Eine objektorientierte Architektur für direkt manipulative, verteilte Bürosysteme

Christian Janssen, Stuttgart[1]

Zusammenfassung

In diesem Beitrag wird eine Architektur für direkt manipulative, verteilte Bürosysteme vorgeschlagen. Die klassische Trennung von Ebenen der Präsentation, Dialogsteuerung und Anwendung wird im Prinzip beibehalten. Es wird aber auf Flexibilität und die Möglichkeit der semantischen Rückkopplung in der Präsentationskomponente geachtet. Außerdem wird die Möglichkeit paralleler Dialoge verschiedener Benutzer mit einer Anwendung vorgesehen. Als Werkzeugunterstützung dient ein objektorientierter Oberflächenbaukasten auf der Ebene der Präsentation. Auf der Ebene der Dialogsteuerung ist ein Anwendungsrahmen oder ein flexibles User Interface Management System (UIMS) erforderlich. Als Demonstrationsbeispiel wird die Implementation von Oberflächenobjekten für ein ikonisches Dateisystem beschrieben, die als wiederverwendbare Bausteine auf der Basis des Fenstersystems NeWS entwickelt wurden.

1. Einleitung

Die Dialogform der Direkten Manipulation (Shneiderman, 1982, 1983) verspricht für die Benutzung von Software-Systemen Vorteile wie leichtes Lernen, gutes Behalten und geringe Fehleranfälligkeit. Diese sind zwar noch nicht vollständig empirisch überprüft (Ilg und Ziegler, 1988), aber dennoch hat sich die Direkte Manipulation in vielen Bereichen, wie z.B. im Bürobereich, verbreitet. Der Programmieraufwand bei der Entwicklung direkt manipulativer Systeme ist jedoch im Vergleich zu anderen Dialogformen besonders hoch (Myers, 1989). Die Unterstützung der Programmierung mit vorgefertigten Bausteinen und/oder Spezifikationssprachen ist hier schwieriger als etwa bei Masken- und Menüsystemen. Zum einen bieten heutige Benutzungsoberflächenbaukästen noch keine vollständige Unterstützung für die Entwicklung direkt manipulativer Systeme an. Zum anderen wird das traditionelle Konzept der User Interface Management Systeme (UIMS) im Zusammenhang mit Direkter Manipulation von einigen Stimmen als gescheitert angesehen (Rosenberg, 1988).

Im zweiten Abschnitt dieses Beitrags wird ausgeführt, wie das traditionelle Architekturkonzept für interaktive Systeme mit der Trennung von Komponenten für Präsentation, Dialogsteuerung und

---

[1] Dieser Beitrag basiert auf meiner Diplomarbeit (Janssen, 1989), die im Rahmen des Esprit-Projektes "Communication Systems Architecture" (CSA, siehe Behr et al., 1988) in Zusammenarbeit mit dem Philips Forschungslabor in Hamburg durchgeführt wurde. Ich danke dem Forschungslabor, namentlich Barbara Fink und Rolf Stecher, für die Unterstützung. Besonders danke ich Horst Oberquelle für die Betreuung der Diplomarbeit an der Universität Hamburg und die Durchsicht dieses Beitrags.

Anwendung zu erweitern ist, damit es auch auf direkt manipulative Systeme angewendet werden kann. Außerdem werden Entwicklungen im Bereich der Fenstersysteme mit einbezogen und parallele Dialoge sowie die Verteilung von Anwendung und Komponenten der Benutzungsschnittstelle berücksichtigt. Im dritten Abschnitt werden Möglichkeiten der Werkzeugunterstützung für den Entwurf und die Implementation von Systemen gemäß der Architektur besprochen. Der vierte Abschnitt beschreibt die Implementation der Benutzungsoberfläche eines ikonischen Dateisystems, die unter Verwendung objektorientierter Programmiertechniken mit Hilfe von wiederverwendbaren Bausteinen konstruiert wurde. Das Beispiel zeigt die Bewegung von Ikonen mit der Maus als eine für Direkte Manipulation typische Interaktionsform. Der fünfte Abschnitt gibt einen Ausblick auf künftige Entwicklungen.

2. Eine Architektur für direkt manipulative Systeme

In Green (1985a) wird das "Seeheim-Modell" für interaktive Systeme beschrieben, das heute vielen Architektur-Modellen zugrunde liegt. Es wird hierbei eine Trennung der Benutzungsschnittstelle von der eigentlichen Anwendung vorgenommen, wobei die Benutzungsschnittstelle selbst noch in eine Präsentations- und eine Dialogsteuerungskomponente zerfällt. Zusätzlich zur Anwendung ist eine gesonderte Komponente für die Anwendungsschnittstelle vorgesehen, die hier aber vernachlässigt werden soll. Im Zusammenhang mit der Verwendung dieses Modells für direkt manipulative Systeme treten einige Probleme auf (vgl. Hudson, 1987):

- Es ist große **Flexibilität in der Präsentationskomponente** erforderlich. Die Präsentationskomponente kann daher nicht, wie im urspünglichen Modell angenommen, auf einer festen Menge von Interaktionstechniken eines Standard-Graphiksystems aufbauen. Es müssen anwendungsabhängig Erweiterungen der Interaktionstechniken möglich sein, um dem Benutzer eine angemessene Rückkopplung geben zu können. Als Beispiel sei das Aufziehen von geometrischen Figuren in einem graphischen Editor angeführt. Steht, wie in vielen Graphiksystemen, bei Bewegungen mit der Maus nur das dynamische Zeichnen und Löschen einer Linie zu einem Referenzpunkt als Rückkopplung zur Verfügung, so kann beim Aufziehen von Kreisen, Rechtecken usw. keine angemessene Rückkopplung gegeben werden.

- In manchen Fällen werden Informationen über die Anwendung benötigt, um eine **semantische Rückkopplung** geben zu können. Zum Beispiel soll in einem ikonischen Dateisystem beim Bewegen eines Dateiikons über ein Verzeichnisikon das Verzeichnisikon invers dargestellt werden, um dem Benutzer die Möglichkeit der Operationsauslösung (Bewegen oder Kopieren einer Datei) anzuzeigen. Werden aber Dateiikone übereinander bewegt, soll keine Invertierung erfolgen. Um diese semantische Rückkopplung korrekt und unmittelbar geben zu können, müssen in der Benutzungsschnittstelle Informationen über die semantischen Beziehungen der Oberflächenobjekte zur Verfügung stehen.

Das in Abb. 1 dargestellte Architekturmodell trägt diesen Problemen Rechnung. Die Präsentationskomponente beruht dem heutigen Stand der Technik entsprechend auf einem Fenstersystem. Fenstersysteme bieten heute für hochdynamische Interaktionen, z.B. bei Mausbewegungen, weit flexiblere Möglichkeiten der Rückkopplung als frühere Standard-Graphiksysteme. Auf den Mechanismen des Fenstersystems bauen Oberflächenbausteine (z.B. Fenster, Ikone und Menüs) auf, die externe Präsentationen verwalten und zugehörige Eingaben (Eingabeereignisse) entgegennehmen.

Um semantische Rückkopplungen geben zu können, wird ein zweistufiges Modell vorgeschlagen: Statische semantische Informationen, die nicht vom Zustand der Anwendung abhängen, werden in den Oberflächenobjekten kodiert, so daß davon abhängige Rückkopplungen unmittelbar gegeben werden können. Zum Beispiel werden bei dem oben angeführten ikonischen Dateisystem nur statische semantische Informationen benötigt. Die Bereitstellung dynamischer semantischer Informationen verbleibt dagegen in der Anwendung. So erfolgt etwa die Prüfung, ob für das Kopieren einer Datei noch genügend Speicherplatz verfügbar ist, in der Anwendung.

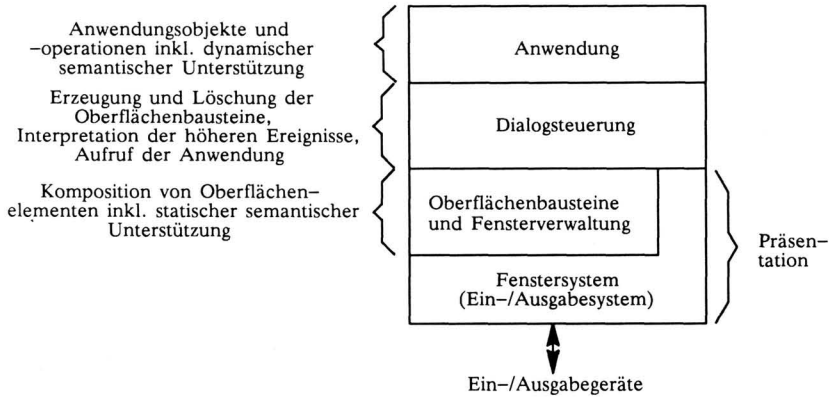

Abb. 1: Geschichtete Darstellung der Architektur

Die Dialogsteuerung hat die Aufgabe der Steuerung des Dialogablaufes aufgrund der Interpretation der Ereignisse, die von den Oberflächenobjekten an sie weitergegeben werden (höhere Ereignisse). Entsprechend der Interpretation wird ein Aufruf der Anwendung durchgeführt. Der neue Anwendungszustand wird abgefragt und durch Oberflächenaufrufe extern dargestellt. Zur Unterstützung paralleler Anwendungen und paralleler Zugriffe auf Anwendungen werden folgende Erweiterungen des Schichtenmodells vorgenommen (Abb. 2):

- Jeder Benutzer kann verschiedene Anwendungen parallel betreiben. Zur Steuerung jedes Dialoges mit einer Anwendung existiert dabei eine separate Dialogsteuerung. Die Zuordnung

der Eingaben des Benutzers zu den einzelnen Dialogsteuerungen übernimmt das Fenstersystem.

- In einer vernetzten Umgebung, wie heute im Bürobereich üblich, können parallele Zugriffe verschiedener Benutzer auf dieselbe Anwendung sinnvoll sein. Man denke etwa an einen elektronischen Raumnutzungsplan zur Koordination der Raumbelegung. Hierbei existiert ebenfalls für jeden Dialog, den ein Benutzer mit der Anwendung führt, eine Dialogsteuerung. (Prinzipiell kann auch ein Benutzer mit derselben Anwendung parallel verschiedene Dialoge führen.) Die parallelen Dialoge entsprechen parallelen Sichten (Views) im Model-View-Controller-Modell (MVC-Modell) von Smalltalk (Krasner und Pope, 1988). Um Zustandsänderungen an alle zugehörigen Dialoge melden zu können, existiert in der Anwendung eine Liste dieser Dialoge. Aufgrund der Änderungsmeldung aktualisieren die Dialogsteuerungen den Oberflächenzustand, so daß dieser überall mit dem Anwendungszustand konsistent ist. Bei echt paralleler Ausführung von Dialogen müssen aber in Erweiterung des MVC-Modells Synchronisationsmechanismen existieren, um die Konsistenz des Anwendungszustandes zu gewährleisten. Im einfachsten Fall wird dies durch strenge Serialisierung der Aufrufe gewährleistet. In der Communication Systems Architecture (CSA; Behr et al., 1988) können dagegen mehrere Operationen in einem Objekt parallel ausgeführt werden. Hierbei werden dann Konsistenzbedingungen mit Hilfe von erweiterten, offenen Pfadausdrücken angegeben.

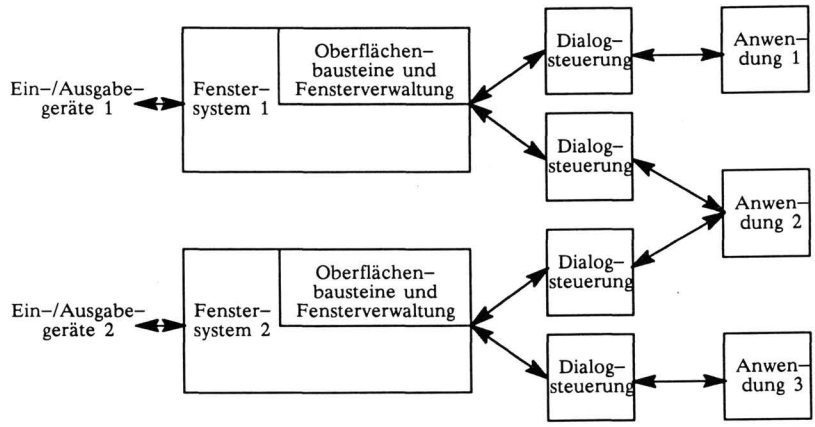

Abb. 2: Berücksichtigung paralleler Dialoge

Zur Erläuterung des beschriebenen Architekturkonzeptes sei als Beispiel die Architektur eines ikonischen Dateisystems beschrieben, dessen Benutzungsoberfläche in Abb. 3 dargestellt ist. Die Oberfläche ist hierarchisch strukturiert und besteht aus einem Fenster mit einem Rahmen und einer Darstellungsfläche, auf welcher sich wiederum die Ikone befinden. (Ein zugehöriges Pop-up-

Darstellungsfläche, auf welcher sich wiederum die Ikone befinden. (Ein zugehöriges Pop-up-Menü ist hier nicht dargestellt.) Mit Hilfe der Maus kann der Benutzer Manipulationen wie Selektion, Bewegung und Menüauswahl vornehmen. Zu jedem Fenster gehört eine Dialogsteuerung, wobei verschiedene Fenster eines Dialoges derselben Dialogsteuerung zugeordnet sind. Die Eingabeereignisse werden zunächst in den Oberflächenobjekten behandelt und nur dann an die Dialogsteuerung weitergegeben, wenn sie für diese relevant sind. Ein für die Dialogsteuerung relevantes Ereignis (höheres Ereignis) ist z.B. die Selektion eines Ikons, da bei einer nachfolgenden Menüauswahl die entsprechende Operation auf das aktuell selektierte Objekt angewendet werden muß. Die Bewegung von Ikonen an der Oberfläche ist für die Dialogsteuerung nur dann von Bedeutung, wenn zwei in semantischer Beziehung stehende Ikonen zur Deckung kommen und eine Anwendungsoperation ausgelöst werden muß. Anderenfalls wird die Bewegung von Ikonen allein von den Oberflächenobjekten behandelt.

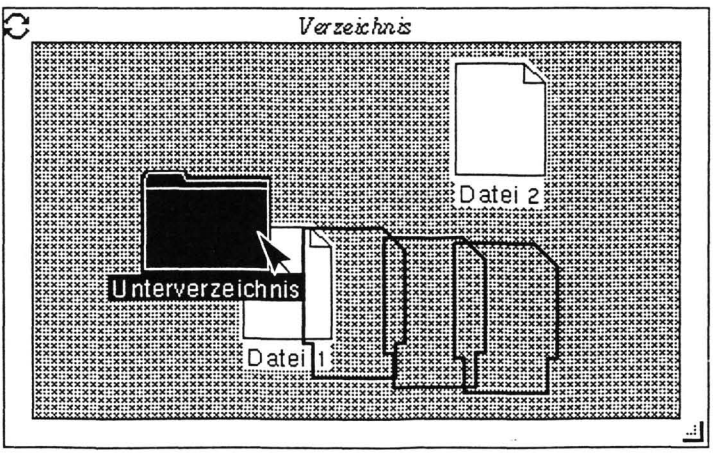

Abb. 3: Benutzungsoberfläche eines ikonischen Dateisystems

Das der Architektur zugrundeliegende Dialogmodell kann als objektorientiert bezeichnet werden und ist eine Variante des Ereignismodells (zu Dialogmodellen siehe Green, 1985b). Eingabeereignisse werden vom Fenstersystem aufgenommen und den Oberflächenobjekten zugeordnet, indem eine zugehörige Operation aufgerufen wird. Höhere Ereignisse werden von den Oberflächenobjekten durch Aufruf entsprechender Operationen in den jeweils zugehörigen Dialogsteuerungen weitergegeben, die hier auch als Objekte im Sinne der objektorientierten Programmierung verstanden werden. Wird zum Beispiel ein Ikon selektiert, so wird in der Dialogsteuerung die Operation "Selektion" mit dem Namen des Ikons als Parameter aufgerufen.

3. Werkzeuge für den Entwurf und die Implementation direkt manipulativer Systeme

Wesentliche Ziele, die durch die Werkzeugunterstützung beim Entwurf und der Implementation interaktiver Systeme erreicht werden sollen, sind

- Reduktion des Entwicklungsaufwandes,
- Unterstützung des Prototyping,
- unabhängige Änderbarkeit von Benutzungsschnittstelle und Anwendung und
- Konsistenz der entstehenden Benutzungsschnittstellen.

In diesem Abschnitt werden zunächst Oberflächenbaukästen als Werkzeuge für die Entwicklung der Präsentationskomponente und dann User Interface Management Systeme (UIMS) und Anwendungsrahmen für die Unterstützung der Dialogsteuerung diskutiert.

Oberflächenbaukästen beinhalten Bausteine für die Konstruktion der Benutzungsoberfläche, wie z.B. Fenster, Menüs und Ikone. Herczeg (1989) formuliert als Anforderungen an Oberflächenbaukästen die Modularisierbarkeit, Spezialisierbarkeit und Aggregierbarkeit der Bausteine sowie die Erweiterbarkeit um neue Bausteine, falls Aggregation und Spezialisierung nicht ausreichen. Insbesondere die Forderung nach Spezialisierbarkeit führt direkt zu objektorientierter Konstruktion und Implementation des Oberflächenbaukastens.

Im vorigen Abschnitt wurde darauf hingewiesen, daß für direkt manipulative Systeme große Flexibilität bei der Entwicklung der Oberflächenbausteine erforderlich ist. Andererseits sollte die Oberfläche dennoch soweit wie möglich auf Standardbausteinen beruhen, um die Konsistenz zu sichern und Mehrfach-Aufwand bei der Programmierung zu vermeiden. Diese Anforderungen werden durch objektorientierte Programmiertechniken optimal erfüllt (siehe auch Abschnitt 4). Heutige Oberflächenbaukästen bieten allerdings vielfach noch nicht diese gewünschte Flexibilität, da sie zwar meist objektorientiert entworfen sind, aber oft keine objektorientierte Implementationssprache verwendet wird.

Der Nachteil bei der ausschließlichen Verwendung von Oberflächenbaukästen für die Entwicklung interaktiver Systeme besteht darin, daß weder eine Trennung von Dialogsteuerung und Anwendung noch die Entwicklung der Dialogsteuerung unterstützt wird. Dementsprechend werden die Ziele der Minimierung des Entwicklungsaufwandes und der möglichst unabhängigen Änderbarkeit von Dialogsteuerung und Anwendung durch Benutzungsoberflächenbaukästen nicht erreicht.

Das Konzept der **User Interface Management Systeme** beinhaltete ursprünglich die Spezifikation der gesamten Benutzungsschnittstelle auf hohem Abstraktionsniveau. Dies ist für

die Benutzungsoberfläche (d.h. die Präsentationskomponente) direkt manipulativer Systeme wegen der auf dieser Ebene benötigten Flexibilität nicht vollständig durchführbar, da die Mächtigkeit von Programmiersprachen benötigt wird. Die Verwendung einer Spezifikationssprache für die Dialogebene ist allerdings möglich, wenn dabei neue Klassen von Oberflächenbausteinen eingebunden werden können. Dieses Konzept verfolgen z.B. Trefz und Ziegler (1989).

Ein anderes Konzept zur Unterstützung der gesamten Benutzungsschnittstelle ist der **Anwendungsrahmen** (application framework; siehe z.B. Schmucker, 1986; Krasner und Pope, 1988). Anwendungsrahmen werden durch vordefinierte Klassen für Benutzungsschnittstellen- und Anwendungsobjekte in objektorientierten Systemen gebildet, wobei allerdings bisher nicht zwischen den Ebenen der Präsentation und der Dialogsteuerung unterschieden wird. Dennoch wird die Dialogsteuerung unterstützt. Im System MacApp (Schmucker, 1986) wird beispielsweise eine Rahmensteuerung für die Umkehrung von Operationen (UNDO) vorgegeben. Zusätzlich ist in Anwendungsrahmen Standardfunktionalität für Anwendungsobjekte vorhanden, die den Steuerfluß zwischen Benutzungsschnittstellen- und Anwendungsobjekten bei Veränderungen betreffen.

Die Entscheidung zwischen der Verwendung einer Dialogspezifikationssprache oder eines Anwendungsrahmens erfordert ein Abwägen zwischen der gewünschten Einfachheit und der erforderlichen Mächtigkeit auf der Ebene der Dialogspezifikation. Dialogspezifikationssprachen bieten gegenüber allgemeinen Programmiersprachen den Vorteil des höheren Abstraktionsniveaus und damit der einfacheren Benutzbarkeit. Nachteilig ist aber die eingeschränkte Mächtigkeit und die Tatsache, daß zusätzlich zur Oberflächen- und zur Anwendungsprogrammiersprache (die, z.B. bei der Verwendung von NeWS, unterschiedlich sein können, s.u.) eine weitere Sprachebene dazukommt. Diese Nachteile werden bei Anwendungsrahmen vermieden. Anders als in den bisher bekannten Anwendungsrahmen sollte aber eine konzeptionelle wie auch implementationstechnische Trennung der Ebenen der Präsentation und Dialogsteuerung in der Benutzungsschnittstelle vorgenommen werden.

4. Oberfläche eines ikonischen Dateisystems

Als Implementationsbeispiel für eine direkt manipulative Oberfläche wird hier das ikonische Dateisystem (siehe Abb. 3) weiter ausgeführt, da die Interaktionstechnik des Bewegens von Ikonen und der damit verbundenen semantischen Rückkopplung für Direkte Manipulation typisch ist. Für die Benutzungsoberfläche eines ikonischen Dateisystems sind im wesentlichen Bausteinklassen für Fenster, Ikone und Menüs erforderlich. Die folgende Darstellung ist weitgehend auf die Ikone beschränkt, da die anderen Bausteine bereits zum Stand der Technik in heutigen Oberflächenbaukästen gehören.

Die Implementation erfolgte auf der Basis des Fenstersystems NeWS (Network extensible Window System; Gosling et al., 1989). Die Bausteinklassen wurden in PostScript (Adobe, 1985) programmiert, für das unter NeWS eine objektorientierte Erweiterung verfügbar ist.

Die Funktionalität für die Selektierbarkeit ist in einer generischen Klasse "Subview" (Abb. 4) enthalten, um außer von Ikonen auch von anderen Klassen von Objekten verwendet werden zu können. Ein Subview besteht im wesentlichen aus einer Darstellungsfläche ("subviewcanvas") und aus Methoden für die Formfestlegung ("reshape") und die Zeichnung ("draw"). Letztere sind in den Unterklassen zu definieren, da verschiedene Sorten von "Subview" unterschiedliche graphische Darstellungen haben.

```
/Subview Object          %Klassenname, Oberklasse
dictbegin                      %Instanzenvariablen
   /objid null def             %Bezeichner des Objektes
   /subviewcanvas null def     %Darstellungsfläche f. d. Objekt
   /superview null def         %Fenster, zu dem d. Objekt gehört
   ...
dictend
classbegin               %Hier beginnen die Klassenmethoden
   ...
   /selection {                %Ereignisroutine für die Selektion.
      highlight                          %Invers darstellen
      subviewcanvas canvastotop          %Nach vorne holen
      self /setselection superview send  %Fenster benachrichtigen
   } def

   /highlight {                %Zeichne invertiert
      /background 0 store
      /foreground 1 store
      draw
   } def
   ...
classend def             %Ende der Klassendefinition
```

Abb. 4: Skizze der Klasse "Subview"

Objekte der Klasse "Subview" gehören zu einem Fensterobjekt, welches die innerhalb des Fensters globale Steuerung im Zusammenhang mit der Selektion (Deselektion der alten Selektion) sowie die Meldung der Selektion an die Dialogsteuerung übernimmt.

Die Klasse "Icon" ist eine Unterklasse von "Subview" und enthält zusätzlich allgemeine Funktionalität für die Bewegung von Ikonen innerhalb eines Fensters und für die Steuerung der semantischen Rückkopplung. Die hierfür notwendige globale Steuerung sowie die Meldung einer semantisch bedeutsamen Bewegung an die Dialogsteuerung übernimmt wiederum das zugehörige Fenster.

Als spezielle Ikone wurden Dokumente und Ordner implementiert, für die es jeweils eine eigene Darstellung und spezifische Eigenschaften in bezug auf die semantische Rückkopplung gibt.

Aufgrund der vordefinierten Funktionalität in den Oberklassen umfaßt der Programmtext für jedes Ikon nur noch eine Seite.

Es zeigte sich, daß NeWS als Basis für die Implementation direkt manipulativer Systeme besonders geeignet ist. Da PostScript als Protokoll zwischen dem NeWS-Server und den Anwendungsprozessen verwendet wird, ist die Funktionalität des Servers erweiterbar, so daß die gesamte Oberfläche innerhalb des Serverprozesses ablaufen kann. Dies bietet Effizienzvorteile gegenüber Fenstersystemen mit festem Protokoll (z.B. dem X-Fenstersystem; Scheifler und Gettys, 1986), bei denen die Oberflächenobjekte dem Anwendungsprozß zugeordnet sind. Bei der Ikonenbewegung ist z.B. das Bewegen des gesamten Ikons (und nicht nur eines Rahmens) als Rückkopplung möglich. Ein weiterer Vorteil ist, daß als äußere Begrenzung von Darstellungsflächen beliebige Formen zugelassen sind. Dadurch entspricht die graphische Darstellung etwa eines Ikons seinem maussensitiven Bereich. Schließlich steht unter NeWS gegenüber anderen Umgebungen eine echt objektorientierte Implementationssprache für die Oberfläche zur Verfügung.

5. Ausblick

Die in diesem Beitrag beschriebene Werkzeugunterstützung ist natürlich noch nicht vollständig. Es fehlen z.B. bisher Möglichkeiten der Mehrfachselektion und -bewegung von Ikonen, sowie die Bewegung eines Ikons aus dem Fenster heraus, was aber keine prinzipiellen Probleme aufwirft. Ferner fehlt eine detailliertere Ausarbeitung der Werkzeugunterstüzung auf der Dialogebene. In einer vollständigen Entwicklungsumgebung müßten außerdem weitere Werkzeuge, wie z.B. graphische Oberflächeneditoren für Formulare, zur Verfügung stehen, die hier vernachlässigt wurden.

Der hier beschriebene Architekturansatz für direkt manipulative Systeme muß sich in der Praxis der Systementwicklung noch bewähren. Insbesondere sind folgende Punkte interessant:

- Ist das Antwortzeitverhalten bei der Verteilung von Anwendung und Oberfläche in realen Anwendungen akzeptabel? Für Rückkopplungen, die von der Oberfläche gesteuert werden, ist dies nach den hier gemachten Erfahrungen der Fall. Bei vollständigen Interaktionszyklen mit Durchgriff auf die Anwendung ist aber noch eine Überprüfung erforderlich.

- Inwieweit führt der parallele Zugriff mehrer Benutzer auf eine Anwendung zu untragbarer Verwirrung? Es wird hier eine Erweiterung der Direkten Manipulation vorgenommen, bei der bisher davon ausgegangen wurde, daß der Benutzer in einer virtuellen Welt allein agiert. Eventuell muß es eine Anzeige für die anderen Benutzer geben, wenn jemand eine Operation mit einem Objekt beginnt. Eine andere Möglichkeit wären Absprachen über andere Medien (z.B. Telefon).

6. Literatur

Adobe Systems (1985). PostScript Language Reference Manual. Reading: Addison Wesley.

Behr, J.P., Fink, B., Krämer, R., Stecher, R (1988). A Distributed Systems Architecture Supporting Multy-Threaded Objects. In: Valk, R. (Hrsg.), GI - 18. Jahrestagung. Band II. Berlin: Springer, 1988, 576-588.

Gosling, J., Rosenthal, D.S.H., Arden, M.J. (1989). The NeWS Book. An Introduction to the Network/extensible Window System. New York, Berlin, Heidelberg: Springer.

Green, M. (1985a). Report on Dialogue Specification Tools. In: Pfaff (1985), 9-20.

Green, M. (1985b). Design Notations and User Interface Management Systems. In: Pfaff (1985), 89-107.

Herczeg, M. (1989). USIT - Ein Benutzerschnittstellen-Baukasten für ein Interaktionskontinuum. In: Maaß und Oberquelle (1989), 254-263.

Hudson, S.E. (1987). UIMS Support for Direct Manipulation Interfaces. Computer Graphics 21 (2), 1987, 120-124.

Ilg, R., Ziegler, J. (1988). Direkte Manipulation. In: Balzert, H. et al. (Hrsg.). Einführung in die Software-Ergonomie. Berlin, New York: de Gruyter, 1988, 175-194.

Janssen, C. (1989). Eine objektorientierte Architektur für direkt manipulative, verteilte Bürosysteme. Diplomarbeit. Universität Hamburg, Fachbereich Informatik.

Krasner, G.E., Pope, S.T. (1988). A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. Journal of Object-Oriented Programming 1(3), 1988, 26-49.

Maaß, S., Oberquelle, H. (Hrsg.) (1989). Software-Ergonomie'89. Stuttgart: Teubner.

Myers, B.A. (1989). User Interface Tools: Introduction and Survey. IEEE Software 6(1), 1989, 15-23.

Pfaff, G. (Hrsg.) (1985). User Interface Management Systems. Berlin: Springer.

Rosenberg, J. (Moderator) (1988). Panel: UIMSs: Threat or Menace? CHI'88 Conference on Human Factors in Computing Systems, 197-200.

Scheifler, R.W., Gettys, J. (1986). The X Window System. ACM Transactions on Graphics 6(3), 1986, 79-109.

Schmucker, K. (1986). MacApp: An Application Framework. BYTE 11(8), 1986, 189-193.

Shneiderman, B. (1982). The Future of Interactive Systems and the Emergence of Direct Manipulation. Behaviour and Information Technology 1(3), 1982, 237-256.

Shneiderman, B. (1983). Direct Manipulation - A Step Beyond Programming Languages. IEEE Computer 16 (8), 1983, 57-69.

Trefz, B., Ziegler, J. (1989). DIAMANT - Ein User Interface Management System für graphische Benutzerschnittstellen. In: Maaß und Oberquelle (1989), 264-273.

Heutige Adresse des Autoren: Christian Janssen
Fraunhofer Institut für Arbeitswirtschaft
und Organisation (IAO)
Nobelstraße 12
7000 Stuttgart 80