DIAMANT - Ein User Interface Management System für grafische Benutzerschnittstellen[1]

Bernhard Trefz, Jürgen Ziegler, Stuttgart

Zusammenfassung

Der softwaremäßige Aufwand für die Benutzerschnittstelle wird im Verhältnis zum Gesamtsystem immer größer. Software-Werkzeuge zum Entwurf und zur Implementation sind wesentliche Hilfsmittel, um diese Problemstellung anzugehen. Für eine Klasse dieser Werkzeuge hat sich der Begriff User Interface Management Systeme (UIMS) durchgesetzt. Der vorliegende Beitrag beschreibt einige typische Problemstellungen der gegenwärtigen Entwicklung von UIMS und stellt DIAMANT, ein objektorientiertes UIMS für grafische Benutzerschnittstellen, vor. DIAMANT ist in mehreren Schichten implementiert, um zum einen verschiedenste Ein-/ Ausgabemedien aufnehmen zu können, zum anderen, um dem Entwickler jeweils für seine Problemstellung angepasste Werkzeuge bieten zu können.

1. Generatoren für Benutzerschnittstellen und ihre Bedeutung für die Software-Ergonomie

Bei zunehmender Interaktivität und graphischer Unterstützung von Benutzerschnittstellen wird der softwaremäßige Aufwand für die Benutzerschnittstelle im Verhältnis zu Gesamtsystem immer größer. Zudem muß der Gestaltung der Benutzerschnittstelle eine immer größere Bedeutung zugemessen werden, so daß auch Ansätze der schnellen Prototypenentwicklung verstärkt zum Einsatz kommen sollten, um eine iterative Optimierung der Benutzerschnittstelle zu erreichen.

Software-Werkzeuge zum Entwurf und zur Implementation von Benutzerschnittstellen sind wesentliche Hilfsmittel, um diese Problemstellung anzugehen. Für diese Werkzeuge hat sich der Begriff User Interface Management Systeme (UIMS) (Pfaff 1985, Olsen 1987) durchgesetzt. Grundvoraussetzung für ein UIMS ist die Trennung des Systems in einen Teil, der die Interak-

---

[1] Dieser Beitrag beruht auf Arbeiten, die im Rahmen des ESPRIT Projektes 385 HUFIT (Human Factors in Information Technology) durchgeführt wurden. Wesentlichen Anteil bei der Implementierung von DIAMANT hatte Klaus Wied.

tion mit dem Benutzer definiert, und einen Teil, der die Funktionalität der Anwendung beschreibt (s. Schulert in Rosenberg 1988). Aufgrund dieser Trennung ist es möglich, maßgeschneiderte Werkzeuge für die Spezifikation der Interaktionskomponente zu entwickeln, die sowohl eine benutzer- und aufgabennähere Entwicklung der Benutzerschnittstelle als auch eine effizientere Implementation ermöglichen.

Vorteile für den Einsatz von UIMS können in den folgenden Punkten gesehen werden:

- UIMS können Aufwand und Kosten für die Entwicklung und Änderung interaktiver Systeme deutlich senken.
- Komponenten für Benutzerschnittstellen werden zu eigenständigen Produkten, deren Qualität auch der Evaluation durch den Markt unterworfen ist.
- Sie unterstützen neue Vorgehensweisen zur Systemgestaltung wie rapid prototyping, die eine stärkere Einbeziehung des Benutzers und eine frühzeitige Systemevaluation ermöglichen.
- Sie erhöhen die Änderungsfreundlichkeit des Systems, so daß festgestellte Verbesserungsnotwendigkeiten überhaupt in der Praxis realisiert werden.
- UIMS und die zugrundeliegenden Bausteine bewirken eine stärkere Standardisierung und damit verbesserte Konsistenz der Benutzerschnittstelle.
- Falls die Beschreibung der Benutzerschnittstelle in geeigneter Weise dem Benutzer zugänglich gemacht wird, kann eine Anpassung bzw. Programmierung des Systems durch den Endbenutzer ermöglicht werden. Diese Möglichkeit wird besonders bei zukünftigen, funktional hoch integrierten Systemen bedeutsam werden, die sehr flexible Anwendungsanforderungen unterstützen.

Das vorliegende Papier beschreibt einige typische Problemstellungen der gegenwärtigen Entwicklung von UIMS und den Prototypen eines Systems, das besonders auf die Entwicklung graphischer Benutzerschnittstellen nach dem Prinzip der direkten Manipulation ausgerichtet ist.

2. Ansätze und Problemstellungen bei UIMS

Ein bekanntes Modell für die Strukturierung der Interaktionskomponente ist das Seeheim-Modell (Pfaff 1985), das drei Hauptkomponenten umfaßt:

- Präsentationskomponente
- Dialogkontrolle
- Anwendungsinterface

Je nach Ausrichtung und Zielsetzung können diese Komponenten in unterschiedlichem Maß unterstützt werden. Auf der Ebene der Präsentationskomponente sind z.B. Toolkits wie X-Toolkit, Apple Toolbox, GEM u.a. angesiedelt. Sie unterstützen die sichtbare Darstellung von Objekten, sowie deren grundlegendes Verhalten (aber nicht ihr Zusammenwirken). Die Dialogkontrollkomponente beschreibt die Struktur der Kommandos und den Ablauf des Dialogs. Häufig benutzte Modelle für die Dialogkontrollkomponente sind Zustandsübergangsnetze, Grammatiken oder ereignisgetriebene Darstellungen. Das Anwendungs-Interface beschreibt die Kopplung zwischen Interaktionskomponente und dem Rest des Systems.

Eine zweite Unterscheidungsmöglichkeit für Benutzerschnittstellenwerkzeuge liegt in dem Grad der Unterstützung des Entwicklers. Diese können von reinen Programmiertechniken bis hin zur graphischen Definition des Systems reichen. Zwischenstufen sind z.B. in der Verwendung von Bibliotheken oder in einem strukturierten, geleiteten Programmieren zu sehen.

Im folgenden werden anhand von Systembeispielen einige typische Problemstellungen aufgezeigt, die mit der Entwicklung von DIAMANT angegangen werden sollen.

Das System DiCo (Dialog Compiler, Grausam 1988) basiert auf dem Ereignismodell (s. z.B. Hill 1986) und erlaubt die Beschreibung formularartiger Schnittstellen, die auch grafische Komponenten wie Buttons, Sliders etc. beinhalten. Die Beschreibung wird in einer Objekt- und in einer Regeldatei abgelegt. In der Objektdatei werden alle an der Schnittstelle verwendeten Objekte definiert (Beschreibung der Präsentationskomponente). Die Regeldatei (Beschreibung der Dialogkontrolle) beschreibt das interaktive Verhalten dieser Objekte durch Regeln, die auf eintretende Ereignisse reagieren. Eine Einschränkung dieses Systems besteht darin, daß zur Laufzeit keine neuen Präsentationsobjekte erzeugt werden können. Dies ist z.B. notwendig, um eine variable, nicht beschränkte Anzahl von Ikonen auf einer Bildschirmoberfläche darzustellen. Weiterhin müssen beide Beschreibungsdateien textuell erstellt werden, wobei die Beschreibungssprache allerdings wesentlich einfacher als eine konventionelle Programmiersprache ist.

Das System Dialog Manager (Dannenberg 1987) basiert auf dem gleichen Modell wie DiCo, erlaubt jedoch die grafische Erstellung der Präsentations- und Regelkomponenten. Die Regelkomponente wird dabei in Form einer Entscheidungstabelle mit Bedingungen und Aktionen visualisiert. Dieses System erlaubt eine einfache Erstellung von Benutzerschnittstellen auch durch nicht programmierende Benutzer. Auch hier können zur Laufzeit keine neuen Objekte erzeugt werden.

Eine grundlegende Problemstellung ist die Modellierbarkeit von Objekten, die im Kontext der Aufgabenstellung des Benutzers sinnvolle Einheiten bilden und nicht nur eine Oberflächenrepräsentation beschreiben. Beispielsweise kann eine Maschine in einem Produktionsplanungssystem

als Piktogramm oder durch ein Fenster mit einer Liste von Arbeitsaufträgen - eventuell auch gleichzeitig- dargestellt werden. Bei beiden genannten Systemen kann dies nur durch eine explizite Verknüpfung mehrerer Oberflächenobjekte mit Regeln ereicht werden.

Hieraus ergaben sich für die Entwicklung von Diamant folgende Ziele:

- dynamische Erzeugung von Objekten zur Laufzeit
- einfache Beschreibungssprache mit grafischen Eingabemöglichkeiten
- abstraktere, aufgabenorientierte Objekt- und Ereignistypen
- Einbeziehung der Kommunikation zu anderen Systemkomponenten in die Dialogkontrolle

3. DIAMANT (Dialogue Management Tool)

Auch Diamant geht von einer Aufteilung eines Systems in einen Anwendungs - und einen Benutzerschnittstellenteil aus. Aufgrund der guten Erfahrungen mit den Systemen Dialog Manager und DiCo basiert auch Diamant auf einem ereignisgetriebenen Modell.

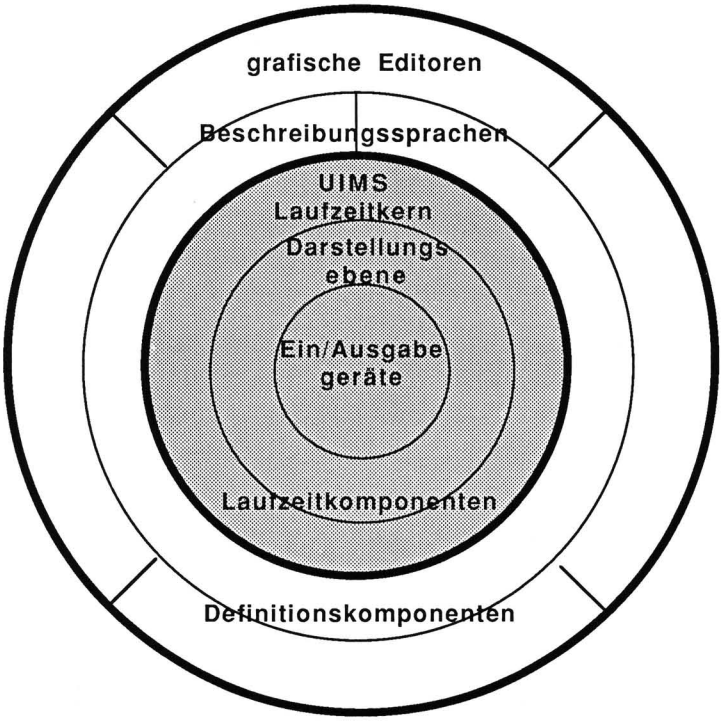

Abb.1: Architektur von DIAMANT

Der Aufbau von Diamant kann in einem Schichtenmodell (Abb. 1) dargestellt werden, wobei die inneren Schichten der Laufzeit und die äußeren Schichten der Definitionszeit zugeordnet sind.

Auf der Darstellungs- und Geräteebene wurden bisher ein auf dem Fenstersystem X-Windows aufsetzender grafischer Werkzeugkasten (InterViews, Linton 1987) sowie Spracheln-Ausgabemöglichkeiten in Diamant integriert. Von X-Windows nicht abgedeckte Ein/Ausgabemedien werden über von Diamant angebotene Interprozesskommunikationsmöglichkeiten erschlossen. Das Ziel von Diamant auf dieser Ebene ist nicht die Implementierung neuer Ein/Ausgabebibliotheken, sondern vielmehr eine Architektur zur Integration bereits vorhandener Softwarepackete zu bieten.

Der Laufzeitkern von Diamant wurde unter Unix in C++, einer objektorientierten Erweiterung von C, implementiert. Diamant präsentiert sich auf dieser Ebene als eine Sammlung von Verwaltungsobjekten und vordefinierten Grundklassen. Grundklassen können von der Ebene der Beschreibungssprachen her benutzt werden und stellen teilweise Schnittstellenobjekte zur Darstellungsschicht dar, können jedoch auch auf höherer Ebene liegen und bereits eine bestimmte Dialogsyntax implementieren, bspw. eine Klasse, welche unter bestimmten Bedingungen das Ereignis "Mausdoppelklick" als "öffnen" interpretiert.

Die Kommunikation von Objekten erfolgt über Ereignisse, die aus einem Typ und einem Wert bestehen und die von einem Sender über einen Verteiler an einen Empfänger geschickt werden. Ereignisse, die von der Darstellungsschicht an den Laufzeitkern gemeldet werden, laufen ebenfalls über den Verteiler, welcher sie entgegennimmt, um sie dann an den angegebenen Empfänger bzw. interessierte Objekte weiterzuleiten. Es ergibt sich dadurch das Modell eines sequentiellen Ereignisstroms, aus dem Ereignisse nach Prioritäten ausgewählt werden, um dann an verarbeitende Objekte weitergeleitet zu werden.

Bisher befinden sich der Laufzeitkern und die Objekte des grafischen Werkzeugkasten aus Performancegründen in einem einzigen Prozeß, während Sprach - Ein/Ausgabe als zwei separate Prozesse implementiert wurden.

3.1 Beschreibungssprachen

Für den Entwickler einer Benutzerschnittstelle bleibt die Laufzeitkomponente mit ihren Details unsichtbar. Sein konzeptuelles Modell der Benutzerschnittstelle wird in einer problemangemessenen Sprache ausgedrückt. Dabei ist die vorgestellte Architektur in der Lage, unterschiedliche Beschreibungsmodelle auf die gleiche Laufzeitkomponente abzubilden (s. auch Green 1986). Gegenwärtig ist eine Beschreibungssprache entwickelt worden (Diamant UIDL -

User Interface Description Language), die in ihren Konzepten und in ihrer Strukturierung weitgehend an die Laufzeitkomponente angelehnt ist. Eine mit UIDL beschriebene Schnittstelle wird dann bei der Systemgenerierung in C++ Klassen übersetzt.

Als Hauptschwierigkeit während der Entwicklung von UIDL erwies es sich, das richtige Maß zwischen Einfachheit und funktionaler Mächtigkeit der Sprache zu bestimmen, d. h. UIDL aufgabenangemessen und gut erlernbar zu gestalten. Ausgehend von einem Modell von über Ereignisse miteinander kommunizierenden Objekten besteht die Hauptaufgabe einer Schnittstellenentwicklung mit UIDL in der Definition, Modifikation und Verknüpfung von Objektklassen. Eine Klasse beschreibt Aufbau und Verhalten der aus ihr erzeugten Objekte (Instanzen). In den meisten Fällen wird eine neue Klasse entweder eine bereits vorhandene Klasse spezialisieren (Vererbung) oder aber Objekte anderer Klassen zu einem speziellen Kontext zusammensetzen. Das Verhalten einer Klasse wird in Reaktionen auf eintretende Ereignisse gesehen. In der Beschreibung der Klasse wird dies durch Ereignissen zugeordnete Regeln ausgedrückt. Ein eingetretenes Ereignis wird durch eine Regel in einem bestimmten Objektzustands interpretiert und kann zu einer Zustandsänderung des Objekts, zum Erzeugen neuer Instanzen oder zum Verschicken von neuen Ereignissen führen. Da die Regeln einer Klasse das Verhalten aller aus ihr erzeugten Instanzen beschreiben und nicht, wie bei den Systemen Dialog Manager oder DiCo, jeder einzelnen Instanz spezielle Regeln zugeordnet werden müssen, können zur Laufzeit beliebig viele Instanzen erzeugt werden.

Um Zustände und Identifikationen von kreierten Objekten abspeichern zu können, wurden Variable in die Sprache aufgenommen. Variable müssen vor ihrer Verwendung nicht definiert werden, d.h. das System stellt automatisch für jede benutzte Variable Speicherplatz bereit. Es wurde auch davon abgesehen, eine Typendeklaration von Variablen in die Sprache aufzunehmen; stattdessen bestimmt das System zur Laufzeit die Bedeutung eines Operators aus der Belegung der beteiligten Operanden. Der Operator "+" wird, wenn er bspw. auf zwei Variable mit einem Zahlenwert angewendet wird, als Addition interpretiert, während er auf zwei Variable mit Zeichenkettenwerten angewandt als Konkatenation interpretiert wird. Da zur Laufzeit beliebig viele Instanzen gezogen werden können, muß die Sprache die Möglichkeit bieten, die Identifikation dieser Instanzen zu speichern. Zur Speicherung von beliebig vielen Werten kann deshalb eine Variable auch als ein Feld mit beliebig vielen Einträgen behandelt werden. Als Index für ein Feld sind Zahlen, Zeichenketten und Objektidentifikationen zugelassen. Um abstrakte, aufgabenorientierte Ereignistypen modellieren zu können, werden als Ereignistypen ebenfalls Zeichenketten, Zahlenwerte und Objektidentifikationen zugelassen

Der Anwendungsteil eines Systems kann von der Benutzerschnittstelle auf drei Arten angesprochen werden. Die erste Möglichkeit besteht im Aufruf einer in C++ geschriebenen Anwendungsfunktion aus einer Regel heraus. Als zweite Möglichkeit kann die Anwendung als eigener

Prozeß implementiert werden, welcher mit der Benutzerschnittstelle über die Interprozesskommunikationsmöglichkeiten von Diamant kommuniziert ("Client - Server" Modell). Eine dritte Möglichkeit besteht darin, Teile der Anwendung als C++ Klassen, die das Kommunikationsprotokoll des Laufzeitkerns befolgen, zu implementieren. Diese Klassen sind dann auf der Ebene der Beschreibungssprache nicht von dem vom Laufzeitkern angebotenen Grundklassen zu unterscheiden. Die Anwendung kann die Benutzerschnittstelle durch das Verschicken von Ereignissen ansprechen.

Für die eigentliche Syntax der Sprache stellt sich die Frage, ob eine ausführlichere, leicht lesbare Syntax oder aber eine kryptischer, dafür aber mit weniger Schreibaufwand verbundene Syntax zu wählen ist. Wir entschieden uns für eine mit weniger Schreibaufwand verbundene Syntax, da wir erwarten, daß ihre Vorteile nach einer gewissen Einarbeitungsphase überwiegen.

Um die in Abb. 2 gezeigte Confirm Box aus den Grundelementen Button und Boxes aufzubauen, muß in Diamant folgender Code geschrieben werden

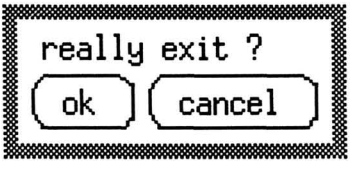

Abb. 2 : Eine mit UIDL definierte Confirm Box

```
eventhandler ConfirmBox is a VBoxHandler {

    on start  with (message){
        hbox = create(HBoxHandler);
        okButton = create(PushButtonHandler,"ok");
        cancelButton = create(PushButtonHandler,"cancel");
        question = create(LabelHandler,message);
        send(hbox,"insert",okButton);
        send(hbox,"insert",cancelButton);
        send(this,"insert",question);
        send(this,"insert",hbox);
        }

    on "pressed" from okButton {
        send(parent,"ok");
        }

    on "pressed" from cancelButton {
        send(parent,"cancel");
        }

    }
```

Eventhandler ist ein Schlüsselwort und leitet eine Klassendefinition ein. Die neue Klasse ConfirmBox ist eine Subklasse von VBoxHandler, ererbt also deren Eigenschaft, in sie eingefügte Objekte ("insert" Ereignis) vertikal ausgerichtet darzustellen. Die Startregel beschreibt Aktionen, welche beim Kreieren einer neuen Instanz durchgeführt werden sollen. Der Parameter "message" enthält den Wert des Startereignisses. In dem Beispiel befinden sich zwei vordefinierte Variable "this"und "parent". In "this" ist immer die eigene Identifikation der Instanz gespeichert, in "parent" ist immer die Identifikation des Objekts gespeichert, welches diese Instanz kreiert hat. Will man nun Sprach-Ein/Ausgabe, realisiert durch zwei Prozesse "speech" und "talk", mit dieser Confirm Box verbinden, so könnte dies folgendermaßen geschehen.

```
eventhandler SpeakingConfirmBox is a ConfirmBox {

    on start with (message) : (message) {
        question = message + ".";        // Für Sprachausgabe Satzabschluß mit Punkt
        send(system,"listen","speech")
        }

    on "input" with (word) from "speech"{
        if (word == "ok")
            send(parent,"ok");
        else if (word == "cancel")
            send(parent,"cancel");
        else {
            reply = "can´t interpret command " + word + ".";
            send("talk","speak",reply);
            }
        }

    on "ask" {
        send(talk,"speak",question);
        }
    }
```

3.2 Grafische Editoren

Viele Benutzerschnittstellen zeigen teilweise ein gleichartiges Verhalten. Deshalb ist zu erwarten, daß eine Programmbibliothek mit wiederverwendbaren UIDL Klassen, welche Standardkomponenten einer Benutzerschnittstelle implementieren, die Entwicklung von Benutzerschnittstellen bedeutend vereinfachen kann. Für einige Standardlösungen kann eine grafische Editiermöglichkeit, mit welcher die Standardlösung modifiziert wird, geschaffen werden. Die modifizierte Standardschnittstelle wird dann in das Laufzeitsystem von Diamant mit einbezogen, so daß sie mit anderen Komponenten, die bspw. mit UIDL definiert wurden, kommunizieren kann. Bisher wurde ein Prototyp für die Modifikation einer Schnittstelle mit Piktogrammen auf einer Schreibtischoberfläche entwickelt (s. Abb. 3).

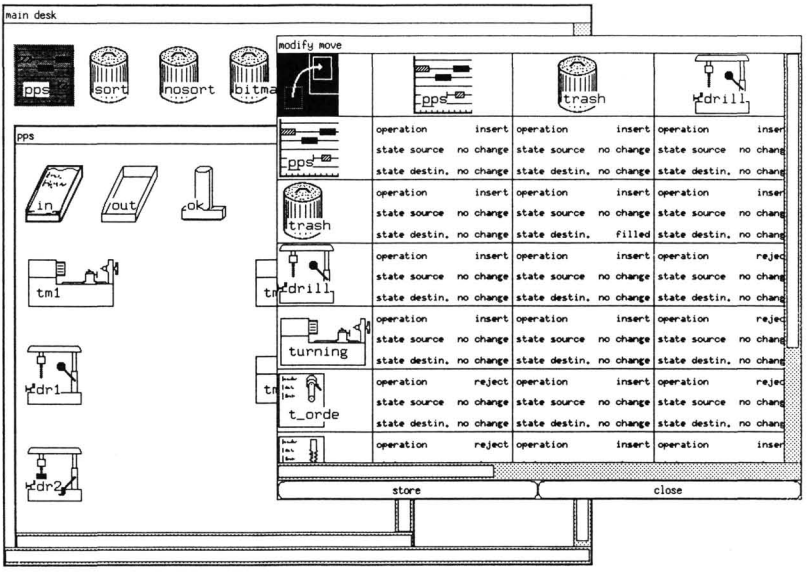

Abb.3: Prototyp eines grafischen Editors für eine Standardschnittstellenkomponente

Dabei können ikonisch repräsentierte Objekte (in diesem Fall Fertigungsaufträge und -maschinen) in einem oder mehreren Fenstern zusammengestellt werden. Das Verhalten der Schnittstelle wird durch Reaktionen auf Ereignisse bestimmt. Die verwendeten Ereignistypen liegen hier auf einer wesentlich höheren Ebene als die typischerweise verwendeten wie "Mausklick" etc. Ein hier verwendetes Ereignis ist "Benutzer hat Ikone A mit Ikone B in Überlappung gebracht". Als Reaktion kann z.B. das Einfügen eines Objekts in eine Ablage oder das Löschen eines Objekts angegeben werden. Die Funktionalität zum Bewegen von Ikonen, wird bereits von der Komponente selbst geliefert.

3.3 Erfahrungen mit DIAMANT

Die vorgestellte Architektur von DIAMANT erweist sich als sehr flexibel, so daß neue Ein/Ausgabemedien einfach in das Gesamtsystem integriert werden konnten. Die bisher entwickelte Beschreibungssprache UIDL eignet sich gut zur Implementierung einer auf einem ereignisgetriebenen Modell basierenden Benutzerschnittstelle. Durch den Vererbungsmechanismus von UIDL können auf einfache Art und Weise Veränderungen an vorgegebenen Klassen durchgeführt werden. UIDL bietet jedoch nur eingeschränkte Möglichkeiten, um komplexere Datenstrukturen zu modellieren. Hier wurde der Einfachheit der Sprache der Vorzug vor ihrer funktionalen Mächtigkeit gegeben. UIDL erlaubt durch ihre freie Definierbarkeit von Klassen

(gegenüber einem festen Satz von Interaktionsobjekten wie Felder, Button etc.) abstraktere, aufgabenbezogenere Objekte zu modellieren. Eine wesentliche Erweiterung besteht weiterhin darin, daß auch die Ereignistypen entsprechend abstrahiert werden können. Während bei konventionellen Werkzeugkästen die Ereignisse aus den vom Benutzer erzeugten Mausklicks, Tastendrücken etc. bestehen, erlaubt Diamant die Definition von Ereignissen, die bereits höherstehende, aufgabenbezogene Aktionen des Benutzers kennzeichnen.

Die Entwicklung aufgabenbezogener User Interface Management Systeme führt dazu, daß ein größerer Anteil der Gesamtanwendung im UIMS lokalisiert ist. Dies bezieht sich insbesondere auf diejenigen Teile der Anwendung, die die Kontrollstruktur bei der Aufgabenbearbeitung durch den Benutzer betreffen. Diese Strukturierung steht im Einklang mit der zunehmenden Anforderung, den Benutzer nicht nur durch das Zugänglichmachen von Interaktionstechniken und geeigneten Informationsdarstellungen zu unterstützen, sondern auch Hilfsmittel zur eigentlichen Aufgabenbearbeitung in der Benutzerschnittstelle bereitzustellen.

Literatur

Dannenberg 1987: M. Dannenberg, J. E. Ziegler. "A Dialogue Management Tool for Prototyping and Development of Interactive Systems", Esprit Project 385 - HUFIT, Working Paper HUFIT/16-IAO-11/87

Grausam 1988: Martin Grausam "Erstellung eines Regel-Compilers zur Dialogsteuerung," FhG-IAO interner Forschungsbericht, Stuttgart, 1988

Green 1986: Mark Green "A Survey of Three Dialogue Models," ACM transactions on Graphics. Vol. 5, No.3, July 1986. pp. 244 - 275

Hill 1986: Ralph D. Hill. "Supporting Concurrency, Communication and Synchronization in Human-Computer Interaction - The Sassafras UIMS," ACM transactions on Graphics. Vol. 5, No.3, July 1986. pp. 179 - 210

Linton 1987: M. Linton, R. Calder, J. Vlissides "The Design and Implementation of InterViews," Proceedings of the USENIX C++ Workshop, Santa Fe, New Mexico, November 1987

Olsen 1987: D.R. Olsen et al.: "ACM SIGGRAPH Workshop on Software Tools for User Interface Management," Computer Graphics, 21, Nr. 2, April 1987, pp. 71 - 147

Pfaff 1985: G.E. Pfaff (Ed.) "User Interface Management Systems," Berlin - Heidelberg, Springer Verlag, 1985

Rosenberg 1988: J. Rosenberg "UIMSs: Threat or Menace ?" Proc. of CHI, 1988 (Washington, May 15-19). ACM, New York, pp 197 - 200.

Adresse der Autoren: Bernhard Trefz und Jürgen Ziegler
Fraunhofer-Institut für Arbeitswirtschaft und Organisation
Nobelstraße 12
7000 Stuttgart 80