Der JX – Windowmanager

Marc Lörner, Johannes Ostler

AKBP II WS 2003/04

1.Einleitung

1.1 Zielsetzung des Praktikums

Im Rahmen des Praktikums AKBP II wurden wir, Marc Lörner und Johannes Ostler, damit beauftragt den Windowmanager des Java-Betriebssystem JX in Beziehung auf die grafische Gestaltung und Perfomance zu verbessern. Die Grundlage hierfür bilden der von Jürgen Obernolte programmierte JX Windowmanager und die darauf aufbauende AWT - Implementierung von Marco Winter.

1.2 Weitere Informationen, Literaturnachweis

Wir werden im folgenden nur grob die Konzepte dieser Vorarbeiten eingehen, um dann die vorgenommen Änderungen zu erläutern. Für nähere Informationen empfehlen sich folgende Studienarbeiten, die unter http://www.jxos.org/publications.html zu finden sind:

- Jürgen Obernolte, Entwurf und Implementierung eines Windowmanagers fuer das Java-Betriebssystem JX , Februar 2002

- Marco Winter, Design und Implementierung der AWT-Schnittstelle fuer das Java-Betriebssystem JX , Oktober 2002

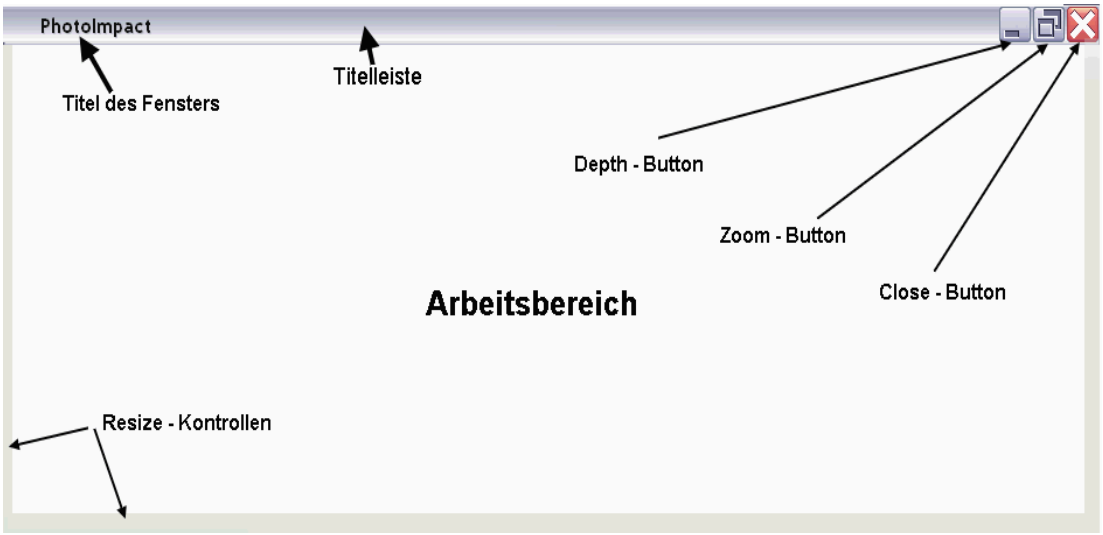
2.Überblick über die Konzeption des JX – Windowmanagers

2.1 Was ist die Aufgabe eines Windowmanagers

Die Aufgabe eine Windowmanagers ist die Verwaltung von Fenstern. Sowohl das optische Aussehen als auch Grundfunktionalitäten, wie das Ändern der Größe oder das Verschieben von Fenstern muss er zur Verfügung stellen. Grundlegende Zeichenfunktionen sind ebenfalls in ihm verankert.
Das Zeichnen von Buttons, Listen und ähnlichen grafischen Objekten hingegen ist Aufgabe z.B. einer AWT – Implementierung und nicht des Windowmanagers.

2.2 Die Bereiche eines Fensters

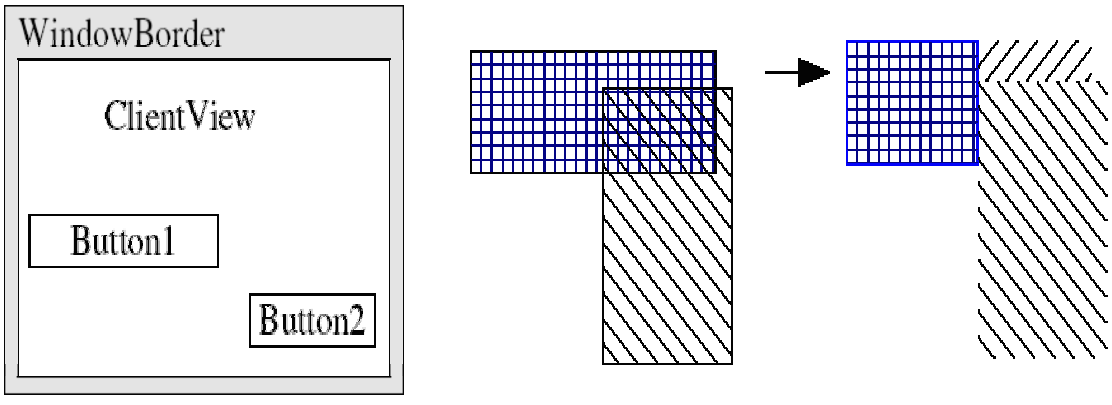
Jedes Fensters ist in zwei Bereiche aufgeteilt, zum einen die Titelleiste und den das Fensters umgebenden Rahmen, zum anderen den Arbeitsbereich im Zentrum. Die Gestaltung des Arbeitsbereichs bleibt der Anwendung überlassen und wird normalerweise mit Hilfe der AWT – Klassen umgesetzt.



2.3 Wichtige Klassen

2.3.1 WView und WRegion

Eine View ist ein rechteckiger Bereich auf dem Bildschirm. In der Klasse WView ist sowohl die Position der linkesn oberen Ecke des Bereichs in Relation zum Bildschirm als auch die Breite und Höhe des sie definierenden Rechteckes gespeichert. Weiterhin kann jede View beliebig viele Views beinhalten, sogennante Childviews, wie z.B Buttons oder Bilder . EinFenster ist durch eine Instanz der Klasse WindowBorder, die von WView abgeleitet ist, realisiert. Diese beinhaltet wiederum eine ClientView, den Arbeitsbereich.
Wird ein Teil einer View überdeckt, so wird dieser Bereich und eventuell auch jeweilige Bereiche von Childviews als beschädigt registriert. Die beschädigten Bereiche werden als Region abgespeichert, das heißt einer Anzahl von möglichst wenigen Rechtecken, die die beschädigte Fläche genau überdecken. Diese Bereiche werden dann neu gezeichnet.



2.3.2 WWindow

Diese Klasse stellt das Kernstück der Aktivität des Fensters dar. Sie implementiert das Interface Runnable und ist für die Abarbeitung von Messages zuständig, die an das Fenster geschickt werden. Zur Implementierung der gewünschten Aktivität muss von dieser Klasse abgeleitet und die jeweiligen Methoden überschrieben werden.
Sie beinhaltet auch die Verbindung zum eigentlichen Fenster durch ein Objekt der Klasse WWindowInterface und zum "Fenstererzeuger" durch eines der Klasse WindowManager.

2.3.3 WindowManager und WindowManagerImpl

Das Interface WindowManager ist das Kernstück des JX – Windowmanagers. Mit Hilfe der Implementierung WindowManagerImpl werden neue Fenster erzeugt und an die Fenster Maus- und Tastaturereignisse weitergeleitet.

2.3.4 WWindowImpl und WindowBorder

WWindowImpl bildet das eigentliche Kernstück des Fensters. Bei Maus und Tastaturereignissen werden die statischen Handler – Funktionen aufgerufen. Hierauf wird entschieden an welches Fenster die jeweiligen Ereignisse weitergeleitet werden. Mit Hilfe der Klasse WindowBorder wird festgestellt, ob die Maus sich über dem Rahmen und wo auf diesem oder auf der Arbeitsfläche befindet.
WWindowImpl reagiert dann auf das Ereignisse und schickt eventuell eine Message an die dem Fenster zugeordnete WWindow – Instanz.

2.3.5 WSprite

Der Mauszeiger ist mit Hilfe der Klasse WSprite realisiert. Es Werden der jeweilige vom Mauszeiger verdeckte Bildbereich sowie der Mauszeiger selbst als WBitmap abgespeichert. Bei der Bewegung des Mauszeigers wird zunächst der gespeicherte Hintergrund an die alte Mausposition gezeichnet, dann der neue, von der Maus verdeckte Bereich gesichert und schließlich der Mauszeiger an der neuen Mausposition angezeigt. Tatsächlich müssen hierfür nur Speicherbereiche kopiert werden.

3.Die Anbindung der AWT – Implementierung an den Windowmanager

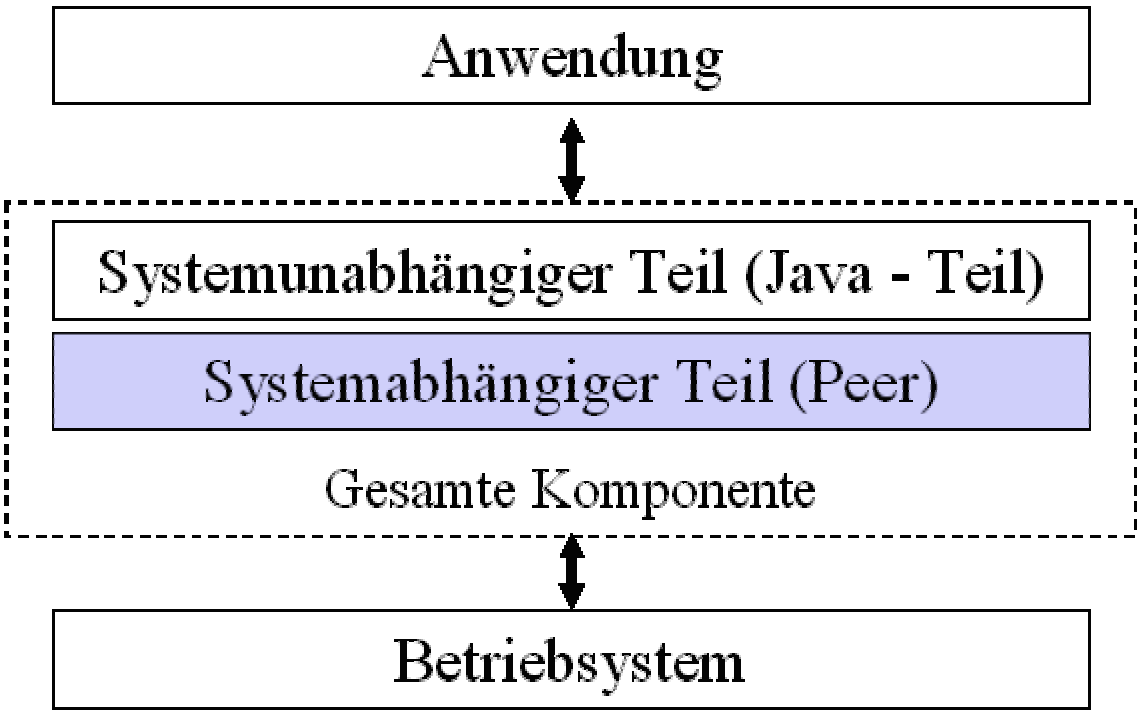
3.1 Was ist AWT

AWT steht für Abstract Window Toolkit. Das AWT stellt eine große Anzahl von

Klassen zur Verfügung, um grafische Oberflächen zu gestalten und auf Ereignisse zu reagieren. Entscheidend ist, dass die Programmierung des AWT aus Sicht des Anwenders plattformunabhängig ist. Die Darstellung der Objekte hängt aber stark von darunter liegenden systemabhängigen Komponenten ab.

3.2 Peerkonzept

Wie oben dargestellt sind die Klassen des AWT systemunabhängig. Damit die einzelnen grafischen Objekte wirklich auf dem Bildschirm erscheinen, muss auf die Zeichenfunktionen der grafischen Oberfläche des Betriebsystems zurückgegriffen werden. Die Anbindung des AWT bilden die sogenannten Peers. In den Peerklassen wird der betriebssystemabhängige Teil einer AWT – Komponente implementiert.



3.3 Wichtige Klassen

3.3.1 JXToolkit

Mit Hilfe dieser Klasse werden die grafischen JX – PeerObjekte erzeugt, die betriebssystemseitig die die AWT – Komponenten darstellen.

3.3.2 GeneralConnector und JXWindowConnector

Die Klasse GeneralConnector bildet die Verbindung zum Windowmanager. Sie ist von WWindow abgeleitet und kann somit auf die Zeichenmethoden des Windowmanagers zugreifen.
JXWindowConnector ist wiederum von GeneralConnector abgeleitet. Diese Klasse ist vor allem wichtig für die Verarbeitung von Events, da sie die entsprechenden Methoden von WWindow überschreibt. Mit Hilfe der Klasse JXComponentPeer werden die Events dann an die EventQueue übergeben.

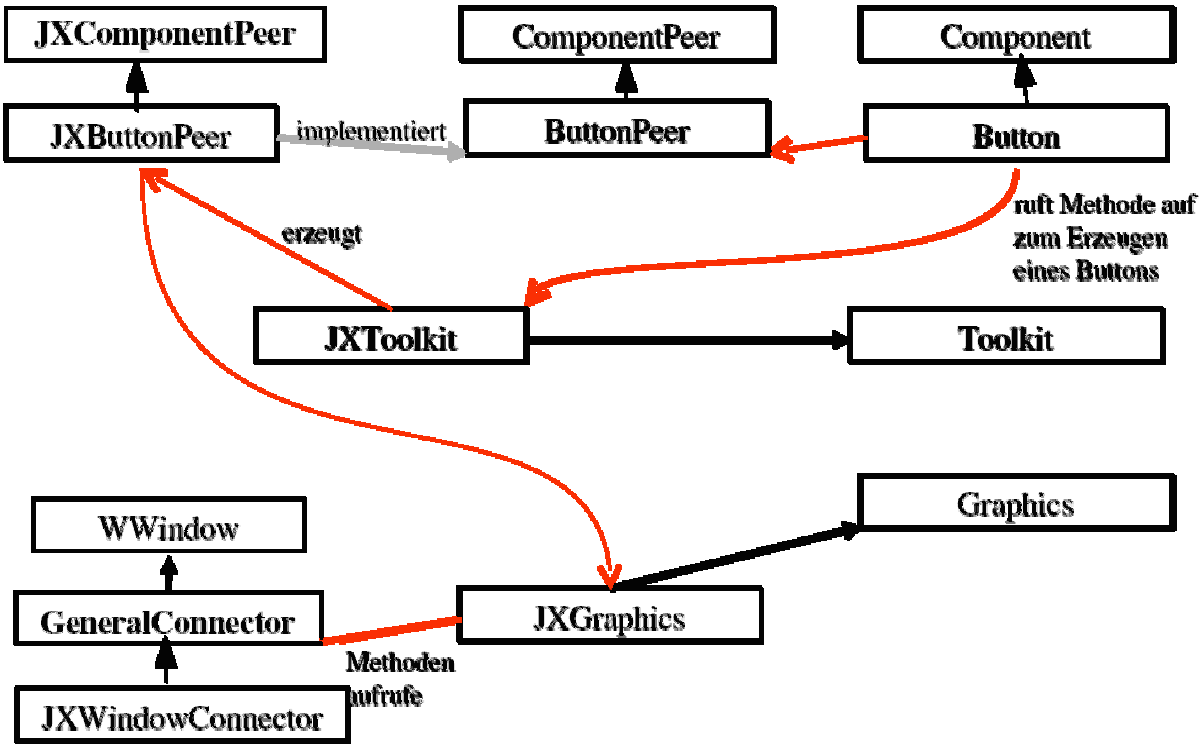
3.3.3 JXGraphics

Diese Klasse ist elementar für alle Zeichenoperationen. Sie ist von java.awt.Graphics abgeleitet und bildet somit die Schnittstelle zwischen Zeichenoperationen in dem AWT und dem JX – Windowmanager. Über ein GeneralConnector – Objekt hat sie Anbindung zum Windowmanager.

3.3.4 EventQueue

Diese Klasse ist das Herzstück der AWT – Event – Verarbeitung. Ein eigener EventDispatchThread arbeitet nach und nach die eintreffenden Events ab, indem er dispatchEvent von Component oder MenuComponent aufruft.

3.4 Übersicht über die Windowmangeranbindung am Beispiel der Klasse Button



4. Welche Änderungen wurden von uns vorgenommen

4.1 Moderneres optisches Auftreten

Der Windomanger ist für die Gestaltung der Fenster verantwortlich. Die Aufgabe bestand darin das Design ein wenig "moderner" zu gestalten. In wie weit so etwas dann wirklich gelungen ist, ist sicherlich eine Geschmacksfrage. Um dieses Ziel umsetzen zu können, erweiterten wir zunächst die Klasse PixelColor um Funktionen zur Berechnung von Transparenz, Helligkeitsstufen und Überdeckung von Farben. Mit Hilfe dieser neuen Funktionalitäten war es uns nun möglich die Titelleiste des Fensters mit einem

Farbverlauf zu hinterlegen.
Zum Zeichnen der Schaltflächen wurde die Klasse WBitmapRGB32MemoryComfortable geschaffen, die von WBitmapRGB32Memory abgeleitet ist und einige Methoden zum Zeichnen von Farbverläufen verschiedener Art bietet. Die Buttons werden einmal beim Erzeugen des ersten Fensters gezeichnet und dann als statische Bitmaps "gespeichert". Der Vorteil, ist das beim Neuzeichen des Titelleiste, was zum Beispiel beim Verschieben des Fensters nötig ist, nur der Speicher der Bitmap des jeweiligen Buttons kopiert werden muss.
Im Rahmen dieser Neuerungen haben wir auch versucht die Klassen, die für das Zeichnen der Fenster zuständig sind, sinnvoll in einzelne Klassen für die jeweiligen Bereiche zu splitten. Wir haben die Klasse AmigaDecorator3 neu hinzugefügt und als standardmäßigen WindowDecorator festgelegt. Sie implementiert das Interface WindowDecorator. Das Gestalten des Fensters haben wir in zwei Bereiche aufgeteilt, den das Fenster umgebenden Rahmen und die Titelleiste. Für ersteres ist die abstrakte Klasse BorderDecorator zuständig. Die Titelleiste wird mit Hilfe der abstrakten Klasse TopDecorator gezeichnet. Die abstrakten Klassen liefern jeweils Grundfunktionen, die bei verschiedenen Arten von Fenstergestaltungen nützlich sind. Zum Beispiel implementieren sie die methode hitTest, die bei übergebener Mausposition die Konstante zurückgibt, die der aktuellen Mausposition entspricht.
Die genauere optische Gestaltung sollen jeweils von diesen Klassen abgeleitete Klassen übernehmen. Die aktuell verwendete Implementierung bilden die Klassen TopDecoratorImpl1 und BorderDecoratorImpl1.

4.2 Mauszeiger

Unsere Änderung besteht darin, dass Mauszeiger direkt aus einer ppm – Datei eingelsen werden können. Es werden beim Start des Windowmanagers die verschiedenen Mauszeiger, z.B. Standardmauszeiger oder die Zeiger zur Änderung der Fenstergröße, in Bitmaps eingelesen. Falls die Maus z.B. auf eine Resize - Kontrolle des Fensters zeigt, wird das Bitmap des aktuellen Mauszeigers auf die Bitmap des jeweiligen Mauszeigers gesetzt und somit dann der andere Mauszeiger angezeigt.
Ein weiterer Performancegewinn könnte durch Abspeicherung des Mauszeigers als dünn besetzte Matrix realisiert werden.

4.3 Verbesserung der Performance des Desktophintergrundes

In der ursprünglichen AWT – Testversion war der Desktophintergrund ein eigener AWT - Frame der ein Canvas zeichnete. Beim Verschieben der Fenster wurde der Hintergrund häufig nur mit großen Verzögerungen nachgezeichnet. Da ja erst die entsprechende paint – Methode in der AWT – Implementierung aufgerufen wurde, was nur unter Zuhilfenahme zweier Nachrichtenschlangen und zahlreicher Methodenaufrufe möglich ist.
Wir haben den Desktophintergrund direkt in der Klasse WindowManagerImpl als ein statisches WBitmap – Objekt implementiert. Im Falle des Freiwerdens eines Teiles des Hintergrundbildes wird nur dieser Bereich direkt aus der Bitmap ausgelesen und gezeichnet, d.h. es wird tatsächlich nur Speicher kopiert.

4.4 Performancegewinn durch Reduzierung des Messageaufkommens

Der Aufwand einer Kommunikation zwischen dem WindowManger und der AWT – Implementierung ist relativ groß. Z.B. wurde bei jeder Bewegung der Maus eine Message an die AWT geschickt, die dort verarbeitet werden muss. Die Performance des Gesamtsystems wurde dadurch nicht unerheblich beeinflusst. Daher wurde versucht die Anzahl dieser Messages zu reduzieren. Es werden nun nur noch Nachrichten an die AWT gesendet, wenn sich die Maus über der Arbeitsfläche des aktiven Fensters befindet.

5. Anregungen für weitere Verbesserungen

Die Anzahl der Messages könnte weiter reduziert werden, wenn beispielsweise Nachrichten nur an Anwendungen geschickt werden die einen Listener für das jeweilige Ereignis installiert haben.

Wenn die Views selbst ihr aktuelles Bild speichern würden, müsste beim Verschieben oder Überdecken von Fenstern nur der freigewordene Bereich aus dem Speicher der View in den entsprechenden Speicherbereich der Grafikkarte kopiert werden und nicht eine WPaintMessage an die AWT geschickt werden. Da diese Änderung einen erhöhten Speicheraufwand und Performanceverlust bei Größenänderung mit sich bringt, sollte dem Anwender die Wahl der View mit und der ohne Speicher überlassen werden.

Es scheinen nicht ganz unerhebliche Probleme bei der Abarbeitung der AWT Events zu bestehen. Die Events werden häufig nur mit großer Verzögerung ausgeführt.