Fenstersysteme im Vergleich - Architektur, Leistungsfähigkeit und Eignung für die Anwendungsentwicklung *

H. Balzert; Triumph-Adler AG, Nürnberg
H. U. Hoppe, J. Ziegler; Fraunhofer-Institut für Arbeitswirtschaft und Organisation (IAO), Stuttgart

Zusammenfassung: Fenstersysteme bilden einen wichtigen Bestandteil interaktiver, insbesondere direkt-manipulativer Benutzerschnittstellen. Im Rahmen dieses Beitrages werden ausschließlich "offene Systeme" zum Vergleich herangezogen, d.h. solche, die sowohl programmierbar als auch unabhängig von speziellen Anwendungen sind. Hinsichtlich der Verwendbarkeit für die kommerzielle Software-Entwicklung läßt sich ein erheblicher Mangel an Standardisierung feststellen. Dies gilt für die allgemeine Funktionalität ebenso wie für die Schnittstellen zu physikalischen E/A-Medien, zur Anwendung bzw. zu höheren Dialogebenen. Aufgrund der Vergleichsergebnisse wird eine Liste von Anforderungen an die Funktionalität von Fenstersystemen erstellt.

1 Die Bedeutung von Fenstersystemen für die Gestaltung von Benutzerschnittstellen

Fenstersysteme bilden einen wesentlichen Bestandteil fortgeschrittener Benutzerschnittstellen. Durch Fenstertechniken kann der Bildschirm voll zur zweidimensionalen Repräsentation von Informationen genutzt werden, durch Überlappung von Fenstern erhält man sogar eine quasi dreidimensionale ("zweieinhalb-dimensionale") Darstellung. CARD et al. /1/ heben folgende Funktionsmerkmale von Fenstersystemen gegenüber herkömmlichen Techniken hervor:

- Größeres Informationsangebot und bessere Nutzung des Bildschirms, insbesondere bei überlappenden Fenstern,
- direkter Zugriff auf verschiedene Informationsquellen zur gleichen Zeit,
- Integration und Austausch von Information aus verschiedenen Quellen durch bildschirmorientierte Interaktion,
- gleichzeitige Kontrolle über mehrere Prozesse möglich,
- Hilfe und Erinnerung durch spezielle Fenster,
- Fenster als Rahmen für bestimmte Interaktionsmöglichkeiten, z.B. aktive Formulare,
- multiple, simultane Repräsentation von Objekten und Aufgaben, z.B. verschiedene externe Darstellungen ("Sichten") ein und desselben Objekts.

---

* Dieser Beitrag beruht auf Arbeiten im Rahmen des vom BMFT geförderten Forschungsvorhabens WISDOM ("Wissensbasierte Systeme zur Bürokommunikation: Dokumentenbearbeitung, Organisation, Mensch-Computer-Kommunikation"). Neben den Autoren haben bei der Analyse der zum Vergleich herangezogenen Systeme weiterhin mitgewirkt: F. Fabian (Universität Stuttgart) ; H. P. Frejek (TA, Nürnberg) ; T. Kreifelts, C. T homas , G. Woetzel (GMD, Birlinghoven).

Fenstersysteme erlauben die Verwendung des Bildschirms zur parallelen Repräsentation von Arbeitsinformationen und bieten dadurch die Möglichkeit, neue Konzepte wie z. B. die Metapher des elektronischen Schreibtisches zu verwirklichen. Diese Schreibtisch-Metapher bietet sich insbesondere für die Schnittstelle zu integrierten Bürosystemen an. Die ersten Bürosysteme mit einer derartigen Schnittstelle waren XEROX STAR /2/ und APPLE LISA /3/, die wiederum beide auf Erfahrungen bei der Entwicklung der interaktiven Programmierumgebung SMALLTALK /4/ beruhen.

Im Rahmen des vorliegenden Beitrages werden ausschließlich solche Fenstersysteme betrachtet, die "offen" (im Sinne der Programmierbarkeit) sind und somit als Werkzeuge für die Software-Entwicklung verwendet werden können. Ein Fenstersystem als Komponente einer Benutzerschnittstelle dient zunächst zur Organisation der Ausgabe auf den Bildschirm. Es ermöglicht die Unterteilung des Bildschirms in mehrere "virtuelle Bildschirme" (Fenster), die von den Anwendungen als Ausgabemedien genutzt werden können.

Ein maßgebliches Prinzip für die Gestaltung von Benutzerschnittstellen besteht darin, eine möglichst weitgehende Unabhängigkeit zwischen den verschiedenen Dialogebenen wie auch gegenüber den Anwendungen herzustellen (vgl. dazu FÄHNRICH und ZIEGLER /5/). In diesem Sinne sollten Fenstersysteme über eine standardisierte Schnittstelle für die Ansteuerung von fensterorientierten Display-Operationen verfügen und Unabhängigkeit gegenüber verschiedenen E/A-Medien gewährleisten.

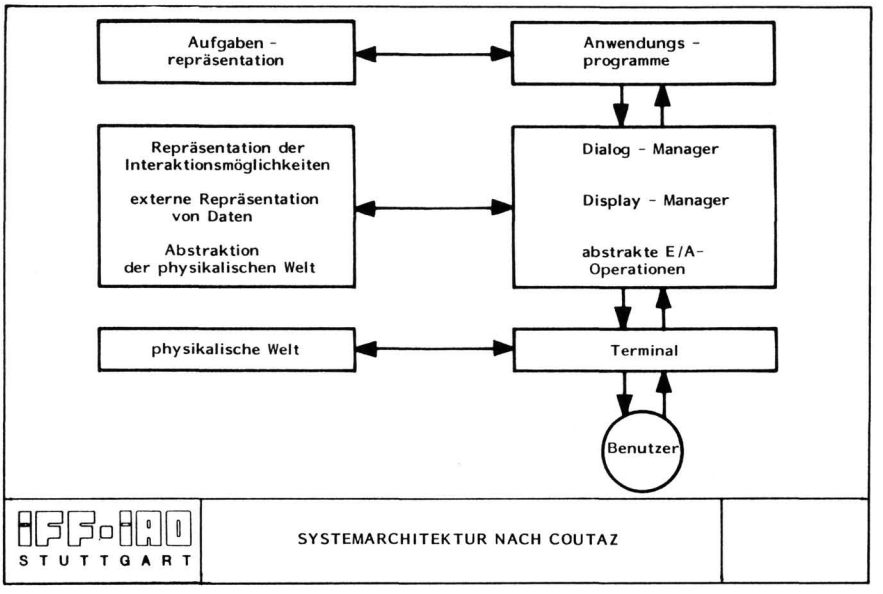

Bild 1: Systemarchitektur nach COUTAZ

Wenn wir das Verständnis von Fenstern als virtuelle Bildschirme zugrundelegen, läßt sich die Funktionalität des Fenstersystems auf solche Operationen eingrenzen, die lediglich die externe Sicht auf die dahinterliegenden Datenobjekte verändern, nicht aber diese Objekte selbst manipulieren. Typische Fensteroperationen sind z.B. das Verschieben oder Nach-Vorn-Holen im Gegensatz etwa zu einem Eintrag in ein Formular. Von daher erscheint es angebracht, diese konzeptionelle Trennung zwischen Operationen auf internen Datenobjekten und Veränderungen der externen Repräsentation auch in der Software-Architektur zu berücksichtigen. Bild 1 gibt einen Vorschlag zur Systemarchitektur in Anlehnung an COUTAZ /6/ wieder. Dabei ist das Fenstersystem der mittleren Schicht zuzuorden.

In den hier betrachteten Systemen besteht ein einzelnes Fenster jeweils aus einer Rechteckregion (Fensterinhalt), die den eigentlichen virtuellen Bildschirm für die Anwendung darstellt, und einem Rahmen, der anwendungsunabhängige Strukturelemente wie Titelleiste, Scroll-Balken usw. enthalten kann. Strukturinformationen über den Fensterinhalt liegen bei der Anwendung und sollten daher nicht vom Fenstersystem verwaltet werden. Auf der Ebene des Fenstersystems sollte eine möglichst unspezifische Repräsentation des Fensterinhalts verwendet werden. Typischerweise geschieht dies durch Bitmaps.

Weiterhin gehört es zu den originären Aufgaben eines Fenstersystems, alle Informationen über die Darstellungsstruktur von Fenstern zu verwalten. Dies betrifft sowohl die Anordnung der Fenster auf dem Bildschirm (z.B. Überlappung) als auch die speziellen Darstellungsattribute einzelner Fenster (z.B. Art des Rahmen, Titelleiste, maus-sensitive Bereiche für Fensteroperationen, Weiterunterteilungen). Die Verwaltung des Fensterinhalts als Bitmap ist zwar unter dem Gesichtspunkt der Anwendungsunabhängigkeit unbedingt sinnvoll, führt aber insbesondere beim Vergrößern oder Verschieben des sichtbaren Ausschnitts zu Schwierigkeiten. Aus Speicherplatzgründen ist es nicht vertretbar, für jedes Fenster ein Bitmap maximaler Größe zu halten. Daher müssen die bei der Ausschnittsveränderung neu hinzutretenden Teile jeweils durch Rückgriff auf die internen Datenstrukturen der Anwendung erzeugt werden. Der direkte Rückbezug vom Fenstersystem auf die Anwendung kann durch die Einführung eines "Display-Managers" vermieden werden, der die internen Daten in eine für die externe Darstellung geeignete Standard-Repräsentation für Text und Grafik überführt und bei Bedarf die entsprechenden Bitmaps erzeugt.

Die Operationen des Verschiebens und Verkleinerns kommen mit der zu dem sichtbaren Ausschnitt gehörigen Bitmap aus, verändern aber ggf. die Überdeckungsstruktur der auf dem Bildschirm sichtbaren Fenster. Wenn jedoch ein einmal erzeugter Fensterinhalt weiterhin als Bitmap verfügbar gehalten wird, kann in diesen Fällen das Auffrischen des Bildschirms grundsätzlich ohne Rückgriff auf außerhalb des Fenstersystems liegende Daten erfolgen. Wir sprechen dann von "automatischen Redraw", wenn die Grundoperationen für das Verschieben und Verkleinern die automatische Ausführung dieses Vorgangs einschließen.

2 Systemvergleich

2.1 Auswahl der Systeme und Ziele des Vergleichs

Die Auswahl der zum Vergleich herangezogenen Systeme erfolgte nach den Kriterien Marktrelevanz, Repräsentativität für eine bestimmte Klasse von Arbeitsplatzrechnern sowie Verfügbarkeit im Rahmen des Verbundprojektes WISDOM. Folgende sieben Systeme wurden berücksichtigt:

- APPLE MACINTOSH/LISA
- BITGRAPH-Fenstersystem der Universität Stuttgart (BITG)
- Digital Research Graphics Environment Manager (GEM)
- MICROSOFT WINDOWS (MS-WINDOWS)
- PERQ Raster-Workstation
- SYMBOLICS 36xx LISP-Maschine
- XEROX 1108 mit Interlisp-D

Eine klare Einschätzung der Bedeutung von Fenstersystemen für die Gestaltung von Mensch-Computer-Schnittstellen stellt eine wichtige Voraussetzung für zukünftige Weiterentwicklungen z.B. in Richtung auf Dialoggeneratoren oder "User Interface Management Systems" (UIMS - vgl. HAYES /7/) dar. Die vorliegende Arbeit soll in diesem Zusammenhang zur Beantwortung der folgenden Fragen beitragen:

- Läßt sich für die untersuchten Systeme ein gemeinsamer Standard hinsichtlich der Grundfunktionalität für den Anwendungsentwickler bzw. den Endbenutzer angeben?

- Besitzen die Systeme klare Schnittstellen sowohl zur physikalischen E/A-Schicht als auch zu den übrigen Komponenten der Dialogschicht und zur Anwendungskomponente?

- In welchem Verhältnis steht die Portabilität der Fenstersysteme zum Komfort und der Leistungsfähigkeit der jeweiligen Entwicklungsumgebung?

- Welche speziellen Anforderungsprofile an Fenstersystemen ergeben sich aus verschiedenen Anwendungszusammenhängen?

- Wie kann in Zukunft eine höhere begriffliche Konsistenz und Übertragbarkeit in der Benutzersicht auf verschiedene Fenstersysteme gewährleistet werden?

2.2 Erläuterungen zu den einzelnen Vergleichskriterien

Wir betrachten hier Fenstersysteme in ihrer Funktion als Hilfmittel für den Anwendungsentwickler beim Entwurf und der Implementierung von Benutzerschnittstellen. Dabei wird ein gewisser Teil der "Oberflächenfunktionalität" durch das Fenstersystem vorgegeben, z.B. Standard-Fenstertypen, interaktive Grundoperationen auf Fensterebene, Menüs usw.. Der Entwicklungsaufwand ist umso geringer, je mehr Unterstützung das Fenstersystem in dieser Hinsicht bietet. Gerade durch ein komfortables Entwicklungssystem wird also die Benutzeroberfläche in

ganz erheblichem Maße vorherbestimmt. Von daher ist es sinnvoll, neben der Funktionalität eines Fenstersystems aus der Sicht des Anwendungsentwicklers auch die Funktionalität aus der Sicht des Endbenutzers zum Vergleich heranzuziehen. Der letztere Aspekt kann sich natürlich nur auf diejenigen Komponenten des Systems beziehen, die standardmäßig verfügbar sind und ohne zusätzlichen Programmieraufwand unmittelbar in die Benutzerschnittstelle übernommen werden können.

Im Bereich der Grundfunktionalität gibt es eine Reihe von Fensteroperationen, die sowohl in interaktiver Form für den Endbenutzer als auch als Bestandteil der Programmierschnittstelle für den Anwendungsentwickler angeboten werden.Dazu gehören solche Operationen wie Aktivieren, Öffnen / Schließen oder Verschieben eines Fensters. In der Regel wird der Funktionsumfang für den Anwendungsentwickler mächtiger sein als der für den Endbenutzer. Im Einzelfall kann es jedoch auch vorkommen, daß ein System bestimmte interaktive Fensteroperationen für den Endbenutzer standardmäßig vorgibt, ohne diese an der Programmierschnittstelle verfügbar zu machen. Dies führt zu einer Standardisierung der Dialogoberfläche - allerdings auf Kosten der Flexibilität für den Anwendungsentwickler.

Operationen auf einer höheren Ebene (Metakommunikation) dienen dazu, bestimmte Interaktionseigenschaften von Fenstern zu definieren und die globale Dialoggestaltung für die Bildschirmoberfläche festzulegen. Dazu gehört insbesondere die Definition von Fenstertypen. Von Seiten des Anwendungsentwicklers geschieht dies in der Regel durch Rückgriff auf vorgegebene Datenstrukturen für Standard-Fenstertypen. Die Definition eines neuen Typs kann dann entweder durch Verfeinerung oder durch Modifikation erfolgen. Der Verfeinerungsansatz geht von einfachen Standards aus, die nur über die allgemein notwendige Grundfunktionalität verfügen und beliebig erweiterbar sind. Im Falle der Modifikation ist die Obergrenze der erreichbaren Funktionalität durch den Standardtyp festgelegt, und neue Fenstertypen werden durch Änderung von Default-Parametern erzeugt. Für den Verfeinerungsansatz spricht die höhere Flexibilität, für die Modifikation dagegen die einfachere Handhabung und die konsistente Darstellung an der Benutzeroberfläche.

Meta-Operationen auf der Benutzerseite müssen in interaktiver Form vorliegen. Als besonders flexible Lösung, die allerdings in keinem der untersuchten Systemenen verwirklicht ist, kann man sich einen interaktiven "Fensterbaukasten" vorstellen, mit dem sich der Benutzer den gewünschten Fenstertyp aus Standard-Elementen wie Scroll-Balken, maus-sensitiven Regionen für das Verschieben und die Größenänderung etc. zusammensetzen kann. Weniger umfassende Meta-Operationen für den Endbenutzer sind z.B. das Festlegen eines Standard-Bildschirm-Layouts oder die Änderung bestimmter Parameter für Fensteroperationen wie etwa die beim Verschieben selektierte Position (links unter - rechts oben).

Im Bereich der Zusatzfunktionalität werden sowohl anwendungsunabhängige Erweiterungen des Fenstersystems (Menüs, Piktogramme, Schnittstelle zu Hilfeinformationen) als auch anwendungsspezifische Zusätze (Masken- und Formularverwaltung) berücksichtigt. Letztere werden in einigen der untersuchten Systeme angeboten, obwohl sie im Sinne unserer Definition nicht zum eigentlichen Fenstersystem gehören.

Die unter der Rubrik "Systemeinbettung und Implementierungsmerkmale" auzfgeführten Eigenschaften geben Auskunft über die Leistungsfähigkeit sowie Hardware- und Betriebssystem-Voraussetzungen der jeweiligen Entwicklungsumgebung. Die Portabilität eines Systems hängt wesentlich von der Einbettung in ein Standard-Betriebssystem ab. Falls das Fenstersystem an vorhandene Anwendungs-Software angebunden werden soll (vor allem in kommerziellen Systemen) sind entsprechende Kommunikationsmöglichkeiten - z.B. "pipes" in UNIX - erforderlich. Für die schnelle Prototyp-Entwicklung und die experimentelle Erprobung verschiedener Konzepte für Benutzerschnittstellen ist die Integration des Fenstersystems in eine komfortable Programmierumgebung von entscheidender Bedeutung.

Unter "Besondere Merkmale" werden schließlich noch solche Gesichtspunkte wie Antwortzeitverhalten, Anpassung an verschiedene E/A-Medien und Dialogfunktionalität zusammengefaßt.

2.3 Ergebnisse des Systemvergleichs

(1) Funktionalität für den Endbenutzer

Hinsichtlich der allgemeinen Operationen besteht weitgehende Übereinstimmung zwischen den untersuchten Systemen. Bei GEM, LISA und XEROX ist keine explizite Operation für das Nach-Hinten-Bringen vorgesehen. MS-WINDOWS bildet insofern eine Ausnahme, als keine Fensterüberlappung möglich ist und deshalb die Operationen Nach-Vorn-Holen und Nach-Hinten-Bringen entfallen. Zwei der untersuchten Systeme (PERQ, SYMBOLICS) verfügen über keine eingebaute Möglickeit, Fenster durch Piktogramme (Icons) zu repräsentieren. Abgesehen von MS-WINDOWS sind Piktogramme bei allen übrigen Systemen frei positionierbar. MS-WINDOWS reserviert einen bestimmten Bildschirmbereich für die Darstellung von Piktogrammen. Keines der Systeme verfügt über eine Zoom-Funktion.

Möglichkeiten der Metakommunikation für den Endbenutzer sind bei den untersuchten Systemen kaum vorhanden. Lediglich die SYMBOLICS LISP-Maschine bietet benutzerseitig einige Metaoperationen an.

(2) Funktionalität für den Anwendungsentwickler

Die meisten Systeme stellen die gleichen Operationen, die dem Benutzer interaktiv zugänglich sind, auch in funktionaler bzw. kommandosprachlicher Form an der Programmierschnittstelle zur Verfügung. Bei MS-Windows, das keine überlappenden Fenster zuläßt, werden Fenster automatisch positioniert. Der Anwendungsentwickler kann als Option lediglich die Fenstergröße spezifizieren. Die Entwiclungsumgebung auf APPLE LISA bietet kein automatisches Redraw in der in Abschnitt 2 erläuterten Form. Der Anwendungsentwickler muß bei diesem System auch im trivialen Fall des Verschiebens die Redraw-Operation explizit programmieren.

Bei BITG und SYMBOLICS erfolgt die Definition neuer Fenstertypen durch Vererbung von Objekteigenschaften. Durch diese Art der Erweiterung können Fenster mit beliebig komplexen Eigenschaften kreiert werden. In allen anderen Systemen werden neue Fenstertypen nach dem Modifikationsprinzip definiert. Die Zahl der vorgegebenen Standardtypen ist in diesem

Zusammenhang wenig aussagekräftig. Wichtiger ist vielmehr die Reichhaltigkeit der Standardfenster, d.h. die Zahl der veränderbaren Parameter. Allen untersuchten Fenstersystemen ist die Unterteilung von Fenstern in eine Inhaltsregion und einen Rahmen, der zumindest eine Titelleiste enthalten kann, gemeinsam. GEM und APPLE LISA geben spezielle Dialogfenster ("dialogue boxes") für Systemmeldungen und Anfragen an den Benutzer vor. Die Beziehung zwischen Fenstern und Menüs ist nicht einheitlich geregelt (s.u.).

Das einzige System, das mit einer automatischen Plazierung von Fenstern arbeitet, ist MS-WINDOWS. Dies ergibt sich daraus, daß keine Überlappung von Fenstern vorgesehen ist. Alle übrigen Systeme erlauben standardmäßig die Plazierung von Fenstern durch das Anwendungsprogramm oder auf interaktivem Wege durch den Benutzer, aber keine automatische Positionierung. Ein "split screen"-Mechanismus existiert nur in MS-WINDOWS, bei PERQ und SYMBOLICS. Eine Weiterunterteilung von Fenstern ist bei MS-WINDOWS, BITG, SYMBOLICS und XEROX vorgesehen. Bis auf PERQ besitzen alle Systeme eine integrierte Menüverwaltung. Hinsichtlich der Realisierung von Menüs bestehen jedoch erhebliche Unterschiede.

GEM und APPLE LISA arbeiten mit sogenannten "pull down"-Menüs, die an einer festen Stelle auf einer Menüliste am oberen Bildrand "heruntergezogen" werden können. MS-WINDOWS verwendet statische Menüs, die ständig sichtbar am unteren Rand eines Fensters plaziert werden. Die übrigen Systeme arbeiten mit "pop up"-Menüs die an beliebigen Stellen des Bildschirms durch Mausklick geöffnet werden.

In allen Systemen kann eine Anwendung mehrer Fenster benutzen. Die gleichzeitige Verwaltung verschiedener Prozesse in einem oder mehreren Fenstern ist nur bei PERQ, SYMBOLICS und XEROX möglich. Die Übertragung von Fensterinhalten in andere Fenster kann vom Fenstersystem selbst nur auf Bitmap-Ebene unterstützt werden. Entsprechende Mechanismen sind in GEM, MS-WINDOWS sowie bei LISA und XEROX vorgesehen.

(3) Systemeinbettung und Architektur

Die SYMBOLICS LISP-Maschine ist dasjenige System, das am einzelnen Arbeitsplatz die höchste Speicherkapazität und Prozessorleistung zur Verfügung stellt. Die Hardware ist speziell für LISP-Programmierung ausgelegt und besitzt außerdem einen zusätzlichen E/A-Prozessor (68000), der einen extrem schnellen Graphikaufbau ohne Beanspruchung der CPU-Zeit des Hauptprozessors ermöglicht. Als einziges mehrplatzfähiges System wurde das Bitgraph-Fenstersystem (BITG) betrachtet. Es basiert auf einer VAX unter UNIX mit Bitgraph-Terminals von BBN.

In puncto Hardware-Leistung ergibt sich folgende Rangfolge für die Einplatzsysteme: 1. SYMBOLICS, 2. XEROX 1108, 3. PERQ, 4. APPLE LISA /MAC, 5. MS-WINDOWS, GEM. BITG muß als Mehrplatzsystem hier gesondert betrachtet werden, da das Laufzeitverhalten selbstverständlich von der Prozessor-Auslastung abhängig ist.

Vier der betrachteten Fenstersysteme basieren auf Standard-Betriebssystemen: GEM und MS-WINDOWS auf MS-DOS, BITG und PERQ auf UNIX. Lediglich GEM und MS-WINDOWS sind jedoch unabhängig von einer speziellen Programmierumgebung. Das PERQ-Fenstersystem wird in C programmiert. Das BITG-System ist in die objektorientierte Programmierumgebung ObjTalk

eingebettet. APPLE verwendet ein spezielles Betriebssystem und eine Entwicklungsumgebung mit Pascal als Programmiersprache. Bei XEROX und SYMBOLICS schließt eine komfortable LISP-Programmierumgebung (Interlisp-D bzw. Zetalisp) sowohl das Betriebssystem als auch das Fenstersystem ein.

Die Programmiereigenschaften der verschiedenen Systeme lassen sich in das Spektrum zwischen objektorientierter und prozedurorientierter Programmierung einordnen. BITG ist mit der objektorienterten FranzLisp-Erweiterung ObjTalk am stärksten objektorientiert, gefolgt von SYMBOLICS mit dem Flavor-Konzept in Zetalisp. Eindeutig prozedurorientiert sind MS-WINDOWS, GEM, PERQ und das APPLE-Entwicklungssystem. Das XEROX Interlisp-System unterstützt objektorientierte Programmiertechniken nicht explizit. Allerdings wird eine objektorientierte Erweiterung (LOOPS) angeboten.

Die meisten der untersuchten Fenstersysteme sind in eine spezielle Programmierumgebung eingebettet. Unter diesen Voraussetzungen ist die Unabhängigkeit des Fenstersystems von den jeweiligen Anwendungen eine Frage des Programmaufbaus. In einer Entwicklungsumgebung, die nur Basis-Routinen für Fenstermanipulation bereitstellt, muß der Programmierer zunächst Zwischenschichten schaffen, um einen komfortablen Zugriff von der Anwendung auf das Fenstersystem zu ermöglichen. Objektorientierte Systeme stellen in dieser Hinsicht bereits hochstehende Programmierwerkzeuge zur Verfügung.

Im Sinne einer geschichteten Systemarchitektur arbeiten nur GEM und MS-WINDOWS mit einem geräteunabhängigen Ein-/Ausgabe-Protokoll ("Virtual Device Interface" bzw. "Graphics Device Interface"). Bei allen anderen Systemen fehlt die Möglichkeit, die E/A-Schicht als virtuelle Maschine anzusprechen. Entsprechende Schnittstellen müßten bei diesen Systemen explizit programmiert werden.

Mit Ausnahme von GEM, MS-WINDOWS und PERQ, die unter MS-DOS bzw. UNIX relativ leicht portiert werden können, ist die Portabilität der untersuchten Fenstersysteme als schlecht bis sehr schlecht einzuschätzen.

(4) Besondere Merkmale

Aufgrund ihrer besonders leistungsfähigen Hardware rangiert die SYMBOLICS Lisp-Maschine hinsichtlich des Antwortzeitverhaltens klar an der Spitze. Am unteren Ende der Rangfolge liegen GEM (auf Systemen mit 8086-Prozessor) und BITG (im Mehrbenutzer-Betrieb). Das System APPLE LISA/MAC zeichnet sich durch recht schnelle Graphik, aber langsame Disk-E/A aus. Obwohl die Hardware-Basis der XEROX Lisp-Maschine nicht optimal ist, läßt sich mit kompiliertem Lisp-Code auch bei diesem System ein recht gutes Antwortzeitverhalten erreichen.

Alle untersuchten Systeme unterstützen zumindest eine bestimmte E/A-Konfiguration und - mit Ausnahme von BITG - auch die Anpassung an verschiedene Drucker. Lediglich drei Systeme (GEM, PERQ, SYMBOLICS) lassen sich auch an verschiedene Maustypen und ebenfalls drei (GEM, MS-WINDOWS, BITG) an verschiedene Bildschirme anpassen. Die Integration neuer E/A-Medien wie Spracheingabe wird bisher nicht hinreichend unterstützt.

3 Auswertung und Aufstellung eines Anforderungenkataloges

Die hier verglichenen Systeme lassen sich eindeutig zwei unterschiedlichen Kategorien zuordnen:

(A) Systeme, die vor allem für wissenschaftliche Anwendungen, insbesondere im KI-Bereich, und für "rapid prototyping" genutzt werden (BITG, SYMBOLICS, XEROX).

(B) Systeme für kommerzielle Anwendungen, deren wesentlicher Einsatzbereich heute vor allem bei der integrierten Dokumentenbearbeitung liegt (APPLE LISA/MAC, GEM, MS-WINDOWS, PERQ).

Die Systeme der Kategorie (A) zeichnen sich durch eine ausgesprochen hochstehende Unterstützung des Anwendungsentwicklers durch eine große Zahl von teilweise sehr speziellen Fensteroperationen und flexibel erweiterbaren Fenstertypen aus. Das Fenstersystem ist hier eingebettet in eine hochstehende interaktive Programmierumgebung (LISP, ObjTalk) und infolgedessen in keiner Weise portabel. Typische Anwendungen für solche Systeme sind etwa die Gestaltung von Benutzerschnittstellen für Expertensysteme und wissensbasierte Hilfesysteme oder auch die experimentelle Realisierung und Erprobung neuer Formen der Mensch-Computer-Kommunikation.

Da die Systeme der ersten Gruppe im wesentlichen zu experimentellen Zwecken eingesetzt werden, müssen sie vor allem flexible Gestaltungsmöglichkeiten und hohen Programmierkomfort bieten. Standardisierungen der Benutzeroberfläche sind hier gerade nicht wünschenswert. Portabilität und Hardware-Unabhängigkeit sind von untergeordneter Bedeutung. Trotzdem sollten auch in solchen Systemen klare Schnittstellen zwischen dem Fenstersystem, der Anwendungskomponente und dem Dialog-Manager definiert und eingehalten werden, um eine spätere Übertragung in kommerzielle Produkte von der Software-Architektur her zu unterstützen. Adäquate Mechanismen zur Erzeugung einer externen (Bitmap-) Repräsentation aus internen Datenstrukturen verschiedener Anwendungen werden derzeit erst entwickelt. Auch dies ist ein wichtiger Anwendungsbereich für experimentelle Systeme mit unmittelbarem Bezug zur Weiterentwicklung von Fenstersystemen.

Fenstersysteme der Kategorie (B) sollten als Werkzeuge für die kommerzielle Software-Entwicklung die Möglichkeit bieten, vorhandene Anwendungsprogramme unter einer einheitlichen Benutzerschnittstelle zu integrieren. Dabei sollten sowohl die Schnittstelle für den Anwendungsprogrammierer als auch die Benutzeroberfläche sinnvollen Standardisierungen unterworfen werden, um eine begriffliche Konsistenz auch unabhängig von speziellen Systemumgebungen zu gewährleisten. Dies läßt sich mit vertretbarem Aufwand nur dann erreichen, wenn die Systeme portabel und hardware-unabhängig sind. Erste Schritte in Richtung auf Portabilität und Hardware-Unabhängigkeit wurden mit GEM und MS-WINDOWS unternommen. Diese beiden Systeme entsprechen jedoch hinsichtlich ihrer Leistungsfähigkeit und ihrer Funktionalität noch nicht den an einen zukünftigen Standard zu stellenden Anforderungen.

Die Systeme der Kategorie (B) erfordern schon bei einfachen Anwendungen einen ausgesprochen hohen Programmieraufwand. Eine Ausnahme bildet in dieser Hinsicht GEM mit den "Application Environment Services" (AES), solange es sich um Anwendungen auf "Desktop"-Ebene handelt. Die spezifische Gestaltung von Fensterinhalten ist jedoch auch bei GEM vergleichbar aufwendig wie bei den anderen Systemen der Kategorie (B).

Ein zusätzliches Problem mit solchen "Desktop-Managern" wie AES ergibt sich daraus, daß sie einen großen Teil höherer Dialogfunktionalität abdecken, die besser einem gesonderten "Dialog-Manager" vorbehalten bliebe. Durch einen solchen Dialog-Manager könnten z.B. bestimmte Benutzereingaben wie etwa die Selektion eines Piktogramms kontextabhängig unterschiedlich interpretiert werden. Der Dialog-Manager sollte dazu über eine formale Aufgabenrepräsentation, eine Beschreibung der Eingabesyntax und - in adaptiven Systemen - über ein Benutzermodell verfügen. Eine Vermischung mit der Funktionalität des Fenstersystems ist von daher nicht sinnvoll.

In der nachfolgend aufgeführten Übersicht wird versucht, eine Standardfunktionalität für Fenstersysteme zu definieren. Wir unterscheiden dabei wiederum zwischen der Funktionalität für den Anwendungsentwickler und dem Endbenutzer. In jedem Feld ist eine Zahl für die Priorität (1: unverzichtbar, 2: Ausbaustufe) und ggf. "A" und/oder "B" notiert, je nachdem ob dieser Punkt durch die untersuchten Systeme der jeweiligen Kategorie hinreichend erfüllt wird.

	Für Anwendungsentwickler	Für Endbenutzer
Standard-Fenstertypen	1 A	
Definition Fenstertypen	1 A	
Generieren/löschen von Fenstern	1 A, B	
Öffnen/Schließen	1 A, B	1 A, B
Größe festlegen	1 A, B	1 A, B
Aktivieren (Selektieren)	1 A, B	1 A, B
Automatisches Redraw	1 A	
Fensterinhalt horizontal/vertikal verschieben	1 A	1 A
Integration von Zeigeopertionen	1 A, B	1 A, B
Überlappung von Fenstern	1 A	1 A
Nach-Vorn-Holen (Nach-Hinten-Bringen)	1 A	1 A
Integration von Piktogrammen	1	1
Integration von Menüs	2 A	2 A
Zoom (auf Bitmap-Ebene)	2	2
Wahlweise automatisches Bildschirmlayout	2	2
Weiterunterteilung von Fenstern	2 A	2 A

Bild 2: Anforderungen an die Funktionalität von Fenstersystemen

Literaturangaben:

/1/ Card, S. K.; Pavel, M.; Farrell, J. E. (1984): Window-Based Computer Dialogues. In: Proc. of Interact 84, Imperial College of Science and Technology, London. S. 1.355-1.359 (1984).

/2/ Smith, D.C.; et. al.: Designing the STAR user Interface. In: Degano, P.; Sandewall, E. (Eds.): Integrated Interactive Computing Systems. North-Holland, Amsterdam (1983).

/3/ Williams, G.: The LISA Computer System. In: BYTE 8, 1983 (2), S. 33-50.

/4/ Goldberg, A.; Robson, D.: Smalltalk-80, The Language and its Implementation. Addison-Wesley, Reading/MA (1983).

/5/ Fähnrich, K.-P.; Ziegler, J: Workstations Using Direct Manipulation as Interaction Mode-Aspects of Design, Application and Evaluation. In: Proc. of INTERACT 84, Imperial College of Science and Technology, London. S. 2.203-2.208 (1984).

/6/ Coutaz, J. : A Paradigm for User Interface Architecture. CMU-CS-84-124 (1984).

/7/ Hayes, P. J.; Szekely, P.A.; Lerner, R.A.: Design Alternatives for User Interface Management Systems Based on Experience with COUSIN. In: Proc. of CHI'85, San Francisco CA S. 169-175 (1985).

Anschriften der Verfasser:

Dr. Helmut Balzert

TRIUMPF-ADLER AG
Basisentwicklung

Nürnberger Straße 159
8510 Fürth

Dr. Heinz Ulrich Hoppe
Dipl.-Ing. Jürgen Ziegler

Fraunhofer-Institut
für Arbeitswirtschaft und
Organisation

Holzgartenstraße 17
7000 Stuttgart 1