Universität Ulm
Abteilung Verteilte Systeme

Entwurfsstrukturen für Application Sharing Systeme auf einer heterogenen Plattform

DISSERTATION
zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Informatik
der Universität Ulm

vorgelegt von
Klaus Heinrich Wolf
aus Freiburg
1999

Amtierender Dekan: Prof. Dr. Uwe Schöning
Gutachter: Prof. Dr. Peter Schulthess
Gutachter: PD Dr. Konrad Froitzheim
Gutachter: Prof Dr. Bernhard Plattner
Tag der Promotion: 11.2.2000

Inhaltsverzeichnis

1. Einleitung

Telearbeit und kooperative Telearbeit sind zentrale Komponenten der Bürowelt der Zukunft. Ein wichtiger Bestandteil von kooperativer Telearbeit ist das gemeinsame Bearbeiten von Dokumenten aller Art, insbesondere Texten und Bildern. Dabei sollen die gleichen Anwendungsprogramme eingesetzt werden, die auch bei der normalen Textverarbeitung oder Bildbearbeitung verwendet werden, denn der Gegenstand der gemeinsamen Arbeit entstammt fast immer der Arbeit mit Standardprogrammen für Text- oder Bildbearbeitung. In der vorliegenden Dissertation werden deshalb Systeme betrachtet, die es ermöglichen, Standardprogramme gemeinsam und über Netzwerke bei kooperativer Telearbeit zu verwenden. Diese Systeme werden gemeinhin Application Sharing Systeme genannt.

In der vorliegenden Dissertation werden Application Sharing Systeme systematisch erfaßt und Entwurfsmuster abgeleitet, die eine Entwicklung von Application Sharing Systemen leiten können. Dabei soll vor allem der heterogene Aspekt des Application Sharings im Vordergrund stehen. Das heißt, daß im Gegensatz zu den existierenden Application Sharing Systemen und Arbeiten zu Application Sharing Systemen hier verschiedene Anwendungsbereiche, Betriebssystemplattformen, Grafiksystemplattformen und Netzwerkbedingungen in die Betrachtungen einbezogen werden. In mehreren Beispielimplementierungen werden die vorgestellten Entwurfsstrukturen angewendet.

Die Resultate haben aber nicht nur Relevanz für die Implementierung von neuen Application Sharing Systemen, sie liefern auch Kriterien und Anregungen für den Entwurf von Grafiksystemen und Toolkits für Benutzerschnittstellen, die für kooperative Telearbeit besonders geeignet sind.

1.1. Kapitelübersicht

Kapitel 1 enthält drei einführende Abschnitte: Im ersten, direkt folgenden, (Kapitel 1.2) wird kurz der Application Sharing Ansatz für computerunterstützte kooperative Telearbeit motiviert. Daran schließt sich im nächsten Abschnitt eine Einführung in die Problemstellung der heterogenen Application Sharing Systeme an. Der Abschnitt soll die Beschäftigung mit der Thematik motivieren (Kapitel 1.3). Mit dem Abschnitt 1.4 der Einleitung schließt sich eine Zusammenfassung der wichtigsten Ergebnisse an. Diese Zusammenfassung soll als Richtschnur für den Hauptteil dienen. Sie verweist jeweils auf die entsprechenden Kapitel im Hauptteil in denen die Sachverhalte und Ergebnisse im Detail beschrieben sind.

Kapitel 2 (Kapitel *Grafiksysteme und Fenstersysteme*) diskutiert Eigenschaften von Grafiksystemen und vergleicht die wichtigsten Systeme. Es bildet die Grundlage für die Kapitel 4 und 5. Im Kapitel 3 (Kapitel *Eingabesysteme*) wird die Verarbeitung von Benutzereingaben diskutiert als Basis für die Kapitel 4 und 6.

Das vierte Kapitel *Komponenten von Application Sharing Systemen* bildet zusammen mit dem fünften Kapitel *Ausgabekonvertierung* den Kern der vorliegenden Arbeit. In Kapitel 4 wird die Funktion der Basiskomponenten von Application Sharing Systemen beschrieben. Dabei wird der heterogene Aspekt ausgeklammert. Die Ergebnisse gelten sowohl für homogene, als auch für heterogene Sharing Systeme. Die Übersetzung zwischen Grafiksystemen im heterogenen Fall wird in Kapitel 5 dargestellt. Kapitel 6 ergänzt den heterogenen Fall um die Übersetzung von Benutzereingaben.

Mit Kapitel 7 werden die Ergebnisse zusammengefaßt. Kapitel 7 verbindet die Basiskomponenten (aus Kapitel 4) mit der Übersetzungskomponente (aus Kapitel 5). Dabei gibt es Einblick in die Implementierungen, die im Rahmen der vorliegenden Arbeit durchgeführt wurden.

Kapitel 8 faßt die Ergebnisse der Arbeit zusammen.

1.2. Wozu Application Sharing ?

Fortschritte in Computer- und Netzwerktechnologie bilden die Grundlage für neue Formen von Kommunikations- und Kooperationsdiensten. Ein Großteil der neuen Anwendungen und Dienste konzentriert sich auf die Verbesserung bestehender Dienste. Telefonie und Fax werden dabei erweitert zu Desktop-Videokonferenz und Remote-Printing. Diese traditionellen Dienste werden unterstützt durch neue Dienste aus dem CSCW Bereich (Computer Supported Collaborative Work = Computerunterstützte Gemeinschaftsarbeit, Groupware), wie z.B. digitale Zeichenbretter und gemeinsame Dokumentenbearbeitung. Diese neuen Dienste werden gewöhnlich mittels spezieller, auf Gruppenkommunikation abgestimmte Programme realisiert.

Bislang war die Akzeptanz von CSCW Anwendungen sehr gering. Ein Grund dafür ist der Unterschied zwischen dem großen Funktionsumfang spezialisierter Anwendungsprogramme (Textverarbeitung, Tabellenkalkulation) und dem beschränkten Funktionsumfang von entsprechenden Programmen in CSCW-Systemen. CSCW-fähige Textbearbeitungsprogramme erreichen bei weitem nicht den heute in dieser Programmklasse üblichen Funktionsumfang, da sie oft von Grund auf neu im Rahmen von CSCW Forschungsprojekten entwickelt wurden. Arbeiten über solche sogenannten Multiuser-Editoren gab es schon sehr früh in der CSCW Forschung. Beispiele sind CES (Collaborative Editing System) [Greif88], Shared Books [LeHo98], Quilt [LeFiKr86] und GROVE [ElGiRe90].

Potentielle Anwender von CSCW-Systemen sind verwöhnt durch Qualität und Quantität ihrer Anwendungsprogramme und sind nicht bereit statt dessen mit schlichteren CSCW-fähigen Ersatzprogrammen zu arbeiten. Hinzu kommt, daß die meisten Anwender mit ihrer Arbeitsumgebung vertraut sind. Sie wollen und können normalerweise weder das Anwendungsprogramm, noch den Rechnertyp wechseln nur, um mit Hilfe des Computers in der Gruppe zu arbeiten. Die beträchtlichen Investitionen in Arbeitsplatzrechner, Software und Ausbildung haben bis jetzt einen weitverbreiteten Einsatz von Telekooperation verhindert.

Die Entwickler von CSCW-fähigen Softwareprodukten sind nicht imstande, mit dem exponentiellen Wachstum der Möglichkeiten spezialisierter Anwendungsprogramme - noch dazu auf mehreren Plattformen - Schritt zu halten. Es ist deshalb auch bei Telekooperation notwendig, die weitentwickelten Standardsoftwarepakete, wie Microsoft Word und Excel, Framemaker, Adobe Photoshop oder CASE Werkzeuge zu verwenden. Darüber hinaus gibt es Individualsoftware, deren Funktionalität auch nicht annähernd durch CSCW-fähige Programme nachgebildet werden kann und für die es deshalb keinen Ersatz gibt. Die Bildschirmdarstellung des Anwendungsprogramms soll bei allen Telekooperationspartnern dargestellt werden. Ein möglicher Ansatz dazu ist die gleichzeitige Darstellung des Inhalts des Bildschirms, auf dem das Programm läuft, zu allen Teilnehmern.

Software zum Bildschirmsharing ist als Produkt erhältlich (Timbuktu/Farallon, Carbon Copy/Compaq, PlanetX/Ascend früher InterCon, usw.). Diese Produkte verteilen grundsätzlich den gesamten Bildschirm zu einem oder mehreren Endgeräten.

Häufig ist es jedoch nicht notwendig, den gesamten Bildschirm eines Arbeitsplatzrechners gemeinsam zu betrachten, um gemeinsam ein Dokument zu bearbeiten. Die Arbeitsumgebung auf dem Bildschirm (Desktop) kann sogar als sehr persönlicher Bereich betrachtet werden. Auf dem Bildschirm sind oft private oder sicherheitsrelevante Daten, wie Email, Kalender oder Dateiverzeichnisse mit Dateinamen dargestellt. Gemeinsam zu bearbeitende Information und Dokumente beschränken sich dagegen gewöhnlich auf eines oder mehrere Fenster, die eine Tabellenkalkulation, einen Text oder einen Film enthalten. Ein Sharing System, das immer den gesamten Bildschirms erfaßt ist deshalb oft nicht befriedigend. Fensterbasiertes oder programmbasiertes Sharing trifft die Bedürfnisse des Anwenders im Rahmen von Telekooperation besser. In diesem Fall werden nur einzelne Fenster oder alle Fenster eines Programms für alle Teilnehmer sichtbar.

> Bildschirmbasiertes Sharing fördert die unbeabsichtigte Bekanntgabe geheimzuhaltender Daten oder erschwert umgekehrt das Sharing nicht geheimer Daten. Sicherheit bei gleichzeitiger Flexibilität kann nur bei selektivem (fensterbasiertem) Sharing gewährleistet werden.

Die Architektur üblicher Arbeitsplatzrechner und Anwendungsprogramme ist ausgerichtet auf die Bedienung durch nur eine Person. Sogar Mehrbenutzerbetriebssysteme, wie UNIX, nehmen eine eins-zu-eins Beziehung zwischen dem Anwendungsprogramm und Eingabegeräten, bzw. Ausgabegeräten an:

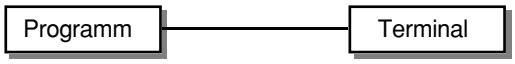


Abbildung Normalbetrieb - 1 Programm, 1 Terminal

Der Kern des Application Sharing Gedankens besteht darin, die eins-zu-eins Beziehung zwischen Anwendungsprogramm und Display in eine 1-zu-n Beziehung umzusetzen. Dazu wird die direkte Verbindung zwischen Programm und Display, die aus einer Netzwerkverbindung oder einer Kette von statischen oder dynamischen Bindungen bestehen kann, gelöst und das Application Sharing System zwischen Anwendungsprogramm und Endgerät eingefügt:

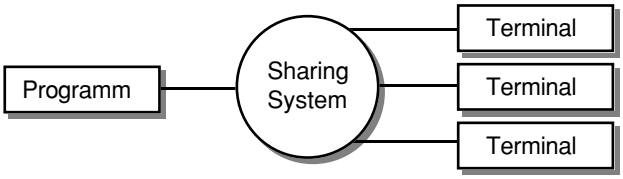


Abbildung Sharing - 1 Programm, n Terminals

1.3. Grund für diese Arbeit

Diese Arbeit baut auf bestehenden Arbeiten auf. Es gibt sowohl theoretische Arbeiten, als auch Implementierungen, Implementierungsberichte und Produkte im Bereich Application Sharing. Die Geschichte des Application Sharings auf der Basis von Programmen und Fenstern beginnt mit X-Multiplexern, die Ausgaben eines X-Window Klienten auf mehreren X-Servern darstellten [Altenhofen90] [CrTo90] [McFarlane91]. Schon 1991 war das Programm XTV [AbWa91] in der Programmsammlung zum X11 Window System (die sogenannte X11 contrib package) enthalten.

Auch das Screen Sharing Programm Timbuktu von Farallon hat Eigenschaften von Application Sharing Systemen. Das Programm repliziert zwar den gesamten Bildschirm eines Apple Macintosh (in der ersten Version) auf einem entfernten Macintosh. Es arbeitet aber auf der Basis von Grafikprimitiven und nicht, wie andere Screen Sharing Systeme, auf der Basis von Pixeln und Bildschirmabzügen. Timbuktu läßt Rechner mit Microsoft Windows und Apple Macintosh Quickdraw Grafiksystemen miteinander kommunizieren. Der Bildschirminhalt eines Windows Rechners kann auf einem Macintosh in einem Fenster dargestellt werden und umgekehrt. Damit ist es eines der wenigen Sharing Systeme, die heterogen, d.h. plattformübergreifend arbeiten.

Andere Screen Sharing Systeme, wie PlanetX, ermöglichen zwar Kommunikation zwischen verschiedenen Grafiksystemplattformen (Macintosh Quickdraw und X11), arbeiten aber nur auf der Basis von Pixeldarstellungen und sind deshalb bandbreitenintensiver als notwendig. Der Grund ist natürlich die einfachere Übersetzung zwischen Grafiksystemplattformen auf Pixelbasis. Aber auch bei diesem Schritt sind Optimierungen möglich auf die in dieser Arbeit eingegangen werden soll.

1.3.1. Grafikschnittstellen

Sharing Systeme, ob bildschirmbasiert, programmbasiert oder fensterbasiert, sind auf die Grafikausgaben von Programmen angewiesen. Diese Ausgaben erfolgen über Programmierschnittstellen und verschiedene andere Komponenten von Grafiksystemen. Eine Analyse von Grafikschnittstellen erfolgt hier, weil zum Zweck des Application Sharings an Grafikschnittstellen die Bildschirmausgaben von Programmen abgefangen (Interception) und auf andere Bildschirme umgeleitet werden können (Redirection).

Bei allen Grafiksystemplattformen gibt es jeweils mehrere Schnittstellen in denen Grafikausgaben abgegriffen werden können. Verschiedene Sharing Systeme verwenden dazu eigene Methoden. Sie setzen jeweils an einer anderen Schnittstelle an. Zum Teil ist die dabei verwendete Schnittstelle nicht optimal gewählt. Die Gründe sind vielgestaltig. Eine systematische Analyse der verfügbaren Schnittstellen auf den relevanten Plattformen, ihrer Eigenschaften und Zugriffsmöglichkeiten fehlt bisher in der Literatur. Darüber hinaus führen Defizite existierender Schnittstellen im Bezug auf Application Sharing zu

Erkenntnissen, die beim Entwurf zukünftiger Schnittstellen von Grafiksystemen nützlich sind, um diese Application Sharing-freundlich zu gestalten.

1.3.2. Grafikübersetzung

Die Existenz und der Gebrauch verschiedener Grafiksysteme, sowie die Entstehung neuer Grafiksysteme (aktuelles Beispiel: Java2d) machen Application Sharing über Grafiksystemgrenzen erforderlich. Tatsächlich gibt es aber fast keine Sharing Systeme, die diese Grenzen überwinden können. Ein Grund dafür ist die Tatsache, daß die Übersetzung zwischen Grafiksystemen allgemein als sehr aufwendig angesehen wird. Tatsächlich ist der Aufwand in Mannjahren zu rechnen [Vodslon92]. Prinzipiell gibt es bei n Grafiksystemen, n(n - 1) Übersetzer. Hinzu kommen noch einige Kodierungen für die Netzwerkübertragung, so daß insgesamt eine relativ große Zahl von aufwendigen Softwarekomponenten entwickelt werden muß, um die wichtigsten Grafiksysteme abzudecken.

Es ist zu erwarten, daß in naher Zukunft mehrere Implementierungen, bzw. Produkte vorliegen, die jeweils Teile des Problems abdecken, d.h. jeweils mehrere Übersetzer zwischen Grafiksystemen enthalten. Bei allen Übersetzern, sowohl konkurrierenden Implementierungen, als auch zwischen verschiedenen Grafiksystemen, müssen ähnliche oder sogar gleiche Probleme gelöst werden. Es ist deshalb sinnvoll, diese Probleme systematisch zu erfassen, losgelöst von Grafiksystem oder Betriebssystemplattformen darzustellen und Lösungen zu erarbeiten, die bei einer ganzen Reihe von Implementierungen eingesetzt werden können.

1.3.3. Grafiktransport

Existierende Application Sharing Systeme sind fast ausnahmslos auf kleine Gruppen von Arbeitsplatzrechnern zugeschnitten, die breitbandig vernetzt sind. Alle Implementierungen gehen von guten bis sehr guten Netzwerkbedingungen aus, d.h. einer geringen Varianz des Durchsatzes auf hohem Niveau. Es hat sich aber gezeigt, daß diese Bedingungen außerhalb der Laborumgebung nicht immer gegeben sind. Application Sharing Systeme müssen deshalb Vorkehrungen treffen, um trotz Netzwerkfehlern, bzw. Durchsatzeinbrüchen den Betrieb aufrechtzuerhalten. Entsprechende Verfahren werden im Hinblick auf die Anwendung für Application Sharing diskutiert. Gleichzeitig können auch die übertragenen Grafikdaten so gestaltet werden, daß der Datenstrom selbst robust gegen Netzwerkfehler ist.

Die Netzwerkkomponente eines Application Sharing Systems transportiert Grafikdaten zu eventuell mehreren Endgeräten. Dort werden die Daten ausgegeben. Der Transport kann auf sehr verschiedene Art realisiert werden. Das Anwendungsszenarium bestimmt deshalb das verwendete Netzwerkprotokoll. Das Szenarium bestimmt auch, ob Software beim Empfänger installiert werden kann oder muß. Ist dies der Fall, dann kann die Netzwerkkomponente verteilt ausgelegt werden und so zusätzliche Dienste, wie Verschlüsselung und auf Grafikströme zugeschnittene Kompression bieten. Es wird besonders auf die Kompression von nicht pixelbasierten Grafikdaten eingegangen.

Die Kommunikation zwischen Programmen und Endgeräten (Displays) ist bidirektional. Programme erhalten neben Eingebeereignissen auch andere Rückmeldungen von Endgeräten. Diese Rückmeldungen sind im wesentlichen Antworten auf Ressourceallokationen. So meldet ein Endgerät zurück, ob ein Farbtabelleneintrag reserviert werden konnte oder ob ein Zeichensatz verfügbar ist. Hängen über ein Application Sharing System mehrere Endgeräte an der Ausgabe eines Programms, dann muß das Sharing System aus mehreren möglichen Antworten eine einzige Antwort erzeugen, die dem Programm zugeführt wird und bestmöglich für die Funktion des Gesamtsystems ist. Dazu werden mehrere Varianten und ihre Anwendungsfälle vorgestellt.

1.3.4. Eingabeumschaltung

Die Teilnehme mehrerer Benutzer an Grafikausgaben bedeutet in den meisten Fällen auch, daß mehrere Ströme von Benutzereingaben dem einen gemeinsamen Programm zugeführt werden müssen. Da fast alle Anwendungsprogramme nur einen Eingabestrom erwarten, d.h. nur einen Zustandsautomaten für eine Eingabefolge implementieren, muß ein Application Sharing System zwischen Eingabeströmen umschalten. Solche Umschaltungsstrategien wurden schon in großem Umfang untersucht. Deshalb soll hier nicht auf diese sogenannten Floor Control Policies (z.B. [GuWe93]) eingegangen werden.

Ein Bereich der bisher vernachlässigt wurde ist die Sicherung der Integrität des Eingabestroms, der einem Programm zugeleitet wird. Bisher wurde entweder die Umschaltung so vorgenommen, daß keine Inkonsistenzen auftreten können, d.h. es wurden nur eine grobe Granularität der Eingabeumschaltung zugelassen. Oder: es wurden Inkonsistenzen toleriert und deren Auswirkungen, die häufig zum Programmabsturz führten, durch ad-hoc Maßnahmen des Sharing Systems oder fehlertolerante Anwendungsprogramme korrigiert. Die Vermeidung von Programm- oder Systemabstürzen wurde bisher als Fehlerbehebung bei Bedarf angesehen und nicht systematisch betrieben. Tatsächlich ist es aber möglich durch Modellierung von Benutzereingaben und kontrollierte Vervollständigung des Eingabestroms dessen Integrität auch bei feingranularer Eingabeumschaltung sicherzustellen.

1.3.5. Abgrenzung zu Grafikterminals

Der Begriff Application Sharing ist deutlich abgegrenzt vom Begriff des sogenannten Thin Clients. Thin Clients können als die dritte Phase der Grafikterminals betrachtet werden. Die Entwicklung der Grafikterminals begann mit Tektronix und DEC Terminals, die in einen Grafikmodus umgeschaltet werden konnten, z.B. Tektronix 4014 und DEC 340. Die zweite Phase kam mit dem X11 Window System. In beiden Fällen wird das Programm, das die Grafikausgaben erzeugt auf einem Applikationsserver ausgeführt. Auf dem Applikationsserver laufen im allgemeinen mehrere Benutzerkontexte (Shells) mit jeweils mehreren Programmen. Jeder Benutzer hat einen eigenen Kontext. Kommunikation zwischen Benutzerkontexten findet normalerweise nicht statt. Besteht die Möglichkeit zur Kommunikation zwischen Benutzerkontexten oder sogar zur Kommunikation im Sinne von Application Sharing, dann arbeitet das verwendete Sharing System nach den gleichen Funktionsprinzipien, ob die beteiligten Benutzer am gleichen Applikationsserver angemeldet sind oder nicht.

> Application Sharing ist keine Eigenschaft des Applikationsservers oder des Terminals, sondern ein Zusatz, der zwischen Applikation und Terminal eingefügt wird.

Genauso verhält es sich bei der dritten Phase, den Thin Clients. Der Begriff Thin Client wird zur Zeit vor allem im Umfeld der Microsoft Windows (NT) Terminal Server verwendet. Er bezeichnet aber nichts anderes, als eine Methode auf einem Applikationsserver mehrere Benutzerkontexte parallel laufen zu lassen und die Grafikausgaben auf Windows-Grafikterminals darzustellen. Die Übertragung der Grafikdaten über das Netzwerk läßt eine Ähnlichkeit zum Application Sharing vermuten.

Tatsächlich ist Beziehung zwischen Windows-basiertem Application Sharing und Applikationsservern mit Windows Terminals die gleiche, wie zwischen X-Multiplexer und X-Terminals. Die Gemeinsamkeiten zwischen einem Sharing System und dem entsprechenden Terminalsystem beschränken sich auf das verwendete Protokoll. Dabei haben MS Windows basierte Systeme den Nachteil, daß die Anwendungsprogramme nicht für den Betrieb über Netzwerke entworfen sind und zum Teil viel zu hohe Datenraten erzeugen (siehe Kapitel 4.3.4.3).

Protokoll	Terminalsystem	Beispiel für Terminalsystem	Beispiel für Application Sharing System
X11	X-Terminal	jedes System mit Xlib	X-Multiplexer
Win32	Windows Terminal 'Thin Client'	Windows NT z.B.: Newmoon: Liftoff Citrix: Winframe [Citrix]	Microsoft Netmeeting

Tabelle Protokolle, Terminals und Sharing Systeme

1.3.6. Historie und Stand der Technik

1.3.6.1. Phase der Grundlagenforschung

Die Arbeit an Application Sharing Systemen reicht etwa 10 Jahre zurück. Die ersten Sharing Systeme bauten auf dem X11 Window System auf, da dieses aufgrund seiner Netzwerkfähigkeit für Application Sharing über Netzwerke prädestiniert ist. Aus dieser Zeit bis 1993 stammen viele X-Multiplexer, d.h.

homogene Application Sharing Systeme für das X-Protokoll. Allerdings waren alle diese Systeme, die bis 1993 entwickelt worden, sehr instabil [BaPl93]. Eine Übersicht ist in [Minenko96] enthalten. Aus der gleichen Zeit stammt Timbuktu von Farallon (jetzt Netopia). Timbuktu ermöglichte homogenes Screen Sharing zwischen Apple Macintosh Computern, wird aber deshalb hier aufgeführt, da es schon sehr früh (1987) als kommerzielles Produkt auf dem Markt war. Carbon Copy war das entsprechende Produkt für Microsoft Windows .

1.3.6.2. Homogene Sharing Systeme

Intensive Forschung führte in der zweiten Phase bis 1996 zu stabilen Produkten für das X Window System, wie Sun's ShowMe [Sun96], und Siemens GroupX [Minenko96]. Gleichzeitig entstanden erste Übersetzer in das X Protokoll für Win16 (WinX von CET, WinX von der Universität Ulm [Schöttner96], Siemens GroupWin) und Apple Quickdraw (MaX [WoFrSchu95]). Timbuktu erhielt einen Übersetzer zwischen Quickdraw und Win16, der Screen Sharing zwischen Apple Macintosh und Microsoft Windows ermöglichte. Intel ProShare integrierte ein homogenes Win16 Application Sharing System und eine Videokonferenzsystem.

1.3.6.3. Aktuelle Produkte

Mit der dritten Phase der Application Sharing Systeme bis 1999 verlagert sich der Schwerpunkt weg von X11 hin zu Windows basierten Systemen. Das dominierende Produkt im Windows für Windows ist Microsoft NetMeeting für homogenes Win16 Sharing. Netmeeting, obwohl für die Win32 Systeme Windows 95 und Windows NT 4 entwickelt und vermarktet, verwendet eine direkte Umsetzung der Win16 Grafikprogrammierschnittstelle als Grafikprotokoll. Microsoft und PictureTel erreichten 1998 die Standardisierung des von Netmeeting verwendeten Protokolls durch die ITU als T.128 (auch T.SHARE) im Rahmen der T.120 Telekonferenzstandards. Während sich in der zweiten Phase alle plattformübergreifenden Entwicklungen am X11 Protokoll orientierten, bewegt sich der Markt seit der Veröffentlichung der T.128 Spezifikation auf ein heterogenes System mit Win16 als gemeinsamem Grafikprotokoll zu. Dazu gehören die aktuellen T.120 konformen Produkte, Timbuktu Pro 4.8 für MacOS 8.5, SunForum 2.0 für Solaris 2.5, Data Connection's DC-Share für mehrere Unix Varianten, SGImeeting 1.0, PictureTel LiveShare Plus und Netmeeting 2.1 für Win32 [Timbuktu98] [SunForum99] [SGImeeting98] [LiveShare98] [DCShare97] [Netmeeting96]. Auch DC-WebShare von Data Connection Ltd. und Net.120 von DataBeam verwenden T.128 [WebShare98] [Net120-97]. Beide setzen ein Java Applet im WWW-Klienten als Endgerät ein.

Die wachsende Zahl kompatibler Application Sharing Produkte ist ein erfreuliches Resultat der Standardisierung durch die ITU. Tatsächlich stellt aber T.128 für moderne Grafiksysteme mit Pfadfunktionen, wie Win32, Postscript und Java2d eine starke Einschränkung dar (siehe Anhang *Bewertung von T.128*).

Terminal Server, wie Citrix Winframe [Citrix], Microsoft Windows Terminal Server (WTS), NCD ThinSTAR [NCD] und Neoware NeoStation sind keine Application Sharing Systeme. Obwohl sie mit Independent Computing Architecture [ICA] auch auf Win16 basieren, entsprechend Sie nicht T.128.

1.3.6.4. Aktuelle Forschung

Neben dem Win16 basierten Application Sharing, das inzwischen das Produktstadium erreicht hat, entwickelt sich mit dem auf Java Schnittstellen basierenden Application Sharing ein neues Forschungsfeld. Maßgeblich ist JCE (Java Collaborative Environment) [AbKvNa96] [AbKiKFa99]. Es sind allerdings weitere Entwicklungen notwendig, da Anwendungsprogramme und Applets noch durch Eingriffe in den Quellcode sharingfähig gemacht werden müssen [KiKFaAb97]. JCE setzt auf Java AWT [StrMi96] auf. Ein Sharing System für Java2d oder für das Swing Toolkit ist noch nicht bekannt.

Application Sharing für sehr große Gruppen durch Multicast wird noch wenig bearbeitet [Maier97]. Mit den Ergebnissen der aktuellen intensiven Forschung bei Protokollen und Mechanismen für reliable Multicast wird aber Application Sharing mit reliable Multicast interessant. Als Referenz sei hier [Jacobson97] genannt. Sehr aktiv ist die Reliable Multicast Research Group [RMRG97]. Eine Standardisierung durch die IETF ist bald zu erwarten [RFC2357-98]. Application Sharing basierend auf reliable Multicast ist noch weitgehend unerforscht. Fast alle Arbeiten in diesem Bereich beschäftigen sich mit Spezialprogrammen, wie Shared Whiteboards oder Schulungssoftware für große Teilnehmerzahlen [GeWe98].

Wissenschaftliche Arbeiten zum Application Sharing befassen sich fast ausschließlich mit X Window basierten Sharing, wobei Arbeiten während der ersten Phase vor allem auf Detailprobleme eingingen, z.B. [AbWa92] [Chung91] und Arbeiten während der zweiten Phase X-Sharing Systeme und deren Optimierung beschrieben [Gutekunst95] [Danskin95] [Minenko96]. Dabei fällt auf, daß die Perspektive dieser Arbeiten oft stark durch die besonderen Bedingungen des X Window Systems geprägt ist. Arbeiten in den oben genannten Gebieten des plattformübergreifenden Application Sharings gibt es sehr wenige, wie. z.B. [DaZh96]. An diesem Punkt setzt die vorliegende Arbeit an.

1.4. Übersicht über die Ergebnisse

Die Systematik von Application Sharing Systemen läßt sich in vier Teilbereiche untergliedern:

1. Ausgabeumleitung
   Ein Application Sharing System muß die Grafikausgaben, die von Programmen nur an ein Display gesendet werden, zu mehreren entfernten Endgeräten umleiten. Diese Art der Ausgabeumleitung ist bei den meisten Grafik-/Betriebssystemen nicht vorgesehen.
2. Kombination mehrerer Eingabeströme
   Die Betriebssysteme von Arbeitsplatzrechnern sich dafür ausgelegt die Ausgaben von mehreren Programmen auf jeweils einem Ausgabegerät zusammenzuführen und Eingaben eines Benutzers an mehrere gleichzeitig laufende Programme zu verteilen. Beim Application Sharing liegt genau der umgekehrte Fall vor. Ein Application Sharing System muß deshalb die fehlende Systemunterstützung für mehrere Eingabeströme nachbilden.
3. Übersetzung von Grafikströmen
   Stimmt das Grafikformat, das vom Anwendungsprogramm verwendet wird, nicht mit dem Format überein, das vom Grafikendgerät verstanden wird, dann muß das Application Sharing System den Grafikstrom übersetzen. Eventuell kann Grafikübersetzung zwischen Quelle und Senke auch mehrmals stattfinden.
4. Effizienter Transport von Grafikströmen und Ausgabeverteilung
   Application Sharing benötigt fast immer eine relativ große Bandbreite. Der Transport von Grafikströmen über Netzwerke muß deshalb so effizient, wie möglich organisiert werden. Das Anwendungsszenarium bestimmt dabei die eingesetzten Techniken. Diese wiederum haben Rückwirkungen auf das Grafikstromformat.

Die Teilbereiche 1 und 2 befassen sich mit der Wechselwirkung von Benutzer, Anwendungsprogramm, Grafikendgerät und Betriebssystem, den Grafikstromquellen und -senken. Die Teilbereiche 3 und 4 beschränken sich dagegen auf die Verarbeitung von Grafikausgabeströmen unabhängig von Stromquellen und -senken.

Im folgenden wird eine Übersicht über die Ergebnisse der Arbeit gegeben. Die Übersicht verweist jeweils auf die detaillierte Beschreibung im Hauptteil.

1.4.1. Granularität von Grafikströmen

Computergrafik, die für längere Zeit aufbewahrt werden soll, wird in Grafikdateien gespeichert. Es gibt sehr viele verschiedene Formate für Grafikdateien. Sie reichen von Pixeldarstellungen, wie BMP-Dateien über sogenannte Metadateien (Postscript, CGM, PICT, WMF, siehe auch [MuVa94]) bis zu Dateiformaten, die auf bestimmte Anwendungen zugeschnitten sind (z.B. CAD Programme), bzw. mit bestimmten Anwendungsprogrammen eingeführt worden sind und nur von diesen verwendet werden.

Allen Dateiformaten gemeinsam ist die Tatsache, daß sie Grafikanweisungen enthalten. Die Anweisungen werden von Programmen interpretiert, um eine Gesamtgrafik aufzubauen, die letztlich einem Anwender präsentiert wird.

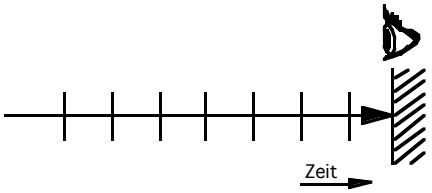


Abbildung Darstellung eines statischen Dokuments

Der Inhalt eines Dokuments wird dekodiert und am Ende dargestellt. Nur der Endzustand ist für das Auge des Betrachters bestimmt. In manchen Fällen wird der Dekodierungsprozeß angezeigt, so daß Zwischenzustände sichtbar werden. Diese sind aber nur zur Anzeige des Fortschritts gedacht. Sie zeigen keine Dokumentenzustände an. Es gibt nur einen Präsentationszeitpunkt.

Bei Video wird eine kontinuierliche Folge von Bildern dargestellt. Die Einzelbilder bestehen jeweils aus einer oder sogar mehreren Grafikanweisungen. Jedes Einzelbild wird dekodiert und dargestellt. Es gibt also - bei fester Bildrate - eine regelmäßige Folge von Präsentationszeitpunkten. Der Inhalt eines Videos wird nicht durch den Endzustand, sondern durch die Zwischenzustände in regelmäßigen Abständen repräsentiert. Typische Videoformate sind Quicktime [Quicktime93], MPEG [MPEG1-92] und AVI [Florence94]. MPEG wird auch in andere (z.B. Quicktime) eingebettet.

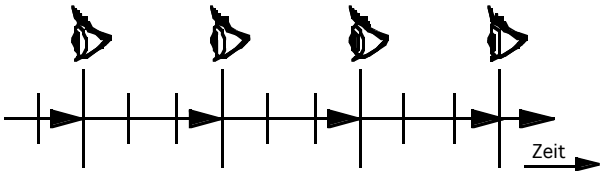


Abbildung Darstellung eines dynamischen Dokuments (z.B. eines Film)

Bilder eines Films werden fortlaufend dekodiert und regelmäßig angezeigt. Es gibt mehrere Präsentationszeitpunkte in gleichen Abständen. Um Störungen zu vermeiden wird bei der Dekodierung einzelner Bilder im allgemeinen nicht der Fortschritt angezeigt, sondern nur das Ergebnis.

Tatsächlich liegen die Präsentationszeitpunkte bei Video nicht notwendigerweise in gleichen Abständen. Die Zeit zwischen Präsentationszeitpunkten wird in der Praxis wesentlich vom Zugriff auf die Daten bestimmt, d.h. Dekodierung und Präsentation werden oft dadurch verzögert, daß Videodaten nicht schnell oder kontinuierlich genug vorliegen. Dieses Verhalten ist sowohl beim Abspielen von Festplatte, in erhöhtem Maß aber beim Abspielen von Compact Disk und besonders über Netzwerke zu beobachten. Vor allem bei der Dekodierung eines Videostroms, der über ein Netzwerk geladen und nicht komplett beim Empfänger zwischengespeichert wird, werden die Präsentationszeitpunkte im wesentlichen von der Geschwindigkeit der Netzwerkübertragung bestimmt. Andererseits können durch eine derartig verzögerte Darstellung auch mehrkomponentige Bilder (z.B. GIF [MuVa94]) wie Filme wirken [WoFrWe96] [WoFr97].

Auch Anwendungsprogramme erzeugen eine Folge von Grafikanweisungen, die nacheinander dekodiert und dargestellt werden. Bei diesen Kommandofolgen fehlt aber das Zeitraster. Grafikanweisungen werden schubweise erzeugt und dargestellt sobald sie vorliegen. In der Software kann man entsprechende Strukturen finden, denn ereignisgesteuerte Programme erzeugen Grafikanweisungen immer als Antwort des Programms auf Benutzereigaben.

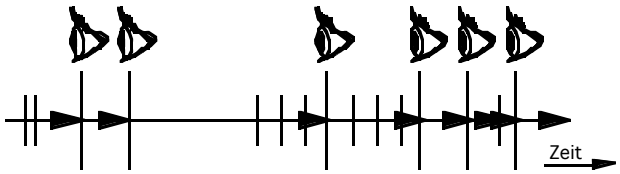


Abbildung Darstellung des Bildschirmausgabestroms von Anwendungsprogrammen

Dekodierung und Darstellung finden kontinuierlich, aber mit Pausen statt. Zeiten in denen das darstellende Grafiksystem ausgelastet ist wechseln sich mit Ruhezeiten ab. Präsentationszeitpunkte folgen in unregelmäßigen Abständen.

1.4.2. Grafikströme

Der Übergang von Video zum Grafikstrom findet dort statt, wo die Präsentationsgeschwindigkeit von der Datenverfügbarkeit statt vom Stromformat bestimmt wird. Während beim Video der Dekodierer die (video)formateigene Präsentationsrate einstellt, bestimmt beim Grafikstrom die Quelle, bzw. das Netzwerk die Darstellungsgeschwindigkeit. Die für die Dekodierung benötigte Zeit wird im Rahmen von Application Sharing vernachlässigt. Grafikanweisungen in Grafikströmen von Anwendungsprogrammen sollen immer so schnell, wie möglich dekodiert und angezeigt werden. Dies gilt auch im Rahmen von Application Sharing.

Die Unterschiede zwischen dem Grafikdateien, Videoformaten und Grafikanweisungen von Anwendungsprogrammen sind nur graduell. Alle bestehen aus Sequenzen von Grafikkommandos. Für jede Grafikbeschreibungssprache kann man Grafikstromformate definieren. Bei manchen Grafiksystemen sind Stromformate durch Standards gegeben. Dies ist vor allem bei videoorientierten und speicherorientierten Grafiksystemen der Fall (MPEG, Postscript) der Fall; aber natürlich auch bei netzwerkfähigen Grafiksystemen (X Window System). Für displayorientierte Grafiksysteme, wie Quickdraw und MS Windows können entsprechende Stromformate definiert werden (siehe Kapitel 5.3.1). Dies ist z.B. im Fall der MS Windows Grafiktoolbox in Gestalt des Win16 GDIs durch T.128 der ITU geschehen [T.128-98].

> Sind Sequenzen von Grafikkommandos inkrementell dekodierbar, d.h. existieren innerhalb der Sequenz keine Referenzen auf nachfolgende Daten, dann können sowohl die Übertragung und Anzeige einer Grafikdatei, eines Videos, als auch die Grafikanweisungen von Anwendungsprogrammen als Grafikströme angesehen werden.

Grafikströme können sich in zwei Dimensionen unterscheiden:

1. in der Präsentationsgranularität, wie oben diskutiert und
2. in der Art der Grafikanweisungen.

Grafikanweisungen in den verschiedenen Grafikstromformaten werden in Kapitel 2.4 einander gegenübergestellt. Die Funktionalität der Grafikanweisungen in den verschiedenen Stromformaten reicht von Pixmap-Sequenzen und komprimierten Pixmaps (Video) über Folgen von einfachen Grafikprimitiven, wie Linien und Rechtecken, bis zu Pfaddefinitionen bei Postscript und verwandten Formaten. Der Unterschied ist auch in diesem Fall nur graduell, nicht prinzipieller Natur (siehe Kapitel 2.4.1.2). Aus diesem Grund können für die Übertragung (und Speicherung) von Application Sharing Sitzungen alle hier besprochenen Formate einschließlich der Videoformate, wie MPEG, verwendet werden.

1.4.3. Ausgabeumleitung

Die Umleitung der Grafikausgaben von Programmen bildet den ersten Arbeitsbereich bei der Beschäftigung mit Application Sharing Systemen auf verschiedenen Plattformen. Für die relevanten Systeme wurden mögliche Schnittstellen untersucht, Semantik und Art des erhaltenen Grafikstroms, sowie die Zugriffsmöglichkeit auf die Schnittstelle bewertet. Zusätzlich werden Spezialfälle diskutiert, die bei anderen zukünftigen Implementierungen hilfreich sein können.

In Kapitel 4.2 werden folgende Schnittstellen diskutiert:

1. Programmierschnittstelle (API = Application Programmers Interface),
2. Gerätetreiberschnittstelle,
3. Netzwerkschnittstelle, falls vorhanden und
4. Hardwareschnittstelle.

Dabei werden jeweils Eigenschaften der Schnittstellen, bzw. der an den Schnittstellen erhaltenen Grafikströme diskutiert bezüglich

1. der Semantik der Grafikdaten an der Schnittstelle,
2. Zugriffsverfahren, und
3. Spezialfälle.

Experimente wurden durchgeführt mit

- X11 / Protokollschnittstelle,
- X11 / Programmierschnittstelle,
- Quickdraw / Treiberschnittstelle,
- Windows 32 / Protokollschnittstelle,
- Windows 32 / Treiberschnittstelle.

Der Zugriff auf eine Schnittstelle und die Möglichkeit, die über diese Schnittstelle fließenden Daten, vollständig zu beobachten, kann je nach System und Schnittstelle sehr unterschiedlich sein. Zum Teil ist die Komplexität des Zugriffs eng gekoppelt mit der Art der Schnittstelle. Eine Netzwerkschnittstelle ist z.B. leicht zu beobachten. Es wirken sich aber auch betriebssystemspezifische Faktoren aus, so daß die gleiche Schnittstelle auf verschiedenen Betriebssystemen andere Eigenschaften haben kann. Ein Faktor, der bisher immer vernachlässigt wurde, ist die Bedienbarkeit des Gesamtsystems (Betriebssystem, Programm, Sharing System). Anwender sollen durch die zum Application Sharing notwendige Ausgabeumleitung keinen zusätzlichen Zwängen unterworfen werden oder Geschwindigkeitseinbußen hinnehmen müssen. Aus diesem Grund wurden hier zur Bewertung des Zugriffs auch die Faktoren Geschwindigkeit und einfache Zuschaltbarkeit der Ausgabeumleitung hinzugenommen.

Die Untersuchungen ergaben, daß die Schnittstellen von Grafiksystemen bezüglich ihrer Eignung für die Ausgabeumleitung bei Application Sharing Systemen zweidimensional angeordnet werden können. Die zweidimensionale Anordnung trägt sowohl der Semantik der Grafikströme, als auch der Zugriffsmöglichkeit auf die Schnittstelle Rechnung. Abbildung *Anordnung der Schnittstellen bezüglich Semantik und Zugriff* faßt die Ergebnisse zusammen.

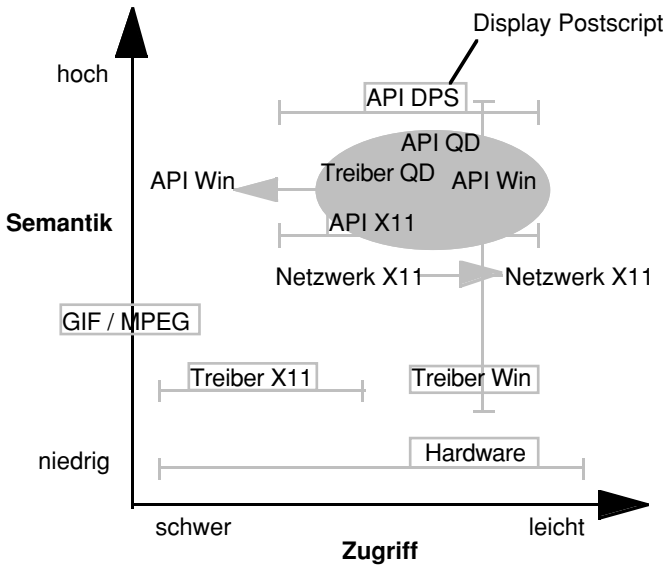


Abbildung Anordnung der Schnittstellen bezüglich Semantik und Zugriff

Die Skala für den Zugriff auf die Schnittstellen soll eine ungefähre Aufwandsabschätzung wiedergeben. Die Einordnung wurde aus den Experimenten abgeleitet. Gründe für die vorgenommene Einordnung sind in Kapitel 4.2 nachzulesen.

Die Programmierschnittstellen der drei wichtigsten Systeme X11, Win32 und Quickdraw sind in Semantik und Zugriff vergleichbar. Dies wird in der Abbildung durch eine umschließende Ellipse verdeutlicht.

1.4.3.1. Windows32 API

Die Programmierschnittstelle von Windows ist prinzipiell leicht zugreifbar durch Installation einer dynamisch gelinkten Bibliothek (Windows DLL). Dies gilt sowohl für Windows 95 /98, als auch für Windows NT. Tatsächlich ist das Win32 API aber für die Ausgabeumleitung alleine nicht geeignet, da Programme über GUI-Schnittstellen Ausgaben machen können, die vom Win32 API nicht erfaßt werden. Die GUI-Elemente (Knöpfe, Menüs, usw.) werden im Windows Grafikkern verwaltet. Dieser verwendet zwar die gleichen Grafikanweisungen, wie Anwendungsprogramme, er setzt aber auf einer eigenen

Instanz des Win32 APIs auf. Der vom Grafikkern erzeugte Teil des Grafikstroms kann nicht durch Ausgabeumleitung an der Programmierschnittstelle erfaßt werden.

1.4.3.2. Windows NT Treiberschnittstelle

Die Treiberschnittstelle ist dagegen hervorragend für die Ausgabeumleitung geeignet. Sie ist einfach zugreifbar und in der Semantik skalierbar (siehe Kapitel 2.2.1.2). Dies kommt Implementierungen entgegen, da sich das Application Sharing System so an die Fähigkeiten der empfangenden Terminals anpassen kann.

1.4.3.3. X11 Netzwerkschnittstelle

Die X11 Netzwerkschnittstelle wird im allgemeinen als sehr einfach zugänglich betrachtet. Aus diesem Grund entstanden die ersten Application Sharing Systeme für das X Window System. Sowohl bei der Netzwerkschnittstelle, als auch bei der Programmierschnittstelle hängt die Schwierigkeit des Zugriffs sehr vom Betriebssystem ab. So kann Ausgabeumleitung an der Programmierschnittstelle auf Systemen mit dynamischer Bibliotheksbindung (shared Library) sehr leicht installiert werden. Ist sie installiert, wird aber nicht verwendet, dann entstehen praktisch keine Geschwindigkeitseinbußen. Ausgabeumleitung an der Netzwerkschnittstelle kann dagegen zu Geschwindigkeitseinbußen führen auch wenn Application Sharing nicht verwendet wird, da der gesamte Netzwerkverkehr über eine Zwischeninstanz geführt wird. Auch im besten Fall, d.h. bei effizienter Implementierung des lokalen Netzwerktransports im Betriebssystem und im Sharing System leidet der Durchsatz unter den notwendigen Prozeßwechseln.

Die X11 Netzwerkschnittstelle muß deshalb bei etwas geringerer Semantik, aber mit gleicher Spannbreite in der Zugriffsdimension angeordnet werden.

1.4.3.4. Zugriff

Der Unterschied zwischen leichtem und schwerem Zugriff wirkt sich auf den Implementierungsaufwand aus. Da sich die Bewertung des Zugriffs aus mehreren Komponenten zusammensetzt, muß im Einzelfall abgewogen werden. Die folgenden Punkte haben sich dabei als relevant herausgestellt:

- Aufwand für die Entwicklung der Technik,
- Aufwand für Implementierung,
- Aufwand für Optimierungen, die negativen Effekten (vor allem bezüglich der Geschwindigkeit) entgegenwirken.
- Aufwand für Softwarewartung bei sich ändernden Betriebssystemschnittstellen und Grafiksystemschnittstellen.

1.4.3.5. Funktionalitätslücke

Wie die Zugriffsmöglichkeit, so wirkt sich auch das Abstraktionsniveau der jeweiligen Schnittstelle auf die Komplexität des Sharing Systems aus. Die Komplexität hängt aber nicht direkt mit der Semantik der Schnittstelle zusammen, sondern mit der Funktionalitätslücke, d.h. dem Unterschied zwischen der Semantik der gewählten Schnittstelle und der Semantik des Formats, das vom Sharing System gegenüber den Endgeräten dargestellt wird. Das Application Sharing System muß Aufgaben eines Grafiksystems übernehmen, um einen Grafikstrom von hoher Semantik auf einen mit niedrigerer Semantik abzubilden. Im umgekehrten Fall ist es fast bei allen Kombinationen möglich einen Grafikstrom niedriger Semantik mit denen eines Formats höherer Semantik nachzubilden.

Bei der Wahl der Schnittstelle für die Ausgabeumleitung müssen deshalb die verwendeten Grafikformate für die Netzwerkübertragung und die der Endgeräte in Betracht gezogen werden. Je ähnlicher die Stromformate, desto geringer ist der Aufwand im Sharing System. Dies gilt sowohl für den Implementierungsaufwand, als auch für den Rechenaufwand beim Betrieb.

1.4.4. Eingabekonzentration

Wird ein Programm im Rahmen eines Application Sharing Systems gleichzeitig von mehreren Sitzungsteilnehmern bedient, dann werden im einfachsten Fall die Eingaben aller Teilnehmer dem Programm quasi gleichzeitig zugeführt. Jeder Teilnehmer erzeugt durch seine Eingabegeräte einen Strom von Eingaben. Diese Eingabeströme werden vom Sharing System gemischt und in die

Ereignisverarbeitung des Betriebssystems oder des Grafiksystems eingefügt. Die dafür zuständige Komponente eines Application Sharing Systems wird Eingabekonzentrator genannt.

Die Funktion eines Eingabekonzentrators kann in drei Teile zerlegt werden. Die Eingaberechtsvergabe bestimmt den Zuteilungsmodus des Eingaberechts, die sogenannte Floor Control Policy, und führt die Umschaltung, d.h. die Mischung durch. Diese Komponente, die selbst wieder in Strategie und Ausführungsteil zerlegt werden kann, wurde in der Fachliteratur schon ausführlich diskutiert [Gutekunst95].

Die unterste Schicht des Eingabekonzentrators ist der Eingabeinjektor. Der Injektor fügt künstliche, d.h. vom Sharing System stammende Eingaben entfernter Benutzer in die Ereignisverarbeitung des Betriebssystems oder des Grafiksystems ein. Die Funktionsweise des Injektor ist systemabhängig. In Kapitel 4.5 werden dazu mehrere Varianten beschrieben.

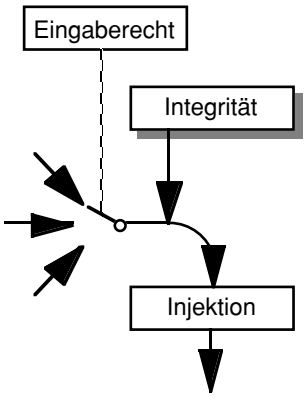


Abbildung Eingabekonzentrator

> Der Eingabekonzentrator besteht aus drei Komponenten (Schichten). Die Eingaberechtsvergabe, die über ein Floor Control Modell die Eingabeumschaltung steuert. Eine zweite Komponente zur Sicherung der Integrität des Eingabestroms und der betriebssystemabhängigen dritten Komponente, die dem System, bzw. dem Programm die künstliche Eingaben zur Verfügung stellt. Die Integritätsbedingung ist dann erfüllt, wenn der kombinierte Eingabestrom, der am Eingabeinjektor anliegt, nicht von dem eines einzelnen Benutzers unterschieden werden kann.

Da gewöhnliche Programme nur für einen Benutzer geschrieben sind, erwarten sie auch nur einen Eingabestrom. Dies bedeutet, daß ein Programm einen Zustandsautomaten implementiert, der bei Eingabeereignissen zwischen Zuständen wechselt und dabei programmspezifische Aktionen auslöst. Solche Automaten decken normalerweise fast alle möglichen Eingabesequenzen eines Eingabestroms ab. Viele Programme werden vom Hersteller im Rahmen der Qualitätssicherung mit einer großen Zahl von Testmustern geprüft. Diese Tests umfassen aber nur Eingabemuster, die von einem Benutzer mit einem Satz von Eingabegeräten erzeugt werden können. Sie decken nicht den weit größeren Raum der Eingabesequenzen von mehreren ineinander gemischten Eingabeströmen ab. Genau dieser Fall, tritt aber beim Application Sharing auf.

Aus diesem Grund sind Programme, bzw. die Zustandsautomaten, die Eingabesequenzen verarbeiten, oft instabil gegenüber gemischten Eingabeströmen. Zwischen der Eingaberechtsvergabe und der Injektion sorgt deshalb eine Komponente für die Stabilität des Gesamtsystems. Diese Komponente hat die Aufgabe, die Integrität des gemischten Eingabestroms zu gewährleisten, d.h. den gemischten Eingabestrom so zu modifizieren, daß der resultierende Strom wie die Eingabe nur eines Benutzers aussieht. Diese Integritätssicherungskomponente existiert bei allen Implementierungen. Sie wurde aber bisher nicht als funktionale Komponente betrachtet. Die Stabilität des Systems bei der Mischung von Eingabeströmen wurde in der Vergangenheit nur als Fehlerbehebung angesehen und ad-hoc, d.h. beim Auftreten eines Fehlers bearbeitet. Tatsächlich ist es aber möglich, systematisch Integritätssicherung zu betreiben. Das Mittel dazu ist eine Eingabeumschaltung auf Eingabegrenzen auf der Basis eines Modells gültiger Eingabeströme kombiniert mit Eingabevervollständigung.

Grundsätzlich darf die Umschaltung von Eingabeströmen immer nur zwischen abgeschlossenen Eingaben geschehen. Damit wird gewährleistet, daß der Eingabestrom, der das Anwendungsprogramm erreicht von der Ereignisverarbeitung des Programms korrekt verarbeitet werden kann. Die Granularität der dabei

betrachteten Eingaben kann von Tastaturanschlägen bis zu Bearbeitungsvorgängen reichen. Sie wird durch das Anwendungsszenarium bestimmt.

1.4.4.1. Eingabemodellierung zur Integritätssicherung

Zur Identifikation von Eingaben wird ein Modell des resultierende Eingabestroms gebildet. Das Modell stellt die für das Zielprogramm gültigen Eingabesequenzen dar. Diese Art der Modellbildung von Eingabeströmen ist angelehnt an Techniken, die beim Entwurf von Benutzerschnittstellen eingesetzt werden. Taskorientierte Interaktionsmodellierung wird dort verwendet, um Benutzerschnittstellen bezüglich Bedienungsfreundlichkeit und -geschwindigkeit zu optimieren [CaMoNe80] [CaMoNe83] [KiWoMe97]. Dazu wird der Eingabestrom hierarchisch zerlegt von der Ebene der Aufgaben über Unterziele bis auf die Ebene von zusammengesetzten und elementaren Benutzereingaben.

Nach der GOMS-Methode (Goals, Operators, Methods & Selection Rules [Kieras88]) werden mögliche Eingabesequenzen in Programmform definiert. Die Eingabesequenzen werden dadurch ausführbar. Ein solches GOMS-Modell enthält Sequenzen von Eingaben, denen jeweils Bearbeitungszeiten zugeordnet sind. Mehrere Alternativen sind möglich. Sie sind jeweils mit Wahrscheinlichkeiten versehen.

GOMS-Modelle können in einem Interpreter ausgeführt werden. Der Interpreter vergleicht die Eingabesequenz mit dem GOMS Programm und kann zu jedem Zeitpunkt über den Zustand des Eingabestroms Auskunft geben. GOMS-Interpreter und GOMS-Modell liefern Informationen über

- Grenzen von Eingaben und
- mögliche Sequenzreste zur Eingabekomplettierung.

Die Verwendung dieser Informationen hängt vom Szenarium ab. Ist die Strategie der Eingabeumschaltung so eingestellt, daß Teilnehmer nicht während Eingaben gestört werden sollen, dann werden Informationen über Eingabegrenzen dazu verwendet, um

- Umschaltzeitpunkte in der Benutzerschnittstelle des Application Sharing Systems anzuzeigen und
- Umschaltung für die Dauer von Eingaben zu unterbinden.

Beim freien Eingabemodus (auch Chaosmodus genannt) sollen nie Eingaben verloren gehen oder blockiert werden. Verwendet die Eingabeumschaltung diese Strategie, dann muß trotz nicht abgeschlossener Eingabesequenzen zwischen zwei Eingabeströmen umgeschaltet werden. In diesem Fall stellt das GOMS-Modell die zur Vervollständigung der Eingabesequenz notwendigen Benutzereingaben zur Verfügung. Die Integritätskontrolle erzeugt diese Eingaben künstlich und fügt sie in den Eingabestrom ein, so daß die Eingabeverarbeitung des Anwendungsprogramms einen gültigen Eingabestrom erhält.

Zusammenfassend bietet die Modellierung der Eingabeströme nach dem GOMS Modell folgende Vorteile:

- Grundlagen: Die Modellierung baut auf den Ergebnissen der Interaktionsforschung auf (HCI: Human Computer Interaction). Zu diesen Ergebnissen gehören die Strukturierung der Interaktion in Ziele (Goals), elementare Eingaben (Operators), Eingabesequenzen (Methods) und Varianten (Selection Rules), sowie die Definition einer entsprechenden Sprache.
- Hilfsmittel: Es gibt Entwurfshilfen für GOMS Programme, z.B. QGOMS [BeSmDe96].
- Daten: GOMS-Modellierung ist eines der wenigen weit verbreiteten Konzepte der HCI-Forschung [JoKi94]. Da GOMS von Entwicklern zur Analyse von Benutzerschnittstellen eingesetzt wird, besteht die Hoffnung, daß zusätzlich zu den selbst erzeugten Modellen geringer Tiefe, auch applikationsspezifische Modelle mit viel Detail von den Softwareherstellern verfügbar sein werden.

1.4.5. Konvertierung von Grafikströmen

Der dritte Arbeitsbereich bei Application Sharing Systemen für heterogene Anwendungen ist die Konvertierung von Grafikausgaben zwischen Grafikstromformaten, d.h. in der Praxis zwischen Grafiksystemen. Beim Application Sharing sind immer mindestens zwei Grafiksysteme beteiligt:

1. das Grafiksystem in dem die Ausgabeumleitung stattfindet und
2. das Grafiksystem in dem Grafikkommandos in einen Pixelspeicher (z.B. Bildschirmspeicher) ausgegeben werden (sog. Rendering).

Jedes der Grafiksysteme definiert ein Grafikformat. Daneben können noch andere Grafikformate eine Rolle spielen; etwa bei der Netzwerkübertragung. Immer dann, wenn sich das Grafikformat ändert, müssen Grafikausgaben übersetzt werden. Bei jedem Übersetzungsvorgang werden Kommandos des Quellsystems in die entsprechende Darstellung des Zielsystems übersetzt. Dies geschieht deshalb eventuell mehrmals innerhalb des Application Sharing Systems. Das Rendering findet dagegen nur einmal statt.

Man kann zwischen zwei Arten der Übersetzung unterscheiden:

1. Rendering im Quellsystem und
2. Rendering im Zielsystem.

Rendering im Quellsystem bedeutet in der Praxis, daß Grafikkommandos noch auf dem System, wo die Ausgabeumleitung stattfindet, in Pixmaps umgewandelt werden und deshalb nur Pixmaps übertragen werden. Dieser Ansatz ist aber auch mit Vorteilen verbunden, wenn sein Hauptnachteil, der größere Bandbreitenbedarf, durch Kompression relativiert wird. Die Vorteile des Renderings im Quellsystem sind

- Vermeidung der Zeichensatzproblematik (siehe Kapitel 5.3.3.6),
- geringere Komplexität des Übersetzers durch Verwendung des Quellgrafiksystems zum Rendering (siehe Kapitel 5.2.1),
- einfache Anbindung an individuelle Bandbreitenadaptierung (CE/SC: Component Encoding/Stream Composition, siehe Kapitel 4.3.4.2).

1.4.5.1. Schnelle Pixmap Konvertierung für Application Sharing

Pixmapformate sind in unterschiedlichen Grafiksystemen sehr ähnlich (Kapitel 2.4.1.2). Für die Anwendung bei Application Sharing Systemen mit Rendering im Quellsystem wurde ein Verfahren entwickelt, mit dem Pixmaps sehr schnell zwischen den Darstellungen verschiedener Grafiksysteme transformiert werden können.

Die Schnelle Pixmapkonvertierung verwendet das Quellgrafiksystem nach dem Rendering nochmals, um die Pixeldaten in das Format des Zielsystems zu transformieren. Da sich an die Übersetzung immer eine Netzwerkübertragung anschließt, wird die Konvertierung durch entsprechende Konfiguration des Quellgrafiksystems in den Netzwerkpuffer durchgeführt, so daß zusätzliche Kopiervorgänge entfallen (sie Kapitel 5.2.2).

1.4.5.2. Einführung des ausgabeorientierten Übersetzers

Kapitel 2.4 dient als Grundlage für den Teilbereich der echten Kommandokonvertierung. Es beschreibt detailliert Gemeinsamkeiten und Unterschiede von Grafikkommandos in verschiedenen Grafiksystemen von displayorientierten Grafiksystemen, wie Quickdraw, Windows (Win32) , und X11 bis zu videoorientierten Systemen, wie GIF und MPEG.

Übersetzung von Grafikkommandos bedeutet, daß grafische Elemente (Grafikprimitive), wie Linien, Rechtecke, Text und Pixmaps, etc. übersetzt werden. Jedes Grafikkommando wird durch einen Satz von Parametern definiert. Die Parameter setzen sich zusammen aus:

- Kontextparametern (übergeben durch Kontextmanipulationskommandos) und
- Prozedurargumenten (übergeben durch Ausgabekommandos).

Kontextparameter werden durch Kontextmanipulationskommandos in einem sog. Grafikkontext vor der Ausgabeanweisung gespeichert. Der Grafikkontext enthält die Parameter, die meistens für mehrere Ausgabeanweisungen gelten.

Ausgabeanweisungen (z.B. Quickdraw Line()) werden beim Aufruf Prozedurargumente mitgegeben. Die Trennung in Kontextparameter und Prozedurargumente rührt daher, daß die Entwickler von Grafiksystemen erwarteten, daß sich einige Parameter ständig ändern, während andere über längere Zeit hinweg gleich bleiben. So wird z.B. die Stiftfarbe meistens als Kontextparameter geführt, da zu erwarten ist, daß mehrere Ausgabeanweisungen mit der gleichen Farbe durchgeführt werden. Dieser Parameter muß deshalb nicht bei jedem Aufruf übergeben werden. Koordinatenparameter ändern sich dagegen normalerweise bei jedem Aufruf. Sie werden deshalb meistens als Prozedurargumente übergeben (Ausnahme: Postscript-ähnliche Grafiksysteme).

Sowohl bei Kontextmanipulationskommandos, als auch bei Ausgabekommandos werden Parameter für Grafikausgaben übergeben. Oft erscheinen mehrere Kontextmanipulationskommandos nacheinander im Grafikstrom gefolgt von einem Ausgabekommando. Neben der Parameterübergabe hat das

Ausgabekommando die wesentliche Funktion, die Ausgabe zu starten, d.h. erst mit dem Ausgabekommando stößt das Anwendungsprogramm die Grafikausgabe an. Die Verteilung von Parametern auf Kontextmanipulationskommandos und Ausgabekommando hängt vom Grafiksystem ab:

- Bei videoorientierten Grafiksystemen werden fast alle Parameter mit dem Ausgabesignal mitgeführt. Nur wenige Parameter werden vorher durch Kontextmanipulation gesetzt (z.B. Transparenzmaske bei GIF, Quantisierungsmatrix bei MPEG).
- Bei displayorientierten Grafiksystemen ist ein großer Teil der Parameter einer Ausgabe im Grafikkontext enthalten. Ausgabekommandos führen nur wenige sich häufig ändernde Parameter mit.
- Bei pfadbasierten Grafiksystemen, wie Display Postscript [Holtzgang90] und Java2d [Java2D] überwiegt die Kontextmanipulation bei weitem. Das Ausgabekommando enthält nur einen Parameter mit Kardinalität 2.

Zum Zweck der Übersetzung von Grafikausgaben ist es vorteilhaft, die Kommandos bezüglich ihrer Funktion zu organisieren. Es soll nicht mehr die Parameterübermittlung im Vordergrund stehen, sondern die Funktion. Damit ergeben sich zwei neue Kommandotypen:

1. die erweiterte Kontextmanipulation, die Manipulation des Grafikkontext, Koordinaten und Grafikprimitiv umfaßt, sowie
2. das Ausgabesignal.

Abbildung *Reorganisation der Ausgabekommandos* zeigt ein kurzes Beispiel:

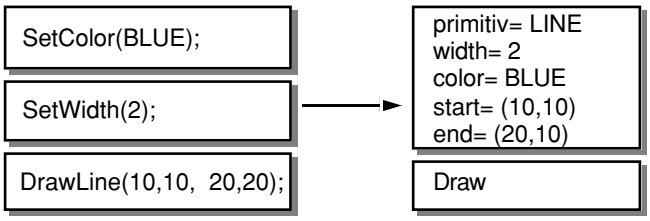


Abbildung Reorganisation der Ausgabekommandos

Alle drei Kommandos auf der linke Seite enthalten Parameter für die Grafikausgabe. Das letzte Kommando startet die Ausgabe. Die rechte Seite zeigt eine Trennung nach Funktionen: alle Parameter werden einem erweiterten Grafikkontext zugeordnet. Das Ausgabesignal veranlaßt das Grafiksystem, die Parameter zu interpretieren, d.h. die Ausgabe durchzuführen.

Diese Art der Umorganisation von Grafikströmen führt zu einer neuen Übersetzerstrategie: dem sogenannten ausgabeorientierten Übersetzer. Der ausgabeorientierte Übersetzer akkumuliert Parameter aus Kontextmanipulationskommandos und dem Ausgabekommando. Er wird erst beim Ausgabesignal aktiviert, d.h. er arbeitet zum gleichen Zeitpunkt, wie das Grafiksystem. Beim Ausgabesignal sind alle für eine Ausgabe notwendigen Parameter im Format des Quellsystems verfügbar. Der Übersetzer rekonstruiert dann die aktuelle Ausgabe mit den Kommandos des Zielsystems.

Zum Vergleich: ein kommandoorientierter Übersetzer arbeitet auf einzelnen Grafikkommandos (Kontextmanipulation und Ausgabekommando). Jedes Kommando wird dabei einzeln in das Format des Zielsystems übersetzt. Dabei kommt es zum Problem der überlappenden Konvertierung, wenn ein zu übersetzendes Kommando nicht genug Information mit sich führt, um ein Kommando im Format des Zielsystems zu erzeugen. In solchen Fällen muß dann Information von einem Verarbeitungsschritt zum nächsten weitergereicht werden.

Ein in der Praxis bedeutender Vorteil des ausgabeorientierten Übersetzers ist die Tatsache, daß ein Softwareentwickler nicht Quellgrafiksystem und Zielgrafiksystem sehr genau kennen muß, sondern nur das Zielsystem, denn es ist seine Aufgabe eine durch entsprechende Parameter definierte Ausgabe im Zielsystem zu erzeugen. Dabei können sogar existierende Programmierschnittstellen für das Zielsystem verwendet werden, z.B. Xlib für Übersetzung nach X11.

1.4.5.3. Detaillierte Implementierungshinweise

Ausgabeorientierte Übersetzung bleibt trotz aller theoretischen Betrachtung ein individuelles Stück Implementierungsarbeit für jeweils eine Kombination von Quellgrafiksystem und Zielgrafiksystem. Es gibt aber eine Reihe von Problemen, die bei verschiedenen Kombinationen von Grafiksystemen immer wieder auftauchen. Da im Verlauf der Untersuchungen und Experimente mit der Übersetzung zwischen

verschiedenen Systemen viele dieser Probleme gelöst wurden, werden die Ergebnisse in Kapitel 5.3.3 ausführlich wiedergegeben.

Zusätzlich gibt es spezielle Probleme, die zwar nur bei bestimmten Grafiksystemen auftauchen, die aber so verallgemeinert dargestellt werden können, daß die gefundenen Lösungen möglicherweise für ähnliche Fälle bei hier nicht behandelten (oder zukünftigen Grafiksystemen) helfen können.

Besonders wichtig ist der Aspekt, daß in Kapitel 5.3.3 auf Problemfälle hingewiesen wird, die nicht offensichtlich sind und deshalb möglicherweise erst während der Implementierung oder noch später bei der Qualitätssicherung auffallen.

1.4.6. Transport

Neben Ausgabeumleitung, Eingabekonzentration und Konvertierung von Grafikströmen ist der Transport von Grafikströmen der vierte Arbeitsbereich.

Prinzipiell können Grafikdaten natürlich wie alle anderen Datentypen über Netzwerke transportiert werden. Dazu steht die gesamte Palette von Netzwerkprotokollen und -diensten zur Verfügung (Übersicht in [Froitzheim97]). Hinzu kommen noch Dienstetypen, die verfügbare Protokollsoftware nicht bietet, die aber auf existierende Dienste aufgesetzt werden kann, indem der Kommunikationsteil des Application Sharing Systems die notwendigen Funktionen implementiert (Application Layer Framing, siehe [ClTe90]). So kann z.B. ein bestätigter Datagrammdienst auf UDP/IP aufgesetzt werden.

1.4.6.1. Anforderungen an Transportsystem und Grafikstromformat

Grafikströme stellen vor allem bezüglich der Zuverlässigkeit Anforderungen an den Transportdienst. Generell sind Grafikdaten empfindlich gegen Bit- und Burstfehler, sowie Paketausfälle. Aus diesem Grund müssen Transportprotokolle mit Fehlerkorrektur verwendet werden. In Kapitel 4.3.3 wird dies für mehrere Szenarien diskutiert.

Auf der anderen Seite können die Randbedingungen auch vom Anwendungsszenarium bestimmt werden. Dies gilt vor allem dann, wenn Multicast zum Transport verwendet werden soll. In diesem Fall muß das Format des Grafikstroms so gewählt werden, daß es multicastfähig ist. Kapitel 4.3.3.3 diskutiert die Anforderungen an das Grafikstromformat.

In großen Multicastgruppen mit heterogenen Endgeräten wurde Application Sharing bisher noch nicht eingesetzt. Für diesen Fall gelten folgende Regeln:

1. es ist eine displayunabhängige und informationsreiche Darstellung zu wählen. Dies bezieht sich vor allem auf:
   - Zeichensätze,
   - Farben (im allgemeinen sollte RGB verwendet, wenn nicht die Anwendung oder die Endgerätetypen so eng eingegrenzt werden, daß auch geringere Farbtiefen genügen),
2. es müssen aktive oder passive Fehlersicherungsmaßnahmen eingesetzt werden,
3. Ressource-IDs müssen vom Sender ohne Vorgaben des Empfängers vergeben werden,
4. Der Datenstrom muß abschnittsweise selbstkonsistent sein, damit
   - eventuell auftretende Restfehler nicht langfristige Auswirkungen haben und
   - später Eintritt ermöglicht wird (Lösung für sog. Late Join Problematik bei Application Sharing durch langlebige Ressourcen, siehe dazu auch [Chung91]).

   Im Rahmen der Diskussion zur abschnittsweisen Vollständigkeit wird im entsprechenden Kapitel die Einführung von verallgemeinerte Stützbildern vorgeschlagen (Kapitel 4.3.3.3).

1.4.6.2. Selektive Kompression

Erlaubt das Anwendungsszenarium die Installierung von Teilen des Application Sharing Systems beim Endgerät, dann ist es möglich Grafikströme zu verschlüsseln und zu komprimieren. In Kapitel 4.3.3 wird besonders eine Kompressionsmethode herausgegriffen, die von Danskin für die Kompression von X11 Protokolldateneinheiten eingeführt wurde [Danskin95]. Diese Kompressionsmethode wird hier mit der oben genannten ausgabeorientierten Übersetzerstrategie zusammengeführt.

Kombiniert ergibt dies eine Kompressionsmethode für jede Art von Sequenzen von Grafikkommandos. Diese sog. selektive Kompression arbeitet auf dem erweiterten Grafikkontext, der alle Parameter von Grafikausgaben umfaßt. Die zeitliche Abfolge der Daten in jedem einzelnen Feld des erweiterten

Grafikkontextes wird als Eingabekanal betrachtet. Ein Grafikstrom besteht so aus vielen Kanälen, die jeweils ihrem Typ entsprechend kodiert werden. Die wichtigsten dabei verwendeten Kodierungen sind:

- Differenzkodierer für Koordinatenparameter,
- Bildkompressionen für Pixmap-Kanäle (auch Füllmuster),
- VLI (Variable Length Integer) Kodierer für Kanäle mit eindeutigen Schwerpunkten, wie Stiftbreite (Schwerpunkt = 1, Messung *Stiftformen*),
- wörterbuchbasierteKompression (z.B. LZW) für Text oder Ersatzbitmaps (siehe Kapitel 5.3.3.6).

1.4.6.3. Bandbreitenanpassung

Die Verkehrscharakteristik von Grafikströmen in Application Sharing Systemen ist 'bursty', d.h. die anfallende Datenrate variiert sehr stark. Die Erfahrung zeigt, daß die Datenrate, die von Anwendungsprogrammen erzeugt und von der Ausgabeumleitung zur Verfügung gestellt wird, häufig die Übertragungskapazität des Netzwerks und die Darstellungsgeschwindigkeit des Endgeräts übersteigt. Wenn der sendende Teil des Sharing Systems blockiert, führt dies zu plötzlichen Stockungen bei der Bedienung des Anwendungsprogramms.

Im Interesse der Bedienbarkeit des Gesamtsystems müssen solche Stockungen auf Anwenderseite so weit, wie möglich vermieden werden. Zu diesem Zweck wurden zwei Methoden entwickelt:

1. globale Adaptierung
   Das Sharing System verwendet eine weiche Kopplung zwischen Programm und Netzwerk indem feingranulare Verzögerungen abhängig von Pufferfüllständen eingefügt werden. Dabei wird der Pufferfüllstand ständig überwacht, um bei Fortschritten die Delayentscheidung zu revidieren (siehe Kapitel 4.3.4.1).
2. individuelle Adaptierung
   Bei paralleler Kodierung für mehrere Endgeräte auf separaten Verbindungen kann sich der Netzwerkdurchsatz der Verbindungen unterscheiden. In diesem Fall soll vermieden werden, daß das Blockieren einer Verbindung alle anderen Verbindungen stört. Gleichzeitig soll bei paralleler Kodierung verwandter Grafikströme der Mehraufwand im Vergleich zur Kodierung nur eines Stroms minimiert werden.
   Beide Ziele werden mit dem sogenannten CE/SC Verfahren erreicht (siehe Kapitel 4.3.4.2). Das CE/SC-Verfahren wurde für reine Pixmap-basierte Grafikströme implementiert. Es ist aber nicht auf Pixmaps beschränkt, sondern kann auch für Grafikstromformate eingesetzt werden, die zusätzlich noch andere Grafikprimitive (Linien, Rechtecke, Bereiche, etc.) enthalten.

2. Grafiksysteme und Fenstersysteme

In diesem Kapitel werden die Eigenschaften der für Application Sharing Systeme wichtigsten Grafiksysteme verglichen. Diese Darstellung dient als Grundlage für Ausgabeanalyse und Ausgabekonvertierung in den Kapiteln 4 und 6. Gleichzeitig werden aus der Analyse Hinweise für die Entwicklung von Grafiksystemen abgeleitet.

2.1. Grafikendgeräte

2.1.1. Notwendige Eigenschaften von Grafikendgeräten

Die Grundfunktion von Grafiksystemen (z.B. Quickdraw, Win32) besteht darin, Grafikkommandos zu interpretieren und darzustellen. Die wenigsten Grafiksysteme verfügen, wie X11 über eine Möglichkeit zum Empfang von Grafikströmen über ein Netzwerk.

2.1.1.1. Kriterien für Grafikendgeräte

Beim Application Sharing werden Ströme von Grafikkommandos über Netzwerke übertragen und dargestellt. Dafür ist es notwendig, Grafikströme über ein Netzwerk zu empfangen. Fehlt diese Funktionalität im Grafiksystem, dann muß sie durch das Sharing System realisiert werden. Man kann unterscheiden zwischen Endgeräten im Sinne der Übertragung eines Grafikstroms zwischen Rechnern und Systemen, die Grafikkommandos lediglich interpretieren können, aber selbst keine Möglichkeit haben, einen Grafikstrom zu empfangen.

Ein Endgerät für den Betrieb im Netzwerk, das auch für Application Sharing eingesetzt werden kann, muß drei Eigenschaften haben:

1. Grafikbibliothek
   Das Endgerät enthält eine Grafikbibliothek von Dekodierungsfunktionen für Grafikkommandos oder Zugriff auf eine Grafikbibliothek. Die Prozeduren der Bibliothek und deren Parameter müssen einer Grafiksyntax entsprechen, die beliebige Pixelanordnungen zuläßt. Das heißt, durch eine endliche Folge von Grafikkommandos soll jedes Pixel einer Pixelgruppe auf einen beliebigen Wert gesetzt werden können.
2. Empfangsmöglichkeit
   Das Endgerät verfügt über einen Mechanismus, um einen Grafikstrom zu empfangen, sei es durch regelmäßiges Abfragen einer RS 232 Schnittstelle oder durch Verarbeitung von Daten aus einer TCP/IP-Verbindung [Postel81], einer ISDN-Verbindung oder von UDP Paketen [Postel80].
3. Streamingfähigkeit
   Dekodierung und Darstellung des Grafikstroms muß inkrementell geschehen. Empfangene Ausgabekommandos müssen sofort ausgegeben werden können, da beim Application Sharing die Ausgabe auf dem betroffenen System sonst gegenüber dem Originalausgabestrom verzögert werden kann. Ausgabekommanods dürfen nur vorher übertragene Information referenzieren. Die Lebensdauer des Grafikstroms, sowie das Gesamtvolumen der Daten, müssen theoretisch unbegrenzt sein.

Soll Application Sharing gleichberechtigt zwischen mehreren Teilnehmern möglich sein, dann muß das Endgerät außerdem auch dazu in der Lage sein, Benutzereingaben entgegenzunehmen und an die Quelle des Grafikstroms zurückzuschicken. Tatsächlich gibt es aber sehr viele Anwendungsfälle in denen nur das gemeinsame Betrachten, nicht aber gemeinsames Editieren von Dokumenten wichtig ist. Diese Bedingung wird deshalb hier nicht zur Definition des Endgeräts herangezogen.

2.1.1.2. Kandidaten für Grafikendgeräte

Die erste Bedingung schließt textorientierte Endgeräte aus. Durch die zweite Bedingung entfallen nicht netzwerkfähige Grafiksysteme und Programme zur Darstellung von statischen und dynamischen Dokumenten, wie MPEG Player, der Adobe Acrobat Reader, Macromedia Player, Powerpoint Projektor usw. Mit der dritten Bedingung werden auch Programme ausgeschlossen, die wie viele Internet Klienten durch HTTP, FTP oder verwandten Protokollen Grafikdateien laden und darstellen. Netzwerkfähige Drucker scheiden ebenfalls wegen der dritten Bedingung aus, obwohl die ersten zwei erfüllt sind.

Es bleibt eine Reihe von Systemen aus verschiedenen Bereichen, die alle zum Empfang und zur Darstellung von Grafikströmen geeignet sind und damit auch zum Application Sharing eingesetzt werden können. Das bekannteste Beispiel aus dem Bereich der Grafiksysteme für Arbeitsplatzrechner ist der Displayserver des X Window Grafiksystems (sog. X-Server) [Xv0-90] [ShGe90] (siehe Kapitel 2.3.4). Der X-Server ist als Front-End zu Anwendungsprogrammen im allgemeinen gedacht und damit natürlich auch zur Benutzung von Anwendungsprogrammen mit mehreren Partnern geeignet. Andere Softwaresysteme, die zur Klasse der Endgeräte gehören sind das Shared Whiteboard [Jacobson97] aus dem MBONE Umfeld [Deering89], und WWW-Klienten [BeCa94], die den vollen Umfang der GIF Spezifikation unterstützen [MuVa94]. In jüngster Zeit erscheinen in Verbindung mit dem WWW noch andere Grafikendgeräte als Komponenten von WWW-Klienten. Diese Komponenten erweitern die Fähigkeiten von WWW-Klienten um die Dekodierung verschiedener Daten- und Stromformate. Die Komponenten sind eigentlich zur Darstellung von Filmen und Multimediapräsentationen in WWW-Klienten vorgesehen. Einige von ihren erfüllen aber alle oben genannten Bedingungen (MPEG Dekoder, Quicktime Dekoder, Macromedia Shockwave, RealAudio). Diese Komponenten liegen in verschiedenen Arten vor. Allen gemeinsam ist aber die Tatsache, daß sie einen WWW-Klienten als Laufzeitumgebung verwenden. Sie unterscheiden sich in der Schnittstelle zum WWW-Klienten und in der Kommandosprache; manche sind direkt ausführbar (Netscape Plug-Ins, ActiveX), andere werden interpretiert (Java Applet [WebShare98], Å Applet [ArJoRo95]) oder zur Laufzeit übersetzt (Java Applet mit JIT).

2.1.2. Grafikfähigkeiten von Grafikendgeräten

Die Fähigkeiten der Endgeräte, bzw. die Syntax der Endgeräte, die in einem Application Sharing System verwendet werden, sind von entscheidender Bedeutung für die entstehenden Datenraten. Sie beeinflussen sowohl die Art der Anwendungsprogramme, die gemeinsam benutzt werden können, als auch das Umfeld in dem (bzw. die Netzwerke über die) Application Sharing überhaupt eingesetzt werden kann. Man kann versuchsweise alle Endgeräte auf einer Skala anordnen, die (unscharf) durch den Funktionsumfang ihrer Grafikbeschreibungssprachen sowie das Datenvolumen von Grafikströmen bestimmt ist. Das Datenvolumen von Grafikströmen hängt natürlich wesentlich von den Inhalten ab. Bei vergleichbaren Inhalten spielen aber die Syntax und das gewählte Stromformat einer Grafikbeschreibungssprache die beiden wesentlichen Rollen für das Datenvolumen (siehe Ankang *Messungen*).

2.1.2.1. Einfache Endgeräte

Endgeräte, die nur pixelweise Ausgaben, bzw. Pixmaps zulassen, bieten die geringste Funktionalität, genügen aber den Mindestanforderungen. Auf der nächsten Stufe steht eine ganze Reihe von Endgeräten, die komprimierte Pixmaps empfangen und darstellen können (z.B. Motion-JPEG). Eine weitere Verbesserung wird erreicht durch die Unterscheidung von Einzelbildern in Stützbilder (Referenzbilder) und Differenzbilder (GIF, Quicktime Apple Video, AVI [Florence94]) in denen nur Änderungen kodiert werden. Wiederverwendung von Bildelementen durch Verschiebung (MPEG), besser Transformation (Fraktale Kompression [Hart96]), bietet weitere Möglichkeiten zur Datenreduktion. Differenzbilder nach der Art der GIF-Kodierung können als eine triviale Variante der Transformation betrachtet werden.

2.1.2.2. Komplexe Endgeräte

Die Fähigkeiten der meisten digitalen Filmformate beschränken sich auf pixmapbezogene Ausgabekommandos. Nur wenige bieten darüber hinaus die Möglichkeit zur Einblendung von Text und grafischen Elementen (Quicktime Text-/Grafikspur, Spritespur). Die Auflösung des festen Zeitrasters für die Präsentation von Einzelbildern und der Verzicht auf Rückwärtsabspielbarkeit erweitern nicht den Funktionsumfang, tragen aber wesentlich zur Reduktion der Datenrate bei. Schon das Quicktime Format bietet mehrere Spuren, die räumlich unabhängig angeordnet werden können. Aber nur dann, wenn eine

Grafiksyntax die dynamische Erzeugung und Verschiebung von Ausgabebereichen erlaubt, können Grafikausgaben in verschiedene Fenster realisiert werden, wie es bei Fenstersystemen üblich ist (X Window Server, NeWS Server [ArGoRo89]). Leider entfallen dafür beim X Window System Mechanismen zur effizienten Kodierung von Pixmaps durch Kompression. OpenStep dagegen erlaubt in Display Postscript auch komprimierte Pixmaps.

2.1.2.3. Programmierbare Endgeräte

Die Programmierbarkeit eröffnet eine andere Dimension in den Fähigkeiten von Endgeräten. Programme in Endgeräten agieren wie ausgelagerte Teile von Anwendungsprogrammen und machen dabei in vielen Fällen Netzwerkverkehr überflüssig. Es gibt mehrere Ansätze, Endgeräte nicht nur für die Ausführung von Grafikkommandos zu verwenden, sondern Grafikkommandos im Endgerät durch die Ausführung von anwendungsspezifischem Code zu erzeugen. Das bekannteste Beispiel ist das auf Postscript basierende NeWS.

Ausführbarer Programmcode im Endgerät dient dazu, Eingaben zu verarbeiten und Grafikkommandos zu erzeugen. Diese Kommandos müssen nicht über das Netzwerk zum Endgerät übertragen werden. Es spielt prinzipiell keine Rolle, wo die Grafikkommandos erzeugt werden, die vom Endgerät ausgegeben werden. Letztlich werden Ausgabekommandos an den darstellenden Teil des Endgeräts übergeben. Teilweise wurden diese Kommandos über ein Netzwerk übertragen, teilweise im Endgerät erzeugt. Zusammen bilden sie einen Grafikausgabestrom. Bei Grafiksystemen und Endgeräten, die ausführbaren Programmcode übertragen können genügt es deshalb, die Untermenge von Kommandos zu betrachten, die der Grafikausgabe dient.

2.1.3. Betrachtete Grafiksysteme

Die Fähigkeiten verschiedener Endgeräte werden durch das Grafiksystem bestimmt, das jeweils als Grafikengine verwendet wird, unabhängig davon, ob die Funktionalität in der Endgerätesoftware selbst implementiert ist oder im Grafiksystem der Betriebssystemplattform, wie bei displayorientierten Grafiksystemen. Im folgenden werden einzelne Grafiksysteme herausgegriffen, die im Rahmen von Application Sharing bzw. entfernter Darstellung von Grafikausgaben wichtig sind. Die hier betrachteten Grafiksysteme sind:

- MS Windows 32 GDI (bezeichnet als Win32 API oder Win32, um das Grafiksystem vom Betriebssytem MS Windows zu trennen) [Petzold96] [SPM95],
- Windows NT 4.0 Grafiktreiberschnittstelle (bezeichnet als Windows NT Treiber) [NTDDK4.0],
- Macintosh ColorQuickdraw API (bezeichnet als Quickdraw) [IM-I88] [IM-V88],
- Macintosh Quickdraw Grafiktreiberschnittstelle (bezeichnet als Quickdraw Treiberschnittstelle oder Quickdraw GrafProcs) [IM-I88] [IM-V88],
- GIF89a Stromformat (bezeichnet als GIF) [MuVa94],
- MPEG 1 Videostromformat (bezeichnet als MPEG) [MPEG1-92],
- MIT X Window System Protokoll Version 11 Release 4 (bezeichnet als X11) [Xv0-90] ohne PEX [Talbott92],
- MIT X Window System API (bezeichnet als Xlib) [Xv2-90],
- Adobe Postscript (bezeichnet als Postscript) [Postscript] [Holtzgang90].

2.1.3.1. Postscript

Postscript ist eine grafikorientierte Programmiersprache. Im Vergleich zu den anderen hier genannten Grafikbeschreibungssprachen und Grafikstromformaten bietet Postscript sehr weitgehende Möglichkeiten zur Programmierung des Endgeräts. Dies steht im Gegensatz zu anderen Grafikbeschreibungssprachen und Stromformaten, die im Endgerät nur sequentiell abgearbeitet werden. In vielen Fällen werden komplexe Programmkonstrukte von Postscript aber nicht verwendet. Postscript wird vor allem bei generierten Postscriptsequenzen (z.B. Druckertreiber) nur als Grafikbeschreibungssprache benutzt, deren Kommandos, wie die anderer Grafikbeschreibungssprachen nur sequentiell abgearbeitet werden. Komplexere Programmstrukturen kommen nicht zum Einsatz, weil das Postscriptprogramm sequentiell aus Grafikausgabekommandos an den Grafiktreiber generiert wird. Prozedurdefinitionen werden in diesem Fall meistens nur im Vorspann in der Funktion von Makros verwendet. Postscript wird in Form

von Display Postscript (einer Obermenge von Postscript) auch als Grafikbeschreibungssprache für Grafikdisplays benutzt. Bei dieser Anwendung benutzen Programme eine Bibliothek von Prozeduren, auf die von Programmiersprachen, wie 'C' oder 'Objective C' zugegriffen werden kann. Die Funktionalität dieser Grafikbiliothek entspricht genau der von Grafiksystemen, wie Win32, Quickdraw und der Xlib [Holtzgang90].

2.1.3.2. Win32

Die Windows NT Treiberschnittstelle wird hier als Grafiksystem behandelt. Das Grafiktreibermodell von Windows NT 4.0 ist sehr flexibel und modern. Es läßt Raum für Optimierungen durch den Grafiktreiber und bietet die volle Funktionalität der Programmierschnittstelle eines Grafiksystems. Es ist abzusehen, daß die Windows NT Treiberschnittstelle über das sogenannte einheitliche Treibermodell Eingang in zukünftige Versionen aller Windows-Varianten findet.

2.1.3.3. Java

Die entstehende Java Grafikprogrammierschnittstelle der zweiten Generation [Java2D] lehnt sich stark an schon länger existierende Programmierschnittstellen für Postscript, bzw. Display Postscript an. Die Diskussion von Postscript ist deshalb leicht auf Java2d übertragbar. Die Java2d Schnittstelle ist allerdings noch nicht vollständig definiert. Es ist auch noch nicht abzusehen, welche Verbreitung sie unter den Anwendungprogrammierern finden wird. Mit Java2d wird ein postscriptähnliche Grafiksyntax wieder wichtiger, nachdem die Bedeutung von Display Postscript in jüngster Zeit mit Openstep zurückging.

2.1.3.4. Dreidimensionale Grafiksysteme

Mit den hier aufgezählten und im folgenden betrachteten Grafiksystemen wird der bei weitem größte Teil von grafikfähigen Arbeitsplatzrechnern erfaßt. Die Auswahl ist so getroffen, daß sehr unterschiedliche Konzepte berücksichtigt werden. Die Bandbreite reicht von Formaten zur Bewegtbildspeicherung (z.B. GIF) über die wichtigsten displayorientierten Grafiksysteme (X, Win32, Quickdraw) bis zu pfadbasierten Grafiksystemen (Windows NT Treiber, Postscript). Am oberen Ende wurden Grafiksysteme mit dreidimensionalen Koordinatensystemen (PHIGS, PEX, Direct3D, OpenGL) ausgespart.

Grundsätzlich gilt aber, daß die hier vorgestellten Prinzipien auch auf dreidimensionale Grafiksysteme übertragen werden können. Vor allem bei dreidimensionalen Grafiksystemen wird die Übersetzung von Grafikkommandos (Kapitel 5.3.3) ein sehr wichtiger Aspekt, da nur so 3d-Grafikbeschleuniger zum Einsatz kommen können.

Beim Application Sharing zwischen 3d-Grafiksystemen ist der Erhalt der Grafikprimitive notwendig, um im Zielsystem von (Grafik-)Hardwareunterstützung zu profitieren. Bei 3d-Grafiksystemen ist Rendering im Zielsystem, d.h. die Übersetzung von Grafikprimitiven notwendig. Ohne wesentliche Geschwindigkeitseinbußen kann nicht auf Rendering im Quellsystem ausgewichen werden.

2.2. Aufbau von Grafiksystemen

Viele Grafiksysteme leisten nicht nur die Bildschirmausgabe, sondern darüber hinaus auch Druckdienste. Tatsächlich sind bei einigen Grafiksystemen die Druckdienste eine wesentlicher Bestandteil, der Einfluß auf Charakteristika des jeweiligen Grafiksystems nimmt. Virtuelle Koordinaten sind nur eine der Eigenschaften, die geräteunabhängige Ausgaben und damit das Drucken unterstützen. Sie wurden in einigen Grafiksystemen vor allem im Hinblick auf WYSIWYG-Publishing (What-You-See-Is-What-You-Get) eingeführt. Eigenschaften, wie virtuelle Koordinaten, sind oft aber auch bei der Bildschirmausgabe nutzbar. Die Orientierung des Grafiksystems an den Anforderungen von WYSIWYG-Publishing kann sehr weit gehen. So werden Adaptierungen der Seitenbeschreibungssprache Postscript als sogenanntes Display-Postscript auch als Grafiksystem für die Bildschirmausgabe verwendet. Beispiele hierfür sind Display Postscript von NextStep (NeXT) und NeWS von Sun Microsystems [ArGoRo89]. In diesem Zusammenhang sollen aber nur die Eigenschaften von Grafiksystemen betrachtet werden, die für Bildschirmausgaben und den Transport von Bildschirmausgaben über Netzwerke relevant sind.

2.2.1. Komponenten von Grafiksystemen

Der darstellende Teil eines Computersystems wird als sein Grafiksubsystem (oder Grafiksystem) bezeichnet. Es besteht aus einer Reihe von Untereinheiten. Man kann sich diese Untereinheiten als Schichten vorstellen, die grafische Ausgaben von Anwendungsprogrammen bis hinunter zur Grafikhardware bearbeiten und weiterreichen. Ähnlich, wie die Schichten eines Protokollstacks im OSI-Referenzmodell [OSI89] verarbeiten die unteren Schichten eines Grafiksystems den Datenfluß der oberen Schichten. Die Schichten sind untereinander verbunden durch sogenannte Output Service Access Points (OSAPs) (Abbildung *Grafikstackmodell*). Über die OSAPs fließen sämtliche Ausgaben von Anwendungsprogrammen in einer Form, die der jeweiligen Schnittstelle entspricht. Der Datenverkehr ist im Gegensatz zu Netzwerkschnittstellen aber unidirektional. Zum Vergleich von Grafikstromformate an verschiedenen Schnittstellen siehe Kapitel 2.4.

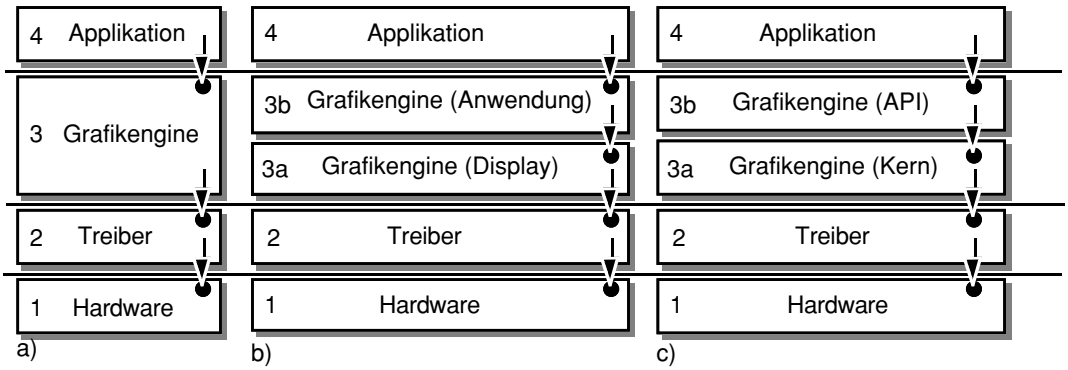


Abbildung Grafikstackmodell

Die Schichten eines typischen Grafiksystems. In der Mitte ein netzwerkfähiges Grafiksystem, dessen Grafikengine aus zwei Teilen besteht. Im Fall des X Window Systems ist die Xlib der anwendungsseitige und der X-Server der displayseitige Teil. Das rechte Teilbild zeigt ein Grafiksystem auf einem Betriebssystem mit getrennten Adresssräumen. Die Programmierschnittstelle kommuniziert mit dem Kern der Grafikengine über shared Memory.

2.2.1.1. Hardwareschicht

Die Hardwareebene bildet die unterste Schicht diese Modells. Sie umfaßt sowohl den physikalischen Bildschirmspeicher, als auch spezialisierte Verarbeitungseinheiten, wie Grafikprozessoren. Der Bildschirmspeicher kann dabei in den Adressraum das Systemprozessors eingeblendet oder von diesem getrennt sein. Er kann linear oder nicht linear, z.B. in Kacheln organisiert sein. Zieht man die Analogie zum OSI-Modell heran, dann entspricht der Bildschirmspeicher der Netzwerkverbindung, während Grafikprozessoren, wie Netzwerkadapter das Medium, in diesem Fall den Bildschirmspeicher manipulieren. Die Schnittstelle der Hardwareebene besteht im allgemeinen aus einer Menge von Adressen auf die Daten geschrieben werden. Ist der Bildschirmspeicher Teil des Adressraums des Systemprozessors, dann werden Daten, die auf Adressen geschrieben werden direkt sichtbar. Im Fall von Grafikhardware mit eigenen Grafikprozessoren steuert der Systemprozessor den Grafikprozessor durch Beschreiben von Registern, die im Adressraum des Systemprozessors liegen.

2.2.1.2. Treiberschicht

Die zweite Schicht ist die Treiberebene. Sie besteht aus Software, die auf die jeweilige Hardware zugeschnitten ist. Treiber von Grafikkarten besitzen Informationen darüber, wie eine spezifische Hardware angesprochen werden muß, um Ausgaben zu produzieren. Beim Austausch der Grafikhardware muß auch jeweils der zur Hardware passende Treiber installiert werden. Die Schnittstelle von Grafiktreibern ist eine Sammlung von Softwareroutinen. Neben Routinen, die der Verwaltung des Treibers im Betriebssystem dienen, gibt es eine meistens kleine Gruppe von Routinen zur Grafikausgabe. Jede dieser Grafikroutinen bietet Zugang zu einer grafischen Basisfunktion, so z.B. zum Setzen eines Bildpunktes, zum Zeichnen einer Linie oder einer Bézier-Kurve. Auf jedem Betriebssystem sind Umfang und Aufgabe der Grafikfunktionen eines Grafiktreibers fest durch das System vorgegeben.

Die Windows NT Grafiktreiberschnittstelle fordert nur die Implementierung einer sehr kleinen Menge von Basisfunktionen. Die Schnittstelle enthält darüber hinaus Funktionen, deren Implementierung dem Treiber freigestellt ist (siehe auch Kapitel 2.4.1). Der Funktionalitätsunterschied zwischen den notwendigen und den komplexeren Funktionen wird bei Bedarf durch die Grafikengine überbrückt. So bricht z.B. die Grafikengine die Ausgabe eines Linienzuges in einzelne Linien auf, falls ein Grafiktreiber die Ausgabe eines Linienzugs verweigert.

2.2.1.3. Grafikengine

Die dritte Schicht eines Grafiksystems (Grafikengine) baut auf den grafischen Basisfunktionen der Treiberebene auf. Sie ist der hardwareunabhängige Teil des Grafiksubsystems. Die Schnittstelle zwischen Anwendungsprogrammen und den Ausgabefunktionen bildet die Programmierschnittstelle (API Application Programmers Interface) des Grafiksystems. Die dritte Schicht enthält oft eine große Zahl verschiedener Ausgabefunktionen. Die Grafikengine dient im wesentlichen dazu die Ausgabekommandos von Anwendungsprogrammen in Kommandos an die Grafiktreiber der zweiten Schicht umzuwandeln. Die dabei anfallenden Aufgaben sind sehr vielfältig. Die folgende Liste führt die wichtigsten Aufgaben auf, die in den meisten Grafiksystemen in der dritten Schicht angesiedelt sind:

- Aufbrechen von Ausgabekommandos in Basisfunktionen der Grafiktreiber,
- Clipping, d.h. Ausblenden von Ausgabeteilen durch eine Maske,
- Farballozierung und -umrechnung, auch Farbtiefenanpassung,
- Koordinatenumrechnung, Skalierung und
- Verwaltung von Parametersätzen für Ausgabekommandos (Grafikkontexte).

Je größer die Funktionalität der hardwareabhängigen Schichten (1+2) ist, desto weniger muß die Grafikengine selbst leisten. Eine große Funktionsvielfalt und Flexibilität auf der Treiberschnittstelle soll meistens die Installation optimierter Hardware und Software ermöglichen. So kann z.B. das Clipping von grafischen Objekten auf Hardwareebene durchgeführt werden. Natürlich steigt damit der Implementierungsaufwand für Entwickler von Displayhardware. Es ist die Aufgabe des Betriebssystems für eine gute Aufgabenverteilung zwischen hardwareabhängigen und hardwareunabhängigen Schichten eines Grafiksystems zu sorgen. Am Beispiel der Windows NT Grafiktreiberschnittstelle (vorhergehendes Kapitel) zeigt sich, daß die Aufgaben auch je nach Vermögen der Schichten dynamisch zur Laufzeit aufgeteilt werden können.

Bei netzwerkfähigen Grafiksystemen ist die dritte Schicht neben den genannten Aufgaben auch für den Transport von Ausgabekommandos zwischen Rechnern zuständig. Die Grafikengine zerfällt dann in zwei Teile; einen anwendungsseitigen und einen displayseitigen Teil. Zwischen diesen findet der Netzwerkverkehr statt (siehe Abbildung *Schichten im netzwerkfähigen Grafiksystem*).

Auch bei nicht netzwerkfähigen Grafiksystemen kann die Grafikengine aus mehreren Teilen bestehen. Oft ist die Programmierschnittstelle von der eigentlichen Grafikengine abgesetzt. Es gibt dann eine knoteninterne (lokale) Kommunikationsverbindung über die Ausgabekommandos von der Programmierschnittstelle an die Grafikengine übermittelt werden. Dies ist besonders bei Betriebssystemen mit Adressraumtrennung der Fall (siehe [Schöttner96]).

Wie am Beispiel des Software-MPEG-Dekoders in Kapitel 2.1 gezeigt, kann die Grafikengine statt einer Treiberschnittstelle auch die Programmierschnittstelle eines anderen Grafiksystems benutzen. In diesem Fall verhält sie sich gegenüber dieser Programmierschnittstelle wie ein Anwendungsprogramm.

2.2.1.4. Anwendungsschicht

Wie im OSI-Modell bildet die Anwendungsebene die oberste Schicht (hier Schicht 4). Die Anwendungsebene umfaßt Programme, die Dienstleistungen von Grafiksystemen zur Bildschirmausgabe nutzen. Der Anteil der Grafikverarbeitung in Anwendungsprogrammen ist sehr unterschiedlich. Viele Programme, wie z.B. Text-Editoren, führen keine eigene Grafikverarbeitung durch, sondern nutzen ausschließlich die Dienste des Grafiksystems. Es gibt aber eine ganze Reihe von hochentwickelten Programmen, die aus verschiedenen Gründen selbst Grafik verarbeiten und nur Basisdienste von Grafiksystemen verwenden, um die erzeugte Grafik darzustellen. Dies geschieht dann, wenn Anwendungsprogramme auf mehreren Grafiksystem-Plattformen laufen sollen oder die Möglichkeiten von Grafiksystemen für die Anwendung zu beschränkt sind. Beispiele hierfür sind die Textdarstellung von Framemaker, die Pixmapmanipulation von Adobe Photoshop und die grafischen Möglichkeiten des Programms Canvas, z.B. für MacOS).

2.2.2. Komponenten von Grafikkommandos

Jede Schnittstelle eines Grafiksystems bietet einen Satz von Ausgabekommandos. Der Satz von Ausgabekommandos einer Schnittstelle bildet, wie in Kapitel 2.1 beschrieben, eine Sprache zur Beschreibung grafischer Ausgaben. Jede Ausgabe wird vollständig beschrieben durch die Angabe des Grafikprimitivs, der Modifikationsparameter und der Kontextparameter.

2.2.2.1. Grafikprimitive

Prinzipiell genügt die Möglichkeit zur Ausgabe einzelner Pixel, um jede beliebige Pixelanordnung zu erzeugen. Ein Ausgabe nur auf der Basis einzelner Pixel ist aber für Anwendungsprogramme nicht praktikabel. Programmierer verwenden deshalb Softwarebibliotheken, die mehr Funktionen bieten. Tatsächlich war bis vor einigen Jahren ein Grafiksystem oft als Funktionssammlung (Toolbox) realisiert, die zu Anwendungsprogrammen gebunden wurde (z.B. bei Borland Turbo Pascal). Heute wird normalerweise nicht mehr das gesamte Grafiksystem zum jedem Programm gebunden, sondern nur noch die Programmierschnittstelle (Schicht 3b, Abbildung *Grafikstackmodell*), während die übrigen Schichten systemweit nur einmal existieren. Trotzdem kann ein Grafiksystem auch heute als Grafikbibliothek angesehen werden, in der eine Reihe von Algorithmen zur Darstellung verschiedener grafischer Grundformen implementiert ist.

Jedes Grafikprimitiv entspricht einem Grafikalgorithmus. So wird zum Zeichnen einer Linie ein anderer Algorithmus verwendet, als zum Zeichnen eines Kreises [FoDaFeHu91]. Der Algorithmus, der die Linien eines - nicht rotierten - Rechtecks zeichnet ist ebenfalls ein anderer, als der für Linien beliebiger Steigung. Zahl und Art der Grafikprimitive sind je nach Grafiksystem verschieden. Es gibt einige Typen, die in allen Grafiksystemen vorkommen. Andere sind nur in einzelnen Grafiksystemen vertreten. Zusammen decken die Grafikprimitive den Grundbedarf der Programmierer and Ausgabemöglichkeiten ab.

Folgende Grafikprimitive werden von den meisten Grafiksystemen angeboten:

In allen Grafiksystemen	Linie Rechteck Ellipsenausschnitt (auch Kreis) Pixmap Text
Nur in einzelnen Grafiksystemen	Pixel Kurve Bereich mit beliebiger Form Verschiebung des Koordinatensytems
Zusammengesetzte Primitive, die von manchen Grafiksystemen wie Basistypen behandelt werden.	Polygon (Linienzug) Rechteck mit abgerundeten Ecken (RoundRect) Rechteckliste

Tabelle Übersicht Grafikprimitive

Überblick über Grafikprimitive in Programmierschnittstellen von Grafiksystemen. Tabelle *Grafikprimitive* enthält eine detaillierte Zuordnung von Grafiksystemen und Grafiksystemen.

2.2.2.2. Modifikationsparameter

Die Angabe des Primitivtyps reicht zur Spezifikation einer grafischen Ausgabe nicht aus. Durch den Primitivtyp wird nur ein Algorithmus einer Grafiktoolbox ausgewählt. Der Algorithmus benötigt zusätzliche Parameter. Die wichtigsten Parameter sind Koordinaten und Dimensionen. Art und Zahl der Koordinatenparameter unterscheiden sich je nach Typ des zu zeichnenden Grafikprimitivs. Tabelle *Modifikationsparameter* zeigt Koordinatenparameter verschiedener Grafikprimitive.

Neben Koordinaten und Dimensionen gibt es für einzelne Grafikprimitive Variantenparameter. Der Variantenparameter führt zu unterschiedlichen Darstellungsarten des Basistyps. Varianten können sich so stark unterscheiden, daß sie in der Praxis durch verschiedene Algorithmen realisiert werden. Der Variantenparameter läßt sich am besten am Beispiel verdeutlichen: ein Objekt kann durch seinen Rand oder durch seinen Inhalt dargestellt werden. Zum Basistyp Rechteck gibt es deshalb die Varianten gerahmtes Rechteck oder gefülltes Rechteck. Zusätzlich unterscheiden viele Grafiksysteme zwischen

dem Füllen mit einer Farbe, bzw. einer Mischung aus Vordergrund und Hintergrundfarbe, und dem Füllen mit einem Muster. Die Varianten unterscheiden sich so sehr in der Realisierung, daß sie eigentlich als verschiedene Basistypen gelten können.

Das Grafikprimitiv Pixmap kann sich ebenfalls unterschiedlich auf den Zielbereich auswirken. Die Wirkung wird bestimmt durch den sogenannten Transfermodus. Eine Pixmap wird oft nicht nur in den Zielbereich kopiert, sondern mit der Pixmap des Zielbereichs verknüpft. Ursprünglich war dies in den meisten Grafiksystemen eine bool'sche Verknüpfung zwischen den Pixeln des Quellbereichs und des Zielbereichs (siehe auch Kapitel 2.4.1.2 und 2.4.2). Bool'sche Operationen werden sogar oft auch auf Farbindexwerte angewandt; nicht nur auf Echtfarben (Truecolor) verschiedener Tiefe, sondern auch auf 8 Bit tiefe Indexwerte in die Farbtabelle. Einige Grafiksysteme beherrschen zusätzlich zu den bool'schen Verknüpfungen auch arithmetische Operationen, die dem menschlichen Farbempfinden mehr entgegenkommen. Als Beispiel sei hier nur ein Transfermodus genannt, der in drei Farbkanälen jeweils den intensiveren Wert der Quell- und Zielpixel in den Zielbereich übernimmt (bei Apple Quickdraw 'adMax' oder Arithmetic Drawing Maximum).

Eine zu zeichnende Linie wird aber nicht nur durch zwei Punkte bestimmt, sondern auch durch ihre Breite, Farbe, Strichmuster und eventuell durch die Art des Linienkopfes. Manche dieser Parameter, wie Farbe und Linienbreite sind auch für andere Grafikprimitive notwendig. Parameter, die von vielen Grafikprimitiven gemeinsam verwendet werden, werden in den meisten Grafiksystemen zu einem sogenannten Grafikkontext zusammengefaßt (siehe folgendes Kapitel). Nur Parameter, die für einzelne Primitive spezifisch sind, gelten als Modifikationsparameter. Tabelle *Modifikationsparameter* zeigt eine Aufstellung von Modifikationsparametern.

Typ Modifikationsparameter	Grafikprimitiv	Werte
Koordinaten	Pixel	Punkt
	Linie	2 Punkte
	Rechteck	2 Punkte
	Ellipsenausschnitt	Punkt, Winkel, 2 Achsen (= Punkt)
	Pixmap	2 Rechtecke (4 Punkte) oder Rechteck + Skalierung
	Text	Punkt + Zeichensatzgröße
	Kurve	Stützpunkte
	Bereich	Punkt
	Polygon	Stützpunkte
	RoundRect	2 Punkte, Radius
Variante	Rechteck, Bereich, Polygon, Ellipsenausschnitt	Rand als Linie zeichnen Inhalt mit Farbe füllen Inhalt mit Muster füllen Inhalt löschen
	Text	Stile
	Pixmap	Transfermodi
	Polygon	offen geschlossen

Tabelle Modifikationsparameter

Die obere Hälfte der Tabelle zeigt welche Koordinatenparameter für einzelne Grafikprimitive relevant sind. Die untere Hälfte zeigt die wichtigsten Varianten für verschiedene Primitive.

2.2.2.3. Kontextparameter

Der Grafikkontext ist eine Sammlung von Parametern, die Grafikausgaben beeinflussen oder bei Ausgaben zu Hilfe genommen werden. Die Zahl und Art der Parameter unterscheidet sich je nach Grafiksystem. Die Zahl der Parameter liegt in der Größenordnung von ca. 20 bis 30 (Win32 [Petzold96], X11 [Xv0-90], Quickdraw [IM-V88]) bei displayorientierten Grafiksystemen. Die Parameter sind aber nicht nur zahlenartige Felder (Zahlen und Aufzählungstypen) in einer Datenstruktur, sondern auch Verweise (Zeiger, Handles oder Nummern) auf komplexere Strukturen, wie Muster, Stifte oder Bereiche.

Der Grafikkontext ist eine Methode zur Optimierung von Grafikausgaben. Er reduziert die Zahl der notwendigen Funktionsargumente für jede Ausgabeoperation. Dies vereinfacht die Programmierschnittstelle. Der Programmierer muß bei Funktionsaufrufen nicht immer alle Parameter angeben, sondern nur die, die sich häufig ändern. Wichtiger war in der Vergangenheit auch die Tatsache, daß möglichst wenige Daten zwischen Anwendungsprogrammen und Grafikbibliotheken ausgetauscht werden. Aus Geschwindigkeitsgründen wurden bei Funktionsaufrufen so wenige Parameter, wie möglich, über den Stack übergeben. Betrachtet man die Verarbeitung von Grafikausgaben auf heutigen Arbeitsplatzrechnern, dann zeigt sich, daß dieser Faktor vernachlässigbar ist. Die Schnittstelle zwischen Programmierschnittstelle (Schicht 2b) und dem Kern des Grafiksystems (Schicht 2a) ist vor allem bei Systemen mit getrennten Adressräumen oft RPC-artig (Remote Procedure Call) realisiert. Diese Art der Kommunikation zwischen Teilen des Grafiksystems ist so aufwendig (siehe Messung *Windows NT LPC*), daß die Zahl der Parameter je Funktionsaufruf praktisch keine Rolle mehr spielt.

Wenn Anwendungsprogramm und grafisches Endgerät räumlich getrennt sind, spielt die Menge übertragener Daten aber eine wesentliche Rolle. Parameter des Grafikkontextes können im Grafikendgerät präsent gehalten werden und müssen nicht für jede Ausgabe über ein Netzwerk übertragen werden. So trägt ein altes Konzept zur Optimierung lokaler Grafikausgaben zur effizienteren Übertragung von Grafikkommandos über Netzwerke bei.

2.2.3. Ressourcen

Ausgabekommandos beziehen sich immer auf grafische Objekte. Manche grafischen Objekte werden in Ausgabekommandos explizit angegeben, z.B. die Form, d.h. der Typ des Grafikprimitivs. Gleichzeitig sind aber bei vielen Ausgaben noch weitere grafische Objekte beteiligt. Sie werden teilweise direkt im Ausgabekommando, teilweise auch indirekt über den Grafikkontext referenziert. Sie gehören zum erweiterten Grafikkontext. Solche Objekte, auf die bei Ausgabeoperationen zurückgegriffen wird, werden Ressourcen genannt (engl. Mittel/Hilfsmittel). Beispiele für Ressourcen sind Pixelmuster, die als Füllmuster für Flächen verwendet werden, Farben bzw. Farbeinträge, Clipping Bereiche, Zeichensätze usw. (siehe Tabelle *Ressourcetypen*). In den folgenden Kapiteln werden die wichtigsten Ressourcen und ihre Verwendung beschrieben. Im Zusammenhang mit der Übertragung von Grafikausgabeströmen über Netzwerke ist vor allem die Verteilung von Ressourcen auf die beteiligten Software und Hardwarekomponenten interessant.

Streng betrachtet müssen sogar Grafikprimitive als Ressourcen angesehen werden. Grafikprimitive sind auch grafische Objekte, die in Ausgabekommandos verwendet werden. Sie bestehen aus dem Programmcode, der den jeweiligen Algorithmus realisiert. Viele Ausgabekommandos enthalten explizite Referenzen auf das jeweilige Grafikprimitiv, d.h. auf den Programmcode, z.B. in Form des Namens des Ausgabekommandos oder des Opcodes (zu PICT Opcodes siehe [Konstanjevec97]).

2.2.3.1. Fenster

Die Fensterzuordnung wird oft in Form von Ressourcen verwaltet. Fenster sind Ausgabebereiche auf dem Bildschirm des Grafikendgeräts (siehe Kapitel 2.3). Anordnung, Form und andere Fensterattribute werden vom Grafikendgerät in einem grafischen Objekt des Typs 'Fenster' zusammengefaßt. Der Zeiger, verallgemeinerter Zeiger (Handle) oder eine stellvertretende Nummer dienen als Referenz auf das Fenster-Objekt. Diese Referenz wird in vielen Grafiksystemen von Ausgabekommandos benutzt, um die Fensterzuordnung von Ausgaben zu bestimmen, d.h. den Fensterbereich zu bestimmen, in dem eine Grafikausgabe vorgenommen wird.

2.2.3.2. Persistenz von Ressourcen

Es gibt zwei Typen von Ressourcen, permanente und temporäre Ressourcen. Temporäre Ressourcen werden von Anwendungsprogrammen während des Betriebs erzeugt, bei Ausgaben verwendet und wieder gelöscht. Sie sind meistens nur im Kontext eines Anwendungsprogramms gültig. Temporäre Ressourcen verschwinden automatisch sobald der Lauf des entsprechenden Programms beendet ist, bzw. sobald die Grafikverbindung zwischen Programm und Endgerät aufgehoben ist. Permanente Ressourcen sind dagegen Eigentum des Grafiksystems. Sie werden Anwendungsprogrammen für die Ausgabe zeitweilig zur Verfügung gestellt, existieren aber unabhängig von Anwendungsprogrammen.

In einigen Grafiksystemen gibt es keine klare Trennung zwischen permanenten und temporären Ressourcen. Es ist dann für Anwendungsprogramme möglich, permanente Ressourcen zu löschen oder

Ressourcen anzulegen, die nach Programmende weiterbestehen, obwohl sie nicht mehr referenziert werden. Dies kann in Einzelfällen vor allem bei fehlerbedingtem Programmabbruch dazu führen, daß der für Ressourcen verfügbare Speicher knapp wird. Ein Beispiel dafür ist der begrenzte Platz des Win32 Heaps unter Windows 95/98 (128 kB). Tabelle *Ressourcetypen* zeigt eine Aufstellung von Ressourcetypen und eine Zuordnung zu Schichten im Grafikstackmodell.

	Ressource	Typ	Schicht [2]
In den meisten Grafiksystemen	Clipping Bereiche	temporär	2, 3a, 3b
	Pixmaps	meistens [1] temporär	3a, 3b,4
	Farben, Farbtabellen	permanent	1, 2
	Zeichensätze	meistens [1] permanent	2, 3a
	Grafikkontexte	meistens [1] temporär [3]	3a
Nur in einzelnen Grafiksystemen	Stifte mit Form, Farbe und/oder Muster	temporär	3a, 3b, 4
	Linienköpfe	temporär	3a, 3b, 4
	Eckpunkte von Polygonzügen	temporär	3a, 3b, 4
	Fenster	temporär	3a
	Cursor/Mauszeigerformen	temporär oder permanent	3a, 3b, 4

1 Bezogen auf die Anzahl der Grafiksysteme im Verhältnis zu allen hier betrachteten.

2 Diese Angabe bezieht sich auf Abbildung *Grafikstackmodell*. Die Zuordnung von Ressourcen zu Komponenten von Grafiksystemen wird in Kapitel 2.2.3.8 diskutiert.

3 Bei Win32 existieren sowohl permanente DCs (Device Context), die reserviert werden müssen, als auch temporäre, die erzeugt werden können.

Tabelle Ressourcetypen

Eine Aufstellung von Ressourcetypen getrennt nach Typen, die in den meisten Grafiksystemen vorkommen und solchen, die nur in einzelnen Systemen existieren; mit einer Zuordnung zu Grafiksystemschichten.

2.2.3.3. Pixmap

Eine Pixmap ist ein zweidimensionales Feld von Farbwerten. Die Farbwerte werden entweder direkt (meistens RGB) oder indirekt als Indexwerte angegeben. Die Indexwerte werden durch eine Farbtabelle auf direkte Farbwerte abgebildet ([IM-V88] Seite 52). Als Indexwerte sind die Farbwerte ein bis acht Bit breit. Die maximale Breite von acht Bit entspricht der Anzahl verfügbarer Farben in typischer Videohardware. Einige Spezialbildschirme und Grafikkarten bieten auch größere Indextabellen, vor allem zur Darstellung von mehr als 256 Graustufen. Die zu den Indexwerten gehörende Farbtabelle ist selbst wieder eine Ressource.

Bei direkter Farbdarstellung geben die Farbwerte der Pixmap – meistens in Form von RGB Komponenten – die Intensität der Farbkanäle an. Die Zahl der verfügbaren Bits je Komponente beeinflußt die Genauigkeit der Farbdarstellung. Sie reicht von zwei Bit bis zu 10 Bit je Farbkomponente. Bei Werten > 8 spricht man von Truecolordarstellung, darunter von Pseudocolordarstellung (X11: Pseudocolor; Win32: Highcolor).

Pixmaps sind die am häufigsten verwendeten Ressourcen (siehe Messung *Reduktion durch Pixmapkompression*). Sie werden auf viele verschiedene Arten bei Ausgabeoperationen verwendet. Außer zum direkten Zeichnen durch das Grafikprimitiv 'Pixmap', spielen Pixmaps eine Rolle beim Clipping, als Füllmuster für andere Grafikprimitive und als Stiftmuster. Wird eine Pixmap direkt zum Zeichnen verwendet, dann ist die Pixmap Teil des Ausgabekommandos. Sie wird dann nicht im voraus erstellt und bei der Ausgabe referenziert, sondern ist Teil eines Ausgabekommandos und nur für dieses gültig.

2.2.3.4. Grafikkontext

Der Grafikkontext (vgl. Kapitel 2.2.2.3) eine Sammlung von Parametern, die Grafikausgaben beeinflussen. Er umfaßt die Parameter von Ausgabeoperationen, die – nach Einschätzung der Hersteller des jeweiligen Grafiksystems – über mehrere Ausgabekommandos hinweg unverändert bleiben. Die Parameter des Grafikkontextes wirken immer als Gruppe auf die jeweilige Ausgabe. Ein Grafikkontext wird dafür bei der Ausgabeoperation referenziert. Dies geschieht, indem ein Verweis (Zeiger, Handle

oder Nummer) des Kontextes in Ausgabekommandos angegeben wird. Der Verweis auf den Grafikkontext steht damit stellvertretend für eine ganze Reihe von Parametern, die so nicht alle explizit bei jedem Ausgabekommando angegeben werden müssen.

Ein Grafikkontext wird durch entsprechende Kommandos erzeugt, bzw. reserviert (X11: CreateGC, Quickdraw: OpenPort/InitPort, Win32: CreateDC/GetDC). Alle Grafiksysteme bieten Kommandos mit denen einzelne Parameter des Grafikkontextes manipuliert werden können. Beispiele hierfür sind das Setzen der Hintergrundfarbe durch das Kommando RGBBackColor (Quickdraw), der Textfarbe (SetTextColor: Win32) oder das Kommando ChangeGC (X11). Vor jedem Ausgabekommando müssen die jeweils relevanten Parameter des Grafikkontextes mittels dieser Kommandos (Kontextmanipulationskommandos) konfiguriert werden. Parameter, die sich nicht ändern, oder die Ausgabe nicht beeinflussen müssen nicht bearbeitet werden.

2.2.3.5. Zeichensätze

Zeichensätze sind Sammlungen von Formen. Auf einzelne Formen wird über den Zeichencode (z.B. ASCII) zugegriffen. Neben dem Zeichencode dienen auch andere Schlüssel, wie Größe, Zeichensatztyp und Stil zur Auswahl einzelner Formen. Siehe dazu Kapitel 2.4.1.2.

In den meisten Grafiksystemen sind Zeichensätze permanente Ressourcen des Grafiksystems. Durch die Bereitstellung von Zeichensätzen vereinfachen Grafiksysteme die Ausgabe von Text. Zusätzlich können in einigen Grafiksystemen (z.B. Display Postscript) Zeichensätze als temporäre Ressourcen zur Programmlaufzeit definiert werden. D.h. Zeichensätze werden über eine Schnittstelle des Grafiksystems definiert und später referenziert. Auf einigen Plattformen besteht für Anwendungsprogramme die Möglichkeit eigene Zeichensätze in ihren Ressourcen mitzuführen und dem Grafiksystem zur Verfügung zu stellen. Die so anwendungsspezifisch verfügbaren Zeichensätze werden aber trotzdem dem Grafiksystem zugerechnet und gelten nicht als vom Anwendungsprogramm zur Laufzeit definiert, da sie über eine Schnittstelle des Betriebssystems und nicht über eine Schnittstelle des Grafiksystems installiert werden.

Der Zeichensatz für Textausgaben wird, wie eine Pixmap, direkt im Ausgabekommando oder indirekt über den Grafikkontext referenziert.

2.2.3.6. Clipping

In vielen Fällen ist es notwendig, Grafikausgaben auf einen Teil, den aktiven Teil, des Ausgabebereichs zu beschränken. Dabei werden Teile von Grafikprimitiven, die über den aktiven Teil hinausragen, bei der Ausgabe abgeschnitten, d.h. nicht angezeigt. Grafiksysteme nehmen den Programmierern von Anwendungsprogrammen die Aufgabe, die resultierenden Ausgaben zu berechnen, ab. Sie bieten die Möglichkeit alle Ausgaben durch eine Maske zu filtern. Dieses automatische Abschneiden von Ausgaben wird Clipping genannt. Die entsprechende Maske heißt Clippingbereich. Der Clippingbereich ist eine Ressource. Er wird entweder direkt bei Ausgaben referenziert (CopyBits: Quickdraw) oder ist einem Grafikkontext zugeordnet, wie z.B. die 'clip-mask'-Komponente des GC (Graphics Context) im X Window System oder die 'clipRgn'-Komponente des Quickdraw Grafports.

Clippingbereiche werden in verschiedenen Grafiksystemen unterschiedlich dargestellt. Es werden aber im wesentlichen drei verschiedene Typen verwendet:

1. Bitmaps,
2. Rechtecklisten oder Kombinationen anderer Grafikprimitive oder
3. Pfade.

Eine Bitmap als Maske bestimmt Pixel für Pixel welche Teile von Grafikprimitiven auf den Bildschirm gelangen (siehe [Xv0-90] S. 115). Die Pixel eines Grafikprimitivs, für die das Bit der Maske nicht gesetzt ist, erscheinen nicht auf dem Display.

Bitmaps können sehr viel Speicherplatz beanspruchen (128 kB je MPixel Maskenfläche). Dies ist vor allem bei netzwerkfähigen Grafiksystemen teuer, wenn Clippingbereiche übertragen werden müssen, um im displayseitigen Teil der Grafikengine zur Wirkung zu kommen. Da die Clippingbereiche in vielen Fällen rechteckig sind, wurde bei den meisten Systemen auch die Möglichkeit vorgesehen einen Clippingbereich als Rechteck oder Menge von Rechtecken zu definieren. Die Definition eines Rechtecks benötigt mindestens 8 Byte (2 Punkte). Aus diesem Grund ist eine rechteckbasierte Definition auch schon bei relativ kleinen nicht rechteckigen Bereichen kompakter als die Bitmapdarstellung vorausgesetzt, die horizontalen Bitstrecken sind über mehr als 64 Pixel gleichförmig (0 oder 1).

Die 'Region' von Quickdraw ist eine Mischform aus Bitmap und Rechteckdarstellung. Sie besteht aus einem umschließenden Rechteck und einer komprimierten Bitmap, die sehr starke Ähnlichkeiten mit einer komprimierten Liste von Rechtecken aufweist [Cohn89]. Das umschließende Rechteck wird bei trivialen rechteckigen Bereichen ohne zusätzliche Daten und bei nichttrivialen Bereichen zur Einschränkung des Testbereichs verwendet.

Postscript arbeitet fast ausschließlich mit Pfaden. Auch der Clippingbereich wird als Pfad spezifiziert. Pfade können in Postscript aus allen Grafikprimitiven aufgebaut werden. Die Konstruktionsvorschrift benötigt deshalb im allgemeinen weniger Daten, als rechteckbasierte oder bitmapbasierte Clippingbereiche (zur Konstruktions von Pfaden siehe Kapitel 2.4.1.3).

Eine Vergleich der Eigenschaften von Pfaden und Bereichen in verschiedenen Grafiksystemen ist in Kapitel 2.4.1.3 zu finden.

2.2.3.7. Hilfsressourcen

Neben den oben genannten Ressourcetypen existieren in einigen Grafiksystemen noch andere spezialisierte Typen, wie Pinsel, Füllmuster, Mauszeiger und Linienköpfe. Prinzipiell werden sie aber von den oben genannten Ressourcetypen abgeleitet. Hilfsressourcen werden nur aus historischen Gründen unterschiedlich behandelt. Da sie den oben genannten Ressourcetypen sehr ähnlich sind, können alle Betrachtungen, die sich auf Pixmaps, Zeichensätze, Pfade und Bereiche beziehen, auch auf die Hilfsressourcen angewendet werden.

2.2.3.8. Zuordnung zu Grafiksystemkomponenten

Im Zusammenhang mit der Übertragung von Grafikausgabeströmen über Netzwerke ist die Verteilung von Ressourcen auf die beteiligten Software und Hardwarekomponenten interessant. Alle Ressourcen, die bei Ausgaben verwendet werden, müssen zum Zeitpunkt der Ausgabe im grafischen Endgerät präsent sein. Sie müssen also vom Anwendungsprogramm zum Endgerät übertragen werden soweit sie dort nicht schon vorhanden sind. Besonders bei der Übertragung von Grafikströmen über Netzwerke ist es wichtig, daß mehrfach verwendete Ressourcen im Endgerät gespeichert und nicht mehrmals übertragen werden.

Man kann unterscheiden zwischen einer Lokalisierung nahe bei Anwendungsprogrammen und nahe beim Grafikdisplay. Die Zuordnung einer Ressource wird bestimmt durch die Schicht des Grafiksystems in der die Ressource tatsächlich gespeichert ist. So werden z.B. anwendungsspezifische Icons von den Anwendungsprogrammen erzeugt und verwaltet. Bei der Ausgabe des Icons sind alle notwendigen Daten – in Form einer Pixmap – im Ausgabekommando enthalten. Sie sind deshalb normalerweise in Schicht 4 (Abbildung *Grafikstackmodell*) präsent.

Vorkehrungen bei netzwerkfähigen Grafiksystemen

Werden Anwendungsprogramme für netzwerkfähige Grafiksysteme implementiert, dann sind sich Programmierer bewußt, daß der gesamte Ausgabestrom über Netzwerke übertragen wird. Sie treffen deshalb Vorkehrungen, um den Netzwerkverkehr auf das notwendige Maß zu beschränken. Die wichtigsten Maßnahmen im Bezug auf Ressourceverwaltung sind dabei:

- Mehrfachverwendung von Ressourcen,
- Lagerung von Ressourcen im Endgerät,
- Modifizierung existierender Ressourcen im Endgerät, statt Erstellung neuer Ressourcen.

Lokale Grafiksysteme

Bei lokalen Ausgaben spielt die logische Zuordnung von Ressourcen zu Komponenten eines Grafiksystems keine wesentliche Rolle. Die Übertragung von Ressourcen und Daten zwischen verschiedenen Schichten eines nicht netzwerkfähigen Grafiksystems findet nur im Speicher eines Arbeitspatzrechners statt. Haben alle Schichten den gleichen Adressraum, dann ist sogar keine explizite Übertragung notwendig. Dies ändert sich aber, wenn der Grafikausgabestrom eines Programms im Rahmen des Application Sharings für die Übertragung über Netzwerke aufbereitet wird (siehe Kapitel 4.3.3). Die Position von Ressourcen, die im lokalen Fall vernachlässigt werden kann, spielt dann eine wesentliche Rolle für das Datenvolumen des aufbereiteten Ausgabestroms. Ressourcen, die von Anwendungsprogrammen nur in Schicht 4 verwaltet werden, müssen jedesmal, wenn sie verwendet werden, über das Netzwerk übertragen werden. Dies geschieht z.B. bei Bildbearbeitungsprogrammen. Bildbearbeitungsprogramme für nicht netzwerkfähige Grafiksysteme verwalten große Pixmaps auf der Anwendungsschicht (siehe Beispielsitzung *Bildbearbeitung*). Diese Pixmaps sind bei Ausgaben jeweils

als temporäre Ressourcen in Ausgabekommandos enthalten. Werden Ressourcen dagegen nahe dem Display, d.h. in den unteren Schichten eines Grafiksystems verwaltet, dann sind nicht die Ressourcen selbst, sondern nur Kommandos zu deren Manipulation im Ausgabestrom enthalten. Bei der Aufbereitung des Grafikstroms zur Übertragung über Netzwerke müssen deshalb auch nur diese Kommandos übertragen werden.

Zeichensätze

Zeichensätze werden vom grafischen Endgerät, bzw. von den displaynahen Schichten eines Grafiksystems für Textausgaben verwendet. Im Gegensatz zu Icons, bei denen die volle Information in Form einer Pixmap vorliegt, wird Text üblicherweise nur durch eine Zeichenkette beschrieben, d.h. als Liste von Verweisen auf Pixmaps. Die Beschreibung des Zeichensatzes ist nicht im Ausgabekommando enthalten. Sie wird nur referenziert. Zeichensätze sind deshalb normalerweise den Schichten 3a oder 2 zugeordnet. Werden Zeichensätze von Anwendungsprogrammen zur Verfügung gestellt, dann werden sie von Schicht 4 nach Schicht 3a geschickt. Bei netzwerkfähigen Grafiksystemen bedeutet dies, daß die Zeichensatzdefinition über das Netzwerk übertragen wird.

2.2.4. Netzwerkfähigkeit

2.2.4.1. Aufbau

Displayorientierte Grafiksysteme können in lokale Grafiksysteme und netzwerkfähige Grafiksysteme unterschieden werden. Für netzwerkfähige Grafiksysteme gilt Teil b) der Abbildung *Grafikstackmodell*. Die Grafikengine ist in einen anwendungsseitigen Teil und einen displayseitigen Teil aufgespalten. Beide Teile kommunizieren über eine Netzwerkschnittstelle. Sie benutzen dabei ein Grafikprotokoll auf der Anwendungsschicht im OSI Referenzmodell.

Die Aufgabe des displayseitigen Teils besteht darin, Ausgabekommandos auszuführen und die Ergebnisse grafisch darzustellen. Er stellt den Anwendungsprogrammen das Display zur Verfügung. Der displayseitigen Teil der Grafikengine wird deshalb Displayserver genannt.

2.2.4.2. Charakteristika

Sowohl Hersteller, als auch Anwendungsprogrammierer von netzwerkfähigen Grafiksystemen, sind sich bewußt (oder sollten sich bewußt sein), daß die Netzwerkübertragung einen Flaschenhals darstellt. In jüngster Zeit ist zwar der Trend festzustellen, dem Netzwerk als Flaschenhals geringere Bedeutung zuzumessen, aber die meisten netzwerkfähigen Grafiksysteme (X11, NeWS, OpenStep) wurden entworfen, als dieser Faktor wichtig war.

Eine der Vorkehrungen, um den Netzwerkverkehr zu minimieren ist die Möglichkeit, Ressourcen im Displayserver zu erstellen, zu lagern und dort zu manipulieren. Dies führt dazu, daß Ressourcen oft langfristig unter Umständen für die gesamte Laufzeit, im Displayserver gespeichert bleiben.

Prozeduraufrufe des Anwendungsprogramms werden vom anwendungsseitigen Teil der Grafikengine in Protokolldateneinheiten (PDU) eines Grafikprotokolls umgewandelt und an die Netzwerksoftware des Systems übergeben. Dort werden Dateneinheiten für das Transportsystem gebildet (TPDU). In der Regel werden jeweils mehrere Ausgabekommandos zu einer TPDU zusammengefaßt. Die TPDUs werden in einem gemeinsamen Speicher (shared Memory) abgelegt. Anschließend werden sie von der Netzwerksoftware dort wieder gelesen. Schließlich empfängt der displayseitige Teil die PDUs und erzeugt wieder Prozeduraufrufe im Displayserver. Jede Datenbewegung im Zusammenhang mit der Netzwerksoftware ist dabei geprägt durch Pufferstrategien. Pufferfüllstände und Timer regeln den Übergang von Daten zwischen verschiedenen Puffern. Im Verlauf der Kommunikation zwischen Anwendungsprogramm und Displayserver werden so an mehreren Schnittstellen Verzögerungen eingeführt.

2.2.4.3. Lokalisierende Eigenschaften

Um den oben genannten Effekt zu vermeiden, wurden Varianten der Grafikengine eingeführt, die den Umweg über die Netzwerkschnittstelle vermeiden. Ein Beispiel ist die sogenannte DirectXlib. Die DirectXlib ersetzt die Programmierschnittstelle des X Window Systems, die Xlib, als dynamisch oder statisch gebundene Bibliothek von Grafikfunktionen. Sie ermöglicht direkten Zugriff auf den Displayserver. Prozeduraufrufe des Anwendungsprogramms werden vom anwendungsseitigen Teil der

Grafikengine nicht in PDUs eines Grafikprotokolls umgewandelt, sondern als Prozeduraufrufe an den Displayserver weitergegeben. Natürlich muß der Displayserver dafür modifiziert werden. Auf diese Weise wird parallel zum netzwerkfähigen Grafiksystem ein lokales Grafiksystem etabliert, das die gleiche Grafiksyntax verwendet.

Anwendungen, die sehr schnelle Grafikausgaben oder Grafikausgaben in Echtzeit erfordern, verwenden oft spezielle Grafikausgabekommandos. Besonders CAD Anwendungen arbeiten häufig mit Hardwarebeschleunigern zusammen, die vom Anwendungsprogramm über eine spezielle Bibliothek angesprochen werden. Aus verschiedenen Gründen laufen diese Anwendungen unter netzwerkfähigen Grafiksystemen. Tatsächlich stellt das Grafiksystem aber oft nur die Oberfläche, den Window Manager und die Fenster zur Verfügung. Der Inhalt der Fenster von CAD Anwendungen wird oft außerhalb des netzwerkfähigen Grafiksystems dargestellt.

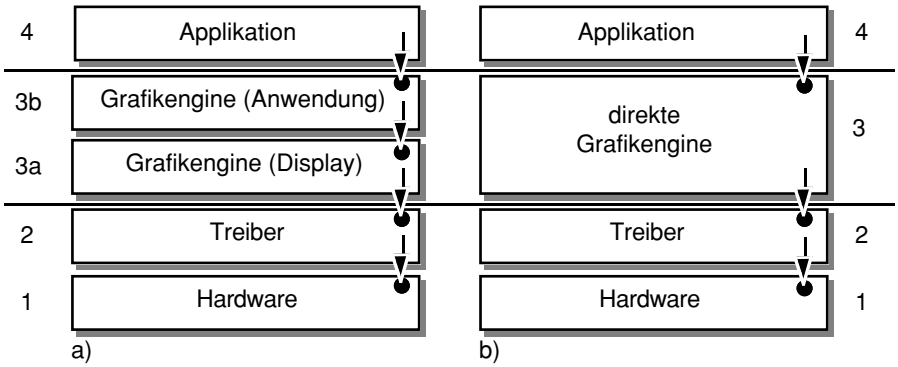


Abbildung Lokalisierende Strukturen

Teil a) zeigt den Aufbau eines netzwerkfähigen Grafiksystems. Teil b) den parallel dazu existierenden optimierten Teil mit einer monolithischen Grafikengine. Die direkte Grafikengine kann mit der gleichen Grafiksyntax, wie der netzwerkfähige Teil arbeiten oder mit anderen auf die Grafikhardware zugeschnittenen Kommandos.

Grafiksysteme mit den hier beschriebenen Eigenschaften gelten im Rahmen dieser Betrachtungen nicht als netzwerkfähige Grafiksysteme, sondern als lokale Grafiksysteme, wie Quickdraw und Win32.

2.3. Fensterverwaltung

Ein Strom von Grafikkommandos erscheint auf dem Bildschirm als bewegte Grafik. Der Grafikstrom wird von einem Programm erzeugt. Tatsächlich erzeugen die meisten Anwendungsprogramme nicht nur einen Grafikstrom, sondern mehrere Teilströme, die unabhängig voneinander in eigenen Teilbereichen des Bildschirms (Fenstern) dargestellt werden. Die Menge der Teilströme ist nicht fest. Teilströme werden dynamisch zur Laufzeit von Programmen erzeugt und beendet.

2.3.1. Schnittstelle der Fensterverwaltung

Beginn und Ende von Grafikströmen werden von den Anwendungsprogrammen gesteuert. Die Programme verwenden dazu die Schnittstelle der Fensterverwaltungskomponente (Window Manager). Während applikationserzeugte Grafikströme den Inhalt von Fenstern füllen, ist der Window Manager für die Anordnung und die Erscheinung von Fenstern verantwortlich. Er gibt Fenstern einen Rahmen und Bedienungselemente über die der Benutzer die Fenster manipulieren kann. Für diese Grafikausgaben benutzt auch der Window Manager die Programmierschnittstelle eines Grafiksystems. Window Manager verwenden für ihre Grafikausgaben oft das gleiche Grafiksystem, wie die Anwendungsprogramme die den Inhalt der Fenster füllen (Abbildung *Fensterverwaltung und Grafiksysteme a*). Ausnahmen gibt es aber dann, wenn die Benutzerschnittstelle eines Anwendungsprogramms unter einem anderen Grafiksystem, als dem bei der Implementierung des Programms verwendeten, dargestellt wird (Abbildung *Fensterverwaltung und Grafiksysteme b*). Beispiele hierfür sind X-Server für MS Windows

oder Apple Macintosh (siehe Kapitel 2.2.4). Auch ein MPEG-Strom besteht aus Ausgabekommandos in MPEG-Syntax, während das zugehörige Fenster z.B. unter MS Windows von einem Window Manager verwaltet wird, der das MS Windows Grafiksystem benutzt.

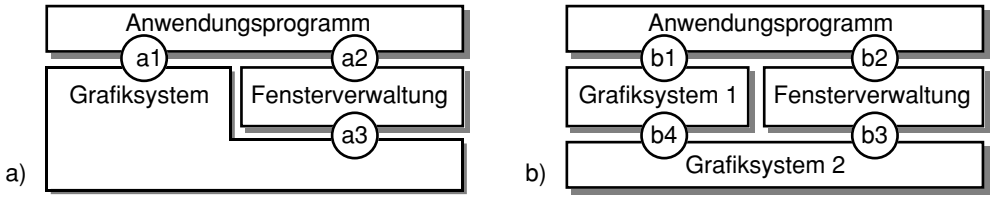


Abbildung Fensterverwaltung und Grafiksysteme

An der Stelle a1 benutzt ein Programm die Programmierschnittstelle des Grafiksystems. Über a2 steuert es die Verwaltung seiner Fenster. Der Window Manager verwendet für seine grafischen Ausgaben die gleiche Schnittstelle (a3), wie Anwendungsprogramme. Im Fall b) setzten Anwendungsprogramm und Window Manager auf verschiedenen Schnittstellen auf (b1 und b3). An der Stelle b4 werden Kommandos aus Grafiksystem 1 in Kommandos für System 2 umgewandelt.

Fensterverwaltung und Grafiksystems können getrennt voneinander betrachtet werden. Die Schnittstelle des Grafiksystems ist für Fensterinhalte zuständig, während die Schnittstelle des Window Managers der Steuerung der Fensterverwaltung dient ([Holtzgang90] S. 32). Es besteht aber trotzdem ein enger Zusammenhang zwischen Grafiksystem und Fensterverwaltung. Neben der historischen Entwicklung gibt es dafür zwei technische Gründe:

1. Der Grafikstrom enthält Fensterzuordnungen für die Grafikkommandos (siehe Kapitel 2.2.3)
2. Softwaremodularisierung durch intensiven Gebrauch von Unterfenstern (siehe Kapitel 2.3.4)

2.3.2. Assoziation von Grafiksystem und Fensterverwaltung

Viele Grafiksysteme, v.a. diejenigen von grafischen Benutzeroberflächen für Arbeitsplatzrechner, sind assoziiert mit einem Fenstersystem. Die Struktur der Programmierschnittstelle, wie Aufrufkonventionen und Datenstrukturen, von Grafiksystem und Fensterverwaltung ähneln sich in diesem Fall oder sind identisch. Da Anwendungsprogramme durch den Grafikausgabestrom sowohl den Inhalt von Fenstern bestimmen, als auch die Fensteranordnung, verwenden sie für beide Bereiche oft den gleichen Ausgabekanal. Besonders deutlich ist dieser Umstand am X Window System und X11 Grafiksystem zu sehen, wo alle Kommandos des Anwendungsprogramms über die gleiche Transportsystemverbindung übermittelt werden. Auf der Transportsystemverbindung wechseln sich Kommandos zur Fensterverwaltung und Kommandos zur Grafikausgabe ab. Da alle Kommandos zur Grafikausgabe für verschiedene Teilströme die gleiche Transportsystemverbindung verwenden, enthalten die Kommandos jeweils einen Verweis auf die Objekte der Fensterverwaltung als Fensterzuordnung.

Prinzipiell könnten sowohl Fensterverwaltung, als auch Grafikteilströme jeweils eigene Transportsystemverbindungen verwenden. Es ist ohne weiteres denkbar, daß jeder Ausgabebereich (Fenster) durch eine Transportsystemverbindung gespeist wird. In diesem Fall müssen der Grafikausgabeströme im Gegensatz zu X11 keine Fensterzuordnungen enthalten. Die Vermischung mehrerer Teilströme von Grafikausgaben und der Fensterverwaltung auf einer Transportsystemverbindung beim X Window System hat sich aber aus praktischen Gründen entwickelt. Durch diese Mehrfachnutzung werden Betriebsmittel, wie Pufferspeicher und Netzwerkverbindungen, eingespart. Ein weiterer Vorteil ist die einfache Möglichkeit zur Synchronisierung mehrerer Ausgabeströme, da die Kommandos auf einer Transportsystemverbindung stets sequentiell übertragen werden.

2.3.3. Funktion von Fenstern

Fenster sind Elemente zur Strukturierung von grafischen Benutzeroberflächen. In einem Fenster wird jeweils ein Grafikstrom oder ein Teilstrom dargestellt. Darüber hinaus haben Fenster aber noch andere Aufgaben:

- Entkopplung der Koordinaten im Grafikstrom von den Koordinaten des Bildschirms,
- Sichtbare Trennung von Grafikströmen,
- Empfänger und Bezugspunkt für Benutzereingaben.

2.3.3.1. Entkopplung von Grafikstromkoordinaten und Bildschirmkoordinaten

Koordinatenparameter von Grafikausgaben werden entsprechend dem aktuellen Koordinatensystem des Grafikkontextes in Pixelkoordinaten umgewandelt (siehe [Petzold96] S. 136). Die Pixelkoordinaten gelten relativ zum Koordinatensystem des Grafikkontextes. Das Grafiksystem arbeitet ausschließlich mit relativen Koordinaten. Die Aufgabe des Fenstersystems ist die Konvertierung der Pixelkoordinaten in absolute Bildschirmkoordinaten. Die entsprechende Transformation hängt von der Lage des Fensters, das den Grafikstrom aufnimmt, ab. Sie ändert sich, wenn das Fenster die Lage - programmgesteuert oder durch Benutzereinwirkung - ändert. Die Koordinaten in Grafikausgaben werden dadurch von der Position des Ausgabestroms auf dem Bildschirm entkoppelt.

2.3.3.2. Trennung von Grafikströmen

Fenster erhalten vom Window Manager eine sogenannte Fensterdekoration, die den Inhalt des Fensters von anderen Elementen auf dem Bildschirm deutlich sichtbar trennt. Die Fensterdekoration besteht im allgemeinen aus einem Rahmen, Bedienungselementen und einem Titel. Die Fensterdekoration wird bestimmt durch den eingesetzten Window Manager und den Fenstertyp. Sie ist verschieden für verschiedene Window Manager. Die meisten Window Manager haben einen Standardtyp für bewegbare, vergrößerbare Fenster, der neben einem Rahmen von einigen Pixeln Breite auch Elemente zur Fenstermanipulation bietet (zu Fenstermanipulation siehe auch Kapitel 2.3.6).

Der Typ eines Fensters bestimmt bei den meisten Window Managern die Manipulationsmöglichkeiten des Benutzers (Fenstereigenschaften). Die wichtigsten Fenstereigenschaften sind bewegbar, modal, vergrößerbar, bzw. verkleinerbar, iconifizierbar und verdeckbar. Sie werden von Programmen über die Programmierschnittstelle der Fensterverwaltung bestimmt und meistens durch entsprechende Bedienungselemente oder Fensterrahmen angezeigt. Die meisten Grafiksysteme bieten zur Vereinfachung einige Typen, wie modale Dialogfenster oder Palettenfenster, bei denen die Fenstereigenschaften fest eingestellt sind. Prinzipiell sind die Eigenschaften aber unabhängig voneinander.

2.3.3.3. Empfänger von Benutzereingaben

In einigen Grafiksystemen dienen Fenster als Empfänger oder Bezugspunkt für Benutzereingaben und anderen Meldungen. Benutzereingaben sind in den meisten Fällen an Anwendungsprogramme gerichtet. Sie dienen der Manipulation von Dokumenten oder zur Steuerung von Abläufen. Sowohl Anwendungsprogramme, als auch Dokumente werden in grafischen Benutzeroberflächen durch Fenster repräsentiert. Das System sendet Benutzereingaben immer an das Objekt, das den Eingabefokus besitzt (siehe Kapitel 2.3.5. Da der Eingabefokus oft an Fenster gebunden ist, wird das jeweilige Fenster automatisch zum Empfänger der Benutzereingaben.

Ereignisgesteuerte grafische Benutzeroberflächen erfordern von Programmen Reaktionen auf Benutzereingaben. Viele Benutzereingaben entstehen durch Fenstermanipulation und nicht durch Eingaben innerhalb von Fenstern. Davon ausgelöste Ereignisse betreffen Sichtbarkeit, Größe und Position von Fenstern. Programme verfügen deshalb über Teile, die auf fensterbezogene Meldungen reagieren. Benutzereingaben an Fensterinhalte, wie z.B. Mausklicks werden durch den gleichen Mechanismus an Programme ausgeliefert, wie Fenstermanipulationseingaben. Sie können deshalb von gleichen oder ähnlichen Programmteilen verarbeitet werden, wie fensterbezogene Meldungen. Dies erleichtert sowohl die Implementierung der Meldungsverarbeitung, als auch die der Warteschlangenverwaltung im Betriebssystem. Bei den meisten Systemen, z.B. MS Windows, X11 und MacOS, findet sogar Interprozeßkomunikation über die gleiche Meldungswarteschlange statt (siehe Kapitel 3. Im Gegensatz zu MS Windows und X11 sind die Meldungen bei MacOS aber nicht an Fenster adressiert, sondern an Prozesse. Nur fensterbezogene Meldungen enthalten bei MacOS eine Fensterreferenz.

2.3.4. Hierarchie

Ein wichtiges Konzept der Fenstersysteme ist die hierarchische Anordnung von Fenstern. Die Hierarchie ist definiert durch eine gerichtete Beziehung zwischen Fenstern: die sogenannte Vater(fenster) -

Kind(fenster) Beziehung. Die Beziehung besteht dadurch, daß Eigenschaften des Kindfensters durch die des Vaterfensters bestimmt werden. Welche Eigenschaften dies sind, hängt vom Fenstersystem ab. Jede der drei, in Kapitel 2.3.3 genannten, Aufgaben von Fenstern findet ihre Fortsetzung in der Beziehung zwischen Vater- und Kindfenster.

- Entkopplung der Koordinaten im Unterfenster von den Koordinaten des Vaterfensters
  Die Koordinaten von Unterfenstern gelten relativ zur Position des Unterfensters im Vaterfenster. Im allgemeinen wird der gesamte Bildschirm als Wurzel der Hierarchie (Wurzelfenster) angesehen. Die erste Stufe von Fenstern unterhalb der Wurzel wird als Toplevel-Fenster bezeichnet.
- Sichtbarkeit und Existenz
  Die Sichtbarkeit von Unterfenstern ist abhängig von der Sichtbarkeit des Vaterfensters. Sie ist durch die Sichtbarkeit des Vaterfensters beschränkt. Das gleiche gilt für die Existenz der Fenster.
- Vorverarbeitung von Benutzereingaben
  Viele Bedienungselemente der Benutzerschnittstelle werden als Unterfenster realisiert. Da die Benutzereingaben von den Unterfenstern verarbeitet werden, können auf dieser Ebene Sequenzen von Benutzereingaben (z.B. Mausklicks, Doppelklicks in höherwertige Kommandos (siehe Kapitel 4.4.2) umgewandelt werden. Die so vorverarbeiteten Meldungen werden von den Unterfenstern dann an die Meldungsverarbeitung des Dokumentenfensters weitergeschickt. Ignoriert ein Unterfenster Eingaben, dann werden die Meldungen automatisch an das Vaterfenster weitergeleitet.

Der Mechanismus der Vorverarbeitung von Benutzereingaben dient der Modularisierung von Softwarestrukturen. Daneben war er ein wichtiges Hilfsmittel zu Reduzierung der Netzwerklast im X Window System. Mit den Athena Widgets [Xv6] wurden Bedienungselemente definiert, die nur vom Window Manager verwaltet werden konnten. So bestehen Knöpfe der Athena Widgets nur aus einem Fenster mit Text, Hintergrundfarbe und Rahmen. Aktivierungen können vom Window Manager ohne Einwirken des Anwendungsprogramms, d.h. ohne Netzwerkverkehr dargestellt werden. Von diesem Mechanismus ist man bei X11 heute abgekommen, da vor allem Knöpfe und Eingabefelder nicht mehr nur durch rechteckige Bereiche dargestellt werden. Die Darstellung von Aktivierungen kann deshalb nicht mehr vom Window Manager alleine durchgeführt werden.

Bei Win32 existiert ein ähnlicher Mechanismus: Bedienungselemente (Knöpfe, Schieberegler, Menüs) werden im displayseitigen Teil der Grafikengine verwaltet (Schicht 3a in Teil c) von Abbildung *Grafikstackmodell*). Sie reagieren ohne Einwirkung des Anwendungsprogramms und melden nur Ergebnisse. Obwohl zwischen den Schichten 3a und 3b bei Win32 kein Netzwerkverkehr stattfindet gleicht das Modell dem von X11 mit Athena Widgets. Die selbständige Aktion der Bedienungselemente, die vom Win32 Grafikkernel gesteuert wird ist der Grund dafür, daß Ausgabeumleitung auf der Ebene der Win32 Programmierschnittstelle nicht den vollständigen Ausgabestrom erfaßt (siehe Kapitel 4.2.5).

Das X Window System und MS Windows sind Fenstersysteme mit den genannten hierarchischen Eigenschaften.

2.3.4.1. Menüs in der Fensterhierarchie

Fast alle Elemente der Benutzerschnittstelle werden letztlich in Fenstern dargestellt. Es gibt einen Konflikt zwischen dem Konzept des Unterfensters und den Bedürfnissen der Benutzerschnittstelle beim Darstellen von Menüs. Ein Menü wird unabhängig von der absoluten Position des Vaterfensters relativ zu dem Fenster positioniert, aus dem das Menü aufgeklappt wird (Menüursprung). Menüs können aber trotzdem nicht als Unterfenster realisiert werden, da sie über das Vaterfenster hinausragen können müssen. Menüs werden deshalb oft als Toplevel-Fenster mit absoluten Koordinaten realisiert. Dies bereitet Probleme beim Application Sharing, wenn die absolute Position des Menüursprungs nicht auf allen Bildschirmen gleich ist. Eine Lösung für diesen Konflikt bietet die

> Einführung einer Klasse von Unterfenstern, die trotz relativer Positionierung nicht durch die Sichtbarkeit des Vaterfensters beschränkt sind. So wie die genannte Vorverarbeitung von Eingaben optional ist, sollen alle Eigenschaften von Unterfenstern unabhängig voneinander wählbar sein. Die Vater-Kind-Beziehung zwischen Fenstern würde damit verallgemeinert auf eine Verwandtschaftsbeziehung in der die Parameter Relativpositionierung, Sichtbarkeit und Eingabeverarbeitung gewählt werden können.

2.3.4.2. Flache Fenstersysteme

Neben den hierarchischen Fenstersystemen gibt es eine Reihe von flachen Fenstersystemen. Flache Fenstersystemen sind Systeme, die keine Abhängigkeitsbeziehung zwischen Ausgabebereichen kennen. Die Möglichkeit der Überlappung von Fenstern ist davon nicht betroffen. Zu den flachen Fenstersystemen gehört der MacOS Window Manager ([IM-I88] S. 267).

Ein sequentieller Grafikstrom im Quicktime-Movie-Format kann mehrere Spuren des gleichen Mediums enthalten. Einzelne Spuren von Video oder Grafikanimationen sind Untergrafikströme. Die Spuren können unabhängig voneinander positioniert und über Spurmasken geformt werden. Die Datenblöcke des Quicktime-Movie-Stroms enthalten Referenzen der Spuren. Mehrere Grafik- bzw. Videospuren verhalten sich so wie unabhängige Fenster eines Fenstersystems. Die Spuren werden zur Laufzeit ein/ausgeblendet.

Natürlich ist der Quicktime-Dekoder (Movie-Player) auf die Dienste des jeweiligen Grafiksystems angewiesen (z.B Quickdraw). Tatsächlich gilt dies aber nur für den Fensterrahmen und die Benutzerschnittstelle des Movie-Players. Der Fensterinhalt des Movie-Players, der für den Quicktime-Dekoder der virtuelle Bildschirm ist, wird aus Geschwindigkeitsgründen nicht über Funktionen des Grafiksystems, sondern über Quicktime-eigene Funktionen dargestellt. Die Spuren eines Quicktime-Movies unterliegen keiner Hierarchie. Das Fenstersystem (die Organisation der Spuren) ist deshalb flach.

2.3.4.3. Einfenstersysteme

Zu den Grafiksystemen zählen alle Grafikausgabesysteme, die sich nach dem Grafikstackmodell (Kapitel 2.2.1) einordnen lassen. Darunter sind auch Grafiksysteme, die mit keiner Fensterverwaltung assoziiert sind. Die jeweiligen Ausgabeströme enthalten keine Kommandos für die Fensterverwaltung und die Grafikkommandos entsprechend keine expliziten Fensterzuordnungen. Ein Beispiel für solch ein Stromformat ohne Fensterverwaltung ist das MPEG-1-Format [MPEG-1]. Ein MPEG-1-Dekoder arbeitet, wie der oben genannte Quicktime-Dekoder. Der MPEG-1-System Strom verwaltet aber nur einen Videostrom. Er ist daher nicht hierarchisch. Die Grafikkommandos des MPEG-Video-Stroms sind in Kapitel 2.4 beschrieben.

Es sei hier angemerkt, daß das im Standardisierungsprozeß befindliche MPEG-4 mit mehreren Spuren (Fenstern), die individuell transformiert und maskiert werden können, ein hierarchisches Fenstersystem ist. Die Standardisierung wird im Dezember 1998 erwartet als ISO/IEC 14496.

2.3.5. Organisation mehrerer Anwendungsprogramme

Auf einem Grafikbildschirm können gleichzeitig die Ausgaben von mehreren Anwendungsprogramme dargestellt werden. Das Fenstersystem sorgt dabei für eine visuelle Trennung der Ausgabeströme. Da es aber nur einen Satz von Eingabegeräten gibt, kann zu jeder Zeit nur ein Fenster, bzw. ein Programm Eingaben vom Benutzer erhalten. Das System muß entscheiden, an welches meldungsverarbeitende Programm (an welchen Event-Handler) die Eingaben der lokalen Eingabegeräte geschickt werden. Bei dieser Entscheidung hilft der sogenannte Eingabefokus. Der Eingabefokus ist eine Eigenschaft von Einheiten der Benutzerschnittstelle. Er wird jeweils einer Einheit zugeordnet und kennzeichnet damit für das System diese Einheit als Empfänger der Benutzereingaben. Die Zuordnung wird im allgemeinen durch Hervorhebung optisch angezeigt.

Die Art der Einheit, welcher der Eingabefokus zugeordnet wird, hängt vom Fenstersystem und vom Betriebssystem ab. Bei hierarchischen Fenstersystemen mit Vorverarbeitung von Benutzereingaben durch Unterfenster (Kapitel 2.3.4) ist die Einheit ein Fenster (MS Windows, X11). Unter MacOS 7/8 gibt es einen Event-Handler für jedes Anwendungsprogramm. Deshalb wird der Eingabefokus einem laufenden Anwendungsprogrammen zugeordnet. Die Einheit, der hier der Eingabefokus zugeordnet wird, ist ein Prozeß. Das Programm, das den Eingabefokus besitzt, wird das aktive Programm genannt.

Die Hersteller von Fenstersystemen haben verschiedene Strategien entwickelt, um sowohl die Koexistenz mehrerer Anwendungsprogramme auf einem Bildschirm, als auch die Zuordnung des Eingabefokus, darzustellen.

Man kann drei Strategien unterscheiden:

1. Fenstersysteme mit applikationseigenem Bildschirm,
2. Fenstersysteme mit applikationseigenem Unterbildschirm und
3. reine Fenstersysteme.

2.3.5.1. Applikationseigener Bildschirm

In Fenstersystemen mit applikationseigenem Bildschirm wird das Anwendungsprogramm, das den Eingabefokus besitzt auf zwei Arten hervorgehoben. Zum einem dadurch, daß seine Fenster über den Fenstern der anderen Programme liegen und zum anderen durch die Menüzeile des Systems, die nur die Menüs des aktiven Programms zeigt [IM-I88].

Eingabefokus

Der MacOS 7/8 Window Manager hebt immer ein Fenster des Programms, das den Eingabefokus besitzt, hervor. Die Kennzeichnung ist aber nur dazu da, um dem Anwendungsprogramm und dem Benutzer das aktive Dokument zu zeigen. Sie steht nicht in direktem Zusammenhang mit dem Eingabefokus. Benutzereingaben werden nicht an den Event-Handler des Fensters geschickt, sondern an den des Anwendungsprogramms. Das ist daran zu erkennen, daß die entsprechenden Nachrichten keine Fensterreferenz enthalten. Nur fensterbezogene Nachrichtentypen enthalten Fensterreferenzen. Diese dient aber nur zur Auswertung der Nachrichten innerhalb des Programms und nicht zur Identifikation des Nachrichtenempfängers, bzw. der Meldungswarteschlange im System.

Menüzeile

Das System verwaltet nur eine Menüzeile. Der Menu Manager wechselt bei jedem Fokuswechsel die Menüzeile aus. Der Wechsel der Menüzeile entsprechend dem aktiven Programm bedeutet, daß zu jeder Zeit nur die Menüs dieses einen Programms bedient werden können. Wenn die Benutzerschnittstelle eines Programms nicht lokal, sondern auf einem anderen Bildschirm, d.h. remote, dargestellt wird, dann können die Menüs dieses Programms nur dann remote bedient werden, wenn die Menüzeile des Programms aktiv ist.

Soll die Menüzeile eines Programms bedient werden, das gerade nicht den Eingabefokus besitzt, dann muß das entsprechenden Programm erst vom Sharing System aktiviert werden. Dabei wird das aktive Programm deaktiviert. Das Umschalten der Menüzeile ist verbunden mit einer Umordnung der Fenster. Sie führt dazu, daß die Inhalt der Fenster des aktivierten Programms eventuell neu gezeichnet wird. Die Umschaltzeit kann deshalb nicht vernachlässigt werden. Während der Umschaltzeit kann kein Programm Eingaben entgegennehmen.

> Um Fehlbedienungen zu vermeiden sollte der Zustand remote bedienter Programme auf allen Bildschirmen angezeigt werden. Dies kann dadurch geschehen, daß die Menüzeile des Programms auch auf entfernten Bildschirmen nur dann dargestellt wird, wenn sie bedienbar ist, d.h. das Programm aktiv ist.

2.3.5.2. Applikationseigener Unterbildschirm

Die Menüzeile ist ein wichtiges Element der Benutzerschnittstelle. Mittels Menüs kann eine große Zahl von Steuerungsmöglichkeiten bei relativ geringem Platzbedarf auf dem Bildschirm bereitgehalten werden. Die jeweiligen Steuerungsmöglichkeiten hängen natürlich direkt vom bearbeiteten Dokumenttyp, d.h. vom verwendeten Anwendungsprogramm ab. Will man das oben genannte Umschalten der Menüzeile vermeiden, so ist in der Benutzerschnittstelle eine Menüzeile für jedes Anwendungsprogramm vorzusehen.

Anwendungsprogramme können meistens mehrere Dokumente gleichen Typs verwalten. Für alle Dokumente gilt die gleiche Menüzeile. Sie muß deshalb nur einmal auf dem Bildschirm existieren. Der Bezug zwischen Dokumenten und Menüzeile kann dann dadurch hergestellt werden, daß alle Dokumente innerhalb eines gemeinsamen Rahmens erscheinen. Der gemeinsame Rahmen bildet gewissermaßen einen Unterbildschirm mit eigener Menüzeile in dem nur die Dokumente eines Programms sichtbar sind. Nebeneinander können mehrere solcher Unterbildschirme existieren. Der prominenteste Vertreter dieser Strategie ist das MDI (Multiple Document Interface) von MS Windows. Beim MDI sind die Dokumentenfenster keine Toplevel-Fenster. Dokumentenfenster sind Unterfenster des Toplevel-Fensters, das den Unterbildschirm mit der Menüzeile repräsentiert.

2.3.5.3. Reine Fenstersysteme

Reine Fenstersysteme bieten keine Mechanismen zur Verwaltung von mehreren Anwendungsprogrammen. Sie beschränken sich auf die Verwaltung von Fenstern. Die Gruppierung von mehreren Dokumenten einer Anwendung und die Organisation von Menüzeilen bleibt den

Anwendungsprogrammen bzw. GUI-Toolkits überlassen. Oft wird in diesem Fall eine Menüzeile je Dokumentenfenster, nicht je Anwendungsprogramm, angezeigt. Wird ein solches Dokumentenfenster remote dargestellt, dann wird die Menüzeile automatisch als Teil des Fensters (z.B. als Unterfenster) mit übertragen.

Im Vergleich der Möglichkeiten zur Organisation mehrerer Anwendungsprogramme auf einem Bildschirm kollidiert der Eingabefokus bei reinen Fenstersystemen am wenigsten mit den Anforderungen von Application Sharing Systemen.

2.3.6. Kontrolle der Fensterverwaltung

Die Aufgaben der Fensterverwaltung sind auf verschiedenen Systemen identisch. Sie werden in Kapitel 2.3.3 beschrieben. Auch die dazu von Anwendungprogrammen verwendeten Funktionen sind bei allen Window Managern sehr ähnlich. Die entsprechenden Kommandos sind in Tabelle *Kommandos der Fensterverwaltung* aufgeführt.

2.3.6.1. Fensterverwaltungskommandos

Fensterverwaltungskommandos kontrollieren die primitiven Funktionen der Fensterverwaltung. Sie können sowohl von Anwendungsprogrammen als auch von Benutzern ausgelöst werden.

Fenstersystem Primitiv	X Window Manager, Befehle aus X-Protokolls	MS Windows	Macintosh System 7 Window Manager
Erzeugen [1]	CreateWindow	CreateWindow CreateWindowEx	NewWindow NewCWindow
Zerstören	DestroyWindow DestroySubWindow	DestroyWindow	DisposeWindow CloseWindow
Darstellen (sichtbar machen)	MapWindow MapSubWindows	ShowWindow	ShowWindow ShowHide
Verbergen	UnmapWindow UnmapSubWindows	ShowWindow	HideWindow ShowHide
Symbolisieren	ConfigureWindow	CloseWindow OpenIcon	[2]
Bewegen	ConfigureWindow	MoveWindow SetWindowPos	MoveWindow
Größe ändern, Standardgröße zuweisen	ConfigureWindow	MoveWindow SetWindowPos	SizeWindow
Stapelfolge ändern	CirculateWindow ConfigureWindow	BringWindowToTop SetWindowPos	SelectWindow BringToFront SendBehind

1 Es gibt oft abgeleitete Funktionen zur Erzeugung von Fenstern mit speziellen Eigenschaften, die auf den genannten Funktionen basieren und deshalb hier nicht aufgeführt sind.

2 Nur für Benutzer über Benutzerschnittstelle des Window Managers; kein Primitiv der Fensterverwaltung, sondern simuliert durch eine Folge anderer Kommandos.

Tabelle Kommandos der Fensterverwaltung

Die Primitive der Fensterverwaltung ähneln sich bei verschiedenen Window Managern.

Alle Window Manager bieten für Anwendungsprogramme Funktionen zum Öffnen, Schließen, Plazieren, bzw. Bewegen, Anordnen und Skalieren von Fenstern. Window Manager unterscheiden sich weniger in ihren Funktionen für Anwendungsprogramme, als in ihrem Erscheinungsbild und den angebotenen Kontrollelementen für den Benutzer. Vergleicht man z.B. den Ablauf einer benutzerinitiierten Größenänderung so zeigen sich wesentliche Unterscheide in der Benutzerschnittstelle des Window Managers, obwohl die Kernfunktion, nämlich die Änderung der Fläche eines Ausgabebereichs identisch ist (siehe unten). Obwohl die Benutzerschnittstelle von Window Managern für Benutzer sehr wichtig ist, soll hier nicht genauer darauf eingegangen werden, weil keine prinzipiellen Unterschiede bestehen.

2.3.6.2. Fensterverwaltungsstrategie

Neben den Kommandos, die Anwendungsprogrammen zur Verfügung stehen, bieten Window Manager auch direkte Einflußmöglichkeiten für Benutzer. Manche der dabei verwendeten Funktionen werden vom Window Manager in enger Zusammenarbeit mit dem jeweiligen Anwendungsprogramm ausgeführt, während andere unter der Kontrolle des Window Manager ablaufen und nur indirekt durch Anwendungsprogramme gesteuert werden können.

Im Bereich der Benutzerinteraktion bestehen große Unterschiede zwischen verschiedenen Window Managern. Besonders beim X Window System, wo Window Manager auf Benutzerebene installiert und einfach ausgetauscht werden können, bieten Window Manager oft sehr viele zusätzliche Funktionen. Diese zusätzlichen Funktionen sind für Anwendungsprogramme weder verfügbar noch steuerbar. Sie sollen hier unbeachtet bleiben, da sie alle auf die oben gezeigten Primitive zurückgeführt werden können und im allgemeinen für das Sharing von Anwendungsprogrammen keine Rolle spielen.

Nehmen Benutzer Einfluß auf die Fensterverwaltung, dann ist nicht nur der Window Manager betroffen, der die entsprechenden Kommandos ausführt, sondern auch das jeweilige Anwendungsprogramm. Das Anwendungsprogramm muß sich in vielen Fällen auf die geänderten Bedingungen (z.B. Größenänderung von Fenstern) einstellen. Viele Funktionen der Fensterverwaltung werden also in Zusammenarbeit zwischen dem Window Manager und Anwendungsprogrammen ausgeführt. Dabei ist die Aufgabenteilung zwischen Anwendungsprogramm und Window Manager sehr wichtig für das Sharing System. Man kann im wesentlichen zwei Strategien zur Aufgabenteilung zwischen Anwendungsprogramm und Window Manager unterscheiden:

- Aktiver Window Manager
- Passiver Window Manager

Aktiver Window Manager

Ein aktiver Window Manager führt Benutzerinteraktionen selbständig durch. Er beachtet dabei die Vorgaben des jeweiligen Anwendungsprogramms für das betroffene Fenster informiert das Anwendungsprogramm aber erst nach dem Ende der Aktion über Ergebnisse. Ein Programm kann das Verhalten des Window Managers steuern indem es für einzelne Fenster im voraus Vorgaben macht, ist aber bei der Durchführung nicht beteiligt. Die Vorgaben betreffen meistens die Einhaltung bestimmter Parameterbereiche, wie maximale und minimale Fenstergröße oder die Position. Das Programm erhält keine Benutzereingaben, sondern nur die Meldung über Änderungen der Fensteranordnung.

Passiver Window Manager

Ein passiver Window Manager agiert nur auf Anforderung von Anwendungsprogrammen. Benutzereingaben, die Window-Management zum Ziel haben, werden, wie andere Benutzereingaben, zuerst an das Anwendungsprogramm geschickt, das den Eingabefokus hält. Das Anwendungsprogramm benutzt daraufhin Funktionen des Window Managers zur Durchführung der Benutzerinteraktion und kontrolliert den Ablauf.

Beispiele

Die Window Manager des X Window Systems sind Beispiele für aktive Window Manager während der Window Manager des Macintosh System 7 als passiver Window Manager gilt. Zwischen den beiden Extremen sind natürlich Mischformen möglich. Auch beim X Window System ist es möglich, die Kontrolle über Benutzerinteraktion ganz bei Anwendungsprogrammen zu belassen. Dies wird in speziellen Fällen getan, wenn Anwendungsprogramme die Anordnung ihrer Fenster exakt kontrollieren müssen. Auf der anderen Seite ist der Window Manager des Macintosh System 7 nicht vollständig passiv. Zwar erhält das Anwendungsprogramm die Meldung über den Beginn der Benutzerinteraktion, kann dann aber die Kontrolle abgeben, bis der Vorgang beendet ist (siehe GrowWindow-Funktion [IM-I88]). Das Umschalten zwischen verschiedenen Anwendungsprogrammen mit der Änderung der Fensterstapelfolge geschieht ohne Beteiligung des aktiven Programms selbständig durch den Window Manager.

Weichen die Strategien verschiedener Window Manager auf Quellsystem und Zielsystem voneinander ab, dann muß das Sharing System dafür einen Ausgleich schaffen (siehe Kapitel 5.4.2).

2.3.7. Relation von Fenster und Grafikkontext

Die Softwarestrukturen Fenster und Grafikkontext haben sehr unterschiedliche Aufgaben. Während der Grafikkontext zur Organisation von Ausgabekommandos innerhalb eines Grafikstroms dient, ist es die Aufgabe von Fenstern mehrere Grafikströme zu arrangieren. Werden beide Strukturen miteinander vermischt, dann können Probleme für Application Sharing Systeme entstehen. Dies soll hier am Beispiel des Macintosh Window Managers und Quickdraw Grafiksystems gezeigt werden.

Beim Macintosh ist die Window-Datenstruktur eine Erweiterung der Grafport-Struktur (Grafikkontext), wie eine abgeleitete Klasse im objektorientierten Sinne. Spezielle Fenster, wie Dialogboxen, sind wiederum Erweiterungen der Window-Datenstruktur. Damit ist jedem Fenster jeweils ein spezieller Grafikkontext zugeordnet. Beim Sharing wird die Zuordnung in umgekehrter Richtung benötigt, d.h. eine Zuordnung des Fensters zum Grafikkontext (siehe auch Kapitel 4.2.6). Die Zuordnung des Fensters zu einem bekannten Grafikkontext ist einfach, da Fenster und Grafikkontext bis auf zusätzliche angehängte Datenfelder identisch sind. Anwendungsprogramme können aber auch andere zusätzliche Grafikkontexte für Ausgabeoperationen verwenden (z.B. bei MacOS 7/8). Außer der Identität der Datenstrukturen existiert aber kein Zuordnungsmechanismus. Es gibt deshalb keine Möglichkeit diesen zusätzlichen Grafikkontexten ein Fenster zuzuordnen.

Die Vermischung von Fenster und Grafikkontext bewirkt, daß kein expliziter Zuordnungsmechanismus zwischen Fenster und Grafikkontext notwendig ist (wie z.B. unter Win32: WindowFromDC). Benutzen Programme für Grafikausgaben andere Grafikkontexte, als den fenstereigenen Grafikkontext, dann ist es nicht möglich, diese Ausgabeoperationen dem richtigen Fenster zuzuleiten.

Sind die Datenstrukturen Fenster und Grafikkontext miteinander vermischt (s.o.), dann darf es entweder nur einen möglichen Grafikkontext je Fenster geben oder es muß ein expliziter Zuordnungsmechanismus eines Fensters zum Grafikkontext existieren (z.B. durch Referenzen im Grafikkontext oder als Systemfunktion).

2.4. Grafikkommandos

In Kapitel 2.2.2 wurden die verschiedenen Teile von Grafikkommandos auf allgemeine Art beschrieben. Diese Analyse beruht auf einer Auswertung der Grafiksysteme, die eingangs aufgezählt wurden (Kapitel 2.1.3). Bei realen Anwendungen spielen jedoch nicht die allgemeinen Eigenschaften eine Rolle, sondern die speziellen Eigenschaften der beteiligten Grafiksysteme. In den folgenden Kapiteln werden die Ausgabekommandos verschiedener Grafiksysteme miteinander verglichen.

2.4.1. Vergleich von Grafikprimitiven

Die unterschiedlichen angebotenen Ausgabekommandos sind die auffallendsten Unterschiede zwischen Grafiksystemen. Es besteht ein deutlicher Unterschied zwischen der Funktionalität der Grafiksysteme von Arbeitsplatzrechnern (Quickdraw, X11, Win32) und den videoorientierten Grafikformaten (MPEG und GIF).

Grafikströme beider Gruppen bestehen aus einer Folge von Kontextmanipulationen und Ausgabekommandos. Dabei sind die Ausgabemöglichkeiten von videoorientierten Formaten aber sehr beschränkt. MPEG bietet nur Pixmaps, die allerdings effizient kodiert sind, während GIF auch Textausgaben zuläßt. Die Ausgabemöglichkeiten der Grafiksysteme Quickdraw, X11 und Win32 sind wesentlich vielfältiger. Aber auch innerhalb der Gruppe der Grafiksysteme von Arbeitsplatzrechnern gibt es deutliche Funktionalitätsunterschiede.

Im Vergleich zu den anderen hier genannten Grafikbeschreibungssprachen und Grafikstromformaten bietet Postscript sehr weitgehende Möglichkeiten zur Programmierung des Endgeräts. Allerdings werden im Fall von generierten Postscriptsequenzen Prozedurdefinitionen, Variablen, Schleifen und Bedingungen nur im Vorspann in der Funktion von Makros verwendet. Alle Ausgabemöglichkeiten, einschließlich der durch Makros definierten Kommandos, basieren auf wenigen grafischen Grundfunktionen, die denen anderer Grafiksysteme sehr ähnlich sind (siehe Tabelle *Grafikprimitive*).

Grafiksystem: Grafikprimitiv	X11 Protokoll-schnitt-stelle[8]	Win32[2] API	Quickdraw API	Windows NT Treiber-schnitt-stelle	Quickdraw Treiber-schnittstelle	MPEG	GIF	Postscript
gerade Linie	PolyLine PolySegment	LineTo	Line LineTo		Line[6]			lineto[11]
Ellipsenausschnitt und Ellipse	PolyArc PolyFillArc	Arc ArcTo AngleArc Chord Pie Ellipse	Oval[1] Arc[1]		Oval[6] Arc[6]			arc[11] arcto[11] arcn[11]
Rechteck	PolyFill-Rectangle PolyRectangle	Rectangle FillRect FrameRect InvertRect DrawFocusRect	Rect[1]		Rect[6]			rectstroke rectfill
Pixel	PolyPoint	SetPixel SetPixelV	SetCPixel					
PixMap verschiedener Tiefe[10]	PutImage ClearArea CopyArea CopyPlane	SetDIBitsToDevice StretchDI-Bits BitBlt PatBlt PolyPatBlt MaskBlt PlgBlt StretchBlt DrawIcon ExtFloodFill	CopyBits CopyMask ScrollRect PlotCIcon	BitBlt StretchBlt CopyBits[4]	Bits[6] Pix[7]	Macroblock	Image	image imagemask
Polygon	PolyLine FillPoly	Polyline PolylineTo Polygon PolyPolygon PolyPolyline	Poly[1]		Poly[6]			
Text	ImageText8 ImageText16 PolyText8 PolyText16	TextOut ExtTextOut PolyTextOut TabbedTextOut DrawText	DrawChar DrawText DrawString DrawJustified	TextOut[4]	Text[6]		PlainText	text[11] charpath[11]
Bereich beliebiger Form[5]		FillRgn FrameRgn InvertRgn PaintRgn	Rgn[1]	FillPath StrokePath[4] StrokeAndFillPath Paint[4]	Rgn[6]			stroke[9] fill[9]
Rechteck mit runden Ecken		RoundRect	RoundRect[1]		RoundRect[6]			
Kurve		DrawBezier PolyBezier PolyBezierTo PolyDraw						curveto[11]
Bereich Füllen		FloodFill		Paint				

1 Dieses Kommando existiert in den sechs Variationen mit den Präfixen 'Frame', 'Paint', 'Erase', 'Invert', 'Fill' und 'FillC'.
2 Einige Kommandos sind auch in modifizierten Versionen verfügbar, die zusätzliche Kontrollparameter aufnehmen (siehe FloodFill und ExtFloodFill).
3 Die SysBeep-Funktion ist nicht Quickdraw zugeordnet, entspricht aber den Tonausgabefunktionen in anderen Grafiksystemen.
4 Wenn der Bildschirmspeicher vom Systemprozessor aus beschreibbar ist, dann können alle Funktionen durch Standardfunktionen der Grafikengine ersetzt werden. Ist dies nicht der Fall, dann ist der Bildschirmspeicher "device controlled", d.h. nur über einen Grafikprozessor ansprechbar. Diese Funktion muß dann im Treiber realisiert sein.
5 Bereiche beliebiger Form decken auch Grafikprimitive ab, die nicht gesondert aufgeführt sind. So werden über die StrokeAndFillPath Funktion der Windows NT Treiberschnittstelle auch Rechtecke, Kreise und (Bezier)Kurven gezeichnet.
6 Mit Präfix 'Std' Teil der Quickdraw Treiberschnittstelle, die sog. CGrafProcs [IM-V88], Seite V-110.
7 Mit Präfix 'Std' Teil der Quickdraw Treiberschnittstelle. Die StdPix-Routine nimmt den Platz der newProc1-Routine der sog. CGrafProcs ein, siehe [IM-V88] Seite V-110.
8 Die Xlib enthält die Umsetzungen der Protokollschnittstelle in Prozeduraufrufe. In der Xlib existieren ähnliche Kommandos, wie in der Protokollschnittstelle. Die Prozeduren der Xlib verwenden alle ein 'X' als Präfix zu den Kommandonamen.
9 Stroke und fill sind keine selbständigen Primitive. Sie dienen zur Ausgabe von Pfaden, bzw. Bereichen, die durch Kombination anderer Primitive zusammengestellt werden. Nach der Erstellung können Pfade aber wie Primitive verwendet werden und sind deshalb hier aufgeführt.
10 Zu Pixmapausgaben werden neben Kopieroperationen von Pixelgruppen in Bildschirmbereiche auch Fülloperationen gezählt, soweit diese sich nicht auf bestimmte Grafikprimitive beziehen und bei diesen Grafikprimitiven aufgeführt sind.
11 Dieses Kommando ist kein Ausgabekommando, d.h. es erzeugt keinen sichtbaren Effekt. Es gehört zur Kontextmanipulation.

Tabelle Grafikprimitive

Aufstellung der in den Grafiksystemen verfügbaren Grafikprimitive und der zugehörigen Ausgabekommandos. Ausgabekommandos, die aufgezeichnete Grafiken (PICT, WMF) ausgeben, wurden hier ausgeklammert, da sie sich ihrerseits wieder auf die angegebenen Kommandos abstützen.

Die oben (Kapitel 2.2.2) angestellten Überlegungen dienen als Grundlage, um verschiedene Grafiksysteme in Beziehung zueinander zu setzten und einzuordnen. Ein wichtiger Faktor sind jeweils von einem Grafiksystem angebotenen Grafikprimitive. Die Tabelle soll Unterschiede und Gemeinsamkeiten verdeutlichen. Daneben zeigt sie die Zahl der angebotenen Ausgabekommandos. In einigen Grafiksystemen, vor allem den historisch gewachsenen displayorientierten Grafiksystemen, gibt es oft mehrere Kommandos, die das gleiche Primitiv auf unterschiedliche Art (mit geringen Modifikationen) ausgeben. Die Zahl der angebotenen Ausgabekommandos ist wichtig für Application Sharing Systeme, wie in Kapitel 4.2 beschrieben.

2.4.1.1. Linien und einfache Formen

Linien und einfache Formen sollen hier nur kurz beschrieben werden, da sie sich in allen Grafiksystemen, wenn vorhanden, nicht wesentlich unterscheiden. Prinzipiell sind zwei Variationen zu unterscheiden:

1. Umrahmen der Form, z.B. FrameRect (Quickdraw), und
2. Füllen (ausgenommen gerade Linien), z.B. FillRect und PaintRect (Quickdraw).

Beim Zeichnen von Rand und Inhalt wird in der Praxis oft unterschieden zwischen dem Zeichnen mit

- einer Farbe (PaintRect) und
- einem Füllmuster, auch Pinsel genannt, definiert durch eine Pixmap (FillRect).

Zeichnen mit einer Farbe ergibt eine homogene Färbung. Diese Betriebsart kann oft noch durch eine Bitmaske modifiziert werden, die dann die Verteilung einer Vordergrund- und einer Hintergrundfarbe bestimmt, so daß tatsächlich mit 2 Farben gezeichnet wird. Ein Pinsel ist dagegen eine Pixmap, die mehrfach nebeneinander in die zu füllenden Fläche gezeichnet wird (vollständige Überdeckung durch eine Füllmuster). Der Pinsel kann beliebige Farben enthalten. Enthält er nur Vordergrund- und

Hintergrundfarbe, dann sind beide Methoden identisch. Die Methode der Überdeckung durch eine Füllmuster ist deshalb die allgemeinere. Es werden aber aus historischen Gründen oft beide Methoden angeboten.

Alle displayorientierten Grafiksysteme bieten Linien, Rechtecke, Polygone und Ellipsen, bzw. Ellipsensegmente (Teilkreise). Eine Besonderheit bietet Win32 mit dem Kreisabschnitt (Chord), der nicht den Kreismittelpunkt enthalten muß. Nur Win32 und Quickdraw kennen Ausgabekommandos für Rechtecke mit abgerundeten Ecken (RoundRect). RoundRect wird besonders häufig für Knöpfe als Dialogelemente verwendet.

Kurven werden nur von Win32 und Postscript angeboten. Beide Systeme verwenden kubische Bézier-Kurven mit vier Kontrollpunkten je Abschnitt.

2.4.1.2. Pixmapausgaben

Für die Ausgabe von Grafikprimitiven sind nicht nur die Primitive und deren Parameter interessant, sondern auch im Ausgabekommando referenzierte Datenstrukturen und Objekte. Die referenzierten Objekte werden von Anwendungsprogrammen vor der aktuellen Ausgabe erzeugt. Sie sind als Teil des jeweiligen Ausgabekommandos zu betrachten und damit Teil des Ausgabestroms. Wird der Ausgabestrom über Netzwerke übertragen, auf Massenspeicher aufgezeichnet oder in ein anderes Format konvertiert, so müssen auch diese referenzierten Objekte bearbeitet werden.

Datenstruktur

Der wichtigste referenzierte Objekttyp ist die Pixmap (siehe Kapitel 2.2.3.3). Die Pixmap wird durch eine Datenstruktur beschrieben. Die Datenstruktur besteht immer aus zwei Teilen: einem Header, der Anordnung und Bedeutung einer Menge von Pixelwerten im Speicher zu beschreibt und den Pixeldaten. Tabelle *Pixmapformate* (siehe Anhang *Pixmapausgaben*) zeigt die Komponenten von Pixmaps in mehreren Grafiksystemen. Zu den Komponenten gehören Informationen über die Pixeldaten, wie die

- Basisadresse, bzw. die Pixeldaten selbst (1),
- Anordnung von Pixelgruppen, bzw. Scanlines, im Speicher (2),
- Dimensionen (3)

und Informationen über einzelne Pixel, wie

- Zahl (4) und
- Anordnung, bzw. Gewicht, (5) der Farbkomponenten.

Statt der Adresse der Pixelwerte im Speicher können die Pixelwerte auch Teil der Pixmap Datenstruktur sein. Zu Pixmaps, die keine Pixel in Echtfarbdarstellung enthalten, gehört eine Farbtabelle, die den Indexwerten Farbwerte zuordnet. Die Farbtabelle besteht im wesentlichen aus

- einer Angabe über die Länge der Tabelle (6) und
- den Echtfarbwerten (7).

Alle Angaben sind notwendig, um eine Pixmap zu dekodieren. Dies gilt für alle Grafiksysteme. Fehlen Angaben in der Pixmap, so sind die Werte durch eine Vereinbarung zwischen Kodierer und Dekodierer eindeutig für alle Pixmaps eines Grafikstroms, bzw. eines Grafiksystems festgelegt.

Ausgabekommando

Pixmapausgabekommandos enthalten neben der Pixmap, bzw. einem Verweis auf die Pixmap,

- den Zielbereich und
- einen Verweis auf den gültigen Grafikkontext oder einzelne Kontextparameter.

Der Zielbereich, d.h. der betroffene Bereich des Ausgabefensters liegt entweder als Aufhängepunkt vor oder als Rechteck. Mit einem Rechteck ist die Möglichkeit der Skalierung beim Ausgabevorgang verbunden (siehe Tabelle *Modifikationsparameter*).

Tabelle *Pixmapausgabekommandos* (siehe Anhang *Pixmapausgaben*) zeigt Pixmapausgabekommandos mehrerer Grafiksysteme. Die Ähnlichkeiten sind, wie schon in Tabelle *Pixmapformate* (Anhang *Pixmapausgaben*), deutlich zu erkennen, obwohl sehr unterschiedliche Grafiksysteme verglichen werden.

An der Syntax des GIF-Grafikstroms für Pixmaps (es gibt auch Textausgabe, siehe Tabelle *Grafikprimitive*) sind die wichtigsten Komponenten von Pixmapausgaben sehr deutlich zu sehen. Es sind alle wesentlichen Elemente, aber eben nur diese, in einfacher Form vorhanden. Die Pixmap (Image) enthält Dimensionen (ImageWidth und ImageHeight), 2 verschiedene Anordnungen von Scanlines (InterlaceFlag) im Speicher, eine optionale Farbtabelle und die Pixeldaten. Das zugehörige Ausgabekommando wird im Ausgabestrom durch den Primitivtyp (ImageSeparator) identifiziert. Es enthält neben der Pixmap nur die Zielkoordinaten. Eine Clippingmaske ist durch die Angabe einer als transparent betrachteten Farbe (TransparencyIndex) durch die Pixeldaten gegeben. Der Verweis auf die Clippingmaske, das Feld TransparencyIndex, wird bei Bedarf vor dem Ausgabekommando im Grafikkontext gesetzt.

2.4.1.3. Pfade und Bereiche

Pfade sind offene oder geschlossene Linien beliebiger Form. Die umschlossene Fläche eines Pfades wird als Bereich oder Gebiet (engl. Region) bezeichnet. Pfade und Bereiche werden durch Kombination anderer Grafikprimitive definiert. Win32, Postscript und sogar Windows NT Treiber kennen Pfade. Quickdraw bietet keine Pfade, aber Bereiche. In diesem Fall wird ein Bereich über die Kombination der Flächen von Grafikprimitiven definiert. Durch das Grafikkommando FrameRgn kann aber der Rand eines Bereichs gezeichnet werden, was der Ausgabe eines Pfades in anderen Grafiksystemen entspricht.

Bei der Kombination von Grafikprimitiven zu Pfaden und Bereichen gibt es verschiedene Arten den Inhalt zu definieren. Eine ausführliche Diskussion ist in [Postscript] zu finden. Die beiden wichtigsten Regeln zur Bestimmung des Inhalts von Pfaden sind:

1. Even-odd (Postscript) auch ALTERNATE genannt (Win32)
2. Winding (Postscript und Win32).

Konstruktion von Pfaden und Bereichen

Grafiksysteme verwenden zur Erzeugung eines Pfads oder Bereichs eine besondere Betriebsart in der die Grafikengine Ausgabekommandos nur aufzeichnet und akkumuliert, aber nicht ausgibt. Im allgemeinen benutzen Anwendungsprogramme Verwaltungsfunktionen der Grafiktoolbox, um Bereiche (bzw. Datenstrukturen, die den Bereich repräsentieren) zu erzeugen. Bei Quickdraw dient dazu der Befehl *OpenRgn* (Win32: *BeginPath*), ab dem alle Ausgabebefehle (bis *CloseRgn*, Win32: *EndPath*) zur Definition der Region verwendet. In Postscript ist dies die Standardbetriebsart. In Postscript werden Linien und einfache Formen immer in Pfade umgewandelt, bevor sie ausgegeben werden. Die Ausnahme sind die Kommandos Rectstroke und Rectfill in Display Postscript, wo dies implizit geschieht.

Beispiele

Win32 verwendet bitmapbasierte und pfadbasierte Bereiche. Quickdraw kennt, wie in Kapitel 2.2.3.6 beschrieben, nur komprimierte Bitmaps als Bereiche. Postscript Bereiche sind nur durch Pfade definiert. In allen vier Grafiksystemen können Bereiche sowohl als Grafikprimitiv, als auch zum Clipping verwendet werden. MPEG kennt keine Bereiche. GIF und X11 verwenden Bereiche nur zum Clipping, nicht aber als Grafikprimitiv. X11-Bereiche werden durch Bitmaps oder als Rechtecklisten definiert. GIF verwendet eine Schlüsselfarbe (Keycolor) zur Definition von Clippingbereichen auf der Basis von Bitmaps.

Bezüglich der Einfärbung von Rand und Inhalt verhalten sich Pfade genauso, wie einfache Formen (siehe oben). Sind Bereiche nur als Bitmaps definiert, dann kann die Möglichkeit zum Zeichnen des Randes entfallen (Bsp: X11, aber nicht Quickdraw).

Eigenschaften von Pfaden und Bereichen in verschiedenen Grafiksystemen

Die Tabelle faßt Darstellungen und Verwendungszweck von Bereichen an der jeweiligen Schnittstelle zusammen:

Grafiksystem	X11[1]	Win32	Quick-draw	Win32 (NT) Treiber	Quick-draw Treiber	MPEG	GIF	Post-script
Darstellung:								
Bitmap	x	x	x	x	x		x	
Rechteckliste	x	x	x	x	x			
Pfad		x		x				x
Funktion:								
Rand Zeichnen[2]		x	x	x	x			x
Inhalt Zeichnen	x	x	x	x	x			x
Clipping	x	x	x	x	x		x	x

1 X11 kennt keine Pfade und Bereiche. Für das Clipping werden Bitmaps verwendet. Die hier aufgeführten Eigenschaften beziehen sich auf die sogenannte clip-mask des X11 Grafikkontextes.

2 Über einen Umweg kann der Inhalt von Bereichen kann natürlich immer gezeichnet werden, indem der Bereich in eine Bitmap umgewandelt wird (siehe die Übersetzung von Quickdraw Bereichen nach X11 in Kapitel 5.3).

Tabelle Darstellung und Funktion von Pfaden und Bereichen

2.4.1.4. Text

Sowohl pfadbasierte, als auch bitmapbasierte Zeichensätze unterscheiden sich bei gleichen Zeichensatzbeschreibungen (Font, Größe) oft auf verschiedenen Grafiksystemen. Abbildung *Lauflängen-unterschiede 1* zeigt für einen Beispieltext die pixelweise Differenz der am meisten verwendeten Zeichensätze (Times/12p), in den Versionen, die zur Standardausstattung von Quickdraw und Windows gehören.

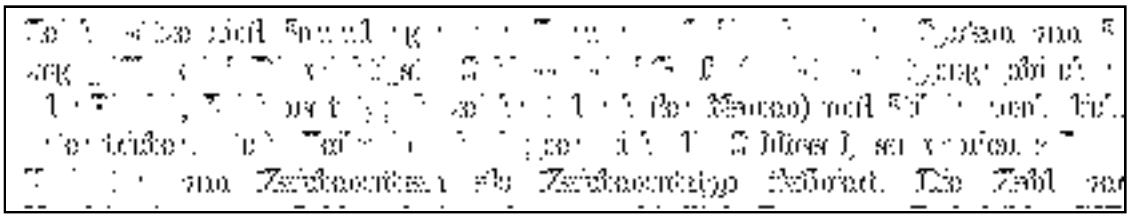


Abbildung Lauflängenunterschiede 1

Differenzbild: Ein Textstück in Times/12p aus der Quickdraw Standardausstattung wurde vom gleichen Text mit Times/12 von MS Windows subtrahiert. Es bleiben die unterschiedlichen Pixel. Die Lauflängen unterscheiden sich vor allem bei längeren Zeichenketten, also auf der rechten Seite der Abbildung. Die Pixel beider Zeichensätze liegen nebeneinander und nicht deckend, so daß im Differenzbild vor allem auf der rechten Seite vertikale Strukturen entstehen.

Die Unterschiede betreffen sowohl die Form, als auch die Laufweite der einzelnen Zeichen. Unterschiedliche Laufweiten führen dazu, daß Texte ineinander laufen oder übermäßig große Wortabstände entstehen. Dies gilt sowohl für abweichende Längen bei gleichen Zeichensätzen, als auch für Ersetzungen bei fehlenden Zeichensätzen.

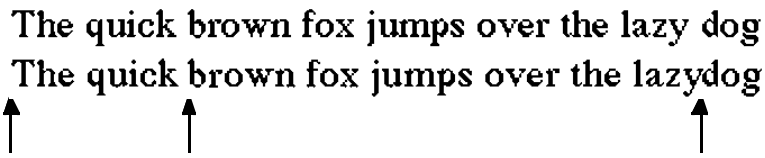


Abbildung Lauflängenunterschiede 2

Der Fehler durch Lauflängen entsteht vor allem dann, wenn sich minimale Abweichungen über längere Zeichenketten akkumulieren. Der Anfang jeder Zeichenkette wird richtig positioniert, durch abweichende Längen ist aber das Ende und damit der Abstand zur nächsten, wieder absolut positionierten, Zeichenkette verschieden. Die Aufteilung der Ausgaben in einzelne Zeichenketten ist applikationsspezifisch und fallspezifisch. Die Lauflängen der zwei hier gezeigten Zeichensätze unterscheiden sich nur gering. Bei der ersten kurzen Sequenz 'The quick' spielt der Unterschied fast keine Rolle. Längere Zeichenketten laufen aber ineinander. Die Pfeile zeigen auf die horizontalen Startpositionen der drei Zeichenketten, die im Quellsystem, wie eine fortlaufende Zeichenkette aussehen.

Man kann die Grafiksysteme bezüglich ihrer Textausgaben in drei Gruppen einteilen.

1. Grafiksysteme ohne oder mit minimaler Textausgabe:
   Videoorientierte Systeme, wie MPEG und GIF verlassen sich fast ausschließlich auf Pixmapausgaben. GIF bietet minimale Textausgaben.
2. Grafiksysteme mit selbständiger Textausgabe:
   Zu dieser Gruppe gehören vor allem die displayorientierten Grafiksysteme X11, Quickdraw, Win32, WinNT Treiber. Diese Grafiksysteme bieten eine Reihe von Funktionen, die speziell auf die Ausgabe von Text zugeschnitten sind.
3. Grafiksysteme mit abgeleiteter Textausgabe:
   Postscript behandelt Text, wie alle anderen Grafikprimitive. Text wird in einen Pfad umgewandelt und durch Pfadmanipulation und Pfadausgaben weiterverarbeitet.

Alle Grafiksysteme mit Textausgabemöglichkeiten bieten Funktionen für die Ausgabe von Zeichenketten und nicht nur für einzelne Zeichen. Diese Funktionen ändern auf manchen Grafiksystemen die Laufweite einer Zeichenkette, indem sie die Zeichenabstände entsprechend der auszugebenden Zeichen anpaßt und zusätzlich eventuell Ligaturen und Kerning einführen Das führt dazu, daß die Länge einer Zeichenkette nicht nur von der Länge der einzelnen Zeichen, sondern auch von anderen Faktoren abhängt. Siehe dazu auch Kapitel 5.3.3.6.

Neben der einfachen Ausgabe von Zeichenketten bieten einige Grafiksysteme der zweiten Gruppe erweiterte Funktionen, wie das Füllen der Zeichen mit Mustern oder Rotation, bzw. Transformation (Win32). Bei Grafiksystemen der dritten Gruppe können alle Modifikationen, die auf Grafikausgaben angewendet werden können, natürlich auch auf Text angewendet werden.

2.4.2. Vergleich von Kontextparametern

Wie in Kapitel 2.2.2.3 beschrieben, ist ein Grafikkontext eine Sammlung von Parametern, die Grafikausgaben beeinflussen. In den folgenden Abschnitten werden Beziehungen zwischen Modifikationsparametern und Kontextparametern in verschiedenen Grafiksystemen basierend auf Funktionsgruppen diskutiert. Die Unterschiede zwischen verschiedenen Grafiksystemen spielen eine wesentliche Rolle für die in Kapitel 5.3.3.3 beschriebenen Übersetzung von Pixmapausgaben unter Anwendung spezieller Transfermodi.

2.4.2.1. Transfermodus

Der Transfermodus bestimmt die Verknüpfung der Pixel der Pixmap mit den bereits existierenden Pixeln des Ausgabefensters. Dies betrifft neben der Ausgabe, bzw. dem Kopieren von Pixmaps, auch das Füllen einer Fläche mit einem Pinsel ein.

Der Transfermodus ist oft Teil des Grafikkontextes. Manche Grafiksysteme führen Pixmapausgaben aber ohne Grafikkontext aus. In diesem Fall wird der Transfermodus im Ausgabekommando explizit angegeben. Die möglichen Einstellungen für den Transfermodus unterscheiden sich deutlich bei verschiedenen Grafiksystemen. Nur die acht logischen Verknüpfungen sind in allen displayorientierten

Grafiksystemen vertreten. Da der Transfermodus aber ein sehr wichtiger Parameter ist, sollen hier auf die wichtigsten Fälle eingegangen werden:

Quickdraw und Quickdraw Treiber

Der Transfermodus wird bei der Pixmapausgabe explizit angegeben. Es gibt acht logische und acht sog. arithmetische Transfermodi. Die logischen Transfermodi sind bool'sche Verknüpfungen zwischen den Pixeln des Quellbereichs und des Zielbereichs.

Zu den wichtigsten logischen Modi gehören:

- srcCopy: Kopieren ohne Verknüpfung (Quelle -> Ziel),
- srcOr: Irreversible Kombination von Quellbereich und Zielbereich und
- srcXOr: Reversible Kombination.

Die anderen logischen Transfermodi notSrcOr, srcXor, notSrcXor, srcBic und notSrcBic (siehe [Quickdraw94] S. 4-33 ff.) werden seltener benutzt (Messung *Transfermodi*). Bool'sche Operationen bewirken jedoch oft unerwartete Ergebnisse, wenn sie auf beliebige Farbwerte oder Farbindexwerte angewendet werden. In vielen Fällen sind Resultate nur für spezielle Farben (z.B. weiß als Hintergrundfarbe) definiert. Die arithmetischen Transfermodi kommen dem menschlichen Farbempfinden eher entgegen:

- addPin: In allen drei Farbkanälen jeweils die Summe begrenzt durch eine obere Schranke (rgbOpColor im grafVars Feld des CGrafport).
- adMax: Übernimmt in drei Farbkanälen jeweils den intensiveren Wert beider Pixel in den Zielbereich.
- transparent: Kopieren ohne Verknüpfung aber maskiert durch die Hintergrundfarbe als Keycolor (siehe GIF: Transparency-Color).
- blend: Gewichteter Mittelwert von Quellpixel und Zielpixel. Verwendet für Einblendeffekte.

Die anderen arithmetischen Transfermodi sind addPin, addOver, subPin, subOver, adMin. Den transparent-Modus kann man sich als die an das Farbempfinden angepaßte Überlagerung von Quellbereich und Zielbereich (vorher srcOr) vorstellen. Die Interpretation des addPin-Modus ist eine additive Farbmischung mit Sättigung. Die Berechnung der arithmetischen Modi ist aufwendiger, als die der logischen Modi. Dies gilt vor allem dann, wenn indizierte Farben vorliegen, die in RGB Werte abgebildet und nach der Verknüpfung wieder in Indexwerte umgewandelt werden müssen.

Der Transfermodus kann in einer besonderen Betriebsart auch zur Hervorhebung (Highlighting) verwendet werden. In diesem Fall wird statt der Vordergrundfarbe der Wert des Feldes rgbHiliteColor (grafVars Feld des CGrafport) eingesetzt. Die Betriebsart Highlighting wird durch einen bestimmten Wertebereich der Moduswerte ausgewählt.

Quickdraw kopiert zwischen Pixmaps beliebiger Farbtiefe. Dabei wird zur Farbreduzierung Dithering automatisch eingesetzt [FoDaFeHu91]. Diese Eigenschaft kann auch zur Konvertierung von Pixmaps verwendet werden (siehe Kapitel 5.2.2).

X11

X11 bietet 16 Transfermodi. Dies ist der vollständige Satz möglicher logischer Verknüpfungen zweier bool'scher Werte. Sie umfassen die acht von Quickdraw bekannten logischen Modi. Die anderen acht Modi werden äußerst selten verwendet (Messung *Transfermodi*). Alle Transfermodi sind bool'sche Operationen der Pixelwerte. Durch die 'plane-mask' können einzelne Bitebenen (planes) von der Verknüpfung ausgeschlossen werden. Dies wirkt wie eine Vorverarbeitung der Pixmap durch die Funktion: Pixel UND plane-mask; ähnlich den dreikomponentigen Win32 ROP2-Codes mit einer homogenen dritten Pixmap, die durch die plane-mask gegeben ist. Der Transfermodus ist Teil des Grafikkontextes (Komponente 'function' des GC).

Win32

Die Windows Grafikengine verwendet einen Funktionscompiler. Der Transfermodus dient dabei als Parameter für den Funktionscompiler. Zur Laufzeit wird aus dem Transfermodus ausführbarer Code für die Verknüpfung erzeugt. Für häufig verwendete Parameterkombinationen ist der Programmcode aber vermutlich ständig verfügbar.

Es gibt zwei Sätze von Verknüpfungen, die durch sogenannte ROP-Codes (Raster Operation) spezifiziert werden:

- Der Basissatz der Modi, die wie Quickdraw mit zwei Pixelwerten (Zielbereich und Pinsel) je Verknüpfung arbeiten. Im Basissatz gibt es 16 ROP-Codes, wie in X11.
- Die erweiterten ROP-Codes. Sie arbeiten mit drei Pixelwerten. Zusätzlich zu Pinsel und Zielbereich gehen auch die Pixel eines Quellbereichs ein. Es gibt 256 ROP-Codes, von denen aber nur ein kleiner Teil tatsächlich verwendet wird. 15 der ROP-Codes werden in der Dokumentation und erwähnt und sind als Konstanten definiert. Auch diese erweiterten Modi sind bool'sche Operationen auf Pixelwerten.

Der Transfermodus ist Teil des Grafikkontextes (Komponente ROP2 des DC).

GIF

Bei GIF werden Pixel immer ohne Verknüpfung in den Zielbereich kopiert (entsprechend Quickdraw: srcCopy). Es wird deshalb kein Transfermodus angegeben.

MPEG Videostrom

Der Transfermodus richtet sich nach dem Macroblocktyp.

Es gibt zwei Transfermodi:

- Bei sog. intra-kodierten Macroblöcken werden die Pixel kopiert (entsprechend srcCopy bei Quickdraw)
- Sog. predictive-coded Macroblocks in P- und B-Frames werden zum Zielbereich addiert entsprechend addPin von Quickdraw ohne künstliche obere Schranke.
  Vor der Verknüpfung wird der Zielbereich unter Umständen durch Kopieren oder Interpolieren modifiziert. Diese Modifikation wird durch Bewegungsvektoren gesteuert, nicht durch den Transfermodus.

Windows NT Treiber

siehe Win32.

Postscript

Postscript arbeitet (logisch) nicht auf Pixmaps. Es gibt deshalb keinen Transfermodus, der Pixel verknüpft. Die Halbton-Funktion bietet ähnliche Möglichkeiten für Überblendeffekte.

2.4.2.2. Koordinatenparameter

Zwischen den Grafiksystemen gibt es wesentliche Unterschiede in der Behandlung von Koordinaten. Grafiksysteme unterscheiden sich vor allem in ihren Möglichkeiten im Bezug auf Koordinatentransformation. Einige Grafiksysteme bieten flexible Formen von virtuellen bzw. geräteunabhängigen Koordinaten, andere nur Pixelkoordinaten mit festem Koordinatenursprung. In Tabelle *Koordinatenparameter* sind die verschiedenen Systeme gegenübergestellt.

Die Koordinatenparameter von Grafikausgaben setzen sich zusammen aus

- aktuellen Koordinatenparametern, die meistens im Ausgabekommando mitgeführt werden,
- explizite Koordinatensystemparameter, die in Kontextmanipulationskommandos gesetzt werden und
- implizite Koordinatensystemparameter.

Koordinatensystemparameter bestimmen die Orientierung, Skalierung und den Koordinatenursprung. Zum Teil sind diese Parameter implizit durch die Spezifikation der Schnittstelle gegeben und werden nie explizit angegeben. Bei X11 ist z.B. das Koordinatensystem immer fest in Pixelkoordinaten vorgegeben während bei Win32 die Skalierung und der Koordinatenursprung im Ausgabestrom angegeben werden.

Koordinaten in Ausgabekommandos beziehen sich immer auf das Koordinatensystem des Ausgabebereichs (z.B. Fenster). Fenster werden, wie in Kapitel 2.3.3.3 beschrieben, nicht nur als Ausgabebereich, sondern auch zur Strukturierung der Benutzerschnittstelle eingesetzt werden. Unter Umständen existieren deshalb viele (Unter-)Fenster innerhalb eines Ausgabebereichs. Die Grafikausgaben, die für Unterfenster bestimmt sind beziehen sich auf die Koordinatensysteme der Unterfenster. Die Koordinaten dieser Grafikausgaben können mit der Position des Unterfensters und dem Koordinatensystem der übergeordneten Fenster in Koordinaten bezüglich des Ausgabebereichs umgewandelt werden (Kapitel 5.3.3.7).

Grafiksystem	Eigenschaften				
	Pixel-koordinaten	virtuelle / geräte-unabhängige Koordinaten	Gebrochen-zahlige Koordinaten	Ursprung zur Laufzeit verschiebbar	Ursprung fest vorgegeben und nicht veränderbar
X11	x				x[1]
Win32	x	x		x	
Quickdraw	x			x	
Win32 (NT) Treiber	x				x[1]
Quickdraw Treiber	x			x	
MPEG	x				x[1]
GIF	x				x[1]
Postscript	2	x	x	x	

1 Linke obere Ecke des Ausgabebereichs ist (0, 0).

2 Mit einer fest eingestellten Auflösung, die der von Bildschirmen entspricht, können Pixelkoordinaten simuliert werden.

Tabelle Koordinatenparameter

Alle Grafiksysteme bieten Pixelkoordinaten, aber nur wenige geräteunabhängige Koordinaten. Bemerkenswert ist die Tatsache, daß der Ursprung bei allen Systemen, wo er fest vorgegeben ist, in der linken oberen Ecke des Ausgabebereichs liegt.

2.4.2.3. Füllmuster

Füllmuster (bzw. Pinsel) sind Pixmaps, die wie eine Füllfarbe zum Füllen von Flächen verwendet werden. Das Muster ist sehr oft kleiner als die zu füllende Fläche und wird deshalb kachelartig wiederholt. Außer der kachelartigen Wiederholung beim Zeichnen in den Zielbereich gibt es keine Unterschiede zwischen der Ausgabe einer Pixmap und dem Füllen mit einem Füllmuster.

2.5. Der Grafikausgabestrom

Im vorangegangenen Kapitel wurden die Komponenten von Grafikkommandos mit Beispielen aus verschiedenen Systemen beschrieben. Ein Grafikstrom besteht aus einer Folge Grafikkommandos, bzw. Grafikausgaben. Jede einzelne Grafikausgabe eines Grafikstroms wird bestimmt durch einen vollständigen Parametersatz, d.h. durch Grafikprimitiv, Modifikationsparameter und Kontextparameter (Kapitel 2.2.2). Ein Grafikstrom ist deshalb prinzipiell eine Folge vollständiger Parametersätze. Tatsächlich wird aber in fast allen Grafiksystemen zwischen dem Erzeuger eines Grafikstroms und dem Darsteller nicht für jede Grafikausgabe der vollständige Parametersatz übermittelt. Dieses Kapitel geht deshalb genauer auf reale Grafikströme ein.

Prinzipiell kann ein Ausgabekommando durch Angabe aller Parameter dargestellt werden. Tatsächlich existieren aber in den meisten Grafiksystemen so viele Parameter, daß diese Methode nicht praktikabel ist. Deshalb wurde bei realen Grafiksystemen eine Unterscheidung der Parameter in Modifikationsparameter und Kontextparameter vorgenommen (siehe Kapitel 2.2.2).

Der Grafikkontext umfaßt die Parameter, die beim Entwurf der jeweiligen Schnittstelle als relativ statisch angesehen werden, während die Parameter, die sich oft ändern, den Modifikationsparametern zugeschlagen werden. Modifikationsparameter werden bei Ausgaben immer angegeben, Kontextparameter dagegen nur wenn sie sich ändern. Es gibt deshalb zwei verschiedene Typen von Grafikkommandos:

1. Kontextmanipulation und
2. Ausgabekommando.

2.5.1. Kontextmanipulation

Kommandos zur Kontextmanipulation produzieren selbst keine Ausgabe. Sie dienen der Vorbereitung von Ausgabekommandos. Kontextmanipulationskommandos setzen Parameter, die bei den folgenden Ausgabekommandos mitwirken. Beispiele für Kontextmanipulation sind das Setzen einer Stiftfarbe in displayorientierten Grafiksystemen, des Quantisierungsfaktors in MPEG oder der transparenten Farbe (TransparencyIndex) in GIF. Alle Parameter des Grafikkontextes eines Grafiksystems können auf diese Art direkt oder indirekt gesetzt werden. Entsprechend der großen Zahl von Kontextparametern gibt es in den meisten Grafiksystemen auch eine große Zahl verschiedener Kommandos zur Manipulation. Sie sollen hier nicht einzeln aufgeführt werden, da sie sich mit Hilfe der jeweiligen Dokumentation leicht aus der Aufstellung der Kontextparameter (Kapitel 2.4.2) ableiten lassen.

Neben den Parametern des Grafikkontextes bilden alle im Endgerät vorhandenen oder angelegten Ressourcen (siehe Kapitel 2.2.3) den erweiterten Grafikkontext. Zum erweiterten Grafikkontext gehören die in Tabelle *Ressourcetypen* genannten Ressourcen: Clipping Bereiche, Farben und Farbtabellen, Zeichensätze, Stifte mit Form, Farbe und/oder Muster sowie Linienköpfe und Eckpunkte von Polygonzügen. Die meisten dieser Ressourcen werden im Grafikkontext referenziert (z.B. "clipRgn" und "txFont" im Quickdraw Grafport). Andere, wie die Farbpalette des X Window Systems, werden indirekt über die Fensterzuordnung referenziert.

Auch die Verwaltung der Ressourcen gehört zur Kontextmanipulation. Während die Komponenten des Grafikkontextes nur mit Werten beschrieben werden, werden Ressourcen angelegt, bearbeitet, abgefragt oder reserviert. Die Art der Operation hängt natürlich vom Grafiksystem ab. So wird z.B. die Verfügbarkeit von Zeichensätzen abgefragt (X11: ListFonts, QueryFonts), Farbtabelleneinträge reserviert und Pixmaps in den meisten Grafiksystemen unsichtbar (offscreen) bearbeitet.

2.5.2. Ausgabekommando

Ausgabekommandos haben zwei Funktionen. Sie führen einerseits die variablen Parameter (Grafikprimitiv und Modifikationsparameter) mit sich. Andererseits lösen sie die Ausgabe aus, d.h. sie veranlassen das Endgerät, bzw. die Grafikengine zu einer sichtbaren Aktion. Man kann sich Ausgabekommandos in zwei getrennte Kommandos aufgeteilt vorstellen: Der erste Teil setzt, wie ein Kontextmanipulationskommando, die variablen Parameter. Damit ist der gesamte Parametersatz für die Ausgabe eingestellt. Der zweite Teil dient als Signal an das Endgerät, eine Ausgabe mit den aktuellen Parametern durchzuführen (Ausgabesignal). Der Grund dafür, daß in fast allen Grafiksystemen beide Funktionen in einem Kommando vereinigt sind liegt darin, daß sie fast immer direkt nacheinander auftreten. Selbst wenn der Grafikkontext unverändert bleibt, müssen bei jeder Ausgabe stets variable Parameter, wie z.B. Koordinaten, geändert werden, bevor die nächste Ausgabe stattfindet.

2.5.3. Parameter und Ausgabesignal

Man kann alle Grafiksysteme bezüglich des Grads der Trennung von Parametermanipulation und Ausgabesignal einordnen:

- Am einen Ende der Skala stehen die videoorientierten Grafiksysteme (MPEG, GIF). Diese Grafiksysteme haben sehr wenige Kontextparameter. Sie führen fast alle Parameter als variable Parameter im Ausgabekommando (MPEG: Macroblock, GIF: Image, PlainText) mit. Der Großteil des vollständigen Parametersatzes ist deshalb im Ausgabekommando enthalten.
- Displayorientierte Grafiksysteme, wie X11, Win32 und Quickdraw verfügen über Grafikkontexte mit vielen Parametern, die durch Kontextmanipulationskommandos gesetzt werden. Nur ein relativ kleiner, vom Grafikprimitiv abhängiger Satz von Modifikationsparametern ist im Ausgabekommando enthalten.
- Das andere Ende der Skala bilden Grafiksysteme, die Parametermanipulation fast vollständig vom Ausgabesignal trennen. Ein Beispiel dafür sind Postscript-basierte Grafiksysteme. In diesen werden fast alle Parameter einschließlich des Grafikprimitivs und der Koordinaten unabhängig vom Ausgabesignal übertragen. Das Ausgabesignal führt in Postscript nur einen einzigen Modifikationsparameter mit sich, der zwischen Umrahmen und Füllen eines Pfades unterscheidet (stroke und fill).

Für die Verarbeitung in Application Sharing Systemen wird der Ausgabestrom neu organisiert. Alle Parameter, ob durch Kontextmanipulation, Ressourceverwaltung oder im Ausgabekommando gesetzt, werden im erweiterten Parametersatz zusammengefaßt. Jede Ausgabe wird durch diesen Parametersatz vollständig beschrieben. Übrig bleibt nur noch das Ausgabesignal, mit dem die Verarbeitung des Parametersatzes angestoßen wird.

Diese Reorganisation hat zwei Gründe:

1. die Implementierung des Übersetzers wird erleichtert (siehe Kapitel 5.3.4) und
2. die Anwendung selektiver temporaler Kompression wird erleichtert (siehe Kapitel 4.3.3.1).

Tabelle *Kommandos und Parametersätze* zeigt die relevanten Parameter einer kurzen Sequenz aus 7 Kontextmanipulationskommandos und 5 Ausgabekommandos.

	1	2	3	4	5
Kommandos im Grafikstrom	SetColor (BLUE) SetWidth (2) DrawLine (10,10,19,10)	SetColor (WHITE) FillRect (20,10,30,20)	SetColor (RED) SetWidth (1) FrameRect (20,10,30,20)	CopyBits (0,0, 15,15, 22,12, 29,19, srcCopy, Pixmap)	SetColor (BLUE) SetWidth (2) DrawLine (31,10,40,10)
Parametersatz Nur relevante Parameter	primitiv: LINE width: 2 color: BLUE start: 10,10 end: 19,10	primitiv: RECT variant: FILL color: WHITE start: 20,10 end: 30,10	primitiv: RECT variant: FRAME width: 1 color: RED start: 20,10 end: 30,10	primitiv: PIXMAP mode: srcCopy srcstart: 0,0 srcend: 15,15 dststart: 22,12 dstend: 29,19 pixmap: <Zeiger>	primitiv: LINE width: 2 color: BLUE start: 31,10 end: 40,10

Tabelle Kommandos und Parametersätze

Die obere Zeile zeigt die Beispielsequenz aus 5 Ausgabekommandos mit den zugehörigen vorgeschalteten Kontextmanipulationskommandos. Die untere Zeile zeigt die Parameter, mit denen die Ausgaben der oberen Zeile beschreiben werden. Statt jeweils 2-3 Einzelkommandos der oberen Zeile wird nur der relevante Parametersatz übertragen. Dieser kann, wie in Kapitel 4.3.3.1 beschrieben, noch zusätzlich temporal komprimiert werden. Bei der Pixmapausgabe gehört die Pixmap zum Parametersatz. Die Pixmap wird hier nur durch einen Zeiger repräsentiert.

2.6. Toolkits

Obwohl Unterschiede des semantischen Informationsgehalts von Kodierungsvarianten vernachlässigbar sind, treten doch wesentliche semantische Unterschiede auf, wenn nicht nur Grafiksysteme in Betracht gezogen werden, sondern die darauf aufsetzenden Benutzerschnittstellen (GUI = graphical user interface). GUI-Elemente sind von wesentlich höherer semantischer Qualität, als einzelne Grafikausgaben auf der Ebene des Grafiksystems. Grafische Benutzerschnittstellen bieten Fensterverwaltung, Menüs und andere Dialogelemente, sowie sehr spezialisierte Fenster (Dialogboxen), wie Dateiauswahldialoge. GUI-Elemente werden über die Funktionen der grafischen Benutzerschittstelle (GUI-Toolbox oder Toolkit) verwaltet und durch das Grafiksystem dargestellt. Viele grafische Benutzerschittstellen sind auf bestimmte Grafiksysteme zugeschnitten. Prinzipiell können GUIs aber auf jedem Grafiksystem aufsetzen. Die bekanntesten Beispiele sind:

- Xt, das X Toolkit, Motif, OpenWindows und XView auf X11,
- Macintosh Toolbox, TCL und PowerPlant auf Quickdraw,
- MS Windows und MFC auf Win32 und
- OpenStep auf Display Postscript.

2.6.1. Verzahnung zwischen Toolkit und Grafiksystem

Der Grad der Verzahnung zwischen Toolkit und Grafiksystem ist sehr verschieden. Einige Kombinationen von Toolkit und Grafiksystem, sind sehr sauber voneinander getrennt, wie X11 und seine Toolkits. Die Toolkits auf X11 verwenden ausnahmslos das API von X11, die Xlib, und damit das X Protokoll, um Grafikausgaben auszulösen. Die Macintosh Toolbox setzt auf dem Quickdraw API auf. Teilweise verwendet die Toolbox aber auch die Quickdraw Treiberschnittstelle und umgeht das Quickdraw API (Beispiel: Umschaltung zwischen den Zuständen 'Preview' und 'kein-Preview' im Dateiauswahldialog). Bei Windows NT ist die Funktionalität von Win32 im Systemkern repliziert. Der Grafikkern von Windows NT verwendet nicht das Win32 API (GDI32) sondern eine sehr ähnliche Schnittstelle innerhalb des Win32k (Win32k.dll) [Schöttner96]. Die GUI-Toolbox von Windows NT umgeht also die Programmierschnittstelle von Win32, die von den Anwendungsprogrammen verwendet wird.

Der Grad der Verzahnung zwischen Toolkit und Grafiksystem, sowie die Art der Schnittstelle, die von Toolkits verwendet wird, wirkt sich auf die Analyse des Grafikausgabestroms aus (siehe dazu Kapitel 4.2).

Verwendet ein Toolkit nicht die gleiche Schnittstelle, wie Anwendungsprogramme, sondern systemeigene oder unveröffentlichte Schnittstellen, dann ist die Analyse des Ausgabestroms an der Programmierschnittstelle des Grafiksystems erheblich erschwert, wenn nicht unmöglich. Der Ausgabeanalysator muß dann auf andere Schnittstellen ausweichen.

Indem ein Anwendungsprogramm ein GUI verwendet, gibt es semantisch hochwertige Informationen über seine Benutzerschnittstelle aus. Prinzipiell können diese Informationen zum Endgerät übertragen und dort für die Darstellung der Benutzerschnittstelle benutzt werden. Die Darstellung der Benutzerschnittstelle von Anwendungsprogrammen erfolgt dann auf der Ebene von GUI-Elementen. Diese Ebene ist unabhängig von Grafiksystemen und von Richtlinien über die grafische Gestaltung der Benutzerschnittstelle, die von vielen Toolkits vorgegeben werden. Damit bleibt dem Endgerät Raum für Änderungen in der grafischen Gestaltung. Tatsächlich wird ein ähnlicher Mechanismus von vielen Herstellern von Anwendungsprogrammen eingesetzt, die Programme auf mehreren Grafiksystemen und Betriebssystemplattformen anbieten. Es werden sogenannte GUI-Bibliotheken verwendet, die, je nach Plattform, auf dem jeweiligen GUI-Toolkit oder auf dem Grafiksystem direkt aufsetzen. Dies ermöglicht eine vereinfachte Programmierung auf einem plattformunabhängigen GUI-API. Die GUI-Bibliothek sorgt dabei dafür, daß das Anwendungsprogramm trotzdem den Richtlinien über die grafische Gestaltung der Benutzerschnittstelle der jeweiligen Plattform entspricht.

Neben den oben genannten GUI-Elementen, die von den Toolkits verwaltet werden, verwenden Anwendungsprogramme aber auch die Schnittstelle des jeweiligen Grafiksystems direkt, um Fensterinhalte darzustellen. Aus diesem Grund muß wenigstens ein Teil des Ausgabestroms auf dem semantischen Level des Grafiksystems, d.h. in Form von Grafikausgabekommandos erfaßt, kodiert, und je nach Anwendung, übertragen werden. Da auch die Darstellung von GUI-Elementen über das Grafiksystem erfolgt, liegt der gesamte Ausgabestrom in Form von Grafikausgabekommandos vor.

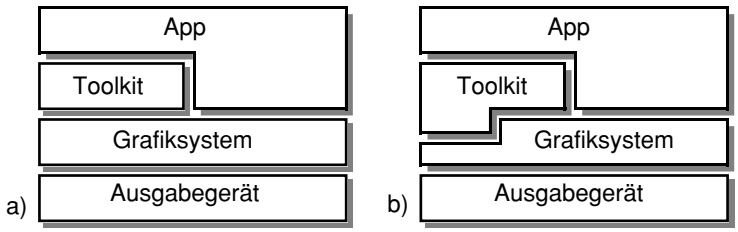


Abbildung Toolkit und Grafiksystem

Anwendungsprogramme verwenden sowohl die GUI-Schnittstelle, als auch die Schnittstelle des Grafiksystems. Wie oben beschrieben setzen manche Toolkits ebenfalls auf dem API des Grafiksystems auf (a). Andere (b) verwenden interne Schnittstellen des Grafiksystems oder eine alternative Schnittstelle statt des APIs, das von den Anwendungsprogrammen benutzt wird.

2.6.2. Sharing von Grafikausgaben vs. GUI-Elementen

Die Übertragung eines Ausgabestroms in Form von Grafikkommandos unter Vernachlässigung der semantischen Qualität der GUI-Elemente stellt im Bezug auf die Anforderungen an das Grafikendgerät eine Vereinfachung und Flexibilisierung dar. Das Grafikendgerät muß nur die Funktionalität eines Grafiksystems bereitstellen. Es muß aber nicht die Funktionalität eines oder mehrerer GUI-Toolkits bereitstellen. Ausgaben aller GUI-Toolkits können auf einem Grafikendgerät dargestellt werden, wenn das GUI-Toolkit das Grafiksystem des Grafikendgeräts verwendet.

Im Rahmen von Application Sharing Systemen hat es sich als nachteilig erwiesen, wenn GUI-Elemente (Knöpfe, Rollbalken, Menüs) auf dem Zielsystem selbständig reagieren, denn dadurch weichen die Darstellungen eines gemeinsam benutzten Programms auf dem Quellsystem und auf den Zielsystemen voneinander ab. Eingaben entfernter Teilnehmer an der Sitzung können so nicht beobachtet werden und das Application Sharing System kann die Eingaben nicht koordinieren (Kapitel 4.4).

Bei Application Sharing Systemen dürfen nur Grafikausgaben übertragen werden, keine aktiven Elemente. Alle Benutzereingaben müssen auf der Ebene elementarer Eingaben and das Anwendungsprogramm übermittelt werden. In der Gegenrichtung soll die Programmreaktion durch Grafikausgaben dargestellt werden. Es werden keine selbständigen Grafikausgaben aktiver Elemente im Zielsystem zugelassen.

Diese Forderung bedeutet, daß die Ausgabeumleitung den Teil des Ausgabestroms, der von GUI-Elementen stammt, erfassen muß.

3. Eingabesysteme

In diesem Kapitel werden die Eigenschaften von Eingabesystemen dargestellt. Das Kapitel dient als Grundlage für die Eingabekonvertierung in Kapitel 6.

3.1. Einleitung

Die Interaktion zwischen Benutzer und Anwendungsprogramm erfolgt über Ausgaben des Anwendungsprogramms und Eingaben des Benutzers. Die Ausgaben von Programmen werden, wie in Kapitel 2 beschrieben, über Grafiksysteme als Ausgabesysteme abgewickelt. Auf der Eingabeseite gibt es auf jedem System, das Interaktion zuläßt, eine Entsprechung: das Eingabesystem.

Eingabesysteme sind im allgemeinen wesentlich weniger komplex, als Grafiksysteme. Sie bieten nur eine geringe Zahl von verschiedenen Eingabetypen im Vergleich zur großen Vielfalt von Grafikprimitiven und Modifikationen, die bei den meisten Grafiksystemen verfügbar sind. Die Überbrückung von Inkompatibilitäten, d.h. die Übersetzung von Benutzereingaben nimmt deshalb in realen Implementierungen von Application Sharing Systemen einen wesentlich geringeren Raum ein, als die Übersetzung der Ausgaben.

Das Eingabesystem (siehe Abbildung *Eingabesystem* und Kapitel 4.5) ist ein Teil der Systemsoftware. Es hat die Aufgabe, Meldungen von Eingabegeräten aufzubereiten und an Anwendungsprogramme weiterzuleiten. Das Eingabesystem definiert dafür Datenstrukturen und Mechanismen durch die diese Datenstrukturen den Anwendungsprogrammen zugänglich gemacht werden.

3.2. Aufbau von Eingabesystemen

Eingabesysteme bestehen, wie Ausgabesysteme aus mehreren Komponenten, die schichtenartig angeordnet sind. Die Komponenten kommunizieren in nur einer Richtung mit einem Nachbarn. Man kann insgesamt fünf Schichten identifizieren, die zur Verarbeitung von Benutzereingaben beitragen: Benutzereingaben werden von den Eingabegeräten aufgenommen, durch Gerätetreiber ausgewertet, anhand des Systemzustand und der Eingabedaten einem Eingabekanal zugeordnet und über eine Warteschlange den Anwendungsprogrammen zugeführt. In den folgenden Abschnitten werden die fünf Schichten und ihre Schnittstellen genauer beschrieben.

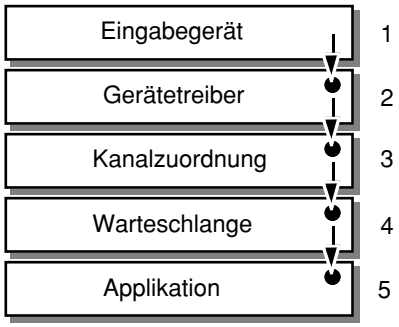


Abbildung Eingabesystem

Benutzereingaben werden von den Eingabegeräten aufgenommen, durch Gerätetreiber ausgewertet, anhand des Systemzustand und der Eingabedaten einem Eingabekanal zugeordnet und über eine Warteschlange den Anwendungprogrammen zugeführt.

3.2.1. Hardwareschicht

Die Hardwareschicht besteht aus den Eingabegeräten. Eingabegeräte liefern, selbständig durch Meldungen (Interrupt) oder auf Anfrage (Polling), Daten über ihren Zustand. Auf allen modernen Plattformen haben sich die gleichen Eingabegeräte als Standardausstattung durchgesetzt. Die wichtigsten - und hier diskutierten - Eingabegeräte sind Tastatur, Mauszeiger und Maustasten.

3.2.2. Treiberschicht

Zu jedem Eingabegerät existiert ein Gerätetreiber. Der Gerätetreiber wertet die hardwarespezifischen Daten der Eingabegeräte aus und erzeugt Meldungen an das Betriebssystem in einem vom System vorgegebenen Format. Dieses Format ist gerätespezifisch, d.h. u. U. unterschiedlich für verschiedene Gerätetypen. Es ist aber einheitlich für alle Geräte gleichen Typs.

Die Meldungen der Treiberschicht an das Betriebssystem werden entweder selbständig durch Aufruf einer Systemfunktion, bzw. eines Software-Interrupts oder auf Anfrage (Polling, z.B. Windows NT 4.0 Tastatureingabe) weitergegeben.

Mit den Eingabegeräten Tastatur, Mauszeiger und Maustasten werden die Eingabemöglichkeiten fast aller Anwendungsprogramme abgedeckt. Unter Mauseingaben werden dabei alle Eingaben von Zeigegeräten zusammengefaßt. Oft bilden andere Zeigegeräte als Mauszeiger zusammen mit den zugehörigen Gerätetreibern sogar den Eingabestrom eines Mauszeigers nach. Sie sind deshalb für die anderen Schichten (3-5) meistens nicht von einem Mauszeiger zu unterscheiden.

3.2.3. Kanalzuordnung

Das Eingabesystem ist dafür verantwortlich, Benutzereingaben an Anwendungsprogramme zu verteilen. Es erzeugt die Zuordnung der Eingaben zu den Eingabekanälen. Die Zuordnung wird entsprechend den absoluten Koordinaten (beim Mauszeiger) und dem Zustand des Eingabefokus (Kapitel 2.3.5) der grafischen Oberfläche, vor allem des Window Managers, durchgeführt. Alle Eingabemeldungen werden durch den Eingabekanal gekennzeichnet und in die für den Eingabekanal zuständige Warteschlange eingefügt.

Eingaben laufen ohne Beteiligung von Anwendungsprogrammen bis zur Schicht 3. In Schicht 3 sucht das Betriebssystem einen Empfänger für die Benutzereingaben. Auf allen hier diskutierten Plattformen müssen sich Anwendungsprogramme zur Annahme von Benutzereingaben (und anderen Meldungen) bereit erklären. Sie tun dies indem sie einen oder mehrere Eingabekanäle einrichten. Mit der Einrichtung eines Eingabekanals erhalten Programm und System eine gemeinsame Referenz über die Daten ausgetauscht werden können.

Der Begriff 'Eingabekanal' ist eine Abstraktion, um die Vorgänge auf verschiedenen Plattformen zu systematisieren. Tatsächlich spielen sich auf allen Plattformen sehr ähnliche Vorgänge ab. Bei MacOS 7/8 gibt es nur einen Eingabekanal je Prozeß. Der Eingabekanal wird automatisch vom System angelegt wird. Anfragen von Anwendungsprogrammen nach Meldungen enthalten deshalb nicht die Referenznummer des Eingabekanals. Im Fall von MS Windows und X11 sind die Fensterreferenzen identisch mit Eingabekanälen. Es existieren auf diesen Plattformen bei den meisten Programmen viele Fenster, die als Eingabekanäle dienen können.

3.2.4. Warteschlangen

Eingabekanäle werden vom Eingabesystem auf Meldungswarteschlangen abgebildet. Dabei kann eine Meldungswarteschlange Meldungen von mehreren Eingabekanälen enthalten. Meldungen über Benutzereingaben werden Anwendungsprogrammen sequentiell zugestellt. Die Warteschlange puffert dabei Meldungen bis das Anwendungsprogramm die Meldungen entgegennimmt. In einer Warteschlange können Eingabemeldungen mehrerer Eingabekanäle zusammengefaßt sein. Typischerweise existiert nur eine systemweite Warteschlange für Eingabemeldungen oder eine für jeden Prozeß. Diese enthält dann z.B. im Fall von MS Windows Eingabemeldungen aller Eingabekanäle (Fenster) eines Threads.

Neben Benutzereingaben wird der Mechanismus der Meldungswarteschlangen auch für andere Meldungen verwendet. Die wichtigsten Bereiche sind dabei Interprozeßkommunikation (IPC),

Systemmeldungen und Netzwerkkommunikation. Diese zusätzliche Nutzung des Eingabemechanismus ist normalerweise für Benutzereingaben transparent, d.h. unsichtbar für Benutzereingaben. Im Fehlerfall können aber z.B. Netzwerkmeldungen die Meldungswarteschlangen blockieren und sich damit auf die Verarbeitung von Benutzereingaben auswirken.

3.2.5. Anwendungsschicht

Alle Eingabesysteme verwenden Datenstrukturen als Eingabemeldungen. Die Datenstrukturen werden an logische Komponenten von Anwendungsprogrammen geschickt. Mit der Zustellung einer Eingabemeldungen erhält ein Programm Zugriff auf die in einer Meldung enthaltenen Daten. Dabei gibt es zwei verschiedene Zustellungsmechanismen. Die am weitesten verbreitete Methode ist das blockierende Warten des Anwendungsprogramms auf eine Benutzereingabe über eine Funktion des Eingabesystems (MacOS: WaitNextEvent, MS Windows: GetMessage, X11: XNextEvent). Das Anwendungsprogramm erhält als Ergebnis der Funktion einen Zeiger auf die Daten einer Meldung. Die zweite Methode ist die Entgegennahme der Eingabemeldung durch eine Funktion des Anwendungsprogramms. In diesem Fall fragt nicht das Programm nach neuen Eingaben, sondern es registriert beim Eingabesystem eine Funktion (eine sogenannte Callback-Funktion), die bei der Zustellung vom Eingabesystem aufgerufen wird und als Argument einen Verweis auf die Daten der Meldung erhält. In der Praxis wird die Methode der Callback-Funktion durch eine GUI-Toolbox (Toolkit) aus der ersten Methode abgeleitet (z.B. MFC Event Handler auf GetMessage). Beiden Methoden gemeinsam ist die Tatsache, daß die Eingabemeldungen in einer oder mehreren Warteschlangen des Systems als Datenstrukturen bereitliegen.

3.2.6. Verknüpfung von Ein- und Ausgabesystem

Benutzereingaben an Anwendungsprogramme werden über andere Schnittstellen übermittelt, als die Grafikausgaben. Die Schnittstellen zwischen den eingabeverarbeitenden Komponenten, bzw. zwischen diesen Komponenten und den Anwendungsprogrammen sind funktional völlig getrennt von den Schnittstellen, über die Grafikausgaben stattfinden.

Wie in Kapitel 2.3.3.3 beschrieben, dienen in einigen Grafiksystemen Fenster aber nicht nur als Ausgabebereiche, sondern auch als Empfänger von Eingaben. Aus diesem Grund sind auf einigen Plattformen Grafiksysteme und Eingabesysteme nicht völlig unabhängig voneinander. Sie sind über die Fenster, die von Grafiksystem und Eingabesystem gemeinsam verwendet werden, assoziiert (Bsp.: MS Windows, X11). Fenster werden in diesem Fall nicht nur vom Grafiksystem als Ausgabebereich, sondern auch vom Eingabesystem als Eingabekanal verwendet. Für derartige Assoziationen gibt es historische Gründe. Die Assoziation von Grafiksystemen und Eingabesystemen über Fenster ist aber eher als Vermischung von Funktionen zu betrachten. Sie widerspricht einer klaren funktionalen Trennung. Dies ist an vielen Indizien abzulesen. So müssen sogar Anwendungsprogramme, die keine Grafikausgabe tätigen, einen - wenn auch unsichtbaren - Ausgabebereich anlegen, der dann als Eingabekanal dient.

Beispiele für Systeme auf denen Eingabe und Ausgabe nicht über die Ausgabebereiche assoziiert sind, sind MacOS 7/8 und OpenStep. Bei MacOS existiert eine klare Trennung zwischen Quickdraw als Ausgabesystem und dem Event Manager als Eingabesystem. Bei MacOS gibt es nur einen Eingabekanal je Prozeß, der automatisch vom System angelegt. Er ist unabhängig von den Quickdraw Ausgabebereichen (Grafports), sondern nur an den Prozeß gebunden (Kapitel 2.3.3.3).

3.3. Eingabemeldungen

3.3.1. Struktur von Eingabemeldungen

Typische Eingabemeldungen bestehen aus

- Typenfeld
  Das Typenfeld unterscheidet sowohl Klassen von Meldungen, d.h. z.B. IPC von Benutzereingabe, als auch Unterklassen, wie Mauszeigerbewegung von Mausklicks.

- meldungsspezifischen Daten
  Die Datenbereiche enthalten bei Mausklicks und -bewegung die Position. Bei der Bewegung oft auch noch den Zustand der Maustasten. Bei Tastatureingaben den Tastencode oder direkt das eingegebene Zeichen. In vielen Fällen den Zustand der Modifizierungstasten.
- Zeitstempel
- Referenz des Eingabekanals
  Auf Systemen, die mehrere Eingabekanäle je Anwendungsprogramm zulassen, enthält die Meldung die Referenz des Eingabekanals. Damit kann das Programm die Eingabe an and die zuständige Komponente (Behandlungsfunktion) weiterleiten.

Feld	MacOS 7/8	Windows	X11 (Maus/Tastatur)	X11 (Expose)
Typenfeld	what	message	message	message
Zeitstempel	when	time	time	time
Referenz des Eingabekanals	- (implizit)	hwnd	eventWindow	window
Aktuelle Mausposition	where	pt	eventX eventY	-
Meldungsspezifische Daten	message modifiers	wParam lParam	rootWindow childWindow rootX rootY state sameScreen/keyButMask	x y width height count

Tabelle Eingabemeldungen

Eingabemeldungen sind sich in verschiedenen Systemen sehr ähnlich. Bei MacOS 7/8 fehlt der Eingabekanal, weil es nur einen Eingabekanal je Prozeß gibt. Applikationen verwenden bei Mausklicks *message*, um die Eingabe einem Fenster zuzuordnen. Bei Tastatureingaben wird die Meldung über die Bestimmung des aktiven Fensters (FrontWindow) einem Dokument zugeordnet.
X11 *Expose*-Events entsprechen MacOS 7/8 *Update*-Events und Windows *PAINT*-Messages. X11 *Expose*-Events gelten immer für Rechtecke, nicht für Bereiche. Der *count*-Parameter gibt deshalb an, ob das aktuelle Event das letzte einer Serie ist. Bei MacOS und Windows gilt die Update-Nachricht prinzipiell für das gesamte Fenster (MacOS : message, Windows: hwnd). Das Programm kann dann über den Clipping Bereich den tatsächlich betroffenen Teil des Fensters bestimmen.

Da sich auf allen Plattformen die gleichen Eingabegeräte als Standardausstattung durchgesetzt haben, werden unter allen grafischen Benutzeroberflächen, die gleichen oder sehr ähnliche Typen von Eingabeereignissen verarbeitet. Obwohl die Eingabemeldungen sehr ähnlich sind, ist die Kodierung der Parameter von Eingabemeldungen von System zu System verschieden. Empfänger von Eingaben sind Anwendungsprogramme.

Man zwei Gruppen von Eingaben unterscheiden:

1. direkte Benutzereingaben und
2. indirekte Eingaben.

Direkte Benutzereingaben an Anwendungsprogramme sind die Eingaben, die über die vom Anwendungsprogramm erzeugten und kontrollierten Eingabeelemente, wie Knöpfe, Menüs und anwendungsspezifische Objekte, ausgelöst werden. Im Gegensatz dazu werden indirekte Eingaben vom Benutzer über die Kontrollelemente anderer Anwendungsprogramme, Kontrollelemente des Systems, der grafischen Oberfläche, des Window Managers oder vom System ausgelöst. Die Zuordnung von Kontrollelementen zu Programmen, bzw. zum System und damit die Aufteilung der Eingaben in direkte und indirekte Eingaben ist plattformabhängig.

3.3.2. Direkte Benutzereingaben

Anwendungsprogramme und die grafische Benutzeroberfläche (Window Manager, evtl. Menu Manager, etc.) teilen sich die Verarbeitung von Eingaben. Alle Eingaben werden zuerst vom Window Manager oder von einem Anwendungsprogramm verarbeitet. Die Reaktion der erstverarbeitenden Komponente kann anschließend weitere Meldungen auslösen. Diese Meldungen werden wieder von einem Anwendungsprogramm oder dem Window Manager verarbeitet. Wird eine Benutzereingabe zuerst von einem Anwendungsprogramm verarbeitet, dann spricht man von einer direkten Eingabe. Erhält dagegen ein Anwendungsprogramm eine Meldung, die durch die Reaktion einer anderen Komponente (Programm oder Window Manager) auf eine Benutzereingabe ausgelöst wurde, dann liegt eine indirekte Eingabe vor.

3.3.2.1. Mausklick

Mausklicks innerhalb von Fenstern sind direkte Eingaben. Der Randbereich, d.h. die Fensterdekoration, wird vom Window Manager verwaltet. Unter passiven Window Managern (siehe Kapitel 2.3.6.2) zählen auch Mausklicks auf den Fensterrand zu den direkten Eingaben, da sie zuerst vom Anwendungsprogramm verarbeitet werden. Damit unterliegen auch Manipulationen an der Fensterdekoration der Eingaberechtsvergabe (Kapitel 4.4.1). Bei MacOS 7/8 ist dies der Fall.

Eine Ausnahme sind Mausklicks in oder an Fenster, die nicht den Eingabefokus des Window Managers besitzen. Bei manchen Window Managern wird dieser Klick nur zur Umschaltung des Eingabefokus verwendet und aus dem Eingabestrom herausgefiltert.

Der Mausklick besteht aus zwei Meldungen: dem Drücken und Loslassen der Taste (genannt Mouse-Down und Mouse-Up Ereignisse). Beim Drücken der Maustaste werden die Position des Mauszeigers und im Fall mehrerer Tasten die Tastennummer gemeldet. Um Verklemmungen zu vermeiden (Kapitel 4.4.2.1) geht die Meldung über das Loslassen der Maustaste immer an den Eingabekanal (Fenster) in dem die Taste gedrückt wurde, nicht in das Fenster, in dem sie losgelassen wurde.

3.3.2.2. Mauszeigerbewegung

Bei manchen Window Managern kann allein die Mausposition zur Bestimmung des Eingabefokus verwendet werden (bei Sun OpenLook und Motif Window Manager einstellbar). Unabhängig davon, ob der Eingabefokus des Window Managers basierend auf der Mauszeigerposition umgeschaltet wird, gehen Meldungen über die Mauszeigerbewegung immer an das Anwendungsprogramm zu dem das unter dem Mauszeiger liegende Fenster gehört. Bei gedrückter Maustaste werden die Meldungen unter fast allen Window Managern allerdings so lange an das Fenster (das Anwendungsprogramm) geschickt, wie die Maustaste gedrückt bleibt. Dies gilt auch dann, wenn der Mauszeiger über andere Fenster fährt.

Die wichtigste Information bei der Mauszeigerbewegung ist natürlich die neue Koordinate des Mauszeigers. Daneben melden manche Eingabesysteme zusätzlich den Zustand der Maustasten (siehe Kapitel 6.2.1).

3.3.2.3. Tastatureingaben

Tastatureingaben sind Mausklicks sehr ähnlich. Auch Tastatureingaben bestehen aus zwei Meldungen, die das Drücken und Loslassen der Taste anzeigen. Statt der Tastennummer wird in den Tastaturmeldungen aber ein Tastaturcode gemeldet. Der Tastaturcode entspricht entweder direkt oder mit dem Umweg über eine Tastaturtabelle der gedrückten Taste. Prinzipiell besteht aber kein Unterschied zwischen Tastatureingaben und Mausklicks.

Wird eine Taste auf der Tastatur gedrückt gehalten, dann werden bei den meisten Eingabesystemen wiederholt Tastatureingaben (Drücken der Taste) erzeugt. Das Typenfeld kann diesen Fall vom ersten Drücken durch einen anderen Typencode unterscheiden.

Neben dem Tastaturcode liefern die meisten Eingabesysteme (MacOS, X11, Windows) bei Tastatureingaben auch den Zustand von Modifizierungstasten. Modifizierungstasten sind Ctrl, Alt, Option, Shift, wobei bei Alt und Shift oft zwischen der rechten und der linken Taste unterschieden wird. Das Drücken dieser Tasten wird von manchen Eingabesystemen anders behandelt, als das der übrigen Tasten. Nur bei X11 werden Modifizierungstasten, wie alle anderen Tasten gemeldet. Bei MacOS und Win32 erscheinen Modifizierungstasten nicht als Eingabemeldungen, sondern sie wirken auf die nachfolgenden Tastendrücke ein und resultieren bei gleichen Scancodes in geänderten Zeichencodes. Im Eingabesystem findet dann in den Schichten 1 und 2 schon Vorverarbeitung statt.

3.3.3. Indirekte Benutzereingaben

Indirekte Benutzereingaben sind Eingaben, die nicht vom Anwendungsprogramm verarbeitet werden. Der Benutzer schafft durch seine Eingaben an die Fensterverwaltung (z.B. Größenänderung des Fensters) oder durch Eingaben an andere Programme eine Situation in der ein Anwendungsprogramm benachrichtigt werden muß. Die Fensterverwaltung oder das Eingabesystem erzeugen daraufhin eine Meldung. Diese Meldung enthält im Gegensatz zu den direkten Eingaben nicht die Meldung über das Eingabeereignis, sondern über dessen Auswirkungen. Die Meldung durchläuft deshalb nicht das gesamte Eingabesystem, sondern wird in Ebene 3 oder 4 in das Eingabesystem eingefügt. Die wichtigsten indirekten Eingaben und deren Auswirkungen sind im folgenden aufgeführt.

3.3.3.1. Fensterverwaltung

Ändert der Benutzer Größe und Anordnung von Fenstern, dann wird diese Eingabe von einem aktiven Window Manager (Kapitel 2.3.6.2) selbständig ausgeführt und nach Beendigung als Mitteilung an den für das betroffene Fenster zuständigen Eingabekanal geschickt. Das entsprechende Programm kann dann auf die Änderung reagieren.

Bei passiven Window Managern zählen Änderungen von Größe und Anordnung von Fenstern zu den direkten Eingaben.

3.3.3.2. Redraw-Events

Redraw-Events werden von aktiven und passiven Window Managern erzeugt. Sie teilen dem Programm mit, daß Fensterinhalte neu gezeichnet werden müssen. Es gibt für den Benutzer keine Möglichkeit direkt Redraw-Events (so wie Tastaturanschläge) einzugeben. Redraw-Events entstehen immer durch andere Eingaben oder Ereignisse. Sie werden ausschließlich von Window Managern entsprechend der Situation der vom Window Manager verwalteten Fenster erzeugt (siehe Kapitel 6.3.2). Der Benutzer kann aber durch Eingaben ein Redraw-Event erzwingen, indem er eine Situation schafft in der ein Window Manager ein Redraw-Event erzeugt.

Ein Beispiel ist die Größenänderung eines Fensters (A), welches ein anderes Fenster (B) verdeckt. Die Benutzereingabe geht an den Eingabekanal, der Eingaben aus Fenster A verarbeitet (passiver Window Manager) oder an den Window Manager (aktiver Window Manager). Die Eingabe ändert die Größe von Fenster A. Fenster B, der Eingabekanal von Fenster B oder das Programm zu dem Fenster B gehört, sind an der Verarbeitung der Eingabe nicht beteiligt. Fenster B wird aber teilweise sichtbar. Der Window Manager erzeugt ein Redraw-Event für den nun sichtbaren Bereich, falls die entsprechenden Pixelwerte im Endgerät (z.B. Display Server) nicht mehr verfügbar sind.

3.3.4. Zusammengesetzte Eingaben

Viele Benutzereingaben sind zusammengesetzte Eingaben. Zusammengesetzte Eingaben bestehen aus einer Folge von Eingabeereignissen von Eingabegeräten, die von der Systemsoftware oder den Anwendungsprogrammen zusammengefaßt und gemeinsam verarbeitet werden. Auch mehrere zusammengesetzte Eingaben können wieder wie eine Eingabe verarbeitet werden. Tatsächlich kann der gesamte Eingabestrom als Hierarchie zusammengesetzter Eingaben verschiedener Ordnung (Hierarchiestufen) angesehen werden.

Bei Application Sharing Systemen ist vor allem die Gruppierung und Trennung von Eingaben auf verschiedenen Hierarchiestufen wichtig für die Integrität des Eingabestroms (Kapitel 4.4.2).

3.3.4.1. Elementare Eingaben

Die Basis der Eingabehierarchie bildet die Gruppe der Eingaben erster Ordnung. Diese elementaren Eingabeereignisse werden von den Eingabegerätetreibern aus Benutzereingaben erzeugt:

- Taste drücken (Maus und Tastatur)
- Taste loslassen
- Zeigerbewegung

Elementare Eingaben erscheinen für Anwendungsprogrammen zu einem definierten Zeitpunkt während des Programmlaufs. Die Ereignisse haben keine Dauer. Nur zusammengesetzte Eingaben haben eine

Dauer, die durch den Abstand der elementaren Eingaben (allgemeiner: die Zeit zwischen erster und letzter elementarer Eingabe) bestimmt ist.

3.3.4.2. Eingaben zweiter Ordnung

Die einfachsten zusammengesetzten Eingaben sind Sequenzen von elementaren Eingaben. Sie bilden die Eingaben zweiter Ordnung. Eingaben zweiter Ordnung werden dadurch voneinander abgegrenzt, daß sich alle Eingabegeräte im Grundzustand befinden. Sie bestehen aus einer Folge elementarer Eingaben zwischen jeweils zwei Grundzuständen aller Eingabegeräte. Der nicht gedrückte Zustand einer Taste ist dabei als der Grundzustand definiert. Mauszeiger werden in jeder Position als im Grundzustand angesehen. In der Praxis bedeutet dies, daß eine Eingabe zweiter Ordnung solange andauert, wie mindestens eine Taste gedrückt ist. Oft findet bereits während der Eingabe Datenverarbeitung statt, wie z.B. bei Direktmanipulation und automatischer Tastenwiederholung.

Auch Doppel- oder Mehrfachklicks gehören zu den Eingaben zweiter Ordnung. Sie fallen aber aus der Definition heraus und sind deshalb in Eingabeströmen oft schwer zu identifizieren. Aus diesem Grund werden softwaremäßig Pseudoeingabegeräte konstruiert, die der Definition zusammengesetzter Eingaben entsprechen. Im Falle des Doppelklicks wird das "Doppelklick-Pseudogerät" durch einen Klick (zwei elementare Eingaben) aktiviert und automatisch nach Ablauf des eingestellten Doppelklickintervalls deaktiviert.

Zusammengesetzte Eingabe zweiter Ordnung	Sequenz elementarer Eingaben
Tastenklick: • Tastaturanschlag und Mausklick, • Automatische Tastenwiederholung.	Taste gedrückt + Taste losgelassen
Tastenklick mit gedrückter Modifizierungstaste	Modifizierungstaste gedrückt + Taste gedrückt + Taste losgelassen + Modifizierungstaste losgelassen
Doppel- oder Mehrfachklick der Maustaste	Maustaste gedrückt + Maustaste losgelassen + Maustaste gedrückt + Maustaste losgelassen innerhalb einer definierten kurzen Zeit
Mauszeigerbewegung bei gedrückter (Maus-)Taste: • Menüauswahl, • Direktmanipulation von Objekten (drag-and-drop), • Bedienung von Schiebereglern und anderen Eingabeelementen.	Maustaste gedrückt + Folge von Mausbewegungsmeldungen + Maustaste losgelassen

Tabelle Zusammengesetzte Eingaben

Zusammengesetzte Eingaben (Eingaben zweiter Ordnung) bestehen aus elementaren Eingaben. So werden z.B. die elementaren Eingaben *Drücken* und *Loslassen* einer Taste zu einer Eingabe zusammengefaßt und als Eingabe eines Zeichens verarbeitet. Auch das Selektieren von Objekten durch einen einfachen Mausklick zählt zu den Eingaben zweiter Ordnung.

3.3.4.3. Eingaben dritter Ordnung

Die dritte Ordnung von Eingaben wird aus Folgen von Eingaben zweiter Ordnung gebildet. Es sind abgeschlossene Bearbeitungsvorgänge auf den Objekten gemeinsam benutzter Anwendungsprogramme. Diese Bearbeitungsvorgänge sind offensichtlich anwendungsabhängig und für jedes Programm verschieden.

Anwendung	Vorgang	Folge von Eingaben zweiter Ordnung
Grafikbearbeitung	Füllen einer Fläche	Auswahl der Aktion (Füllen), der Farbe und des Anfangspunktes.
	Anwendung eines Filters	Auswahl eines Bildbereichs, der Aktion, Wahl von (Filter-)Parametern, Start der Verarbeitung.
Textverarbeitung	Wort-/Satzeingabe	Folge von Tastenanschlägen.
viele Programme	Parametereinstellung	Auswahl eines Dialogfeldes, Bearbeitung von Dialogelementen, Schließen und/oder Bestätigen der Einstellungen.

Tabelle Beispiele für Bearbeitungsvorgänge

Beispiele für Bearbeitungsvorgänge (Eingaben dritter Ordnung). Bearbeitungsvorgänge umfassen jeweils mehrere zusammengesetzte Eingaben zweiter Ordnung.

Die vierte Ordnung von Eingaben umfaßt die gesamte Bearbeitung eines Dokuments, bzw. alle Eingaben, die der Steuerung eines Programms dienen. Sie wird aus Folgen von Bearbeitungsvorgängen gebildet und ist deshalb, wie diese anwendungsabhängig.

3.3.5. Quantitative Analyse

In diesem Kapitel soll die bei Eingabeströmen anfallende Datenrate (in Byte pro Sekunde) abgeschätzt werden. Natürlich hängt die Datenrate wesentlich von der Eingabe des Benutzers ab. Finden keine Eingaben statt, dann werden von den Eingabegeräten keine Daten erzeugt. Die untere Grenze liegt deshalb bei 0 Byte/s. Interessant im Zusammenhang von Application Sharing sind aber folgende Parameter:

1. die maximale Datenrate, die von den Eingabegeräten erzeugt werden kann und
2. die Datenrate bei typischen Arbeitsabläufen.

Wie die Grafikausgaben, so hängt auch die Datenrate von Eingabeströmen von der Darstellung und den gewählten Datenstrukturen ab. Auch in diesem Fall spielen Kodierung, Kardinalität und Optimierungen, wie Kompression eine Rolle. Bei den meisten Eingabesystemen wird aber für alle hier betrachteten Eingabemeldungen (Mausklick, Mauszeigerbewegung, Tastatureingaben) und für Meldungen durch indirekte Eingaben die gleiche Datenstruktur verwendet. Die Datenstrukturen unterscheiden sich bei verschiedenen Eingabesystemen, haben aber die gleiche Funktion und vergleichbare Größe (X11: 29 relevante Byte von 32; MacOS: 16 Byte; Win32: 26 Byte).

3.3.5.1. Maximale Datenrate

Generell müssen alle Eingabemeldungen von Anwendungsprogrammen verarbeitet werden. Die Datenrate ist deshalb beschränkt durch die Verarbeitungsleistung von Computersystemen auf denen die Programme laufen. Geht man davon aus, daß die Zeitscheiben in Multitaskingsystemen in der Größenordnung von 1 ms liegen (SunOS 4.x: 20 ms; MacOS 7/8: 1/60 s) und, daß je Zeitscheibe eine Meldung verarbeitet wird, dann erhält man eine obere Grenze für die Datenrate von 1000 Meldungen pro Sekunde (bei X11: ca. 4000 Byte/s).

Eine andere Abschätzung erhält man durch die Betrachtung von realen Eingabemöglichkeiten. Hier kommen vor allem Mauszeigerbewegungen und Tastaturwiederholungen in Frage. Alle anderen Eingaben treten mit viel geringerer Häufigkeit auf. Tastaturwiederholungen werden in Eingabesystemen automatisch erzeugt, wenn eine Taste gedrückt bleibt. Man kann davon ausgehen, daß eine Tastaturwiederholung, die 4 Zeilen je 100 Zeichen pro Sekunde erzeugt, als schnell gilt. Daraus ergibt sich eine Rate von 400 Meldungen pro Sekunde (bei X11: ca. 1600 Byte/s).

Ein üblicher Monitor bietet eine Auflösung von ca. 1000 Pixeln (horizontal häufig 1024x768 oder diagonal bei 800x600). Die Meldung von Mauszeigerbewegungen ist nur dann sinnvoll, wenn sich die Koordinate des Zeigers ändert. Typisch sind Geschwindigkeiten bis zu 3000 Pixeln pro Sekunde. Daraus ergibt sich eine obere Grenze von 3000 Meldungen pro Sekunde (bei X11: ca. 12000 Byte/s).

3.3.5.2. Datenrate bei typischen Arbeitsabläufen

Bei typischen Arbeitsabläufen sind die Raten von Eingabemeldungen wesentlich geringer, als die Maximalen. Schnelle Tastatureingabe läuft mit bis zu 10 Anschlägen pro Sekunde.

Tastaturwiederholungen mit 50 Zeichen pro Sekunde sind ausreichend schnell. Bei Mauszeigertreibern sind oft polynomiale oder exponentielle Beschleunigungen eingebaut, so daß bei weitem nicht alle berührten Koordinaten gemeldet werden.

Die Größe von Eingabemeldungen und Ausgabekommandos (außer Pixmap, Polygon und Bereich) ist ähnlich. Im allgemeinen besteht eine Programmreaktion auf eine oder wenige Eingabemeldungen aber aus sehr vielen Ausgabekommandos.

Berücksichtigt man, daß Benutzereingaben fast immer Programmreaktionen in Form von Grafikausgaben hervorrufen, dann kann das Datenvolumen der Eingaben gegenüber dem der Ausgaben vernachlässigt werden.

4. Komponenten von Application Sharing Systemen

In diesem Kapitel werden die vier Basiskomponenten von Application Sharing Systemen ohne verschiedener Grafiksysteme diskutiert. Kapitel 5 befaßt sich anschließend mit dem Komplex der Konvertierung. Die Kombination der Basiskomponenten und der Konvertierungskomponente wird in Kapitel 7 beschrieben.

4.1. Einleitung

Die Architektur von Anwendungsprogrammen ist auf die Bedienung durch nur eine Person ausgerichtet. Sogar Programme für Mehrbenutzerbetriebssysteme, wie UNIX, gehen von einer eins-zu-eins Beziehung aus zwischen

- dem Anwendungsprogramm,
- den Ausgabegeräten (darstellender Bildschirm) und
- den Eingabegeräten (Maus, Tastatur, Mikrofon, Kamera).

Die Programme werden dabei von den Strukturen und Funktionen der Systemdienste üblicher Arbeitplatzrechner unterstützt. Moderne Systemsoftware ist dafür ausgelegt, zu gleichen Zeit mehrere Anwendungsprogramme zu verwalten, die durch präemptives oder kooperatives Multitasking quasi-gleichzeitig ablaufen. Die Systemsoftware sorgt dafür, daß ein Benutzer gleichzeitig mehrere Anwendungsprogramme bedienen kann. Der Eingabestrom wird dazu durch Umschaltung des Eingabefokus' auf mehrere Programme aufgeteilt. Gleichzeitig werden die Ausgabeströme mehrerer Programme zur Ausgabe auf einem Ausgabegerät zusammengefaßt. Normalerweise geschieht dies erst sehr nahe bei den Ausgabegeräten durch die Gerätetreiber. Das kann man am Beispiel der Grafikausgaben sehen, die getrennt durch ein Grafiksystem geführt werden und erst von einem Bildschirmtreiber in den gleichen Bildschirmspeicher gezeichnet werden.

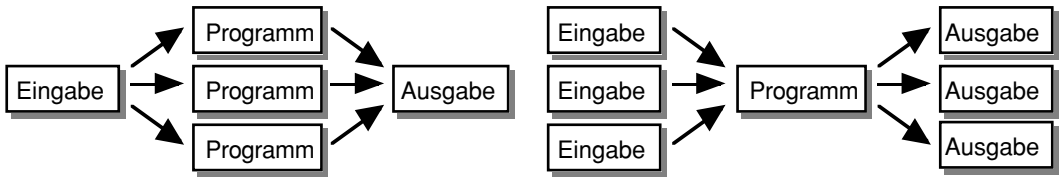


Abbildung I/O Sharing vs. Application Sharing

Die Systemsoftware sorgt durch Mehrfachbenutzung von Eingabegeräten und Ausgabegeräten dafür, daß sich mehrere Programme einen Satz von Geräten teilen können. Beim Application Sharing stellt sich die Situation aber genau umgekehrt dar. Beim Application Sharing müssen mehrere Eingabeströme zu einem einzigen Strom zusammengefaßt werden, während der Ausgabestrom auf mehrere Ausgabegeräte verteilt werden muß. Die Systemsoftware, die von den genannten eins-zu-eins Verhältnissen ausgeht, ist dafür jedoch nicht ausgelegt. Diese Aufgaben müssen deshalb vom Application Sharing System übernommen werden. Die Funktionalität des Zusammenfassens von Eingabe- und des Verteilens von Ausgabeströmen komplementiert die Funktionen des Betriebssystems. Auf den meisten Betriebssystemen bzw. Plattformen ist ein Application Sharing System deshalb zumindest teilweise auf der Ebene von Systemdiensten angesiedelt.

Dem Zusammenfassen und Verteilen von Eingabeströmen und Ausgabeströmen dienen vier Basiskomponenten:

1. Ausgabeanalysator,

2. Ausgabereplikator,
3. Eingabekonzentrator und
4. Eingabeinjektor.

Sind verschiedene Grafiksysteme beteiligt, dann wirkt noch eine fünfte Komponente mit:

- der Grafikstromübersetzer

In den folgenden Kapiteln werden die vier Basiskomponenten zum Zusammenfassen und Verteilen von Eingabeströmen und Ausgabeströmen dargestellt und die Implementierung auf einzelnen Plattformen beschrieben. Die Übersetzung von Ausgabeströmen, bzw. von Benutzereingaben wird hier noch ausgeklammert und erst weiter unten diskutiert (Kapitel 5 und Kapitel 6). Die Übersetzerkomponente kann an zwei verschiedenen Stellen, vor und nach dem Replikator, eingefügt werden. Die daraus entstehende Anordnung der Komponenten wird im Kapitel 7 diskutiert.

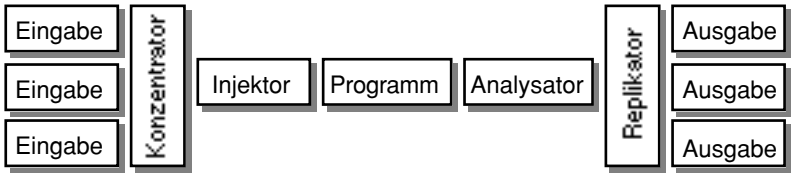


Abbildung Sharing System Komponenten

Anordnung der Basiskomponenten eines Application Sharing Systems ohne Übersetzer.

4.2. Der Ausgabeanalysator

Die Aufgabe eines Ausgabeanalysators besteht darin, die gesamte Ausgabe eines Anwendungsprogramms an der Benutzerschnittstelle zu erfassen und an den Ausgabereplikator weiterzuleiten.

Der Ausgabeanalysator untersucht (analysiert) die grafikrelevanten Aktionen von Programmen. Durch die Analyse von Prozeduraufrufen, bzw. PDUs eines Grafikprotokolls, erzeugt der Ausgabeanalysator einen Grafikausgabestrom, der die sichtbaren Ausgaben wiedergibt. Eine einfache Umleitung von Grafikausgabekommandos, liegt auf keinem System vor. Auf allen Systemen sind eingehende Analysen der Programmaktionen notwendig, um einen korrekten Grafikausgabestrom zu rekonstruieren.

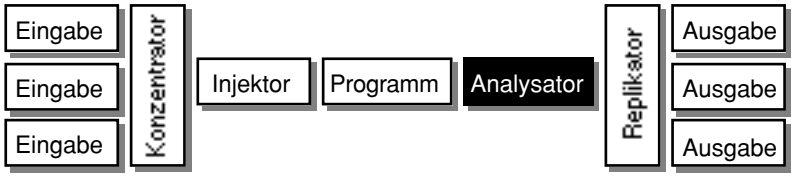


Abbildung Sharing System Komponenten - Analysator

Der Ausgabeanalysator wird zwischen dem Anwendungsprogramm und dem Ausgabegerät eingefügt. Er schickt den extrahierten Grafikausgabestrom zur Verteilung an den Ausgabereplikator

Die Vollständigkeit der Erfassung spielt dabei eine wesentliche Rolle. Fehlen Teile des Ausgabestroms, dann können offensichtlich unvollständige oder falsche Darstellungen bei anderen Teilnehmern entstehen.

Ausgaben an die Benutzerschnittstelle werden oft von vielen verschiedenen Teilen eines Anwendungsprogramms erzeugt. Es ist deshalb im allgemeinen keine Programmfragment oder -modul identifizierbar, das alle Ausgaben erzeugt. Umgekehrt verwenden Programme eine große Zahl von Funktionen der Systemsoftware, um Grafikausgaben zu erzeugen. Ausgaben werden nur teilweise direkt durch Aufruf von Funktionen eines Grafiksystems ausgelöst. Viele Ausgabeoperationen geschehen indirekt, d.h. mit einem Umweg über Verwaltungsfunktionen der Systemsoftware, die selbst wiederum

Grafiksystemfunktionen verwendet. Auch der Einsatz von Bibliotheken für die Verwaltung der Benutzerschnittstelle, sogenannte Toolkits (X11: Motif; MS Windows: MFC) erhöht den Anteil indirekter Ausgaben am Ausgabestrom.

Ein Beispiel für indirekte Ausgaben sind Elemente der Benutzerschnittstelle wie Schieberegler oder Knöpfe. Solche Objekte werden von Programmen erzeugt und in Fenstern plaziert. Verwaltet werden sie aber oft von Teilen der Systemsoftware oder Toolkits. Die Animation bei der Bewegung eines Schiebereglers wird dann nicht vom betreffenden Anwendungsprogramm selbst durchgeführt, sondern von einem Toolkit. Trotzdem gehört der Zustand des Schiebereglers zur Darstellung des Programmzustands. Die Animation der Bewegung gehört deshalb zum Ausgabestrom ohne, daß die Grafikausgaben vom Anwendungsprogramm selbst erzeugt werden.

4.2.1. Ausgabeanalyse durch Umleitung der Ausgaben

Die Erfassung von Grafikausgaben ist an verschiedenen Stellen während des Ausgabeflusses zwischen Anwendungsprogramm und Bildschirmspeicher möglich. Datenvolumen und Art der erhaltenen Informationen unterscheiden sich, je nachdem, wo der Analysator ansetzt. Grundsätzlich muß ein Ausgabeanalysator in die Struktur des Grafiksystems passen, um dessen Funkton nicht zu stören. Man kann dies erreichen, indem man den Analysator mit den gleichen Schnittstellen versieht, wie eine ausgewählte Komponente des Grafiksystems. Wird der Analysator dann statt der betreffenden Komponente als Teil des Grafiksystems installiert, so kann er dem Replikator den Teil des Ausgabestroms zuleiten, der durch die ursprüngliche Komponente bearbeitet würde.

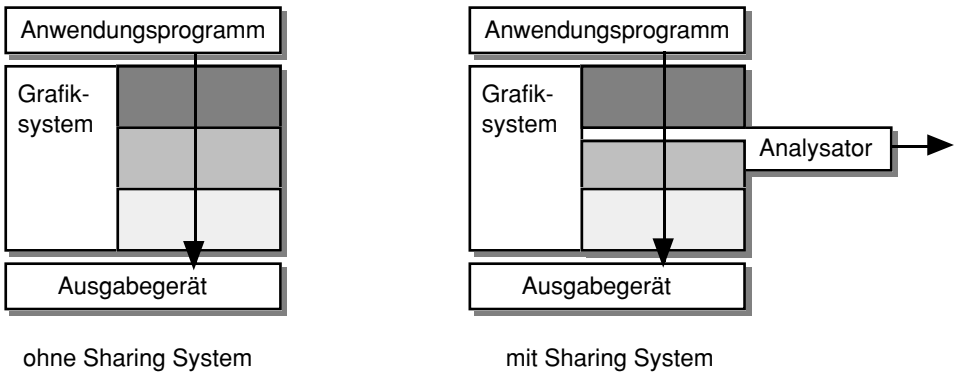


Abbildung Ausgabeanalysator als Interceptor im Grafiksystem

> Der Ausgabeanalysator hat die gleiche Schnittstelle, wie eine Komponente des Grafiksystems. Er erfüllt die gleiche Aufgabe, wie die ersetzte Komponente und extrahiert zusätzlich den Ausgabestrom eines Anwendungsprogramms. Dabei wird in die ersetzte Komponente zur Darstellung der Originalfunktion verwendet.

Der Ausgabeanalysator muß deshalb die Funktion der ersetzten Komponente exakt nachbilden. Am einfachsten ist dies zu realisieren, indem man den erhaltenen Ausgabestrom im Analysator zusätzlich an die ursprüngliche ersetzte Komponente des Grafiksystems weiterleitet. Das Duplikat des Ausgabestroms geht an den Ausgabereplikator.

4.2.2. Ausgabeanalyse durch Systemschnittstellen

Unterstützt ein Grafiksystem externe Zuhörer, dann muß der Ausgabeanalysator nicht durch Ersetzen einer Komponente installiert werden. Alle an der Schnittstelle registrierten Zuhörer erhalten zu jedem Zeitpunkt eine genaue Kopie des Ausgabestroms. Solch eine Schnittstelle wurde von Microsoft für Windows NT 5 in Aussicht gestellt.

Der Ausgabeanalysator für das MaX System [WoFrSchu95] (QDTracker) realisiert die Ausgabeanalyse durch Umleitung der Ausgaben, wie oben beschrieben. Er wurde aber so implementiert, daß er für MacOS 7/8 eine externe Zuhörerschnittstelle bereitstellt. Bei dieser Zuhörerschnittstelle registriert sich das MaX Application Sharing System.

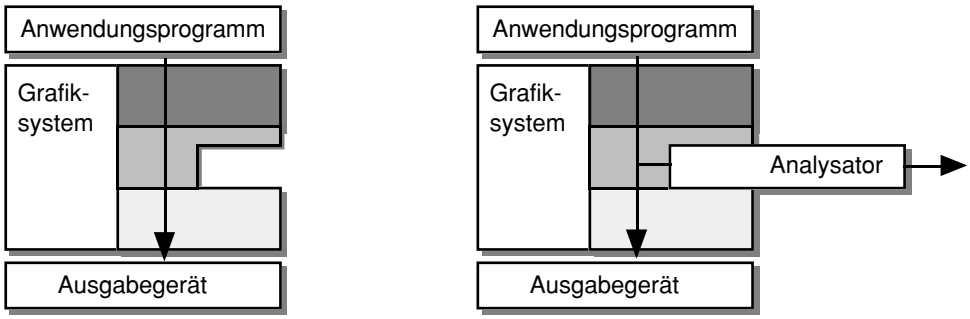


Abbildung Ausgabeanalysator an Systemschnittstelle

Der Ausgabeanalysator wird an der Zuhörerschnittstelle registriert. Die Ausgaben werden vom Grafiksystem dupliziert und externen Zuhörern zugeleitet.

4.2.3. Fensterverwaltung

Durch die Kommandos der Fensterverwaltung werden Ausgabeströme eröffnet, dirigiert und geschlossen. Diese Kommandos müssen zusätzlich zu den Ausgabeströmen analysiert und bearbeitet werden, um die gleiche Fensterverwaltung auf entfernten Displays wiederzugeben. Ein Ausgabeanalysator muß deshalb neben den Grafikausgaben von Anwendungsprogrammen auch deren Fensterverwaltung analysieren. In den meisten Fällen wird der Analysator für die Fensterverwaltung durch den gleichen Mechanismus installiert und betrieben, wie der Analysator für die Grafikausgaben. Vor allem dann, wenn Grafiksystem und Fensterverwaltung assoziiert sind (Kapitel 2.3) können beide Analysatoren integriert werden.

Arbeitet der Grafikanalysator auf einer Ebene, auf der keine Fensterverwaltungskommandos vorliegen, dann ist diese Integration nicht möglich. Ein Beispiel ist der Grafikausgabeanalysator auf Treiberebene. Wird der Grafikausgabeanalysator als Treiber installiert, dann erhält er nur Informationen über Grafikausgaben (den Inhalt von Fenstern), aber nicht über die Fensterverwaltung. Grafikbefehle des Window Managers, die nur die Fensterdekoration zeichnen, sind wertlos, da die semantische Qualität der Fensterverwaltungskommandos (Kapitel 2.3.6.1) fehlt. In diesem Fall muß der Grafikausgabeanalysator auf Treiberebene durch einen Analysator für die Fensterverwaltung auf anderer Ebene (z.B. API) ergänzt werden.

4.2.4. Tonausgaben

Es gibt zwei Typen von Tonausgaben: komplexe und simple Ausgaben. Simple Tonausgabe dient nur der Erzeugung von Warn-/Signaltönen, wie sie auch schon in textbasierten Benutzerschnittstellen (ASCII) als BEL oder CTRL-G zur Verfügung standen. Bei der Entwicklung der graphischen Benutzerschnittstellen wurde diese Möglichkeit der simplen Tonausgabe in die Programmierschnittstelle aufgenommen. Die Art des ausgegebenen Tons ist dabei meistens nicht oder nur minimal durch das Anwendungsprogramm beeinflußbar. Die Programmierschnittstelle für simple Tonausgaben ist deshalb auch meistens sehr einfach. Oft besteht sie nur aus einer Prozedur (Apple Macintosh: SysBeep; X11: Bell; MS Windows: MessageBeep). Im folgenden wird die Programmierschnittstelle für simple Tonausgaben als Teil des Grafiksystems behandelt, da für simple Tonausgaben die gleichen Regeln gelten, wie für Grafikausgaben. Komplexe Tonausgaben zählen zur Multimediaprogrammierschnittstelle.

4.2.5. Ausgabeanalyse an der Programmierschnittstelle

Die Programmierschnittstelle (API: Application Programmers Interface) dient dem Zugriff des Anwendungsprogrammierer auf die Funktionen des Grafiksystems. Aus der Sicht des Anwendungsprogramm ist die Programmierschnittstelle der erste Teil des Grafiksystems.

Die Routinen der Programmierschnittstelle werden zum Anwendungsprogramm gebunden. Dies geschieht entweder dynamisch zur Laufzeit bzw. bei Programmstart oder statisch. Ist die

Programmierschnittstelle statisch gebunden, so kann sie nicht auf einfache Art durch den Ausgabeanalysator ersetzt werden (Beispiel: X-Clients mit statisch gebundener Xlib). Die einzige Möglichkeit den Ausgabeanalysator auf der Ebene der Programmierschnittstelle anzubringen besteht dann in einer Analyse und Manipulation der ausführbaren Programmdatei. Der Ausgabeanalysator arbeitet dann nicht systemweit, sondern nur für das entsprechende Anwendungsprogramm.

Der Fall der dynamisch gebundenen Programmierschnittstelle ist wesentlich häufiger. Beispiele hierfür sind die Software-Interrupts (Traps) des Apple Macintosh, MS Windows DLLs und viele andere Arten von shared Libraries. Beim Binden werden die Namen von Bibliotheksprozeduren oder Prozedurgruppen in Einsprungadressen aufgelöst. Die Prozeduren werden, wenn notwendig, in den Adressraum eines Prozesses eingeblendet und Sprungbefehle im Anwendungsprogramm mit den Einsprungadressen versehen. Das Programm verwendet dann den Code, auf den die Sprungbefehle zeigen.

Dynamisches Binden bedeutet, daß dieser Vorgang bei Programmstart oder zur Laufzeit beim Aufruf einer Bibliotheksprozedur abläuft. Es bietet sich deshalb die Möglichkeit, andere Einsprungadressen in die Sprungbefehle einzubringen, indem man die Namensauflösung ändert. Die Namensauflösung muß so manipuliert werden, daß die Sprungbefehle auf Einsprungadressen von Routinen des Ausgabeanalysators verweisen. Das Anwendungsprogramm benutzt dann die Routinen des Ausgabeanalysators statt denen der Programmierschnittstelle.

4.2.5.1. Der Ausgabestrom auf API-Ebene

Der Ausgabeanalysator erhält die Informationen über den Ausgabestrom aus den Parametern der Prozeduraufrufe. Da dies die Schnittstelle ist, über die alle Grafikkommandos an das Grafiksystem geschickt werden, stehen komplette Ausgabekommandos zur Verfügung, also Grafikprimitiv, Modifikationsparameter und Grafikkontextparameter sowie Fensterzuordnung:

- Das **Grafikprimitiv** wird bestimmt durch die Wahl der Prozedur (Die Einsprungadresse oder Prozedurnummer).
- **Modifikationen** erhält der Ausgabeanalysator aus den Argumenten des Prozeduraufrufs. Teile der Modifikationsparameter können auch durch die Wahl der Prozedur bestimmt sein.
- **Kontextparameter** werden aus Prozeduraufrufen abgeleitet, die der Auslösung der Ausgabeoperation vorausgehen.
- Die **Fensterzuordnung** ergibt sich aus Prozedurargumenten, die mit der Ausgabeauslösung oder vorher von den Anwendungsprogrammen übergeben werden.

An der Programmierschnittstelle erscheinen die Grafikausgaben in der vom Programmierer beabsichtigten Form und Reihenfolge. Da die Programmierschnittstelle der erste Teil des Grafiksystems ist, werden Ausgabekommandos von keiner anderen Instanz verändert. Außer den Parametern ist deshalb auch die Semantik des Ausgabestroms auf Grafikprimitivebene noch voll erhalten. Die Semantikerhaltung ist wichtig im Bereich der Grafikprimitive und der Farbdarstellung:

- Grafikprimitive
  Grafikprimitive werden vom Grafiksystem bei der Ausgabe auf einige Basistypen reduziert. Dies geschieht sowohl bei der Anwendung des Clippings, als auch bei der Ansteuerung von Bildschirmtreibern, die bis auf Ausnahmen, nur wenige Grafikprimitive direkt unterstützen. Höherwertige Primitive, wie Kurven, Rechtecke usw., müssen deshalb in den unteren Schichten von Grafiksystemen auf die verfügbaren Typen (z.B. Linien) abgebildet werden. In der Programmierschnittstelle ist dies aber noch nicht der Fall.
- Farbdarstellung
  Programmierschnittstellen, die geräteunabhängige oder Echtfarbausgaben ermöglichen, liefern dem Ausgabeanalysator einen Ausgabestrom mit geräteunabhängigen Farbspezifikationen. Geräteunabhängige Farbspezifikationen von Grafikkommandos müssen zur Ausgabe in geräteabhängige umgewandelt werden. Bei Bildschirmen mit indizierten Farben, z.B. Bildschirmspeicher mit 8 Bit je Pixel oder Graustufendarstellung, ist deshalb in hardwarenäheren Schichten des Grafiksystems, z.B. auf Treiberebene, nur noch ein reduzierter und verfälschter Farbumfang im Ausgabestrom enthalten.
- Zeichensätze
  Es gibt Grafiksysteme, die nach einer Zeichensatzanforderung durch Anwendungsprogramme selbständig einen Zeichensatz auswählen, der den Anforderung möglichst nahe kommt, bzw. entspricht. Eventuell wird ein Ersatzzeichensatz ausgewählt und für die folgenden Textausgaben

verwendet. Nur an der Programmierschnittstelle erscheint die Zeichensatzspezifikation (Zeichensatz, Größe, Stil, Farbe usw.), wie von einem Anwendungsprogramme angegeben. In hardwarenäheren Schichten des Grafiksystems sind nur Informationen über den tatsächlich verwendeten Zeichensatz verfügbar.

Ausgaben von Programmen, welche die Programmierschnittstelle des Grafiksystems umgehen und direkt den Bildschirmtreiber verwenden oder sogar in den Bildschirmspeicher schreiben, können auf diese Art nicht erfaßt werden. Auch Ausgaben von Multimedia-Systemerweiterungen, die eigene optimierte Ausgabemechanismen verwenden, erscheinen nicht im Ausgabestrom an der Programmierschnittstelle des Grafiksystems.

4.2.5.2. Umfang von Ausgabeanalysatoren auf API-Ebene

Um die Funktion des Anwendungsprogramms nicht zu stören, muß der Ausgabeanalysator die Programmierschnittstelle exakt nachbilden. Die Programmierschnittstelle ist eine veröffentlichte Schnittstelle. Sie wird von vielen Anwendungsprogrammen verwendet und ist deshalb relativ stabil. Gewöhnlich wird die Programmierschnittstelle vom Hersteller des Grafiksystems im Laufe der Zeit nur erweitert, aber nicht grundlegend geändert, um den Ablauf älterer Anwendungsprogramme nicht zu stören. Die Abhängigkeit des Herstellers des Grafiksystems von den Anwendungsprogrammen ist ein Vorteil für die Funktionsfähigkeit des Ausgabeanalysators auf der Programmierschnittstelle.

Programmierschnittstellen von Grafiksystemen wurden entwickelt um den Ansprüchen vieler Anwendungsprogrammierer nach komfortabler Grafikausgabe zu genügen. Sie stellen deshalb eine sehr große Zahl von Routinen zur Verfügung. Teilweise überdeckt sich sogar die Funktionalität einiger angebotener Routinen. Zur Implementierung müssen für alle Routinen Ersatzroutinen erstellt werden, die Ausgabeoperationen analysieren und deren Parameter extrahieren. Die Gesamtzahl der Routinen kann in der Größenordnung von 100 liegen (siehe Kapitel 2.4). Tatsächlich muß aber oft nicht die gesamte Programmierschnittstelle ersetzt werden, sondern nur eine Untermenge. Wie groß der notwendigen Anteil ist hängt von der Programmierschnittstelle ab. Man kann für den Ausgabeanalysator drei Gruppen von Funktionen unterscheiden:

1. Ausgabeauslösung,
2. Kontextmanipulation und
3. Hilfsfunktionen und Ressourcemanagement.

Die zentralen Ausgabefunktionen bilden den kleinsten Teil der Programmierschnittstelle. Ihr Aufruf muß auf jeden Fall analysiert werden. Die Funktionen zu Manipulation des Grafikkontextes müssen nur dann verfolgt werden, wenn keine Möglichkeit besteht während einer Ausgabeoperation die relevanten Parameter des Grafikkontextes abzufragen. Hilfsfunktionen und Funktionen für das Ressourcemanagement, wie für das Erzeugen von Clippingbereichen, Laden und Anlegen von Ressourcen müssen nicht oder nur in Ausnahmefällen analysiert werden.

Unter der Voraussetzung, daß die Namensauflösung beim Binden nur für eine Untermenge der Routinen geändert werden kann, während die anderen unverändert bleiben, kann auf einen Großteil der Ersatzroutinen verzichtet werden. Dies trägt zur Überschaubarkeit, Wartungsfreundlichkeit und Stabilität des Ausgabeanalysators bei.

Die oben erwähnten indirekten Ausgaben durch Teile der Systemsoftware verwenden in manchen Systemen (Bsp.: Macintosh Toolbox auf Quickdraw und Motif auf der Xlib) auch die Programmierschnittstelle des Grafiksystems. In diesem Fall kann der gesamte Ausgabestrom an der Programmierschnittstelle analysiert werden. Dafür müssen zwei Bedingungen erfüllt sein:

1. Fensterzuordnung
   Die Fensterzuordnung muß auch bei Ausgaben durch GUI Toolkits rekonstruiert werden können. Manche GUI Toolkits verwenden in ihren Ausgabeoperationen nicht Fenster der Anwendungsprogramme, sondern eigene künstliche Fenster in denen die Ausgaben so positioniert werden, daß sie an der gleichen Stelle auf dem Bildschirm erscheinen. In diesen Fällen kann es notwendig sein, Aufrufe von Routinen des GUI Toolkits zu verfolgen, um die Fensterzuordnung zu erhalten. (Bsp: MacOS 7/8: PopUpMenuSelect).
2. Erfassung indirekter Ausgaben.
   Indirekte Ausgaben können auch unter Umgehung der Programmierschnittstelle von GUI Toolkits direkt ausgegeben werden. Die GUI Toolkits verwenden dann eine Schnittstelle ähnlich der Programmierschnittstelle, aber mit anderen Einsprungpunkten. Bei Windows NT ist die gesamte

Funktionalität des Grafiksystems einschließlich der Programmierschnittstelle im Systemkern repliziert. Ein Ausgabeanalysator an der Programmierschnittstelle von Anwendungsprogrammen erhält deshalb keine Informationen über indirekte Ausgaben. In diesem Fall muß auf andere Schnittstellen ausgewichen werden.
Bei MacOS 7/8 verwendet auch die GUI Toolbox die Programmierschnittstelle von Quickdraw. Es ist deshalb möglich den gesamten Ausgabestrom einschließlich der indirekten Ausgaben an der Programmierschnittstelle zu analysieren.

4.2.5.3. Beispiel MS Windows GDI (Win32) Analysator

Die Programmierschnittstelle von MS Windows ist das sogenannte Windows GDI (Graphical Device Interface). Zur Unterscheidung der verschiedenen Versionen siehe [Schöttner96]. Hier soll nur auf das Win32 API von Windows 95 und Windows NT Bezug genommen werden. Aber prinzipiell gilt der Mechanismus auch für andere GDI Versionen.

Beim Programmstart oder bei Systemstart (je nach Windows Version) wird die Datei GDI.DLL geladen. Sie enthält neben den Routinen der Programmierschnittstelle viele Funktionen des Grafiksystems. Zur Manipulation der Namensauflösung für Routinen der Programmierschnittstelle kann die Datei GDI.DLL ersetzt werden. Statt dem echten Windows GDI werden dann die Ersatzprozeduren in die Anwendungsprogramme gebunden. Dadurch werden alle Routinen der Programmierschnittstelle gleichzeitig ersetzt.

Unter Windows NT besteht die Möglichkeit zur Laufzeit eines Anwendungsprogramms den Ausgabeanalysator in den Adressraum einzublenden. Der Ausgabeanalysator manipuliert dann die Tabelle der Einsprungpunkte der GDI32.DLL, um die Adressen seiner Ersatzprozeduren einzutragen [Schöttner96].

4.2.5.4. Beispiel Xlib Analysator

Bei dynamisch gebundenen Anwendungsprogrammen für das X Window System kann ein sehr ähnlicher Mechanismus verwendet werden, wie für das MS Windows GDI. Die Xlib (normalerweise als Datei libX11.a) wird durch eine Datei gleichen Namens ersetzt, welche die Ersatzroutinen enthält. Die Ersetzung geschieht entweder direkt durch eine andere Datei oder durch eine geänderte Umgebungsvariable (LD_LIBRARY_PATH) des Linkers.

Ein Problem stellen Programme dar, die UI Toolkits, wie OpenWindows oder Motif verwenden. Die Bibliotheken der Toolkits verwenden zwar fast ausnahmslos die Xlib, manchmal ist diese aber statisch zu den Tookitbibliotheken gebunden. Es gibt dann keine einfache Möglichkeit den Ausgabeanalysator an der Xlib, der Programmierschnittstelle des X Window Systems, anzubringen.

Zusätzlich werden noch Informationen von Prozeduren zur Kontextmanipulation benötigt, da die Parameter des Grafikkontextes nicht abfragbar sind. Der Analysator der Fensterverwaltung benutzt die gleiche Schnittstelle.

4.2.5.5. Beispiel Apple Macintosh Quickdraw Analysator

Die Funktionen des Grafiksystems Quickdraw des Apple Macintosh sind ähnlich einer shared Library nach dem Systemstart systemweit verfügbar. Die Sprungbefehle zur Programmierschnittstelle sind in Anwendungsprogrammen fest mit Verweisen auf die Prozeduren der Programmierschnittstelle versehen. Der Aufruf geschieht durch Softwareinterrupts (Traps) über eine Tabelle der Einsprungadressen, die von der Systemsoftware verwaltet wird (Traptable). Die Prozeduraufrufe können nicht geändert werden. Aber die Tabelle der Einsprungadressen kann manipuliert werden, so daß die Einträge auf die Ersatzprozeduren des Ausgabeanalysators zeigen.

Die Analyse der Ausgabe an der Programmierschnittstelle von Quickdraw genügt aber nicht, um die gesamte Benutzerschnittstelle zu erfassen. Wie oben beschrieben, muß auch der Aufruf anderer Routinen der graphischen Benutzerschnittstelle (Macintosh Toolbox) verfolgt werden, um die Fensterzuordnung für Elemente, wie Aufklappmenüs und Schieberegler zu gewinnen.

Die Zahl der ersetzten Prozeduren von Quickdraw beträgt 47 (siehe Anhang *Apple Macintosh Quickdraw Analysator*). Da die Parameter des Grafikkontextes abfragbar sind, müssen die Prozeduren zur Kontextmanipulation nicht verfolgt werden. Zusätzlich sind aber Informationen aus dem Aufruf von etwa 14 weiteren Prozeduren der Macintosh Toolbox notwendig, um indirekte Ausgaben zu erfassen. Der Analysator der Fensterverwaltung benutzt die gleiche Schnittstelle. Er bearbeitet 11 Funktionen. Soll die Mauszeigerform auch als Teil des Ausgabestroms übertragen werden, dann sind zwei Prozeduren

notwendig. Zwei zusätzliche Toolboxaufrufe waren notwendig, um in allen Anwendungsprogrammen die Ausgeben von Rollbalken zu erfassen.

4.2.6. Ausgabeanalyse auf Gerätetreiberebene

Nach der Verarbeitung von Grafikkommandos im Grafiksystem (siehe Kapitel 2.2.1) erfolgt die Ausgabe durch Bildschirmtreiber. Bildschirmtreiber bieten eine weitere gut definierte und publizierte Schnittstelle im Grafiksystem. Im Gegensatz zum Ersetzen der Programmierschnittstelle erlauben die meisten Grafiksysteme die Installation von Treibern für Grafikkarten. Ein Ausgabeanalysator kann deshalb mit expliziter Unterstützung der Systemsoftware als Bildschirmtreiber installiert werden. Er benutzt dann wiederum den originalen Bildschirmtreiber zur lokalen Ausgabe.

4.2.6.1. Der Ausgabestrom auf Gerätetreiberebene

Wenn alle Grafikausgaben über Bildschirmtreiber ausgegeben werden, ist der Ausgabestrom an der Aufrufschnittstelle des Bildschirmtreibers vollständig. Nur die Ausgabe von Programmen, die direkt in den Bildschirmspeicher schreiben, wird nicht erfaßt.

Der Ausgabestrom kann sich aber stark von dem an der Programmierschnittstelle des Grafiksystems unterscheiden. Das Grafiksystem paßt die Ausgabekommandos an die Bedürfnisse von Grafiktreibern an. Die Semantik des Ausgabestroms kann sich dadurch ändern und der Informationsgehalt sinken:

- Reduzierung von Grafikprimitiven auf die von Treibern unterstützen Basistypen. Die Menge der Basistypen kann sehr unterschiedlich sein. Bei manchen Grafiksystemen müssen alle Grafiktreiber einen Standardsatz von Basistypen unterstützen, während andere Grafiksysteme sich auf die Fähigkeiten des Grafiktreibers einstellen.
- Bei Grafiksystemen, die das Clipping nicht dem Treiber überlassen, werden Teile von Ausgabeoperationen oder ganze Ausgabeoperationen unterdrückt bevor die Kommandos an den Treiber gehen. Dies ist dann von Vorteil, wenn das Clipping vom Anwendungsprogramm beabsichtigt ist. Es ist nachteilig, wenn Fenster auf dem Quellsystem verdeckt sind, auf dem Zielsystem aber nicht. Beide Fälle treten zur Laufzeit häufig und nebeneinander auf.
- Virtuelle Koordinaten werden vom Grafiksystem für Grafiktreiber in Pixelkoordinaten umgerechnet. Dies kann von Vorteil sein für die Konvertierung des Ausgabestroms in andere Grafiksysteme, stellt aber einen Informationsverlust dar bei Anwendungen, die eine auflösungsunabhängige Darstellung verlangen. Für reine Sharinganwendungen, die nur Bildschirminhalte replizieren, sind Pixelkoordinaten im allgemeinen kein Nachteil.
- Bei mehreren Bildschirmen wird die Ausgabe vom Grafiksystem auf die Anteile reduziert, die auf den einzelnen Bildschirmen dargestellt werden. Ausgabeoperationen werden dadurch in mehrere Teile aufgespalten, die an unterschiedliche Instanzen des Ausgabeanalysators gehen. Sie werden zwar wieder zu einem Ausgabestrom zusammengefaßt, die komplette Ausgabeoperationen kann aber nicht einfach rekonstruiert werden.

Je höher der Funktionsumfang des Ausgabeanalysators als Grafiktreiber gegenüber dem Grafiksystem erscheint, desto geringer ist der Informationsverlust im Ausgabestrom. Dabei muß beachtet werden, daß neben der Duplizierung des Ausgabestroms auch der Originaltreiber bedient werden muß. Weicht der Funktionsumfang des Ausgabeanalysators von dem des lokalen Originaltreibers ab, dann muß dieser Unterschied durch den Ausgabeanalysator überbrückt werden. Der Ausgabeanalysator muß dazu Funktionalität darstellen, die auch schon im Grafiksystem vorhanden ist.

Die Fensterzuordnung ist an der Treiberschnittstelle im allgemeinen nicht direkt verfügbar. Damit fehlt eine wesentliche Komponente für den Ausgabestrom. Im Gegensatz zur Fensterzuordnung wird Bildschirmtreibern aber der Grafikkontext zur Verfügung gestellt, weil er viele Parameter enthält, die bei der Ausgabe berücksichtigt werden müssen. Oft ist die Fensterzuordnung über den Grafikkontext rekonstruierbar. Dazu drei Beispiele:

- Apple Macintosh
  Der Datentyp Fenster ist eine Erweiterung des Datentyps Grafikkontext. Die Referenzen auf Fenster und Grafikkontext sind deshalb identisch (siehe auch Kapitel 2.3.7).
- MS Windows 3.x

Es existiert eine - nicht offiziell dokumentierte - Liste, die Zuordungen zwischen Grafikkontexten und Fenstern enthält. Durch die Auswertung der enthaltenen Informationen kann über den DC das Fensterhandle bestimmt werden [Schöttner96].

- MS Windows NT
  Im 32-Bit GDI existiert eine Funktion (WindowFromDC), die bei Angabe eines DC-Handles das Handle des Fensters zurückliefert, dem der DC zugeordnet ist.

Viele Grafiktreiber führen Clipping von Ausgaben selbst durch. Das Grafiksystem muß dem Treiber deshalb Informationen über die relevanten Clippingbereiche zur Verfügung stellen. Da die Clippinginformation von Treibern bei jeder Ausgabeoperation ausgewertet wird, muß sie für Treiber in direkt verwendbarer einfacher Form vorliegen. Grafiksysteme bereiten deshalb die Clippinginformation auf, indem sie die Clippingbereiche des Grafikkontextes und die der Fensterverwaltung miteinander kombinieren. Grafiktreiber erhalten dann nur einen einzigen Clippingbereich, der die Schnittmenge aller relevanten Clippingbereiche darstellt.

Die Kombination aller Clippingbereiche ist eine der wichtigsten Aufgaben des Grafiksystems (der Grafikengine, Schicht 3a). Die Grafikengine bildet die Schnittmenge der Clippingbereiche des Grafikkontextes, eventuell des Fensters und des Clippingbereichs, der durch die Stapelfolge der Fenster entsteht. Bei Quickdraw ist der resultierende Clippingbereich in der Datenstruktur des Fensters (Grafports) enthalten (ClipRgn).

4.2.6.2. Umfang von Ausgabeanalysatoren auf Gerätetreiberebene

Wie in Kapitel 2.2.1 beschrieben, besteht die Bildschirmtreiberschnittstelle aus weniger Funktionen, als die Programmierschnittstelle des Grafiksystems. Die Treiberschnittstelle bietet Basistypen von Grafikprimitiven und Verwaltungsfunktionen. Die Zahl der Verwaltungsfunktionen ist sehr verschieden bei unterschiedlichen Grafiksystemen. Interessant sind hier aber nur die Funktionen für Grafikausgaben.

4.2.6.3. Beispiel Quickdraw GrafProcs

Die Grafiktreiberschnittstelle vom Quickdraw besteht aus 15 Prozeduren (Fensterverwaltung: 11 Prozeduren an der API Schnittstelle; zusätzlich 10 Prozeduren der Macintosh Toolbox). Verweise auf 10 der 15 Grafikausgabefunktionen sind in jedem Grafport enthalten. Bei allen Ausgaben werden jeweils die Funktionen verwendet, die in den Grafports eingetragen sind. Der Funktionensatz der Grafports wird von einer systemweiten Tabelle abgeleitet, in der die Adressen der Standardtreiber stehen. Ein Hersteller von Bildschirmkarten liefert Software mit, die diese systemweite Tabelle überschreibt und so den Satz von Grafiktreiberadressen für alle Grafports ändert.

Der Ausgabeanalysator kann wie ein Grafiktreiber installiert werden. Seine Funktionen werden so automatisch in allen Grafports eingetragen und verwendet. Damit werden Ausgabeströme aller Anwendungsprogramme über den Ausgabeanalysator geleitet. Alternativ kann der Funktionensatz der Grafports auch in jeder Instanz eines Grafports (bei seiner Erzeugung: InitPort) individuell gesetzt werden. So ist es möglich nur die Ausgabeströme einzelner Anwendungsprogramme über den Ausgabeanalysator zu leiten. Andere Anwendungsprogramme, deren Grafports nicht manipuliert werden, sind dann nicht beeinflußt.

4.2.6.4. Beispiel MS Windows NT Grafiktreiber

Zur Analyse am Windows NT Grafiktreiber müssen vom Ausgabeanalysator mindestens 7 Grafikfunktionen angeboten werden (Kapitel 2.4.1). Zusätzlich sind noch wenige Funktionen notwendig über die Ressourcen (Clippingbereiche und Füllmuster) übergeben und verwaltet werden. Der Ausgabeanalysator wird als Grafiktreiber installiert und verwendet den Originaltreiber für die Ausgaben auf dem lokalen Display. Die Analyse der Fensterverwaltung verwendet 8 Prozeduren.

Ressourcen werden dem Treiber immer dann übergeben, wenn sie für nachfolgende Ausgaben gültig werden. Werden Ressourcen nicht mehr benötigt, dann werden sie von der Grafikengine verworfen und nicht mehr referenziert. Der Treiber darf keine Referenzen speichern. Er erhält deshalb kein Signal, wenn die Ressourcen ungültig werden und nicht mehr benötigt werden. Dies führt dazu, daß der Ausgabeanalysator alle Ressourcen speichern muß ohne diese verwerfen zu können. Der Ausgabeanalysator kann aber zu jeder Zeit prüfen, ob eine Speicherreferenz noch gültig ist und so Ressourcen schließlich doch freigeben.

4.2.7. Ausgabeanalyse an der Netzwerkschnittstelle

Netzwerkfähige Grafiksysteme bieten im Zusammenhang mit der Datenübertragung einige weitere Schnittstellen, die verwendet werden können, um einen Ausgabestrom zu analysieren. Wie in Kapitel 2.2.1 beschrieben, werden Ausgabekommandos über eine Netzwerkverbindung übermittelt. Die Kommandos werden von einer Komponente des Grafiksystems zu Protokolldateneinheiten (PDU: Protocol Data Unit) zusammengestellt und an das Netzwerktransportsystem übergeben. Dieses verschickt den Strom von PDUs an einen Transportsystemendpunkt beim Endgerät. Vom Transportsystemendpunkt werden die Ausgabekommandos an die Grafiksystemsoftware weitergeleitet.

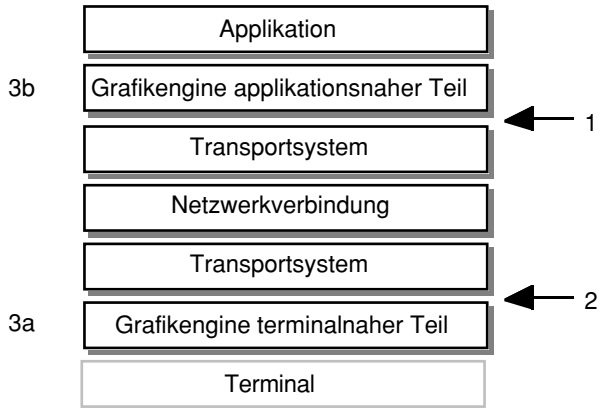


Abbildung Schichten im netzwerkfähigen Grafiksystem

> Bei netzwerkfähigen Grafiksystemen gibt es zwei Netzwerkschnittstellen. Im allgemeinen wird die Schnittstelle 2 für den Ausgabeanalysator verwendet, da hier ausschließlich der grafikrelevante Netzwerkverkehr anfällt. An der Schnittstelle 1 zwischen dem applikationsnahen Teil der Grafikengine und dem Transportsystem muß der grafikrelevante Netzwerkverkehr vom übrige Netzwerkverkehr getrennt werden. Diese Schnittstelle wurde deshalb bisher nicht verwendet.

Abgesehen von Hardwarelösungen, bei denen die Netzwerkverbindung manipuliert wird, bietet sich an zwei Schnittstellen die Möglichkeit, den Ausgabestrom zu analysieren. Jeweils eine auf der Seite des Anwendungsprogramms und eine auf der Seite des Terminals. Der Ausgabestrom ist an beiden Schnittstellen gleich, da er vom Transportsystem nur weitergeleitet wird.

Auf der Seite des Anwendungsprogramms kann der Ausgabestrom bei der Übergabe vom Grafiksystem zum Transportsystem analysiert werden. Dazu muß die Programmierschnittstelle des Transportsystems oder eine entsprechende Komponente (z.B. der Treiber des Netzwerkprotokolls) ersetzt werden. Diese Methode hat den Nachteil, daß der Ausgabeanalysator im Transportsystem die Grafikausgaben nicht auf einfache Art vom übrigen Datenverkehr unterscheiden kann. Er müßte deshalb jeden Datenverkehr untersuchen, um die Grafikausgaben zu identifizieren. Die Programmierschnittstelle des Transportsystems wird für alle Netzwerkausgaben verwendet, nicht nur für die des Grafiksystems. Das kann zu Geschwindigkeitsverlusten für den Netzwerkverkehr des gesamten Systems führen.

Die terminalseitige Schnittstelle zwischen Transportsystem und Grafiksystem ist besser zur Analyse des Ausgabestroms geeignet. Über diese Schnittstelle fließen nur Grafikausgaben. Der Ausgabestrom kann umgeleitet werden, indem die Netzwerkverbindung zu einem anderen Transportsystemendpunkt - dem des Ausgabeanalysators - geleitet wird (siehe [Gutekunst95]). Der Ausgabeanalysators dupliziert den Ausgabestrom und leitet ihn an den Transportsystemendpunkt der Terminalsoftware weiter.

4.2.7.1. Der Ausgabestrom an der Netzwerkschnittstelle

Der Ausgabestrom besteht aus einer Folge von PDUs des analysierten Grafikprotokolls. Er enthält alle notwendigen Informationen zur vollständigen Darstellung der Benutzerschnittstelle. Dies sind für jede Ausgabeoperation, wie in Kapitel 4.2.5.2, Grafikprimitiv, Modifikationsparameter und Grafikkontextparameter sowie die Fensterzuordnung. Allerdings werden in den Funktionen im anwendungsprogrammseitigen Teil der Grafiksystems Veränderungen am Ausgabestrom vorgenommen:

- Reduzierung von Grafikprimitiven, die an der Programmierschnittstelle angeboten werden, auf die Basistypen des Grafikprotokolls. Darunter fällt auch die Aufspaltung von Ausgabekommandos mit großen Datenmengen, wie das Kopieren von Pixmaps, in kleinere Einheiten.
- Zusammenfassung von Änderungen des Grafikkontextes, die an der Programmierschnittstelle sequentiell ausgegeben werden.
- Bei Grafikprotokollen, die nur geräteabhängige oder terminalangepaßte Farbspezifikationen vorsehen findet eine Wandlung von geräteunabhängigen in geräteabhängige Farbwerte statt.
- Anpassung der Ausgabe an die terminalseitig verfügbaren Ressourcen: freie Einträge der Farbtabelle, alternative Zeichensätze usw.

Einige der Anpassungen ändern die Semantik des Ausgabestromes. Die semantische Änderung wirkt sich aber nur dann negativ auf die anschließende replizierte Darstellung aus, wenn sich der Informationsgehalt des Ausgabestroms verringert, wie im Fall der terminalangepaßten Farbwerte. Der Ausgabeanalysator präsentiert sich dem Grafiksystem als Display. Je größer die Fähigkeiten dieses pseudo-Displays gegenüber dem anwendungsprogrammseitigen Teil der Grafiksystems erscheinen, desto geringer ist der Informationsverlust, da das Anwendungsprogramm.

Prinzipiell birgt die Netzwerkschnittstelle den Vorteil, daß der dort erhaltene Ausgabestrom für die Netzwerkübertragung durch das Application Sharing System gut geeignet ist. Im Gegensatz zum Ausgabestrom an der Programmierschnittstelle von lokalen Grafiksystemen werden im allgemeinen so wenig Daten bewegt, wie möglich.

4.2.7.2. Umfang eines Ausgabeanalysators an der Netzwerkschnittstelle

Die Netzwerkschnittstelle scheint sehr schmal, wenn alle Daten eines Grafikausgabestroms über eine Netzwerkverbindung übertragen werden. Tatsächlich werden aber Parameter vieler verschiedener Funktionen übertragen. Die Zahl von Funktionen ist allerdings geringer als an der Programmierschnittstelle. Sie ist der Zahl an der Treiberschnittstelle vergleichbar, d.h. eine Funktion für jeden Grafikprimitivtyp. Es gibt kaum funktionale Überlappungen.

4.2.7.3. Beispiel X Window System

Der Ausgabeanalysator wird als X-Displayserver installiert. Dies geschieht entweder durch Ersetzen des Displayservers auf dem standard TCP-Port (6000), oder durch die Umleitung der Transportsystemverbindungen zu dem Port des Ausgabeanalysators (Umgebungsvariable DISPLAY). Der Ausgabeanalysator kann nicht auf einfache Art zur Laufzeit eingeschaltet werden, da die Transportsystemverbindung normalerweise nicht umgeleitet werden kann.

Beim X Window System existiert mindestens eine Netzwerkverbindung je Anwendungsprogramm. Durch das Protokoll werden 17 Grafikausgabefunktionen, eine Funktion für Kontextmanipulation und 21 Funktionen für Ressourcemanagement kodiert. Zusätzlich trägt die Netzwerkverbindung 15 verschiedene Kommandos der assoziierten Fensterverwaltung.

4.2.8. Ausgabeanalyse an der Hardwareschnittstelle

Grafikausgaben erscheinen auf dem Bildschirm, indem der Prozessor des Displays schreibend auf den Bildschirmspeicher zugreift. Auf manchen Systemen ist dies Aufgabe des Systemprozessors (CPU), auf anderen die Aufgabe eines speziellen Grafikprozessors. Wenn die CPU Grafik selbst erzeugt, dann kann die Speicherverwaltungseinheit (MMU: Memory Management Unit) moderner Prozessoren verwendet werden, um die Zugriffe auf den Bildschirmspeicher zu verfolgen. Die MMU muß dazu so programmiert werden, daß ein schreibender Zugriff auf den Bildschirmspeicher eine Ausnahmebehandlung auslöst. Dem Prozessor wird ein sogenannter "Page-Miss" vorgespielt, also eine nicht eingeblendete Speicherseite. Die Ausnahmebehandlung kann dann den Schreibzugriff analysieren und daraus Informationen über Grafikausgaben ableiten. Der Ausgabeanalysator setzt so an der Schnittstelle zwischen Software und Hardware an.

Aufbauend auf dieser Methode können verschiedene Strategien implementiert werden. Dabei muß darauf geachtet werden, daß nicht die gesamte Ausgabe des Systems zu stark abgebremst wird. Zwei Beispiele:

1. Anhand der Adresse des Schreibzugriffs kann die Position im Bildschirmkoordinatensystem berechnet werden. Im Ausgabestrom ergibt sich eine Folge von Pixelausgaben, d.h. Zeichenoperationen für einzelne Pixel. Aus Geschwindigkeitsgründen werden zur weiteren Verarbeitung im Application

Sharing System zeitlich und räumlich begrenzte Gruppen von Pixelausgaben zu Pixmaps zusammengefaßt.

2. Unter der Annahme, daß jede Ausgabeoperation auf Treiberebene eine Sequenz von Speicherzugriffen auslöst, die nahe beieinander liegen, kann ein Großteil der Ausnahmebehandlungen (Interrupts) vermieden werden. Der erste "Page-Miss" dient der ungefähren Lokalisierung der Grafikausgabe. Daraufhin wird ein Timer gestartet und die MMU so programmiert, daß Schreibzugriffe in einer Umgebung des ersten Zugriffs keine Ausnahmebehandlung mehr auslösen. Nach Ablauf des Timers wird die Umgebung als Pixmap dem Ausgabestrom zugefügt und die MMU wieder für Zugriffe auf den gesamten Bildschirmspeicher sensibilisiert. Findet während dieser Zeit ein "Page-Miss" statt, dann wird der Timer gestoppt, die Pixmap ausgegeben und weiter verfahren, wie nach dem ersten "Page-Miss".

Im Idealfall kann die MMU so programmiert werden, daß der Bildschirmspeicher in kleine Kacheln zerlegt wird. Meistens sind aber Größe und mögliche Positionen von MMU-Seiten vorgegeben, so daß der Bildschirmspeicher nur scheibenweise aufgeteilt werden kann.

4.2.8.1. Der Ausgabestrom an der Hardwareschnittstelle

Der Ausgabestrom besteht nur aus einer Folge von Pixelausgaben oder Pixmapausgaben. Es fehlen wesentliche Informationen im Vergleich zu höheren Schnittstellen im Grafiksystem:

- Eine Rekonstruktion von Grafikprimitiven ist nicht einfach möglich.
- Farbwerte sind geräteabhängig.
- Die Fensterzuordnung ist verschwunden. Sie kann über ein Zuordnung zwischen den Koordinaten von Pixelausgaben und den Fensterpositionen nur unvollständig wiederhergestellt werden.
- Grafikausgaben in verdeckte Teile von Fenstern fehlen im Ausgabestrom.

Andere Methoden, die nach der Darstellung auf dem Bildschirmspeicher arbeiten, leiden unter den gleichen Einschränkungen. Dazu zählen vor allem Ausgabeanalysatoren, die Veränderungen im Bildschirmspeicher durch Checksummenbildung oder Auswertung von Redraw-Events feststellen.

4.3. Der Ausgabereplikator

4.3.1. Aufgabe des Ausgabereplikators

Der Ausgabeanalysator erzeugt aus den Informationen, die er aus der Ausgabe eines Anwendungsprogramms extrahiert, ein Abbild des Ausgabestroms des Anwendungsprogramms. Dieses Duplikat des Ausgabestroms leitet der Ausgabeanalysator weiter an den Ausgabereplikator. Der Ausgabereplikator übernimmt die Verteilung des Ausgabestroms über Netzwerke zu den entfernten Endgeräten. Eine Software in den Endgeräten nimmt die Daten entgegen und stellt die entsprechenden Ausgaben auf den angeschlossenen Ausgabegeräten dar. Die Ausgabe an das lokale Endgerät, bzw. das Endgerät an dem der Benutzer arbeitet, geschieht meistens direkt vom Ausgabeanalysator und ohne Umweg über den Ausgabereplikator. In Ausnahmefällen kann die direkte lokale Ausgabe entfallen. Der Ausgabeanalysator dupliziert in diesem Fall nicht den Ausgabestrom, sondern er leitet den gesamten Ausgabestrom an den Ausgabereplikator um. Das lokale Endgerät wird dann vom Ausgabeanalysator genauso, wie alle anderen entfernten Endgeräte behandelt.

Der Ausgabereplikator verteilt einen oder mehrere Grafikströme über Netzwerke zu einem oder mehreren entfernten Endgeräten bei denen die Ausgabe stattfindet.

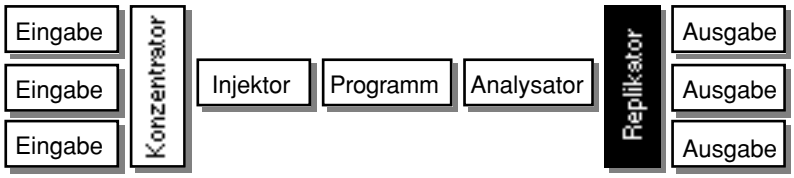


Abbildung Sharing System Komponenten - Replikator

Der Ausgabereplikator verteilt den vom Ausgabeanalysator extrahierten Grafikausgabestrom an die Endgeräte.

Bei der Übertragung von Grafikausgabeströmen über Netzwerke können hohe Datenraten anfallen (siehe auch Messung *Reduktion durch Pixmapkompression*). Die Architektur des Ausgabereplikators für den Transport von Grafikströmen ist deshalb von großer Bedeutung für die Effizienz eines Sharing Systems. Mehrere Varianten mit verschiedenen Eigenschaften werden in Kapitel 4.3.3 beschrieben.

Die Bedienung von Anwendungsprogrammen über Netzwerke hängt vom Zustand (z.B. aktueller Last, Störungen) und von den Eigenschaften (z.B. Verzögerung) der Netzwerke ab. In netzwerkfähigen Grafiksystemen wurden deshalb spezielle Vorkehrungen getroffen, um die Bedienung so weit, wie möglich unabhängig von den Netzwerkbedingungen zu machen. Neben den Grafiksystemen selbst sind auch Anwendungsprogramme für netzwerkfähige Grafiksysteme oft auf die Bedienung über Netzwerke zugeschnitten. Asynchrone Ausgabeoperationen ohne Bestätigungen tragen so z. B. erheblich zur flüssigen Ausgabe bei. Natürlich muß das Netzwerk dabei trotz allem, oft sehr hohe, Anforderungen bezüglich Durchsatz und Verzögerung erfüllen. Beim Sharing von Programmen treten in diesem Zusammenhang zwei Problembereiche auf:

1. Programme für lokale Grafiksysteme sind nicht auf die Bedienung über Netzwerke zugeschnitten. Sie wurden nicht unter Beachtung der bei Netzwerken gültigen Randbedingungen entwickelt, werden beim Application Sharing aber über Netzwerke bedient.
2. Beim Sharing mit mehreren Teilnehmern sind viele Netzwerkverbindungen mit möglicherweise stark unterschiedlichen Bedingungen beteiligt. Es muß entschieden werden, wie weit der Zustand einzelner Verbindungen die gesamte Konferenz und alle Teilnehmer beeinflußt (Kapitel 4.3.4).

Ein weiterer Faktor, der das Verhalten eines Sharing Systems beeinflußt, ist die Vorgehensweise des Ausgabereplikators bei Anfragen von Anwendungsprogrammen an das graphische Endgerät, bzw. die Grafiktoolbox. In fast allen Grafiksystemen können Anwendungsprogramme die Verfügbarkeit von Ressourcen und Endgeräteeigenschaften abfragen, um sich an die entsprechenden Verhältnisse anzupassen. Die Antwort auf solche Anfragen ist endgerätespezifisch. Bei mehreren angeschlossenen Endgeräten muß das Sharing System deshalb eine Antwort zurückgeben, die allen Endgeräten so gut, wie möglich gerecht wird (Kapitel 4.3.2).

Nach der Antwort des Ausgabereplikators an das Anwendungsprogramm liefert das Programm einen entsprechenden Ausgabestrom. Dieser Grafikstrom muß entsprechend den Möglichkeiten der einzelnen Endgeräte angepaßt werden. Unterscheiden sich die Endgeräte nicht nur bezüglich ihrer Ressourcen, sondern auch in der Art der Grafikkommandos, dann müssen die Kommandos im Grafikausgabestrom entsprechend konvertiert werden (Kapitel 5).

4.3.2. Reaktion auf Programmanfragen

Die oben genannten Abfrage-Anpassungs-Abläufe gibt es in vielen verschiedenen Zusammenhängen. Besonders wichtig ist die Abfrage der Verfügbarkeit von Ressourcen, wie Zeichensätzen oder Farbtabelleneinträgen. Daneben ermöglichen einige Grafiksysteme auch die Berechnung der Abmessungen von Textsequenzen. Anwendungsprogramme können damit Textfelder richtig positionieren und graphische Objekte an Texte anpassen. Da die Grafik auf mehreren Endgeräten gleichzeitig ausgegeben wird, existieren auch mehrere, möglicherweise verschiedene Antworten auf Anfragen von Anwendungsprogrammen. Der Ausgabeanalysator beobachtet solche Anfragen, wie alle Ausgabekommandos, und leitet sie an den Ausgabereplikator weiter. Da das Anwendungsprogramm eine Antwort erwartet muß der Ausgabereplikator eine entsprechende Antwort erzeugen. Er kann sich dabei von der echten Antwort der angeschlossenen Endgeräte leiten lassen, muß diese aber zu einer einzigen Antwort kombinieren, da das Anwendungsprogramm nur eine einzige Antwort erwartet. Es gibt hier drei verschiedene Strategien, die mit unterschiedlichem Arbeitsaufwand für den Ausgabereplikator verbunden

sind und zu verschiedenen Ergebnissen führen. Sie unterscheiden sich dadurch, wie (als welche Art von Endgerät) der Ausgabereplikator gegenüber Anwendungsprogrammen auftritt:

- wie eines der angeschlossenen Endgeräte,
- mit dem größten gemeinsamen Nenner aller Endgeräte,
- als ideales Endgerät.

4.3.2.1. Ausgabereplikator mit den Eigenschaften eines der Endgeräte

Die einfachste Methode gültige Antworten zu erzeugen besteht darin, nur die Antwort von einem der angeschlossenen Endgeräte an das Anwendungsprogramm zurückzugeben. Dazu gibt es zwei Möglichkeiten:

- der Ausgabereplikator leitet die Anfrage nur an eines der Endgeräte weiter. Das heißt, die Anfrage des Anwendungsprogramms wird aus allen Ausgabeströmen bis auf einen herausgefiltert. Oder
- die Anfrage wird, wie alle Ausgabeoperationen an sämtliche Endgeräte geschickt, dann aber nur die Antwort von einem einzigen Endgerät unverändert zurückgegeben. Eines der Endgeräte, das sogenannte führende Endgerät, repräsentiert so alle anderen.

Da Fragen nach der Verfügbarkeit von Ressourcen von einem Endgerät für alle anderen beantwortet werden, müssen alle Endgeräte gleich ausgestattet sein und unter ähnlichen Bedingungen laufen. Anwendungsprogramme, bzw. Grafikbibliotheken stellen sich auf die vorhandene Farbtiefe und produzieren einen entsprechenden Ausgabestrom (siehe X11: XPutImage). Bei Grafiksystemen ohne displayunabhängige Farbdarstellung muß deshalb die Farbtiefe aller Endgeräte übereinstimmen. Zur Textausgabe sollten auf allen Endgeräten die gleichen Zeichensätze bereitstehen, wie auf dem führenden Endgerät. Ist dies nicht der Fall, dann muß der Ausgabereplikator das Defizit ausgleichen, indem er den Ausgabestrom für einzelne Endgeräte entsprechend anpaßt. Werden einem Anwendungsprogramm vom führenden Endgerät Zeichensätze gemeldet, die auf einem anderen Endgerät nicht verfügbar sind, dann sorgt der Ausgabereplikator aufgrund seines Wissens über die tatsächlich verfügbaren Ressourcen dafür, daß die betreffenden Endgeräte mit Ersatzzeichensätzen arbeiten. Tut er dies nicht, entstehen Fehler, die den Ausfall von Teilen des Ausgabestroms bewirken.

Im Fall von Farbtabelleneinträgen wird ähnlich vorgegangen, wie bei den Zeichensätzen. Sind Farbtabelleneinträge nicht auf allen Endgeräten exakt gleich belegbar, was der Normalfall ist, dann paßt der Ausgabereplikator Farbwerte entsprechend den Verhältnissen der einzelnen Endgeräte an. Nicht belegbare Farbeinträge müssen auf andere oder mehrere andere Farben abgebildet werden während Farbindexwerte von verfügbaren, aber nicht identisch allozierbaren Farben in allen Ausgabeoperationen durch andere (verfügbare) Farbwerte ersetzt werden.

Im allgemeinen erfordern unterschiedliche Systembedingungen oder Systemkonfigurationen Anpassungsaufwand im Ausgabereplikator. Dieser Aufwand reicht von der schlichten Ersetzung von Zeichensatznamen bis zur Neukodierung von Pixmapausgaben für andere Farbtiefen. Die durchschnittliche Darstellungsqualität über alle Endgeräte nimmt bei Unterschieden zwischen Endgeräten im Vergleich zum Einzelbenutzerbetrieb fast immer ab.

4.3.2.2. Ausgabereplikator als größter gemeinsamer Nenner aller Endgeräte

Anpassungsaufwand im Ausgabereplikator kann minimiert werden durch die Strategie des größten gemeinsamen Nenners. Der Ausgabereplikator stellt sich gegenüber Anwendungsprogrammen mit Eigenschaften und Möglichkeiten dar, die für alle angeschlossenen Endgeräte gelten. Dazu werden Anfragen eines Anwendungsprogramms an alle Endgeräte weitergegeben. Aus den Antworten und mit Hilfe zusätzlicher Anfragen erzeugt der Ausgabereplikator durch Schnittmengenbildung eine gemeinsame Antwort aller Endgeräte an das Programm. Im Fall von identischen Endgeräten entspricht die Antwort der oben genannten Strategie. Bei Endgeräten mit unterschiedlichen Eigenschaften, speziell Farbtiefe und Zeichensätze, kann davon ausgegangen werden, daß die meisten Endgeräte einen gewissen Mindeststandard erfüllen. Heute sind fast alle Systeme mit einem ähnlichen Basissatz von Zeichensätzen ausgestattet. Endgeräte mit Echtfarbdisplays verfügen gewöhnlich auch über Darstellungsmodi mit indizierter Farbe, sowie schwarzweiß Darstellungen. Durch diese Vorgehensweise wird im allgemeinen die mittlere Darstellungsqualität über alle Endgeräte vermindert. Alle Ausgabeströme können aber weitgehend identisch sein. Sie müssen nicht vom Ausgabeanalysator speziell an die Verhältnisse der einzelnen Endgeräte angepaßt werden.

Die Farbindexwerte bei indizierter Farbdarstellung sind in diesem Zusammenhang besonders kritisch. Werden auch die Farbindexwerte wie beschrieben behandelt, d.h. nur die Farben verwendet (als verwendbar an Anwendungsprogramme zurückgemeldet), für die sich auf allen Endgeräten die gleichen Indexwerte belegen lassen, dann kann sich der Farbumfang sehr stark reduzieren. Es ist deshalb abzuwägen, ob in Grafiksystemen mit displayabhängiger Farbdarstellung für die Farben das Prinzip des größten gemeinsamen Nenners zugunsten von endgerätspezifischen Anpassungen aufgegeben wird.

4.3.2.3. Ausgabereplikator als ideales Endgerät

Die beste Darstellungsqualität für alle Endgeräte sowohl im Mittel, als auch für jedes einzelne wird erreicht, wenn der Ausgabereplikator gegenüber Anwendungsprogrammen als ideales Endgerät auftritt. Das heißt, der Ausgabereplikator beantwortet alle Anfragen bezüglich der Verfügbarkeit von Ressourcen positiv. Ein Anwendungsprogramm oder die Grafikbibliothek, die es benutzt, kann deshalb von idealen Ausgabebedingungen ausgehen. Im allgemeinen wird dann ein Ausgabestrom erzeugt, der so gut wie nur möglich den Anforderungen des Anwendungsprogramms entspricht. Ein Bildverarbeitungsprogramm wird z.B. einen Echtfarbmodus wählen und anschließend alle Farbspezifikationen in Echtfarbdarstellung ausgeben. Auf die Weise geht keine Information zwischen Anwendungsprogramm und Ausgabereplikator verloren.

Der Ausgabereplikator ist dafür verantwortlich den Ausgabestrom für jedes Endgerät individuell anzupassen. Im Einzelfall sind dazu, wie bei der Strategie des führenden Endgeräts, Anpassungen bei vielen Ausgabeoperationen notwendig. Für jedes Endgerät werden z.B. Farben in die verfügbaren Darstellungen umgesetzt und Zeichensätze verwendet, die den Anforderungen des Anwendungsprogramms so nahe kommen, wie möglich.

Anfragen über Aufstellungen von Listen verfügbarer Ressourcen werden durch die Vereinigungsmenge der Ressourcen aller Endgeräte beantwortet. Damit können auch spezielle Konfigurationen, wie zusätzliche, nicht standardmäßig verfügbare Zeichensätze auf einzelnen Endgeräten berücksichtigt werden. Es ergibt sich eine Doppelstrategie, gekennzeichnet durch

1. positive Antworten auf Fragen nach Verfügbarkeit einzelner Ressourcen und
2. Vereinigung der verfügbaren Ressourcen aller Endgeräte auf Fragen nach Aufzählung von Ressourcen.

Anfragen über Ressourceeigenschaften können ähnlich der Strategie des führenden Endgerätes beantwortet werden. Diese Anfragen werden an eines der Endgeräte weitergeleitet, bei denen die fragliche Ressource verfügbar ist, und von diesem beantwortet. Alternativ könnten auch alle Endgeräte befragt und aus den zugehörigen Antworten eine als Rückmeldung an das Anwendungsprogramm berechnet werden. Konflikte zwischen verschiedenen Antworten werden dabei abhängig von der Anfrage, z.B. durch Mehrheitsentscheidung, gelöst.

Eine Anfrage nach Textabmessungen, die von mehreren Endgeräten verschieden beantwortet wird, wird z.B. durch das Maximum aller erhaltenen Werte beantwortet. Damit wird sichergestellt, daß Anwendungsprogramme Text und graphische Objekte so positionieren, daß kein Text überschrieben wird. Ein Problem sind Textselektionen. Der für eine Selektion zu färbende Bereich wird nach der - vom Endgerät abgefragten - Dimension von Texten berechnet. Das Maximum der Werte ergibt bei allen Endgeräten den gleichen Bereich. Dieser Bereich kann auf allen Endgeräten, außer einem, über den fraglichen Text hinausreichen, falls Zeichensätze fehlen und durch andere ersetzt werden. Bei zu stark abweichenden Zeichensatzdefinitionen muß auf die Ersetzung durch Bitmaps ausgewichen werden (wie in Kapitel 5.3.3.6 beschrieben).

4.3.3. Replikatorarchitektur

Application Sharing kann auf viele verschiedene Arten und in sehr unterschiedlichen Anwendungsszenarien eingesetzt werden. Mögliche Anwendungen reichen von der Kleinbesprechung zwischen zwei Teilnehmern im lokalen Netzwerk bis zur Televortrag für viele Zuschauer über größere Entfernungen. Die Anforderung der Benutzer an ein Sharing System ist dabei aber immer gleich. Stets soll ein Sharing System dafür sorgen, daß mehr als ein Benutzer ein Anwendungsprogramm bedienen oder zumindest die Bedienung eines Programms beobachten kann. Die Beobachtung soll mit möglichst guter Qualität geschehen. Das heißt, die graphischen Ausgaben sollen bei allen Benutzern so auf dem jeweiligen lokalen Arbeitsplatzrechner erscheinen, wie bei einem rein lokalen Einbenutzerbetrieb des entsprechenden Anwendungsprogramms. Sowohl die räumliche Komponente (das grafische

Erscheinungsbild) des Ausgabestroms, als auch die zeitliche Komponente, soll der des Einzelplatzbetriebs entsprechen. Das heißt ein Sharing System soll so wenig Verzögerung, wie möglich einführen. Natürlich sind diese Forderungen nicht ohne Abstriche erfüllbar. So führt die Netzwerkübertragung im Vergleich zur rein lokalen Ausgabe immer eine Verzögerung ein. Wie gut die Anforderungen der Benutzer tatsächlich erfüllt werden können, hängt davon ab, wie gut die gewählte Architektur des Ausgabereplikators zum Anwendungsszenarium paßt.

Graphische Ausgabeströme von Anwendungsprogrammen haben, gemessen an der Netzwerkanbindung typischer Arbeitsplatzrechner, oft hohe Datenraten (Messung *Reduktion durch Pixmapkompression*). Die Art der Übertragung des Ausgabestroms von einem Arbeitsplatzrechner auf mehrere andere spielt deshalb eine wesentliche Rolle für die Effizienz eines Sharing Systems. Prinzipiell gibt es für die Übermittlung von Grafikströmen von einem auf mehrere andere Rechner zwei Methoden auf der Netzwerkebene:

1. mehrere Punkt-zu-Punkt-Verbindungen, die jeweils einen Ausgabestrom tragen und
2. eine eins-zu-viele-Verbindung (Multicast), die genau einen Ausgabestrom zu mehreren Empfängern transportiert.

In diesem Zusammenhang wird auch dann von einer Verbindung zwischen zwei Rechnern gesprochen, wenn sogenannte verbindungslose Protokolle eingesetzt werden. Der Verbindungskontext existiert dann nicht im Transportsystem, wohl aber im Sharing System (Kapitel "Dienste und Verbindungen" in [Froitzheim97]).

Eingabeströme haben meistens wesentlich geringere Datenraten, als Ausgabeströme (Kapitel 3.3.5). Die durch sie erzeugte Netzwerklast bekommt nur dann Gewicht gegenüber den Ausgabeströmen, wenn sehr viele Teilnehmer beteiligt sind. Da sehr viele Teilnehmer den Einsatz von Multicast bedingen, wird der Fall vieler Eingabeströme in Kapitel 4.3.3.3 behandelt.

Neben der Datenrate der Ausgabeströme spielt auch die Systemlast eine Rolle bei der Entscheidung für die Architektur eines Sharing Systems. Die zusätzliche Last, die das Sharing System auf den beteiligten Arbeitsplatzrechnern hervorruft, entsteht bei der Replikation von Ausgabeströmen. Wie in den Kapiteln 4.3.2 und 5 beschrieben, werden graphische Ausgabeströme in vielen Fällen angepaßt und konvertiert. Solche Manipulationen können Rechenleistung beanspruchen, die in der Größenordnung von Grafikausgaben auf dem lokalen Bildschirm liegt. Zusätzlich zu den Netzwerkoperationen verbrauchen die Manipulationen an Ausgabeströmen unter Umständen einen wesentlichen Anteil der Rechenleistung eines Arbeitsplatzrechners. Werden Grafikausgabeströme für mehrere Teilnehmer produziert, bremst das Sharing System einen Arbeitsplatzrechner deutlich ab. Es ist deshalb vorteilhaft, einen Teil der benötigten Verarbeitungsleistung in die Endgeräte zu verlegen (Kapitel 4.3.3.1). Dazu muß eine Komponente des Sharing Systems nahe beim oder im Endgerät ablaufen. Der Ausgabereplikator ist dann über die beteiligten Arbeitsplatzrechner verteilt. Soll allerdings eine Softwareinstallation beim Endgerät vermieden werden, dann kommt beim End-zu-End Betrieb nur eine monolithische, nicht verteilte Architektur in Frage (Kapitel 4.3.3.2). Stellt das Netzwerk die Replikatorfunktionalität zur Verfügung, dann kann die Rechenleistung zum Großteil aus Quellsystem und Zielsystem ausgelagert werden.

Bei mehr als zwei Teilnehmern, das heißt mehr als einem entfernten Empfänger eines Grafikausgabestroms, kann die Verwendung von Multicast vorteilhaft gegenüber mehreren Einzelverbindungen sein. Multicast spart Datenverkehr in der Nähe der Quelle eines Grafikstroms und Verarbeitungsleistung in der Quelle. Allerdings schränken einige Randbedingungen die Verwendung von Multicast ein. Die Netzwerkinfrastruktur für Multicast ist auch nach Jahren nur für einen kleinen Teil aller Internetbenutzer verfügbar. Der Großteil der potentiellen Benutzer von Application Sharing Systemen ist nicht an multicastfähige Netzwerke angeschlossen. Für viele Anwendungsszenarien muß der Ausgabereplikator deshalb ein Netzwerk von Punkt-zu-Punkt-Verbindungen benutzen. Darüber hinaus sind nicht alle Grafikausgabeströme multicastfähig. Das heißt sie enthalten Teile, die für jedes der angeschlossenen Endgeräte verschieden sein können. Die Voraussetzungen für Multicastverwendung im Ausgabereplikator und die damit verbundenen Vorteile werden in Kapitel 4.3.3.3 diskutiert.

4.3.3.1. Verteilte Architektur

Ein Application Sharing System ist ein verteiltes System. Einer oder mehrere Grafikausgabeströme werden auf einem Arbeitsplatzrechner erzeugt, über Netzwerke verschickt und auf anderen Arbeitsplatzrechnern dargestellt. Zur Darstellung eines Grafikstroms über Netzwerke sind dabei vier verschiedene Softwarekomponenten notwendig. Je eine Komponente zum

- Erzeugen,

- Senden,
- Empfangen und
- Darstellen

des Grafikstroms.

Beim Application Sharing wird der Grafikausgabestrom vom Ausgabeanalysator erzeugt. Der Ausgabereplikator sendet den Strom von einem Rechner und empfängt ihn auf - eventuell mehreren - anderen. Der empfangende Teil übergibt der Grafikstrom zur Darstellung an das lokale Grafiksystem, bzw. den Displayserver. Die beiden Teile des Ausgabereplikators verhalten sich wie Client und Displayserver eines netzwerkfähigen Grafiksystems. Sie unterscheiden sich von Client und Displayserver eines netzwerkfähigen Grafiksystems dadurch, daß der sendende Teil des Ausgabereplikators den Grafikstrom nicht selbst erzeugt, sondern ihn vom Ausgabeanalysator erhält und der empfangende Teil den Grafikstrom nicht selbst ausgibt, sondern an den Displayserver, bzw. das Grafiksystem des Endgeräts weiterleitet.

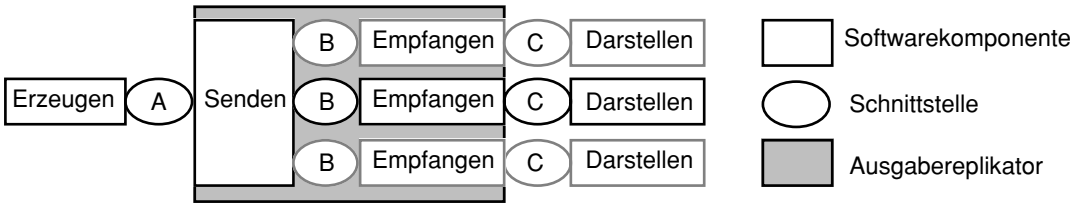


Abbildung Replikator Schnittstellen

Der Ausgabereplikator kommuniziert mit dem Erzeuger des Ausgabestroms (Ausgabeanalysator) und dem Darsteller (Display des Zielgrafiksystems) über die externen Schnittstellen (A, C). Die interne Schnittstelle (B) ist frei wählbar.

Bei netzwerkfähigen Grafiksystemen kann der empfangende Teil des Ausgabereplikators zusammen mit Schnittstelle B entfallen, weil der Displayserver von netzwerkfähigen Grafiksystemen selbst einen Grafikstrom empfangen und ausgeben kann. Diese Konfiguration entspricht dann einer monolithischen Architektur des Ausgabereplikators, die ohne zusätzliche Softwareinstallation beim empfangenden Arbeitsplatzrechner auskommt (Kapitel 4.3.3.2).

Schnittstellen des Ausgabereplikators

Neben eventuellen Steuerungsschnittstellen verwendet der verteilte Ausgabereplikator zwischen den vier oben genannten Komponenten drei verschiedene Datenschnittstellen. Die Schnittstelle A bildet den Eingang des Ausgabereplikators. Sie gleicht im wesentlichen der Schnittstelle an der ein Ausgabeanalysator im Grafiksystem ansetzt, um die Ausgabe eines Anwendungsprogramm zu beobachten (siehe Kapitel 4.2). Diese Schnittstelle ist Bestandteil des Grafiksystems für das ein Anwendungsprogramm entworfen ist. Der Ausgabeanalysator kann die Schnittstellenspezifikation des Grafiksystems in Einzelheiten geringfügig abändern. Oft werden im Ausgabeanalysator so z.B. Sequenzen von Kontextmanipulationen zu einzelnen Operationen zusammengefaßt. Damit kann der Ausgabeanalysator die Zahl der Prozeduraufrufe oder Protokolldateneinheiten verringern, bevor sie an den Ausgabereplikator weitergeleitet werden. Es ist aber nicht die Aufgabe des Ausgabeanalysators aufwendige Konvertierungen oder Anpassungen vorzunehmen. Im wesentlichen bildet die Schnittstelle A deshalb eine Schnittstelle des Grafiksystems ab. Sie ist damit festgelegt durch die entsprechende Spezifikation vom Hersteller des Grafiksystems.

Schnittstelle B ist eine Schnittstelle innerhalb des Ausgabereplikators. Über diese Schnittstelle kommunizieren die beiden Komponenten des verteilten Ausgabereplikators, um den Grafikausgabestrom von einem Arbeitsplatzrechner zu - eventuell mehreren anderen - zu senden. Da die Schnittstelle B nur von Komponenten des Ausgabeanalysators benutzt wird und nicht von externen Softwarekomponenten, ist sie bei der Implementierung das Sharing Systems frei wählbar.

Die Schnittstelle C ist wie A eine externe Schnittstelle des Ausgabereplikators. Sie bildet den Ausgang des Ausgabereplikators zum darstellenden Endgerät. Über die Schnittstelle C wird der replizierte Grafikausgabestrom an das Zielgrafiksystem geschickt. Die Spezifikation dieser Schnittstelle richtet sich deshalb nach dem Grafiksystem auf dem der Grafikstrom ausgegeben werden soll. Sie ist festgelegt durch den Hersteller des Zielgrafiksystems.

Ob die Schnittstellen A und C als Prozeduraufrufe oder als Protokollspezifikation vorliegen, hängt von den beteiligten Grafiksystemen ab. Wird die Ausgabe eines Anwendungsprogramm aus einem lokalen Grafiksystem, wie Win32, auf einem anderen lokalen Grafiksystem dargestellt, dann sind die Schnittstellen A und C Prozeduraufrufe. Dagegen sind beim Sharing innerhalb eines netzwerkfähigen Grafiksystems beide Schnittstellen als Grafikprotokolle definiert. Schnittstelle B, innerhalb des Ausgabereplikators, ist immer als Grafikprotokoll für die Netzwerkübertragung definiert.

Die freie Wahl der Schnittstelle B bietet die Möglichkeit das Sharing System auf das Anwendungsszenarium einzustellen. Ein besonderer Fall, die Verwendung von Multicast innerhalb des Ausgabereplikators für eine Anwendung mit vielen Teilnehmern, wird in Kapitel 4.3.3.3 beschrieben. Zwei andere wichtige Optionen sind

- Kompression und
- Verschlüsselung

des Grafikstroms.

Kompression im Ausgabereplikator

Die Reduzierung der Datenrate eines Ausgabestroms bei die Übertragung innerhalb des Ausgabereplikators ist in zwei Anwendungsfällen wichtig:

1. Application Sharing zwischen mehreren Teilnehmern im lokalen Netzwerk ohne Multicast
   Wie in Messung *Reduktion durch Pixmapkompression* gezeigt, beanspruchen Grafikströme oft Bandbreiten von einige hundert Kilobit bis zu einigen Megabit pro Sekunde. Selbst bei einem einzigen Anwendungsprogramm variiert die Datenrate sehr stark. Mit der benötigten Bandbreite ist jeweils die Spitze der Datenrate bei typischem Benutzerverhalten gemeint. Bei einer typischen Netzwerkanbindung von Arbeitsplatzrechnern mit einer Bruttorate von 10 Megabit pro Sekunde trägt das Netzwerk nur einen oder einige wenige unkomprimierte Grafikausgabeströme.
2. Application Sharing über Verbindungen mit relativ geringer Bandbreite
   Netzwerkanbindungen mit weniger als einem Megabit pro Sekunde bieten für viele Anwendungen zu wenig Bandbreite, als zum Transport von Grafikausgabeströmen notwendig. Besonders interessant sind die ISDN-Geschwindigkeiten von 64 und 128 kBit/s. Während einige Anwendungsprogramme, vor allem diejenigen für netzwerkfähige Grafiksysteme, über ISDN betrieben werden können (z.B. Terminalemulation xterm), gibt es viele Programme, deren Bandbreitenbedarf weit über 64 kBit/s hinausgeht. Gerade Programme, die für nicht netzwerkfähige Grafiksysteme entworfen wurden, erfordern einen sehr hohen Kompressionsgrad für ihre Grafikausgabeströme, sollen sie über ISDN betrieben werden (siehe dazu Kapitel 4.3.4.3).

Grafikausgabeströme von Anwendungsprogrammen bestehen aus verschiedenen Grafikprimitiven. Bei vielen Programmen wird aber ein großer Teil der Datenrate allein von Pixmaps erzeugt. Kompression von Pixmaps nimmt deshalb eine besondere Stellung bei der Reduktion der Datenrate ein. Für viele Anwendungen genügt es, bekannte Methoden zur Kompression von Standbildern, basierend auf Lauflängenkodierung (RLE), LZW [Welch84] [ZiLe77], oder Deflate [SchKa64] auf einzelne Pixmaps anzuwenden [NeHa88] [RaJo91].

Lauflängenkodierung

Run Length Encoding ist eine einfache Methode zur Redundanzunterdrückung. Bei der eindimensionalen Variante werden mehrere aufeinanderfolgende gleiche Werte durch ihre Lauflänge (n) und den Wert (w) repräsentiert. Die erreichbare Kompressionsrate hängt vom Wertebereich von n und dem Verhältnis der Längen von n und w ab. Im besten Fall ergeben sich sehr große Kompressionsraten von typischerweise 128 (bei n/w = 1/1 und 8 Bit-Werten für n und w). Interessant ist vor allem der schlechteste Fall, da dieser die totale Kompressionsrate stark herabsetzt, wenn auch nur Teile der Eingabedaten für RLE Kodierung ungeeignet sind.

Bei einem Verhältnis von 1:1 kann die Kompressionsrate einen Wert von 0,5 annehmen, d.h. die Datenmenge verdoppelt sich. Selbst dann wenn 80 % der Eingabedaten optimal für eindimensionale RLE Kodierung geeignet sind, die übrigen 20 % aber ungeeignet, d.h. stark variierend, ist die beste erreichbare Rate 2,46. Die Grenze zur Vergrößerung der Datenmenge liegt etwa bei 50 % Anteilen jeder Klasse. Bestehen die Daten aus Pixelwerten mit mehreren Farbkanälen, die direkt aufeinanderfolgend angeordnet sind, dann ist es vorteilhaft, jeweils den zusammengefaßten Pixelwert, d.h. die Kombination aller Kanäle, als Eingabe des RLE Kodierers zu verwenden. Bei 24 Bit Pixelwerten und bei 8 Bit Lauflänge erreicht die Kompression im schlechtesten Fall einen Faktor von 1,3, d.h. eine Erhöhung der Datenmenge um ein

Drittel. Die Grenze zwischen Verkleinerung und Vergrößerung der Datenmenge liegt dann bei 0,74 % Anteil ungeeigneter Pixelsequenzen. Zusammenfassend läßt sich sagen:

> RLE Kodierung darf nicht grundsätzlich auf alle Pixeldaten angewendet werden, sondern nur nach Überprüfung der Kompressionsrate im Einzelfall.

Verlustbehaftete Kompression

Auch verlustbehaftete Kompressionsalgorithmen, wie JPEG (Joint Photographics Experts Group) [JPEG91] [Wallace91] können für Pixmaps in Grafikausgabeströmen verwendet werden. Wird eine verlustbehaftete Kompression auf alle Pixmaps im Grafikausgabestrom angewendet, dann darf der Kompressionsfaktor nicht zu hoch sein, sonst werden graphische Elemente der Benutzerschnittstelle (z.B. Icons) nicht richtig wiedergegeben. Bei verlustbehafteter JPEG-Kompression auf 2 Bit pro Pixel treten keine sichtbaren Qualitätsverluste auf. 2 Bit bedeutet je nach Bildschirmtiefe einen Kompressionsfaktor von 4 - 12. Bei höheren Kompressionsfaktoren verursachen vor allem die hohen Farbgradienten der grafischen Elemente der Benutzerschnittstelle Kompressionsartefakte, während Bilder mit kontinuierlichen Farbverläufen scheinbar intakt bleiben.

Für die Beispielsitzung *Bildbearbeitung* ergibt sich bei Kompression der Pixmaps auf 2 Bit pro Pixel eine Reduktion des gesamten Grafikausgabestroms auf 25% der ursprünglichen Datenrate (Messung *Reduktion durch Pixmapkompression*). Bei der Beispielsitzung *Textbearbeitung im ASCII-Editor* bewirkt die Kompression von Pixmaps eine geringere Datenreduktion des Ausgabestroms (52 %). Trotz deutlich weniger Pixmapausgaben stellen Pixmaps in diesem Fall einen wesentlichen Anteil des gesamten Datenvolumens.

Selektive Kompression von Grafikparametern

Eine Methode zur temporalen Kompression von Grafikströmen wurde in [Danskin95] für das X-Protokoll vorgeschlagen. Diese Methode wird hier nicht nur auf einzelne PDUs angewendet, sondern auf die Datenstrukturen, die den gesamten Parametersatz von Ausgabekommandos enthalten (siehe Kapitel 2.5.3).

Neben Pixmaps können auch andere Teile von Grafikausgabeströmen komprimiert werden. Ein Grafikstrom ist aber nicht, wie eine Pixmap, eine homogene Einheit. Ein Grafikstrom besteht aus einer Folge von Dateneinheiten. Jede Dateneinheit setzt sich aus mehreren Feldern unterschiedlicher und eventuell variabler Länge zusammen. Die Felder enthalten Parameter von Grafikausgabekommandos, wie Koordinaten, Stiftfarbe oder Grafikprimitivtyp. Viele dieser Felder sind voneinander unabhängig. Datenreduktion, die auf dem gesamten Ausgabestrom arbeitet, zeigt nur deshalb Ergebnisse, weil Nullen unterdrückt werden können und viele Felder trotz großer Kardinalität nur kleine Zahlen enthalten. LZW-Kompression über einen typischen X11-Grafikausgabestrom ergibt deshalb nur einen Faktor von 2,4 [Danskin95].

Es bestehen aber deutliche Abhängigkeiten zwischen wiederkehrenden Instanzen gleicher Felder. In Grafikausgabeströmen gibt es oft Sequenzen in denen ähnliche Daten mehrmals wiederholt werden. Ein Beispiel hierfür ist Beispielsitzung *Textbearbeitung im ASCII-Editor*, wo nach jeder Tastatureingabe ein Zeichen ausgegeben wird. In den aufeinanderfolgenden PDUs ändert sich dabei nur die x-Koordinate und das Zeichen selbst. Alle anderen Felder bleiben gleich.

Es ist deshalb vorteilhaft, Sequenzen gleicher Felder als Eingabekanäle für einen Kompressor zu betrachten. In jedem Eingabekanal kann dann der Kodierer an die Daten angepaßt werden. Nach der Kompression werden die Daten aller Kanäle wieder zusammengefaßt und auf einer Netzwerkverbindung übertragen. Als Kanäle werden hier alle Parameter des erweiterten Parametersatzes aus Kapitel 2.5.3 herangezogen. Dabei ist es von Vorteil zusätzlich zwischen gleichnamigen Parametern für verschiedene Grafikprimitive zu unterscheiden, da sich in Grafikausgabeströmen oft Sequenzen von Grafikprimitiven (sog. Trains) mit jeweils leicht geänderten Koordinaten wiederholen. Ein Beispiel ist die Textbearbeitung im ASCII-Editor bei der sich das Löschen der Einfügemarke durch ein weißes Rechteck und die Ausgabe des neuen Zeichens abwechseln. Sowohl Rechteckkoordinaten als auch Textkoordinaten bewegen sich mit konstantem Increment. In diesem Fall arbeitet ein Differenzkodierer 2. Ordnung besonders effizient, wenn Rechteckkoordinaten und Textkoordinaten unabhängig voneinander betrachtet werden. In [Danskin95] sind Kompressionsresultate für verschiedene Kanäle im einzelnen aufgeführt.

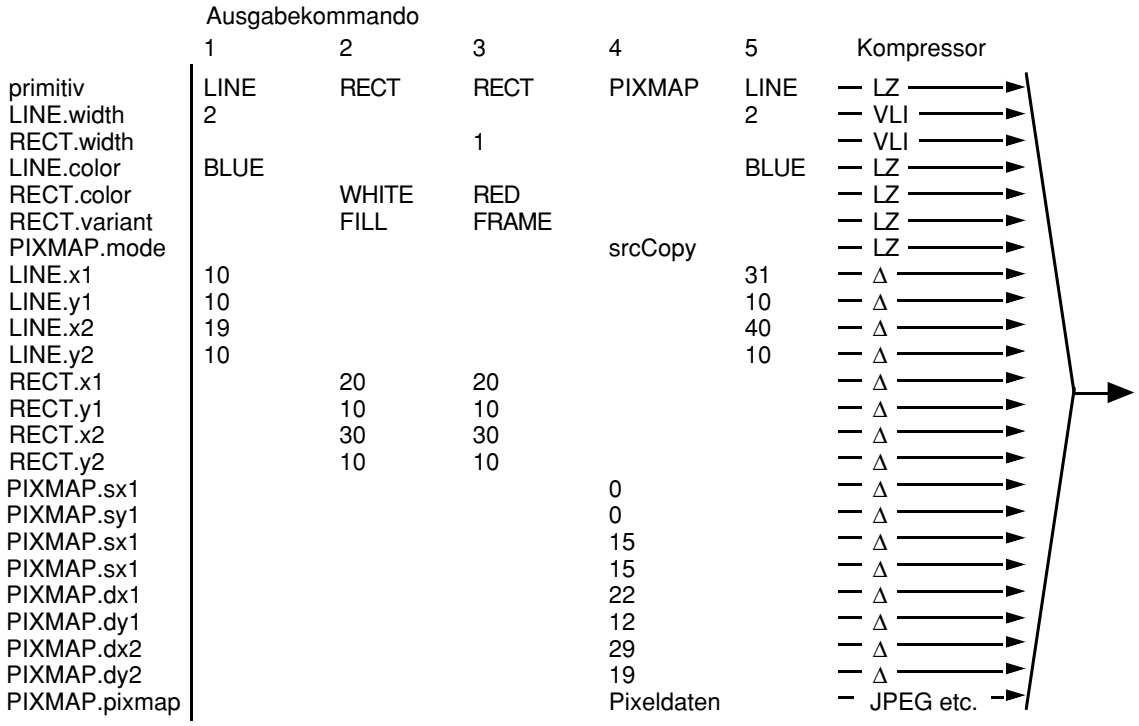


Abbildung Kanäle im Grafikstrom

Die relevanten Parameter der fünf Ausgabekommandos sind spaltenweise aufgetragen. Der Eingabestrom wird zeilenweise in Kanäle aufgeteilt, die Sequenzen gleicher Felder aufeinanderfolgender PDUs entsprechen. Jeder Kanal wird separat kodiert. Die Sequenz ist dem Beispiel aus Tabelle *Kommandos und Parametersätze* entnommen. Die Kodierer entsprechend den Empfehlungen aus [Danskin95], die für das X11-Protokoll entwickelt wurden.

Der Kodierer für Pixeldaten kann anhand verschiedener Grafikparameter gewählt werden. Bei sehr kleinen Pixmaps (Icons) ist ein RLE Kodierer zu bevorzugen. Größere Pixmaps werden LZW oder Deflate kodiert. Aber auch das Anwendungsprogramm kann in die Wahl des Kodierers eingehen. So kann man bei Bildbearbeitungsprogrammen davon ausgehen, daß größere Pixmaps jenseits der typischen Icongröße von 32x32 Pixel, zum bearbeiteten Dokument gehören. Damit kann ein verlustbehafteter Algorithmus, wie in JPEG, eingesetzt werden.

Temporale Kompression kann auch auf die anfallenden Pixmaps angewendet werden. Kandidaten für temporale Kompression werden identifiziert durch identische Koordinaten bei mehreren Ausgaben im gleichen Ausgabebereich. Gerade Redraw-Events lösen oft identische Ausgaben aus, die bei Differenzkodierung sehr gute Kompressionsraten ergeben. In [MPEG1-92] ist die dabei notwendige Wahl zwischen Bewegungskompensation, Differenzkodierung und Neukodierung beschrieben. Analysen von MPEG kodierten Bildsequenzen haben ergeben, daß die räumliche Redundanzunterdrückung einen Faktor von 5 - 20 zum Kompressionsfaktor beiträgt, während eine zeitliche Redundanzunterdrückung durch Differenzbilder und Bewegungskompensation die Datenraten um den Faktor 3, in Ausnahmefällen auch höher, reduziert [LaEf94]. Zusammen ergibt dies einen Kompressionsfaktor von 15 - 60 für Sequenzen von Bildern mit kontinuierlichen Farbverläufen.

Verschlüsselung im Ausgabereplikator

Beim Application Sharing werden Dokumente auf mehreren Displays dargestellt. Jeder, der den entsprechenden Grafikausgabestrom empfangen kann, erhält eine Ansicht des Dokuments. Das Dokument kann sogar in editierbarer Form auf der Ebene von Grafikkommandos rekonstruiert werden. Die Schnittstelle B des Ausgabereplikators an der Grafikausgabeströme über Netzwerke übertragen werden, ist ein Angriffspunkt für Ausspähung. Bei Anwendungen von Application Sharing im kommerziellen Bereich ist deshalb die Möglichkeit zur Verschlüsselung unverzichtbar. Da über diese Schnittstelle nur Komponenten des Sharing Systems kommunizieren, kann sie leicht - unabhängig von Kompatibilität mit externer Software - durch bekannte Verschlüsselungsverfahren geschützt werden. Siehe hierzu [RSA] [Zimmerman92].

Alle Daten eines Grafikausgabestroms müssen gleich gut geschützt werden. Der gesamte Grafikausgabestroms kann deshalb mit der gleichen Verschlüsselungsmethode bearbeitet werden. Im Gegensatz zur oben genannten Kompression sind keine auf Grafikausgaben angepaßten Verfahren notwendig.

Spezielle Bemerkungen zur verteilten Architektur

Beim Einsatz unter schwierigen Netzwerkbedingungen (z.B. TCP/IP über ATM mit restriktiver Flußkontrolle) hat sich gezeigt, daß ein verteilter Ausgabereplikator anfällig ist gegen den Abbruch von Netzwerkverbindungen. Während der Ausfall einer Netzwerkverbindung bei einem monolithischen Ausgabereplikator nur ein Endgerät vom Application Sharing System trennt, kann ein verteilter Ausgabereplikator in seiner Funktion gestört werden. Störungen auf einzelnen Verbindungen wirken sich dann auf das Gesamtsystem aus, wenn die Programmlogik des Application Sharing Systems sich implizit darauf verläßt, daß auf keiner der Netzwerkverbindungen Störungen auftreten. Ein Beispiel hierfür ist eine Pufferfreigabestrategie, die n aus n Erfolgsmeldungen benötigt. Ein anderes Beispiel ist blockierende Kommunikation mit nur einem Thread für alle Netzwerkverbindungen, die an einer der Netzwerkverbindungen blockiert und damit auch die anderen Verbindungen lahmlegt.

Es ist deshalb empfehlenswert, den Betrieb des Ausgabereplikators nicht von guten Netzwerkbedingungen abhängig zu machen.

Ein Application Sharing System soll tolerant gegenüber der netzwerkbedingten Unterbrechung einzelner Verbindungen sein. Ein Session Layer stellt virtuelle Verbindungen zur Verfügung und verwaltet reale Netzwerkverbindungen transparent für den Ausgabereplikator.

Ein Mechanismus, der unterbrochene Verbindungen selbständig wiederherstellt hat sich dabei bewährt. Die Netzwerksoftware des Ausgabereplikator versucht nach einer kurzen Verzögerung, die Netzwerkverbindung wieder herzustellen ohne, daß andere Module davon Kenntnis erhalten (Session Layer). Es wurden Verzögerungen im Sekundenbereich verwendet. Wichtig ist, daß sendender und empfangender Teil des Ausgabereplikators die verbindungsbezogenen Datenstrukturen behalten, so daß der Zustand der Sitzung bei Wiederherstellung der Verbindung unverändert ist. Dies muß vor allem bei objektorientierter Programmierweise beachtet werden, da häufig die Tendenz besteht, durch Vererbung den Verbindungskontext des Ausgabereplikators von Netzwerkklassen des Systems abzuleiten. Damit werden Instanzen des Verbindungskontextes im Ausgabereplikator synchron mit Netzwerkverbindungen angelegt und verworfen, was dem Prinzip der Toleranz gegen Verbindungsunterbrechung zuwiderläuft.

4.3.3.2. Monolithische Architektur

Im verteilten Ausgabereplikator dient die interne Schnittstelle B dem Transport von Grafikströmen über Netzwerke. Der empfangende Teil des Ausgabereplikators speist die Grafikströme in das Grafiksystem der Endgeräte ein. Bei netzwerkfähigen Grafiksystemen ist diese Konstruktion nicht notwendig. Die Software eines Arbeitsplatzrechner kann selbst Grafikströme über Netzwerke empfangen und darstellen. Um den Grafikausgabestrom eines Anwendungsprogramms darzustellen, ist keine Komponente des Sharing Systems auf dem Endgerät notwendig. Die empfangende Komponente des Ausgabereplikators beim Endgerät kann deshalb entfallen. Mit der empfangenden Komponente entfällt auch die Schnittstelle B innerhalb des Ausgabereplikators. Der Ausgabereplikator ist dann nicht mehr über das Netzwerk verteilt, sondern monolithisch. Er kommuniziert nur über die zwei fremdbestimmten Schnittstellen A und C (Kapitel 4.3.3.1). Die Schnittstelle C ist bei einem monolithischen Ausgabereplikators immer als Grafikprotokoll definiert. Dieses Protokoll muß der Ausgabereplikator verwenden, um die Grafikausgabeströme zu transportieren.

Die in Kapitel 4.3.3.1 genannten Möglichkeiten im verteilten Ausgabereplikator sind eng verbunden mit der internen Schnittstelle B. Fehlt diese, dann gehen viele Optimierungsmöglichkeiten verloren. Der Ausgabereplikator ist dann auf die Möglichkeiten des Grafikprotokolls der Schnittstelle C beschränkt. Wird z.B. das X-Windows Protokoll (X11R6) an der Schnittstelle C verwendet, dann ist weder die Kompression von Pixmaps oder anderen Datenfeldern in PDUs möglich, noch die Verschlüsselung des Grafikstroms.

Die monolithische Architektur hat aber nicht nur Nachteile, sondern bietet andererseits auch Vorteile im praktischen Betrieb.

1. Betrieb von zentraler Stelle,
2. Verwendung standardisierter Schnittstellen und
3. Einfachere Implementierung.

Betrieb von zentraler Stelle

Ein typisches Szenarium für Application Sharing ist eine Konferenz in der nur ein Teilnehmer einzelne Dokumente zur gemeinsamen Bedienung zu den anderen Teilnehmern exportiert. Es gibt in diesem Fall nur eine Quelle für Grafikausgabeströme, aber eventuell mehrere Senken. Software des Sharing Systems muß nur an einer Stelle, nämlich der Quelle, installiert werden. Die beteiligten Endgeräte empfangen die Grafikausgabeströme direkt. Sie sind für den Empfang von Grafikströmen nicht auf das Sharing System angewiesen. Das Sharing System muß im Verlauf der Konferenz auch nur von einer zentralen Stelle (meistens der Quelle) aus bedient werden. Die anderen Teilnehmer können gemeinsam Anwendungsprogramme bedienen, als ob sie jeweils lokal laufen würden. Sie sind aber zu keinem Zeitpunkt mit dem Betrieb des Sharing Systems befaßt. Weder mit der Installation, noch mit dem Start, bzw. der Aktivierung oder der Steuerung. Das entspricht dem Ablauf bei einer Telefonkonferenz bei der nur einer der Teilnehmer die Konferenz initiieren muß, während alle Teilnehmer mit Standardendgeräten gleichberechtigt teilnehmen sobald die Verbindung besteht.

Verwendung standardisierter Schnittstellen

Application Sharing soll nicht nur innerhalb einer Arbeitsgruppe oder einer Organisation (Unternehmen, Verwaltungseinheiten), sondern auch über die Grenzen von Organisationen hinweg eingesetzt werden. Die Ausstattung (Rechner und Software) potentielle Teilnehmer von Konferenzen mit Application Sharing Komponenten ist deshalb nicht homogen. Da Neuinstallationen und Versionsänderungen von Software nicht synchron laufen, ist es vorteilhaft für die Schnittstellen zwischen den Teilnehmern öffentliche Standards oder de facto Standards zu wählen.

Einfachere Implementierung

Ein monolithischer Ausgabereplikator muß nur auf der Betriebssystemplattformen implementiert werden, von der aus Anwendungsprogramme exportiert werden sollen. Die Endgeräte verwenden für Empfang und Ausgabe von Grafikströmen Standardsoftware, die nicht zum Sharing System gehört, sondern aus anderen Quellen stammt, wie z.B. kommerziell erhältliche X-Server. Im Gegensatz dazu muß bei einem verteilten Ausgabereplikator für jede Plattform, die an einer Application Sharing Konferenz teilnehmen soll, wenigstens der empfangende Teil des Ausgabereplikators implementiert werden. Ein Sharing System mit monolithischem Ausgabereplikator ist deshalb einfacher zu implementieren und zu warten, als ein verteiltes System.

4.3.3.3. Multicast

Multicast von Grafikausgabeströmen kann Netzwerkbandbreite sparen, wenn mehrere Teilnehmer einer Application Sharing Konferenz zumindest teilweise über die gleichen Netzwerkverbindungen mit der Quelle des Grafikstroms verbunden sind. Multicast stellt allerdings besondere Anforderungen an das Übertragungsprotokoll des Grafikstroms:

1. Displayunabhängige und informationsreiche Kodierung.
2. Ressource-IDs werden entweder vom Sender ohne Einfluß des Empfängers bestimmt oder Ressource-IDs im Protokoll überhaupt vermieden.
3. Aktive oder passive Fehlersicherungsmaßnahmen.
4. Abschnittsweise Vollständigkeit des Grafikstroms.

Auf diese Anforderungen wird in den folgenden Abschnitten einzeln eingegangen.

Displayunabhängigkeit

Multicast ermöglicht die gleichzeitige Ausgabe eines Grafikstroms auf vielen, u.U. hunderten oder tausenden von Endgeräten. Man kann deshalb davon ausgehen, daß sich unter den Endgeräten oft viele verschiedene Typen mit verschiedenen Konfigurationen und Ausstattungen befinden.

Wegen der Vielfalt der Konfigurationen muß der Grafikausgabestrom so kodiert sein, daß er von jedem Endgerät mit möglichst wenig Qualitätsverlust dargestellt werden kann. Jeder unnötige Qualitätsverlust soll vermieden werden. Natürlich ist es nicht möglich einen farbigen Grafikstrom auf einem schwarz/weiß Display ohne Qualitätseinbußen darzustellen, aber die begrenzten Fähigkeiten einzelner Endgeräte sollen nicht alle Endgeräte beeinträchtigen. Bei einer sehr großen Zahl von Endgeräten ist es nicht praktikabel, deren Fähigkeiten abzufragen, um eine gemeinsame Basis (z.B. die beste Bildschirmtiefe) zu finden. Der Ausgabereplikator muß eine Kodierung benutzen, die allen Endgeräten gerecht wird, ohne zu wissen, welche Endgeräte den Grafikstrom empfangen. Dazu nimmt er ein *virtuelles Endgerät* mit definierten Fähigkeiten (Displaytiefe, Zeichensätze) als Empfänger an. Der Grafikausgabestrom wird für die Übertragung auf die Fähigkeiten dieses Standardendgeräts zugeschnitten. Im Grafikprotokoll werden Ressourcen, wie Farben und Zeichensätze displayunabhängig spezifiziert. Die einzelnen Endgeräte können dann den so kodierten Grafikstrom auf die eigenen Fähigkeiten abbilden.

Für Ressourcen, wie Zeichensätze, die im Endgerät verfügbar sein müssen, ist der Ansatz des virtuellen Endgeräts nicht ausreichend. Es muß die Möglichkeit bestehen, über das Grafikprotokoll Ressourcen in das Endgerät zu laden. Wird z.B. in einem Grafikausgabestrom ein Zeichensatz verwendet, der nicht zu den (vom Ausgabereplikator erwarteten) Ressourcen des virtuellen Endgeräts gehört, dann soll der fehlende Zeichensatz als Teil des Grafikausgabestroms an die Endgeräte geschickt werden können.

Mehrere Standards zur displayunabhängigen, bzw. geräteunabhängigen Kodierung von Dokumenten (Postscript, Adobe Acrobat und z.B. HTML) verwenden ähnliche Mechanismen, wenn andere, als die standardmäßig verfügbaren Zeichensätze zu verwendet werden sollen. Genauso, wie ein auf Multicast basierender Ausgabereplikator, sind auch sie Austauschformate mit deren Hilfe Dokumenteninhalte auf sehr vielen Endgeräten mit unbekannter Konfiguration ausgegeben werden können. Sie sind das statische Analogon zum Multicast von Grafikausgabeströmen. Besonders Methoden, um Zeichensätze mit geringem Datenaufkommen zu kodieren und zu übertragen, können von diesen Standards entliehen werden .

Bei großen Empfängergruppen tritt der Ausgabereplikator gegenüber dem Anwendungsprogramm als ideales Endgerät auf.

Ressource-IDs

Identifikationsnummern (Ressource-IDs) werden von Grafikprotokollen benutzt, um Ressourcen zu referenzieren. Es gibt Ressource-IDs für Fenster, Zeichensätze, Pixmaps usw. Da alle angeschlossenen Endgeräte - abgesehen von Paketverlusten - exakt die gleichen Daten erhalten, müssen die Ressource-IDs im Grafikstrom für alle Endgeräte gelten. Daraus folgt, daß die Wahl der ID - auch bei nur wenigen Endgeräten - nicht den Endgeräten überlassen werden kann.

Beim X Window Protokoll [Xv0-90] werden Ressource-IDs von der Quelle des Grafikstroms, dem X-Client bestimmt. Der Client ist aber an einen Bereich gebunden, den der X-Server vorgibt. Durch die Wahl von verschiedenen Bereichen für jeden Grafikstrom sorgt der X-Server für eindeutige Ressource-IDs. Soll ein Grafikprotokoll multicastfähig sein, dann darf der Displayserver, beim Application Sharing der empfangende Teil des Ausgabereplikators, keinen Einfluß auf die Wahl von Ressource-IDs haben. Das Problem der Ressource-IDs ist ein wesentlicher Grund dafür, daß das X-Window Protokoll nicht multicastfähig ist. Für Multicast muß es erst adaptiert werden. Dazu werden die Ressource-IDs in den X-Protokolldateneinheiten durch virtuelle Ressource-IDs ersetzt, die für alle angeschlossenen Endgeräte gleich sind. Beim Endgerät findet dann eine Rückabbildung der virtuellen IDs auf die realen IDs statt [Gutekunst95].

Eine einfache Methode das Problem der Ressource-IDs zu umgehen ist die Vermeidung von Ressource-IDs, soweit dies möglich ist. Gibt es nur eine Ressource je Typ, dann ist eine Numerierung überflüssig. Das führt zu einem schlichteren Grafikprotokoll mit jeweils nur einer möglichen Pixmap, einem Füllmuster, einen Clippingbereich, etc. Für jede Ausgabeoperation sind diese Ressourcen entsprechend zu setzen. Es gibt dann keine Möglichkeit, Ressourcen für den späteren Gebrauch vorsorglich anzulegen. Dies kommt der Forderung nach abschnittsweiser Vollständigkeit des Datenstroms entgegen, die ebenfalls langlebige Ressourcen verbietet (siehe Kapitel 4.3.3.3 unten).

Fenster bilden eine Ausnahme. Beim Application Sharing ist es normalerweise erwünscht, daß Ausgaben von Anwendungsprogrammen nicht nur in ein Fenster, sondern in mehrere Fenster zu allen Teilnehmern verteilt werden. Jede Ausgabeoperation enthält deshalb eine Fensterzuordnung. Fenster werden auch als

Ressourcen behandelt, die im Endgerät angelegt werden. Fensterzuordnungen sind deshalb ebenfalls Ressource-IDs, auf die aber nicht generell verzichtet werden kann.

Das anhand des X Window Protokolls geschilderte Problem der Ressource-IDs kann durch sendergewählte Ressource-IDs vermieden werden. Jeder Sender achtet auf Eindeutigkeit seiner Ressource-IDs. Für das Endgerät wird die Eindeutigkeit durch die Verbindung der sendergewählten Ressource-IDs mit der (Referenz der) Netzwerkverbindung gewährleistet.

Multicastfähigkeit des Grafikprotokolls bedingt sendergewählte Ressource-IDs, die ohne Einfluß der Empfänger gewählt werden.

Fehlersicherungsmaßnahmen

Grafikströme sind sehr anfällig gegen unvollständige Paketauslieferung der Netzwerkschicht. Bei Grafikausgabeströmen von Anwendungsprogrammen haben Paketverluste unter Umständen große Auswirkungen auf die Darstellung der Benutzeroberfläche. Der Grund dafür liegt darin, daß Grafikausgabekommandos Differenzen zwischen Zuständen der Benutzeroberfläche darstellen. Sind diese Differenzen nicht vollständig und exakt, dann akkumulieren die Fehler in der Darstellung der Benutzeroberfläche. Dazu kommt, daß fehlerhafte oder fehlende Kommandos sich auf Sequenzen von folgenden Kommandos negativ auswirken können, indem sie diese verfälschen oder deren Ausgabe verhindern.

Grafikausgabeströme von Anwendungsprogramme enthalten keine Vollbilder, wie MPEG I-Frames [MPEG1-92], die akkumulierte Fehler wieder zurücksetzen. Grafische Objekte, die durch Ausgabekommandos erzeugt werden, können sehr lange existieren, ohne von anderen Objekten überdeckt zu werden. Sind solche Objekte wegen Paketverlusten fehlerhaft oder unvollständig dargestellt, dann bleibt der Fehler unter Umständen lange Zeit sichtbar. Die Auswirkungen von Fehlern in Grafikströmen sind nicht quantifizierbar, da die Wirkung von Ausgabekommandos sehr verschieden sein kann und vom Kontext abhängt.

In paketvermittelnden Netzen wird Multicast über unzuverlässige Protokolle auf der Netzwerkschicht betrieben. Da Multicast in paketvermittelnden Netzen ohne weitere Vorkehrungen die Eigenschaften der Netzwerkschicht (OSI Schicht 3) erbt, ist die Auslieferung der Daten an einzelne Empfänger nicht garantiert. Die Anfälligkeit von Grafikausgabeströmen gegen Paketverluste macht aber einen gewissen Grad an Fehlersicherung notwendig. Ein multicastfähiges Transportsystem muß sich nicht nur auf die Netzwerkschicht verlassen. Es kann zusätzliche Sicherungmechanismen vorsehen, die auch bei sehr vielen Empfängern einen gewissen Grad an Sicherheit für die Paketauslieferung geben. Die wichtigsten sind:

1. sendergesteuerte Vorwärtsfehlerkorrektur durch Redundanz [LaBöEf94] [Maier97] und
2. empfängergesteuerte Rekonstruktion [Jacobson97].
3. und eine Kombination beider Methoden [NoBiTo97]

Alle Methoden sind Lösungen oder Näherungslösungen für sogenannte Reliable-Multicastprotokolle.

Neben paketvermittelnden Netzen sind natürlich auch leitungsvermittelnde Netze für Application Sharing vorstellbar. Gerade die Anfälligkeit von Grafikströmen gegen Paketverluste prädestiniert Application Sharing durch Multicast als Anwendung auf leitungsvermittelnden Netzen.

Gibt es keine netzwerkbedingten Ausfälle in Grafikströmen, dann entfallen zwei Anforderungen an multicastfähige Protokolle, nämlich die Fehlersicherung und die abschnittsweise Vollständigkeit.

Abschnittsweise Vollständigkeit

Wie oben beschrieben, können einzelne fehlerhafte Teile im Grafikausgabestrom andere, folgende Kommando verfälschen oder deren Ausgabe verhindern. Die geschieht vor allem wenn Ressourcen für die spätere Verwendung angelegt oder bearbeitet werden. Bei nicht gesicherter Paketauslieferung muß vermieden werden, daß einzelne Fehler langfristig wirksame Folgen haben. Ressourcen, wie Pixmaps oder Clippingbereiche, dürfen deshalb nur für kurze Zeit angelegt werden. Eine lange Ressourcenspeicherung, wie im X Window System, bedeutet, daß der Grafikausgabestrom langfristig vollständig sein muß. Unter Multicastbedingungen mit unzuverlässiger Paketauslieferung können aber

trotz Fehlersicherungsmaßnahmen einzelne Verluste auftreten. Der Datenstrom ist also nur kurzfristig vollständig. Daraus folgt:

> Unter Multicastbedingungen ohne zusätzliche Sicherungmechanismen muß der Grafikausgabestrom innerhalb von kurzen Abschnitten vollständig sein, d.h. sich nicht auf länger zurückliegende Ressourcen abstützen.

Eine einfache Methode, um persistente Ressourcen zu vermeiden besteht darin, im Grafikprotokoll nur eine Ressource je Typ zuzulassen. Diese Ressource muß dann jeweils vor ihrer Verwendung neu gesetzt werden. Damit werden alte Zustände überschrieben und nie länger aufbewahrt.

Dies funktioniert allerdings nicht bei Fenstern. Es müssen jederzeit mehrere Fenster gleichzeitig existieren können. Gehen Fensterverwaltungskommandos verloren, z.B. ein Kommando zur Erzeugung eines Fensters, dann wird ein Teil (oder der gesamte) Ausgabestrom nicht dargestellt. Bei Netzwerkprotokollen, die eine Auslieferung nicht garantieren, ist deshalb prinzipiell eine zusätzliche Sicherungsmaßnahme auf der Ebene des Grafikprotokolls notwendig. Diese Sicherung soll abschnittsweise Vollständigkeit realisieren, so daß Übertragungsfehler oder -verluste nicht über die gesamte Dauer wirken.

Verallgemeinerte Stützbilder

Zur Realisierung abschnittsweiser Vollständigkeit werden sogenannte *verallgemeinerte Stützbilder* vorgeschlagen. Ein Stützbild ist in der Videokodierung ein Bild dessen Bilddaten komplett kodiert werden, ohne auf vorherige oder nachfolgende Bilder zurückzugreifen. Es dient als Referenzbild für andere Bilder bei denen nur die Differenzen zum Stützbild kodiert werden. Der Grafikausgabestrom eines Anwendungsprogramms kann aber, im Gegensatz zu Videos, aus mehreren Fenstern bestehen. Das Stützbild muß deshalb die Bildinhalte aller Fenster enthalten. Neben den Fensterinhalten ist auch der Zustand der im Display angelegten Ressourcen wichtig für den weiteren Betrieb. Das verallgemeinerte Stützbild muß also einen kompletten Abzug aller Ressourcen (einschließlich der Fenster) enthalten. Zur Information über die Fenster gehört neben den Inhalten auch deren Größe, Position. Um ein solches Stützbild zu erzeugen, muß der sendende Teil des Ausgabereplikators alle Ressourcen speichern, die in seinem Gegenstück, dem empfangenden Teil, angelegt werden. Die Fensterinhalte sind im Gegensatz zu den Ressourcen im sendenden Teil des Ausgabereplikators nicht einfach zu erhalten. Um Fensterinhalte für die Stützbilder mitzuführen, müßte der gesamte Ausgabestrom im sendenden Teil des Ausgabereplikators gerendert werden. Statt dessen wird ein Abzug der Fensterinhalte durch künstlich eingefügte Redraw-Events an das Anwendungsprogramm erzeugt.

> Ein verallgemeinertes Stützbild enthält den Zustand aller im Endgerät angelegten Ressourcen. Es wird zur Realisierung abschnittsweiser Vollständigkeit regelmäßig in den Ausgabestrom eingefügt. Die Frequenz kann dynamisch von der aktuellen Last und seiner Größe abhängen.

4.3.3.4. Eingaben und Rückmeldungen

Benutzereingaben und Rückmeldungen sollten auf zuverlässigen Netzwerkverbindungen übertragen werden. Gehen Benutzereingaben durch Netzwerkfehler verloren, dann kann der Fall eintreten, daß zusammengesetzte Eingaben nicht abgeschlossen werden. Es entsteht dann die sogenannte hanging-Button Problematik, die in Kapitel 4.4.2 beschrieben wird. Button-UP und Key-UP Meldungen *müssen* zuverlässig ausgeliefert werden.

4.3.4. Anpassung an Netzwerkbedingungen

Wird ein Anwendungsprogramm über Netzwerke interaktiv bedient, dann werden die Antwortzeiten nicht nur durch die Rechenzeit des Programms bestimmt, sondern auch durch den Umfang der Antwort und den Durchsatz des Netzwerks. Beides sind externe Faktoren. Sie liegen nicht im Einflußbereich des Sharing Systems. Sie setzen aber Randbedingungen, mit denen ein Sharing System arbeiten muß.

Grafikausgabeströme werden von Anwendungsprogrammen erzeugt. Der Ausgabeanalysator beobachtet den Grafikstrom und macht ihn verfügbar für die Verteilung durch den Ausgabereplikator. Für Menge und Art der Daten, die der Ausgabereplikator erhält, ist nur das Anwendungsprogramm, bzw. die verwendete Grafikbibliothek, verantwortlich. Probleme treten offensichtlich dann auf, wenn der erzeugte

Grafikausgabestrom die Kapazität des Netzwerks langfristig überfordert. Durch Variation der benutzten Anwendungsprogramme und Netzwerkanbindungen sind viele verschiedene Konfigurationen vorstellbar in denen Application Sharing gar nicht oder nur sehr stark eingeschränkt möglich ist. Auf der anderen Seite gibt es auch viele Situationen in denen gar keine netzwerkbedingten Probleme auftreten. Man kann davon aus gehen, daß die Anwender von Application Sharing die Größenordnung der Datenraten von Grafikausgabeströmen einschätzen können. Netzwerkanbindungen und Arbeitsplatzrechner werden deshalb im allgemeinen so dimensioniert und Anwendungsprogramme so gewählt, daß ein Application Sharing System prinzipiell funktioniert. Hier soll deshalb nur auf Grenzfälle eingegangen werden, die auftreten, wenn die Datenrate eines Grafikstroms in der gleichen Größenordnung liegt, wie die verfügbare Netzwerkkapazität oder diese häufig kurzfristig überfordert.

Die Datenrate von Grafikströmen in Application Sharing Systemen variiert sehr stark. Dies ist vor allem auf die ereignisgesteuerte Betriebsweise zurückzuführen, bei der Grafikausgaben jeweils als Antwort auf Benutzereingaben erzeugt werden. Diese Variation führt dazu, daß in manchen Fällen zeitweise sehr viele Daten zur Übertragung anstehen. Dann kann es passieren, daß die Puffer im sendenden Teil des Ausgabereplikators vollaufen und keine weiteren Grafikdaten mehr gepuffert werden können. Der Grund für solchen Stau ist dabei unwesentlich. Er kann sowohl im Endgerät, im empfangenden Teil des Ausgabereplikators, als auch beim Netzwerk liegen. Die Stockung wirkt sich ohne weitere Vorkehrungen direkt auf den Benutzer - auch den Benutzer beim Quellsystem - aus. Sobald der Ausgabereplikator keine Grafikdaten mehr verarbeiten kann, blockiert er die Ausgaben und damit den Ablauf des Programms. Werden Puffer frei, dann löst der Replikator die Blockierung. So wechseln sich Phasen normaler Bedienung mit Phasen des Stillstands ab. Der Stillstand kann dabei zu jeder Zeit erfolgen, so z.B. auch während Menüs aufgeklappt oder Rollbalken bedient werden. Stockender Betrieb an der Grenze der Belastbarkeit führt so zu erschwerter Bedienung oder sogar zu Bedienungsfehlern.

Übertragungsprobleme wirken sich unmittelbar auf die Bedienbarkeit von Programmen aus. Ein Application Sharing System muß für eine elastische Kopplung (siehe unten "Globale Bandbreitenadaptierung") zwischen Applikation und Netzwerk sorgen, um die Auswirkungen auf die Benutzer zu reduzieren.

Übertragungsprobleme und überlaufende Puffer können natürlich nicht in allen Fällen vermieden werden. Ein Application Sharing System muß aber Vorkehrungen treffen, um die Auswirkungen auf die Benutzer zu minimieren. Dazu werden hier zwei Verfahren vorgeschlagen:

- Globale Bandbreitenanpassung: Es wurde eine Pufferstrategie entwickelt, die eine elastische Kopplung realisiert (Kapitel 4.3.4.1)
- Individuelle Bandbreitenadaptierung: Bei mehreren Transportsystemverbindungen können die Netzwerkbedingungen für jedes Endgerät verschieden sein. Stromindividuelle Kodierung entkoppelt die Verbindungen untereinander und von den Ausgaben der Applikation (Kapitel 4.3.4.2)
- Einen Sonderfall, der in diesem Zusammenhang auch diskutiert werden soll, bilden Anwendungsprogramme, die in einem Application Sharing System unerwartet hohe Datenraten erzeugen. In einigen Fällen kann ein solches Verhalten erkannt und verbessert werden (Kapitel 4.3.4.3).

4.3.4.1. Globale Bandbreitenadaptierung

Das Ziel der globalen Adaptierung ist es, den Ablauf eines Grafikdaten-produzierenden Programms so abzubremsen, daß der Benutzer nur einen verlangsamten Betrieb bemerkt, aber keine Stockungen, wie oben beschrieben. Dazu müssen in erster Linie hohe Pufferfüllstände vermieden werden.

Die bei M-MaX [WoFrSchu95] verwendete Pufferstrategie kann sowohl bei einem verteilten (Kapitel 4.3.3.1), als auch bei einem monolithischen Ausgabereplikator (Kapitel 4.3.3.2) verwendet werden. Bei M-MaX liegt der monolithische Fall vor.

Abhängig vom Pufferfüllstand wird eine Verzögerung eingeführt, die den Ablauf des Anwendungsprogramms - und damit dessen Grafikausgaben - anhält. Die Verzögerung beginnt bei niedrigen Füllständen mit wenigen Millisekunden und wächst dann exponentiell an. Zusätzlich zu diesem Standardverfahren hat sich eine Modifikation bewährt, die dem dynamischen Charakter der Netzwerke entgegen kommt: Macht der Abbau des Pufferfüllstands gute Fortschritte, dann wird die Verzögerung abgekürzt. Der Fortschritt wird an einem Zielfüllstand gemessen, der vom Startwert abhängt. Bei

Füllständen über 60% ist die Zielmarke 50%. Darunter liegt die Zielmarke jeweils um ca. 10 % niedriger, als der Startwert.

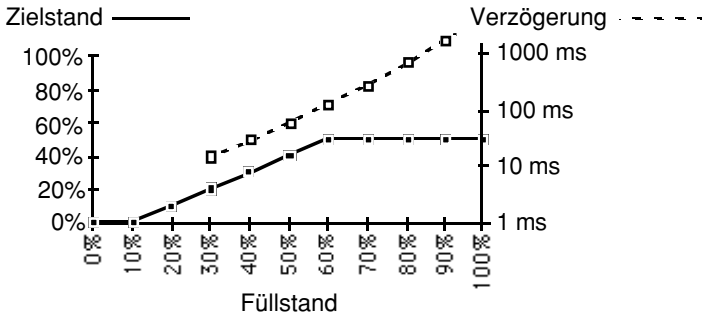


Abbildung Verzögerung (Pufferfüllstand)

Die gestrichelte Kurve zeigt ein Standardverfahren zur Überlaufvermeidung: Eine exponentiell wachsende Verzögerung, die dem Netzwerk Zeit gibt, die Pufferinhalte abzubauen. Die durchgezogene Linie beschreibt den einen Zielfüllstand. Wenn dieser Stand während der Verzögerung erreicht wird, dann wird der Programmablauf sofort fortgesetzt. Damit wird vermieden, daß trotz sich leerender Puffer große Verzögerungen eingefügt werden.

4.3.4.2. Individuelle Bandbreitenadaptierung

Oft sind bei mehreren Teilnehmern nur einzelne Transportsystemverbindungen überlastet. In diesem Fall kann es wünschenswert sein, alle anderen Teilnehmer, einschließlich des Teilnehmers am Quellsystem, mit unverminderter Geschwindigkeit arbeiten zu lassen. Der Teilnehmer, dessen Verbindung überlastet ist, soll trotzdem an der Sitzung teilhaben. Dabei soll die Sitzung für diesen Teilnehmer nicht langsamer ablaufen, sondern mit der gleichen Geschwindigkeit, aber verminderter Qualität. Die Sitzung soll bei allen Teilnehmern unabhängig vom Zustand ihrer jeweiligen Transportsystemverbindung gleich schnell (in Echtzeit) ablaufen und immer den aktuellen Zustand der Grafikausgaben darstellen.

Im Fall guter Netzwerkbedingungen und schneller Endgeräte tritt kein Überlastproblem auf. Alle anfallenden Grafikdaten werden dann direkt übertragen. Es ist keine Bandbreitenanpassung notwendig ist. Im folgenden sollen deshalb nur Bedingungen betrachtet werden, in denen die verfügbare Bandbreite kurzfristig oder langfristig nicht ausreicht, um den anfallenden Grafikausgabestrom ohne Pufferüberlauf zu übertragen.

Gleichzeitige Anpassung an verschiedene Netzwerkbedingungen

Prinzipiell unterscheiden sich die Bedingungen auf allen Transportsystemverbindungen. Wenn auf jeder Verbindung die verfügbare Bandbreite so gut, wie möglich, ausgenutzt werden soll, dann muß für jede Verbindung individuell ein Grafikstrom kodiert werden. Individuelle Kodierung für jede Verbindung ist aber nicht nur bei verschiedenen Netzwerkbedingungen notwendig, sondern auch bei verschiedenen Typen von Endgeräten. Besonders im Fall des monolithischen Ausgabereplikators, der direkt mit den Endgeräten (z.B. X-Terminals) verbunden ist, wird jeder Ausgabestrom auf die im Endgerät verfügbaren Ressourcen und dessen Eigenschaften (z.B. Byte-Ordering) zugeschnitten.

Im Fall begrenzter Bandbreite werden neue Grafikströme kodiert, die nicht exakt dem originalen, vom Ausgabeanalysator erhaltenen Ausgabestrom des Anwendungsprogramms entsprechen. Es sind Grafikströme, deren Bandbreite unter der für den vollständigen Grafikausgabestrom notwendigen Bandbreite liegen. Die Bandbreite kann wie bei allen Kompressionsverfahren für Grafiksequenzen, durch Redundanzunterdrückung und Auslassung herabgesetzt werden:

1. durch entsprechende Einstellung der Kompressionsrate für einzelne Grafikausgaben (Kapitel 4.3.3.1).
2. durch eine geringere Bildrate und Redundanzunterdrückung in Bildfolgen.

Optimierung durch Synergieeffekte

Trotzdem sind die Grafikströme natürlich miteinander verwandt. Sie sind alle vom Grafikstrom des Ausgabeanalysators abgeleitet und enthalten deshalb zum Teil die gleichen Grafikausgaben. Die Verwandtschaft der Grafikströme kann ausgenutzt werden, um einen wesentlichen Teil des

Kodierungsaufwands einzusparen. Dazu wird das in [WoFr97] [Kaufmann99] für Videoströme vorgestellte CE/SC (Component Encoding/Stream Composition) Verfahren eingesetzt:

Der zu kodierende Grafikstrom wird in Komponenten (z.B. Teilbilder) zerlegt. Die Komponenten werden kodiert, bzw. komprimiert und in einem Komponentenspeicher abgelegt. Zu jeder Zeit kann die gesamte Darstellung eines Anwendungsprogramms aus dem Komponentenspeicher rekonstruiert werden. Der Komponentenspeicher enthält dafür immer mindestens ein aktuelles *verallgemeinertes Stützbild* (Kapitel 4.3.3.3).

Generationenkarte

Eine der Komponenten ist eine Generationenkarte, die für alle Ausgabebereiche (bis auf Pixelebene) und Ressourcen Zähler enthält. An der Generationenkarte ist das Alter jedes Teils des Ausgabebereichs, bzw. jeder Ressource abzulesen. Die Stromkodierer führen jeweils einen Zähler, der die Generation des letzten kodierten Grafikkommandos enthält. Bei Videoströmen, wie MPEG und GIF wird eine fortlaufende Bildnummer als Generation verwendet; bei X11 die Sequenznummer. Anhand der eigenen Generationsnummer und der Generationenkarte wählt jeder Kodierer die für das nächste Grafikkommando notwendigen Komponenten aus dem Komponentenspeichers aus. Nur Generationen, die jünger sind, als der Generationenzähler des Stromkodierers müssen bearbeitet werden. Ältere Generationen sind bereits übertragen worden. Wenn die Bandbreite nicht ausreicht um jede Generation, d.h. jedes Ausgabekommando des originalen Grafikausgabestroms zu übertragen, werden dabei automatisch Generationen übersprungen, die von jüngeren verdeckt worden sind und so nichts zum aktuellen Zustand eines Ausgabestroms beitragen.

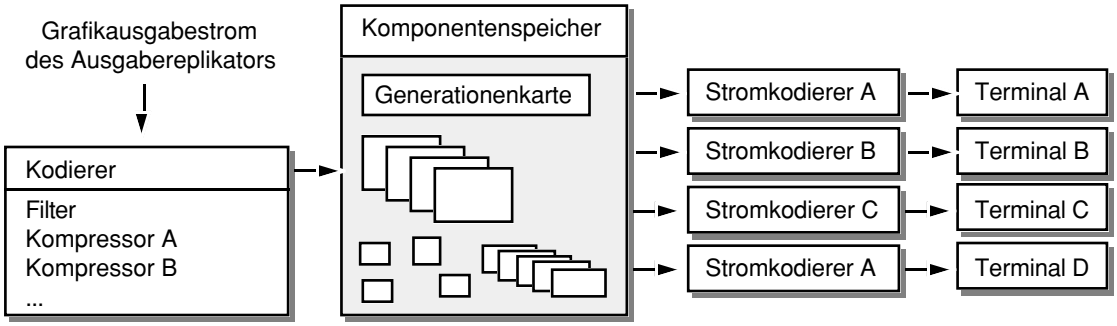


Abbildung CE/SC

Das CE/SC Verfahren wird zur Kodierung verwandter Grafikströme eingesetzt. Die Stromkodierer (rechts) verwenden eine gemeinsame Datenbasis von Stromkomponenten. Die Komponenten werden zu individuell angepaßten Grafikströmen zusammengesetzt. Die Anpassung bezieht sich dabei sowohl die verfügbare Bandbreite, als auch auf die Fähigkeiten der Endgeräte (verfügbare Dekompressoren und Ressourcen). Große Teile der Grafikströme werden nur einmal kodiert, bzw. komprimiert und von allen Stromkodierern gemeinsam verwendet.

Garbage Collection

Komponenten werden erst dann freigegeben, wenn sie von keinem Stromkodierer mehr verwendet werden können, d.h. wenn der Generationszähler aller Stromkodierer jünger ist, als die Generation der Komponente. Damit ist im Extremfall ein sehr großer Komponentenspeicher notwendig. In der Praxis wird die notwendige Größe beschränkt durch

- Verdeckungen
  Bei typischen Grafikströmen bleiben nur kleine Teile der Ausgabebereiche für lange Zeit unverändert sichtbar. Der größte Teil wird im Verlauf der Sitzung ungültig, d.h. irrelevant für den aktuellen Zustand des Anwendungsprogramms (Bsp.: temporäre Dialogboxen, veränderliche Dokumenteninhalte).
- die Größe eines Stützbilds.
  Beim Standardverfahren werden alle Ausgabekommandos gepuffert, wenn die anfallende Datenrate die verfügbare Bandbreite übersteigt. Die obere Schranke für den notwendigen Pufferspeicher ist deshalb das gesamte Datenvolumen des Ausgabestroms. Beim CE/SC Verfahren wird nicht in jedem

Fall der gesamte Ausgabestrom übertragen, sondern jeweils der aktuelle Zustand (verallgemeinertes Stützbild). Der notwendige Puffer ist deshalb beschränkt durch die Größe des verallgemeinertes Stützbilds und der gerade in Übertragung befindlichen Ressourcen.

Eigenschaften des CE/SC Verfahrens

Die Vorteile des CE/SC Verfahrens:

1. Reduzierung des Kodierungsaufwands,
2. Entkopplung von Grafikstromquelle (Ausgabeanalysator) und -senke (Endgerät) und
3. Begrenzung der Puffergröße

werden mit Kosten für die Verwaltung der Generationenkarte erkauft. Der Verwaltungsaufwand ist vergleichbar mit der lokalen Bildschirmausgabe. Der Einsatz des CE/SC Verfahrens lohnt sich daher nicht bei drei oder weniger entfernten Endgeräte. CE/SC ist sinnvoll bei kleinen Gruppen ab vier Teilnehmern.

Das CE/SC Verfahren wurde für Videosequenzen implementiert. Es ist aber prinzipiell für alle Displays einsetzbar, bei denen nur der jeweils neueste Zustand interessant ist. Das CE/SC Verfahren kann deshalb auch auf Grafikströme angewendet werden, die nicht nur Pixmaps, sondern auch andere Grafikprimitive enthalten.

4.3.4.3. Programme mit überhöhter Datenrate

Programme, die für lokale Grafiksysteme entwickelt wurden, sind oft nicht für die Bedienung über Netzwerke geeignet. Sie erzeugen zur Darstellung ihrer Benutzerschnittstelle Grafikausgabeströme mit höheren Datenraten, als notwendig. Bei manchen Anwendungsprogrammen beträgt der Unterschied zwischen notwendigem und aktuellem Datenvolumen sogar mehrere Zehnerpotenzen. Trotzdem muß ein Application Sharing System so weit wie möglich versuchen, auch das Sharing solcher Programme zu ermöglichen.

Netzwerkfähigen Grafiksystemen und lokale Grafiksysteme haben unterschiedliche Randbedingungen für grafische Ausgaben. Bei netzwerkfähigen Grafiksystemen ist der begrenzende Faktor meistens die Übertragungsleistung des Netzwerks. Es gibt nur wenige Anwendungen in denen die Übertragungsleistung eines Netzwerks die Darstellungsgeschwindigkeit eines Arbeitsplatzrechners übersteigt. Nur wenn viele kurze Grafikkommandos jeweils große Bildschirmbereiche bedecken, dann begrenzt die Leistung des Endgerätes die Ausgabegeschwindigkeit. Programmierer, die Anwendungsprogramme für netzwerkfähige Grafiksysteme entwickeln sind sich dieser Randbedingung bewußt. Die Programme werden so entworfen, daß möglichst wenig Netzwerkverkehr stattfindet. So werden zum Beispiel Ressourcen im Endgerät angelegt, die das Programm dann mehrfach für die Ausgabe verwenden kann.

Bei Programmen für lokale Grafiksysteme begrenzt nur die Geschwindigkeit des Endgeräts, bzw. des lokalen Grafiksystems die Darstellungsgeschwindigkeit. Die Übertragung des Grafikausgabestroms findet komplett im Speicher des Arbeitsplatzrechners statt. Sie ist deshalb im Vergleich zum Aufwand für die Darstellung (das Rendering) vernachlässigbar. Auf die Datenmenge muß der Programmierer nicht achten. Der Programmierer kann sogar die Darstellungsgeschwindigkeit erhöhen, indem er mehr Daten erzeugt.

Mit dem Sharing von Anwendungsprogrammen werden die Randbedingungen netzwerkfähiger Grafiksysteme auf Programme übertragen, die für lokale Grafiksysteme entworfen sind. Für Ausgabeoperationen, die diesen neuen Randbedingungen widersprechen gibt es mehrere Gründe:

- Mehrfache gleiche Ausgaben,
- Schnelle Darstellung komplexer Grafiken,
- Vereinfachung von Softwarestrukturen bzw. Ersatz für fehlende Ausgabemöglichkeiten und
- Falsche Wahl bei äquivalenten Kommandos (oder Gedankenlosigkeit).

Durch eine Analyse und Umwandlung des Grafikausgabestroms kann das Datenvolumen zum Teil wieder reduziert werden. Dies ist Aufgabe des Ausgabereplikators, nicht des Ausgabeanalysators, denn die entsprechenden Umwandlungen hängen von den Möglichkeiten des zum Transport des Grafikstroms verwendeten Protokolls ab. In den folgenden Abschnitten werden die genannten Punkte diskutiert und, soweit möglich, Maßnahmen angeboten.

Mehrfache gleiche Ausgaben

Wenn abzusehen ist, daß während des Programmablaufs mehrfach die gleiche Ausgabe wiederholt wird, dann wird diese Ausgabe oft vorbereitet indem eine entsprechende Pixmap, u.U. schon bei der Implementierung, angelegt wird (z.B. Icons). Bei netzwerkfähigen Grafiksystemen wird diese Pixmap meistens im Endgerät gespeichert, damit sie nicht mehrfach übertragen werden muß. Dagegen legen Programme für lokale Grafiksysteme derart vorbereitete Ressourcen im eigenen Speicher ab. Bei jeder Ausgabe dieser Pixmaps erscheint dann die Pixmap als Teil des Ausgabestroms. Der Ausgabeanalysator eines lokalen Grafiksystems erhält also Pixmaps mehrfach zu verschiedenen Zeiten, während sie in netzwerkfähigen Grafiksystemen nur einmal übertragen und dann für die Ausgabe mehrfach referenziert werden. Ähnliches passiert mit anderen Ressourcen, wie z.B. Clippingbereichen.

Als Gegenmaßnahme müßte der Ausgabestrom im Ausgabereplikator vollständig gerendert werden. Ausgabekommandos werden nur dann übertragen, wenn sich der Bildinhalt durch die Ausgaben ändert. Werden keine Pixel geändert, dann werden Ausgabekommandos des Anwendungsprogramms im Ausgabereplikator unterdrückt. Dieses Verfahren ist allerdings sehr aufwendig und wurde bis jetzt nicht implementiert.

Schnelle Darstellung komplexer Grafiken

In vielen Fällen kann eine Pixmap schneller auf dem Bildschirm dargestellt werden, als die gleiche Grafik aus grafischen Primitiven aufgebaut werden kann. Soll eine Grafik in sehr kurzer Zeit dargestellt werden, so daß sie für den Benutzer auf einen Schlag erscheint, dann wird sie oft als Pixmap erzeugt und durch einen einzigen Ausgabebefehl dargestellt. Die Pixmap wird bei einem netzwerkfähigen Grafiksystem im Endgerät aus grafischen Primitiven erzeugt. Nur die grafischen Primitive werden dabei übertragen. Die Pixmap muß nicht über das Netzwerk übertragen werden, sondern wird zur Darstellung nur referenziert. Bei lokalen Grafiksystemen erscheint die Pixmap dagegen zur Ausgabezeit im Grafikstrom. Damit trägt eine Methode, die im lokalen Fall die Ausgabe beschleunigt dazu bei, daß die Ausgabe über Netzwerke langsamer wird.

Es wurden keine Maßnahmen entwickelt, um diesen Fall zu erkennen und das Verhalten des Ausgabereplikators zu optimieren.

Vereinfachung von Softwarestrukturen

Die Darstellung sämtlicher Grafik als Pixmaps kann zur Vereinfachung von Softwarestrukturen beitragen. Einige Anwendungsprogramme verwenden die Clipping-Fähigkeit von Grafiksystemen intensiv, um Teile von Ausgabeoperationen auszublenden. Programme müssen dann nicht berechnen, welcher Teil der Ausgabe wirklich sichtbar wird. Diese Aufgabe übernimmt das Clipping des Grafiksystems. Wird Clipping allerdings auf große Pixmaps angewendet und dabei die Ausgabe großer Teile unterdrückt, dann werden unter Umständen wesentlich mehr Daten transportiert, als notwendig.

Als Maßnahme zur Unterdrückung überflüssiger Daten analysiert der Ausgabereplikator deshalb alle Ausgabekommandos für Pixmaps unter Beachtung des aktuellen Clippingbereichs und überträgt nur die Teile, die tatsächlich auf den Endgeräten sichtbar werden.

Realisierung fehlender Ausgabemöglichkeiten

Bildbearbeitungsprogramme und Grafikprogramme bieten heute einen Funktionsumfang, der über eine reine Abbildung der Programmierschnittstelle des verwendeten Grafiksystems hinausgeht. Wenn z.B. graphische Primitive bei der Ausgabe durch Effekte modifiziert werden sollen, die ein Grafiksystem nicht bietet, dann muß dieses Defizit durch das Anwendungsprogramm ausgeglichen werden. Spezielle Effekte werden realisiert, indem das Anwendungsprogramm vor der endgültigen Ausgabe noch Veränderungen an Grafikausgaben vornimmt. Grafikobjekte werden deshalb im Speicher des Anwendungsprogramms in eine Pixmap ausgegeben, die nicht auf dem Bildschirm sichtbar ist. Dort wird die Pixmap modifiziert und dann als Pixmap ausgegeben.

Falsche Wahl bei äquivalenten Kommandos

In vielen Fällen gibt es mehrere mögliche Ausgabekommandos, die alle die gleiche grafische Ausgabe erzeugen. Programmierer können dann die Methode für die Ausgabe auswählen. Soll z.B. ein Bildbereich, der bereits auf dem Bildschirm dargestellt wird, verschoben werden, so bieten die meisten Grafiksysteme zwei Methoden. Nämlich das Kopieren innerhalb eines Fensters (XPutImage/X11) oder das Verschieben des Bildschirminhalts (XScrollRect/X11). Beide Kommandos erzeugen die gleiche Ausgabe. Das Kopieren erfordert aber wesentlich mehr Daten, da die Pixmap als Teil des Kommandos im

Ausgabestrom erscheint, während beim Verschieben die im Zielsystem schon existierende Quellpixmap nur referenziert wird.

Bsp.: Microsoft Word 5 für Macintosh verwendet beim Hochrollen des Fensterinhalts einen Kopierbefehl für Pixmaps (CopyBits/Quickdraw). Wenn der Effekt dieses Kommandos richtig erkannt wird, kann es durch Verschieben des Fensterinhalts ersetzt werden, das keine Pixmap enthält, sondern nur Referenzen auf den zu verschiebenden Bereich. Der Ausgabereplikator analysiert deshalb jeden Kopierbefehl daraufhin, ob Quellbereich und Zielbereich der Kopieroperation

1. sich auf dem Bildschirm und nicht im Speicher befinden,
2. im gleichen Fenster liegen und
3. gleich groß sind.

Ist dies der Fall, dann wird die aufwendige Übertragung einer großen Pixmap durch ein Verschiebungskommando mit wenigen Byte ersetzt. Kopierbefehle entstehen nur dann, wenn der Quellbereich des betroffenen Fensters nicht verdeckt ist. Ist das Fenster verdeckt, dann erzeugt ein Anwendungsprogramm eine echte Ausgabe, die vom Grafiksystem entsprechend der Sichtbarkeit unterdrückt wird.

Gegen viele Fälle von Gedankenlosigkeit bei der Implementierung kann man nicht mit allgemeinen Regeln vorgehen. So gibt es ASCII-Texteditoren (z.B. BBEdit 3.0 für Macintosh), die bei jedem Tastaturanschlag statt des neuen Zeichens die gesamte Zeile neu zeichnen. Im lokalen interaktiven Betrieb besteht kein merklicher Geschwindigkeitsunterschied zwischen der Ausgabe eines Zeichens und einer Zeichenkette. Beim interaktiven Betrieb über Netzwerke kann die wiederholte Übertragung ähnlicher Zeichenketten aber so viel Bandbreite beanspruchen, daß der Betrieb merklich verlangsamt wird. Für den Ausgabereplikator ist nicht erkennbar, ob ein anderes Ausgabekommando, in diesem Fall eben die Übertragung nur eines Zeichens, den gleichen Effekt hat, wie ein vorliegendes Kommando, das eine Zeichenkette enthält. Es bleibt deshalb anderen Komponenten im Ausgabereplikator überlassen, diese Art von Redundanz aus dem Ausgabestrom zu entfernen. Gerade in solchen Fällen wiederholter ähnlicher Ausgaben ist aber die in Kapitel 4.3.3.1 vorgestellte selektive Kompression sehr effizient

4.4. Der Eingabekonzentrator

Zur gemeinsamen Nutzung von Anwendungsprogrammen gehört neben gemeinsamer Betrachtung auch gemeinsame Bedienung. Alle Teilnehmer sollen ein Programm im Rahmen des Application Sharing Systems so bedienen können, als ob es lokal ausgeführt würde. Das Konzept der Fernbedienung von Programmen, d.h. der räumlichen Trennung von Benutzereingaben und Datenverarbeitung ist schon vielfältig realisiert worden. So zum Beispiel durch den kommandozeilenorientierten Terminalbetrieb oder durch netzwerkfähige Grafiksysteme, wie dem X Window System. Gemeinsame Nutzung bedeutet aber zusätzlich auch das Zusammenfassen von Eingabeströmen mehrerer entfernter Benutzer. Mehrere parallele Eingabeströme werden durch Sequenzialisierung kombiniert. Der kombinierte Eingabestrom wird zur Weiterverarbeitung an den Eingabeinjektor weitergeleitet.

Der Eingabekonzentrator faßt den lokalen Eingabestrom und einen oder mehrere Eingabeströme entfernter Benutzer zusammen.

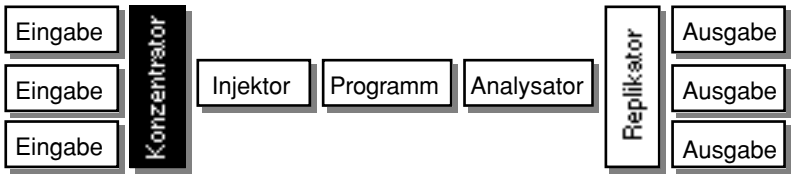


Abbildung Sharing System Komponenten - Konzentrator

Der Eingabekonzentrator kombiniert mehrere Eingabeströme und leitet den resultierenden Eingabestrom an den Eingabeinjektor weiter.

Es gibt ein breites Spektrum von Möglichkeiten zur Kombination mehrerer Eingabeströme. Welche der Möglichkeiten gewählt wird hängt von der Anwendung des Application Sharing Systems ab. Dabei müssen 2 Ebenen unterschieden werden:

1. Vergabe und Anwendung des Eingaberechts (Floor Control) und
2. Sicherstellung der Integrität des kombinierten Eingabestroms.

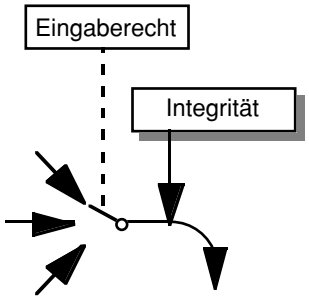


Abbildung Eingabekonzentrator - Ebenen

Auf oberer Ebene kontrolliert die *Eingaberechtsvergabe* in Interaktion mit den Teilnehmern den Zugang einzelner Teilnehmer zu gemeinsam benutzten Anwendungsprogrammen. Auf unterer Ebene sorgt die *Integritätskontrolle* unter Berücksichtigung technischer Randbedingngen der Betriebssystemplattform und der Anwendungsprogramme für die Konsistenz bei der Kombination mehrerer Eingabeströme.

4.4.1. Eingaberechtsvergabe

Das Eingaberecht bestimmt für jeden Teilnehmer die Möglichkeiten zur Eingabe an gemeinsam benutzte Programme. Der Eingabekonzentrator filtert die Eingabeströme anhand der Eingaberechte. Das heißt, er läßt nur Eingaben von einer Untermenge der Teilnehmer (aktive Teilnehmer) zu. Die Teilnehmer, deren Eingaben ignoriert werden, sind sogenannte Beobachter (passive Teilnehmer). Der Status von Teilnehmern kann sich über die Zeit ändern. Die verschiedenen Mechanismen, die zu diesen Statusänderungen führen werden als Modi der Eingaberechtskontrolle (Floorcontrol Modes) bezeichnet.

Die wichtigsten am häufigsten auftretenden Modi sind

- der sogenannte freie Eingabemodus (sog. "Chaos Mode").
  Im freien Eingabemodus gibt es keine explizite Steuerung des Eingaberechts. Alle Teilnehmer können Eingaben machen. Eingaben aller Teilnehmer werden in der Reihenfolge ihres Auftretens ohne Filterung an den Eingabeinjektor geschickt.
- Der Moderationsmodus (sog. "Chaired Mode").
  Ein ausgezeichneter Teilnehmer weist das Eingaberecht zu. Oft ist dies der Teilnehmer auf dessen Rechner das Anwendungsprogramm läuft. Nur ein Teilnehmer kann Eingaben machen.
- Der Weitergabemodus (sog. "Token Passing Mode").
  Das Eingaberecht wird von einem zum nächsten Teilnehmer weitergereicht. Es gibt verschiedene Versionen willkürlicher und automatischer Weitergabe.
- Der sogenannte Reservierungsmodus (sog. "Grab Mode").
  Zu jeder Zeit hat nur ein Teilnehmer das Eingaberecht. Jeder Teilnehmer kann sich das Eingaberecht nehmen, d.h. es einem anderen Teilnehmer entreißen. Dazu ist aber eine bewußte Aktion notwendig. Dies unterscheidet den Reservierungsmodus vom freien Eingabemodus.

Im freien Eingabemodus wird im Gegensatz zu allen anderen Modi keine Filterung von Eingaben durchgeführt. In Application Sharing Systemen, die verschiedene Floorcontrol-Modi anbieten, wird der Floorcontrol-Eingabefilter dann umgangen. Um zu vermeiden, daß dieser Modus einen algorithmischen Sonderfall darstellt, wird er im Eingabefilter oft durch eine automatische Zuweisung des Eingaberechts bei Anliegen einer Eingabe realisiert. Das heißt, sobald Eingaben eines Teilnehmers vorliegen, wird dem Teilnehmer das Eingaberecht zugeteilt. Diese Umschaltung des Eingaberechts kann so schnell erfolgen, daß Eingaben mehrerer Teilnehmer scheinbar gleichzeitig entgegen genommen werden. Während einer Eingabefolge gibt es dann keinen Schutz vor Unterbrechungen durch andere Teilnehmer. Deshalb muß

die Eingabe zwischen den Teilnehmern über einen zusätzlichen Kommunikationskanal synchronisiert werden. Dies geschieht in der Praxis durch Absprache über Audio (Telefon oder computerbasiert) und eventuell Video.

Alle Modi bis auf den freien Eingabemodus erfordern eine Interaktion der Teilnehmer mit der Steuerungskomponente des Application Sharing Systems, bevor Eingaben möglich sind. Je umfangreicher dieser Verwaltungsaufwand ist, desto störender wirkt er, da er nicht der gemeinsamen Arbeit dient, sondern der Steuerung des Application Sharing Systems. Den geringsten Aufwand an Interaktion, außer dem freien Eingabemodus, benötigt der Reservierungsmodus. Er kann so realisiert werden, daß nur ein Teilnehmer, nämlich derjenige, der Eingaben machen möchte, eine minimale Aktion in der Benutzerschnittstelle des Application Sharing Systems, z.B. einen Mausklick, ausführen muß, um das Eingaberecht zu erhalten. Wird die Methode ergänzt durch die Verwaltung einer Untergruppe von Kandidaten (aktive Teilnehmer), dann kann der aktive Zugang zu Programmen für einzelne Teilnehmer beschränkt werden. Das System erfüllt dann gewisse, wenn auch eingeschränkte, Sicherheitsanforderungen.

Auf die Vergabe des Eingaberechts soll hier nicht weiter eingegangen werden. Vertiefendes über diese Thematik ist nachzulesen in [GuWe93] [PaTi90] [Pieres93] [ReSchVöWe94].

4.4.2. Integrität des Eingabestroms

Beim Übergang des Eingaberechts von einem Teilnehmer auf den anderen Teilnehmer wird ein Eingabestrom unterbrochen. Die Unterbrechung kann zu jeder Zeit, auch während einer Eingabe stattfinden. Unter Umständen entstehen dadurch unvollständige Eingaben, die zu einem anderen, als vom Benutzer beabsichtigten Ergebnis führen. Zur gleichen Zeit wird ein anderer Eingabestrom plötzlich eingeschaltet. Für ein Anwendungsprogramm wird eine unvollständige Eingabe des unterbrochenen Stroms durch die Eingaben des nächsten Stroms fortgesetzt. Die Reaktion von Anwendungsprogramm und Systemsoftware kann sehr verschieden sein. Auf diese Art sinnentstellte Eingaben werden oft ignoriert, können aber auch falsch interpretiert werden. Im ungünstigsten Fall entstehen Verklemmungen im Zustand der Systemsoftware oder des Anwendungsprogramms. Der Eingabekonzentrator muß deshalb dafür sorgen, daß die gemeinsam benutzten Anwendungsprogramme über den Eingabeinjektor nur vollständige Eingaben erhalten.

Beispiel für solch eine Verklemmung ist die sogenannte hanging-Button Problematik. Falls vom Druck der Maustaste nur die Mitteilung über das Drücken der Taste (nicht aber das Loslassen) zum Anwendungsprogramm gelangt, dann bleibt das Programm in einem vom Benutzer ungewollten Zustand. Ein Mausklick kann so unbeabsichtigt zur drag-and-drop Operation werden. Das gleiche gilt für Tastatureingaben. Bei Tastatureingaben kommt aber verschärfend hinzu, daß von den meisten Eingabesystemen Tastatureingaben automatisch wiederholt werden, solange eine Taste nicht losgelassen wird.

Um Inkonsistenzen und Verklemmungen zu vermeiden, soll die Umschaltung der Eingabe immer nur auf Grenzen zwischen (evtl. zusammengesetzten) Eingaben erfolgen (siehe Kapitel 3.3.4). Die dabei betrachtete Ordnung von Eingaben hängt von den Anforderungen an ein Application Sharing System ab. Diese werden vor allem durch die geplante Anwendung bestimmt. Die Strategie, die bei der Eingabeumschaltung angewendet wird, beeinflußt einerseits die Möglichkeiten der gemeinsamen und gleichzeitigen Eingabe, andererseits aber auch den Schutz vor Fehlbedienung.

Verschiedene Anforderungen an ein Application Sharing System bezüglich Flexibilität und Sicherheit bei der Umschaltung der Eingabe zwischen Teilnehmern sind vorstellbar:

1. größte Flexibilität bei der Umschaltung von Eingabeströmen durch Umschaltung zwischen elementaren Eingaben.
2. Vermeidung von Störungen eines Teilnehmers durch andere Teilnehmer während zusammengesetzter Eingaben.
3. Vermeidung von Störungen eines Teilnehmers durch andere Teilnehmer während Bearbeitungsvorgängen.

Für die Entscheidung, welche Ordnung von Eingaben (Kapitel 3.3.4) bei der Integritätssicherung als unteilbar betrachtet werden soll, gibt es drei Möglichkeiten, nämlich die Ordnungen eins bis drei. Die vierte Ordnung kommt als Basis für die Umschaltung von Eingabeströmen nicht in Betracht. Trennung bei der vierten Ordnung würde den Ausschluß der Umschaltung von Eingabeströmen auf der unteren

Ebene der Eingabekontrolle bedeuten und damit im Widerspruch stehen zu einer Eingaberechtsvergabe auf oberer Ebene, die Umschaltung zuläßt.

4.4.2.1. Umschaltung zwischen elementaren Eingabeereignissen

Eine Anforderung an ein Application Sharing System kann größte mögliche Flexibilität bei der Umschaltung von Eingabeströmen sein. Größte Flexibilität bedeutet, daß die Umschaltung zwischen zwei Eingabeströmen jeweils auf der Grenze von elementaren Eingabeereignissen erfolgen können soll. In diesem Fall kann eine zusammengesetzte Eingabe eines Benutzers durch einen anderen Benutzer fortgesetzt werden. So ist es zum Beispiel möglich, daß ein Teilnehmer mit der Manipulation eines Objekts beginnt (drag-and-drop), ein anderer Teilnehmer nach dem Umschalten der Eingabe die Aktion durch Bewegen der Maus fortführt und durch Loslassen des Mausknopfs beendet. Derartige Fälle können nützlich sein und mit Absicht herbeigeführt werden. Sie können aber auch aufgrund absichtlicher oder unabsichtlicher Störungen entstehen.

Elementare Eingaben (z.B. Knopf gedrückt/losgelassen) sind eindeutig zu trennen. Es sind Ereignisse ohne Dauer, die durch ihr individuelles Auftreten getrennt sind. Sollten tatsächlich Eingaben von verschiedenen Endgeräten gleichzeitig auftreten, dann werden sie durch die sequentielle Abarbeitung mehrerer Eingabekanäle im Eingabekonzentrator zeitlich geordnet und damit trennbar.

Vermeidung von Verklemmungssituationen

Neben der Forderung nach größter Flexibilität gibt es aber auch eine notwendige Bedingung für einen erfolgreichen Betrieb. Die notwendige Bedingung ist die Vermeidung von Verklemmungssituationen, die durch unvollständige Eingaben entstehen. In Einzelfällen können unvollständige zusammengesetzte Eingaben, d.h. einzelne elementare Eingaben, Verklemmungen in der Systemsoftware oder Anwendungsprogrammen auslösen. Ein Beispiel dafür ist das Drücken einer Modifizierungstaste, wie der Steuerungstaste (Ctrl). Wird der Eingabestrom nach dem Drücken einer Modifizierungstaste umgeschaltet von Teilnehmer A zu Teilnehmer B, dann interpretieren System und Anwendungsprogramm alle folgenden Tastatureingaben von B, wie wenn dieser selbst die Modifizierungstaste gedrückt hätte. B kann diese Verklemmung lösen, indem die Modifizierungstaste gedrückt und wieder losgelassen wird. Das Drücken der Taste wird im allgemeinen ignoriert, aber das Loslassen der Taste setzt den Zustand der Taste in der Systemsoftware, bzw. dem Anwendungsprogramm zurück. Solche Fälle können vermieden werden, indem nur zwischen zusammengesetzten Eingaben zweiter Ordnung umgeschaltet wird. Diese Strategie widerspricht aber der oben genannten Forderung nach größter möglicher Flexibilität. Ob sie zum Einsatz kommt entscheidet das Anwendungsszenarium.

Korrektur des Eingabestroms

Zusätzlich zur reinen Umschaltung zwischen elementaren Eingaben muß deshalb bei jeder Umschaltung der Zustand des kombinierten Eingabestroms so korrigiert werden, daß er dem Zustand der Eingabegeräte des Teilnehmers B entspricht. Der Zustand der Eingabegeräte von A wird bei der Umschaltung ungültig. Da Systemsoftware oder Anwendungsprogramme den Zustand der Eingabegeräte aber intern festhalten, muß dieser interne Zustand durch Einfügen künstlicher Eingabeereignisse an den Zustand der Eingabegeräte von B angepaßt werden. Im Fall des oben genannten Beispiels bedeutet dies das Einfügen eines künstlichen Eingabeereignisses, mit dem das Loslassen der Modifizierungstaste vorspiegelt wird.

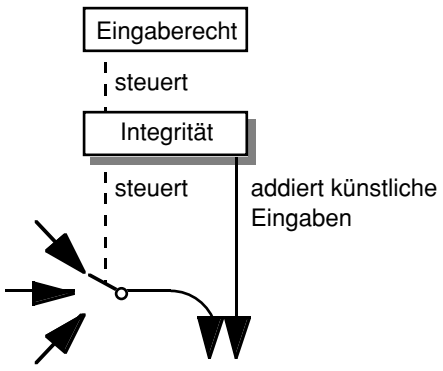


Abbildung Künstliche Eingaben durch Integritätskontrolle

Eingabeströme werden auf Veranlassung der Eingaberechtsvergabe umgeschaltet. Die Integrität des resultierenden Eingabestroms wird durch künstliche Eingabe sichergestellt.

4.4.2.2. Umschaltung zwischen zusammengesetzten Eingaben

Die Möglichkeit zur Störung, bzw. Weiterführung fremder Eingaben kann für den Anwender zu verwirrenden Effekten führen. Eine Anforderung an ein Application Sharing System kann deshalb in der Vermeidung solcher Störungen bestehen. Gerade im freien Eingabemodus der Eingaberechtsvergabe werden Eingabeströme jeweils beim Anliegen eines Ereignisses umgeschaltet. Das heißt, das jede unbeabsichtigte Mausbewegung über einem gemeinsamen Fenster eine gerade ablaufende Eingabe eines anderen Teilnehmers stört. Störungen dieser Art können durch Umschaltung auf den Grenzen von Eingaben zweiter Ordnung vermieden werden.

Die Erkennung von Eingaben zweiter Ordnung ist komplexer, als die elementarer Eingaben. Entsprechend der Definition (Kapitel 3.3.4.2) sind zusammengesetzte Eingaben zweiter Ordnung durch Kombination aller Eingabekanäle erkennbar. Das verbindende Element, über eine Folge elementarer Eingaben hinweg, ist die Aktivierung mindestens eines Eingabegeräts. Die Integritätskontrolle schaltet deshalb erst dann zwischen Eingabeströmen um, wenn alle Eingabegeräte im Grundzustand sind. Dies ist in der Praxis der Fall, wenn keine Taste, bzw. Maustaste gedrückt ist.

Zusammengesetzte Eingaben, die aus der Definition herausfallen, sind so nicht als Eingaben zweiter Ordnung identifizierbar. Besonders wichtig sind aber z.B. Doppelklicks von Maustasten, die oft andere Aktionen auslösen als zwei einzelne Klicks. Nach dem ersten Klick ist der Mausknopf, wie alle Eingabegeräte, wieder im Grundzustand. Zu diesem Zeitpunkt ist ein einzelner Klick nicht von einem Doppelklick unterscheidbar. Erst nach Ablauf des Doppelklickintervalls ist eine Entscheidung über die Art der vorliegenden zusammengesetzten Eingabe möglich. Ein Eingabekonzentrator, der Doppelklicks oder ähnlich geartete zusammengesetzte Eingaben erkennen soll, kann dies durch die Einführung von Pseudogeräten tun. Die Eigenschaften dieser Pseudoeingabegeräte werden so festgelegt, daß sie der Definition zusammengesetzter Eingaben entgegenkommen. Im Falle des Doppelklicks wird das "Doppelklick-Pseudogerät" durch einen Klick (zwei elementare Eingaben) aktiviert und automatisch nach Ablauf des Doppelklickintervalls deaktiviert. Da die Umschaltung von Eingabeströmen nur geschehen kann, wenn alle Eingabegeräte deaktiviert sind, werden so Doppelklicks oder ähnliche zusammengesetzte Eingaben nicht unterbrochen. In der Praxis blockiert der genannte Mechanismus eine mögliche Umschaltung für eine kurze Zeit nach jeder elementaren Eingabe, die ein Pseudoeingabegerät aktiviert.

4.4.2.3. Umschaltung zwischen Bearbeitungsvorgängen

Anwendungsabhängige Eingaben dritter Ordnung, wie das Füllen einer Fläche oder die Eingabe eines Wortes sind Bearbeitungsvorgänge auf den Objekten von Anwendungsprogrammen. Soll ein Application Sharing System Eingabeströme nur zwischen Bearbeitungsvorgängen umschalten, dann müssen die Grenzen von Bearbeitungsvorgängen eindeutig identifiziert werden. Die Grenzen der Eingaben sind, wie in den Fällen von elementaren Eingaben und zusammengesetzten Eingaben, durch die Definition der Eingaben bestimmt. Für die Eingaben dritter Ordnung ist die Definition allerdings abhängig vom Anwendungsprogramm und vom Anwendungsszenarium. Der Satz potentieller Eingaben wird für jedes Programm basierend auf dessen Eingabemöglichkeiten mit Rücksicht auf das Einsatzszenarium bestimmt. Da der Benutzer Bearbeitungsvorgänge abbrechen kann, müssen dabei auch unvollständige Bearbeitungsvorgänge berücksichtigt werden. Auf diese Art könnten für jedes Programm Muster von Eingabefolgen erstellt werden, die Bearbeitungsvorgänge darstellen. Durch Vergleich des aktiven Eingabestroms mit den möglichen Eingabefolgen kann der Eingabekonzentrator zur Laufzeit nach jeder elementaren Eingabe entscheiden, ob eine Eingabegrenze vorliegt.

Trennung von Bearbeitungsvorgängen nach GOMS

Nach der GOMS-Methode (Goals, Operators, Methods & Selection Rules [Kieras88]) werden mögliche Eingabesequenzen in Programmform definiert. Die Eingabesequenzen werden dadurch ausführbar. Ein solches GOMS-Modell enthält Sequenzen von Eingaben, denen jeweils Bearbeitungszeiten zugeordnet sind. Mehrere Alternativen sind möglich. Sie sind jeweils mit Wahrscheinlichkeiten versehen. Die GOMS-Methode entstammt der taskorientierten Interaktionsmodellierung [CaMoNe80] [CaMoNe83] [KiWoMe97]. Sie wird dort verwendet, um Benutzerschnittstellen bezüglich Bedienungsfreundlichkeit und -geschwindigkeit zu optimieren. Im Rahmen von Application Sharing können die gleichen GOMS-Modelle als applikationsspezifische Muster typischer und wahrscheinlicher Eingabefolgen benutzt werden. GOMS-Modellierung ist eines der wenigen weit verbreiteten Konzepte der HCI-Forschung [JoKi94]. Da GOMS von Entwicklern zur Analyse von Benutzerschnittstellen eingesetzt wird, besteht die berechtigte Hoffnung, daß zusätzlich zu den selbst erzeugten Modellen geringer Tiefe, auch applikationsspezifische Modelle mit viel Detail von den Softwareherstellern verfügbar sein werden.

Trennung von Bearbeitungsvorgängen mit Applikationsunterstützung

Viele Programme gehen heute über die Verarbeitung von Eingabemeldungen hinaus. Sie kennen typische Bearbeitungsvorgänge, versuchen den Benutzern kontextabhängig zu helfen, reagieren auf Verzögerungen, die auf Bedienungsprobleme hindeuten, usw. Diese Informationen könnten verwendet werden, um einem Application Sharing System Nachricht über Beginn und Ende von Bearbeitungsvorgängen zu geben.

Spracheingabe bietet in diesem Fall ganz besondere Möglichkeiten. Die Eingaben bestehen nicht aus elementaren Eingaben, wie Mausklicks, sondern spezifizieren komplette Bearbeitungsvorgänge. Sie liegen auf der Ebene der Eingaben zweiter Ordnung.

Typische heute verwendete Anwendungsprogramme verwalten aber nicht genügend Informationen, um dem Application Sharing System Nachricht über Beginn und Ende von Bearbeitungsvorgängen zu geben. Die Information muß vom Benutzer selbst kommen. Der Benutzer muß dem Application Sharing System Beginn und Ende seines Bearbeitungsvorgangs mitteilen. Hier geht die Umschaltung zwischen Eingabeströmen auf der Ebene der Integritätskontrolle in die Eingaberechtsvergabe über.

4.5. Der Eingabeinjektor

Benutzereingaben werden von lokalen Eingabegeräten, wie Tastatur, Maus oder Digitalisiertablett aufgenommen. Die Systemsoftware nimmt die digitalisierten Eingaben entgegen und wandelt sie in Eingabemeldungen für Anwendungsprogramme um. Anhand von systemspezifischen Regeln (Eingabefokus siehe Kapitel 2.3.5) entscheidet die Systemsoftware über die Zuordnung zu Programmen, d.h. welche Eingabe für welches Programm bestimmt ist. Die Eingabemeldungen werden dann an die Programme verschickt, indem sie in die entsprechende Meldungswarteschlange eingefügt werden.

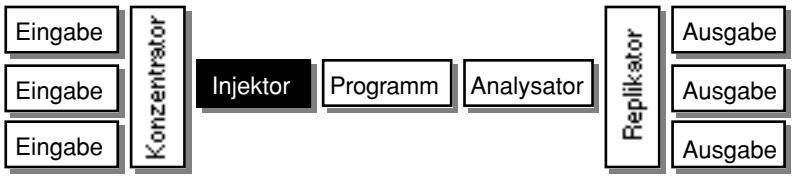


Abbildung Sharing System Komponenten - Injektor

> Der Eingabeinjektor speist den vom Eingabekonzentrator erhaltenen Eingabestrom der entfernten Benutzer in das lokale Eingabesystem ein.

Im Application Sharing System müssen Eingaben, die nicht von den lokalen Eingabegeräten stammen, einem gemeinsam benutzten Anwendungsprogramm zugeführt werden. Die Eingaben werden von entfernten Eingabegeräten aufgenommen und kommen als Folge von Dateneinheiten vom Eingabekonzentrator zum Eingabeinjektor. Sie sind für das System auf dem das Anwendungsprogramm abläuft nicht durch Benutzereingaben ausgelöst, sondern künstlich.

Der Eingabeinjektor hat die Aufgabe aus den Dateneinheiten vom Eingabekonzentrator künstlichen Eingaben zu erzeugen und diese so and die entsprechenden Anwendungsprogramme oder das System zu verschicken, daß sie sich nicht von lokalen Benutzereingaben unterscheiden.

Da die Treiber der Eingabegeräte nur auf echte Benutzereingaben reagieren, können diese nicht benutzt werden, um künstliche Eingaben zu erzeugen. Es bieten sich aber andere Möglichkeiten an Schnittstellen zwischen Anwendungsprogrammen und der Systemsoftware. Prinzipiell kann man zwei Methoden unterscheiden:

1. Synthese und Einfügen von Eingabemeldungen in die Meldungswarteschlange und
2. Synthese von Eingaben durch Pseudotreiber.

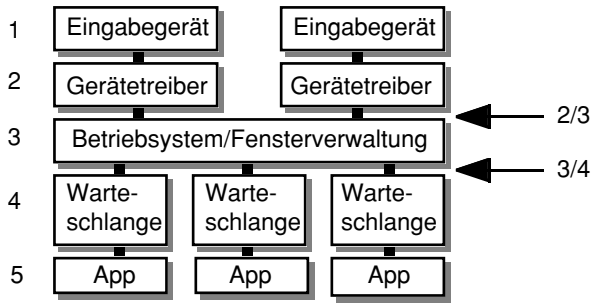


Abbildung Arbeitspunkte des Eingabeinjektors

Künstliche Eingaben können an verschiedenen Ebenen in das Eingabesystem eingefügt werden.

Das Einfügen von Eingabemeldungen in die Meldungswarteschlange ist der Eingabe durch Pseudotreiber, wenn möglich, vorzuziehen. Treibereingabe bedeutet, daß die Systemsoftware Kenntnis von den künstlichen Eingaben erhält. Dadurch beeinflussen die künstliche Eingaben den Systemzustand und unter Umständen alle Anwendungsprogramme. Tatsächlich sind Eingaben vom Eingabekonzentrator aber nur für einzelne gemeinsam benutzte Anwendungsprogramme bestimmt. Das Einfügen von Meldungen in die Meldungswarteschlange von Programmen vermeidet die Beeinflussung an derer Programme durch die Systemsoftware. Beide Möglichkeiten werden in den folgenden Abschnitten beschrieben und einzelne Implementierungen vorgestellt.

4.5.1. Künstliche Eingabemeldungen

Meldungen von der Systemsoftware oder anderen Instanzen an Anwendungsprogramme spielen auf vielen Betriebssystemplattformen eine zentrale Rolle. Die Meldungen benachrichtigen Programme über Ereignisse und die Programme reagieren durch Datenverarbeitung und Ausgaben. Dieses Programierparadigma wird als "ereignisgesteuerte Programmierung" bezeichnet. Die Art der Ereignisse kann je nach Betriebssystem sehr vielfältig sein. Man kann einige verschiedene Typen unterscheiden, wobei auf einzelnen Plattformen aber auch einzelne Typen fehlen können:

1. Meldungen der Systemsoftware über Programm und Systemzustand,
2. Benutzereingaben,
3. Meldungen zur Interprozeßkommunikation,
4. Meldungen von programmeigenen oder systemeigenen Zusatzkomponenten, wie Transportsystem, Spracherkennung und vor allem der Objekte der graphischen Benutzeroberfläche.

Benutzereingaben sind nur eine Untermenge der möglichen Meldungen an Anwendungsprogramme. Alle Meldungen gelangen über Meldungswarteschlangen zu Anwendungsprogrammen. Auf den meisten Systemen gibt es eine solche Meldungswarteschlange für jedes Anwendungsprogramm. Die Meldungen werden also über die Auswahl der Warteschlange den Anwendungsprogrammen zugeordnet. Unter der Voraussetzung, daß die Systemsoftware schreibenden Zugriff auf die Meldungswarteschlange zulassen, kann der Eingabeinjektor die künstlichen Eingaben in die entsprechende Meldungswarteschlange einfügen.

4.5.1.1. Beispiel MS Windows Messages

Unter MS Windows können Programme Meldungen (Messages) zur Interprozeßkommunikation nutzen. Die Messages sind dabei an einzelne Fenster adressiert. Der Eingabeinjektor verschickt die Eingabeereignisse als Messages an die Fenster der Anwendungsprogramme. Sie werden automatisch in die Meldungswarteschlange des Anwendungsprogramms eingefügt, zu dem das Fenster gehört.

4.5.1.2. Beispiel X Window Events

Im X Window System werden alle Benutzereingaben als Eingabemeldungen (X Events) ausgeliefert. Es ist nicht möglich, den Zustand von Eingabegeräten direkt abzufragen. Die Eingabemeldungen werden über die gleiche Transportsystemverbindung übermittelt, wie die Ausgaben eines

Anwendungsprogramms. Die Meldungswarteschlange besteht aus den Puffern der Transportsystemverbindung. Um Meldungen in diese Verbindung einzufügen muß der Eingabeinjektor auf den terminalseitigen Transportsystemendpunkt schreibend zugreifen können. Geschieht die Analyse des Ausgabestroms an der Netzwerkschnittstelle ebenfalls am terminalseitigen Transportsystemendpunkt (siehe Kapitel 4.2.7.1), dann liegt es deshalb nahe, bei der Implementierung für X11 Ausgabeanalysator und Eingabeinjektor zu integrieren.

4.5.2. Künstliche Eingabe durch Treiber

Auf manchen Systemen werden nicht alle Benutzereingaben als Eingabemeldungen an Anwendungsprogramme verschickt. Programme verfolgen dann einen Teil der Benutzereingaben direkt durch Abfrage des Gerätezustands, statt mittels Eingabemeldungen informiert zu werden. In diesem Fall genügt es nicht, Eingabemeldungen in die Meldungswarteschlange einzufügen. Gleichzeitig muß auch die Systemsoftware so präpariert werden, daß sie als Antwort auf eine direkte Abfrage den Zustand der künstlichen Eingaben meldet. Die Eingabe muß deshalb durch Treiber geschehen.

Bei Eingabesystemen mit mehreren Eingabekanälen, besonders hierarchische Grafiksystemen, die Fenster als Eingabekanäle nutzen, muß der Eingabeinjektor die Kanalzuordnung übernehmen. Dazu werden Informationen über den Eingabefokus des Window Managers benötigt. Diese Informationen sind nur in Schicht 3 des Eingabesystems (Abbildung *Eingabesystem*) verfügbar. Findet die Eingabe dagegen durch Treiber statt, dann kann der Eingabeinjektor die Funktion der Schicht 3 des Eingabesystems für die Zuordnung der künstlichen Eingaben mitbenutzen.

Auf vielen Systemen ist es möglich zu den vorhandenen Treibern noch andere Eingabetreiber zu installieren, um den Betrieb zusätzlicher oder anderer Eingabegeräte zu ermöglichen. Ein Beispiel dafür ist ein Digitalisiertablett, das als Zeigegerät der Maus ähnlich ist, aber absolute Koordinaten generiert und deshalb nicht vom Maustreiber unterstützt wird. Für den Eingabestrom vom Eingabekonzentrator wird eine Komponente des Eingabeinjektors als spezieller Eingabetreiber installiert. Dieser Treiber setzt den Eingabestrom in Eingabemeldungen an die Systemsoftware um. Für die Systemsoftware unterscheiden sich diese Meldungen nicht von denen der lokalen Eingabegeräte.

4.5.2.1. Beispiel Apple Macintosh Maustreiber

Auf dem Apple Macintosh erhalten Anwendungsprogramme Benutzereingaben auf beide der oben diskutierten Arten. Es werden sowohl Meldungen über Eingaben verschickt, als auch der Zustand der Geräte direkt abgefragt. Die Systemsoftware meldet den Anfang einer Eingabesequenz (Mausknopf gedrückt) als Ereignis. Anschließend fragen viele Anwendungsprogramme den Zustand des Knopfes und die Mausposition direkt ab, bis der Benutzer den Mausknopf losläßt. Meldungen über Bewegungen des Mauszeigers werden dagegen oft ignoriert. Diese Verfahrensweise von Anwendungsprogrammen ist nicht systembedingt, wird aber von den Funktionen der Macintosh Toolbox gefördert (siehe z.B. DragWindow(), Button() [IM-I88]). Sie muß beim Entwurf des Eingabeinjektors berücksichtigt werden.

Die Systemsoftware des Apple Macintosh erlaubt die Installation zusätzlicher Zeigegeräte neben der Standardmaus. Das System verwaltet nur einen Mauszeiger. Dieser Zeiger kann aber von mehreren Geräten, bzw. Gerätetreibern gleichzeitig betrieben werden. Alle Gerätetreiber verwenden die Schnittstellen des sog. Cursor Device Managers um ihre Eingabemeldungen an die Systemsoftware weiterzugeben. Obwohl der Eingabeinjektor wie ein Eingabetreiber arbeitet, muß er auf dem Macintosh nicht als Treiber implementiert sein. Es genügt ein Codefragment, das die Dateneinheiten des Eingabekonzentrators in die entsprechenden Aufrufe des Cursor Device Managers umwandelt.

5. Ausgabekonvertierung

In diesem Kapitel werden Techniken zur Übersetzung von Grafikausgaben zwischen verschiedenen Grafiksystemen beschrieben. Dafür werden für in jedem Teilbereich Lösungen für Klassen von Grafiksystemen vorgestellt und anschließend Spezialfälle gesondert behandelt.

5.1. Einleitung

Konvertierung ist immer dann notwendig, wenn die Schnittstelle des Grafikendgeräts (bzw. des Zielgrafiksystems) von der des Quellgrafiksystems abweicht. Dies ist besonders in einer heterogenen Umgebung der Fall.

Ein anderer, gleichermaßen wichtiger Grund für die Konvertierung eines Grafikausgabestroms ist der Schritt vom Prozeduraufrufen zum netzwerkfähigen Grafikprotokoll. Werden im Rahmen eines Sharing Systems Anwendungsprogramme verwendet, die auf einem lokalen Grafiksysteme aufsetzen, dann muß der Grafikausgabestrom in ein netzwerkfähiges Grafikprotokoll umgesetzt werden. Aufrufe von Prozeduren des Grafiksystems, die zugehörigen Parameter und Kontextinformationen müssen in PDUs eines Grafikprotokolls umgewandelt werden.

Konvertierung eines Grafikstroms bedeutet die Umwandlung von Grafikkommandos in ein für das Endgerät geeignetes Format. In den folgenden Kapiteln werden zwei unterschiedliche Ansätze beschrieben, die in wesentlichen Eigenschaften voneinander abweichen:

1. Rendering im Quellsystem und
2. Rendering im Zielsystem.

Obwohl sich beide Ansätze stark unterscheiden, liefern sie das gleiche Ergebnis, nämlich die Konvertierung eines Grafikausgabestroms.

5.2. Rendering im Quellsystem

Tabelle *Grafikprimitive* vergleicht die Kommandosyntax von Grafiksystemen. Sie zeigt, daß viele Grafikprimitive in mehreren Grafiksystemen existieren während einige Primitive nur in wenigen Grafiksystemen verfügbar sind. Selbst bei Primitiven, die in allen Grafiksystemen existieren, gibt es oft semantische oder funktionelle Unterschiede.

Nur das Primitiv Pixmap ist in allen Grafiksystemen mit sehr ähnlichen Eigenschaften vorhanden (siehe Kapitel 2.4.1.2). Es liegt deshalb nahe, nur Pixmaps von einem Grafiksystem in das andere zu übersetzen. Dazu muß der gesamte zu konvertierende Grafikausgabestrom als Folge von Pixmaps vorliegen. Es sind also zwei hintereinander geschaltete Konvertierungsschritte notwendig:

1. die Umsetzung aller Ausgabekommandos des Quellsystems in Pixmaps des Quellsystems und
2. die Übersetzung der Pixmaps vom Quellsystem in das Zielsystem.

5.2.1. Pixmap-Wandlung im Quellsystem

Die Umsetzung aller Ausgabekommandos in Pixmaps ist der Darstellung auf dem Bildschirm (dem sog. Rendering) sehr ähnlich. Beim Rendering werden alle Grafikprimitive auf eine Pixmap, nämlich den Bildschirmspeicher abgebildet. Nimmt man nach jeder Ausgabeoperation einen Abzug des Bildschirmspeichers auf, dann entsteht eine Folge von Pixmaps, die den Ausgabestrom komplett wiedergibt. Ein Bildschirmabzug ist aber nicht geeignet als Umsetzung der Grafikprimitive in Pixmaps. Der Bildschirminhalt umfaßt nicht nur einen Ausgabestrom, sondern die Ausgaben aller Programme.

5.2.1.1. Aufnahme durch Offscreen Ausgabe

Es ist aber möglich, nur den Teil der Bildschirmausgaben zu erfassen, der zu dem zu konvertierenden Ausgabestrom gehört. Der Mechanismus ist dem Rendering in den Bildschirmspeicher ähnlich. Statt in den Bildschirmspeicher, wird der Ausgabestrom in eine Pixmap ausgegeben. Die Pixmap befindet sich im Speicher und ist im Gegensatz zu Bildschirmspeicher nicht sichtbar (sog. Offscreen-Pixmap). Diese Ausgabe in eine Pixmap kann im allgemeinen mit den Mitteln des Quellgrafiksystems durchgeführt werden. Die meisten Grafiksysteme bieten Möglichkeiten, um Grafikprimitive in Offscreen-Pixmaps auszugeben. Dieser Mechanismus wird oft dazu verwendet, um - für den Benutzer unsichtbar - komplexe Grafiken aufzubauen, die anschließend sehr schnell auf dem Bildschirm präsentiert werden (siehe Kapitel 4.3.4.3). Alle Grafikkommandos eines Ausgabestroms, der in eine Abfolge von Pixmaps umgesetzt werden soll, werden nacheinander in eine Offscreen-Pixmap ausgegeben. Der Zustand der Pixmap wird nach jeder Ausgabeoperation aufgezeichnet. Zwischen Ausgaben wird die Pixmap nicht zurückgesetzt. Es werden dabei verschiedene Optimierungen mit dem Ziel verwendet, die Datenmenge soweit, wie möglich zu verringern. Das Verfahren zur Pixmap-Wandlung im Quellsystem verwendet die Mittel (Funktionen) des Quellgrafiksystems zur Grafikausgabe in eine Offscreen-Pixmap.

5.2.1.2. Beschränkung auf aktive Gebiete

Die aufgezeichneten Pixmaps müssen nicht immer den gesamten Bereich des Ausgabestroms umfassen. Es genügt nach jeder Ausgabe eines Grafikprimitivs nur den jeweils betroffenen Bereich zu kopieren. Ausdehnung und Position des Bereichs, der von einer Grafikausgabe betroffen ist, kann an den im Ausgabekommando enthaltenen Koordinatenparametern abgelesen werden. In den meisten Grafiksystemen sind als Pixmaps nur rechteckig angeordnete Pixelmengen erlaubt. Der Bereich, der als Pixmap in den konvertierten Ausgabestrom übernommen wird, ist deshalb das Rechteck, das die Pixel umfaßt, die von der Ausgabe eines Grafikprimitivs betroffen sind.

5.2.1.3. Getaktete Ausgabe

Bei schnell aufeinanderfolgenden und sich eventuell überschreibenden Ausgaben ist es nicht notwendig, nach jeder Ausgabe eine Pixmap in den konvertierten Ausgabestrom aufzunehmen (abzutasten). Es genügt die betroffenen Bereiche in regelmäßigen Abständen abzutasten. Der zeitliche Abstand der Abtastung soll so gewählt werden, daß Ausgaben nicht wesentlich verzögert in den konvertierten Ausgabestrom übernommen werden. Es können damit unter Umständen mehrere Ausgabeoperationen durch eine Pixmap ersetzt werden. Die jeweils erzeugte Pixmap soll alle Änderungen der Offscreen-Pixmap erfassen. Dazu muß während der Ausgabe in die Offscreen-Pixmap über die Änderungen Buch geführt werden (siehe *Generationenkarte* in Kapitel 4.3.4.2).

Das Resultat dieses Vorgangs ist eine Folge von Pixmaps mit relativen Koordinaten, die den Aufhängepunkt der Pixmap im Ausgabestrom angeben. Der Transfermodus, bzw. die Rasteroperation der Pixmapausgaben wird auf 'Copy', d.h. deckendes Kopieren der Quelle gesetzt.

5.2.2. Schnelle Pixmap-Übersetzung

Die Definition von Pixmaps ist in unterschiedlichen Grafiksystemen sehr ähnlich. Wie in Kapitel 2.4.1.2 gesehen, bestehen sie aus einer Beschreibung der Pixmap, den Pixmapdaten und einer Farbtabelle für den Fall, daß die Pixmap nicht Pixel in Echtfarbdarstellung enthält.

5.2.2.1. Aufwand für Pixmap Übersetzung

Die Pixmapdaten sind eine Folge von Pixelwerten, die - im Fall von Echtfarben - jeweils aus Werten für drei Farbkanäle bestehen. Die Anordnung der Farbkanäle in jedem Pixelwert wird bestimmt durch die Beschreibung der Pixmap. Fast alle Grafiksysteme verwenden das RGB-Farbmodell. In einzelnen Grafiksystemen sind auch andere Farbmodelle verfügbar. Videoorientierte Grafiksysteme (MPEG, Quicktime) bieten nur das YUV-Modell oder bevorzugen dieses Farbmodell. Bei allen Grafiksystemen, die das RGB-Farbmodell für Pixmaps bieten, ist die Übersetzung von Pixmaps trivial. Die Farbwerte der drei Kanäle einzelner Pixel können direkt übernommen werden. Sie müssen nur entsprechend der gültigen Bitanordnung der Farbkomponenten im Pixel des Zielgrafiksystems angeordnet werden. D.h. es müssen unter Umständen die Bits des G-kanals mit denen des B-Kanals vertauscht werden. Bei der Konvertierung einer Echtfarbpixmap in eine 16-Bit Pseudocolorpixmap ("High Color" bei Win32) mit der Bitanordnung 5:5:5 für die Kanäle R:G:B ist zwar keine Umrechnung, wohl aber eine Reduzierung

der Farbwerte notwendig. Bei der Reduzierung der Farbkanäle von 8 auf 5 Bit je Pixel müssen die Bits entsprechend maskiert und im 16-Bit-Wort verschoben werden.

Eine Umwandlung von oder in indexbasierte Pixmaps ist mit noch größerem Aufwand verbunden. Im einfacheren Fall, der Konvertierung einer indexbasierten Pixmap in Echtfarbdarstellung muß für jeden Pixel anhand des Indexwertes in der Farbtabelle der Echtfarbwert ermittelt werden. Anschließend wird, wie bei der Konvertierung zwischen Echtfarbpixmaps verfahren.

Bei der Konvertierung von Echtfarbdarstellung oder Pseudocolordarstellung in indizierte Farben muß fast immer die Farbvielfalt reduziert werden. Dazu können bekannte Algorithmen zur Farbreduzierung verwendet werden, z.B. Dithering [FoDaFeHu91]. Es muß aber beachtet werden, daß einige Grafiksysteme spezielle Anforderungen an die Verteilung von Farbwerten in der Farbtabelle stellen. Farbtabellen, die bei der Farbreduktion entstehen, können deshalb nicht ohne weiteres verwendet werden. Beispiele für solche speziellen Anforderungen an die Belegung der Farbtabelle sind Quickdraw und Win32. Bei Quickdraw muß der erste Farbwertwert (Index 0) immer Weiß, der letzte (Index 255) immer Schwarz sein; zusätzlich soll auch Index 1 aus Kompatibilitätsgründen Schwarz sein. Bei Win32 sind die ersten 16 Farbtabelleneinträge immer für Standardfarben reserviert, die vom Window Manager verwendet werden. Im X Window System gibt der X-Server die Indexwerte für Schwarz und Weiß vor. Im X-Protokoll vorgesehen sind außerdem X-Server mit fest eingestellter Farbtabelle. In diesem Fall sind alle Farbwerte unveränderbar. Algorithmen zur Farbreduzierung müssen deshalb so angepaßt werden, daß bestimmte Farbtabelleneinträge fest vorgegeben werden können.

Der aufwendigste Teil der Konvertierung von Pixmaps zwischen Grafiksystemen ist die eventuell notwendige Farbreduktion auf eine teilweise vorgegebene Farbtabelle. Aber auch die Expansion von indexbasierten Pixmaps in Echtfarbpixmaps oder Anpassung der Bits je Farbkanal zwischen 16- und 24-Bit Pixmaps ist ein erheblicher Aufwand, da jedes Pixel bearbeitet werden muß. Dabei werden einzelne Bits und Teile von Bytes im Speicher verarbeitet, was für moderne Prozessoren sehr aufwendig ist. Die Algorithmen müssen deshalb effizient implementiert werden.

5.2.2.2. Übersetzung unter Verwendung des Grafiksystems

Die Umwandlung von Pixmaps bezüglich Farbtiefe ist genau der gleiche Prozeß, wie beim Rendering von Pixmaps für die Bildschirmausgabe. Aus diesem Grund liegt es nahe, die entsprechenden Pixmaptransfer-Funktionen der jeweils verfügbaren Grafikengine für die Konvertierung zu verwenden. Die Pixmaptransfer-Funktionen von Grafiksystemen sind normalerweise optimiert für Geschwindigkeit, da schnelle Pixmapausgaben ein wichtiger Bestandteil jedes Grafiksystems sind. Um Funktionen des Quellgrafiksystems für die Konvertierung von Pixmaps zu verwenden, müssen zwei Voraussetzungen erfüllt sein:

1. Flexible Offscreenpixmaps
   Das verwendete Grafiksystem (das Quellsystem bei der Konvertierung) muß Offscreenpixmaps bereitstellen, die bezüglich Farbpalette und Farbtiefe konfigurierbar sind. Es muß die Möglichkeit zur Grafikausgabe durch Pixmaptransfer-Funktionen in diese Offscreen-Pixmaps bieten.
2. Die Datenbereiche der Pixmaps des Quellgrafiksystems müssen hinreichend kompatibel mit denen des Zielsystems sein. Das Quellgrafiksystem wird in diesem Fall verwendet, um Pixmaps für das Zielgrafiksystem mit den vorgegebenen Eigenschaften (Farbpalette und Farbtiefe, Pixelkodierung) zu erzeugen. Im Anschluß wird die konvertierte Pixmap an das Format des Zielgrafiksystems (siehe Tabelle *Pixmapformate*) angepaßt. Bei dieser Anpassung wird im Idealfall der Datenbereich der Pixmap, der die Pixelwerte enthält unverändert übernommen und nur ein für das Zielgrafiksystem geeigneter Header angefügt.

Die Ausgabekommandos eines konvertierten Grafikausgabestroms werden über ein Netzwerk an andere Rechner verschickt. Um dabei unnötige Kopiervorgänge eines Stroms von Pixmaps zu vermeiden, ist es vorteilhaft, die konvertierten Pixmaps direkt an der Stelle im Speicher zu erzeugen, von der sie an die Netzwerkschnittstelle (z.B. MacOS: MacTCP/OpenTransport; MS Windows: Winsock; SunOS/Solaris: Berkeley Sockets; etc.) übergeben werden. Kopiervorgänge innerhalb des Transportsystems können damit natürlich nicht vermieden werden. Durch entsprechende Einstellung der Parameter der Offscreen-Pixmap, vor allem der Länge von Scanlines (Quickdraw: rowBytes), kann der Pixmapkonverter die Ausgabe der Grafikengine in die Offscreen-Pixmap steuern. Er kann dafür sorgen, daß die Datenbereiche der erzeugten Grafik PDUs so im Speicher liegen, daß sie einschließlich der eingefügten Headerinformation einen Puffer des Transportsystems sequentiell füllen.

Abbildung *Pixmapkonvertierung* zeigt den Ablauf einer Pixmapkonvertierung. Sind die Datenbereiche der Pixmaps in Quellgrafiksystem und Zielgrafiksystem nicht kompatibel, dann muß der Pixmapkonverter auch Schritt 4, Farbtabellenanpassung, Farbtiefenanpassung und Pixelkodierung übernehmen.

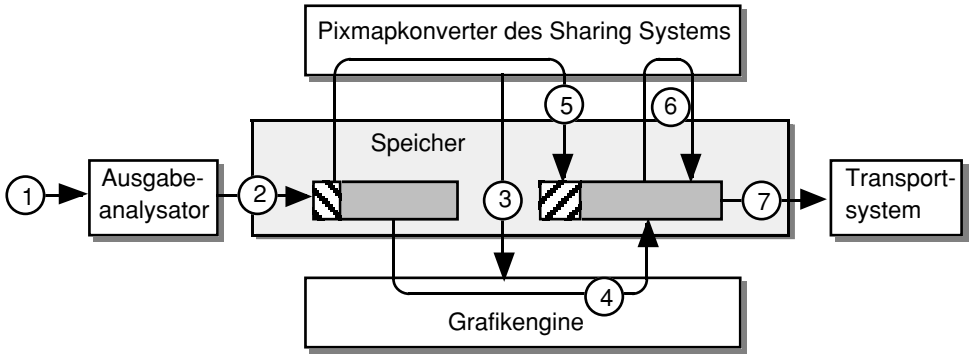


Abbildung Pixmapkonvertierung

Ablauf einer effizienten Methode zur Pixmapkonvertierung. Die Grafikengine des Quellgrafiksystems konvertiert den Datenbereich, während der Pixmapkonverter den Pixmapheader anpaßt:

1. Pixmapausgabe durch ein Programm oder in Pixmaps gewandelte Ausgaben anderer Grafikprimitive.
2. Pixmaps werden im Speicher im Format des Quellgrafiksystems abgelegt.
3. Der Pixmapkonverter veranlaßt das Quellgrafiksystem den Datenbereich (die Pixelwerte) der Pixmap für das Zielgrafiksystem zu erzeugen.
4. Das Grafiksystem führt Farbtabellenanpassung und Farbtiefenanpassung durch.
5. Der Pixmapkonverter ergänzt die Pixelwerte durch einen Pixmapheader im Format des Zielgrafiksystems.
6. Der Pixmapkonverter nimmt eventuell notwendige Anpassungen an der Pixelanordnung vor.
7. Die Pixmap wird im Format des Zielgrafiksystems an das Transportsystem übergeben. Die Daten sollten dabei nicht kopiert werden, sondern als Referenz übergeben werden.

Das Verfahren zur Pixmap-Übersetzung verwendet die Mittel (Funktionen) eines Grafiksystems zur Konvertierung der Pixeldaten.

5.2.3. Merkmale des Renderings im Quellsystem

Die Methode des Renderings im Quellsystem bietet Vorteile gegenüber dem Rendering im Zielsystem. Diese Vorteile werden aber durch Eigenschaften des erzeugten Grafikstroms erkauft, die sich bei gewissen Anwendungen negativ auswirken können. Der wesentliche Unterschied zur Methode des Renderings im Zielsystem besteht darin, daß Grafikprimitive in dem Grafiksystem, aus dem sie stammen, in Pixmaps umgesetzt werden. Damit kann für die Interpretation des Ausgabestroms die Grafikengine des Quellsystems verwendet werden. Probleme des Renderings im Zielsystem (siehe nächstes Kapitel) werden dadurch vermieden:

- Es ist keine Konvertierung von anderen Grafikprimitiven, als Pixmaps notwendig.
- Ressourcenprobleme, wie die Verfügbarkeit von Zeichensätzen, entfallen.
- Clippingmasken werden im Quellsystem beim Rendering angewendet und müssen nicht zum Zielsystem übertragen werden.

Der aus der Konvertierung hervorgehende Grafikstrom hat folgende Eigenschaften:

- Erhöhte Datenrate

Der konvertierte Grafikstrom besteht nur aus einer Folge von Pixmaps. Pixmaps sind die Grafikprimitive mit dem schlechtesten durchschnittlichen Pixel/Byte-Verhältnis. Der Grafikstrom kann deshalb eine deutlich größere Datenrate aufweisen, als der originale Grafikausgabestrom.

- Semantikverlust
  Der konvertierte Grafikstrom enthält weniger Informationen, als der Originale. Es fehlen die Informationen über die ursprünglich verwendeten Grafikprimitive und ihre Parameter. Ein bedeutender Informationsverlust besteht darin, daß alle Ausgabeoperationen auf rechteckige Pixmaps abgebildet werden. Damit sind im Grafikstrom nur noch die umschließenden Rechtecke, aber keine Informationen über die tatsächlich bei einzelnen Grafikausgaben modifizierten Bildbereiche enthalten.

5.2.3.1. Anwendungsfall: Heterogene Umgebung

Eine heterogene Umgebung bedeutet, daß mehrere Rechner verschiedener Ausstattung an einem Application Sharing System angeschlossen sind und einen Ausgabestrom empfangen. Die hier relevanten Unterscheidungsmerkmale sind Grafiksystemsoftware und Bildschirmtiefe.

Wird der konvertierte Grafikausgabestrom gleichzeitig zu mehreren verschieden ausgerüsteten Rechnern verschickt, dann muß der Ausgabestrom für jeden Rechner individuell konvertiert werden. Wenn das Rendering im Quellgrafiksystem, wie oben beschrieben, unter Zuhilfenahme der Grafikengine durchgeführt wird, dann trägt der Quellrechner die gesamte Last der Konvertierung. Er muß gleichzeitig mehrere konvertierte Grafikausgabeströme erzeugen. Die dabei notwendige Verarbeitungsleistung kann hier nicht quantitativ angegeben werden. Sie kann aber zur Verarbeitungsleistung für lokalen Grafikausgaben in Beziehung gesetzt werden. Da bei der Konvertierung eines Grafikstroms ähnliche Vorgänge ablaufen, wie bei der Ausgabe eines Grafikstroms auf dem Bildschirm, ist der Aufwand für Konvertierung und lokale Ausgabe vergleichbar. Jeder zusätzliche konvertierte Grafikausgabestrom erzeugt also eine Systemlast, die der bei der lokalen Grafikausgabe entspricht. Diese Last kann allerdings reduziert werden indem das in Kapitel 4.3.4.2 vorgestellte CE/SC Verfahren eingesetzt wird. Gerade bei Grafikströmen, die ausschließlich aus Pixmaps bestehen läßt sich das CE/SC Verfahren besonders einfach anwenden. Insofern ist das Rendering im Quellsystem besonders gut für CE/SC geeignet.

5.2.3.2. Anwendungsfall: Geringer Netzwerkdurchsatz

Eine erhöhte Datenrate wirkt sich nur dann aus, wenn der verfügbare Durchsatz der Netzwerkverbindung geringer ist, als die Datenrate des Grafikstroms. Grafikausgabeströme von Anwendungsprogrammen erreichen oft Spitzenraten von mehreren MBit/s. Nach einer Umwandlung in Pixmaps kann die Datenrate noch deutlich höher liegen. Selbst mit Kompression, wie in Kapitel 4.3.3.1 beschrieben, übersteigt die Datenrate von Pixmapsequenzen die Grenzen schmalbandiger Netzwerke. Als schmalbandige Netzwerke gelten in diesem Fall Netzwerke mit ISDN- (64/128 kBit/s) oder Modemgeschwindigkeiten (28,8-56 kBit/s) (siehe Messung *Reduktion durch Pixmapkompression*). Für schmalbandige Netzwerke ist die Ausgabekonvertierung durch Rendering im Quellsystem nicht geeignet.

5.2.3.3. Anwendungsfall: Weiterverarbeitung

In manchen Fällen wird ein konvertierter Grafikausgabestrom nicht nur zu einem anderen Rechner übertragen und dort ausgegeben. Es sind Anwendungen möglich in denen der Grafikstrom weiter verarbeitet wird. Solche Weiterverarbeitung kann daraus bestehen, daß die Grafikausgaben wieder in ein anderes Format umgewandelt werden. Ein Beispiel ist die Verteilung von Grafikausgabeströmen als Dienst im Netzwerk oder als Mehrwertdienst des Netzwerksanbieters. Ein anderes ist die permanente Aufzeichnung der Grafikausgaben. Auch eine Aufzeichnung ist im allgemeinen auch mit einer Umwandlung in ein anderes Grafikformat verbunden ist. Es ist auch eine Anwendung vorstellbar, die Übertragung und Aufzeichnung verbindet. Die aufzeichnende Komponente tritt dann gegenüber dem Sharing System, wie ein Grafikendgerät auf, was spezielle Vorkehrungen zur Aufzeichnung überflüssig macht.

Grundsätzlich kann eine weiter verarbeitende Einheit nur die Informationen verwenden, die nach der ersten Konvertierung im Quellsystem noch im Grafikstrom enthalten sind. Je weniger Informationen verfügbar sind, desto beschränkter sind die Möglichkeiten zur Weiterverarbeitung. Informationen über andere Grafikprimitive können nicht mehr ohne sehr großen Aufwand rekonstruiert werden. Damit geht auch eine eventuell vorhandene Auflösungsunabhängigkeit des Ausgabestroms verloren, die für Druckausgaben gebraucht wird. Ein Strom von Pixmapausgabekommandos kann nur noch in andere pixmaporientierte Formate (MPEG, GIF, etc.) oder in Pixmaps anderer Stromformate umgesetzt werden.

Die Ausgabekonvertierung durch Rendering im Quellsystem ist deshalb nur dann geeignet, wenn keine Weiterverarbeitung stattfindet, die auf höherwertige Informationen angewiesen ist, als die, die in Pixmaps enthalten sind.

5.3. Rendering im Zielsystem

Durch Rendering im Quellgrafiksystem werden alle Grafikausgaben auf Pixmaps reduziert. Sie werden als Pixmaps übertragen und so vom Grafikendgerät ausgegeben. Soll die Übertragung als Pixmap vermieden werden, dann müssen statt dessen andere Grafikprimitive über das Netzwerk übertragen werden. Der Grafikausgabestrom liegt im Ausgabeanalysator als Folge von Grafikausgabekommandos und Kontextmanipulationen vor. Es liegt deshalb nahe, genau diese Kommandos zu den angeschlossenen Grafikendgeräten zu übertragen. Die Grafikendgeräte interpretieren dann den Strom von Grafikkommandos und führen die entsprechenden Ausgaben durch.

In diesem Kapitel soll davon ausgegangen werden, daß ein typischer Arbeitsplatzrechner als Endgerät dient. Die gemeinsam benutzten Anwendungsprogramme sollen auf dem Bildschirm, wie lokal ablaufende Programme erscheinen. Eine Softwarekomponente übernimmt die Funktion des Grafikendgeräts für das Sharing System. Diese Softwarekomponente (Displayserver genannt) verwendet für die Bildschirmausgaben wiederum das lokal verfügbare Grafiksystem, wie in Kapitel 2.1 beschrieben. Der Displayserver empfängt Grafikkommandos über das Netzwerk, interpretiert diese und ruft die Programmierschnittstelle des lokalen Grafiksystems.

Dazwischen findet eine Umsetzung von den Protokolldateneinheiten (PDUs) des Grafikprotokolls in Prozeduraufrufe statt. Dieser Prozeß ist die genaue Umkehrung dessen, was gleichzeitig auf dem absendenden Arbeitsplatzrechner stattfindet. Auf dem absendenden Arbeitsplatzrechner werden Prozeduraufrufe und deren Argumente in PDUs umgewandelt. Im empfangenden Arbeitsplatzrechner werden die PDUs wieder in Prozeduraufrufe mit den zugehörigen Parametern gewandelt. Der gesamte Ablauf ist genau dann trivial, wenn es bei beiden Konvertierungen jeweils direkte Abbildungen zwischen den Darstellungen der Ausgabekommandos gibt. Eine derartige direkte Abbildung wird im folgenden Kapitel beschrieben.

5.3.1. Grafikkommandos als Protokolldateneinheiten

Für den Fall, daß das gesamte Application Sharing System, einschließlich des Displayservers, implementiert werden soll, gibt es keine einschränkenden Randbedingungen für das verwendete Grafikprotokoll. Das Protokoll der Schnittstelle B (Kapitel 4.3.3.1) kann dann im Rahmen des Application Sharing Systems frei definiert werden. Es kann so gewählt werden, daß eine einfache Abbildung der Ausgabekommandos in PDUs möglich ist.

Ein Beispiel für eine solche einfache Abbildung ist folgende Konstruktionsvorschrift für PDUs. Das so konstruierte Protokoll entspricht der gleichen Grafiksyntax, wie die Ausgabekommandos:

1. Aus jedem Ausgabekommando (siehe Tabelle *Grafikprimitive*) wird beim Ausgabesignal (Kapitel 2.5.3) eine PDU konstruiert.
2. Die Art des Ausgabekommandos wird als Zahl oder durch den Prozedurnamen (z.B. als Text) kodiert.
3. Die Parameter der Ausgabekommandos werden entsprechend der Reihenfolge der Prozedurparameter in PDUs des Grafikprotokolls eingetragen.
4. Referenzierte Objekte (Ressourcen, siehe Kapitel 2.2.3) werden serialisiert und als Teil der PDU angehängt.

Diese vier Vorschriften beschreiben eine Standardmethode, um jegliche Art von Prozeduraufrufen in PDUs für die Netzwerkübertragung umzuwandeln. Die letzten beiden Vorschriften sind zugeschnitten auf Grafikausgabeströme.

5. Bei jeder Änderung des bei der Ausgabe verwendeten Grafikkontextes wird entweder der gesamte Grafikkontext oder nur die Änderung in eine eigene PDU abgebildet und vor der PDU des Ausgabekommandos in den Ausgabestrom eingefügt.

6. Es wird nur ein Grafikkontext verwaltet, so daß auch die Umschaltung zwischen verschiedenen Grafikkontexten im Quellsystem (z.B. Quickdraw: SetPort) einer Änderung des aktuellen Kontextes gleichkommt.

Ein Grafikstrom wird durch direkte Umsetzung eines Prozeduraufrufs (oben) in Protokolldateneinheiten eines Grafikprotokolls konstruiert. Durch die genannte Konstruktionsvorschrift entsteht eine Folge von Kontextmanipulationskommandos und Ausgabekommandos.

Der Displayserver führt den umgekehrten Prozeß durch. Anhand der PDU-Typen, Parameter und eingebetteten Objekte werden die Prozeduraufrufe rekonstruiert. Damit erhält der Displayserver wieder eine Folge von Kontextmanipulationen und Prozeduraufrufen (eine Folge von Beschreibungen von Prozeduraufrufen). Anhand dieser Beschreibungen ruft der Displayserver Prozeduren der Programmierschnittstelle des lokal vorhandenen Grafiksystems. Gibt es hier eine direkte Abbildungen zwischen den PDUs und Prozeduren des lokalen Grafiksystems, dann spricht man von homogenem Application Sharing. Quellgrafiksystem, Zielgrafiksystem und das vermittelnde Grafikprotokoll entsprechen dann der gleichen Grafiksyntax.

5.3.2. Abbildung von Grafikausgaben

Entspricht das Zielgrafiksystem nicht der gleichen Grafiksyntax, wie das Quellsystem, dann liegt heterogenes Sharing vor. In diesem Fall muß es nicht zwangsläufig, wie beim homogenen Sharing, eine Möglichkeit geben, aus den PDUs direkt Prozeduraufrufe für das Zielsystem abzuleiten. Unter Umständen ist es notwendig, Ausgabekommandos des Quellsystems, die im Zielsystem nicht existieren (vgl. Tabelle *Grafikprimitive*), durch die verfügbaren Ausgabekommandos des Zielsystems nachzubilden.

Damit stellt sich die Frage, ob für alle Ausgabekommandos eines beliebigen Grafiksystems (Quellsystem) eine Abbildung in Ausgabekommandos eines beliebigen anderen Systems (Zielsystem) existiert. Angenommen, das Quellsystem kann durch Ausgabekommandos beliebige Pixelanordnungen erzeugen. Nach Kapitel 2.1 werden hier nur solche Systeme als Grafikendgeräte betrachtet, die einer Grafiksyntax entsprechen, die beliebige Pixelanordnungen zuläßt. Wenn das Zielgrafiksystem als Grafikengine eines Grafikendgeräts (Displayserver) dient, dann muß es gleichfalls die Beschreibung beliebiger Pixelanordnungen erlauben. Jede Pixelanordnung, die das Quellsystem vorgibt, kann deshalb auch durch das Zielsystem beschrieben werden. Ein Ausgabekommando im Quellsystem manipuliert eine gegebene Pixelanordnung so, daß sie einen neuen Zustand einnimmt. Geht man im Zielsystem von der gleichen Anfangsanordnung aus, dann kann man auch hier durch eine Folge von Kommandos des Zielsystems den neuen Zustand erreichen. Es gibt also für jedes Ausgabekommando eines Grafiksystems eine Abbildung auf ein oder mehrere Ausgabekommandos eines anderen Grafiksystems.

Da alle relevanten Grafiksysteme die Ausgabe von Pixmaps unterstützen, kann natürlich prinzipiell jede Grafikausgabe in einem anderen Grafiksystem durch Pixmaps repräsentiert werden. Für diesen Fall gibt es zwei Optionen. Die erste, genannt Rendering im Quellsystem, wurde im Kapitel 5.2 beschrieben. Die zweite Möglichkeit besteht darin, daß der Displayserver alle Grafikausgabekommandos des Quellsystems interpretiert, selbst ausführt und nur Pixmaps an das Zielgrafiksystem übergibt. Dies bedeutet allerdings, daß der Displayserver des Sharing Systems die gesamte Funktionalität der Grafikengine des Quellgrafiksystems nachbilden muß.

Wie in Kapitel 2.4 beschrieben, gibt es in den meisten Grafiksystemen Ausgabekommandos, die sich mit ähnlicher Funktionalität auf die gleichen Grafikprimitive beziehen. Es liegt deshalb nahe, nicht alle Grafikkommandos im Displayserver auf Pixmaps, sondern auf vergleichbare Kommandos des Zielsystems abzubilden. Die Herausforderung besteht darin, für alle Ausgabekommandos des Quellsystems eine möglichst einfache und effiziente Kodierungsvariante im Zielsystem zu finden. Damit kann das Sharing System die beteiligten Grafiksysteme nutzen und muß nicht die Funktionalität einer Grafikengine nachbilden. Dies reduziert einerseits die Komplexität des Sharing Systems und steigert gleichzeitig die Effizienz des Gesamtsystems, da die Reimplementierung einer Grafikengine durch das Sharing System im allgemeinen weniger effizient ausfällt, als Routinen, die von den Herstellern der Grafiksysteme über Jahre gewartet und optimiert wurden. Stellt sich bei Messungen heraus, daß einzelne Grafikprimitive oder Modifikationen von Primitiven in den Grafiksystemen nicht effizient implementiert sind, dann können Teile der Funktionalität eines der beteiligten Grafiksysteme im Sharing System natürlich trotzdem nachgebildet werden.

Auf welchem der beteiligten Rechner die Übersetzung des Grafikausgabestroms stattfindet, ist dabei unerheblich. Die Aufgabenverteilung zwischen den Komponenten des Sharing Systems ist nur für das jeweilige Einsatzszenarium relevant (siehe Kapitel 7).

Videoorientierte Grafiksysteme sollen hier nicht betrachtet werden, da sie im wesentlichen nur das Primitiv Pixmap unterstützen. Zur Konvertierung in diese Grafiksysteme muß ein Sharing System Pixmaps erzeugen und deshalb auf jeden Fall eine Grafikengine verwenden. Da z.B. ein MPEG-Endgerät nur MPEG als Stromformat empfangen kann, muß die Konvertierung nach MPEG im Quellsystem geschehen (siehe Kapitel 5.2).

5.3.3. Übersetzung von Grafikprimitiven

In diesem Kapitel wird anhand von Beispielen die Konvertierung von Grafikausgaben in korrespondierende Ausgaben anderer Grafiksysteme dargestellt. Es kann hier keine vollständige Aufstellung für die Übersetzung aller möglichen Ausgaben zwischen allen Grafiksystemen gegeben werden. In den folgenden Abschnitten sollen deshalb wichtige Sonderfälle diskutiert, Probleme genannt und geeignete Lösungen angeboten werden. Die Diskussion wird der Übersicht wegen nach Grafikprimitiven gegliedert. Für jedes Grafikprimitiv werden jeweils allgemeine Methoden diskutiert und spezielle Fälle behandelt. Jeder Fall besteht dabei aus einer Beschreibung, dem Problem und einem Lösungsvorschlag. Bei Lösungen wird jeweils angegeben, ob die Lösung durch Implementierung überprüft wurde (markiert durch "[realisiert]"), oder durch die Analyse von Spezifikationen oder Betrachtung ähnlicher Lösungen abgeleitet wurde (markiert durch "[analysiert]").

Viele Probleme entstehen daraus, daß spezielle Eigenschaften einzelner Grafiksysteme nicht mit den gleichen Primitiven in anderen Grafiksystemen dargestellt werden können. In diesen Fällen muß auf andere Primitive ausgewichen werden. Da sowohl durch Pfade und Bereiche, als auch durch Pixmaps mit Clippingmaske beliebige Formen erzeugt werden können, bieten sich Bereiche und Pixmaps natürlich als Ersatz an.

Eine weitere Problemklasse ausgelöst durch Ressourcendefizite wird in Abschnitt 5.3.3.6 dargestellt.

5.3.3.1. Linien

Linien sind in den meisten Grafiksystemen in gleicher Weise definiert. Sie können zwischen Grafiksystemen direkt übersetzt werden. Nur dann, wenn nicht beide Grafiksysteme die gleichen Stiftparameter unterstützen, ist besondere Beachtung notwendig. Konflikte treten dann auf, wenn ein Grafiksystem die Breite von Linien durch den Parameter Linienbreite, ein anderes durch die Stiftform definiert.

Sonderfall 1: Abbildung einer schrägen Linie mit rechteckigem Stift in ein Grafiksystem ohne Stiftform.

Problem: Die Linienbreite variiert mit dem Winkel während die Linienabschlüsse für alle Winkel gleich bleiben.

Lösung: Umwandlung der Linie in ein gefülltes Polygon. Füllmuster (bzw. Füllfarbe) des Polygons entsprechen dem Muster (Farbe) der Linie [realisiert].

Sonderfall 2: Abbildung einer Linie mit Stift irregulärer Form in ein Grafiksystem ohne beliebige Stiftformen.

Problem: Die Linienbreite variiert mit dem Winkel während die Linienabschlüsse für alle Winkel gleich bleiben. Irreguläre (nicht-rechteckige) Linienabschlüsse sind im Zielgrafiksystem nicht darstellbar.

Lösung: Umwandlung der Linie in einen Bereich. Eine Möglichkeit, den Bereich zu erstellen ist das Zeichnen eines Linienbündels indem für jedes Pixel des Stiftes eine Linie gezogen wird [realisiert].

Sonderfall 3: Abbildung einer schrägen Linie, die durch ihre Breite definiert ist, in ein Grafiksystem, das mit Stiftformen arbeitet.

Problem: An den Linienabschlüssen erscheinen im Zielsystem immer Teile der Stiftform. Der tatsächliche Fehler hängt von der Form des Linienabschlusses im Quellsystem ab.

Lösung: Bei eckiger Form des Linienabschlusses (z.B. X11 cap_style: CapButt, oder CapProjecting) Umwandlung der Linie in ein gefülltes Polygon [realisiert], sonst (z.B. X11 CapRound) in einen Bereich [realisiert].

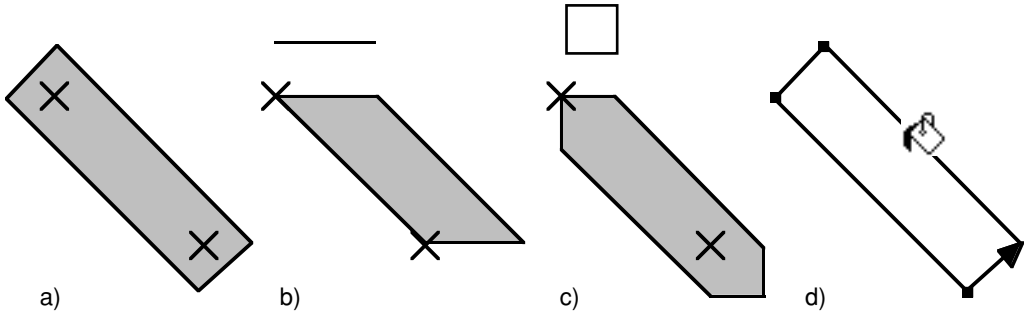


Abbildung Nachbildung Linienabschluß

Linie a) (Grafiksystem arbeitet mit Linienbreiteparameter, hier cap_style = CapProjecting) soll in einem Grafiksystem mit rechteckigen Stiftformen nachgebildet werden. Es gibt keine Stiftform, die a) genau wieder gibt. b) und c) zeigen Beispiele für eine Linie mit den gleichen Koordinaten, aber verschiedene Stiftformen. Die Linie wird deshalb als gefülltes Polygon d) ausgegeben.

5.3.3.2. Einfache Formen

Bei der Umrandung von Rechteck, Polygon, Kreis und vergleichbaren Primitive treten ähnliche Probleme auf, wie bei einzelnen Linien. Die Lösungen entsprechen deshalb den oben genannten.

Sonderfall 1: Abbildung der Umrahmung einfacher Formen zwischen Systemen mit unterschiedlicher Liniendefinition.

Problem: Die tatsächliche Form kann sich durch nichttriviale Stiftformen (ungleich 1x1 Pixel) stark ändern (Vergleiche Anhang *Strichstärke und Stiftformen*).

Lösung: Durch Umwandlung in einen Bereich, wie oben. Der Bereich kann sowohl im Quellsystem, als auch im Zielsystem erstellt werden. Im Quellsystem, indem die Ausgabe in eine 1 Bit tiefe Pixmap (Bitmap) durchgeführt wird, um eine Bitmaske und damit einen bitmapbasierten Bereich zu erstellen [realisiert]. Im Zielsystem, indem das Primitiv für jedes Pixel des Stiftes wiederholt wird. Die Grafikausgaben im Zielsystem erfordern in diesem Fall nur einmalige Kontextmanipulation und danach direkt aufeinanderfolgende Ausgabekommandos, da sich außer den Koordinaten keine Parameter ändern [analysiert]. Hier ist unter Umständen zwischen der Bandbreite der Übertragung einer Bereichsdefinition und der mehrerer Grafikausgaben abzuwägen (siehe Kapitel 7).

Einige häufig auftretende Fälle können und sollten effizienter implementiert werden, indem sie unabhängig vom allgemeinen Ansatz behandelt werden. So werden z.B. in Win32 Rechtecke sehr häufig mit einer quadratischen Stiftform umrahmt (Messung *Stiftformen* und Anhang *Strichstärke und Stiftformen*). Für den Fall eines Rechtecks mit rechteckiger Stiftform kann auf effiziente Weise ein pfadbasierter oder rechteckbasierter Bereich erstellt werden, was gegenüber den oben verwendeten bitmapbasierten Bereichen von Vorteil ist [realisiert].

Gefüllte Formen ähneln sich in allen Grafiksystemen (siehe auch Kapitel "Grafikkommandos: Vergleich von Grafikprimitiven: Linien und einfache Formen"). Gefüllt wird sowohl mit einer homogenen Farbe, als auch mit einem Muster. Für Grafiksysteme, die nur Füllmuster, aber keine Füllfarbe kennen (z.B. Windows NT Treiber), wird ein Muster mit homogener Farbe erzeugt [analysiert]. Die Übersetzung von Füllmustern geschieht, wie bei Pixmaps in Kapitel 5.3.3.3 beschrieben. Die Verknüpfung eines Füllmusters mit den Pixeln des Zielbereichs geschieht analog zur Ausgabe einer Pixmap (Kapitel "Rendering im Zielsystem: Übersetzung von Grafikprimitiven: Pixmapausgaben"). Füllen einer Form mit einem Muster ist nichts anderes, als Replikation einer Pixmap über die Zeichenebene begrenzt durch die jeweilige Form.

Ist ein Grafikprimitiv im Zielsystem nicht verfügbar, dann muß auf Bereiche, Pixmaps mit Clipping, oder Kombinationen von anderen Grafikprimitiven zurückgegriffen werden. Werden Bereiche im Zielsystem angeboten, dann kann die aktuelle Grafikausgabe in einen Bereich umgewandelt und als Bereich ausgegeben werden. Die Umwandlung in einen Bereich kann allerdings nicht im Zielsystem geschehen, da das Grafikprimitiv dort nicht verfügbar ist und deshalb nicht zur Definition eines Bereich herangezogen werden kann. Bereiche werden durch eine spezielle Betriebsart des jeweiligen Grafiksystems erstellt (siehe dazu Kapitel 2.4.1.3).

Kann die Grafikausgabe einer einfachen Form nicht durch einen Bereich ersetzt werden, dann muß auf Übersetzung im Quellsystem zurückgegriffen werden. Durch die Anwendung einer Clippingmaske wird dabei erreicht, daß trotz rechteckiger Pixmaps nur der Bildbereich geändert wird, der das gewünschte Grafikprimitiv enthalten soll . Eine Clippingmaske wird, wie im oben diskutierten Sonderfall 1 erstellt. Der Transfermodus für die Pixmapausgabe im Zielsystem entspricht dem Transfermodus des Füllmusters im Quellsystem (zu Transfermodi siehe auch Kapitel 5.3.3.3.

Sonderfall 2: Das Rechteck mit abgerundeten Ecken (RoundRect) gibt es nur in Win32 und Quickdraw (bzw. Quickdraw Treiber).

Problem: Keine Möglichkeit zur direkten Übersetzung.

Lösung: Zusammensetzung eines RoundRect aus fünf Rechtecken und vier Viertelkreisen (bzw. Viertelellipsen) [realisiert]. Eine Umrandung kann aus vier Strichen und vier Viertelkreisen aufgebaut werden [realisiert]. Eine Definition als Pfad/Bereich ist nicht unbedingt vorteilhaft, weil dann die Pfaddefinition übertragen werden muß (siehe Kapitel 5.3.3.5). Die Pfaddefinition eines RoundRects benötigt nicht weniger Daten, als die Ausgabe von Teilstücken. Nur in Grafiksystemen, wo Füllmuster nicht unabhängig vom Startpunkt nahtlos aneinander anschließen, muß die Ausgabe von Teilstücken vermieden und durch einen Pfad oder Bereich ersetzt werden [analysiert].

Sonderfall 3: Übersetzung der Ausgabe eines Kreisabschnitts (Chord) von Win32 in ein anderes Grafiksystem.

Problem: Keine Möglichkeit zur direkten Übersetzung, da der Kreisabschnitt so nicht von anderen Grafiksystemen geboten wird.

Lösung: Chord ist immer der Teil einer Ellipse, die durch eine Halbebene beschnitten wird. Das Grafikprimitiv Chord kann deshalb durch eine Ellipse nachgebildet werden, der durch Clipping teilweise ausgeblendet ist.

Sonderfall 4: Polygon fill rules.

Problem: Eine vom Quellgrafiksystem geforderte Polygon fill rule steht im Zielgrafiksystem nicht zur Verfügung.

Lösung: Wenn im Quellgrafiksystem und Zielgrafiksystem mit Bereichen gerechnet werden kann, d.h. Bereiche addiert und subtrahiert werden können, dann können fehlende Polygon fill rules simuliert werden. Im Quellgrafiksystem wird ein Polygon mit einer im Zielgrafiksystem existierenden Polygon fill rule erzeugt (Näherungsform). Anschließend wird die Differenz zwischen dem auszugebenden Polygon und der Näherungsform berechnet (Differenzform). Im Zielsystem wird aus dem Polygon ein Bereich erzeugt, welcher der Näherungsform entspricht (mit der vorher im Quellgrafiksystem verwendeten Polygon fill rule). Die berechnete Differenzform wird ins Zielsystem übertragen und von der Näherungsform subtrahiert [analysiert]. Damit ist ein Bereich konstruiert, der dem auszugebenden Polygon entspricht.

5.3.3.3. Pixmapausgaben

Bei der Übersetzung von Pixmapausgaben zwischen Grafiksystemen gibt es zwei wichtige Teilbereiche. Der eine ist die Konvertierung der Pixmap in eine für das Zielsystem geeignete Datenstruktur. Dies wurde in Kapitel 5.2.2 schon diskutiert [realisiert]. Der zweite Teil ist die korrekte Steuerung der Ausgabe unter Beachtung des Transfermodus (siehe Kapitel 2.4.2.1).

Die bei weitem am häufigsten verwendeten Transfermodi sind die 16 logischen Verknüpfungsmöglichkeiten von zwei Pixmaps Messung *Transfermodi*. Der Transfermodus bezeichnet dabei die Art der Verknüpfung der auszugebenden Pixmap (Quellpixmap) und der Pixel des Zielbereichs (Zielpixmap). Da die Zielpixmap nur im Endgerät verfügbar ist, kann auch die Verknüpfung nur dort durchgeführt werden. Es ist deshalb zwingend notwendig, daß das Zielgrafiksystem die gleichen Transfermodi bietet, wie die, die vom Quellgrafiksystem verwendet werden.

Sonderfall 1: Transfermodus des Quellsystems wird vom Zielsystem nicht unterstützt.

Problem: Keine Möglichkeit zur direkten Übersetzung. Die Verknüpfung kann nicht im Quellsystem durchgeführt werden, weil die Zielpixmap fehlt.

Lösung 1: Emulation durch einen anderen Transfermodus: der ähnliche Resultate liefert.

Messung *Transfermodi* zeigt, daß nur wenige Transfermodi oft, viele dagegen sehr selten verwendet werden. Eine falsche Darstellung kann toleriert werden, wenn der Fehler nur sehr selten oder nur mit bestimmten Anwendungsprogrammen auftritt [realisiert].

Lösung 2: In manchen Grafiksystemen ist die Pixmap des Ausgabebereichs abfragbar. In diesem Fall kann die Zielpixmap dem Übersetzer zur Verfügung gestellt werden. Der Übersetzer kann dann die Verknüpfung vollständig durchführen und das Ergebnis im deckenden Transfermodus (Quickdraw: srcCopy) ausgeben. Geschieht die Ausgabe über Netzwerke, dann stellt die doppelte zu übertragende Datenmenge einen Nachteil dar. Darüber hinaus muß die gesamte Grafikausgabe angehalten werden, bis die Zielpixmap zum Übersetzer übertragen und verarbeitet ist. Eine asynchrone Grafikausgabe, die nur von der Geschwindigkeit des Endgeräts beschränkt wird, ist nicht möglich.

Die Alternative zur Abfrage der Zielpixmap besteht darin, die gesamte Zielpixmap eines Grafikausgabestroms im Übersetzer aktuell verfügbar zu halten. Dazu müssen alle Grafikausgaben zusätzlich zur Übersetzung auch in eine Offscreen-Pixmap im Übersetzer ausgegeben werden. Dieser Ansatz wurde im Rahmen des Renderings im Quellsystem beschrieben (Kapitel 5.2.2).

Manche Grafiksysteme bieten Transfermodi, die drei Pixmaps verknüpfen. Die zusätzliche Pixmap ist eine Maske, welche die Gültigkeit der Quellpixmap bestimmt (siehe Win32: MaskBlt). Die Quellpixmap wird mit der Maske verknüpft und das Ergebnis anschließend mit der Zielpixmap. Bei Übersetzung von einem System mit dreikomponentigen Transfermodi in eines mit zweikomponentigen kann die erste Verknüpfung im Quellsystem durchgeführt werden. Für die zweite Verknüpfung gilt die obige Diskussion. Diese Verfahren funktioniert auch dann, wenn die dritte Pixmap nicht die Funktion einer Maske hat oder wenn sogar vier Pixmaps verknüpft werden (Win32: ROP4). Es sind immer alle Pixmaps außer der Zielpixmap im Übersetzer verfügbar und können damit, wie oben diskutiert, vorverarbeitet werden. Der umgekehrte Fall, einer Übersetzung von zweikomponentigen Transfermodi zu Transfermodi mit mehr Komponenten ist trivial, da zweikomponentige Transfermodi eine Untermenge der mehrkomponentigen sind.

5.3.3.4. Kurven

Kurven werden nicht in allen Grafiksystemen angeboten. Probleme und Lösungen bei der Übersetzung von Kurven entsprechen denen der oben genannten Grafikprimitive. Die Problematik der Kontextparameter Stiftform vs. Linienbreite und Linieneabschluß wurde schon im Rahmen der Übersetzung von Linien diskutiert. Möglichkeiten zur Ersetzung der Ausgabe, falls kein vergleichbares Grafikprimitiv im Zielsystem verfügbar ist, wurden im Kapitel über einfache Formen dargestellt.

5.3.3.5. Pfade und Bereiche

Wie in Kapitel 2.4.1.3 dargestellt, kennen fast alle Grafiksysteme Pfade und Bereiche. Bereiche sind nicht nur als Grafikprimitiv bei Ausgaben wichtig, sondern werden bei fast allen Grafikausgaben beim Clipping eingesetzt. Es ist deshalb besonders wichtig, Bereiche des Quellgrafiksystems auch im Zielgrafiksystem als Bereiche zur Verfügung zu stellen zu können, statt sie in Pixmaps umzuwandeln.

Eigenschaften

Im Gegensatz zu Linien, einfachen Formen und Pixmaps ist die Definition von Pfaden und Bereichen aber in vielen Grafiksystemen nicht öffentlich. Grafiksysteme bieten Funktionen mit denen Pfade und Bereiche angelegt und bearbeitet werden können. Die Daten, die den jeweiligen Bereich repräsentieren, werden vom Grafiksystem verwaltet und sind außerhalb der Grafikengine nicht zugreifbar (abfragbar). In einigen Grafiksystemen können Bereiche nicht als Datenstruktur übergeben (installiert) werden, sondern können nur durch die Grafikengine aus einer Folge von Grafikprimitiven konstruiert werden. Anwendungsprogramme halten normalerweise nur Verweise (z.B. Handles) auf die Bereiche, nicht die Datenstrukturen selbst. Oft sind die Datenstrukturen nicht offiziell dokumentiert. Tabelle *Definition und Dokumentation von Pfaden und Bereichen* zeigt eine Zusammenstellung der hier relevanten Eigenschaften von Bereichen in verschiedenen Grafiksystemen. Für die Tabelle wurde auch inoffizielle (nicht vom Hersteller des Grafiksystems) verfügbare Dokumentation berücksichtigt.

In einigen Grafiksystemen liegen die zu einem Pfad oder Bereich gehörenden Daten nicht vor, wenn die Ausgabe zur Übersetzung ansteht. Das heißt, daß eine Grafikausgabe einen Bereich referenziert, dessen Datenstruktur aber nicht abfragbar oder außerhalb der Grafikengine interpretierbar ist. In diesem Fall gibt es zwei Möglichkeiten: Wenn der Bereich gezeichnet werden soll und im Zielgrafiksystem nicht als Bereich benötigt wird, dann kann der Übersetzer die Ausgabe des Bereichs in eine Pixmapausgabe umwandeln. Dies geschieht genauso wie oben im Kapitel über einfache Formen dargestellt.

Grafiksystem	X11[1]	Win32	Quick-draw	Win32 (NT) Treiber	Quick-draw Treiber	GIF	Post-script
Abfragbar	-	x	x	x	x	-	-
Dokumentiert	x	x	x	x	x	x	-
Installierbar	x	x	x	x	x	x	-
Konstruierbar	x[2]	x	x	x	-	-	x

1 X11 kennt keine Pfade und Bereiche. Für das Clipping werden Bitmaps verwendet. Die hier aufgeführten Eigenschaften beziehen sich auf die sogenannte clip-mask des X11 Grafikkontextes.

2 Durch eine Liste von Rechtecken.

Tabelle Definition und Dokumentation von Pfaden und Bereichen

Eigenschaften von Bereichen in verschiedenen Grafiksystemen:

Abfragbar: Die Daten, die Pfade oder Bereiche in der jeweiligen Grafikengine repräsentieren sind vollständig außerhalb der Grafikengine verfügbar. Ein Beispiel ist die Quickdraw Region, deren Handle auf den Speicherbereich zeigt, der die Datenstruktur enthält.

Dokumentiert: Die Datenstrukturen sind bekannt durch Dokumentation des Herstellers, andere Dokumentation oder Reverse-Engineering.

Installierbar: Pfade oder Bereiche können als Datenstruktur an die Grafikengine geschickt werden.

Konstruierbar: Pfade oder Bereiche können durch eine Folge anderer Grafikprimitive über Funktionen des Grafiksystems erstellt werden.

Konstruktion im Zielsystem

Die zweite Möglichkeit ist aufwendiger, bietet aber eine echte Übersetzung auf der Ebene von Pfaden und Bereichen: Pfade und Bereiche werden von Anwendungsprogrammen über die Schnittstellen von Grafiksystemen konstruiert. Die dabei verwendeten Funktionen gehören nicht zu den Ausgabefunktionen, sondern zum Ressourcemanagement (siehe Kapitel 4.2.5.2). Die Kommandos erscheinen nicht im Grafikausgabestrom. Es sind daher zusätzliche Informationen erforderlich, die von einem erweiterten Ausgabeanalysator geliefert werden müssen.

Kommando	Analysator erfaßt	
	nur Grafik	Grafik und Konstruktion von Bereichen
OpenRgn		Signal an den Übersetzer, die folgende Sequenz zur Konstruktion eines Bereichs im Zielsystem zu verwenden.
Line		Übersetzung als Teil der Bereichsdefinition.
LineTo		
LineTo		
FrameRect		
CloseRgn		Ende der Konstruktion des Bereichs im Zielsystem.
FillRgn	Übertragung des Bereichs zum Zielsystem (falls Bereichsdatenstruktur abfragbar) und Ausgabekommando im Zielsystem.	Ausgabekommando im Zielsystem.

Tabelle Konstruktion im Zielsystem

Vergleich der beiden oben genannten Varianten zur Übersetzung von Bereichen für eine typische Sequenz von Ausgabekommandos. Linke Spalte: Ausgabekommando. Mittlere Spalte: Übersetzeraktionen für den Fall, daß Bereiche übersetzt werden. Rechte Spalte: Übersetzeraktionen für den Fall, daß die Konstruktion von Bereichen übersetzt wird.

Der Ausgabeanalysator muß zusätzlich zu den Ausgabekommandos auch den Teil des Ressourcemanagements mitverfolgen, der sich auf Pfade und Bereiche bezieht. Der jeweilige Bereich wird nicht erst dann in das Zielgrafiksystem übersetzt, wenn er im Ausgabestrom referenziert wird,

sondern schon während er vom Anwendungsprogramm konstruiert wird. Jedes Grafikprimitiv, das in den Bereich eingeht, wird dabei individuell in das Zielgrafiksystem übersetzt und dort zum Bereich hinzugefügt [analysiert].

Ein Bereich kann effizient übersetzt werden indem die Bereichskonstruktion vom Ausgabeanalysator verfolgt und der Bereich aus den gewonnenen Informationen im Zielsystem konstruiert wird. Damit kann die u.U. aufwendige Übertragung der Bereichsdaten vermieden werden.

Sonderfall 1: Übersetzung eines Quickdraw Bereichs (Region) in ein anderes Grafiksystem, das Bereiche als Grafikprimitiv bietet.

Problem: Die Datenstruktur der Quickdraw Region ist bekannt und die Daten sind über die Referenz zugreifbar. Die Datenstruktur im Zielsystem ist nicht bekannt oder nicht installierbar, d.h. das Zielgrafiksystem akzeptiert den Bereich nicht als Datenstruktur.

Lösung: Die Quickdraw Region liegt als komprimierte Rechteckliste vor [Cohn89]. Da das Zielgrafiksystem den Bereich nicht als Datenstruktur akzeptiert, kann die Region nicht in die Datenstruktur des Zielgrafiksystems übersetzt werden. Statt dessen müssen die Funktionen des Zielgrafiksystems zur Konstruktion von Bereichen verwendet werden (z.B. Win32: BeginPath, EndPath). Durch Interpretation der Datenstruktur der Quickdraw Region wird die Rechteckliste wiedergewonnen. Aus den Rechtecken kann im Zielgrafiksystem ein Bereich konstruiert werden. Dazu müssen die Rechtecke vom Quellgrafiksystem (Quickdraw) in das jeweilige Zielgrafiksystem übersetzt werden [realisiert]. In X11 können nur Clippingmasken durch Rechtecklisten erzeugt werden, aber keine anderen Bereiche. Eine Region kann deshalb durch SetClipRectangles nur zum Clipping erzeugt werden [realisiert]. Soll ein Bereich in X11 erzeugt werden, der nicht ausschließlich zum Clipping eingesetzt wird, dann muß der Umweg über echtes Zeichnen von Rechtecken (PolyRectangle) in eine 1-Bit tiefe Pixmap gegangen werden [realisiert].

5.3.3.6. Text

Alle Grafiksysteme unterstützen Textausgaben. Fast immer wirkt dabei eine große Zahl von Parametern auf die Ausgabe ein. Dazu kommen verschiedene grafiksystemspezifische Besonderheiten, die in einigen Fällen eine direkte Übersetzung von Textausgaben des Quellsystems in des Zielsystems erschweren. Grundsätzlich können alle Textausgaben, für die es keine direkte Übersetzung gibt, in Pixmaps umgewandelt und mit einer Clippingmaske ausgegeben werden. Pixmap und Clippingmaske werden, wie schon bei den anderen Grafikprimitiven diskutiert, im Quellgrafiksystem erstellt.

Übersetzung von Standardausgaben

Tatsächlich ist dieser Ansatz aber unbefriedigend, wenn, wie im Fall von Textverarbeitung, ein Großteil der Grafikausgaben aus Text besteht. Obwohl eine große Zahl von Textparametern existiert, werden bei den meisten Textausgaben nur sehr wenige Parameter wirklich variiert. Alle anderen Parameter stehen meistens auf Standardwerten oder sind unwirksam indem sie zu Null gesetzt sind. Die Variation umfaßt vor allem die Auswahlschlüssel *Zeichensatztyp* und *Größe* und die Stile *normal*, *dick*, *unterstrichen* und *kursiv* (Messung *Textvarianten*) Diese Schlüssel sind in allen Grafiksystemen bekannt. Bei den Stilen gibt es Unterschiede auf die in Sonderfall 2 beispielhaft eingegangen wird. Textausgaben, bei denen nur diese wenigen Parameter eine Rolle spielen können bis auf Ausnahmen (siehe Sonderfall 1) direkt übersetzt werden.

Sonderfall 1: Übersetzung nach X11.

Problem: Bei X11 werden Zeichensätze nach XLFD (X Logical Font Description) benannt [Xv0-90]. Jede Parameterkombination wird als Zeichensatz behandelt und auf Anfrage vom Endgerät mit einer Referenznummer versehen. Auf die Zeichensatzvariante kann dann ohne Angabe von Stil, Größe oder anderer Parameter allein über die Referenznummer zugegriffen werden. In anderen Grafiksystemen werden die Referenznummern einzelnen Zeichensatztypen zugeordnet, nicht der Kombination aller Parameter. Die Zeichensatzvariante wird bei jeder Grafikausgabe durch die Textparameter spezifiziert.

Lösung: Für jede Zeichensatzvariante, die in einer Grafikausgabe verwendet wird, muß der Übersetzter beim Endgerät eine Referenznummer anfordern (OpenFont). Dazu ist auf ein sogenannter Round-Trip-Request erforderlich. Die gesamte Grafikausgabe des betroffenen Ausgabestroms wird solange angehalten. Der Übersetzer führt Buch über die Referenznummern und Parameter der im Lauf der Sitzung verwendeten Zeichensatzvarianten. Damit muß die

Ausgabe nur jeweils beim ersten Auftreten einer Zeichensatzvariante angehalten werden [realisiert].

Sonderfall 2: Übersetzung von Textausgaben mit unterstrichenem Stil nach X11.

Problem: In X11 R4/R5 ist der unterstrichene Stil nicht vorgesehen. Deshalb gibt es keine Möglichkeit zur direkten Übersetzung.

Lösung: Die Textausgabe wird ohne unterstrichenen Stil nach X11 übersetzt. Anschließend wird der Unterstrich als horizontale Linie gezeichnet. Die Parameter der Linie werden aus den Parametern der Textausgabe abgeleitet. Der vertikale Abstand von der Basislinie des Textes entspricht der Linienbreite. Die Länge ist gleich der Laufweite des auszugebenden Textes im Zielgrafiksystem. Dazu kann die Funktion zur Laufweitenmessung des Zielsystems verwenden. Wenn diese einen Round-Trip-Request auslöst, sollte statt dessen die entsprechende Funktion des Quellsystems benutzt werden [realisiert]. Unterschiedliche Laufweiten führen zu Unterstrichen falscher Länge. Der hier eventuell auftretende Fehler liegt in der gleichen Größenordnung, wie die unten diskutierte abweichende Zeichensatzdefinitionen. Er kann deshalb durch passende Zeichensätze vermieden werden.

Prinzipiell wird mit der direkten Übersetzung eines kleinen Satzes wichtiger Parameter ein Großteil der Textausgaben abgedeckt. Probleme entstehen dabei in zwei Bereichen:

1. Abweichende Zeichensatzdefinitionen
   Die Zeichensatzdefinitionen auf verschiedenen Grafiksystemen, bzw. Zeichensätze von verschiedenen Herstellern können sich unterscheiden auch wenn sie die gleiche Bezeichnung tragen.
2. Unterschiedliche Ausstattung
   Auf verschiedenen Grafiksystemen gehören verschiedene Zeichensätze zur Grundausstattung. Die Menge der auf allen Systemen verfügbaren Zeichensatztypen ist sehr klein. Zusätzliche Zeichensätze müssen fast immer beim Zielgrafiksystem installiert werden. Sie können nur in Ausnahmefällen zur Laufzeit der Grafikengine des Zielsystems zur Verfügung gestellt werden.

Abweichende Zeichensatzdefinitionen

Abweichende Zeichensatzdefinitionen verursachen nur geringe Unterschiede bei einzelnen Textausgaben, d.h. einzelnen Zeichen oder Zeichenketten. Oft werden aber mehrere Zeichenketten abhängig von ihrer Laufweite relativ zueinander positioniert. Die Position wird von den Anwendungsprogrammen berechnet. Die meisten Grafiksysteme unterstützen die Anwendungsprogramme dabei durch Funktionen, welche die Laufweite einer Zeichenkette bei bestimmten Textparametern messen. Entsprechen sich die Laufweiten nicht exakt, dann erscheinen Lücken zwischen Zeichenketten oder Zeichenketten laufen ineinander. Der Fehler hängt von der Differenz der Laufweiten ab. Er wird beeinflußt durch die Länge der Zeichenketten. Je länger die Zeichenkette desto größer ist der akkumulierte Fehler.

Der Fehler kann durch folgendes Verfahren zum Teil unterdrückt werden: Der Übersetzer bestimmt die Differenz zwischen Laufweiten in Quellsystem und Zielsystem. Falls diese mehrere Pixel (bzw. typografische Punkte) voneinander abweichen, kann der Übersetzer den Fehler stark verringern indem die Ausgabe einer Zeichenkette in mehrere Einzelausgaben von Teilstücken aufgespalten wird. Die Koordinaten der Einzelausgaben bestimmen sich aus der Laufweite der Teilstücke im Zielsystem [analysiert].

Unterschiedliche Zeichensatzausstattung

Unterschiedliche Ausstattung mit Zeichensätzen ist kein prinzipielles, sondern ein praktisches Problem. Während der gemeinsamen Bearbeitung von Dokumenten muß jeder der verwendeten Zeichensätze auf allen Endgeräten verfügbar sein. Bei den meisten Grafiksystemen müssen die Zeichensätze schon präsent sein, wenn die verwendeten Anwendungsprogramme starten, da diese oft in der Initialisierungsphase die verfügbaren Zeichensätze abfragen. Das Ressourcendefizit kann durch Installation aller notwendigen Zeichensätze gelöst werden, wenn

1. die potentiellen Partner und
2. die verwendeten Zeichensätze einer Application Sharing Sitzung bekannt sind und
3. Zeichensätze bei allen Endgeräten installiert werden können.

Ist eine der drei Bedingungen nicht erfüllt, dann muß mit dem Auftreten des Ressourcedefizits gerechnet und es müssen Vorkehrungen getroffen werden, um die negativen Auswirkungen zu mindern.

Sonderfall 3: Behebung des Ressourcedefizits bei Übersetzung nach und Darstellung unter X11.

Problem: Ab der Release 5 des X Window Systems können X11-Endgeräte (X-Server) Zeichensatzvariationen von einem Zeichensatzdienst im Netzwerk (Font-Server) beziehen. Damit ist eine Mechanismus gegeben, um einem darstellenden X-Server zur Laufzeit zusätzliche Zeichensätze zur Verfügung zu stellen. Der Font-Server benötigt Zugriff auf die von Anwendungsprogrammen im Quellsystem verwendeten Zeichensätze.

Lösung: Bereitstellung eines X11 Font-Servers auf dem Quellsystem als Komponente des sendenden Teils des Ausgabereplikators. Der Font-Server hat Zugriff auf alle Zeichensätze des Quellsystems. Anwendungsprogramme verwenden nur die verfügbaren Zeichensätze oder suchen selbst Alternativen. Im Ausgabestrom werden deshalb nur die im Quellsystem verfügbaren Zeichensätze referenziert. Ein Font-Server im Quellsystem kann Endgeräten, alle im Ausgabestrom referenzierten Zeichensätze zur Verfügung stellen. Damit ist der Zugriff des Endgerätes auf alle benötigten Zeichensatzvariationen gesichert. Insbesondere ist die Laufweite von Zeichenketten in Quellsystem und Zielsystem identisch. Unterscheiden sich die Darstellungsmöglichkeiten, bzw. Auswahlmöglichkeiten von Zeichensatzvariationen, kann trotzdem eine, wie oben beschriebene Ersatzausgabe nötig sein.

Sonderfall 4: Abweichende Lauflängen.

Problem: Im Zielsystem ist ein ähnlicher Zeichensatz verfügbar. Die Lauflängen unterscheiden sich aber geringfügig.

Lösung: Lauflängenunterschiede wirken sich vor allem bei langen Zeichenketten aus. Um negative Effekte zu minimieren, können Zeichenketten zerlegt werden. Die Fragmente werden jeweils absolut positioniert.

Tritt der Fall ein, daß eine im Ausgabestrom verwendete Zeichensatzvariation auf einem Endgerät nicht verfügbar ist und nicht zur Laufzeit verfügbar gemacht werden kann, dann muß der Übersetzer einen Ersatz finden. Die einfachste Variante ist der Ersatz durch eine Pixmapausgabe. Besonders Pixmapausgaben einfarbiger Zeichenketten, die durch fehlende Zeichensätze ausgelöst werden, können im Vergleich zu anderen Pixmapausgaben sehr effizient durchgeführt werden. Die Textausgaben werden im Quellgrafiksystem in eine Bitmap ausgegeben. Diese Bitmap dient sowohl als 1 Bit tiefe Pixmap, als auch als Clippingmaske. Je nach Zielgrafiksystem braucht die Bitmap nur einmal zum Zielgrafiksystem übertragen werden und kann trotzdem beide Funktionen erfüllen.

Textkodierung durch komprimierte Bitmaps

Eine unkomprimierte Bitmap als Ersatz für die Textausgaben des obigen Absatzes (944 Zeichen) besteht bei Times/10p (p= Pixel, da für Bildschirmausgaben Pixel relevant sind, nicht typographische Punkte) aus ca. 5700 Byte, d.h. etwa dem 6-fache Datenvolumen einer entsprechenden UTF-8 Zeichenkette. Die tatsächliche Größe der Bitmap hängt natürlich stark vom gewählten Zeichensatz ab.

Für Endgeräte, die komprimierte Bitmaps zulassen, sinkt der relative Aufwand bei LZW-Kodierung um 38%, d.h. auf ein 3,7-faches Datenvolumen im Vergleich zu UTF-8. Die Kompressionsrate von Kodierungen der LZ-Familie hängt allerdings von der Größe der Eingabedaten ab. Bei sehr kleinen Bitmaps, die für Textausgaben in Grafikströmen typisch sind, liegt der Datenaufwand bei ca. 1 Bit pro Pixel (gemessen bei 75x12 Pixel entsprechend ca. 15 Zeichen Times/10p). Der Aufwand reduziert sich auf 0,5 - 0,6 Bit pro Pixel für sehr große Bitmaps angefüllt mit Times/10p. Da sich alle Bitmaps, die aus Zeichensätzen ähnlicher Größe entstehen, sehr ähnlich sind, kann man eine Neuinitialisierung des LZW-Kodierers vermeiden. Alle Bitmaps werden dann, wie eine sehr große Bitmap mit fortlaufendem Fenster kodiert und erreichen damit die Kompressionsrate sehr großer Bitmaps.

> Bei LZW-Komprimierung von Bitmaps für Textdarstellungen ist der Zustand von Kodierer und Dekodierer über alle Textausgaben hinweg fortzuführen.

Die exakte Kompressionsrate, bzw. die Zahl der Bits pro Pixel hängt vom Zeichensatz und besonders von der Zeichengröße ab. Abbildung *BitmapText-BpP* zeigt den Zusammenhang zwischen der Zeichengröße und dem Datenaufkommen. Man sieht deutlich, daß die Kompression pro bedeckter Fläche bei großen Zeichensätzen deutlich steigt. Der Grund dafür ist die größere Zahl gleichförmiger Bitstrecken bei großen Zeichensätzen.

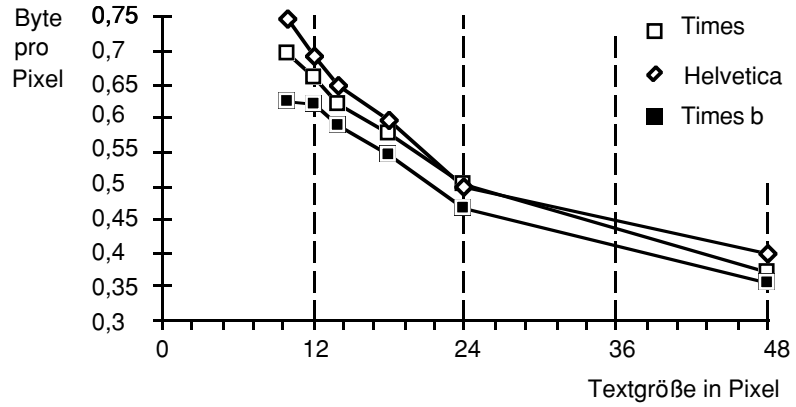


Abbildung BitmapText-BpP

Die Anzahl Bits pro bedeckter Fläche sinkt bei Bitmaps, die aus großen Zeichensätzen stammen. Große Zeichen werden besser komprimiert. Diese Grafik gibt den Grenzfall sehr großer Bitmaps, d.h. kontinuierlich laufender Kodierer wieder.

Besonders wichtig im Vergleich zwischen Textübertragung durch komprimierte Bitmaps und ASCII Text ist der Aufwand pro Zeichen. Abbildung *BitmapText-BpC* zeigt die Abhängigkeit des Datenaufkommens von der Textgröße. Entgegen der Erwartung wächst das Datenvolumen nicht quadratisch mit der Zeichengröße, sondern im wichtigen Bereich zwischen 10 und 24 Pixel fast linear. Komprimierte Bitmaps benötigen generell mehr Daten, als ASCII Text. Der Aufwand beginnt bei einem Faktor von ca. 3 für die kleinsten auf Bildschirmen lesbaren Zeichen (Times/10p: 3,1; Helvetica/10p: 3,7, TimesBold/10p: 2,8). Er steigt auf ca. 10 für Zeichensätze mit 18 Pixel Höhe. Dies gilt auch für Grafiksysteme mit virtuellen Koordinaten, die Zeichen skalieren, da auch auf diesen letztlich Pixel ausgegeben werden und die gleiche Ausgabe mit anderer Skalierung durch möglichst kleine Bitmaps erzeugt werden kann.

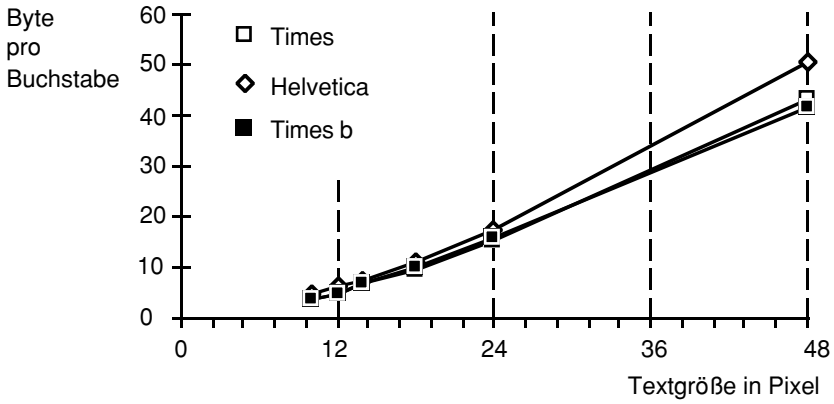


Abbildung BitmapText-BpC

Anzahl Bytes je dargestelltes Zeichen. Das Datenvolumen komprimierter Bitmaps wächst im wichtigen Bereich fast nur linear mit der Textgröße.

Ein 10-facher Aufwand für die Darstellung von Zeichen mit 18 Pixeln ist natürlich nicht zu vernachlässigen. Dieser Aufwand relativiert sich aber dadurch, daß auf gleicher Fläche entsprechend weniger Zeichen ausgegeben werden. Die Zahl ausgegebener Zeichen reduziert sich durch die verfügbare Bildschirmfläche bei den meisten Anwendungen mit dem Quadrat der Zeichengröße. Verknüpft man dies mit dem linearen Wachstum des Datenvolumens pro Zeichen, dann sinkt das übertragene Datenvolumen bei größeren Zeichensätzen. Für Times/24p (Helvetica/24p) beträgt die Reduktion im Vergleich zu Times/10p (Helvetica/10p) 32% (34%). Dies allerdings ausgehend von einem Faktor 3 Mehraufwand für Bitmaps aus 10p Zeichen im Vergleich zu ASCII Text.

Natürlich läßt sich auch ASCII Text komprimieren. Nach [BeClWi90] liegen erreichbare Kompressionsraten bei 3 bis 4 Bit pro Zeichen. Auch hier wird vorausgesetzt, daß Kodierer und

Dekodierer immer mit großen Fenstern laufen und nicht bei jeder Zeichenkette neu gestartet werden. Das Datenvolumen von komprimierten Bitmaps und komprimiertem Text unterscheidet sich damit um den Faktor 6 - 8.

Dieser Faktor bezieht sich jedoch nur auf die Nutzlast ohne Header der Netzwerk- und Grafikprotokolle. Da bei den meisten Anwendungsprogrammen jeweils nur kurze Zeichenketten ausgegeben werden, relativiert sich der oben genannte Faktor. Textausgaben umfassen höchstens eine Zeile, d.h. maximal 100 Zeichen (Times/10p), meistens weniger (im Mittel 23,53 Zeichen beim [Danskin95] text-Trace für textorientierte X11 Programme). Im Vergleich dazu besteht z.B. der kombinierte X11/TCP/IP Overhead aus 62 Byte.

Ein Zahlenbeispiel: Ein X11-ImageText8 Request mit einer komprimierten Zeichenkette von 24 Zeichen in Times/10p besteht aus 72 Bytes. Ein entsprechender PutImage Request mit LZW Kompression umfaßt 144 Bytes.

Ersetzung von Text durch komprimierte Bitmaps ist dann sinnvoll, wenn Textausgaben nicht auf einfache Art in die Darstellung des Zielgrafiksystems übersetzt werden können und Bitmapkompression möglich ist.

5.3.3.7. Koordinaten

Um einen Grafikausgabestrom pixeltreu zwischen Grafiksystemen zu übersetzen, müssen die Koordinaten der Ausgaben exakt transformiert werden. Prinzipiell werden die Koordinaten von Ausgabekommandos dafür mit Hilfe der Koordinatensystemparameter des Quellsystems in Koordinaten bezüglich des Ausgabebereichs umgewandelt. Es werden auch Relativkoordinaten in Unterfenstern berücksichtigt und über die Position des Unterfensters im Ausgabebereich in Absolutkoordinaten für den Ausgabebereich des Zielsystems transformiert. Dadurch verschwinden Bezüge auf Unterfenster. Dies ist je nach Schachtelung der Unterfenster ein mehrstufiger Prozeß. Da Bezüge auf Unterfenster verschwinden, kann der Grafikausgabestrom auch in flache Grafiksysteme übersetzt werden (Kapitel 2.3.4) [realisiert]. Die Koordinaten bezüglich des Ausgabebereichs werden mit Hilfe der Koordinatensystemparameter des Zielsystems in die Koordinaten der Ausgabekommandos im Zielsystem transformiert.

Sind Quell- und Zielgrafiksystem hierarchisch organisiert, dann kann die Hierarchie beibehalten werden. Dabei müssen die Koordinaten von Unterfenstern, wie alle anderen Koordinaten transformiert werden [realisiert].

Ist das Quellgrafiksystem nicht hierarchisch, dann gibt es einen einzigen Transformationsschritt, der die Koordinaten des Quellsystems in die des Zielsystems überführt [realisiert].

5.3.4. Übersetzerstrategie

Die Übersetzung eines Quickdraw-Grafikstroms nach X11 wurde für zwei verschiedene Anwendungen auf unterschiedliche Weise implementiert (siehe Kapitel 7). Für die Übersetzung im Rahmen des Projektes M-MaX [WoFrSchu95] wurde ein sog. kommandorientierter Ansatz gewählt. Die Erfahrungen aus diesem Projekt führten zur Wahl eines ausgabeorientierter Übersetzers im Rahmen der Diplomarbeit von M. [Maier97]. Beide Varianten werden in den folgenden Kapiteln diskutiert.

5.3.4.1. Kommandorientierter Übersetzer

Der Übersetzer erhält als Eingabe einzelne Grafikkommandos. Darunter sind Kontextmanipulationskommandos und Ausgabekommandos. Jedes Kommando wird einzeln ausgewertet und in Grafikkommandos für das Zielsystem übersetzt. So wird z.B. genau dann die Vordergrundfarbe im Zielsystem gesetzt, wenn das entsprechende Kontextmanipulationskommando im Ausgabestrom erscheint. Der Übersetzer transformiert so jedes einzelne Kommando in die Darstellung des Zielsystems.

Prinzipiell erscheinen die Kommandos im Ausgabestrom am Eingang des Übersetzers so, daß jeweils alle notwendigen Kontextparameter gesetzt werden, bevor das Ausgabekommando die Ausgabe auslöst. Dies gilt aber nur im Quellsystem. Da das Zielsystem im allgemeinen einen abweichenden Parametersatz hat, ist es nicht notwendigerweise der Fall, daß alle zur Ausgabe im Zielsystem benötigen Parameter durch transformierte Kontextmanipulationskommandos eingestellt werden. Das Zielsystem kann zur korrekten Ausgabe noch andere zusätzliche Parameter benötigen, die nicht vom Ausgabestrom bereitgestellt

werden. Dieser Fall tritt vor allem dann ein, wenn das Zielsystem andere oder mehr Funktionalität und Parameter bietet, als das Quellsystem. Bei einem Ausgabekommando kann sich der Übersetzer deshalb nicht auf einen korrekten Zustand des Grafikkontextes verlassen, sondern muß gegebenenfalls zusätzliche Parameter einstellen.

Bei jedem Ausgabekommando analysiert der Übersetzer den aktuellen Grafikkontext des Quellsystems und überprüft die, für die aktuelle Ausgabe relevanten, Felder des Grafikkontextes im Zielsystem. Je nachdem, wo sich der Übersetzer befindet und abhängig davon, ob die Grafikkontexte von Quell- und Zielsystem abfragbar sind, müssen zu diesem Zweck einer oder beide Grafikkontexte als lokale Kopie im Übersetzer mitgeführt werden.

Beim kommandorientierten Übersetzer wird jedes Kommando einzeln übersetzt. In vielen Fällen sind dies nur Kontextmanipulationskommandos, d.h. Teile von Grafikausgaben.

5.3.4.2. Ausgabeorientierter Übersetzer

Ein ausgabeorientierter Übersetzer reagiert nur auf Ausgabekommandos. Tritt ein Ausgabekommando im Ausgabestrom auf, dann werden Grafikkontext und Ausgabekommando analysiert. Der Übersetzer verhält sich anschließend wie ein Anwendungsprogramm und führt die gleiche Ausgabe basierend auf einer Programmierschnittstelle des Zielsystems durch. Der Übersetzer bildet dabei die Programmierschnittstelle des Zielsystems intern nach, rendert die Grafikausgaben aber nicht, sondern erzeugt in den Prozeduren dieser Programmierschnittstelle die Grafikkommandos für das Zielsystem.

Da der Ausgabestrom am Ausgang des Übersetzers in der Syntax des Zielgrafiksystems vorliegt, hängt die Komplexität der im Übersetzer verwendeten Programmierschnittstelle von ihrer Ähnlichkeit zur Syntax des Zielgrafiksystems ab. Für die Übersetzung nach X11 bietet sich die Xlib als Programmierschnittstelle für den Übersetzer an, da für die Xlib eine vollständige Dokumentation und Quellcode verfügbar sind.

Beim ausgabeorientierten Ansatz werden jeweils komplette Grafikausgaben auf einer Programmierschnittstelle produziert. Damit ergibt sich die gleiche Situation, wie bei der Anwendungsprogrammierung. Durch den Grafikausgabestrom am Eingang des Übersetzers vorgegebene Grafikausgaben werden auf der Programmierschnittstelle des Zielsystems gewissermaßen gezeichnet. Dies hat den Vorteil, daß Programmierer die gewohnte Sichtweise beibehalten können, statt sich mit der Transformation von einzelnen Grafikkommandos zu beschäftigen.

5.4. Fensterverwaltung

Zum Betrieb von Anwendungsprogrammen gehört neben den Fensterinhalten auch die Fensterverwaltung. Wie in Kapitel 2.3.3 beschrieben, haben Fenster neben der Anordnung von Grafikausgabeströmen aber noch andere Funktionen. Die Funktionen können im Quell- und Zielfenstersystem verschieden sein. Dies bedeutet, daß nicht notwendigerweise für jedes Fenster im Quellsystem ein Fenster im Zielsystem angelegt wird. Ist aber mit einem vom Anwendungsprogramm angelegten Fenster ein Ausgabebereich verbunden und sind Grafikausgaben in diesen Ausgabebereich zu erwarten, dann muß ein entsprechender Ausgabebereich für das Zielsystem angelegt werden. Der Ausgabebereich für das Zielsystem wird entweder im Zielfenstersystem oder im Übersetzer angelegt. Dies hängt von der Funktionalität des Zielfenstersystems ab. Wird der Ausgabebereich nur zur Übersetzung von Koordinaten benötigt, dann kann der Übersetzer diese Aufgabe übernehmen, ohne ein Fenster im Zielfenstersystem anzulegen.

Weitere wichtige Funktionen der Fensterverwaltung betreffen die Manipulation existierender Fenster. Zu diesem Zweck bieten verschiedene Fenstersysteme ähnliche Funktionen (Kapitel 2.3.6.1). Wie in Kapitel 2.3.6.2 beschrieben, kann der Ablauf der Fenstermanipulation je nach Window Manager aber verschieden sein. Vor allem bei der Übersetzung von Fenstermanipulation zwischen aktiven und passiven Window Managern müssen besondere Vorkehrungen getroffen werden.

Wie in Kapitel 5.3.3 für die Übersetzung von Grafikausgaben sollen hier für die Übersetzung der Fensterverwaltung Lösungen vorgestellt werden. Zusätzlich werden einzelne Problemfälle herausgegriffen und dafür spezielle Lösungen diskutiert.

5.4.1. Öffnen und Schließen von Ausgabebereichen

Unabhängig davon, wie viele und zu welchem Zweck ein Anwendungsprogramm Fenster im Sinne der Fensterverwaltung erzeugt, gibt es immer eines oder mehrere Fenster, in dem alle Grafikausgaben erscheinen. Diese Fenster werden als Toplevel-Fenster bezeichnet (siehe Kapitel 2.3.4).

Für jedes Toplevel-Fenster im Quellgrafiksystem wird ein Toplevel-Fenster im Zielsystem angelegt. Dies gilt für alle Ausgabebereiche, die als Toplevel-Fenster dargestellt werden. Dazu gehören Dokumentenfenster, Dialogfelder (Dialogbox), Menüleisten (Menubar) und Knopfleisten (Buttonbar), sowie Menüs.

In den meisten Grafiksystemen ist das Anlegen eines Fensters und die Existenz des Fensters als Datenstruktur unabhängig von dessen Sichtbarkeit. Es gibt deshalb im allgemeinen sowohl Kommandos zum Erzeugen und Vernichten der Datenstruktur (Create/Destroy-Window), als auch Kommandos zum Darstellen und Verbergen (Show/Hide-Window). Die Möglichkeit unsichtbare Fenster anzulegen oder Fenster temporär zu verbergen ist vor allem softwaretechnisch nützlich, weil oft mit der Datenstruktur des Fensters noch zusätzliche anwendungsspezifische Datenstrukturen verbunden sind, die dann beim Darstellen und Verbergen eines Fensters nicht wiederholt erzeugt werden müssen.

> Für die entfernte Darstellung ist nur die Sichtbarkeit, nicht die Existenz eines Fensters bedeutsam. Das Signal zum Erzeugen eines Fensters im Zielfenstersystem wird deshalb aus dem Darstellungskommando im Quellfenstersystem, nicht aus dem Erzeugungskommando abgeleitet.

Dies hat den Vorteil, daß nur sichtbare Fenster im Zielsystem angelegt werden müssen. Unsichtbare Fenster, die anderen Zwecken, als der Grafikausgabe, z.B. der Nachrichtenverarbeitung dienen, werden vom Übersetzer ignoriert und im Zielfenstersystem nie erzeugt.

Sonderfall 1: Zuordnung eines Ausgabebereichs zu Grafikkontexten ohne vom System unterstützten Zuordnungsmechanismus.

Problem: In manchen Grafiksystemen können Anwendungsprogramme über verschiedene Grafikkontexte in das gleiche Fenster zeichnen. Zu diesem Zweck werden Grafikkontexte irgendwann zur Laufzeit angelegt. Das Anlegen des Grafikkontextes ist nicht korrelliert mit dem Öffnen eines Ausgabebereichs. Die im Grafikkontext angegebene Größe und die Position des Koordinatenursprungs müssen nicht notwendigerweise mit den Daten des beschriebenen Fensters übereinstimmen (Bsp: Quickdraw, siehe auch Kapitel 2.3.7). Als Folge gibt es kein Signal mit räumlich und zeitlich korrekter Information zum Öffnen und Schließen eines Ausgabebereichs.

Lösung: Es gibt keine allgemeine und zuverlässige Lösung des Problems. Die Lösung hängt vom Quellfenstersystem bzw. der Kombination von Quellfenstersystem und Quellgrafiksystem ab. In vielen Fällen wird die Lösung auch von einzelnen Anwendungsprogrammen oder sogar Implementierungsvarianten bestimmt.

Eine allerdings häufig mögliche, d.h. für viele Anwendungsprogramme gültige, Lösung ist die Zuordnung über die absolute Position und Größe des Grafikkontextes und des an der gleichen Position auf dem Bildschirm sichtbaren Fensters. Zeichnet ein Anwendungsprogramm in einen Grafikkontext dem nicht eindeutig ein Fenster zugeordnet ist, und stimmen die absoluten Positionen von Fenster und Grafikkontext überein, dann können die Grafikausgaben mit hoher Wahrscheinlichkeit dem entsprechenden Fenster zugeordnet werden [realisiert]. Alternativ kann auch das oberste sichtbare Fenster bestimmt und diesem die Ausgaben zugeordnet werden. Diese Methode birgt allerdings die Gefahr, daß Ausgaben, die für verdeckte Fenster bestimmt sind und deshalb von der Grafikengine ausgeblendet werden, fälschlicherweise sichtbar werden [realisiert].

Sonderfall 2: Übersetzung mehrerer Toplevel-Fenster in Einfenstersysteme

Problem: Mehrere unabhängige Ausgabebereiche können nicht gleichzeitig im Zielfenstersystem angelegt werden.

Lösung: Verwaltung aller Ausgabebereiche im Übersetzer. Neben der Koordinatentransformation muß der Übersetzer auch das Clipping übernehmen. Die Lösung entspricht der Darstellung, die von X-Serveremulationen innerhalb eines Fensters des Gastfenstersystems bekannt ist.

Sonderfall 3: Übersetzung nichtrechteckiger Fenster im Quellsystem.

Problem: Sind Toplevel-Fenster nicht rechteckig, dann gibt es nur dann eine Möglichkeit zur direkten Übersetzung in das Zielfenstersystem, wenn auch das Zielfenstersystem nicht-rechteckige Fensterformen bietet.

Lösung: Bietet das Zielfenstersytem nur rechteckige Fenster, dann besteht die Lösung darin, das umschließende Rechteck als Fenster im Zielsystem anzulegen und dessen Clippingbereich entsprechend der echten Fensterform zu setzen.

Gibt es im Zielfenstersystem keinen Clippingbereich für Fenster, aber transparente Fenster, transparente Fensterhintergründe oder neben einer rechteckigen Grundform die Möglichkeit einen sichtbaren Auschnitt anzugeben, dann kann eine beliebige Fensterform simuliert werden, indem der überstehende Bereich ausgeblendet wird [realisiert].

Sonderfall 4: Übersetzung der Macintosh Menüzeile in Fenstersysteme ohne bilddschirmweite Menuzeile.

Problem: Die Menüzeile ist kein Fenster. Sie wird durch einen Übersetzungsmechanismus für Fenster nicht erfaßt. Die Menüzeile ist jeweils dem Anwendungsprogramm zugeordnet, das den Eingabefokus besitzt (Kapitel 2.3.3.3). Die Darstellung der Menüzeile eines nicht gemeinsam bearbeiteten Anwendungsprogramms soll vermieden werden.

Lösung: Die Menüzeile ist ein Ausgabebereich, dessen Größe und Position bestimmt werden kann. Für die Menüzeile wird im Zielfenstersystem ein Fenster angelegt für das es keine direkte Entsprechung im Quellfenstersystem gibt. Das Macintosh-System gibt bei der Umschaltung des Eingabefokus Signale, die dazu verwendet werden können, die Menüzeile im Zielfenstersystem zu aktivieren und zu deaktivieren. Hat keines der gemeinsam bearbeiteten Anwendungsprogramme den Eingabefokus im Quellsystem, dann wird die Menüzeile im Zielfenstersystem entweder geschlossen oder unkenntlich gemacht, so daß nur die Ausgaben gemeinsam bearbeiteter Programme im Zielsystem beobachtet werden können [realisiert].

5.4.2. Fenstermanipulation

Die wichtigsten Aktionen im Bereich der Fenstermanipulation sind Positions- und Größenveränderung. Entsprechende Kommandos von Anwendungsprogrammen des Quellsystems können im allgemeinen direkt in entsprechende Kommandos im Zielfenstersystem übersetzt werden.

Im Gegensatz dazu lassen sich benutzerinitiierte Konfigurationsänderungen nicht direkt von einem Fenstersystem in das andere übertragen. Der Grund dafür ist vor allem ein möglicher Konflikt zwischen aktiver und passiver Fensterverwaltung, aber auch ein eventueller Informationsmangel im Quellfenstersystem.

Die Übersetzung von benutzerinitiierten Positionsänderungen hängt von der Strategie des Application Sharing Systems ab. Programmbasiertes bzw. fensterbasiertes Sharing erlaubt im Gegensatz zum Screen-Sharing eine individuelle Anordnung der Toplevel-Fenster auf jedem Endgerät. Soll diese Freiheit genutzt werden, dann werden benutzerinitiierte Positionsänderungen nicht in das Zielsystem übertragen.

5.4.2.1. Aktive Fensterverwaltung im Quellsystem

Bei aktiver Fensterverwaltung führt der Benutzer mit Hilfe des Window Managers Konfigurationsänderungen durch. Die Aktion wird zwar geleitet durch Vorgaben des betroffenen Anwendungsprogramms (siehe Kapitel 2.3.6), wird aber nicht aktiv von diesem beeinflußt. Nach dem Ende der Aktion wird das Anwendungsprogramm über die Aktion und ihr Ergebnis benachrichtigt. In manchen Grafiksystemen geschieht dies nicht automatisch, sondern nur dann, wenn sich das Anwendungsprogramm für die Benachrichtigungen registriert hat. Ignorieren Anwendungsprogramme die Meldungen des Window Managers, dann fehlt die Information, die notwendig ist, um die gleiche Konfiguration im Zielsystem herzustellen. Das Sharing System muß deshalb unabhängig vom Anwendungsprogramm die entsprechenden Informationen beim Window Manager beantragen.

Bei aktiver Fensterverwaltung im Quellsystem spielt es keine Rolle, ob die Fensterverwaltung des Zielsystems aktiv oder passiv ist. Benutzerinitiierte Konfigurationsänderungen können vom Quellsystem wie vom Zielsystem ausgelöst und an das jeweils andere System weitergegeben werden. Da, wie oben bemerkt, Größenänderungen unter den Vorgaben des Anwendungsprogramms ablaufen, ist es dabei wichtig, daß die gleichen Vorgaben auch für den Window Manager im Zielsystem gelten. Ist dies nicht möglich, so muß die Freiheit, Konfigurationsänderungen vom Zielsystem aus durchzuführen, so eingeschränkt werden, daß die Vorgaben des Anwendungsprogramms eingehalten werden können.

5.4.2.2. Passive Fensterverwaltung im Quellsystem

Ist die Fensterverwaltung des Quellsystems passiv, dann führt das Anwendungsprogramm alle Konfigurationsänderungen selbst durch. Es kann sich dabei der Hilfe des Window Managers bedienen, bestimmt aber anschließend die neue Konfiguration per Konfigurationskommando (z.B. Größenänderung). Die benutzerinitiierten Konfigurationsänderungen im Quellfenstersystem unterscheiden sich deshalb aus der Sicht des Application Sharing Systems nicht von solchen, die von Anwendungsprogrammen ausgelöst werden.

Problematisch sind benutzerinitiierte Konfigurationsänderungen, wenn das Zielsystem mit einer aktiven Fensterverwaltung arbeitet. Der Window Manager des Zielfenstersystems ist dann in der Lage, wie oben diskutiert, Konfigurationsänderungen selbst durchführen und nach ihrem Abschluß das Anwendungsprogramm - in diesem Fall das Application Sharing System - zu informieren. Dies widerspricht der Strategie des Quellfenstersystems und möglicherweise den Anforderungen des Anwendungsprogramms. Das Anwendungsprogramm ist nicht in der Lage Vorgaben abzugeben um den Prozeß zu beeinflussen. Dies ist im Quellfenstersystem nicht notwendig, da der Prozeß der Konfigurationsänderung vom Anwendungsprogramm direkt kontrolliert wird. Entsprechend der Aussage im vorangegangenen Kapitel muß deshalb die Freiheit des Window Managers im Zielsystem, Konfigurationsänderungen durchzuführen sehr stark eingeschränkt werden. Konfigurationsänderungen können vom Zielfenstersystem aus nur über Kontrollelemente des Anwendungsprogramms durchgeführt werden, soweit diese im Zielfenstersystem erreichbar sind.

Kontrollelemente der Fensterverwaltung des Quellfenstersystems, die zum Fensterrahmen gehören, sind im allgemeinen im Zielsystem nicht verfügbar, da der Fensterrahmen durch den Rahmen des Zielfenstersystems ersetzt wird. Das Zielfenstersystem bietet zwar ähnliche oder gleiche Funktionen, aber über eigene Kontrollelemente.

Viele Operationen sind deshalb vom Zielsystem aus nicht durchführbar. Dazu gehört z.B. die Standardgröße-Funktion des Macintosh System 7/8 Window Managers, deren Knopf im Fensterrahmen untergebracht ist. Die Kontrollelemente zur Größenänderung sind dagegen Teil des Fensterinhalts und deshalb auch im Zielsystem verfügbar. Dies führt im Zielsystem zu der ungewohnten Situation, daß Größenänderungen nicht durch die Kontrollelemente des lokalen Window Managers ausgelöst werden können, sondern nur durch die vom Anwendungsprogramm, durch das Application Sharing System gezeichneten Kontrollelemente im Inhaltsbereich des Fensters.

Sonderfall 1: Passive Fensterverwaltung im Quellsystem und aktive Fensterverwaltung im Zielsystem.

Übersetzung der Bedienung des Kontrollelements zum Schließen eines Fensters.

Problem: Das Kontrollelement des Quellsystems zum Schließen eines Fensters ist im Zielsystem nicht erreichbar. Eine Bedienung des Kontrollelements des Zielsystems zum Schließen eines Fensters kann nicht direkt in das Quellsystems übersetzt werden, da im Quellsystem das Anwendungsprogramm den Vorgang kontrollieren muß, im Zielsystem die Information über den Vorgang aber erst nach Abschluß des Vorgangs vorliegt.

Lösung: Die Benutzeranweisung zum Schließen eines Fensters kann simuliert werden, indem im Quellsystem eine entsprechende Meldung in die Meldungswarteschlange eingefügt wird. Bietet das Fenstersystem keinen entsprechenden Meldungstyp, dann können dazu auch Tastaturshortcuts verwendet werden. Gegebenenfalls kann durch Auswertung der Menüs überprüft werden, ob ein entsprechender Tastaturshortcut zulässig ist. Diese Methode wurde zwischen MacOS 7/8 und X11 [realisiert].

6. Eingabekonvertierung

Dieses Kapitel behandelt die Konvertierung von Eingaben. Die Übersetzung von Eingaben ist mit wesentlich geringerem Aufwand verbunden, als die Übersetzung von Grafikkommandos. Nur bei indirekten Benutzereingaben ist eine genauere Betrachtung notwendig.

6.1. Einleitung

Beim Application Sharing greifen mehrere Teilnehmer auf ein Anwendungsprogramm zu. Im heterogenen Umfeld bedeutet dies, daß Benutzereingaben von mehreren verschiedenen Systemen gesammelt und einem Programm zugeführt werden können. Die Prozesse des Serialisierens mehrerer Eingabeströme und das Einfügen des kombinierten Eingabestroms sind in den Kapiteln 4.4 und 4.5 beschrieben. In diesem Kapitel werden die Möglichkeiten zur Übersetzung von Benutzereingaben zwischen den Systemen diskutiert.

Alle Eingaben, die auf verschiedenen Systemen erzeugt werden, werden dem gemeinsam benutzten Programm zugeführt. Sie müssen deshalb so aufbereitet werden, daß sie der Spezifikation des Quellsystems entsprechen, d.h. wie Eingaben nach der Spezifikation des Quellsystems aussehen. Sie müssen also in die Darstellung des Quellsystems übersetzt werden.

Generell ist die Übersetzung von Eingaben wesentlich weniger komplex, als die Ausgabeübersetzung. Jede Eingabemeldung kann direkt übersetzt werden, entsprechend dem kommandoorientierten Übersetzer (Kapitel 5.3.4.1) auf der Ausgabeseite.

Anmerkung: Die Begriffe Quellsystem und Zielsystem werden hier im Sinne der Ausgabekonvertierung verwendet. Um die Begriffe nicht zu verwirren sollen sie die gleichen Systeme bezeichnen, obwohl sich die Richtung der Übersetzung bei der Eingabekonvertierung im Vergleich zur Ausgabekonvertierung natürlich umkehrt.

6.2. Direkte Benutzereingaben

6.2.1. Mauszeigerbewegung

Bei der Übersetzung der Mauszeigerbewegung werden die Koordinaten, wie bei der Grafikausgabe, transformiert. Als Beispiel sei hier ein Auszug aus der Implementierung das Eingabeübersetzers von X11 nach MacOS aufgeführt:

```
CASE theMsg.message OF
   ...
   kMotion:
      thePort := X2MacgetPortByWindow(theMsg^.motion.window);
      SetPt(thePos, theMsg^.button.eventX + thePort^.portRect.left,
              theMsg^.button.eventY + thePort^.portRect.top);
      event.what := osEvt;
      event.where := thePos;
      event.when := 0;
      event.message := thePort;
      event.modifiers := 0;
   ...
END; {CASE}
```

Wenn das Quellsystem bei der Mauszeigerbewegung den Zustand der Maustasten mitliefert, das Zielsystem aber nicht, dann muß das Application Sharing System den Zustand der Maustasten im Zielsystem (für jedes Endgerät) zwischenspeichern und bei jeder Mauszeigerbewegung in die Eingabemeldung einsetzen. Das gleiche gilt für den Zustand der Modifizierungstasten. Im obigen Beispiel wird der Zustand von Maustasten, Modifizierungstasten auf Treiberebene eingefügt (siehe Kapitel 4.5.2.1).

6.2.2. Mausklicks

Die Übersetzung von Mausklicks und Tastatureingaben ist fast identisch zur Mauszeigerbewegung. Darüber hinaus sind aber bei Mausklicks und Tastatureingaben unter bestimmten Umständen spezielle Vorkehrungen notwendig.

Auf Plattformen, wo der Zustand von Eingabegeräten direkt abgefragt werden kann, muß die Einspeisung der übersetzten Eingaben auf Schicht 2 des Eingabesystems erfolgen (Kapitel 4.5.2). Damit ist aber gleichzeitig das System für die Zuordnung der Eingaben zu Eingabekanälen zuständig.

Mausklicks, werden auf Schicht 2 durch Treiber in globalen (Bildschirm) Koordinaten eingespeist. Tastatureingaben, die durch Treiber eingespeist werden, sind ebenfalls systemweit sichtbar. Auf Treiberebene gibt es noch keine Zuordnung zu Eingabekanälen. Wie in Kapitel 2.3.5 beschrieben, werden Benutzereingaben über den Eingabefokus den Eingabekanälen zugeordnet. Das Sharing System muß deshalb den Eingabefokus so einstellen, daß das System die global eingespeisten Eingaben den richtigen Eingabekanälen zuordnen kann. Der Eingabefokus wird dafür auf das Original des in der Eingabemeldung referenzierten Fensters eingestellt. Zu diesem Zweck muß u.U. die Stapelfolge von Fenstern und Programmen manipuliert werden.

Die Umschaltung des Eingabefokus ist eine Funktionalität des Eingabeinjektors. Sie wird aber hier im Rahmen der Eingabekonvertierung diskutiert, da sie Teil der Übersetzung von Eingabemeldungen ist. Mit der Manipulation der Stapelfolge durch den Eingabeinjektor wird letztlich die in der Eingabemeldung enthaltene Kanalzuordnung in das Quellsystem übersetzt. Dies ist bei den oben genannten Systemen (z.B. MacOS 7/8) auf andere Art nicht möglich.

6.2.3. Tastatureingaben

Für Tastatureingaben müssen die in den Eingabemeldungen des Zielsystems enthaltenen Zeichencodes in die des Quellsystems übersetzt werden. Dafür ist natürlich eine Zuordnungstabelle notwendig. Diese Tabelle wird mit dem Application Sharing System ausgeliefert. Sie sollte aber vom Benutzer änderbar sein.

Es hat sich bewährt, die Einstellung der Zuordnungstabelle als Teil des Sharing Systems zu realisieren: Der Eingabekonverter präsentiert dem entfernten Benutzer auf dessen Anforderung ein Kontrollfeld zur Konfiguration in dem die im Quellsystem existierenden Tasten abgebildet sind. Der Eingabekonverter verwendet für diese Grafikausgabe die Schnittstelle des Ausgabeanalysators. Die Zuordnung von Tastaturcodes geschieht dann, indem der entfernte Benutzer die Darstellung einer Taste auswählt (z.B. mit der Maus) und die entsprechende Taste drückt.

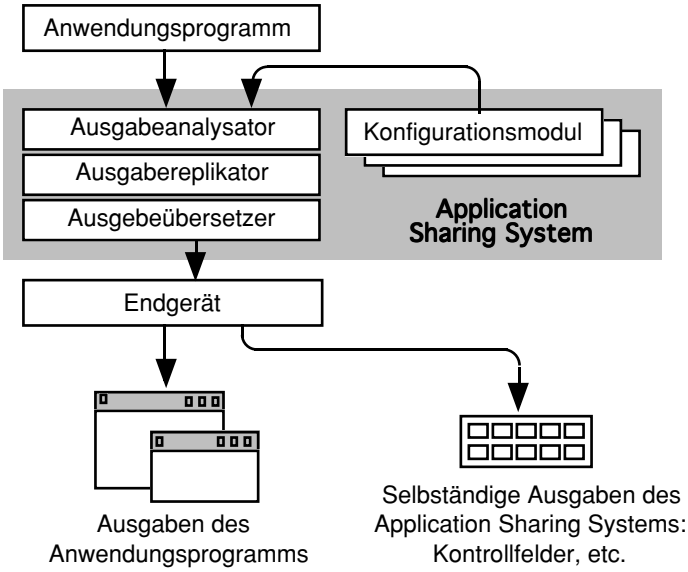


Abbildung Eigenausgaben des Sharing Systems

Alle selbständigen Ausgaben eines Application Sharing Systems, die nicht auf Ausgaben von Anwendungsprogrammen zurückgehen, müssen die Grafikschnittstelle des Ausgabeanalysators verwenden. Dazu gehören Darstellungen der Bedienungsoberfläche, Konfigurationsmöglichkeiten und sogar Telepointer.

Auf diese Weise werden für Ausgaben des Sharing Systems automatisch alle Dienste des Ausgabekonverters und des Ausgabereplikators für die Anzeige im Zielsystem verwendet. Das Endgerät benötigt keine Funktionalität, die über die Anzeige des Grafikstroms hinaus geht.

6.3. Indirekte Benutzereingaben

6.3.1. Fensterverwaltung

Es ist eine der wesentlichen Eigenschaften von Application Sharing Systemen, daß die Positionen von Fenstern in Quell- und Zielsystem unabhängig voneinander sind. Das gleiche gilt für die Stapelfolge mit der Ausnahme, daß modale Fenster des Quellsystems auf allen Endgeräten im Rahmen des Sharing Systems modal dargestellt werden müssen. Dies wird gewährleistet durch entsprechende Konfiguration der Fenster in der Ausgaberichtung.

Man kann drei Gruppen von Ereignissen unterscheiden:

1. Positionierung und Stapelfolge
   Die Meldungen werden ignoriert und nicht übersetzt. Für den Sichtbarkeitsstatus gelten die gleichen Grundsätze, wie für die Position.
2. Größenänderung
   Stimmt die Fensterverwaltungsstrategie (Kapitel 2.3.6.2 des Zielsystems mit der des Quellsystems überein, dann ist fast in allen Fällen eine direkte Umsetzung möglich. Bei unterschiedlicher Fensterverwaltungsstrategie sind die gleichen Anpassungen durchzuführen, wie bei der Ausgabekonvertierung der Fensterverwaltung (siehe Kapitel 5.4.2).
3. Existenz
   Für das Schließen von Fenstern gelten die gleichen Regeln, wie bei 2). Meldungen über die Erzeugung von Fenstern werden nur innerhalb des Sharing Systems verarbeitet, aber nicht übersetzt.

6.3.2. Redraw-Events

Redraw-Events (Update-Events), veranlassen die Applikation, ein Fenster neu zu zeichnen. Ist das Zielsystem in der Lage nur Fensterteile (spezifiziert durch Rechtecke oder Bereiche) zum Update zu veranlassen, das Quellsystem aber nicht, dann kann der Bereich auch ignoriert werden [realisiert].

Bei der Einspeisung von Redraw-Events ist zu beachten, daß Anwendungsprogramme als Folge unter Umständen große Datenmengen erzeugen. Generell zeigt jedes Redraw-Event die Notwendigkeit zur Bereitstellung der entsprechenden Daten an. Die Daten können ohne weitere Vorkehrungen (z.B. Einsatz von CE/SC) nur vom Anwendungsprogramm zur Verfügung gestellt werden. Das Anwendungsprogramm muß also durch Weiterleitung der Redraw-Events zur Ausgabe der Daten veranlaßt werden. Redraw-Events mehrerer Endgeräte können aber nicht ohne weiteres zusammengefaßt werden, da alle Events sequentiell auftreten und bei Ihrem Auftreten ohne Verzögerung weiter geleitet werden müssen. Dadurch kommt es in der Praxis eventuell zu mehreren aufeinanderfolgenden Redraw-Events für die gleichen Fensterbereiche. Da auch Redraw-Events als indirekte Eingaben der Eingaberechtsvergabe unterliegen tritt der Fall aber nur beim freien Eingabemodus mit automatischer Umschaltung von Eingabeströmen auf.

Durch das in Kapitel 4.3.4.2 vorgestellte CE/SC Verfahren kann vermieden werden, daß als Folge mehrerer Redraw-Events mehrmals die gleichen Grafikausgaben über Netzwerke übertragen werden.

7. Kombination von Verteilung und Konvertierung

Diese Kapitel behandelt die Kombination der zwei Basiskomponenten in Ausgaberichtung, Ausgabeanalysator und Ausgabereplikator, mit dem Ausgabeübersetzer. Die zwei Architekturvarianten des Replikators führen in Verbindung mit den zwei Anordnungsvarianten des Übersetzers zu vier verschiedenen Designs für Application Sharing Systeme mit jeweils anderen Eigenschaften und Einsatzgebieten.

7.1. Anordnung von Komponenten

In den vorangegangenen Kapiteln wurden die Komponenten von Application Sharing Systemen diskutiert. Diese Komponenten können auf verschiedene Art zusammengeschaltet werden, um die gemeinsame Funktion zu erfüllen. Wir betrachten dazu die vier Komponenten, die mit Grafikausgabeströmen arbeiten:

1. Ausgabeanalysator (A),
2. Ausgabereplikator (R),
3. Ausgabekonverter (K) und
4. das Endgerät (E).

Das ganze Sharing System arbeitet auf dem Datenstrom, der vom Analysator gewonnen wird. Der Ausgabeanalysator ist deshalb die Komponente, die dem Anwendungsprogramm am nächsten ist und immer am Anfang der Verarbeitungskette steht. Die Endgeräte, auf deren Bildschirmen die Grafikausgaben dargestellt werden, befinden sich am anderen Ende der Verarbeitungskette. Für die zwei übrigen Komponenten, Ausgabereplikator und Ausgabekonverter, bleiben dann nur noch zwei Anordnungsvarianten: der Ausgabekonverter kann entweder zwischen Ausgabeanalysator und Ausgabereplikator oder zwischen Ausgabereplikator und Endgerät angeordnet werden.

Für den Ausgabereplikator gibt es zwei Architekturvarianten: verteilt oder monolithisch (Kapitel 4.3.3). Daraus ergeben sich prinzipiell vier Anordnungsvarianten für ein heterogenes Application Sharing System. Die vier Modelle sind in Tabelle *Sharing System Modelle* zusammengestellt.

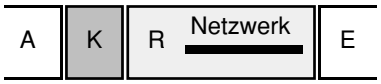
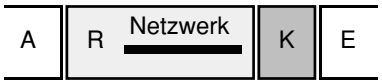
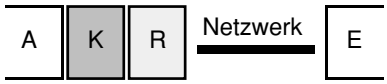
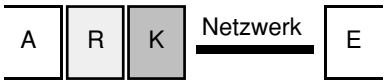
Replikator Architektur-variante:	Ausgabereplikator / Ausgabekonverter Anordnung Ausgabekonverter zwischen Ausgabeanalysator und Ausgabereplikator	Ausgabekonverter zwischen Ausgabereplikator und Endgerät
verteilt	1. Fremdsystemkapselung 	2. Übersetzung im Endgerät 
monolithisch	3. Homogene netzwerkfähige Endgeräte 	4. Heterogene netzwerkfähige Endgeräte 

Tabelle Sharing System Modelle

Die Tabelle zeigt vier Modelle. Sie entstehen durch Kombination von Anordnungsvarianten mit Replikator-Architekturvarianten. Die Grafik zeigt jeweils die Anordnung der Komponenten relativ zueinander und zum Netzwerk. Das eingezeichnete Endgerät symbolisiert mehrere mögliche angeschlossene Endgräte. Das Netzwerk wird durch einen Strich symbolisiert. Die Netzwerktopologie kann aber je nach Ausgabereplikator verschieden sein. Über der Grafik steht jeweils der Name des Modells.

Bei einem verteilten Ausgabereplikator läuft der Netzwerkverkehr innerhalb des Sharing Systems ab. Komponenten des Sharing Systems sind auf beiden Seiten des Netzwerks installiert, beim Quellsystem und bei den Zielsystemen. Ist die Architektur des Ausgabereplikator monolithisch, dann läuft der Netzwerkverkehr nicht innerhalb des Sharing Systems. Das Netzwerk verbindet Komponenten des Sharing Systems mit den Endgeräten. Es muß keine Sharing Komponente beim Endgerät installiert werden.

Ist der Ausgabekonverter zwischen Ausgabereplikator und Endgerät angeordnet, dann kann der Konverter den Ausgabestrom speziell auf die Fähigkeiten (z.B. Farbtiefe) des Endgeräts und der Verbindung zum Endgerät (Durchsatz) zuschneiden. Dies ist nicht möglich, wenn der Ausgabekonverter zwischen Ausgabeanalysator und Ausgabereplikator plaziert ist. In diesem Fall erzeugt der Konverter nur einen Ausgabestrom, der von allen Endgeräten interpretiert wird. Die Endgeräte müssen deshalb hinreichend ähnlich sein, so daß alle den Ausgabestrom gut darstellen können. Die Alternative ist, daß der Konverter einen Ausgabestrom erzeugt, der so viel Information, wie möglich trägt (z.B. nur Echtfarben) und von allen Endgeräten gut dargestellt werden kann.

Jede der in Tabelle *Sharing System Modelle* aufgeführten Modelle hat eigene Charakteristiken und ist deshalb für verschiedene Anwendungen geeignet. Die Eigenschaften der vier Modelle und ihre Anwendungen werden im folgenden Kapitel anhand von Beispielen erläutert.

7.2. Modellsysteme

7.2.1. Fremdsystemkapselung

7.2.1.1. Allgemein

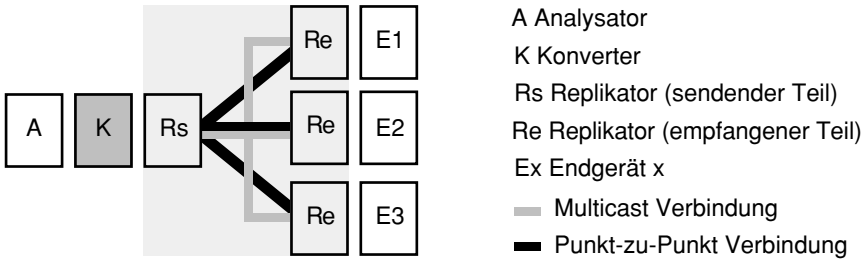


Abbildung Fremdsystemkapselung

Die in Tabelle *Sharing System Modelle* gezeigte Fremdsystemkapselung mit mehreren Endgeräten. Der verteilte Ausgabereplikators kann sowohl mehrere Punkt-zu-Punkt Verbindungen, als auch Multicast verwenden.

Es gibt Anwendungen in denen das vom Ausgabereplikator verwendete Protokoll fest vorgegeben ist. Dies ist z.B. dann der Fall, wenn verschiedene Implementierungen von Sharing Systemen über die Netzwerkschnittstelle miteinander kommunizieren sollen. Der Konverter muß dann aus den Daten des Ausgabeanalysators einen Ausgabestrom in dem vom Replikator verwendeten Protokoll (genannt Replikatorprotokoll) erzeugen. Dazu ist der Konverter zwischen Ausgabeanalysator und Replikator angeordnet. Das für das Replikatorprotokoll fremde Grafiksystem des Anwendungsprogramms wird durch den Konverter so gekapselt, daß die Kombination von Anwendungsprogramm, Ausgabeanalysator und Konverter für das übrige System wie eine Grafikstromquelle erscheint, die das Replikatorprotokoll verwendet.

Die verteilte Architektur ermöglicht dem Ausgabereplikator, jedes Grafikprotokoll für die Netzwerkübertragung anzupassen. Auch Grafikprotokolle, die nicht multicastfähig sind, können entsprechend adaptiert werden. Ressource-IDs, Ressourcenamen und Sequenznummern werden im sendenden Teil des Ausgabereplikators in eine verallgemeinerte Darstellung abgebildet und im empfangenden Teil wieder in eine für das Endgerät angepaßte Darstellung zurückgesetzt.

Durch das vom Ausgabereplikator verwendete Protokoll wird neben Quell- und Zielgrafiksystem eventuell ein drittes Grafiksystem eingeführt. Ein Beispiel: die Ausgaben eines auf Windows 3.11 laufenden Programms werden nach X11 konvertiert, mit einem X-Sharing System verteilt und wieder von einem X-Server unter MacOS 7/8 in Quickdraw konvertiert. Dabei sind zwei Konvertierungsschritte notwendig.

7.2.1.2. Implementierung

Fremdsystemkapselung wurde für eine Umgebung von X-Servern als Endgeräte implementiert. Als Ausgabereplikator dient die sogenannte XWedge [GuPl93] [Gutekunst95]. Da als Endgeräte nur X-Server vorgesehen waren, mußten alle Grafikausgabeströme, die zu den Endgeräten gelangten, dem X-Protokoll entsprechen. Das X-Protokoll ist nicht multicastfähig [AbWa92]. Es wurde deshalb eine verteilte Architektur für den Ausgabereplikator gewählt mit der das X-Protokoll entsprechend adaptiert werden kann. Dadurch entsteht ein Netzwerk von Komponenten des Replikators (siehe Abbildung *WinX/MaX-XWedge a*). Für jeden beteiligte Arbeitsplatzrechner gibt es zwei Komponenten:

1. den sogenannten Pseudo-X-Server und
2. den sogenannten Pseudo-X-Client.

Der Pseudo-X-Server erfüllt in einer reinen X-Umgebung die Funktionen

1. des Ausgabeanalysators an der Netzwerkschnittstelle (Kapitel 4.2.7),
2. des Eingabeinjektors an der Meldungswarteschlange (Kapitel 4.5.1) und
3. des sendenden Teils des Ausgabereplikators (Kapitel 4.3.3.1).

Der Pseudo-X-Client erfüllt die Funktion des empfangenden Teils des Ausgabereplikators.

Anwendungsprogramme, die nicht als X-Clients implementiert sind, sondern auf anderen Grafiksystemen aufsetzen, werden durch das Application Sharing System so gekapselt, daß sie für das übrigen Netzwerk von X-Clients, X-Servern, Pseudo-X-Clients und Pseudo-X-Servern wie X-Clients erscheinen. Damit kann der existierende Verteilungsmechanismus der XWedge auch für nicht-X-Programme verwendet werden. Die XWedge [GuPl95] verwendet Punkt-zu-Punkt Verbindungen über TCP/IP oder Multicast über XTP [Strayer92]. Statt der XWedge können als Ausgabereplikator aber auch andere X-basierte Application Sharing Systeme eingesetzt werden.

Für die Grafiksysteme Win16 (das sogenannte 16-Bit Windows GDI von Windows 3.11) und Quickdraw wurden Ausgabeanalysatoren und Konverter zum X-Protokoll implementiert:

- Win16: WinX von CET, Portugal:
  Ausgabeanalysator auf Treiberebene und ein Eingabeinjektor für die Meldungswarteschlange;
- Quickdraw: MaX von der Universität Ulm:
  Ausgabeanalysator auf Treiberebene (genannt QDTracker), Konverter (MaX) und Eingabeinjektor auf Treiberebene.

WinX und MaX können, mit Einschränkungen, selbst als Application Sharing Systeme betrachtet werden. Sie stellen Anwendungsprogramme für Win16 auf Windows 3.11, bzw. Quickdraw auf MacOS 7/8, auf einem X-Server dar. Der Ausgabereplikator ist dabei monolithisch. Er kann aber nur ein einziges entferntes Endgerät bedienen.

Win16- und Quickdraw-basierte Anwendungsprogramme werden durch WinX/MaX so gekapselt, daß sie dem X-Sharing System XWedge wie X-Clients erscheinen (siehe Abbildung *WinX/MaX-XWedge b*). Das Gesamtsystem stellt die Ausgaben von Programmen, die auf MacOS 7/8 laufen und Quickdraw verwenden, von Programmen unter Windows 3.11 und von beliebigen X-Clients auf mehreren X-Displays dar.

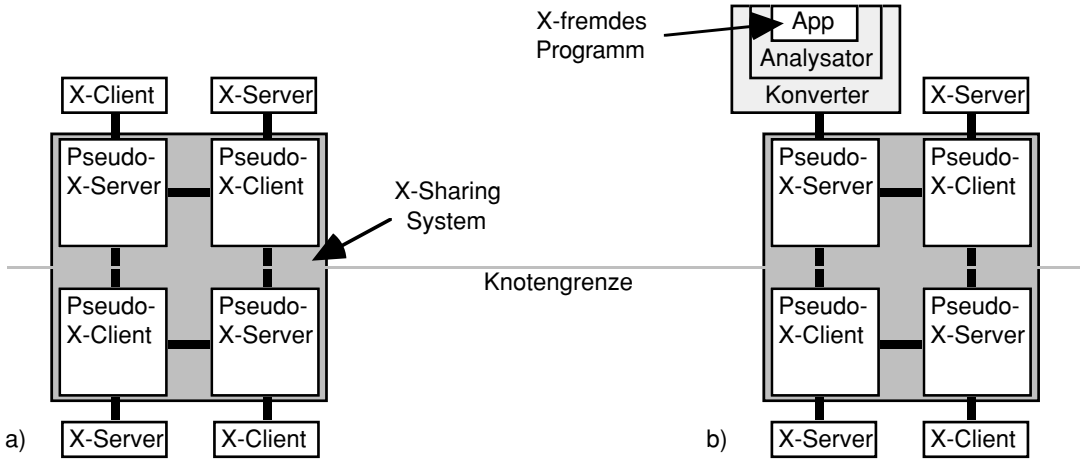


Abbildung WinX/MaX-XWedge

a) Die Architektur eines reinen X-Application Sharing Systems. b) Nicht-X-Programme werden so gekapselt, daß sie für ein X-basiertes Sharing System wie X-Clients erscheinen.

Zusammengefaßt: Fremdsystemkapselung wird vor allem dann gewählt, wenn

- das Sharing System transportorientiert arbeiten soll, d.h. Grafikausgabeströme effizient übertragen werden sollen, aber gleichzeitig
- nur ein bestimmtes Replikatorprotokoll im Netzwerk verwendet werden soll und
- Komponenten des Sharing Systems beim Endgerät installiert werden können.

7.2.2. Übersetzung im Endgerät

7.2.2.1. Allgemein

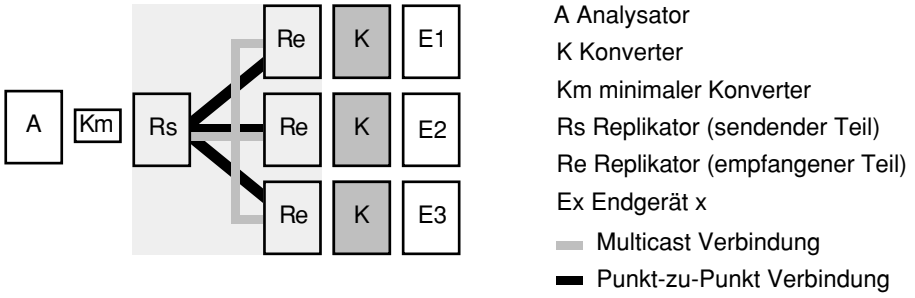


Abbildung Übersetzung im Endgerät

Die in Tabelle *Sharing System Modelle* gezeigte Übersetzung im Endgerät mit mehreren Endgeräten. Der Ausgabereplikator verwendet die Syntax des Quellgrafiksystem. Das Grafikprotokoll für die Netzwerkübertragung wird von einem minimalen Konverter erzeugt (Kapitel 5.3.1). Erst auf dem Zielsystem wird der Ausgabestrom in das Zielgrafiksystem übersetzt.

Ist das Protokoll des verteilten Ausgabereplikators fest vorgegeben, wie bei der oben beschriebenen Implementierung (Fremdsystemkapselung), dann muß der Ausgabestrom eventuell für die Verteilung konvertiert werden. Stimmt die Syntax des Zielgrafiksystems nicht mit der des Replikatorprotokolls überein, dann wird der empfangene Ausgabestrom nochmals für das Endgerät konvertiert. In der Implementierung mit WinX/MaX und XWedge tritt dieser Fall dann ein, wenn die Ausgaben eines Anwendungsprogramms auf einem Endgerät dargestellt werden, das nicht auf dem X-Protokoll basiert. Der Ausgabestrom wird also eventuell zweimal konvertiert.

Im Gegensatz zur Fremdsystemkapselung ist der Konverter nahe beim Endgerät angebracht. Der Ausgabereplikator verteilt den Ausgabestrom in dem Protokoll, das vom Ausgabeanalysator (und damit vom Grafiksystem des Anwendungsprogramms) vorgegeben wird. Der Ausgabestrom wird erst in den Zielsystemen für das Endgerät konvertiert. Ein mit Übersetzung im Endgerät arbeitendes Application Sharing System verwendet im obigen Beispiel (Sharing von Win16 nach Quickdraw) statt des X-Protokolls ein auf Win16 basierendes Netzwerk-Grafikprotokoll (zur Konstruktion eines Netzwerk-Grafikprotokolls aus Prozeduraufrufen siehe Kapitel 5.3.1). Auf dem empfangenden MacOS 7/8 wird der Win16-kodierte Grafikausgabestrom nach Quickdraw konvertiert und ausgegeben. Damit wird, im Vergleich zur Fremdsystemkapselung, ein Konvertierungsschritt eingespart.

Der Vorteil der Übersetzung im Endgerät liegt darin, daß unnötige Konvertierungen vermieden werden, weil kein eventuell von Quell- und Zielgrafiksystem abweichendes drittes Grafiksystem durch das Replikatorprotokoll eingeführt wird. Dabei muß aber in Kauf genommen werden, daß für ein vollständiges Sharing System möglicherweise mehr Konverter implementiert werden müssen, als bei Fremdsystemkapselung. Für jedes Grafiksystem ist ein Konverter erforderlich, der die Syntax des vom Ausgabeanalysator vorgegebenen Grafiksystems in die Syntax des Endgeräts übersetzt, d.h. es sind n(n-1) Konverter erforderlich.

7.2.2.2. Implementierung

Übersetzung im Endgerät wurde im Rahmen einer Diplomarbeit an der Universität Ulm implementiert [Maier97]. Als Netzwerkprotokoll für den Ausgabereplikator wurde MBONE-Multicast mit UDP/IP gewählt. Als Ausgabeanalysator konnte der oben erwähnte QDTracker verwendet werden. Der sendende Teil des Ausgabereplikators wurde für MacOS 7/8 implementiert, der empfangende Teil für SunOS 4.1.x. Der Ausgabereplikator verwendet eine datagrammbasierte Umsetzung der Quickdraw Treiberschnittstelle (Kapitel 5.3.1). Die Datagramme werden mit einfacher Vorwärtsfehlerkorrektur versehen durch zusätzliche Pakete, die binäre XOR-Kombinationen der Datenpakete enthalten. Zusätzliche Fehlersicherungsmaßnahmen auf der Ebene des Grafikprotokolls, wie in Kapitel 4.3.3.3 vorgestellt, werden nicht durchgeführt. Insbesondere ist das Protokoll nicht abschnittsweise selbstkonsistent.

Der Konverter wurde ebenfalls für SunOS 4.1.x implementiert. Er konvertiert Prozeduraufrufe der Quickdraw Treiberschnittstelle nach X11. Der Konverter tritt gegenüber dem X-Server als X-Client auf.

Er kommuniziert über das X-Protokoll mit dem X-Server. Das Gesamtsystem stellt die Ausgaben von Programmen, die auf MacOS 7/8 laufen und Quickdraw verwenden auf mehreren X-Displays dar.

Zusammengefaßt: Übersetzung im Endgerät wird dann gewählt, wenn

- das Sharing System transportorientiert arbeiten soll, d.h. Grafikausgabeströme effizient übertragen werden sollen, aber gleichzeitig
- eine möglichst gute Darstellung auf allen Endgeräten erzielt werden soll,
- Komponenten des Sharing Systems beim Endgerät installiert werden können und
- keine Konvertierung zwischen Grafiksystemen im Quellsystem stattfinden soll.

7.2.3. Homogene netzwerkfähige Endgeräte

7.2.3.1. Allgemein

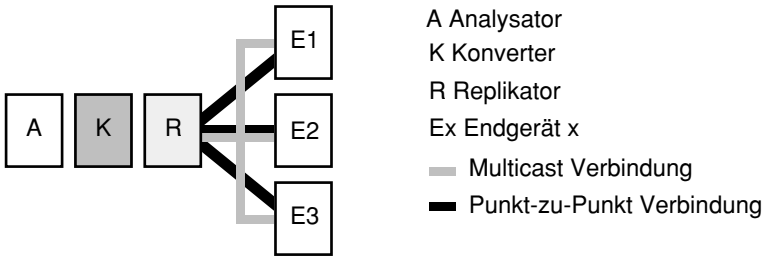


Abbildung Homogene netzwerkfähige Endgeräte

Der Konverter erzeugt einen Ausgabestrom im vom Endgerät akzeptierten Format. Der Ausgabestrom wird dann gleichzeitig an mehrere Endgeräte verschickt. Dabei können Multicast oder Punkt-zu-Punkt Verbindungen verwendet werden.

Bei Szenarien mit netzwerkfähigen Endgeräten wird keine Komponente des Sharing Systems im Endsystem installiert. Das Sharing System läuft im Quellsystem und kommuniziert über die Netzwerkverbindung mit dem Endgerät. Der Ausgabereplikator muß deshalb das vom Endgerät akzeptierte Grafikprotokoll verwenden. Als Endgeräte kommen natürlich vor allem Displayserver von netzwerkfähigen Grafiksystemen (Kapitel 2.2.4), z.B. X-Server, in Frage. Diese Endgeräte können selbständig Grafikströme empfangen und ausgeben. Ist der Endgerätetyp durch äußere Randbedingungen vorgegeben, dann sind dadurch auch das Grafikprotokoll und das Ausgabeformat des Konverters bestimmt.

Der Konverter wandelt den Ausgabestrom des Ausgabeanalysators in das Stromformat genau eines anderen Grafiksystems um. Da dieses Format vom Replikator verteilt wird, kann ein Sharing System prinzipiell nur einen einzigen Typ von Endgeräten unterstützen, nämlich die Endgeräte, die dem Ausgabeformat des Konverters entsprechen.

Ist der Endgerätetyp nicht vorgegeben, dann kann das Zielsystem mit einer netzwerkfähigen Endgerätesoftware ausgerüstet werden, die für das Sharing System geeignet ist. Das Endgerät gehört streng betrachtet nicht zum Sharing System. Es wird aber in diesem Fall mit dem Sharing System ausgeliefert, da es auf das vom Sharing System verwendete Protokoll zugeschnitten ist.

7.2.3.2. Implementierung

Eine populäre Methode, um Endgerätesoftware zu den Zielsystemen zu liefern, ist HTTP in Verbindung mit Java Laufzeitumgebungen in WWW-Klienten. Ganz allgemein bieten WWW-Klienten mit Java Laufzeitumgebungen den Erzeugern dynamischer Daten die Möglichkeit, ohne Vorleistung des Benutzers einen Interpreter für die jeweiligen Daten zu installieren. Im Bezug auf Application Sharing bedeutet dies, daß der Interpreter für den Grafikausgabestrom als Java-Programm zum Rechner des Betrachters übertragen wird. Das Sharing System kann dann sowohl das Grafikprotokoll, als auch das Netzwerkprotokoll bestimmen.

Im einfachsten Fall kann das Grafikprotokoll eine direkte Umsetzung von Prozeduraufrufen sein (Kapitel 5.3.1). Dies bedeutet aber gleichzeitig, daß in der Endgerätesoftware die Funktionalität der Grafikengine des Quellgrafiksystems implementiert sein muß. Ist das Quellgrafiksystem z.B. Win32, dann wird im Endgerät die Win32 Grafikengine benötigt. Momentan bietet nur der Microsoft Internet Explorer 4 für

Win32 die Möglichkeit, aus der Java Laufzeitumgebung, auf eine Win32 Grafikengine zuzugreifen (JDirect). Für andere WWW-Klienten müßte die entsprechende Grafikengine in Java implementiert werden. Dies kann sich aber schnell ändern, da sich momentan sowohl die für Java in WWW-Klienten verfügbaren Softwarebibliotheken (genannt APIs), als auch die Arbeitsweise von WWW-Klienten schnell entwickeln.

Statt der Programmierschnittstelle eines Grafiksystems, wie Win32, können natürlich auch alle anderen, in Kapitel 4.2 erwähnten, Schnittstellen oder deren PDU-Äquivalent als Grafikprotokoll verwendet werden. An der Universität Ulm wurde Application Sharing mit homogenen netzwerkfähigen Endgeräten für Windows NT folgendermaßen implementiert [SchKa97]:

- Der Ausgabeanalysator arbeitet an der Windows NT Treiberschnittstelle.
- Konvertierung im Quellsystem (Kapitel 5.2):
  Der Konverter erzeugt einen Strom von Pixmaps. Die Pixmaps werden in PDUs umgewandelt.
- Der Replikator verwendet TCP/IP Punkt-zu-Punkt Verbindungen.
- Das Endgerät wurde als Java-Applet implementiert. Es interpretiert den vom Konverter erzeugten Strom von Pixmap-PDUs. Das Java-Applet läuft in jeder Java 1.0 [StrMi96] kompatiblen Java Laufzeitumgebung.

Bei der Implementierung wurde die oben, anhand des Win32 Beispiels, beschriebene Problematik der Grafikengine umgangen, indem nur Pixmaps übertragen wurden. Eine Grafikengine für Pixmaps ist in der Java Grafikprogrammierschnittstelle der ersten Generation (AWT [StrMi96]) enthalten. Auch DC-WebShare verwendet die Java Laufzeitumgebung eines WWW-Klienten als Endgerät [WebShare98]. WebShare setzt mit T.128 als Grafikprotokoll auf Win16 auf.

In absehbarer Zeit wird eine weitere Grafikengine für alle Java Laufzeitumgebungen in WWW-Klienten verfügbar sein: die Java2d Grafikprogrammierschnittstelle. Java2d ist eine funktionsreiche und sehr mächtige Grafikprogrammierschnittstelle. Es ist natürlich keinesfalls sicher, daß sich Java2d als Programmierschnittstelle für Java-Programme durchsetzen wird. Vom heutigen Standpunkt ist aber anzunehmen, daß Java2d in WWW-Klienten integriert wird und die Grafikprimitive von AWT ablöst. Damit ergibt sich eine weitere Möglichkeit für das Replikatorprotokoll: ein auf Java2d basierendes Grafikprotokoll. Der Konverter übersetzt das Format des Ausgabeanalysators in ein Grafikprotokoll, das der Syntax von Java2d entspricht, d.h. ein Protokoll dessen PDUs die Parameter von Java2d-Prozeduraufrufen (Methoden) enthalten. Bei der Konvertierung von einem beliebigen Quellgrafiksystem nach Java2d bereitet keine Schwierigkeiten, da Java2d eine große Funktionalität bietet. Dies spiegelt sich darin wieder, daß in Kapitel 5.3.3 keine Problemfälle für postscriptartige Zielgrafiksysteme auftreten.

Zusammengefaßt: Application Sharing mit homogenen netzwerkfähigen Endgeräten wird dann gewählt, wenn

- Standardendgeräte von netzwerkfähigen Grafiksystemen verwendet werden sollen,
- das Sharing System transportorientiert arbeiten soll (z.B. durch Multicast) und die existierenden Endgeräte dies unterstützen oder entsprechende Endgerätesoftware installiert werden kann, oder wenn
- nur eine minimale Endgerätesoftware im Zielsystem installiert werden soll.

7.2.4. Heterogene netzwerkfähige Endgeräte

7.2.4.1. Allgemein

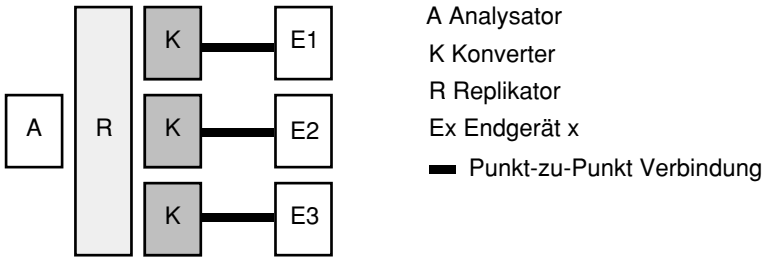


Abbildung Heterogene netzwerkfähige Endgeräte

Der vom Ausgabeanalysator extrahierte Ausgabestrom wird mehreren Konvertern zugeleitet. Jeder Konverter wandelt den Ausgabestrom individuell in das vom Endgerät benötigte Format um. Dabei kann sich jeder Konverter auf die Fähigkeiten des Endgeräts einstellen.

Bei der Konvertierung zwischen verschiedenen Grafiksystemen wird ein Grafikausgabestrom vollständig dekodiert und im Format des Zielgrafiksystems neu kodiert. Die Neukodierung bietet die Möglichkeit, den Ausgabestrom an die Fähigkeiten des Endgeräts anzupassen. Dazu gehören Parameter, wie die Farbtiefe, verfügbare Zeichensätze, Speicherplatz für Ressourcen usw. Ein Sharing System, das auf allen Endgeräten die bestmögliche Darstellung erzielen soll, kodiert den Ausgabestrom für jedes Endgerät individuell. Es besteht aber nicht nur die Notwendigkeit, sich an die Fähigkeiten einzelner Endgeräte anzupassen. Wie bei Application Sharing mit homogenen netzwerkfähigen Endgeräten muß sich das Sharing System auch bei heterogenen Endgeräten an die Schnittstelle der Endgeräte anpassen, d.h. es muß das Grafikprotokoll verwenden, das von den Endgeräten unterstützt wird. Die Konverter erzeugen deshalb für jedes Endgerät einen eigenen Ausgabestrom.

Die Komponenten werden dazu so angeordnet, daß die Konverter direkt mit den Endgeräten verbunden sind. Der Ausgabereplikator ist nicht an der Netzwerkübertragung beteiligt. Er dient nur dazu, den Ausgabestrom vom Ausgabeanalysator an mehrere Konverter zu verteilen. Die Konverter können dabei verschieden sein. Das Sharing System kann unterschiedliche Typen von Konvertern verwenden, um gleichzeitig verschiedene Endgerätetypen zu unterstützen. Je nach Endgerätetyp unterscheiden sich dann die verwendeten Grafik- und Netzwerkprotokolle.

Natürlich müssen nicht unbedingt Konverter für existierende netzwerkfähige Displayserver eingesetzt werden. Wie bei Application Sharing mit homogenen netzwerkfähigen Endgeräten beschrieben, kann auch hier eine spezielle Endgerätesoftware im Zielsystem installieren werden. Die Endgerätesoftware unterliegt dann den gleichen Randbedingungen bezüglich der Verfügbarkeit einer Grafikengine, wie oben beschrieben. Tatsächlich kann sogar die gleiche Endgerätesoftware verwendet werden.

7.2.4.2. Implementierung

Application Sharing mit heterogenen netzwerkfähigen Endgeräten wurde an der Universität Ulm für Application Sharing von MacOS 7/8 nach X11 implementiert [WoFrSchu95]. Das Sharing System heißt M-MaX. Es umfaßt

- den Ausgabeanalysator "QDTracker" an der Quickdraw Treiberschnittstelle,
- den Ausgabereplikator "Mux",
- einen Ausgabekonverter von Kommandos der Quickdraw Treiberschnittstelle zu X11,
- einen Eingabekonverter von X11-Events zu MacOS 7/8 Eingaben,
- einen Eingabekonzentrator mit Eingaberechtsvergabe und
- einen in den QDTracker integrierten Eingabeinjektor;
- zusätzlich ein Telepointer-Modul, das auf der Grafikschnittstelle des Ausgabereplikators aufsetzt, wie in Kapitel 6.2.3 beschrieben.

Die Software MaX aus der Implementierung von der Fremdsystemkapselung wurde so weiterentwickelt, daß sie gleichzeitig gegenüber mehreren X-Servern als X-Client auftreten kann. Jeder Konverter unterhält eine X11/TCP/IP Verbindung zu einem entfernten X-Server. Ein Ausgabestrom wird so auf mehreren X-Servern gleichzeitig dargestellt.

Es wurde ein Ausgabereplikator entwickelt, der die Verteilung der vom QDTracker extrahierten Ausgabeströme and die Konverter übernimmt. Der Ausgabereplikator verarbeitet gleichzeitig die Ausgabeströme mehrere Anwendungsprogramme. Er verwaltet dabei die Umschaltung zwischen Anwendungsprogrammen, um jeweils die korrekte Menüleiste darzustellen, bzw. die Menüleiste auszublenden, falls sie zu keinem der gemeinsamen Programme gehört. Die Kommunikation zwischen Analysator, Replikator und Konverter besteht aus Prozeduraufrufen. Replikator und Konverter sind dafür in einem Programm integriert.

Die Konverter kodieren einen Ausgabestrom für jeden X-Server. Die Farbdarstellung wird für jedes Display individuell angepaßt. Da Grafikausgaben in Quickdraw von Anwendungsprogrammen meistens in Echtfarbdarstellung durchgeführt werden, ergibt sich auf entfernten Echtfarbschirmen eine sehr gute Bildqualität. Gleichzeitig werden die Farben für Bildschirme mit indizierten Farben reduziert. Dabei wird die ein Kapitel 5.2.2 vorgestellte Methode zur Konvertierung von Pixmaps verwendet. Ist der Bildschirm des Quellsystems nicht mit einem Echtfarbdisplay ausgestattet, dann kann die Bildqualität auf einem entfernten Bildschirm sogar besser sein, als auf dem lokalen Bildschirm des Systems, wo das Anwendungsprogramm läuft.

Zusammengefaßt: Application Sharing mit heterogenen netzwerkfähigen Endgeräten wird dann gewählt, wenn

- Standardendgeräte von netzwerkfähigen Grafiksystemen verwendet werden sollen,
- eine möglichst gute Darstellung auf allen Endgeräten erzielt werden soll und
- keine Konvertierung im Zielsystem stattfinden soll.

8. Zusammenfassung

Diese Kapitel faßt die Ergebnisse der Arbeit zusammen. Das Kapitel ist unterteilt in drei Abschnitte für

1. die Ergebnisse in Ausgaberichtung,
2. die Ergebnisse im Zusammenhang mit Benutzereingaben und
3. Ergebnisse, die die Entwicklung von Grafiksystemen betreffen.

8.1. Ergebnisse auf der Ausgabeseite

Die vorliegende Arbeit gab zunächst einen Überblick über die Historie des Application Sharings bis zum heutigen Stand der Technik. Sie vergleicht anschließend Grafikausgabeströme von Anwendungsprogrammen mit einem schon bekannten Prinzip, nämlich den Sequenzen von Grafikkommandos, die aus anderen Anwendungen, wie Grafikdateien und Filmen bekannt sind. Grafikströme von Programmen im Rahmen des Application Sharings sind eine Verallgemeinerung dieser klassischen Grafiksequenzen. Durch eine Betrachtung der Granularität der Grafikausgabe von Grafiksequenzen wird der wesentliche Unterschied zwischen Grafikströmen und klassischen Grafiksequenzen identifiziert. Dieser geht schließlich ein in die Definition der drei notwendigen Eigenschaften von netzwerkfähigen Grafikendgeräten, d.h. von Grafikendgeräten, die für Application Sharing eingesetzt werden können.

Application Sharing Systeme stellen die Ausgaben eines Anwendungsprogramms auf mehreren Bildschirmen dar. Gleichzeitig werden Eingaben von allen berechtigten Teilnehmern kombiniert und dem Programm zugeführt. Sind bei einer Sitzung verschiedene Typen von Betriebssystemen und Grafiksystemen beteiligt, dann übersetzt ein Application Sharing System den Grafikausgabestrom des Programms in die Schnittstelle der empfangenden Grafikendgeräte.

Die Behandlung von Grafikausgabeströmen nahm in dieser Arbeit breiten Raum ein. In Ausgaberichtung bestehen Application Sharing Systeme aus drei Kernkomponenten: Ausgabeanalysator, Ausgabereplikator und Ausgabeübersetzer. Für diese drei Komponenten wurden Softwaretechniken entwickelt, die sowohl bei der Entwicklung von zukünftigen Application Sharing, als auch bei Grafiksystemen als Leitlinie dienen können. Fast alle der in dieser Arbeit vorgestellten Techniken wurden durch Implementierungen getestet. Im folgenden werden die wichtigsten bearbeiteten Punkte zusammengefaßt:

1. Ausgabeanalysator
   - Schnittstellenbewertung
     Kapitel 4.2 enthält die erste vollständige Klassifizierung von Schnittstellen der wichtigsten Ausgabesysteme im Bezug auf die Semantik der Grafikdaten an der Schnittstelle, Zugriffsverfahren und Spezialfälle, die bei Implementierungen auftreten.
   - Analysetechniken auf verschiedenen Ebenen
     Es wurden detailliert Ausgabeanalysatoren an verschiedenen Schnittstellen von Grafiksystemen beschrieben. Anhand dieser Ergebnisse können reale Ausgabeanalysatoren implementiert werden.
   - Ausgabeanalyse auf Hardwareebene
     Es wurde eine Technik zur hardwareunterstützten Ausgabeanalyse vorgestellt, die mit Standardprozessoren realisiert werden kann.
   - Beschränkung auf Grafikausgaben
     Eine wichtige Erkenntnis ist die Beschränkung des Ausgabeanalysators auf Grafikausgaben. Der Ausgabestrom darf keine aktiven Elemente enthalten. Benutzereingaben müssen auf der Ebene elementarer Eingaben and das Anwendungsprogramm übermittelt werden. Es werden keine selbständigen Grafikausgaben aktiver Elemente im Zielsystem zugelassen, da sonst Elemente der Benutzerschnittstelle in Quell- und Zielsystem verschiedene Zustände annehmen können.
   - Selbständige Ausgaben des Sharing Systems
     Alle selbständigen Ausgaben eines Application Sharing Systems, die nicht auf Ausgaben von Anwendungsprogrammen zurückgehen, müssen die Grafikschnittstelle des Ausgabeanalysators verwenden. Dazu gehören Darstellungen der Bedienungsoberfläche, Konfigurationsmöglichkeiten und sogar Telepointer. Diese Ausgaben erscheinen dann automatisch am Ausgang des

Ausgabeanalysators, so daß selbständige Ausgaben ohne zusätzlichen Aufwand die Dienste der übrigen Komponenten des Application Sharing Systems nutzen können.

2. Ausgabereplikator
   - Verfahren zur Verteilung von Grafikausgabeströmen.
     Kapitel 4.3 beschrieb notwendige Eigenschaften von Ausgabereplikatoren unter verschiedenen Netzwerkbedingungen. Dabei wurden Lösungen für mehrere für Application Sharing Systeme typische Probleme eingeführt.
   - Verfahren zur Reaktion auf Programmanfragen
     In der Literatur wurde dieser Komplex bisher nur für das jeweils vorgestellt System diskutiert. Dies geschieht meistens sogar unter der Annahme, daß es keine Alternativen zur dort vorgestellten Lösung gibt. In dieser Arbeit wurde gezeigt, daß es drei verschiedene Verfahrensweisen für die Reaktion des Ausgabereplikators auf Programmanfragen gibt.
   - Optimierung der internen Schnittstelle
     Erstmals wurde eine Analyse von Ausgabeanalysatoren unabhängig von einer speziellen Implementierung durchgeführt. Das Ergebnis ist die Identifizierung der internen Schnittstelle als Ansatzpunkt von Optimierungen. Die Methode von Danskin zur X11-Beschleunigung wurde zur Anwendung auf die interne Schnittstelle des Ausgabereplikators verallgemeinert.
   - Forderungen an Implementierungen verteilter Ausgabereplikatoren
     Die Analyse existierender Application Sharing Systeme und Erfahrungen bei der Implementierung führten zu folgenden zusätzlichen Forderungen an zukünftige Implementierungen:
     - Toleranz gegenüber netzwerkbedingter Unterbrechung einzelner Verbindungen durch die Einführung eines Session-Layers.
     - Sendergewählte Ressource-IDs ohne Einfluß der Empfänger.
     - Abschnittsweise Vollständigkeit des Grafikausgabestroms durch die Einführung verallgemeinerter Stützbilder.
     - Weiche Kopplung zwischen Anwendungsprogramm und Netzwerk durch globale Bandbreitenadaptierung.
   - Individuelle Bandbreitenadaptierung
     In der Literatur wurde der Fall verschiedener Bandbreite innerhalb einer Sitzung bisher nicht behandelt. Dieses Problem wurde als nur durch globale Bandbreitenadaptierung lösbar angesehen. In dieser Arbeit wurde dagegen eine Methode zur individuellen Anpassung an die für einzelne Netzwerkverbindungen verfügbare Bandbreite eingeführt: das CE/SC Verfahren.

3. Ausgabeübersetzer
   - Verfahren zur Übersetzung zwischen Grafikschnittstellen
     Kapitel 5 beschrieb Techniken zur Übersetzung von Grafikausgabeströmen unter verschiedenen Bedingungen. Die Arbeit lieferte konkrete Anweisungen für bisher als schwer oder unlösbar betrachtete Übersetzungsschritte.
   - Schnelle Pixmap-Übersetzung
     Da Pixmaps einen wesentlichen Teil der von Anwendungsprogrammen erzeugten Daten darstellen, wurde eine Methode zur schnellen Konvertierung von Pixmaps unter Zuhilfenahme des Quellgrafiksystems entwickelt.
   - Methoden zur Adaptierung abweichender Fensterverwaltungsstrategie
     Das Problem abweichender Fensterverwaltungsstrategien trat bisher nicht auf, da außer den im Zusammenhang mit dieser Arbeit entwickelten Implementierungen keine heterogenen fensterbasierten Application Sharing Systeme existierten. Es wurde eine Technik zur Übersetzung von passiver Fensterverwaltung in aktive Fensterverwaltung entwickelt.
   - Ausgabebereiche
     Eine wichtige Technik zur Verwaltung von Ausgabebereichen besteht darin, nur die Sichtbarkeit, nicht die Existenz eines Fensters als Signal zur Erzeugung eines Fensters im Zielfenstersystem heranzuziehen. Das Signal wird aus dem Darstellungskommando im Quellfenstersystem, nicht aus dem Erzeugungskommando abgeleitet werden.

8.2. Ergebnisse auf der Eingabeseite

In Eingaberichtung wirken die Komponenten: Eingabekonzentrator, Eingabeinjektor und Eingabeübersetzer. Es wurden detaillierte Hinweise für die Implementierung von Eingabeinjektor und Eingabeübersetzer für die wichtigsten Plattformen gegeben. Die Behandlung der Eingabeseite ist aber insgesamt wesentlich weniger aufwendig, als die der Ausgabeseite.

Diese Arbeit beschäftigte sich erstmals mit der in der Literatur bisher vernachlässigten Thematik der Integritätssicherung bei der Kombination mehrerer Eingabeströme. Dazu wurde die aus der taskorientierten Interaktionsmodellierung bekannte GOMS-Methode auf die Modellierung von Eingabeströmen im Eingabekonzentrator von Application Sharing Systemen übertragen. Dadurch wurde es erstmals möglich, die Konsistenz des Eingabestroms für Anwendungsprogramme zu garantieren. Bisher wurde entweder die Umschaltung zwischen Eingabeströmen verschiedener Teilnehmer so vorgenommen, daß keine Inkonsistenzen auftreten konnten, d.h. es wurden nur eine grobe Granularität der Eingabeumschaltung zugelassen. Oder: es wurden Inkonsistenzen toleriert und deren Auswirkungen, die zum Programmabsturz führten können, durch ad-hoc Maßnahmen des Sharing Systems oder fehlertolerante Anwendungsprogramme korrigiert. Die Vermeidung von Programm- oder Systemabstürzen wurde bisher als Fehlerbehebung bei Bedarf angesehen und nicht systematisch betrieben.

8.3. Ergebnisse für Grafiksysteme

Als Grundlage für Ausgabeanalyse und Ausgabekonvertierung wurden die Mechanismen von Grafiksystemen untersucht und mit den Bedingungen beim Einsatz von Application Sharing Systemen verglichen. Daraus wurden Hinweise für die Entwicklung zukünftiger Grafiksystemen abgeleitet. Die Beachtung dieser Richtlinien begünstigt die Entwicklung von Application Sharing Systemen für das jeweilige Grafiksystem ohne Entwurfsmuster für Grafiksysteme zu verletzen.

1. Abfragbarkeit des Grafikkontextes
   Kann der Zustand des Grafikkontextes von Anwendungsprogrammen, bzw. vom Ausgabeanalysator ermittelt werden, dann entfällt die Notwendigkeit zur Aufzeichnung von Kontextmanipulationskommandos. Gleichzeitig begünstigt dies eine ausgabeorientierte Konvertierung im Gegensatz zur häufig angewendeten kommandoorientierten Konvertierung.
2. GUI-Toolbox auf API
   Verwendet die GUI-Toolbox die Programmierschnittstelle des Grafiksystems, dann gelangen Ausgaben von aktiven Elementen der Benutzerschnittstelle ohne zusätzlichen Aufwand in den Ausgabestrom. Umgeht dagegen die GUI-Toolbox die Programmierschnittstelle, dann ist der Einsatz eines erweiterten Ausgabeanalysators notwendig, der auch die GUI-Toolbox umfaßt. Gleichzeitig sind in diesem Fall oft zusätzliche Eingriffe in das Betriebssystem, bzw. in das Grafiksystem notwendig, die eine Entwicklung von Ausgabeanalysatoren für das betreffende Grafiksystem erschweren.
3. Fensterzuordnung
   Application Sharing Systeme benötigen eine eindeutige Zuordnung zwischen Fenster und Grafikkontext. Mit einem Grafikkontext muß zu jeder Zeit das zugehörige Fenster ermittelt werden können. Sind die Datenstrukturen Fenster und Grafikkontext miteinander vermischt, dann darf es entweder nur einen möglichen Grafikkontext je Fenster geben oder es muß ein expliziter Zuordnungsmechanismus eines Fensters zum Grafikkontext existieren (z.B. durch Referenzen im Grafikkontext oder als Systemfunktion).
4. Ressource-Installation
   Auf Ressourcen die dem Grafikendgerät des Quellsystems zur Verfügung stehen, können andere Endgeräte nicht zugreifen. Dies gilt vor allem für Zeichensätze, aber auch für andere Ressourcetypen, wie Pfade und Bereiche. Ein netzwerkfähiges Grafiksystem soll die Möglichkeit zur dynamischen Installation von Ressourcen zur Laufzeit bieten.
5. Verallgemeinerte Unterfenster
   Einführung einer Klasse von Unterfenstern, die trotz relativer Positionierung nicht durch die Sichtbarkeit des Vaterfensters beschränkt sind. Alle Eigenschaften von Unterfenstern müssen unabhängig voneinander wählbar sein. Die Vater-Kind-Beziehung zwischen Fenstern wird damit

verallgemeinert auf eine Verwandtschaftsbeziehung in der die Parameter Relativpositionierung, Sichtbarkeit und Eingabeverarbeitung getrennt gewählt werden können.

In der vorliegenden Arbeit wurden wesentliche Aspekte von Application Sharing Systemen erstmals systematisch erfaßt. Es wurden Entwurfsstrukturen und Techniken vorgestellt, die bei Anwendung zu besseren, d.h. schnelleren und benutzerfreundlicheren Application Sharing Systemen führen. Dies gilt vor allem für den Bereich der heterogenen Systeme, die bisher, bis auf Ausnahmen, nur auf Pixelbasis arbeiten.

Die hier entwickelten Strukturen, Empfehlungen und praxisnahen Lösungen können dazu beitragen, daß zukünftige fensterbasierte Application Sharing Systeme den Anforderungen der Benutzer im Hinblick auf Einsatzmöglichkeiten und Benutzerfreundlichkeit wesentlich weiter entgegen kommen, als die existierenden Systeme. Sie decken alle wichtigen Bereiche ab und bilden damit einen umfassenden Leitfaden für die Entwicklung von Application Sharing Systemen.

9. Bibliographie

[AbKiKFa99] H. Abdel-Wahab, O. Kim, P. Kabore, J.P. Favreau: Java-based Multimedia Collaboration and Application Sharing Environment; angenommen für Colloque Francophone sur l'Ingenierie des Protocoles, (CFIP'99), Nancy, France, April 1999

[AbKvNa96] H. Abdel-Wahab, B. Kvande, S. Nanjangud: Using Java for Multimedia Collaborative Applications, Proceedings of PROMS'96: Third International Workshop On Protocols for Multimedia Systems, 1996, pp. 49-62, Madrid, October 1996.

[AbWa91] H. Abdel-Wahab, M. A. Feit: XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration; Chapel Hill, 1992.

[AbWa92] H. Abdel-Wahab, K. Jeffay: Issues, Problems and Solutions in Sharing X Clients on Multiple Displays; Technical Report TR92-043, University of North Carolina, Chapel Hill, 1992.

[Altenhofen90] M. Altenhofen: Erweiterung eines Fenstersystems für Tutoring-Funktionen; Diplomarbeit, Universität Karlsruhe, 1990.

[ArGoRo89] M.J. Arden, J. Gosling D.S.H. Rosenthal: The NeWs book: an introduction to the Networked extensible Window System; Springer, 1989, ISBN 0387969152.

[ArJoRo95] K.O. Arisland, S.A. Johansen, G. Ronning: Introducing Candleweb and Å (awe) - Bringing Animation Power to the World Wide Web; World Wide Web Journal, Proceedings of the Fourth International World Wide Web Conference, O'Reilley, 1995, ISBN 1-56592-169-0.

[BaPl93] J. Baldeschwieler, T. Gutekunst, B. Plattner: A Survey of X Protocol Multiplexors; in: Computer Communication Review, Vol. 23, Nr. 2, ACM Press, 1993.

[BeCa94] Berners-Lee, T., Cailliau, et al.: The World-Wide Web; Communications of the ACM, Vol. 37, No. 8, August 1994, S. 76 - 82.

[BeClWi90] Bell, B.C., Cleary, J.G., Witten, I.H.: Text Compression; Engelwood Cliffs, 1990, ISBN 0-13-911911-4.

[BeSmDe96] D.V. Beard, D.K. Smith, K.M. Denelsbeck: QGOMS: A direct-manipulation tool for simple GOMS models; Proceedings CHI 96, Vancouver, 1996.

[BiWaBi93] K.W. Bibb, L. Wake, K.D. Bibb: Practical Xview Programming; John Wiley & Sons, ISBN 0471574600.

[CaMoNe80] S. Card, T. Moran, A. Newell: The keystroke-level model for user performance time with interactive systems. CACM 23(7) 396-410, 1980.

[CaMoNe83] S. Card, T. Moran, A. Newell: The psychology of human-computer interaction, Hillsdale, NJ: Lawrence Erlbaum Associates, 1983.

[ChBaVi92] M. Chen, T. Barzilay, and H. Vin. Software Architecture of DiCE: A Distributed Collaboration Environment. In Proceedings of the Fourth IEEE COMSOC International Workshop Multimedia '92, Monterey, CA, April 1992.

[Chung91] G. Chung: Accommodating Latecomers in a System for Synchronous Collaboration, Master Thesis, University of North Carolina, Chapel Hill, 1992.

[Citrix] http://www.citrix.com/

[ClTe90] C. Clark, C. Tennenhouse: Architectural Considerations for a New generation of Protocols; Proceedings ACM SIGCOMM 1990.

[Cohn89] T. Cohn: Assembly Lab - Convert PICTs to Regions; MacTutor Magazin, Juni 1989.

[CrTo90] T. Crowley, P. Milazzo, E. Baker, H. Forsdick, R. Tomlinson: MMConf: An Infrastructure for Building Shared Multimedia Applications; Proceedings CSCW'90, Los Angeles, 1990.

[Danskin95] J.M. Danskin: Compressing the X Graphics Protocol; Dissertation Princeton University, 1995.

[DaZh96] J.M. Danskin, Q. Zhang: Bitmap Reconstruction for Document Image Compression; SPIE International Symposium on Voice, Video, and Data Communications, 1996.

[DCShare97] http://www.Datcon.co.uk/

[Deering89] Deering, S.: RFC 1112: Host extensions for IP multicasting; August 1989. http://ds.internic.net/rfc/rfc1112.txt

[ElGiRe90] C.A.Ellis, S.J. Gibbs, G.L. Rein: Desgin and use of a group editor; Engineering for Human-Computer Interaction, Elsevier Science Publishers B. V., 1990.

[FDFH96] Compter Graphics: Principles and Practice; J. D. Foley, A. van Dam, S. J. Feiner, J. F. Hughes, Addison-Wesley, 1996, ISBN 0-201-84840-6.

[Florence94] M. Florence: Digital Video File Formats; Dr. Dobbs Sourcebook of Multimedia Programming, 1994.

[FoDaFeHu91] Foley, van Dam, Feiner, Hughes: Computer graphics: Principles and Practice; Addison-Wesley, ISBN 0-201-12110-7.

[Froitzheim97] K. Froitzheim: Multimedia Kommunikation; dpunkt Verlag, Heidelberg, 1997, ISBN 3-920993-61-6.

[GeWe98] W. Geyer, R. Weis: A Secure, Accountable, and Collaborative Whiteboard; 5th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services, Oslo, Norway, September 1998, ISBN 3-540-64955-7.

[Greif88] I. Greif, Ed.: Computer Supported Cooperative Work: A Book of Readings; Morgan Kaufmann, San Mateo, California, 1988.

[GuPl93] T. Gutekunst, B. Plattner: Sharing Multimedia Applications Among Heterogeneous Workstations; Proceedings, Second International Conference on Broadband Islands. Edited by O. Spaniol, F. Williams. Athens, 1993.

[GuPl95] T. Gutekunst, D. Bauer, G.Caronni, Hasan, B. Plattner: A distributed and Policy-Free General Purpose Shared Window System; IEEE/ACM Transactions on Networking, Hannes P. Lubich (Ed.), Februar 1995.

[Gutekunst95] T. Gutekunst: Shared Window Systems; Diss. ETH No. 11120, Zürich, 1995.

[GuWe93] T. Gutekunst, T. Schmidt, G. Schulze, J. Schweitzer, M. Weber: A Distributed Multimedia Joint Viewing and Tele-Operation Service for Heterogeneous Workstation Environments; Verteilte Multimedia Systeme, Feb '93, Stuttgart, ISBN 3-598-22407-9.

[Hart96] Hart, J.C.: Fractal Image Compression and Recurrent Iterated Function Systems; IEEE Computer Graphics and Applications; Band 16, Nr. 4, Juli 1996; S. 26 – 33.

[Holtzgang90] Display Postscript Programming; D.A. Holzgang, Addison-Wesley, 1990, ISBN 0-201-51814-7.

[ICA] Citrix, Inc.: Citrix ICA Technology Brief; http://www.citrix.com/Products/icatech.htm.

[IM-I88] Inside Macintosh Volume I; Apple Computer, Addison-Wesley, 1988, ISBN 0-201-17731-5.

[IM-V88] Inside Macintosh Volume V; Apple Computer, Addison-Wesley, 1988, ISBN 0-201-17719-6.

[ISO3309/4335] Data Communication - High Level Data Link Control Procedures; International Standard, ISO 3309/4335, Genf 1979.

[Jacobson97] Floyd, S., Jacobson, V., Liu, C., McCanne, S., and Zhang, L.: A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing; IEEE/ACM Transactions on Networking, December 1997, Volume 5, Number 6, pp. 784-803.

[Java2D] Java 2D API: Enhanced Graphics and Imaging for Java; Sun Microsystems, Inc., 1997

[JoKi94] B.E. John D.E. Kieras: The GOMS Family of Analysis Techniques: Tools for Design and Evaluation;

[JPEG91] International Organization for Standardization: Information Technology - Digital Compression and Coding of Continous-tone Still Images; ISO/IEC DIS 10918-1; ISO 1991.

[Kaufmann99] L. Kaufmann: CESC Component Encoding Stream Construction; Diplomarbeit, Universität Ulm, 1999.

[Kieras88] D. Kieras: Towards a practical GOMS model methodology for user interface design. In M. Helander (ed.): Handbook of human-computer interaction, North-Holland, Amsterdam, 1988.

[KiKFaAb97] O. Kim, P. Kabore, J.P. Favreau, H. Abdel-Wahab: Issues in Platform-Independent Support for Multimedia Desktop Conferencing and Application Sharing, Proceedings of the Seventh IFIP Conference on High Performance Networking (HPN'97), White Plains, NY, pp. 115-139, April 1997.

[KiWoMe97] D. Kieras, S. Wood, D. Meyer: Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task. Trans. Comput.-Human Interact. 4(3), 1997, 230-275.

[Konstanjevec97] D. Konstanjevec: Verteilung von Grafikobjekten om Multicastszenarien; Diplomarbeit, Universität Ulm, 1997.

[LaBöEf94] Lamparter, B., Böhrer, O., Effelsberg, W.: Vorausschauende Fehlerkorrektur für multimediale Datenströme; in: Effelsberg, W., Rothermel, K. (Hrsg): Verteilte Multimedia-Systeme; München, 1993.

[LaEf94] Lamparter, B., Effelsberg, W.: eXtended Color Cell Compression - A Runtime-efficient Compression Scheme for Software Video; in: Steinmetz, R. (Ed.): Multimedia: Advanced Teleservices and High-Speed Communication Architectures; Proceedings of the 2nd International Workshop, IWACA '94, Heidelberg, 1994.

[LeFiKr86] M.D.P. Leland, R.S. Fish, R.E. Kraut: Collaborative document production using Quilt; Proceedings of the First Conference on Computer Supported Collaborative Work, Austin, Texas, ACM, New Yprk, 1986.

[LeHo98] B.T. Lewis, J.D. Hodges: Shared Books: Collaborative publication management system for an office information system; Proceedings of the Conference on Office Information Systems, Palo Alto, California, ACM, New York, 1998.

[LiveShare98] http://www.PictureTel.com/

[Maier97] M. Maier: Fensterbasiertes Applikationssharing im MBONE; Diplomarbeit, Universität Ulm, 1997.

[McFarlane91] G. McFarlane: Xmux - A System for Computer Supported Collaborative Work; Proceedings 1st Australian Multi-Media Communications, Applications & Technology Workshop, Sydney, 1991.

[Minenko96] W. Minenko: Advanced Design of Efficient Application Sharing Systems under X Window; Dissertation Universität Ulm, Januar 1996.

[MPEG1-92] Information Technology - Coding of moving pictures and associated audio for digital storage media up to 1,5 Bbit/s; ISO Draft International Standard, ISO/IEC DIS 11172, 1992.

[MuVa94] Graphics File Formets; J. Murray, W. Vanryper, O'Reilley & Associates, 1994, ISBN 1-56592-058-9.

[NCD] http://www.ncd.com/

[NeHa88] A. Netravali, B. Haskell: Digital Pictures, Representation and Compression; Plenum Press, 1988.

[Net120-97] http://www.DataBeam.com/net120/

[Netmeeting96] http://www.Microsoft.com/netmeeting/

[NoBiTo97] J. Nonnenmacher, E. Biersack, D. Towsley: Parity-Based Loss Recovery for Reliable Multicast Transmission; Proceedings of ACM SIGCOMM'97, ACM Press, 1997, ISBN 0146-4833.

[NTDDK4.0] Windows NT DDK, Microsoft.

[OSI89] ISO/IEC: 7498: Information Processing Systems – Open Systems Interconnection – Basic Reference Model; Genf, 1989.

[PaTi90] F. De Paoli, F. Tisato: A Model or Real-Time Co-operation; 2nd European Conference on Computer-Supported Cooperative Work, 1991.

[Petzold96] Programming Windows 95; C. Petzold, Microsoft Press, 1996, ISBN 1-55651-676-6.

[Pieres93] N. Pires: Floor Control; EU RACE 2060 CIO WP 4.2 Internal report, Version 2, Revision 2, 1993.

[Postel80] J. Postel: User Datagram Protocol; IETF STD 0006, 1980.

[Postel81] J. Postel: Transmission Control Protocol; IETF STD 0007, 1981.

[Postscript] POSTSCRIPT Language Reference Manual; Adobe Systems Incorporated, Addison-Wesley, 1987, ISBN 0-201-10174-2.

[Quickdraw94] Imaging With QuickDraw; Apple Computer, Addison-Wesley, 1994, ISBN 0-201-63242-X.

[Quicktime93] QuickTime; Apple Computer, Addison-Wesley, 1993, ISBN 0-201-62201-7.

[RaJo91] M. Rabbani; P.W. Jones: Digital Image Compression Techniques; Bellingham, 1991.

[ReSchVöWe94] W Reinhard, J. Schweitzer, G. Völksen M. Weber: CSCW Tools: Concepts and Architectures; IEEE Computer, May 1994.

[RFC1459] J, Oikarinen, D.Reed: Internet Relay Chat Protocol; IETF RFC 1459, 1993.

[RFC2357-98] A. Mankin, A. Romanow, S. Bradner, V. Paxson: IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols; IETF Informational RFC 2357, Juni 1998.

[RMRG97] http://www.east.isi.edu/rm/

[RSA] CCITT, Recommendation X.509: The Directory: Authentication Framework; Annex C.

[SchKa64] E.S. Schwartz and B. Kallick: "Generating a canonical prefix encoding." Comm. ACM, 7,3 (Mar. 1964), pp. 166-169.

[SchKa97] M. Schöttner, A. Kassler: Private Mitteilung; Abteilung Verteilte Systeme Universität Ulm, 1997.

[Schöttner96] M.Schöttner: Application-Sharing mit MS-Windows; Diplomarbeit, Universität Ulm, 1996.

[Schöttner97a] M. Schöttner: Private Mitteilung; Abteilung Verteilte Systeme Universität Ulm, 1997.

[Schöttner98] M. Schöttner: Private Mitteilung; Abteilung Verteilte Systeme Universität Ulm, 1997.

[SGImeeting98] http://www.SGI.com/software/sgimeeting/

[ShGe90] Sheifler, R., Gettys, J.: The X-Window System; Bedford, 1990.

[SPM95] Osborne Windows Programming Series, Vol. 2; H. Schildt, Ch. Pappas, W.H. Murray, Osborne McGraw-Hill, 1995, ISBN 0-07-881991-1.

[Strayer92] W.T. Strayer, B.J. Dempsey, A.C. Weaver: XTP: The Xpress Transfer Protocol; Addison Wesley, 1992, ISBN 0-201-56351-7.

[StrMi96] Strobel, S., Middendorf, S.: Java – Programmierhandbuch und Referenz; Heidelberg, 1996.

[Sun96] Sun Microsystems, Inc.: ShowMe 2.0 User Guide; 1994.

[SunForum99] http://www.sun.com/desktop/products/software/sunforum/docs/HTML/overview.htm

[T.128-98] Multipoint Application Sharing; ITU-T Recommendation T.128, 1998.

[Talbott92] S. Talbott: PEXlib Reference Manual - 3D Programming in X; O'Reilly, 1992.

[Timbuktu98] http://www.TimbuktuPro.com/

[Vodslon92] M. Vodslon: Feasibility of Translating native Windowing Systems to X-Window; RACE-CIO, Internal Report, 1992.

[Wallace91] Wallace, G.K.: The JPEG Still Picture Compression Standard; in: Communications of the ACM, April 1991, Vol. 34, No. 4, S. 30 ff.

[WebShare98] http://www.Datcon.co.uk/

[Welch84] Welch, T. A.: A Technique for High-Performance Data Compression, in: Computer, Volume 17, Number 6, June 1984, S.8 – S.19.

[WoFr97] K.H. Wolf, K. Froitzheim: WebVideo - a Tool for WWW-based Teleoperation; Proceedings IEEE ISIE'97, Guimaraes, July 1997.

[WoFrSchu95] K.H. Wolf, K. Froitzheim, P. Schulthess: Multimedia Application Sharing in a Heterogenous Environment; Proceedings ACM Multimedia, 1995, ISBN 0-201-87774-0.

[WoFrWe96] K.H. Wolf, K. Froitzheim, M. Weber: Interactive Video and Remote Control via the World Wide Web; Lecture Notes in Computer Science 1045: Interactive Distributed Multimedia Systems and Services, March 1996, Berlin.

[Xv0-90] X Protocol Reference Manual - Volume Zero; O'Reilly & Associates, 1990, ISBN 0-937375-50-1.

[Xv2-90] Xlib Reference Manual - Volume Two; O'Reilly & Associates, 1990, ISBN 0-937175-12-9.

[Zhang93] Zhang, L., Deering, S., Estrin, D., Shenker, S., Zappala, D.: RSVP: A New Ressource Reservation Protocol; IEEE Network, September 1993.

[ZiLe77] J. Ziv , A. Lempel, "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343., 1977

[Zimmerman92] P. Zimmerman. PGP User's Guide, Volume 1: Essential Topics. Distributed by MIT, 1992.

10. Anhang

10.1. Apple Macintosh Quickdraw Analysator

Macintosh Quickdraw API

Das Quickdraw API umfaßt 47 Prozeduren [Quickdraw94]. Darunter sind jeweils mehrere Varianten für das gleiche Grafikprimitiv.

Macintosh Toolbox API

MenuSelect
MDEFHook
PopUpMenuSelect
GetMenu
CalcMenuSize
PlotCIcon
HiliteMenu
InitHookJump
CopyMask
CopyBits
BeginUpdate
EndUpdate
DrawMenubar;
DrawGrowIcon

Macintosh Window Manager API

NewWindow
NewCWindow
NewDialog
OpenCPort
ClosePort
CloseCPort
ShowWindow
HideWindow
ShowHide
SizeWindow
ZoomWindow

Mauszeiger API

SetCursor
SetCCursor

Zusätzliche Macintosh Toolbox Aufrufe

NewControl
SetControlTitle

10.2. Bewertung von T.128

Die Bewertung von T.128 (auch T.SHARE) wurde durchgeführt für die ITU Study Group 16 während des Standardisierungsprozesses im Jahr 1997.

1. T.SHARE is a direct conversion of the (old) Microsoft Win16 GDI calls into protocol data units. It includes neither the additional functions of the Win32 GDI nor functionality similar to the one offered by the Windows NT display driver model.
2. Postscript and related graphics interfaces (Display Postscript, PDF, java2d) will not map well onto T.SHARE, because they are at a higher abstraction level. The other direction is theoretically simpler, but costly because of special Windows GDI features built into the protocol.
3. Implementors of T.SHARE terminals will re-implement or licence the MS Windows graphics engine, although there are other more graphics system independent approaches to encoding of graphics output available.
4. A stroke/fill/clip-path oriented semantics is much more platform independent. E.g. the 3 graphics systems represented by the Windows NT diplay driver model, Quickdraw GX and (Display-)Postscript map much better onto each other than Win16.
5. The graphics primitives curve (e.g. quadratic or cubic beziers), polygon and their respective attributes usually known as cap-style, join-style, etc. are missing.
6. There are no substantial differences between legacy-mode and base-mode. T.SHARE represents the NetMeeting protocol except for few syntactical changes, which leave parts of the protocol open for extensions. There are no differences on semantical level.
7. Font uploading is missing although it proved to be essential in heterogenous scenarios. Font matching is not sufficient. The lack of font uploading restricts application sharing to equally equipped terminals. But even on a homogeneous operating and graphics system platform additional font installations will not be shared properly. This is a major drawback, which made sharing between X Window terminals difficult in the past.
8. The late-joining problem showed that all resources, which are allocated in the terminal must be interrogatable (downloadable from the terminal). All resources (fonts, bitmaps, clip-regions, color palettes, etc.) which have been transmitted and which are still valid are neccessary to accomodate for late-joiners.
9. Bitmap definitions are identical to Windows DIBs (Device Independent Bitmaps). The restriction to 1, 4 and 8 bit per pixel is unsubstantiated. The 2-dimensional RLE encoding is fine.
10. All color specifications are palettized as opposed to true color values. However true color values would allow for device independent graphics output streams, and of course for true color applications. True color values avoid artificial diminishing of presentation quality. They proved to be not significantly more bandwidth consuming, if used in conjunction with state of the art RLE- and delta-encoding.
11. Other compression schemes for bitmap encoding, which are already offered by T.126 are desirable. Namely JPEG (T.81) for continuous tone images as a counterpart to RLE-encoded computer generated graphics. PNG (Portable Network Graphics) is a good candidate for compressed bitmaps with large color spaces.
12. Bitmap decoding may perform worse on MSByte-first (e.g. IBM, Motorola, SPARC) than on LSByte-first (Mips, Intel) processors, because of fixed LSByte-first ordering as opposed to a MSBit-first ordering in the same document. Switching to so called Internet network byte order would not harm Intel processors.

Trotz der hier genannten Bedenken wurde der Vorschlag von Microsoft und PictureTel fast nicht verändert, da andere betroffene Firmen (z.B. Sun, Adobe) keinen Einfluß nahmen. Punkt 4 stellt für moderne Grafiksysteme, wie Win32 und Java2d eine große Einschränkung dar. Vor allem pfad- und bereichsorientierte Eigenschaften sollen in die nächste Revision des Standards eingehen.

10.3. Beispielsitzungen

Die folgenden Beispielsitzungen sollen Arbeitsabläufe darstellen, die für die jeweilige Anwendungsprogramm und ähnliche Programme typisch sind. Sie wurden, wenn verfügbar, aus den Programmdokumentationen oder Anwendungsbeispielen übernommen. An diesen Arbeitsabläufen wurden Messungen durchgeführt (Anhang *Messungen*), die bestimmte Charakteristika von Grafikströmen hervorheben.

Natürlich können die Beispielsitzungen nicht alle möglichen Fälle abdecken. Es wurde aber sowohl bei der Entwicklung der in Kapitel 7 genannten Sharing Systeme, als auch im Rahmen der Messungen eine große Zahl verschiedener Szenarien durchgespielt. Dabei traten keine wesentlichen Abweichungen von den in Anhang *Messungen* dargestellten Ergebnissen auf.

Die Messungen wurden auf MacOS 7 mit aktuellen Programmversionen durchgeführt. Dazu wurde in das Sharing System M-MaX eine Statistikkomponente integriert. Die Ausgabe auf einem entfernten Endgerät wurde abgeschaltet, um die Messungen der von Anwendungsprogrammen erzeugten Grafikströme fast ohne Einwirkung des Application Sharing Systems durchzuführen.

Eine vergleichbare Statistikkomponenente wurde auch in das aus [Schöttner96] hervorgegangene Windows-Application Sharing System integriert. Dabei stellte sich heraus, das die in Windows (Win32 GDI) gemessenen Grafikstromeigenschaften den unter MacOS 7 gemessenen sehr ähnlich sind.

Es wurden vier verschiedene Klassen von Büro-EDV Anwendungen untersucht:

1. Bildbearbeitung wie beim Grafikdesign,
2. Textbearbeitung im ASCII-Editor wie bei Programmierung,
3. Textverarbeitung wie im Schreibbüro (Brief),
4. Tabellenkalkulation in kaufmännischer Anwendung (Rechnung).

10.3.1. Bildbearbeitung

Programm: Adobe Photoshop Version 3.0

Dokumente: Bild1: 1250x1850 Pixel Darstellung 33%, Bild2: 1750x2350 Pixel Darstellung 33%. Beide Bilder von den Photoshop Beispielen.

Zusammenfassung: Bereich färben, Ausschneiden eines Teilbildes mit Freiform-Rand, Einsetzen und Transformieren in der Objektebene eines zweiten Bildes.

Startkonfiguration: Photoshop gestartet, Werkzeugpaletten sichtbar, kein Dokumentenfenster.

Bemerkung: Diese Beispielsitzung soll einen kurzen Bearbeitungsvorgang eines mit dem Programm geübten Benutzers wiedergeben.

Dauer: ca. 75 Sekunden.

Aktion:	Zeit in Sek.
Datei Dialog aus Menü	1
Bild1 laden/darstellen	3
Werkzeug selektieren (Zauberstab)	0,5
Werkzeug einstellen (Schwellwert)	1
Werkzeug anwenden (Klick) -> Selektion	0,5
Bereich löschen	0,5
Farbe wählen im Farbauswahl Dialog	6
Werkzeug wählen (Fläche Füllen)	0,5
Datei Dialog aus Menü	2
Bild2 laden/darstellen	3
Werkzeug selektieren (Zauberstab)	0,5
Werkzeug einstellen (Schwellwert)	2
Werkzeug anwenden (Klick) -> Selektion	0,5
Auswahl invertieren	1
Umschalten in Maskenmodus	0,5
Werkzeug selektieren (Pinsel)	0,5
Werkzeug einstellen (Pinseltyp)	1
Mehrfache Malbewegungen mit gedrückter Maustaste (ca. 30 Sek.)	30
Umschalten in Standardmodus	0,5
Kopieren durch Tastaturkürzel	0,5
Wechseln zu Bild1	1
Einsetzen durch Tastaturkürzel	0,5
Freie Transformation aus Menü	2
Objekt Verschieben	1
Objekt Skalieren	1
2 mal Objekt Drehen	3
Transformation Ausführen durch Eingabetaste	0,5
Speichern als Datei Dialog aus Menü	2
Bild1.neu speichern	5
Bild1 schließen am Fensterrahmen -> Bild1 schließt	0,5
Bild2 schließen am Fensterrahmen	0,5
Dialognachfrage des Programms (Änderungen speichern?)	1
Antwort (Nein) durch Klick auf Knopf -> Dialog und Bild2 schließen	0,5

10.3.2. Textbearbeitung im ASCII-Editor

Programm: BBEdit 3.0

Dokument: 71 kB Pascal-Quellcode.

Zusammenfassung: Zeichenkette suchen und wenige Zeichen ändern.

Startkonfiguration: BBEdit gestartet, kein Dokumentenfenster.

Bemerkung: Diese Beispielsitzung soll einen kurzen Editiervorgang eines geübten Programmierers wiedergeben. Aus diesem Grund werden fast alle Kommandos durch Tastaturkürzel gegeben.

Dauer: ca. 40 Sekunden.

Aktion:	Zeit in Sek.
Datei Dialog mit Tastaturkürzel	0,5
Kurze Suche nach Datei	10
Datei öffnen	1
Suchfunktion mit Tastaturkürzel	0,5
Suchdialog	0,5
Eingabe Zeichenkette und Eingabetaste	4
Programm springt an gesuchte Zeichenkette	0,5
Zeichenkette hervorgehoben (3 mal, jeweils weiter mit Tastaturkürzel)	1,5
Einfügemarke mit Tastatur positionieren	3
mehrmals wenige Zeichen eingeben und Einfügemarke setzen	18
Datei speichern mit Tastaturkürzel	0,5

10.3.3. Textverarbeitung

Programm: Framemaker 4

Dokument: dreiseitiger Text mit einer viertelseitigen Grafik aus den Framemaker 4 Beispieldokumenten.

Zusammenfassung: Absatz hinzufügen und Absatz ändern.

Startkonfiguration: Framemaker gestartet, kein Dokumentenfenster,

Bemerkung: Textausgabebeschleunigung wurde ausgeschaltet, da sonst Text als Bitmap ausgegeben wird und sich das Programm damit verhält, wie ein Bildbearbeitungsprogramm.

Dauer: ca. 450 Sekunden.

Aktion:	Zeit in Sek.
Datei öffnen durch Doppelklick auf Dateisymbol	0,5
Dokumentenfenster öffnet sich	1
nächste Seite zeigen	0,5
nächste Seite zeigen	0,5
Einfügemarke mit Maus positionieren	1
Absatz eingeben (insgesamt ca. 1000 Zeichen),	410
dabei einmal drei Zeilen mit Maus selektieren und löschen	
Einfügemarke mit Maus positionieren	0,5
ca. 100 Zeichen einfügen.	35
Datei speichern durch Tastaturkürzel	0,5

10.3.4. Tabellenkalkulation

Programm: Excel Office 98

Dokument: einseitiges Rechnungsformular aus Office 98 Beispieldokumenten.

Zusammenfassung: Rechnung ändern und drucken.

Startkonfiguration: Excel gestartet, Werkzeugpaletten sichtbar, kein Dokumentenfenster.

Dauer: ca. 30 Sekunden.

Aktion:	Zeit in Sek.
Datei öffnen durch Doppelklick auf Dateisymbol	0,5
Paletten verschwinden und erscheinen wieder	1
Dokumentenfenster öffnet sich	1
Änderung einer Zelle durch Doppelklick und kurze Eingabe	5
Eingabe einer neuen Zeile: mehrmals, wie oben	20
Drucken Dialog mit Tastaturkürzel	0,5
Bestätigen durch Eingabetaste	0,5
Datei speichern durch Tastaturkürzel	0,5
Datei schließen	1

10.4. Messungen

10.4.1. Windows NT LPC

Um den Aufwand für die Übertragung von Grafikkommandoparametern im Vergleich zur Grafikausgabe selbst abzuschätzen, wurde die maximal erreichbare Rate von Windows NT Local Procedure Calls (LPC) gemessen. Die Messung wurde auf einem Pentium 100 mit 32 MB Hauptspeicher unter Windows NT Workstation durchgeführt. Die schnellste LPC Variante (QuickLPC) ausgeführt zwischen Anwendungsprogramm und Betriebssystemkern erreicht 6000/Sek. [Schöttner97a].

Die Messung wird vom verfügbaren Hauptspeicher nicht beeinflußt. Cache Performance und Systemlast durch Hintergrundtasks spielen dagegen eine wesentliche Rolle. Hier soll aber nur die Größenordnung der Kosten eines LPC im Vergleich zur Grafikausgabe abgeschätzt werden.

Ergebnis: Je nach Prozessortyp benötigt ein LPC 5000-10000 Instruktionen. Dies ist vergleichbar mit der Ausgabe kleinerer Pixmaps oder Rechtecke ohne Grafikbeschleuniger. Die Verwendung von Interprozeßkommunikation bremst Grafikausgaben ab. Die Kosten sind aber nur bei Grafikhardwarebeschleunigung unverhältnismäßig. Aus diesem Grund wurde der LPC Server für Grafikausgaben mit NT 4.0 in den Kernel verlegt.
Die Zahl der bei einem LPC übergebenen Parameter spielt keine Rolle, da die Zahl der Instruktionen je Parameter im Vergleich vernachlässigbar ist.

10.4.2. Stiftformen

Stiftformen wurden nach Voruntersuchungen in mehrere Klassen eingeteilt und für jede Beispielsitzung (Anhang *Beispielsitzungen*) über die gesamte Sitzung je Klasse gezählt.

	1x1	2x2	3x3	1x2	1x3	1xX	<=8	<=16	>16
Bildbearbeitung	5057	50	49			14			14
Textbearbeitung (ASCII)	1380	2	18			2			2
Textverarbeitung	1686	6	17						
Tabellenkalkulation	5779		17						

Tabelle Stiftformen

Verteilung der Stiftformen, die in den Beispielsitzungen zum Zeichnen von Grafikprimitiven verwendet werden. Quickdraw Stifte sind immer durch Rechtecke definiert. Die Kategorien (Spalten) beziehen sich auf Stiftfläche und Form. 1xX bedeutet Breite oder Höhe 1. Die drei rechten Spalten bezeichnen die Stiftfläche in Fällen, die nicht durch die 6 linken Spalten abgedeckt werden.

Ergebnis: Bei weitem der größte Teil der Grafikprimitive bei denen die Stiftform relevant ist, wird mit einem einfachen Stift der Größe 1x1 Pixel durchgeführt. Dies gilt auch für Win32 [Schöttner98].

10.4.3. Transfermodi

Für jede Beispielsitzung wurden die bei CopyBits und CopyMask verwendeten Transfermodi gezählt

	src Copy	src Or	src Xor	src Bic	not Src Copy	not Src Or	not Src Xor	not Src Bic	arith-metic
Bildbearbeitung	4437			584					
Textbearbeitung (ASCII)	129	2		1					
Textverarbeitung	458	2							
Tabellenkalkulation	1037	15		9			12		

Tabelle Transfermodi

Verteilung der Transfermodi in den Beispielsitzungen. Die acht arithmetischen Modi erschienen in keiner der Beispielsitzungen.

Ergebnis: Fast alle Pixmaps werden mit dem deckenden Transfermodus gezeichnet Die arithmetischen Transfermodi werden kaum verwendet. SrcBic wird bei der Bildbearbeitung verwendet, um die sogenannte schwebende Auswahl anzuzeigen.

10.4.4. Reduktion durch Pixmapkompression

Gemessen wurde die Zahl der ausgegebenen Pixel, sowie die Zahl aller Grafikprimitivtypen. Für Pixmaps wurde 1 Byte pro Pixel angesetzt. Dies entspricht Ausgaben mit 8-Bit Farbpalette. Die tatsächlich durch Pixmaps erzeugte Datenrate hängt wesentlich von der eingestellten Bildschirmtiefe ab, da sich gerade Bildbearbeitungsprogramme bei ihren Bildschirmausgaben an die Bildschirmtiefe anpassen, um eine optimale Darstellung zu erzielen.

	Zahl der Pixmapausgaben	Zahl der Pixel	Daten aus allen Primitivtypen	Datenvolumen nach Pixmapkompression
Bildbearbeitung	5268	17.600.000	17.700.000	25%
Textbearbeitung (ASCII)	267	344.000	534.000	52%
Textverarbeitung	480	1.940.000	2.000.000	27%
Tabellenkalkulation	1261	3.720.000	3.830.000	27%

Tabelle Pixmapkompression

1. Spalte: Zahl der Pixmapausgaben pro Beispielsitzung.
2. Spalte: Zahl der Pixel (gerundet).
3. Spalte: Gesamtes Datenvolumen bei 1 Byte pro Pixelausgabe (gerundet).
4. Spalte: Reduktion bei Pixmapkompression auf 2 Bit pro Pixel.

Ergebnis: Auch bei Textverarbeitung und Tabellenkalkulation überwiegt der Anteil der Pixmapausgaben trotz deutlich geringeren absoluten Zahlen, als bei der Bildverarbeitung. Entsprechend reduziert sich das Datenvolumen bei Pixmapkompression fast proportional. zum Kompressionsfaktor für Pixmaps. Nur bei der ASCII Textbearbeitung überwiegen andere Grafikprimitive, so daß Pixmapkompression etwas weniger ins Gewicht fällt.

Ergebnis: Beim Application Sharing von Anwendungsprogrammen, die nicht für die Bedienung über Netzwerke entworfen wurden fallen hohe Datenvolumina an. Natürlich dominiert die Bildverarbeitung mit 18 Millionen Pixeln in 75 Sekunden, d.h. 0,25 MByte (8 Bit) bis ca. 0,75 MByte (24 Bit) pro Sekunde, je nach Bildschirmtiefe. Aber auch die Tabellenkalkulation erzeugte im Mittel 130 kByte bis ca. 400 kByte pro Sekunde.

10.4.5. Textvarianten

	Plain	B	I	U	BI	BU	IU	BIU	andere
Bildbearbeitung	1330								11
Textbearbeitung (ASCII)	4864								2
Textverarbeitung	923	56	10						
Tabellenkalkulation	947	82	49						

Tabelle Textvarianten

Verteilung der Textmodifikationen in den Beispielsitzungen. Es wurden nur die Kombinationen von Bold (b), Italic (I) und Underline (U) getrennt aufgenommen. Zeichengrößen wurden nicht berücksichtigt.

Ergebnis: Fast alle Textausgaben werden mit der unmodifizierten Basisvariante durchgeführt. Bei Textverarbeitung und Tabellenkalkulation ca. 10% mit einfachen Modifikationen.

10.5. Strichstärke und Stiftformen

Sowohl eine geschlossene Form (Rechteck, Kreis, Polygon), als auch die Form einer Linie kann wesentlich von der Geometrie des verwendeten Stifts abhängen. Bei X11 ist die Breite einer Linie unabhängig vom Winkel. X11 kennt nur den Parameter Linienbreite, aber keine Stiftform.

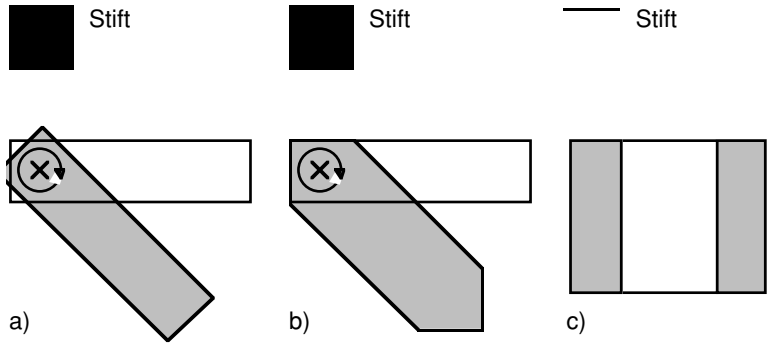


Abbildung Strichstärke und Stiftformen

Auswirkung der Stiftform auf die Ausgabe von Grafikprimitiven.

a) und b) zeigen einen quadratischen Stift. Wird der Stift, wie in Quickdraw, über die Pixelebene gezogen und jedes berührte Pixel eingefärbt, dann hängt die Breite einer Linie vom Winkel ab, falls die Stiftfläche größer als 1 ist (c).

d) zeigt ein Quadrat, das mit der exzentrischen Stiftform 1x10 gezeichnet wurde.

10.6 Pixmapausgaben

10.6.1 Datenstruktur

Grafiksystem	Datenstruktur	Kommentare
Quickdraw API und Treiber-schnittstelle	PixMap baseAddr (1) rowBytes (2) bounds (3) pmVersion packType packSize hRes vRes pixelTyp pixelSize (5) cmpCount (4) cmpSize planeBytes pmTable pmReserved ColorTable ctSeed ctFlags ctSize (6) ctTable[] (7)	baseAddr ist ein Zeiger auf die Pixeldaten. pmTable ist Zeiger auf die Farbtabelle. ctTable ist eine Folge von (index, RGBColor)
Win32 API	DIB biSize biWidth (3) biHeight (3) biPlanes (4) biBitCount (5) biCompression biXPelsPerMeter biYPelsPerMeter biClrUsed (6) biClrImportant Colors[][1] (7) Bits[] (1)	Device Independent Bitmap (DIB). Nach dem Header der Bitmap folgen die Farbtabelle, falls indizierte Farben vorliegen. Die Pixeldaten sind direkt angehängt. Colors ist ein Feld der Datenstruktur RGBQUAD. Bits enthält die Pixeldaten. Die Pixeldaten können auch komprimiert sein. RLE[2] wird von Win32 unterstützt.
X11 Protokoll-schnittstelle	Pixmap width (3) height (3) dst-x dst-x left-pad (2) depth (4) data[] (1)	data[] ist eine Folge von Bytes, die Pixeldaten enthalten. Die Anordnung der Farbkomponenten in einzelnen Pixeln ist nicht für jede Pixmap verschieden, sondern gegeben durch den Displayserver. Sie wird beim Verbindungsaufbau vereinbart, d.h. vom X-Server mitgeteilt (siehe [Xv0-90] Seite 50).

Tabelle Pixmapformate - Teil 1

Grafiksystem	Datenstruktur	Kommentare
GIF	Image ImageWidth (3) ImageHeight (3) ColorTableFlag InterlaceFlag (2) Sort Flag Reserved ColorTableSize (6) ColorTable[][1] (7) Minimum Code Size Data[] (1)	ColorTable[] ist eine Folge von 24-bit RGB-Werten. Die Farbtabelle ist optional. Die Existenz wird durch das ColorTableFlag angezeigt. Ist sie nicht vorhanden, dann gilt die globale Farbtabelle. Dies entspricht genau dem Verhältnis von DDB (Device Dependent Bitmap) und DIB bei Win32. Das InterlaceFlag bestimmt die Anordnung der Scanlines. Data[] ist eine LZW kodierte Sequenz von Farbindexwerten [Welch84].
MPEG	Macroblock quantizer_scale[1] (5) Bewegungsvektoren[1] coded_block_pattern[1] (2) block[] (1)	block[] ist eine Folge von bis zu 6 8x8 Blöcken, die DCT-Koeffizienten (siehe [MPEG1-92]) für Farbkomponenten im Y:U:V Raum (2x2:1:1) enthalten. Die Kombination der Farbkomponenten ergibt 16x16 Pixel. Die Komponente quantizer_scale beeinflußt die Interpretation der Pixeldaten. Die Definition des Macroblocks ist fest vorgegeben. Die Komponenten Höhe (3), Breite (3) und Bitanordnung (4/5) sind deshalb nicht in der Pixmap enthalten. Die Komponente coded_block_pattern bestimmt welche der 6 8x8 Blöcke in block[] enthalten sind. Die Basisadresse (1) entfällt, weil die Daten, wie im Fall des X Protokolls direkt folgen. Zeilenabstand bzw. Padding (2) entfallen, da alle Bits gepackt und nicht auf Byte oder Wortgrenzen beginnen müssen.
Postscript	Image width (3) height (3) bits/sample (5) matrix function Data[] (1)	Eine Postscript-Funktion stellt die Daten für des Image zur Verfügung. In einem Postscriptstrom folgen normalerweise die Daten einem Verweis auf die Datenfunktion, während die Funktion selbst im Vorspann definiert ist. Die Daten Bytes, die je nach bits/sample-Wert (bps) einen oder mehrere Pixel enthalten. Postscript 2 erlaubt RLE-kodierte Daten und RGB-Werte, d.h. 24 Bit/Pixel Die Farbtabelle ist fest auf eine Rampe von 0 bis 2^{bps}-1 eingestellt.

1 Komponente ist optional.
2 RLE: Run Length Encoding = Lauflängenkodierung (siehe Kapitel 4.3.3.1).

Tabelle Pixmapformate

Die Komponenten der Pixmaps sind entsprechend der Dokumentation des jeweiligen Grafiksystems benannt. Die klein geschriebenen Felder sind in diesem Zusammenhang nicht von Bedeutung. Sie sind nur zur Vollständigkeit aufgeführt, um die Anordnung der Komponenten in den Datenstrukturen nicht zu verfälschen. Die Zahlen in Klammern zeigen an, welche Art von Information jede Komponente enthält (siehe Text). Eckige Klammern bezeichnen Byte-Arrays.

10.6.2 Ausgabekommando

Grafiksystem	Pixmapausgabekommando

Quickdraw API und Treiber-schnittstelle	`CopyBits` `Primitivtyp`
	`srcBits` `Pixmap`
	`dstBits` `Zielbereich`
	`srcRect` `Auszugebender Teil der Pixmap`
	`dstRect` `Zielkoordinaten und Skalierung`
	`mode` `Transfermodus`
	`maskRgn` `Clippingbereich`
	Das CopyBits Kommando wird ohne Berücksichtigung eines Grafikkontextes ausgeführt. Deshalb sind Transfermodus und Clippingbereich im Kommando enthalten.
	Bei der Ausgabe kann nicht nur skaliert werden (Rechteck dstRect als Zielbereich), sondern auch nur ein Teil der Pixmap tatsächlich ausgegeben werden (srcRect)

Win32 API	`BitBlt` `Primitivtyp`
	`hdcTarget` `Zielbereich und Grafikkontext`
	`nTargetX` `Zielkoordinate`
	`nTargetY` `Zielkoordinate`
	`nWidth` `Betroffener Teil der Pixmap`
	`nHeight` `Betroffener Teil der Pixmap`
	`hdcSource` `Pixmap`
	`nSourceX` `Betroffener Teil der Pixmap`
	`nSourceY` `Betroffener Teil der Pixmap`
	`dwRaster` `Transfermodus`

X11 Protokoll-schnittstelle	`PutImage` `Primitivtyp`
	format, request-length
	`drawable` `Zielbereich`
	`gc` `Grafikkontext`
	depth, width, height
	`dst-x` `Zielkoordinate`
	`dst-y` `Zielkoordinate`
	`Pixmap`
	Transfermodus und Clipping sind im Grafikkontext (gc) festgelegt.
	Die Komponenten depth, width und height sind in der PDU vor den Zielkoordinaten angeordnet. Sie gehören aber eigentlich zum Pixmapheader (siehe Tabelle *Pixmapformate*).

GIF89a	`ImageSeparator` `Primitivtyp`
	`ImageLeftPosition` `Zielkoordinate`
	`ImageTopPosition` `Zielkoordinate`
	`Image`
	Es gibt nur einen einzigen Grafikkontext. Der Grafikkontext enthält die Felder TransparencyFlag, DelayTime, TransparencyIndex, DisposalMethod und UserInputFlag. Von diesen Feldern übernehmen TransparencyFlag und TransparencyIndex die Funktion des Transfermodus, bzw. eines Clippingbereichs.
	Der Grafikkontext kann durch ein Kontextmanipulationskommando (sog. Graphic Control Extension) vor dem Ausgabekommando manipuliert werden.

Tabelle Pixmapausgabekommandos - Teil 1

Grafiksystem	Pixmapausgabekommando

MPEG Videostrom	`macroblock_address_increment` `Zielkoordinaten`
	`macroblock_type` `Primitivtyp`
	`quantizer_scale`[1]
	`Bewegungsvektoren`[1]
	`Macroblock` `Pixmap`
	Der Transfermodus bestimmt sich aus dem Macroblocktyp. Je nach Typ werden die Pixel kopiert oder zum Zielbereich addiert.
	Die Komponente quantizer_scale gehört zur Definition der Pixmap.
	Die Bewegungsvektoren modifizieren den Zielbereich durch Kopieren eines 16x16 Pixelblocks aus dem Zielbereich von der durch die Bewegungsvektoren angegebenen Stelle bevor die Pixmap ausgegeben wird.

Postscript	`matrix` `Zielkoordinaten und Skalierung` **`image`** Clipping wird durch den 'clipping path' des Grafikkontextes bestimmt. Die 'transfer'-Funktion beeinflußt die Abbildung der Pixelwerte in Farbwerte.

1 Komponente ist optional.

Tabelle Pixmapausgabekommandos

Die Komponenten der Ausgabekommandos sind entsprechend der Dokumentation des jeweiligen Grafiksystems benannt. Die klein geschriebenen Felder sind in diesem Zusammenhang nicht von Bedeutung. Sie sind nur zur Vollständigkeit aufgeführt, um die Anordnung der Komponenten in den Datenstrukturen nicht zu verfälschen. Die hervorgehobenen Komponenten sind die Pixmaps aus Tabelle *Pixmapformate*.

10.7. Verzeichnis der Abbildungen

10.8. Verzeichnis der Tabellen

10.9 Glossar

Funktion = Funktion im ursprünglichen Wortsinn, kein Terminus von Programmiersprachen.

globale Koordinaten = bezogen auf den Bildschirm oder einen Punkt für mehrere Bildschirme.

GUI-Toolbox = Prozedursammlung

Interrupt = Die Unterbrechung des Programmablaufs (Anwendungsprogramm oder System) durch ein Hardwaresignal ausgelöst von einer externen Einheit.

ISDN = Integrated Services Digital Network

JIT = Just In Time Compiler

modal = Im Bezug auf Fenstereigenschaften die Eigenschaft, einen Dialog mit dem Benutzer zu erzwingen.

Offscreen-Pixmap = Eine Teil des Hauptspeichers, in dem Pixeldaten abgelegt werden, der aber nicht auf einem Bildschirm dargestellt wird.

PDU = Protocol Data Unit = Protokolldateneinheit

Pixel = Picture Element: Bildpunkt, elementarer Teil einer Pixmap.

Polling = Der Prozeß des wiederholten Abfragens eines Zustands.

Programm = Anwendungsprogramm = Application

Prozedur = ausführbares Codefragment

Quellrechner = Rechner auf dem ein gemeinsam verwendetes Anwendungsprogramm läuft. Das Grafiksystem des Quellrechners wird als Quellgrafiksystem bezeichnet.

Rendering = Prozeß der Umwandlung von Grafikkommandos in Pixelgruppen und Ausgabe der entsprechenden Pixel in eine Pixmap.

Reverse-Engineering = Untersuchung der Funktionsweise eines Systems, um die Funktionsprinzipien zu erschließen.

Round-Trip-Request = Beim X11-Protokoll ein Kommando, das eine Antwort von der Gegenstelle erzwingt.

Routine = Prozedur

Software-Interrupt = Methode zum Aufruf einer Prozedur über die Tabelle der Interrupt-Vektoren.

Toolbox = Prozedursammlung

Toolkit = GUI-Toolbox

Lebenslauf

Geboren wurde ich als erstes Kind der Eheleute Dr. Heinz Wolf und Rosmarie Wolf, geborene Kunz, am 27.10.1966 in Freiburg. Dort besuchte ich von 1972 bis 1976 die Grundschule. Ab 1976 besuchte ich das Keplergymnasium in Freiburg, wo ich 1985 die Reifeprüfung ablegte. Danach leistete ich meinen Grundwehrdienst vom Juli 1985 bis Oktober 1986.

Direkt im Anschluß daran begann ich das Studium der Physik an der Universität Freiburg. Im Herbst 1988 bestand ich das Vordiplom und am 1.6.1992 die Diplomprüfung. Während der Diplomarbeit verbrachte ich ein Jahr am CERN/Genf.

Seit Februar 1993 arbeite ich als wissenschaftlicher Mitarbeiter in der Abteilung Verteilte Systeme der Universität Ulm.

Name	Klaus Heinrich Wolf
Anschrift	Fischergasse 20/1 89073 Ulm
Telefon	privat: 0731 / 67779 dienstl.: 0731 / 5024145
geboren am	27.10.1966
Familienstand	ledig
Schulbildung	Grundschule, 1972 - 1976 Keplergymnasium in Freiburg, 1976 - 1985.
Hochschulstudium	Physik an der Universität Freiburg, 1986 - 1992.
Hochschulabschluß	Diplom in Physik am 1.6.1992, Nebenfach Mathematik.
Anstellung	seit dem 1.2.93 als wissenschaftlicher Angestellter der Universität Ulm in der Abteilung Verteilte Systeme.

Ulm, den 14. Februar 2000, Klaus H. Wolf

Danksagungen

Ich bedanke mich ganz herzlich bei Herrn Professor Dr. Schulthess dafür, daß er mir mit der Anstellung in seiner Abteilung diese Arbeit ermöglicht hat. Ich bin außerdem zu Dank verpflichtet für die intensive Unterstützung, seine ständige Bereitschaft zur Diskussion, die unzähligen Anregungen und Hinweise. Als besonders wichtig und fördernd empfand ich die von Herrn Schulthess geschaffene Arbeitsatmosphäre. Herrn Schulthess leitete mich auch dazu an, selbständig über ungewöhnliche Lösungen nachzudenken und dazu Probleme anzugehen und schließlich zu lösen von denen andere behaupteten, sie seien nicht lösbar.

Mein besonderer Dank gilt PD Dr. Konrad Froitzheim für die jahrelange sehr gute und enge Zusammenarbeit in Projekten, wie in der Doktorarbeit. Neben vielen großen und kleinen Dingen lernte ich von Herrn Froitzheim, daß Computer doch deterministisch sind, und daß es sich lohnt, Interna auf jeder Ebene verstehen zu wollen.

Ich danke den Diplomanden und Hilfskräften der Abteilung, die durch Gespräche meine Gedanken befruchtet und mit ihrer Arbeit wesentlich zu verschiedenen Implementierungen beigetragen haben. Besonders möchte ich diesem Zusammenhang Markus Maier, Jan Ulbrich, Michael Schöttner, Stefan Fiedler und Lars Kaufmann danken.

Neben Herrn Schulthess ist das gute Arbeitsklima auch den Kollegen der Abteilung zu verdanken. Sie waren stets hilfsbereit und haben durch viele Gespräche ihren Teil beigetragen. Das gilt besonders für Herrn Bönisch, daß er mir im letzten Jahr der Doktorarbeit den Rücken frei gehalten hat.