentationsformen mit zu den wichtigsten Mustern im Bereich der Entwicklung und Benutzung von Rahmenwerken zählt, haben wir hier die von Dirk Riehle vorgestellte Beschreibung von Mustern für die Interaktions- und Präsentationsformen nachempfunden. Ziel ist es hierbei für uns, das Verständnis auf der Seite eines Anwendungsentwicklers für dieses Muster zu stärken.

Wie eingangs dieser Arbeit erwähnt, haben wir in dieser Arbeit nur einen kleinen Ausschnitt aus dem Bereich der grafischen Benutzerschnittstellen behandelt. Gleichwohl sind wir der Meinung, daß die Einbettung von GUI Toolkits in Rahmenwerken eine große Bedeutung einnimmt. Die Vision einer einheitlichen Implementierung von Widgets für alle zur Verfügung stehenden Systemplattformen wird, bedingt durch große wirtschaftliche Interessen der Hersteller, wohl nie erfüllt werden. Im großen Maße wahrscheinlich ist auch, daß die Erstellung von grafischen Benutzerschnittstellen auch in der Zukunft einen großen Zeitanteil innerhalb eines Softwareprojektes in Anspruch nehmen wird. Somit besteht ein realistischer Anspruch darin, durch die Entwicklung geeigneter Konzepte die Wartbarkeit der Software im Bereich der Anbindung von GUI's zu erhöhen.

Begriffe

AWT	Abkürzung für das Abstract Windows Toolkit der Programmiersprache Java.
Callback	Unter einem Callback verstehen wir den Aufruf einer Methode, welcher die Rückkehr des Kontrollflusses nach sich zieht, nachdem dieser zum Beispiel an das GUI weitergegeben worden ist.
Event	Als Event werden Nachrichten bezeichnet, die von der grafischen Benutzerschnittstelle an die darunter liegende Software gesendet werden. So löst zum Beispiel die Betätigung eines Buttons durch einen Benutzer ein Ereignis aus, das an die dieses Ereignis verarbeitende Software gesendet wird.
GUI	Kurzform von Graphical User Interface. Hiermit wird in unserem Zusammenhang die Gesamtheit einer grafischen Benutzerschnittstelle bezeichnet.
GUI Toolkit	Hiermit werden Softwareprodukte bezeichnet, die einem Anwendungsentwickler die zur Erstellung eines GUI's benötigten Bausteine zur Verfügung stellen.
JDK	Kurzform für das Java Development Kit. Zusammen mit einer Versionsnummer wird hiermit eine Sprachversion der Programmiersprache Java bezeichnet.
JWAM	Diese Bezeichnung steht für das am Arbeitsbereich Softwaretechnik entwickelte Rahmenwerk, wobei das J für die Programmiersprache Java und WAM für das umgesetzte Leitbild (siehe unten) steht.
Look and Feel	Zu jedem verfügbaren Betriebssystem existiert eine Empfehlung der Hersteller bezüglich des Designs von grafischen Benutzerschnittstellen. Im Rahmen dieser Empfehlung wird auch das Aussehen der verwendbaren Widgets vor dem Hintergrund definiert, daß die für ein Betriebssystem geschriebenen Applikationen ein möglichst einheitliches Erscheinungsbild haben sollen.
Listener	Innerhalb von Java stellen Listener Methoden dar, die auf das eintreten von bestimmten Ereignissen warten und geeignet auf diese reagieren können.
MVC	Kurzform für das Model View Controller Muster, welches eine bestimmte Sicht auf die Aufteilung von Zuständigkeiten innerhalb von Software vertritt.
SWING	Dieser Name steht für ein von der Firma Sun entwickeltes GUI Toolkit.
WAM	Kurzform für das am Arbeitsbereich Softwaretechnik vertretene Leitbild Werkzeug, Automat und Material.
Widget	Hinter dieser Bezeichnung verbergen sich alle grafischen Elemente, aus denen eine Benutzerschnittstelle innerhalb eines Fensters zusammengesetzt wird. Beispiele hierfür sind Felder zur Texteingabe und Listboxen oder auch einfache Labels.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Berl93] Thomas Berlage: *Object-Oriented Application Frameworks for Graphical User Interfaces – The GINA Perspective*, GMD-Bericht Nr. 213, R. Oldenbourg Verlag, 1993

[BKWW98] Thom Blum, Doug Keislar, Jim Wheaton, Erling Wold*: Programming a Dynamic User Interface*, http://developer.java.sun.com/developer/technicalArticles/GUI/DynaGUI/index.html, Sun Microsystems Inc., 1998

[Blee99] Wolf-Gideon Bleek, Thorsten Görtz, Carola Lilienthal, Martin Lippert, Stefan Roock, Wolfgang Strunk, Ulfert Weiss, Henning Wolf: *Interaktionsformen zur flexiblen Anbindung von Fenstersystemen*, Mitteilung 285, Fachbereich Informatik, Universität Hamburg, 1999

[Booch94] Grady Booch: *Object-oriented analysis and design with applications*, second edition, The Benjamin/Cummings Publishing Company Inc., 1994

[Engl97] Robert Englander: *Developing Java Beans*, O'Reilly & Associates, 1997

[Flan98] David Flanagan: *Java in an Nutshell*, deutsche Ausgabe, 2. erweiterte Auflage, O'Reilly Verlag, 1998

[GHJV96] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software*, Addison-Wesley, 1996

[Görtz98] Thorsten Görtz: *Abstraktion der GUI-Komponente in einem objekt-orientierten Rahmenwerk*, Diplomarbeit am Fachbereich Informatik, Universität Hamburg, 1998

[Grand95] Sven Grand: *Trennung von Interaktion und Funktion in der „Werkzeug und Material"-Metapher*, Diplomarbeit am Fachbereich Informatik, Universität Hamburg, 1995

[Half98] Tom R. Halfhill: *Java – neun Wege aus der Performance-Falle*, Byte, deutsche Ausgabe, Juni, The McGraw-Hill Companies Inc., 1998

[Hol93] Ian Holyer: *Functional Programming with Miranda*, UCL Press, 1993

[HopUll94] John E. Hopcroft, Jeffrey D. Ullmann: *Einführung in die Automaten-theorie, Formale Sprachen und Komplexitätstheorie*, deutsche Ausgabe, 3. korrigierte Auflage, Addison-Wesley, 1994

[Howa95] Tim Howard: *The Smalltalk Developer's Guide to VisualWorks*, SIGS Books, 1995

[Keil90] Reinhard Keil-Slawik: *Konstruktives Design: Ein ökologischer Ansatz zur Gestaltung interaktiver Systeme*, Habilitationsschrift, TU Berlin, 1990

[KGZ94] Klaus Kilberth, Guido Gryczan, Heinz Züllighoven: *Objektorientierte Anwendungsentwicklung Konzepte, Strategien, Erfahrungen*, 2. ver-besserte Auflage, Vieweg, 1994

[KLSZ92] A. Kieback, H. Lichter, M. Schneider-Hufschmidt, Heinz Züllighoven: *Prototypen in industriellen Software-Projekten – Erfahrungen und Analysen*, Informatik-Spektrum, Band 15, Nr. 2, 1992

[Kopp98] Markus Kopp: *Optimierung von Java-Programmen*, Java Spektrum, Ausgabe 4, Juli/August, Nr. 14, 1998

[Lewis95] Ted Lewis et. al.: *Object Oriented Application Frameworks*, Manning Publications Co., 1995

[Lipp97] Martin Lippert: *Konzeption und Realisierung eines GUI-Frameworks in Java nach der WAM-Metapher*, Studienarbeit am Fachbereich Informatik, Universität Hamburg, 1997

[Lipp99] Martin Lippert: *Die Desktop-Metapher in Systemen nach dem Werkzeug- und Material-Ansatz*, Diplomarbeit am Fachbereich Informatik, Universität Hamburg, 1999

[Mage97a] MageLang Institute: *Swing Short Course, Part I*, http://developer.java.sun.com/developer/onlineTraining/GUI/Swing1/index.html, 1997

[Mage97b] MageLang Institute: *Swing Short Course, Part II*, http://developer.java.sun.com/developer/onlineTraining/GUI/Swing2/index.html, 1997

[Meyer90] Bertrand Meyer: *Objektorientierte Softwareentwicklung*, deutsche Ausgabe, Prentice Hall International, 1990

[Micro93] Microsoft Inc.: *Microsoft Foundation Classes - Documentation*, Redmond, WA, 1993

[MST95] Aaron Marcus, Nick Smilonich, Lynne Thompson: *The Cross-GUI Handbook – For Multiplatform User Interface Design*, Addison-Wesley, 1995

[Myers96] Brad A. Myers: *UIMSs, Toolkits, Interface Builders*, 1996 in: Jakob Nielson, editor: *Handbook of User Interface Design*, , 1997

[Pawl98] Monica Pawlan: *JDK 1.1 and Beyond – Making the Transition*, http://developer.java.sun.com/developer/technicalArticles/GUI/Transition/index.html, Sun Microsystems Inc., 1998

[Reen95] Trygve Reenskaug, P. Wold, O.A. Lehne: *Working with objects : the ooram software engineering method*, Manning Publications Co., 1995

[Rieh97a] Dirk Riehle: *Arbeiten mit Java-Schnittstellen und -Klassen Teil 1*, Java Spektrum, Ausgabe Nr. 5, September/Oktober, 1997

[Rieh97b] Dirk Riehle: *Arbeiten mit Java-Schnittstellen und -Klassen Teil 2*, Java Spektrum, Ausgabe Nr. 6, November/Dezember, 1997

[Rieh97c] Dirk Riehle: *A Role-Based Design Pattern Catalog of Atomic an Composite Patterns Structured by Pattern Purpose*, Ubilab Technical Report 97-1-1, Dirk Riehle and Ubilab, 1997

[RieZül95] Dirk Riehle, Heinz Züllighoven: *A Pattern Language for Tool Construction and Integration Based on the Tools and Materials Metaphor*, in: James O. Coplien, Douglas C. Schmidt (Hrsg.): *Pattern Languages of Program Design*, Addison-Wesley, 1995

[RooWol97] Stefan Roock, Henning Wolf: *Konzeption und Implementierung eines „Reaktionsmusters" für objektorientierte Softwaresysteme*, Mitteilung Nr. 274, Fachbereich Informatik, Universität Hamburg, 1997

[Sedg93] Robert Sedgewick: *Algorithmen in C*, Addison-Wesley, 1993

[Shnei98] Ben Shneiderman: *Designing the user interface : strategies for effective human-computer-interaction*, third edition, Addison-Wesley, 1998

[Star93] Star Division: *StarView C++ Klassenbibliothek Version 2.0: Benutzer- und Programmierhandbuch*, StarDivision, Hamburg, 1993

[SteLam97] Eike Steffen, Sven Lammers: *Einbettung eines Interaktionstypen zur grafischen Bearbeitung von Netzen in VisualWorks 2.5*, Studienarbeit am Fachbereich Informatik, Universität Hamburg, 1997

[Szyp98] Clemens Szyperski: *Component Software, Beyond object-oriented programming*, Addison-Wesley, 1998

[Traub96] Horst-Peter Traub: *Objektorientierte Behälterklassen-Bibliotheken – Konzepte, Entwurf und Implementation*, Mitteilung Nr. 252, Fachbereich Informatik, Universität Hamburg, 1996

[Vlis95] John Vlissides: *Using Design Patterns: Elements of Reusable Architectures*, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598, 1995

[WeiAsb98] Scott R. Weiner, Stephen Asbury: *Programming with JFC*, Wiley Computer Publishing, 1998

[WeiGam95] André Weinand, Erich Gamma: *ET++ – a Portable, Homogeneous Class Library and Application Framework*, in [Lewis95]

[Weiß97] Ulfert Weiß: *Konzeptionelle und technische Weiterentwicklung eines objektorientierten Frameworks nach der Werkzeug-Material-Metapher*, Diplomarbeit am Fachbereich Softwaretechnik, Arbeitsbereich Informatik, Universität Hamburg, 1997

[Wilson90] David A. Wilson: *Programming with MacApp*, Addison-Wesley, 1990

[Züll94] Heinz Züllighoven: *Arbeitsunterlagen zur Lehrveranstaltung „Grundzüge der Informatik A2"*, Skript, Arbeitsbereich Softwaretechnik, Universität Hamburg, 1994

[Züll98] Züllighoven et. al.: *Das objektorientierte Konstruktionshandbuch nach dem Werkzeug & Material-Ansatz*, dpunkt-Verlag, 1998

Erklärung:

Hiermit versichern wir, diese Arbeit selbständig und unter ausschließlicher Zuhilfenahme der in der Arbeit aufgeführten Hilfsmittel erstellt zu haben.

Hamburg den

Eike Steffen
Wilhelm–Cohrs–Weg 2a

21217 Seevetal

Sven Lammers
Geschwister–Scholl–Straße 7a

23795 Bad Segeberg