

Prüfer: Prof. Dr. Kurt Rothermel  
Betreuer: Dipl. Inform. Hartmut Benz  
Beginn am: 03.08.1998  
Beendet am: 01.02.1999  
CR-Nummer: H.5.2

Diplomarbeit Nr. 1695

**Design und Implementation  
eines Recorder/Viewers  
für Java-AWT**

Jörg Schließer

## Inhaltsverzeichnis

Inhaltsverzeichnis .....	1
Abbildungsverzeichnis .....	3
Zusammenfassung .....	4
1. Einleitung .....	5
1.1 Motivation .....	5
1.2 Anwendungsbeispiele für die Anwendungsaufzeichnung .....	6
1.2.1 DIANE - Multimedia Annotationen .....	6
1.2.2 Aufzeichnung von verteilten, computerunterstützten Arbeitsumgebungen .....	7
1.2.3 Microsoft Word 97 Makroaufzeichnung .....	7
1.3 Aufzeichnung von Anwendungen über die grafische Benutzungsschnittstelle .....	8
1.3.1 Die Aufzeichnung der grafischen Benutzungsschnittstelle als Teilbereich der Anwendungsaufzeichnung im Allgemeinen .....	8
1.3.2 Variablen der Aufzeichnung der grafischen Benutzungsschnittstelle .....	9
1.3.3 Architekturentwurf für ein Toolkit zur Aufzeichnung und Wiedergabe einer grafischen Benutzungsschnittstelle .....	14
2. Die grafische Benutzungsschnittstelle .....	19
2.1 Definition einiger grundlegender Begriffe .....	19
2.1.1 Schnittstelle, Benutzungsschnittstelle .....	19
2.1.2 Grafische Benutzungsschnittstelle .....	19
2.1.3 Fenster (Windows), Komponenten (Components), Bedienelemente etc. ....	20
2.2 Klassifikation der Elemente einer grafischen Benutzungsschnittstelle .....	21
2.2.1 Die Perspektive des Betrachters .....	21
2.2.2 Die Herkunft eines Bedienelements .....	24
2.2.3 Funktionsbereiche von Elementen grafischer Benutzungsschnittstellen .....	29
2.3 Beispiele für grafische Benutzungsschnittstellen .....	31
2.3.1 Das X Window-System .....	31
2.3.2 Microsoft Windows .....	36
2.3.3 Java, das Java-AWT und die Java Foundation Classes .....	41
2.4 Standardelemente grafischer Benutzungsschnittstellen .....	46
2.4.1 Gebräuchliche Bedienelemente im Überblick .....	46
2.4.2 Detaillierte Betrachtung der Interaktion mit einem Bedienelement .....	49
3. Design eines Recorder/Viewers für das Java-AWT .....	51
3.1 Begriffe und Schreibweisen .....	51
3.2 Die Funktionalität des Java Recorder/Viewer-Systems .....	53
3.2.1 Anwendungsbereiche .....	53
3.2.2 Aufzeichnung zur optischen Wiedergabe .....	53
3.2.3 Aufzeichnung zur wiederholten Ausführung .....	54
3.3 Die Paketarchitektur des JRV-Systems .....	54
3.4 Aufzeichnen mit dem JRVRecorder .....	55
3.4.1 Die Klassen des JRVRecorder-Pakets .....	55
3.4.2 Das Zusammenspiel der Komponenten des JRVRecorder-Pakets .....	57
3.4.3 Die Initialisierung des Aufzeichnungssystems .....	58
3.4.4 Die Initialisierung einer Aufzeichnung .....	59

3.4.5 Das Aufzeichnen von Ereignissen.....	61
3.5 Wiedergabe mit dem JRVPlayer.....	64
3.5.1 Die Klassen des JRVPlayer-Pakets.....	64
3.5.2 Das Zusammenspiel der Komponenten des JRVPlayer-Pakets.....	66
3.5.3 Die Initialisierung des Wiedergabesystems.....	66
3.5.4 Die Wiedergabe mit dem Runtime-Player.....	68
3.5.5 Die Wiedergabe durch einen anwendungsspezifischen Player.....	70
3.6 Datenorganisation mit dem JRVDDataManager.....	72
3.6.1 Die Klassen des JRVDDataManager-Pakets.....	72
3.6.2 Das Zusammenspiel der Komponenten des JRVDDataManager-Pakets.....	74
3.7 Hilfsklassen und Konstanten im JRVEvents-Paket.....	74
4. Die Implementierung des JRV-Systems.....	75
4.1 Entwicklungs- und Systemumgebung.....	75
4.1.1 Das Java Development Kit.....	75
4.1.2 Die JBuilder 2 Entwicklungsumgebung von Inprise.....	75
4.1.3 Der Mauszeiger als betriebssystemabhängige Komponente.....	76
4.2 Implementierung der RecordSource- und PlayTargetManager.....	76
4.2.1 Die Klassenhierarchien der RecordSource- und PlayTargetManager.....	77
4.2.2 Die Serialisierung von Objekten.....	77
4.2.3 Ermitteln, Sichern und Wiederherstellen des Zustands von Komponenten.....	78
4.2.4 Grenzen der Ereignisaufzeichnung durch Listener.....	83
4.2.5 Low-Level-Ereignisse und High-Level-Ereignisse.....	85
5. Das JRV-System im Einsatz.....	87
5.1 Komponententests.....	87
5.1.1 Ausführen der Komponententests.....	87
5.1.2 SwingTest.....	88
5.1.3 MDITest.....	89
5.1.4 AWTTest.....	90
5.1.5 Der Aufbau der Testprogramme.....	90
5.2 Der Runtime-Player.....	91
5.3 Die DrawLine-Beispielanwendung.....	92
6. Erweiterungen, Kritik und Fazit.....	94
6.1 Ansatzpunkte für Erweiterungen des JRV-Systems.....	94
6.1.1 Unterstützung weiterer Bedienelemente.....	94
6.1.2 Zustandsermittlung statt Objektserialisierung.....	95
6.1.3 Zusätzliche Funktionalität bei der Wiedergabe und ein Editor.....	95
6.2 Kritische Anmerkungen.....	96
6.2.1 JDK 1.1 oder JDK 1.2?.....	96
6.2.2 Existieren noch Fehler in den AWT- und JFC/Swing-Bibliotheken?.....	97
6.2.3 Der Einsatz des JRV-Systems in einer „echten“ Anwendung.....	97
6.3 Fazit.....	98
Literaturverzeichnis.....	99
Erklärung.....	101

## Abbildungsverzeichnis

Abbildung 1: Verarbeitungsmöglichkeiten der Aufzeichnung von Benutzungsschnittstellen .....	10
Abbildung 2: Architektur eines Systems zur Aufzeichnung und Wiedergabe der Benutzungsschnittstelle .....	14
Abbildung 3: Datenfluß bei einem vom Anwendungsprogramm unabhängigen Recorder.....	15
Abbildung 4: Datenfluß bei einem in das Anwendungsprogramm integrierten Recorder .....	16
Abbildung 5: Datenfluß bei Verwendung eines Runtime-Moduls als Player .....	17
Abbildung 6: Datenfluß bei Verwendung eines vom Anwendungsprogramm unabhängigen Players .....	18
Abbildung 7: Datenfluß bei einem in das Anwendungsprogramm integrierten Player .....	18
Abbildung 8: Ein einfaches Dialogfeld .....	21
Abbildung 9: Einige Standardelemente von Benutzungsschnittstellen .....	24
Abbildung 10: Ein Aufzeichnungskonzept für Standardelemente .....	25
Abbildung 11: Ein Aufzeichnungskonzept für individuell entwickelte Bedienelemente .....	26
Abbildung 12: Ein Microsoft Calendar Control als Beispiel für ein Component.....	27
Abbildung 13: Das Calendar Control im Microsoft Spy++ .....	27
Abbildung 14: Ein Aufzeichnungskonzept für Components .....	28
Abbildung 15: Das Grundkonzept des X Window-Systems (Quelle: Figure 1-3, S.9, [NY95]).....	32
Abbildung 16: Die Softwarehierarchie unter X (Quelle: Figure 1-5, S.8, [NO93]) .....	33
Abbildung 17: Das Grundgerüst eines systemnahen X-Programms.....	35
Abbildung 18: Die Integration verschiedener Dienste im Betriebssystem Microsoft Windows.....	37
Abbildung 19: Das Grundgerüst einer fensterbasierten Anwendung unter Microsoft Windows .....	39
Abbildung 20: Die Klassenhierarchie der Java-Komponenten .....	43
Abbildung 21: Das Grundgerüst einer fensterbasierten Java-Anwendung .....	44
Abbildung 22: Interaktion mit einem Kontrollkästchen. ....	49
Abbildung 23: Unterteilung des JRV-Systems in Pakete .....	54
Abbildung 24: Die Klassen des JRVRecorder-Pakets.....	56
Abbildung 25: Zusammenarbeit der Komponenten des JRVRecorder-Pakets.....	57
Abbildung 26: Initialisierung des Aufzeichnungssystems .....	58
Abbildung 27: Initialisierung einer Aufzeichnung.....	60
Abbildung 28: Aufzeichnen von Ereignissen .....	63
Abbildung 29: Klassen des JRVPlayer-Pakets.....	64
Abbildung 30: Das Zusammenspiel der Wiedergabekomponenten mit einem Anwendungsprogramm.....	66
Abbildung 31: Initialisierung des Wiedergabesystems.....	67
Abbildung 32: Wiedergabe mit dem Runtime-Player .....	69
Abbildung 33: Wiedergabe mit einem anwendungsspezifischen Player .....	71
Abbildung 34: Datentypen und Klassen des JRVDatamanager-Pakets.....	72
Abbildung 35: Das Zusammenspiel der Komponenten des JRVDatamanager-Pakets.....	74
Abbildung 36: Ein Label mit einem Rahmen .....	79
Abbildung 37: Der Aufbau eines Borderobjektes .....	80
Abbildung 38: Eine java.awt.List-Komponente.....	83
Abbildung 39: Die Klassenhierarchie der AWT-Komponenten .....	84
Abbildung 40: Das Dialogfeld zum Start der Komponententests .....	87
Abbildung 41: Das Record-Panel-Dialogfeld zur Steuerung des Aufzeichnungsprozesses.....	87
Abbildung 42: Die Dialogseiten der SwingTest-Anwendung.....	88
Abbildung 43: Das Desktop-Rahmenfenster der MDITest-Anwendung .....	89
Abbildung 44: Das AWTTTest-Fenster.....	90
Abbildung 45: Das Dialogfeld des Runtime-Players.....	91
Abbildung 46: Das Play-Panel-Dialogfeld zur Steuerung des Wiedergabeprozesses.....	92
Abbildung 47: Die DrawLine-Beispielanwendung .....	93

## Zusammenfassung

Anwendungsbereiche für die Aufzeichnung und Wiedergabe von Vorgängen, die mit Hilfe von Computerprogrammen abgewickelt werden, sind bspw. die Erzeugung von annotierten Multimediadokumenten, von Präsentationen oder von online Tutorials sowie die Erstellung von Makros oder die Wartung von Programmen auf entfernten Rechnern. Abhängig von der Art der Anwendung kann ein mehr oder weniger großer Teil des Zustandes sowie des Ablaufs eines Vorgangs bereits allein durch die Information repräsentiert werden, die in den Elementen der grafischen Benutzungsschnittstelle des Anwendungsprogramms enthalten ist.

Im Rahmen dieser Arbeit werden zunächst grafische Benutzungsschnittstellen allgemein betrachtet und es wird untersucht, welche Informationen in den einzelnen Komponenten (z.B. Fenstern, Schaltflächen, Textfeldern, Listenfeldern usw.) der Benutzungsschnittstelle enthalten sein können und ob bzw. wie sich diese Informationen generisch, d.h. unabhängig von einer konkreten Anwendung, einer bestimmten Benutzungsschnittstelle und dem verwendeten Betriebssystem, aufzeichnen und wiedergeben lassen.

Im Hauptteil der Arbeit wird ein Recorder/Viewer, d.h. ein Aufzeichnungs- und Wiedergabesystem für Komponenten des Java-AWT (Abstract Windowing Toolkit) und der JFC/Swing-Bibliothek entworfen und prototypisch in der Programmiersprache Java implementiert. Der Recorder/Viewer, der in Form einer Bibliothek in Anwendungsprogramme integriert werden kann, ermöglicht es, die Benutzungsschnittstelle von und die Interaktion des Benutzers mit Java-Programmen, die entsprechende Komponenten verwenden, aufzuzeichnen, zu speichern und wiederzugeben.

Der als JRV-System bezeichnete Recorder/Viewer wird mit Hilfe von Beispielprogrammen getestet und das zugrunde liegende Konzept anhand dieser Tests bewertet. Den Abschluß der Arbeit bildet eine Kritik der erzielten Ergebnisse sowie ein kurzer Ausblick auf denkbare Verbesserungen oder Erweiterungen.

# 1. Einleitung

## 1.1 Motivation

Aktuelle Entwicklungen bei den Anwendungen und Einsatzbereichen von Informationstechnologie werden nicht zuletzt durch Verteilung und Vernetzung, anhaltende Trends der letzten Jahre, beeinflusst oder bestimmt. Mit Hilfe von Internet- und Intranettechnologie kann heute nahezu jeder Computerarbeitsplatz in verteilte Umgebungen oder Prozesse eingebunden werden. Gängige Schlagworte im Umfeld dieser Entwicklung sind bspw.

- „CSCW“ (Computer Supported Collaborative Work), computerunterstützte Team- oder Gruppenarbeit.
- „Workflow Management“, computergesteuerte Verarbeitung von mehrstufigen Standardvorgängen durch Sachbearbeiter.
- „Information on Demand“, bedarfsgerechte, elektronische Informationsbeschaffung.

Ein anderer langfristiger und anhaltender Trend ist der Einsatz von Multimediatechnologie. Wobei dieser Begriff, bedingt durch die Verwendung als werbewirksames Schlagwort, in der Umgangssprache beinahe regelmäßig für beliebige grafische („bunte“) Darstellungen auf Bildschirmen verwendet wird. Eine gängige formale Definition für den Begriff „Multimedia“ besagt dagegen nach [SN95] S.14ff., daß im Rahmen einer Multimediadarstellung mehrere Medien zum Einsatz kommen müssen, wovon wenigstens eines zeitabhängig ist.<sup>1</sup>

Im Zuge dieser technischen Entwicklungen wird nicht nur im gewerblichen Bereich, wie die oben genannten Schlagworte zeigen, sondern auch im privaten Bereich eine immer größere Zahl von Vorgängen computerunterstützt und am Bildschirm abgewickelt. Einsatzbereiche sind hier z.B. online Shopping, Home Banking, die private Korrespondenz oder die Freizeitgestaltung

Wann immer Abläufe am Bildschirm stattfinden, können jedoch auch diese Abläufe selbst zum Gegenstand von Kommunikation und Informationsaustausch werden. Sei es im Rahmen des Benutzersupports, für die Dokumentation oder bei der automatischen Ausführung von Vorgängen, in allen diesen Fällen sind Anwendungen für einen Mechanismus vorstellbar, der es ermöglicht, den Ablauf eines Vorgangs am Bildschirm aufzeichnen, speichern, und wie bspw. ein Textdokument oder eine Grafik, weiter- und wiedergeben zu können. Ein solcher Mechanismus: die Aufzeichnung und Wiedergebe der grafischen Benutzungsschnittstelle von Anwendungsprogrammen ist der Gegenstand dieser Arbeit. Im Folgenden werden einige konkrete Beispiele für bereits existierende Anwendungen und laufende Entwicklungen in diesem Bereich kurz vorgestellt.

---

<sup>1</sup> Anmerkung: Nach dieser Definition ist ein Textdokument, das Grafiken enthält, kein Multimediadokument. Es werden zwar mehrere Medien, hier Text und Grafik, verwendet, jedoch existiert bei der Darstellung des Dokuments keine zeitliche Komponente. Eine Präsentation, die Text und Grafik sowie Animationen und Audioclips enthält, ist dagegen ein Multimediadokument, da sowohl die Wiedergabe der Animationen als auch der Audiodaten ein zeitabhängiger Vorgang ist.

## 1.2 Anwendungsbeispiele für die Anwendungsaufzeichnung

Die in diesem Kapitel genannten Beispiele für die Anwendungsaufzeichnung verdeutlichen die Bandbreite möglicher Einsatzbereiche dieses Konzeptes. Das erste Beispiel betrifft den Einsatz bei der Erzeugung eines neuen Typs von Multimediadokumenten, für den vielfältige Anwendungen vorstellbar sind: Präsentationen, Tutorials, remote Support. Bei dem zweiten Beispiel liegt die Betonung auf dem Einsatz im CSCW Umfeld. In beiden Fällen sind grundsätzlich vielfältige und plattformunabhängige Anwendungen sowie eine weitere Entwicklung vorgesehen. Das dritte Beispiel zeigt die Verwendung der Anwendungsaufzeichnung in dem engeren Rahmen eines spezifischen Anwendungsprogramms.

Nachdem so die Motivation für den Einsatz von Anwendungsaufzeichnung im Allgemeinen gezeigt wurde, wird in den folgenden Kapiteln näher auf die Aspekte eingegangen, welche die Aufzeichnung der Benutzungsschnittstelle im Besonderen betreffen. Daraus können sowohl notwendige Anforderungen als auch wünschenswerte Eigenschaften für Werkzeuge für die Unterstützung der Aufzeichnung und Wiedergabe einer grafischen Benutzungsschnittstelle abgeleitet werden.

### 1.2.1 DIANE - Multimedia Annotationen

DIANE (Design, Implementation and Operation of a Distributed Annotation Environment) ist ein von der Europäischen Union finanziertes Projekt, das von der Firma KAPSCH AG, Vienna [DIANE] und dem IPVR, dem Institut für parallele und verteilte Höchstleistungsrechner der Universität Stuttgart in Kooperation mit dem European Centre for Parallel Computing at Vienna, der Firma Systemas y Tratamiento de Informacion SA und dem Hospital General de Manresa in Spanien sowie der Firma Silogic S.A. in Frankreich durchgeführt wird.

Im Rahmen des DIANE Projektes wird ein System entwickelt, das es ermöglicht annotierte Multimediadokumente zu erzeugen, zu bearbeiten, in einer verteilten Umgebung zu nutzen und wiederzugeben. Ein solches annotiertes Multimediadokument entsteht aus dem Annotat, bspw. der Bildschirmausgabe einer Anwendung, die mit einem als AOR (Application Output Recording) bezeichneten Verfahren aufgezeichnet wird, und multimedialen Annotationen, z.B. Zeigeoperation mit der Maus, Audioclips etc., die dem Annotat hinzugefügt werden. Der Annotationsprozeß kann dabei rekursiv fortgesetzt werden. D.h. der Empfänger eines annotierten Multimediadokuments kann dem Dokument weitere Annotationen hinzufügen.

Die in [BHB+97] beschriebene Systemarchitektur sieht vor, daß beliebige Anwendungen aufgezeichnet werden können. Das System arbeitet Client-Server basiert, und stellt Mechanismen für die Synchronisation der multimedialen Ausgabeströme bei der Wiedergabe sowie für die Beachtung der sicherheitsrelevanten Erfordernisse in einer verteilten Umgebung zur Verfügung. In einer prototypischen Realisierung wird die Aufzeichnung der visuellen Ausgabe von beliebigen Anwendungen durch spezielle Screen-Grabbing-Verfahren ermöglicht. Die Realisierung erfolgt weitgehend (d.h. abgesehen von systemnahen Komponenten für die Medienwiedergabe) plattformunabhängig in Java, konkret sowohl für Solaris 2.5 auf Sun Sparc Rechnern als auch für Microsoft Windows NT auf INTEL PCs [BFM97].

## 1.2.2 Aufzeichnung von verteilten, computerunterstützten Arbeitsumgebungen

In [MP96] wird ein am Department of Electrical Engineering and Computer Science der University of Michigan entwickeltes System beschrieben, das es ermöglicht, Sitzungen in einer computerunterstützten Arbeitsumgebung aufzuzeichnen, zu bearbeiten und wiederzugeben.

Der Gegenstand des Projektes ist es, Anwendungssitzungen, an denen in einer bestimmten Arbeitsumgebung (bisher) nur eine synchrone Teilnahme möglich war, asynchron nutzen zu können. „Synchron“ bedeutet in diesem Zusammenhang, daß alle Interessierten bei der Anwendungssitzung anwesend sein mußten, persönlich oder mit Hilfe einer CSCW Umgebung, wenn sie nicht nur am Ergebnis, z.B. Dokumenten, die während der Sitzung erzeugt wurden, interessiert waren, sondern auch am Entstehungsprozeß des Ergebnisses.

Die in [MP96] vorgestellte Architektur, dort als Replayable Workspace bezeichnet, erlaubt es, Anwendungssitzungen in einer computerunterstützten Arbeitsumgebung aufzuzeichnen und in Form von sog. Session Objects (Sitzungsobjekten) zu speichern. Die Replayable Workspace wird dabei durch drei Dimensionen charakterisiert: die Verarbeitung, die Präsentation und die Kommentierung von Vorgängen.

Das System stellt Kontroll- und Synchronisationsdienste zur Verfügung, die für die Aufzeichnung und Wiedergabe mit den einzelnen Anwendungen (Tools) in der jeweiligen Dimension integriert werden. In die Arbeitsumgebung direkt integrierte Komponenten stellen Interfaces für die Aufzeichnung, die Bearbeitung und Wiedergabe von Sitzungsobjekten zur Verfügung. Eine prototypische Realisierung des Konzeptes erfolgte mit einer Anwendung zur Diagnose von Röntgenbildern in C unter Sun Solaris 2.4.

## 1.2.3 Microsoft Word 97 Makroaufzeichnung

Innerhalb des Textverarbeitungssystems Microsoft Word 97 wird ein Makrokonzept realisiert, das es erlaubt, Operationen beim Bearbeiten von Texten aufzuzeichnen, als Makroprogramm zu bearbeiten und wiederzugeben.

Mit Hilfe eines integrierten Makrorecorders können Aktionen aufgezeichnet werden, die über die Benutzungsschnittstelle, z.B. den Texteingabebereich, Tastaturkommandos, Schaltflächen, Menüs oder Dialogfelder des Programms ausgeführt werden. Diese Makros werden als Quelltext für sog. VBA-Programme (Visual Basic for Applications) gespeichert und können anschließend in dieser Programmiersprache, die eine Untermenge von Microsoft Visual Basic darstellt, editiert und erweitert werden.

Die Anwendungsaufzeichnung mit Hilfe des Makrorecorders bietet reinen Benutzern des Textverarbeitungssystems die Möglichkeit, immer wiederkehrende Arbeitsabläufe einfach und intuitiv aufzuzeichnen. Die Makroprogramme selbst können ihrerseits Schaltflächen oder Menüpunkten zugewiesen und damit in die Benutzungsschnittstelle integriert werden.

Für den Anwendungsentwickler eignet sich die Anwendungsaufzeichnung außerdem dazu, den Grundstock oder Teile für komplexe Programme zu erzeugen, die dann im Rahmen der sehr weitgehenden Programmierbarkeit der Microsoft Office Produkte zu integrierten Anwendungen ausgebaut werden können. Die dabei zum Einsatz kommenden Konzepte sind das von der Firma Microsoft verwendete COM (Component Object Model) und die darauf aufbauende ActiveX Technologie.

## **1.3 Aufzeichnung von Anwendungen über die grafische Benutzungsschnittstelle**

### **1.3.1 Die Aufzeichnung der grafischen Benutzungsschnittstelle als Teilbereich der Anwendungsaufzeichnung im Allgemeinen**

Gegenstand dieser Arbeit ist das Design und die prototypische Implementation eines Systems zur Aufzeichnung und Wiedergabe der grafischen Benutzungsschnittstelle von Programmen, die das Java-AWT bzw. die JFC/Swing-Komponenten (Java Foundation Classes) nutzen. Ein solches System, das die Möglichkeit bietet, Anwendungen bzw. Anwendungssitzungen über die grafische Benutzungsschnittstelle aufzeichnen, solche Aufzeichnungen abspeichern, weiter- und wiedergeben zu können, kann als Hilfsmittel für die Realisierung von Anwendungen, wie sie im vorangegangenen Kapitel beschrieben wurden, eingesetzt werden.

Abhängig von dem angestrebten Einsatzgebiet und den technischen Gegebenheiten (Hardware, Software, Betriebssystem, Anwendungen, die aufgezeichnet werden sollen, etc.) sind unterschiedliche Anforderungen an die Aufzeichnung der grafischen Benutzungsschnittstelle denkbar. Ein Ziel dieser Arbeit ist es, ein generisches Hilfsmittel, im Folgenden als „Toolkit“ bezeichnet, zu entwickeln, d.h. ein System, das für möglichst viele Anwendungsbereiche eingesetzt werden kann.

In [BFMW97] wird eine generische Architektur für die Anwendungsaufzeichnung im Allgemeinen, dort als Application Output Recording (AOR) bezeichnet, vorgestellt. Danach befaßt sich die Anwendungsaufzeichnung damit, Ausgaben von Anwendungen und die Interaktion von Benutzern mit Anwendungen so aufzuzeichnen, daß die Wiedergabe dem ursprünglichen Vorgang ausreichend ähnelt.<sup>2</sup> „Ausreichend“ bedeutet in diesem Zusammenhang, daß die eingesetzten Verfahren am jeweils gewünschten Einsatzbereich gemessen werden müssen. Es wird also eine anwenderorientierte Sicht eingenommen. Fehlende Akzeptanz beim Anwender wird den Einsatz eines Verfahrens vereiteln, wenn im konkreten Fall auf der einen Seite, bei der Anwendungsentwicklung, ein zu hoher Aufwand betrieben werden muß, oder auf der anderen Seite, beim täglichen Einsatz des Systems, eine zu geringe Bandbreite an Einsatz- oder Einflußmöglichkeiten für den Anwender besteht.

Bei der Anwendungsaufzeichnung im Allgemeinen werden mit Bezug auf den jeweiligen Einsatzbereich unterschiedliche Ströme von Ausgabedaten einer Anwendung aufgezeichnet. Dabei kann es sich um Bildschirmausgaben, akustische Ausgaben oder auch um Ausgaben zur Steuerung von Automaten oder Veränderungen in entfernten Systemen, z.B. Datenbanken, handeln. Die Aufzeichnung der grafischen Benutzungsschnittstelle stellt insofern nur einen Unterbereich der Anwendungsaufzeichnung im Allgemeinen dar. Abhängig von dem konkreten Anwendungsbereich kann die Aufzeichnung der Benutzungsschnittstelle bereits allein ausreichend sein. In anderen Fällen kann sie (nur) einen Teilbeitrag leisten.

Ein Toolkit zur Aufzeichnung und Wiedergabe der grafischen Benutzungsschnittstelle sollte so konzipiert sein, daß es in möglichst vielen Anwendungsbereichen eingesetzt werden kann, wobei ein möglichst geringer Aufwand für die Anpassung an eine konkrete Anwendung erforderlich sein sollte. Welche Parameter diese unterschiedlichen Faktoren: Anforderungen, Aufwand und Einsetzbarkeit, beeinflussen, wird im Folgenden detailliert dargestellt.

---

<sup>2</sup> Zitat: „Application Output Recording is the task of recording the output of applications. The output is recorded in such a way that the behaviour of the application and the interaction of the user with it can be replayed at a later time.“ ... „The aim of AOR is to record the output of a given application in such a way that the reproduction of this recording is sufficiently similar to the original output performance.“ [BFMW97]

### 1.3.2 Variablen der Aufzeichnung der grafischen Benutzungsschnittstelle

In [BFMW97] werden verschiedene Variablen für die Anwendungsaufzeichnung im Allgemeinen definiert, die sowohl technische als auch anwendungsbezogene Aspekte betreffen und die Anforderungen und Einsatzmöglichkeiten entsprechender Systeme bestimmen.

Im Folgenden werden verschiedene Ausprägungen dieser Variablen mit besonderem Bezug auf die Aufzeichnung der grafischen Benutzungsschnittstelle betrachtet und daraus allgemeine Anforderungen und mögliche Einsatzbereiche für ein Toolkit zur Aufzeichnung und Wiedergabe der grafischen Benutzungsschnittstelle hergeleitet. Die Klassifikation der Variablen und die verwendete Bezeichnungsweise lehnen sich unmittelbar an [BFMW97] an. Die englischen Bezeichnungen sind diesem Text entnommen.

#### a) Variablen, die das Anwendungspotential bestimmen

Das Anwendungspotential (usability potential) definiert die möglichen Einsatzbereiche der Anwendungsaufzeichnung aus der Sicht des Anwenders und wird durch folgende Variablen bestimmt:

- **Verarbeitungsmöglichkeiten (processing capabilities)**
- **Wiedergabekontrolle (replay controllability)**

#### **Verarbeitungsmöglichkeiten**

Für die Ausprägungen dieser Variable können in zwei Dimensionen je zwei Eckpunkte postuliert werden. Die erste Dimension „Wiedergabe als wiederholte Ausführung“ beschreibt, inwiefern die Wiedergabe eine Wiederholung des Vorgangs bei der Aufzeichnung, inklusive aller Seiteneffekte, ist. Eckpunkte möglicher Ausprägungen sind: 1. Die Wiedergabe kommt einer erneuten Ausführung gleich. 2. Die Wiedergabe dient (ausschließlich) dazu, den Prozeß und die Umstände bei der Aufzeichnung zu visualisieren oder nachvollziehbar zu machen.

Die zweite Dimension „Semantisches Umfeld im Vergleich zur Aufzeichnung“ betrifft das anwendungsspezifische semantische Umfeld, in dem die Wiedergabe stattfindet. Eckpunkte für die Verarbeitungsmöglichkeiten in dieser Dimension sind: 1. Aufgezeichnete Vorgänge können bei der Wiedergabe auf ein anwendungsspezifisch anderes semantisches Umfeld angewandt werden. 2. Das semantische Umfeld bei der Wiedergabe entspricht exakt dem der Aufzeichnung. Abbildung 1 zeigt Beispiele für konkrete Anwendungen, die den hier genannten Eckpunkten des Spektrums der Verarbeitungsmöglichkeiten zugeordnet werden können, und verdeutlicht die verwendeten Begriffe.

#### **Konsequenzen für die Aufzeichnung der Benutzungsschnittstelle**

Abhängig von der konkreten Anwendung können die Anforderungen an die Verarbeitungsmöglichkeiten in beiden Dimensionen beliebig zwischen den genannten Eckpunkten liegen. Ein Toolkit zur Aufzeichnung und Wiedergabe der Benutzungsschnittstelle sollte hier in möglichst vielen Situationen einsetzbar sein. Dem kann durch ein System Rechnung getragen werden, das nicht nur in unterschiedliche Anwendungen integriert werden kann, d.h. tatsächlich den Charakter eines Toolkits besitzt, sondern selbst auf Erweiterbarkeit ausgerichtet ist.

		Semantisches Umfeld im Vergleich zur Aufzeichnung	
		Anderes Umfeld	Identisches Umfeld
<b>Wiedergabe als wiederholte Ausführung</b>	<b>Wiederholte Ausführung</b>	<p>„Makro-Funktionalität“</p> <p>Mit Hilfe des Makrorecorders eines Textverarbeitungssystems können Anweisungen zum Editieren und Formatieren des Textes aufgezeichnet werden.</p> <p>Die Wiedergabe führt dieselben Funktionen auf beliebigen Texten und in verschiedenen Situationen aus, bspw. innerhalb einer Tabelle sowie innerhalb normalen Fließtextes.</p>	<p>„Remote Support“</p> <p>Ein Benutzer möchte einen (vermeintlichen) Fehler in einer Anwendung an den Benutzersupport melden.</p> <p>Er schickt eine Aufzeichnung derjenigen Arbeitsschritte, die zum Fehler führen, sowie das Dokument, bei dessen Bearbeitung der Fehler auftritt, an einen Supportmitarbeiter. Mit Hilfe der Aufzeichnung versucht dieser festzustellen, ob ein System- oder Bedienungsfehler vorliegt.</p>
	<b>Keine wiederholte Ausführung</b>	<p>„Präsentationen“</p> <p>Es wird eine Präsentation eines Anwendungsprogramms für Werbezwecke erstellt. Dazu wird die Anwendung auf einem bestimmten System und in einer bestimmten Situation aufgezeichnet, so daß die Vorzüge der Anwendung gut zur Geltung kommen.</p> <p>Bei der Wiedergabe steht die Anwendung selbst nicht zur Verfügung, sondern eine Art Runtime-Player, z.B. in Form eines Kiosk-Systems, der einen optischen Eindruck der Anwendung vermittelt.</p>	<p>„Dokumentation“</p> <p>Ein Computerspiel bietet die Möglichkeit, einen Spielverlauf aufzuzeichnen.</p> <p>Die Aufzeichnungen können später zur Analyse der Spielstrategie oder zur Dokumentation bspw. im Rahmen einer „Highscore-Tabelle“ wiedergegeben werden.</p>

Abbildung 1: Verarbeitungsmöglichkeiten der Aufzeichnung von Benutzungsschnittstellen

### Wiedergabekontrolle

Unter der Wiedergabekontrolle sind die Möglichkeiten zur unmittelbaren Einflußnahme des Benutzers bei der Wiedergabe einer Aufzeichnung zu verstehen. Denkbare Funktionen, die den Kontrollmöglichkeiten bei Videorecordern oder CD-Spielern nachempfunden sind, wären z.B. Start und Stop der Wiedergabe, Vorwärts- und Rückwärtswiedergabe entlang der Zeitachse, Wiedergabe ab einem bestimmten Punkt, schrittweise Wiedergabe, Wiedergabe mit beschleunigter oder reduzierter Geschwindigkeit. Art und Umfang der Kontrollmöglichkeiten werden im konkreten Anwendungsfall unmittelbar durch die gewünschten oder geforderten Verarbeitungsmöglichkeiten beeinflusst. Die Ausprägungen der Variable „Wiedergabekontrolle“ werden demzufolge, in Abhängigkeit von der Anwendung, für die hier genannten Funktionen zwischen „unabdingbar“ (z.B. Start- und Stop der Wiedergabe) und „unmöglich“ (z.B. Rückwärtswiedergabe oder Sprung zu einer bestimmten Stelle) liegen.

### Konsequenzen für die Aufzeichnung der Benutzungsschnittstelle

Ein Toolkit für die Aufzeichnung und Wiedergabe der grafischen Benutzungsschnittstelle sollte geeignete Mechanismen und Datenformate zur Verfügung stellen, um je nach Bedarf eine weitgehende Kontrolle der Wiedergabe entsprechend den oben genannten Funktionen zu ermöglichen. Dies erfordert insbesondere die Aufzeichnung geeigneter und ausreichender Daten für die verschiedenen Funktionen sowie eine effiziente Verwaltung dieser Daten.

## **b) Variablen, die technische Aspekte betreffen**

Zwischen den folgenden Variablen, die technische Aspekte der Anwendungsaufzeichnung und -wiedergabe bestimmen, bestehen sowohl untereinander als auch im Verhältnis zu den zuvor genannten Variablen des Anwendungspotentials wechselseitige Abhängigkeiten. Es wird für den einzelnen Anwendungsfall detailliert zu untersuchen sein, welche technischen Maßnahmen einerseits notwendig und andererseits sinnvoll sind.

- **Anwendungseinbezug bei der Aufzeichnung (application awareness for recording)**
- **Datenformat der Aufzeichnung (recording data format)**
- **Punkt der Datenaufnahme (point of data interception)**
- **Auswahl der Aufzeichnungsquelle (selection of recording source)**
- **Timing der Aufzeichnung (timing of recording)**

### **Anwendungseinbezug bei der Aufzeichnung**

Für diese Variable existieren zwei Ausprägungen: 1. Die aufgezeichnete Anwendung ist in den Aufzeichnungsprozeß unmittelbar einbezogen. D.h. das Anwendungsprogramm ist „sich bewußt“, daß es aufgezeichnet wird und kann dementsprechend aktiv Daten und Funktionen für die Aufzeichnung zur Verfügung stellen. 2. Die aufgezeichnete Anwendung ist in den Aufzeichnungsprozeß nur mittelbar, d.h. als reines Objekt, einbezogen. Die Anwendung „weiß nicht“, daß sie aufgezeichnet wird.

### **Konsequenzen für die Aufzeichnung der Benutzungsschnittstelle**

Ein wesentliches Argument für die zweite Alternative ist, daß so beliebige Anwendungen aufgezeichnet werden können und sich schnell ein breites Anwendungsspektrum erschließen läßt. Für bestimmte Ausprägungen der zuvor beschriebenen Variable „Verarbeitungsmöglichkeiten“ ist jedoch der unmittelbare Einbezug der aufgezeichneten Anwendung unabdingbar, wenn die Zustandsinformation in Form von Daten, die in den Komponenten der grafischen Benutzungsschnittstelle enthalten sind, von außen nicht zugänglich ist.

Wenn eine Anwendung „unbemerkt“ aufgezeichnet werden soll, so wird sich die Aufzeichnung im Wesentlichen auf Ereignisse beschränken, die durch Aktionen des Benutzers ausgelöst werden sowie auf das rein äußerliche Erscheinungsbild der Oberfläche. Dies zieht jedoch eine Einschränkung der Verarbeitungsmöglichkeiten nach sich, da Zustandsänderungen, die durch das Anwendungsprogramm selbst, ohne unmittelbares Zutun des Benutzers erfolgen, bspw. die Initialisierung von Listen oder Schaltflächen, u.U. nicht so erfaßt werden können, wie dies für die gewünschte Verarbeitungsmöglichkeit notwendig ist.

Im Rahmen dieser Arbeit wird im Allgemeinen vorausgesetzt werden, daß die aufgezeichnete Anwendung den Aufzeichnungsprozeß aktiv unterstützt. Ein Toolkit für die Aufzeichnung der Benutzungsschnittstelle sollte demzufolge dem Anwendungsentwickler Methoden zur Verfügung stellen, die den Aufwand für die Anpassung von Programmen reduzieren. Ein Ziel oder ein pragmatischer Erfolg wäre es, wenn dadurch Anwendungsaufzeichnung auch in Anwendungen integriert würde, für die das Konzept zunächst nur ein „nice-to-have“ Feature ist.

Die Variable „Anwendungseinbezug“ läßt sich unmittelbar von der Aufzeichnung auf die Wiedergabe ausdehnen. Wenn als Verarbeitungsmöglichkeit gefordert wird, daß die Wiedergabe einer erneuten Ausführung gleichkommt, so wird die Unterstützung der Wiedergabe durch die aufgezeichnete Anwendungen in der Regel unabdingbar sein.

Wenn bei der Wiedergabe dagegen lediglich der visuelle Eindruck des Vorgangs bei der Aufzeichnung wiederholt werden soll, ist das zur Verfügung Stellen eines Runtime-Players eine sinnvolle Alternative zur Verwendung der ursprünglichen Anwendung für die Wiedergabe. Aufzeichnungen können dann wie Audio- oder Videoclips genutzt und weitergegeben werden. Ein Toolkit für die Aufzeichnung und Wiedergabe der Benutzungsschnittstelle sollte demzufolge für entsprechende Anwendungsbereiche ein Wiedergabemodul enthalten, das unabhängig von der aufgezeichneten Anwendung genutzt werden kann.<sup>3</sup>

### **Datenformat der Aufzeichnung**

Das Datenformat der Aufzeichnung beschreibt die konkrete Struktur, sowohl Art als auch Umfang der Daten, die aufgezeichnet werden.

### **Konsequenzen für die Aufzeichnung der Benutzungsschnittstelle**

Das Datenformat muß dahingehend mit Bedacht gewählt werden, daß eine generische Unterstützung für möglichst viele Verarbeitungsmöglichkeiten realisiert werden kann.

### **Punkt der Datenaufnahme**

Punkte der Datenaufnahme sind Schnittstellen, an denen Daten für die Aufzeichnung gewonnen werden können.

### **Konsequenzen für die Aufzeichnung der Benutzungsschnittstelle**

Für die Aufzeichnung der grafischen Benutzungsschnittstelle bieten sich für die Datenaufnahme in erster Linie folgende Punkte an: 1. Die Systemschnittstelle als Eingabeschnittstelle, an der Ereignisse, die durch den Benutzer ausgelöst wurden, an das Anwendungsprogramm weitergeleitet werden. Auf dieser Ebene werden Ereignisse in der Regel unabhängig von ihrer anwendungsspezifischen Semantik betrachtet. Im Wesentlichen kommen dabei Tastaturereignisse sowie Mausbewegungen, Mausklicks etc. in Betracht. 2. Das Anwendungsprogramm selbst als Schnittstelle, an der Ereignisse verarbeitet oder auch ausgelöst werden, die Zustandsänderungen der grafischen Benutzungsschnittstelle nach sich ziehen, bspw. die Initialisierung von Bedienelementen, Listen oder Schaltflächen. 3. Die Systemschnittstelle als Ausgabeschnittstelle, an welcher die Darstellung der grafischen Benutzungsschnittstelle erfolgt.

Welche Informationen an diesen Punkten der Datenaufnahme gewonnen werden können und unter welchen Bedingungen die Aufnahme nur an einzelnen oder an allen Punkten sinnvoll oder möglich ist, hängt sowohl von der Systemumgebung, als auch von der Anwendung ab.

---

<sup>3</sup> Anmerkung: Bei den hier gemachten Ausführungen sollte allgemein berücksichtigt werden, daß es sich um „weiche“ Kriterien handelt. „Weich“ bedeutet in diesem Zusammenhang, daß allgemeingültige Aussagen nicht zu machen sind, wenn nur ganz allgemein von „Anwendungen“ die Rede ist. Das folgende, zugegebenermaßen etwas pathologische Beispiel verdeutlicht dies: Man nehme ein Slideshow-Programm, das lediglich eine Reihe von Grafiken anzeigt. Von diesem soll der Ausgabestrom aufgezeichnet werden, mit dem Ziel eine Wiedergabe zu ermöglichen, die einer erneuten Ausführung gleichkommt. In diesem Fall ist weder für die Aufzeichnung noch für die Wiedergabe eine unmittelbare Unterstützung durch die Anwendung selbst nötig. Die Aufzeichnung kann mit einem Screen-Capture-Tool erfolgen, die Wiedergabe mit einem anderen Slideshow-Programm.

Wenn jedoch im Text davon die Rede ist, daß „in der Regel“ bestimmte Eigenschaften einer Anwendung oder der Aufzeichnungs- und Wiedergabekomponenten erforderlich sind, soll damit zum Ausdruck gebracht werden, daß es für einen möglichst breiten Einsatzbereich der genannten Techniken sinnvoll ist, wenn diese Vorgaben erfüllt sind. Das Ziel ist es, generische, d.h. für möglichst viele Anwendungsbereiche geeignete Methoden zu entwickeln, die im Einzelfall das Erreichen des Anwendungsziels erleichtern. Es sei daher zugestanden, daß Anwendungen denkbar sind, die dieser Unterstützung nicht bedürfen und deshalb entsprechende, hier postulierte Voraussetzungen nicht erfüllen müssen.

### **Auswahl der Aufzeichnungsquelle**

Unter der Auswahl der Aufzeichnungsquelle ist das Festlegen der Gegenstände oder Objekte der Aufzeichnung zu verstehen. Denkbare Ausprägungen für diese Variable sind: 1. Die Anwendung bestimmt, ob, wann und welche Vorgänge aufgezeichnet werden, bspw. im Rahmen einer automatischen Dokumentation, zur Kontrolle oder Leistungsmessung.<sup>4</sup> 2. Der Benutzer wählt aktiv den Zeitpunkt und, soweit im konkreten Anwendungsfall vorgesehen, den Gegenstand und die Reichweite der Aufzeichnung aus.

### **Konsequenzen für die Aufzeichnung der Benutzungsschnittstelle**

Ein Toolkit für die Aufzeichnung der grafischen Benutzungsschnittstelle sollte einerseits eine vorgefertigte Benutzungsschnittstelle für die Steuerung des Aufzeichnungsprozesses zur Verfügung stellen, andererseits jedoch auch eine Programmierschnittstelle beinhalten, über die Anwendungen den Prozeß steuern oder eine individuelle Benutzungsschnittstellen realisieren können.

### **Timing der Aufzeichnung**

Das Timing der Aufzeichnung bestimmt die Zeitpunkte, zu denen Datenpakete, vergleichbar etwa sog. „Frames“ (Rahmen) bei der Aufzeichnung von Videos, aufgezeichnet werden. Denkbare Konzepte sind hierbei: 1. Die Aufzeichnung erfolgt kontinuierlich, in festen Zeitabständen. 2. Die Aufzeichnung erfolgt ereignisgesteuert. 3. Die Aufzeichnung erfolgt benutzergesteuert.

### **Konsequenzen für die Aufzeichnung der Benutzungsschnittstelle**

Ein Toolkit für die Aufzeichnung der grafischen Benutzungsschnittstelle sollte alle drei Formen des Timings unterstützen.

## **c) Schlußfolgerungen**

Wie bereits mehrfach angedeutet wurde, stellt sich im Grunde nicht die Frage, welche konkreten Kombinationen von Ausprägungen der hier genannten Variablen berücksichtigt werden müssen, wenn ein generisches Toolkit für die Aufzeichnung und Wiedergabe der grafischen Benutzungsschnittstelle entwickelt werden soll. Wünschenswert ist die Unterstützung möglichst vieler Kombinationen, so daß vielfältige Verarbeitungsmöglichkeiten gegeben sind. Aus der Allgemeinheit der Aufgabenstellung: Aufzeichnung der Benutzungsschnittstelle beliebiger Anwendungen, kann jedoch die Konsequenz gezogen werden, daß das Design eines solchen Systems seine Erweiterbarkeit beinhalten sollte. Es erscheint unwahrscheinlich, daß aus dem Stand heraus ein System realisiert werden kann, daß allen Anforderungen in dem Sinne gerecht wird, daß beliebige Anwendungen mit vertretbarem Aufwand an ein bereits abgeschlossenes System angepaßt werden könnten. Erweiterbarkeit sollte hier also nicht als wünschenswerte Option angesehen, sondern als ausdrückliche Anforderung gestellt werden.

---

<sup>4</sup> Anmerkung In solchen Fällen sind ggf. gesetzliche Vorschriften zum Schutz der Arbeitnehmer zu berücksichtigen, die im BetrVG (Betriebsverfassungsgesetz) oder auch in innerbetrieblichen Vereinbarungen geregelt sind.

### 1.3.3 Architekturentwurf für ein Toolkit zur Aufzeichnung und Wiedergabe einer grafischen Benutzungsschnittstelle

Der folgende Entwurf einer Architektur für ein Toolkit zur Aufzeichnung und Wiedergabe von grafischen Benutzungsschnittstellen hat zunächst eine grobe Unterteilung des Systems in seine wesentlichen Komponenten zum Ziel. Basierend auf den im vorhergehenden Kapitel beschriebenen Variablen der Aufzeichnung der grafischen Benutzungsschnittstelle werden dabei unterschiedliche Alternativen diskutiert.

Aufbauend auf diesem Entwurf werden in den nächsten Kapiteln drei existierende grafische Benutzungsschnittstellen mit Bezug auf die Problematik der Aufzeichnung und Wiedergabe untersucht und anschließend eine Spezifikation und ein Design für die einzelnen Komponenten des hier skizzierten Systems im Detail angefertigt. Schließlich erfolgt eine prototypische Realisierung eines Systems zur Aufzeichnung und Wiedergabe der Benutzungsschnittstelle für solche Anwendungen, die das Java-AWT bzw. die JFC/Swing-Komponenten verwenden.

#### a) Gliederung des Gesamtsystems

Ein Toolkit zur Aufzeichnung und Wiedergabe einer grafischen Benutzungsschnittstelle kann als Gesamtsystem auf der obersten Ebene zunächst in drei Komponenten zerlegt werden: Eine Aufzeichnungskomponente (Recorder), eine optionale Bearbeitungskomponente (Editor) und eine Wiedergabekomponente (Player). Abbildung 2 zeigt verschiedene Alternativen für die Integration dieser Komponenten in die aufgezeichnete Anwendung.

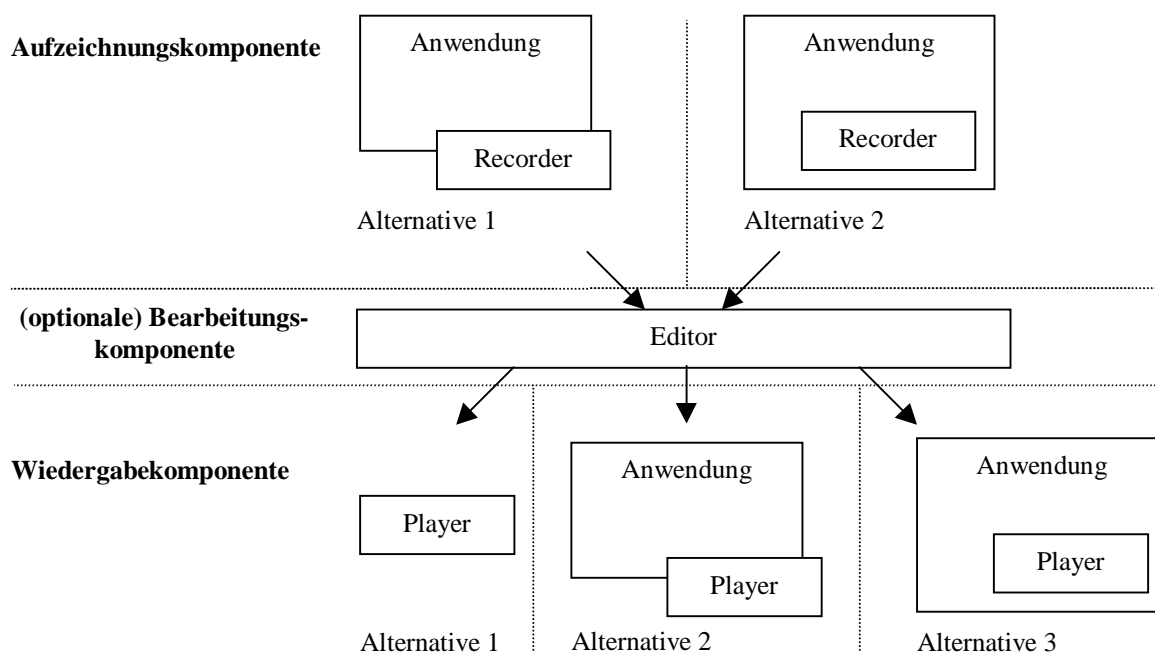


Abbildung 2: Architektur eines Systems zur Aufzeichnung und Wiedergabe der Benutzungsschnittstelle

In den folgenden Abschnitten werden die Funktionen der einzelnen Komponenten sowie Vor- und Nachteile der unterschiedlichen Alternativen für die Integration in das Anwendungsprogramm dargestellt.

## b) Die Aufzeichnungskomponente (Recorder)

Die Aufgabe des Recorders ist es, alle notwendigen Daten für die Aufzeichnung an den ausgewählten Punkten der Datenaufnahme zu erfassen, zu verarbeiten, in das Datenformat des Aufzeichnungs- und Wiedergabesystems zu konvertieren und zu speichern. Darüber hinaus stellt der Recorder ggf. für den Endbenutzer eine Benutzungsschnittstelle für die Auswahl der Aufzeichnungsquelle und die Steuerung und das Timing der Aufzeichnung zur Verfügung. Für die Beziehung zwischen Recorder und aufgezeichneter Anwendung sind zwei Alternativen denkbar: 1. Der Recorder wird in das Anwendungsprogramm integriert. Die Anwendung kann aktiv in den Aufzeichnungsprozeß einbezogen werden. 2. Der Recorder stellt ein vollständig eigenständiges Programm dar. Die Anwendung wird „unbemerkt“ aufgezeichnet. Die folgenden Abbildungen skizzieren den Datenfluß für beide Alternativen.

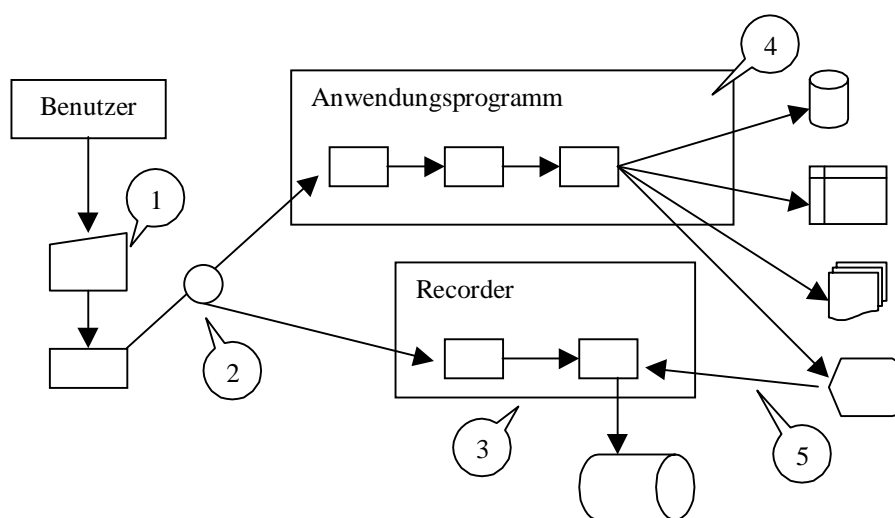


Abbildung 3: Datenfluß bei einem vom Anwendungsprogramm unabhängigen Recorder

Abbildung 3 skizziert den Datenfluß bei der Aufzeichnung mit einem vom Anwendungsprogramm unabhängigen Recorder: (1) Ereignisse, die durch den Benutzer über die grafische Benutzungsschnittstelle ausgelöst werden, z.B. Bewegungen der Maus, das Drücken von Maustasten etc., werden vom Betriebssystem durch gerätenahe Treiber erfaßt, verarbeitet, in abstrakte, von der Semantik des Anwendungsprogramms unabhängige Systemereignisse der Form „Button\_X\_Clicked“, „Mouse\_Moved\_XY“ umgewandelt und an das entsprechende Anwendungsprogramm weitergeleitet. (2) Durch einen sog. „Hook“, einen Mechanismus, den das Betriebssystem zur Verfügung stellt, können solche Systemereignisse überwacht werden.<sup>5</sup> Der Hook stellt einen Punkt der Datenaufnahme dar und leitet die Ereignisse an den Recorder weiter. (3) Der Recorder registriert die Ereignisse und speichert sie im Datenformat des Aufzeichnungssystems. (4) Das Anwendungsprogramm verarbeitet die Ereignisse und produziert Ausgaben in Dateien, am Bildschirm usw. (5) Als zweiter Punkt der Datenaufnahme können Screenshots vom Bildschirm angefertigt und vom Recorder verarbeitet werden.

<sup>5</sup> Anmerkung: Die Existenz eines solchen Hooks wird an dieser Stelle vorausgesetzt. Wenn kein solcher Mechanismus existiert, sind einfallsreiche Programmierer gefragt, die ggf. unter Umgehung der Sicherheitsmechanismen des Betriebssystems und „zu Fuß“ einen Monitor für die Eingabehardware realisieren.

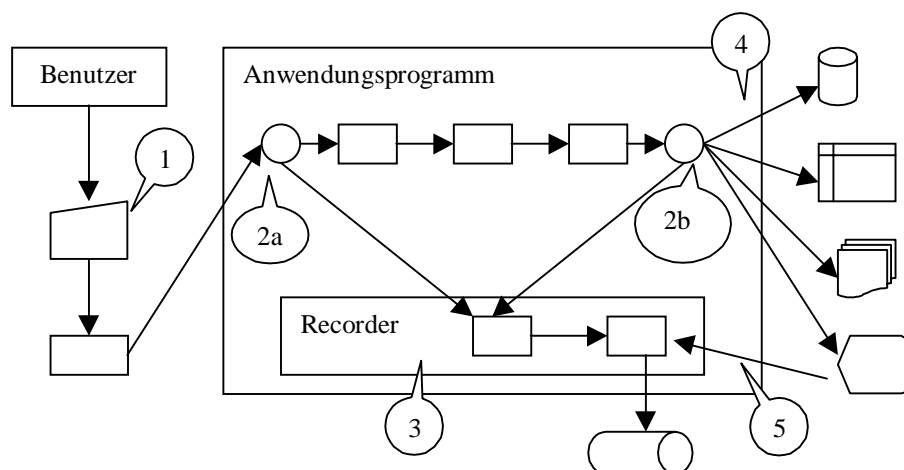


Abbildung 4: Datenfluß bei einem in das Anwendungsprogramm integrierten Recorder

Abbildung 4 skizziert den Datenfluß bei Verwendung eines Recorders, der in das aufzuzeichnende Anwendungsprogramm integriert ist: Zunächst erfolgt die Verarbeitung von Aktionen, die durch den Benutzer ausgelöst werden, durch das Betriebssystem (1), wie zuvor bereits beschrieben. Für die Datenaufnahme durch den Recorder können jetzt jedoch verschiedene Punkte vorgesehen werden: (2a) Systemereignisse können sowohl „pur“ als auch inklusive ihrer anwendungsspezifischen semantischen Bedeutung erfaßt werden, etwa in einer Form „Button\_Load\_Clicked“, „Window\_DataView\_Moved“. Darüber hinaus können zusätzliche Daten, die aus der Verarbeitung der Eingabe durch das Anwendungsprogramm resultieren aufgezeichnet werden (2b). Dabei kann es sich um die Initialisierung von Bedienelementen, handeln oder um andere Zustandsänderungen, die sich aus dem aktuellen Verarbeitungskontext ergeben und die als relevant für die Wiedergabe angesehen werden. Die Registrierung und Speicherung der Ereignisse durch den Recorder (3), sowie die Verarbeitung und die Erzeugung von Ausgaben durch das Anwendungsprogramm (4) erfolgen wie gehabt. Darüber hinaus können bei Bedarf wieder Screenshots angefertigt werden (5).

## Schlußfolgerungen

Die erste Alternative, bei der Systemereignisse ausschließlich unabhängig vom semantischen Kontext sowie Screenshots aufgezeichnet werden, bietet offensichtlich den Vorteil, daß keine Eingriffe in das aufzuzeichnende Programm notwendig sind. Sie wird in der Regel dann jedoch keine ausreichenden Daten liefern, wenn gefordert wird, daß die Wiedergabe einer nochmaligen Ausführung gleichkommt. Die Wiedergabe ab einem bestimmten Punkt wird nur möglich sein, wenn die Anwendung praktisch zustandslos arbeitet oder der Zustand an einer beliebigen Position der Aufzeichnung durch das (erneute) Abspielen der bis zu diesem Punkt aufgezeichneten Ereignisse wieder hergestellt werden kann. Darüber hinaus muß gewährleistet sein, daß bei jedem Start der Anwendung auch derselbe Initialzustand hergestellt wird. Eine Diskussion dieser Problematik erfolgt bspw. in [MP95].

Im weiteren Verlauf dieser Arbeit wird das Konzept eines von der aufgezeichneten Anwendung unabhängigen Recorders nicht weiter verfolgt werden. Es wird vorausgesetzt, daß die Aufzeichnungskomponente in das Anwendungsprogramm integriert werden kann.

### c) Die Bearbeitungskomponente (Editor)

Ein Editor kann dazu verwendet werden, Aufzeichnungen zu verwalten, zu analysieren und ggf. auch zu bearbeiten. Er stellt die dazu notwendigen Funktionen sowie eine entsprechende Benutzungsschnittstelle für den Endbenutzer zur Verfügung. Welche Operationen im Rahmen eines Editors im Detail realisiert werden können, wird stark von den Anwendungen abhängen, für die Aufzeichnungen erzeugt wurden. Bei Aufzeichnungen, die z.B. eine Art Makrofunktionalität realisieren sind andere Bearbeitungsfunktionen vorstellbar, als bei Aufzeichnungen zur Dokumentation oder Visualisierung von Vorgängen.

In jedem einzelnen Fall muß dabei untersucht werden, welche Operationen möglich sind und zugelassen werden können, damit eine editierte Aufzeichnung noch als sinnvoll oder funktionierend im Sinne der Anwendung angesehen werden kann. Im Allgemeinen dürfte man sich einen Editor für die Anwendungsaufzeichnung eher als ein Werkzeug zur Verwaltung und Analyse von Aufzeichnungen vorstellen, im Gegensatz etwa zu Text-, Grafik- oder auch Animationseditoren, bei denen die Sinnhaftigkeit eines Ergebnisses auch unter ästhetischen und nicht nur unter funktionalen Gesichtspunkten betrachtet werden kann. Im Rahmen dieser Arbeit werden Editoren nicht näher betrachtet.

### d) Die Wiedergabekomponente (Player)

Die Aufgabe des Players besteht darin, die notwendigen Funktionen für die Wiedergabe von Aufzeichnungen zur Verfügung zu stellen sowie ggf. eine Benutzungsschnittstelle für den Endbenutzer. Ebenso wie beim Recorder steht auch beim Player zur Disposition, inwieweit eine Integration in das aufgezeichnete Anwendungsprogramm erfolgt. Darüber hinaus ist für die reine visuelle Wiedergabe ein Player denkbar, der vollständig ohne die aufgezeichnete Anwendung arbeitet. Die folgenden Abbildungen detaillieren diese drei Alternativen.

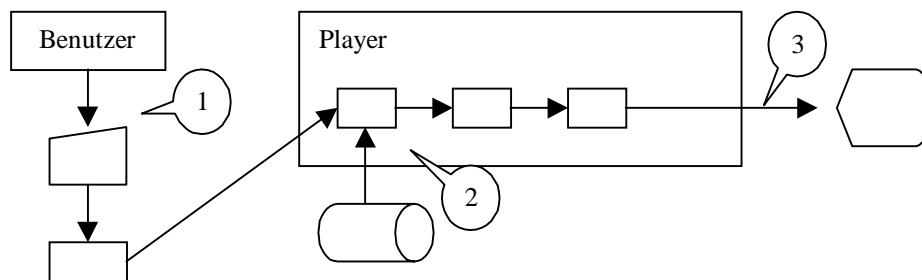


Abbildung 5: Datenfluß bei Verwendung eines Runtime-Moduls als Player

Abbildung 5 skizziert den Datenfluß bei Verwendung eines Runtime-Moduls als Player zur Wiedergabe ohne die aufgezeichnete Anwendung. (1) Der Benutzer steuert den Player über die integrierte Benutzungsschnittstelle. (2) Der Player liest die Aufzeichnung und erzeugt aus den Aufzeichnungsdaten die visuelle Wiedergabe (3).

Die Vorteile eines Runtime-Players liegen auf der Hand: Aufzeichnungen können zusammen mit dem Player weiter- und wiedergegeben werden, ohne daß die Anwendung benötigt wird. Falls die Wiedergabe jedoch nicht nur aus einer visuellen Darstellung bestehen soll, wird die Integration der aufgezeichneten Anwendungen in der Regel unabdingbar oder wenigstens ein gezielt für den Anwendungszweck erstellter Runtime-Player erforderlich sein.

Das im weiteren Verlauf dieser Arbeit entwickelte System wird einen Runtime-Player enthalten, der für die visuelle Wiedergabe von Aufzeichnungen von Java-AWT- und JFC/Swing-Komponenten verwendet werden kann.

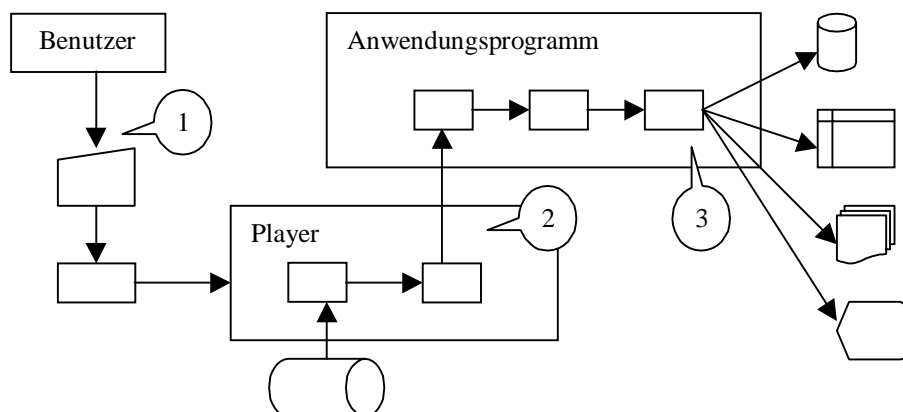


Abbildung 6: Datenfluß bei Verwendung eines vom Anwendungsprogramm unabhängigen Players

Abbildung 6 skizziert den Datenfluß bei Verwendung eines Players, der als eigenständiges Programm realisiert ist und das aufgezeichnete Anwendungsprogramm ansteuert. (1) Der Benutzer steuert den Player über dessen integrierte Benutzungsschnittstelle. (2) Der Player liest die Daten der Aufzeichnung und erzeugt Systemereignisse, die dem Anwendungsprogramm wie „echte“ Benutzereingaben zugespielt und von diesem zu Ausgaben entsprechend dem Vorgang bei der Aufzeichnung verarbeitet werden (3). Diese Alternative korrespondiert zu der oben genannten Variante eines Recorders ohne Integration des Anwendungsprogramms. Es gelten die dort genannten Vor- und Nachteile: Es ist im Idealfall keine Anpassung des Anwendungsprogramms nötig, jedoch sind die Anwendungsmöglichkeiten eingeschränkt.

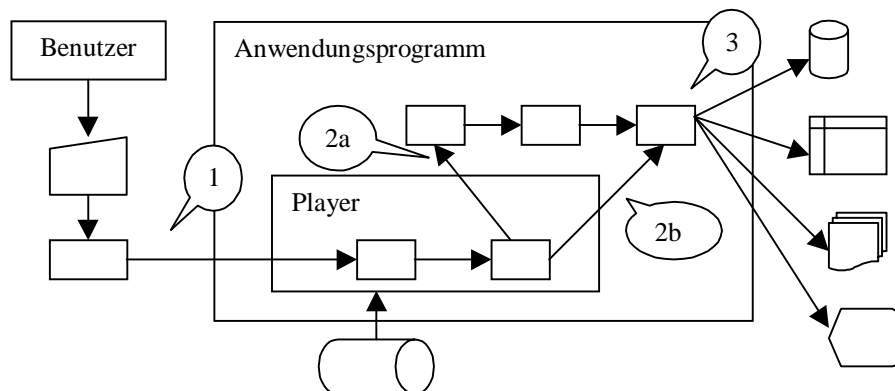


Abbildung 7: Datenfluß bei einem in das Anwendungsprogramm integrierten Player

Abbildung 7 skizziert den Datenfluß bei einem in das Anwendungsprogramm integrierten Player. Die Steuerung des Players kann mittelbar durch die Anwendung selbst erfolgen oder durch den Benutzer (1) über eine spezielle, integrierte Benutzungsschnittstelle. (2a) Der Player kann sowohl Ereignisse, die „echten“ Benutzereingaben gleichkommen (z.B. Bewegungen der Maus, Tastatureingaben etc.) als auch zusätzliche Daten, die während der Aufzeichnung gewonnen wurden, in den Wiedergabeprozess einbringen (2b). Die Anwendung erzeugt die entsprechenden Ausgaben (3), je nach Anforderung nur visueller Art, oder in einer Form die einer nochmaligen Ausführung gleichkommt. Der integrierte Player bietet die größtmögliche Flexibilität bei der Wiedergabe, erfordert dementsprechend jedoch auch einen erhöhten Aufwand für die Anpassung des Anwendungsprogramms. Dieses Konzept wird, neben dem oben geschilderten Runtime-Player, im Verlauf dieser Arbeit weiter verfolgt werden.

## 2. Die grafische Benutzungsschnittstelle

Grafische Benutzungsschnittstellen sind heute für nahezu jeden Computeranwender ein alltäglicher Begriff. Selbst der unerfahrene Benutzer liegt meist nicht falsch, wenn er einfach auf den Bildschirm zeigt, mit dem Finger eine kreisende Bewegung macht und damit die grafische Benutzungsschnittstelle umschreibt.

Für die Aufzeichnung und Wiedergabe einer grafischen Benutzungsschnittstelle sollte jedoch im Detail untersucht werden, was konkret unter dieser Schnittstelle verstanden werden soll, welche Funktion sie erfüllt und aus welchen Elementen sie besteht. Daher werden in diesem Kapitel grafische Benutzungsschnittstellen in erster Linie konzeptionell, d.h. anwendungs- und betriebssystemunabhängig, betrachtet. Die dabei identifizierten Elemente und Strukturen werden im Einzelnen weiter auf ihre Eigenschaften und im Hinblick auf die Aufgabenstellung Aufzeichnung und Wiedergabe analysiert.

### 2.1 Definition einiger grundlegender Begriffe

#### 2.1.1 Schnittstelle, Benutzungsschnittstelle

Entsprechend [DIN44-1] ist unter einer Schnittstelle im Allgemeinen der Übergang an der Grenze zwischen zwei Einheiten mit vereinbarten Regeln für die Übergabe von Daten oder Signalen zu verstehen. [SH97] definiert auf S. 103 den Begriff Benutzerschnittstelle wie folgt: „Unter der *Benutzerschnittstelle* ... versteht man die Benutzerführung, die dem Benutzer am Bildschirm für den Dialog mit dem Computer zur Verfügung gestellt wird...“. In diesem Sinne bestimmt also die Benutzungsschnittstelle eines computerbasierten Anwendungssystems, wie an der Grenze zwischen den technischen Systemkomponenten, Hardware und Software, und der menschlichen Komponente, dem Benutzer, Daten ausgetauscht werden können.<sup>6</sup>

#### 2.1.2 Grafische Benutzungsschnittstelle

In [STA96] heißt es auf S. 132: „Graphische Benutzungsschnittstellen (Graphical User Interfaces (GUIs)) bezeichnen eine Generation von Benutzungsschnittstellen, die sich aus der Idee der Visualisierung von Daten und Kontrollmanipulation entwickelt hat“. Dementsprechend wird der Begriff „grafische Benutzungsschnittstelle“ im Allgemeinen im Zusammenhang mit Bildschirmarbeitsplätzen verwendet, die wenigstens einen grafikfähigen Farbbildschirm, eine Tastatur sowie ein Zeigeinstrument (meist eine sog. Maus) zur Verfügung stellen.

---

<sup>6</sup> Anmerkung: Die Begriffe „Benutzerschnittstelle“ und „Benutzungsschnittstelle“ sowie „Benutzeroberfläche“ und „Benutzungsoberfläche“ werden je nach Autor und Quelle für den gleichen Sachverhalt verwendet. Gegen den Terminus „Benutzerschnittstelle“ oder „Benutzeroberfläche“ wird gelegentlich eingewandt, daß es sich nicht um die Oberfläche des Benutzers (mithin also um ein dermatologisches Problem) handelt, sondern um die Schnittstelle, zwischen System und Benutzer, die für die Benutzung desselben (des Systems nicht des Benutzers) zur Verfügung gestellt wird. Im Allgemeinen Sprachgebrauch scheint der Begriff „Benutzerschnittstelle“ bzw. „Benutzeroberfläche“ gebräuchlicher zu sein, jedenfalls nach der Alltagserfahrung des Autors dieser Arbeit. Dennoch sollen hier entsprechend [DUDEN] konsequent die Begriffe „Benutzungsoberfläche“ bzw. „Benutzungsschnittstelle“ verwendet werden.

Ein grafikfähiger Bildschirm zeichnet sich dadurch aus, daß er nicht nur die Ausgabe von bestimmten Zeichen, d.h. Buchstaben, Ziffern und Sonderzeichen, die durch einen konkreten Zeichensatz vorgegeben sind, erlaubt, sondern auch die Darstellung von beliebigen Symbolen und geometrischen Darstellungselementen wie Linien, Kreisen, Flächen etc..

Tastatur und Zeigeinstrument dienen als Eingabegeräte, wobei Aktionen, die der Benutzer unmittelbar über diese Geräte auslöst, mittelbar in Ereignisse auf dem Bildschirm umgesetzt werden. So wird bspw. die Bewegung der Maus auf dem Schreibtisch in eine Bewegung des Mauszeigers auf dem Bildschirm umgesetzt oder ein Tastendruck in das Öffnen eines Auswahlmenüs. Der Datenfluß erfolgt dabei in beiden Richtungen, vom Anwendungssystem zum Benutzer und vom Benutzer zum Anwendungssystem, über die am Bildschirm sichtbaren Elemente der Benutzungsschnittstelle. Der Bildschirm fungiert also nicht nur als Ausgabe-, sondern mittelbar auch als Eingabegerät. Wenn von der grafischen Benutzungsschnittstelle die Rede ist, sollen demzufolge darunter die am Bildschirm sichtbaren Elemente verstanden werden und nicht die Systemkomponenten, die durch diese Elemente repräsentiert werden.

D.h. es wird lediglich der Mauszeiger als abstraktes Gerät am Bildschirm betrachtet, nicht das Zeigegerät selbst, bei dem es sich um eine Maus, eine Rollkugel (Trackball) oder auch ein Grafiktablett handeln kann. Am Bildschirm dargestellte Elemente werden als Repräsentanten für die physischen Geräte angesehen. Ein Piktogramm, das einen Drucker darstellt, und die darüber ansprechbaren Dialogfelder ersetzen das Bedienfeld eines Druckers, „echte“ Tasten, Leuchtdioden usw..

Diese Interpretation, wonach die grafische Benutzungsschnittstelle die am Bildschirm sichtbaren Elemente umfaßt, die für den Datenaustausch zwischen Benutzer und Anwendungssystem zur Verfügung stehen, wird im Folgenden allen Betrachtungen zu Grunde gelegt.

### 2.1.3 Fenster (Windows), Komponenten (Components), Bedienelemente etc.

Die Benennung der einzelnen Elemente einer grafischen Benutzungsschnittstelle hängt nicht zuletzt von dem untersuchten System sowie von der Perspektive des Betrachters als Benutzer oder als Anwendungsentwickler ab.

Der Benutzer hat, abhängig vom Erfahrungsschatz, in der Regel eine intuitive Anschauung von einem Fenster als einem Teilbereich des Bildschirms, in dem Daten eines Programms angezeigt werden und mit dem bestimmte Operationen durchgeführt werden können. Bspw. das Öffnen, Schließen, Verschieben usw. mit der Maus.

Unter Microsoft Windows wird aus der Sicht des Programmierers jedes Element, das durch eine bestimmte Datenstruktur beschrieben wird und mit bestimmten Betriebssystemfunktionen manipuliert werden kann, als Fenster (Window) bezeichnet. Unter Java dagegen stellt die Klasse „Component“ die allgemeine Oberklasse für Elemente der grafischen Benutzungsschnittstelle dar. Ein Window bezeichnet dort lediglich bestimmte Objekte. In einem anderem Kontext werden wiederum bei der Anwendungsentwicklung zusammenfügbare Programmbausteine als Components (kurz für Componentware) bezeichnet.

Die teilweise Doppeldeutigkeit dieser und anderer Begriffe wird bei der Beschreibung der unterschiedlichen Systeme im Folgenden beibehalten, wenn aus dem jeweiligen Kontext ersichtlich ist, welcher Sachverhalt durch einen bestimmten Begriff bezeichnet wird.

Im Übrigen werden allgemeine Begriffe wie „Element“ oder „Bedienelement“ verwendet, wenn verdeutlicht werden soll, daß keine besondere Klasse von Objekten bezeichnet werden soll oder muß. Darüber hinaus werden als allgemein bekannt vorausgesetzte Begriffe wie „Schaltfläche“, „Kontrollkästchen“ usw. verwendet, die hier nicht explizit definiert werden.

## 2.2 Klassifikation der Elemente einer grafischen Benutzungsschnittstelle

Die Elemente, aus denen sich eine grafische Benutzungsschnittstelle zusammensetzt, können nach verschiedenen Kriterien oder Merkmalen klassifiziert werden. In den folgenden Abschnitten werden dabei unterschiedliche Perspektiven eingenommen und daraus Rückschlüsse auf verschiedene Notwendigkeiten bei der Aufzeichnung und Wiedergabe der grafischen Benutzungsschnittstelle gezogen.

### 2.2.1 Die Perspektive des Betrachters

Aus welchen Elementen setzt sich eine grafische Benutzungsschnittstelle zusammen? Die Beantwortung dieser Frage hängt unter anderem von der Perspektive des Betrachters ab. Ein „reiner“ Benutzer, der selbst nicht mit der Entwicklung von Anwendungsprogrammen befaßt ist, wird in der Regel andere Elemente der grafischen Benutzungsschnittstelle identifizieren oder vorrangig benennen als der Anwendungsentwickler. Dessen Perspektive wird wiederum von der eines Systementwicklers abweichen.<sup>7</sup>

Abbildung 8 zeigt ein einfaches Dialogfeld mit einer Schaltfläche, durch das eine Meldung ausgegeben wird und das als ein Standardelement einer grafischen Benutzungsschnittstelle betrachtet werden kann. Aus welchen Elementen setzt sich dieses Dialogfeld zusammen? Diese Frage wird im Folgenden aus der Sicht der drei Gruppen: Benutzer, Systementwickler und Anwendungsentwickler beantwortet.

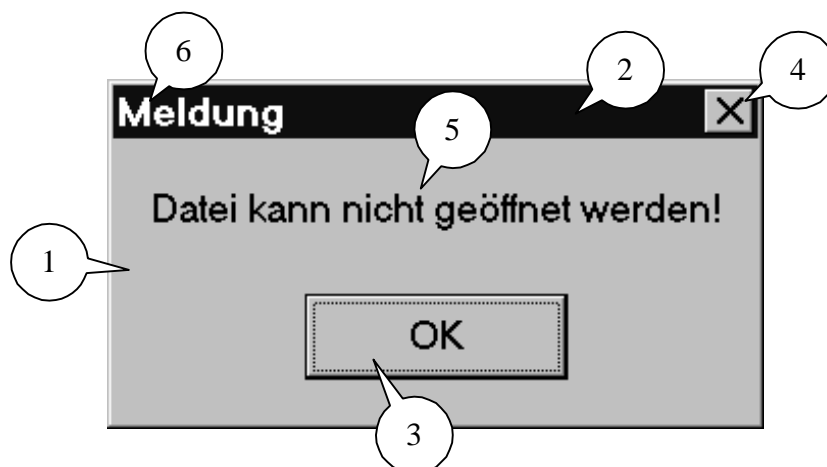


Abbildung 8: Ein einfaches Dialogfeld

<sup>7</sup> Anmerkung: Unter einem Anwendungsentwickler ist in diesem Zusammenhang ein Entwickler zu verstehen, der ein Anwendungsprogramm entwirft und bei der Codierung der Benutzungsschnittstelle auf Standardelemente zurückgreift, die das Betriebssystem oder eine Programmbibliothek zur Verfügung stellt. Der Systementwickler ist derjenige Entwickler, der das Betriebssystem oder die Programmbibliothek, die dieses Element zur Verfügung stellt, entwirft oder codiert.

### a) Die Perspektive des Benutzers

Aus der Perspektive des Benutzers könnte das in Abbildung 8 dargestellte Dialogfeld aus den folgenden Elementen bestehen: (1) dem Dialogfeld als Fenster an sich, (2) der Titelleiste des Fensters, (3) der Schaltfläche „OK“, (4) der Systemschaltfläche zum Schließen des Dialogs, (5) dem Text der Meldung und (6) dem Titel des Dialogfeldes.

Diese Aufzählung ist in gewissem Maße spekulativ und mag abhängig von der Erfahrung des Benutzers im Detail auch anders ausfallen. Bspw. könnte der Titel nicht von der Titelleiste unterschieden oder der Rahmen des Dialogfeldes als weiteres Element benannt werden. Aus der Sicht des Benutzers wird es jedoch naheliegend sein, diejenigen Elemente, die bereits durch ihre grafische Darstellung als abgeschlossene Elemente identifiziert werden können, auch als funktionale Elemente der Benutzungsschnittstelle anzusehen. Nach [STA96] ergibt sich z.B. aus Untersuchungen der menschlichen Wahrnehmung und aus den Gestaltungsgesetzen, daß Benutzungsschnittstellen genau so gestaltet werden sollten, daß funktionale Elemente intuitiv durch ihre Darstellung identifiziert werden können.

### b) Die Perspektive des Systementwicklers

Aus der Perspektive des Systementwicklers, der Standardelemente einer grafischen Benutzungsschnittstelle selbst entwickelt, besteht das Dialogfeld in Abbildung 8 aus ungleich mehr Elementen. Allein die Schaltfläche „OK“ (3) kann in folgende Unterelemente zerlegt werden: 1. Den Rahmen, der die Schaltfläche mit einem dreidimensionalen Effekt vom Untergrund abhebt, 2. den „OK“ Text, 3. den grauen Untergrund, 4. die schwarze Umrandung, die anzeigt, daß die Schaltfläche das Standardelement des Dialogfeldes ist und unabhängig vom Tastaturfokus durch einen Druck auf die Eingabetaste bestätigt werden kann, und 5. einen gepunkteten Rahmen um den „OK“ Text, der anzeigt, daß die Schaltfläche den Fokus besitzt.<sup>8</sup> Auf dieselbe Art und Weise kann mit den anderen Elementen des Dialogfeldes verfahren werden. Elemente, die der Benutzer als funktional nicht zerlegbare Einheiten betrachtet, sind für den Systementwickler zusammengesetzte, komplexe Elemente. Während die Sichtweise des Benutzers, wie oben beschrieben, variabel sein und von dessen Erfahrung und Erwartung abhängen kann, ist die Identifikation und detaillierte Betrachtung aller Unterelemente eines Bedienelements für den Systementwickler unabdingbar. Jedes einzelne Unterelement besitzt Attribute, die sowohl die Darstellung (Farbe, Ausmaße, Form etc.) als auch die Funktion des Elements betreffen und bei der Implementierung berücksichtigt werden müssen.

### c) Die Perspektive des Anwendungsentwicklers

Die Position des Anwendungsentwicklers wird in der Regel zwischen der des Benutzers und der des Systementwicklers liegen. Er kann einerseits auf Systemroutinen oder Routinen eines Toolkits für die Programmierung der Benutzungsschnittstelle zurückgreifen. Er muß sich also bei der Verwendung von Standardelementen nicht um die Details der grafischen Darstellung kümmern oder bei der Verwendung von Programmbausteinen nicht um die unmittelbare Verarbeitung einzelner Teilschritte bei der Benutzerinteraktion. Andererseits steht ihm neben den Funktionen, die das Betriebssystem oder ein Toolkit bieten, auch die Möglichkeit offen, die Schnittstelle mit Hilfe von Low-Level-Routinen selbst zu programmieren. Auf diese unterschiedlichen Möglichkeiten soll im Folgenden detaillierter eingegangen werden.

---

<sup>8</sup> Anmerkung: Der Begriff „Fokus“ hängt mit der Verarbeitung von Tastatureingaben zusammen. Eingaben und Tastenkombinationen, die nicht durch systemweite Dienste bearbeitet werden, werden durch das Betriebssystem an dasjenige Bedienelement weitergeleitet, das den Fokus besitzt.

Das Dialogfeld aus Abbildung 8, das sowohl für den Benutzer als auch für den Systementwickler ein verhältnismäßig komplexes, zusammengesetztes Element darstellt, kann aus der Sicht des Anwendungsentwicklers als ein elementares, nicht zerlegbares Standardelement mit wenigen Attributen betrachtet werden. Die Erzeugung dieses Dialogfeldes kann bspw. unter Microsoft Windows 98 folgendermaßen mit einer einzigen Anweisung erfolgen:

In Microsoft Visual C++ 6.0 mit der Anweisung:

```
MessageBox(NULL,"Datei kann nicht geöffnet werden!", "Meldung",MB_OK);
```

In Microsoft Visual Basic 6.0 mit der Anweisung:

```
MsgBox "Datei kann nicht geöffnet werden!", vbOKOnly, "Meldung"
```

Bei dem ersten Funktionsaufruf handelt es sich um eine Routine der Win32 API.<sup>9</sup> Im zweiten Fall wird eine Routine der Visual Basic Runtime-Bibliothek aufgerufen, die (vermutlich) auf dieselbe Betriebssystemroutine zurückgeführt wird. Nach der Erzeugung des Dialogfeldes bestehen für den Anwendungsentwickler keine weiteren Einflußmöglichkeiten mehr. Auch dies ist ein Beleg dafür, daß es sich aus seiner Sicht um ein elementares Element handelt. Diejenigen Anweisungen, die jeweils auf die obigen Anweisungen folgen, werden ausgeführt, wenn der Benutzer das Dialogfeld beantwortet oder geschlossen hat. Der Kontrollfluß wird während dieser Zeit, während das Dialogfeld geöffnet ist, an die Bibliothek übergeben, die das Dialogfeld realisiert.<sup>10</sup>

#### d) Schlußfolgerungen

In diesem Abschnitt wurde gezeigt, daß es von der Perspektive des Betrachters abhängt, aus welchen Elementen sich eine grafische Benutzungsschnittstelle zusammensetzt, bzw. wie detailliert Elemente in Unterelemente zerlegt werden können oder müssen. Im Hinblick auf die Entwicklung eines Toolkits für die Aufzeichnung und Wiedergabe einer grafischen Benutzungsschnittstelle wird es erforderlich sein, eine Position zu wählen, die zwischen der des Systementwicklers und der des Anwendungsentwicklers liegt.

Auf der einen Seite sollten einzelne Elemente weitgehend, d.h. bis hinab zu untergeordneten elementaren Einheiten, betrachtet werden, um das Verhalten ebenso wie das Erscheinungsbild der Benutzungsschnittstelle realitätsnah aufzeichnen und wiedergeben zu können. Inwieweit dies im Einzelfall überhaupt möglich ist, wird im Weiteren noch zu analysieren sein.

Andererseits sollte dem Anwendungsentwickler eine Schnittstelle für die Integration der Aufzeichnungs- und Wiedergabefunktionalität in Anwendungen zur Verfügung gestellt werden, die ungefähr dieselbe Granularität besitzt, wie die Programmierschnittstelle, die ohnehin bei der Anwendungsentwicklung genutzt wird. Bezogen auf den hier gezeigten Fall wird man bspw. von einem Anwendungsentwickler nur ausnahmsweise erwarten können, daß er ein Dialogfeld, das er eigentlich mit nur einer Programmanweisung erzeugen könnte, „von Hand“ programmiert, um dessen Aufzeichnung zu ermöglichen. Wenn Aufzeichnung und Wiedergabe nicht unabdingbare Bestandteile der Anwendung sind, wird der Anwendungsentwickler sonst mit Recht Widerstand leisten.

---

<sup>9</sup> Anmerkung: Auf die Bibliotheken zur Programmierung der grafischen Benutzungsschnittstelle unter Microsoft Windows, unter dem X Window/OSF Motif-System sowie unter Java wird in Kapitel 2.3 näher eingegangen.

<sup>10</sup> Anmerkung: Von nebenläufigen Programmfäden (Threads), die in demselben Anwendungsprogramm quasi zeitgleich ablaufen können, soll hier abgesehen werden.

## 2.2.2 Die Herkunft eines Bedienelements

Unter der Herkunft eines Elements der grafischen Benutzungsschnittstelle soll die Quelle verstanden werden, aus welcher der Anwendungsentwickler das Element bezieht. Dabei werden drei Quellen unterschieden: Standardelemente, individuell entwickelte Bedienelemente und vorgefertigte Programmbausteine (Components), die im Folgenden im Hinblick auf ihre Eignung für die Aufzeichnung und Wiedergabe näher untersucht werden.

### a) Standardelemente

Unter den Standardelementen einer grafischen Benutzungsschnittstelle sind solche Elemente zu verstehen, die entweder vom Betriebssystem oder einer allgemein verwendbaren, systemnahen Bibliothek zur Verfügung gestellt werden.

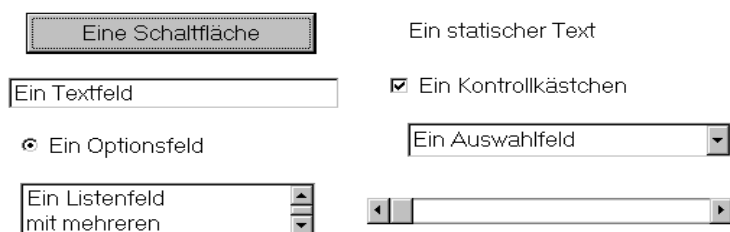


Abbildung 9: Einige Standardelemente von Benutzungsschnittstellen

Abbildung 9 zeigt einige Standardelemente grafischer Benutzungsschnittstellen. In der Regel können solche Elemente über eine öffentliche Schnittstelle erzeugt und programmiert werden. Bestimmte Attribute der Bedienelemente können in einem vorgegebenen Rahmen verändert und an die Erfordernisse der jeweiligen Anwendung angepaßt werden. Die Darstellung ist in gewissem Rahmen vorgegeben; es wird ein bestimmtes „Look and Feel“<sup>11</sup> gewährleistet, auf daß auch unterschiedliche Anwendungen sich dem Benutzer gegenüber erwartungskonform verhalten; nach [SH92] ein wesentlicher Aspekt bei der Erstellung effektiver Benutzungsschnittstellen.<sup>12</sup> Der Mechanismus, über den die Kommunikation mit dem Betriebssystem, das die unmittelbare Instanz für die Verarbeitung von Benutzereingaben darstellt, stattfindet, ist offengelegt und kann in Anwendungsprogrammen direkt genutzt werden. In der Regel handelt es sich um einen Mechanismus zum Nachrichtenaustausch über Callback-Funktionen. Für die Aufzeichnung und Wiedergabe besitzen Standardelemente gute Voraussetzungen. Eine in das Anwendungsprogramm integrierte Aufzeichnungskomponente kann sowohl programminterne Verarbeitungsschritte, z.B. zur Erzeugung oder Initialisierung der Elemente, als auch Ereignisse, die durch den Benutzer ausgelöst werden, aufzeichnen. Dabei können die gleichen Schnittstellen verwendet werden, die allgemein in Anwendungsprogrammen zur Verfügung stehen und damit wiederverwendbare Standardkomponenten für die Aufzeichnung und Wiedergabe entwickelt werden.

<sup>11</sup> Anmerkung: Unter dem Look and Feel einer Benutzungsschnittstelle ist ein typisches, von der Anwendung unabhängiges Aussehen und Verhalten von Programmen verstehen. Dies betrifft bspw. Fragen wie „Welche Systemschaltflächen existieren zum Schließen eines Fensters und wie werden sie angeordnet, rechts oder links in der Titelleiste?“ oder „Wie werden Schaltflächen dargestellt, quasi dreidimensional oder flach?“

<sup>12</sup> Zitat [SH92] S.72: „Eight Golden Rules of Dialog Design... 1. Strive for consistency. This principle is the most frequently violated one, and yet is the easiest one to repair and avoid...“

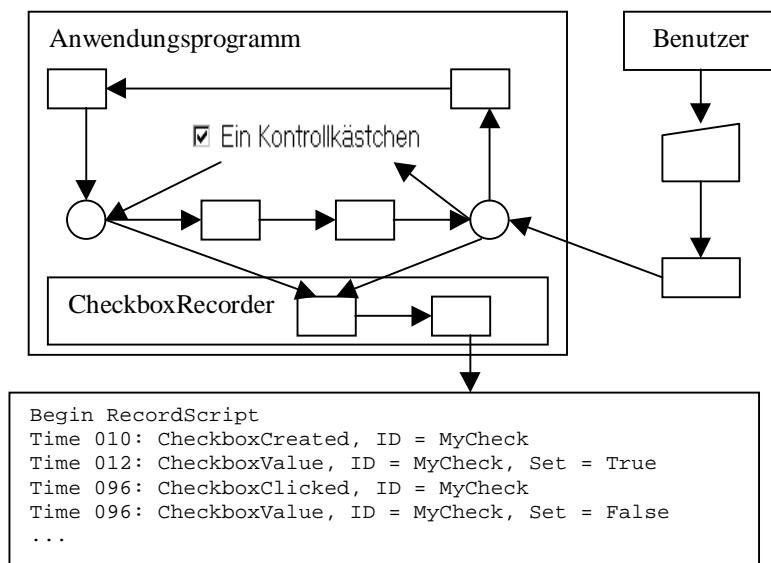


Abbildung 10: Ein Aufzeichnungskonzept für Standardelemente

Abbildung 10 skizziert ein Konzept für die Aufzeichnung von Standardelementen, das den zuvor genannten Umständen Rechnung trägt. Eine hier als CheckboxRecorder bezeichnete Komponente für die Aufzeichnung eines Standardelements wird in die Anwendung integriert und ermittelt die notwendigen Daten für die Wiedergabe des Elements. Da die Schnittstellen, das Verhalten und die Bedeutung der Nachrichten, die das Bedienelement, hier ein Kontrollkästchen (Checkbox) erhält, bekannt und standardisiert sind, kann auch die Aufzeichnungskomponente wiederverwendet und in unterschiedliche Anwendungen integriert werden.

In Abschnitt 2.3 werden die grafischen Benutzungsschnittstellen von Microsoft Windows, des X Window/OSF Motif-Toolkits und des Java-AWT beschrieben und Mechanismen gezeigt, die aufbauend auf dem oben dargestellten Konzept für die Aufzeichnung der Benutzungsschnittstelle verwendet werden können. Gebräuchliche Standardelemente werden in Abschnitt 2.4 näher beschrieben.

## b) Individuell entwickelte Bedienelemente

Unter individuell entwickelten Bedienelementen sind im Folgenden solche Elemente der grafischen Benutzungsschnittstelle einer Anwendung zu verstehen, die mit Hilfe von Low-Level-Routinen vom Anwendungsentwickler individuell codiert werden. Die Realisierung solcher Elemente kann bspw. dadurch erfolgen, daß das optische Erscheinungsbild mit elementaren Grafikfunktionen zum Zeichnen von Linien, Flächen, Bitmaps etc. in ein Fenster der Anwendung gezeichnet wird und Mausereignisse, die innerhalb der Fläche des Elements erfolgen, entsprechend der Anwendungslogik interpretiert werden. Praktisch werden also Aufgaben, die sonst vom Betriebssystem oder einer Bibliothek übernommen werden, nachprogrammiert.

Gründe für selbst programmierte Bedienelemente sind z.B. das Fehlen von bestimmten Eigenschaften bei den vorhandenen Standardelementen (bspw. Bildlaufleisten mit proportionalem Schieber unter Microsoft Windows 3.x) oder die Notwendigkeit von grafisch besonders gestalteten Bedienelementen, die vom Look And Feel des Betriebssystems abweichen, z.B. für Unterhaltungssoftware, die nicht wie eine „gewöhnliche“ Anwendung erscheinen soll.

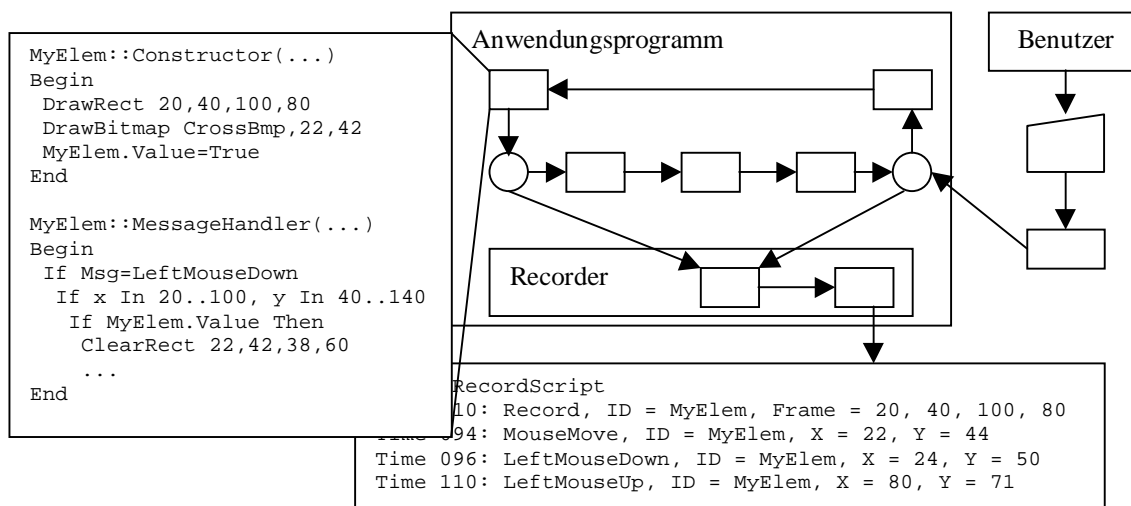


Abbildung 11: Ein Aufzeichnungskonzept für individuell entwickelte Bedienelemente

Für die Aufzeichnung von individuell entwickelten Bedienelementen kann zwangsläufig nur eine allgemeine rudimentäre Unterstützung zur Verfügung gestellt werden. Während eine Anwendung bei Standardelementen einer Aufzeichnungskomponente mitteilen kann: „Zeichne ein Element vom Typ XY auf“ und anschließend erfolgende Zustandsänderungen oder Nachrichten auf Basis derjenigen Informationen verarbeitet werden, die das Aufzeichnungssystem über Elemente vom entsprechenden Typ besitzt, können für individuell entwickelte Bedienelemente nur Ereignisse auf der niedrigeren Abstraktionsebene des Betriebssystems, bspw. Maus- und Tastaturereignisse, aufgezeichnet werden. Eine weitergehende Bedeutung im Zusammenhang mit dem Element kann diesen Ereignissen bei der Wiedergabe nur durch die aufgezeichnete Anwendung selbst wieder zugeordnet werden. Abbildung 11 liefert eine schematische Darstellung eines entsprechenden Aufzeichnungskonzeptes.

Der Aufzeichnungsmechanismus könnte zwar allgemein so erweitert werden, daß beliebige Daten in binärer Form abgespeichert werden können. Dennoch bleibt es die Aufgabe der aufgezeichneten Anwendung, diese Daten bei der Wiedergabe zu interpretieren und ihnen eine Bedeutung im Zusammenhang mit aufgezeichneten Ereignissen zuzuordnen. Dies wird einen höheren Aufwand für den Anwendungsentwickler nach sich ziehen, der die entsprechende Unterstützung zusätzlich implementieren muß.

### c) Vorgefertigte Programmbausteine (Components)

Sog. „Components“ oder „Componentware“ sind vorgefertigte Programmbausteine, die in Anwendungsprogramme integriert oder zu Anwendungen zusammengebaut werden können und die eine eigene Anwendungslogik mit integrierter Verarbeitung besitzen. Es handelt sich quasi um wiederverwendbare Anwendungen im Kleinen. Components können sowohl von Fremdherstellern, als auch vom Systemanbieter, als auch aus eigener Entwicklung stammen. Wesentliche allgemeine Merkmale von Components sind jedoch im Gegensatz zu den unter a) genannten Standardelementen und auch im Gegensatz zu Klassenbibliotheken:

- daß es sich um abgeschlossene, zusammengesetzte, komplexe Elemente handelt, die sowohl eine eigene Benutzungsschnittstelle als auch eine integrierte Verarbeitung (nicht nur sondern auch) für Benutzerinteraktionen besitzen können.
- daß die Programmbausteine für den Anwendungsentwickler in der Regel nicht in Form des Programmquelltextes sondern bereits in ausführbarer Form vorliegen.

- daß die Kommunikation mit dem Component ausschließlich über eine exakt definierte Schnittstelle erfolgt, die auf den Anwendungsbereich des Components zugeschnitten ist, wobei dieser Anwendungsbereich eine auf den Endbenutzer abzielende Größe ist.<sup>13</sup>

Abbildung 12 zeigt bspw. ein Component, ein Microsoft Calendar Control 8.0, das zusammen mit der Entwicklungsumgebung Microsoft Visual Basic 6.0 ausgeliefert wird. Das Component besitzt eine eigene Benutzungsschnittstelle, die aus mehreren Bedienelementen besteht, sowie folgende integrierte Verarbeitung für Benutzerinteraktionen: Die Wahl eines Monats oder Jahres über die Auswahlfelder führt zur automatischen Neuberechnung der Ausgabe des Kalenderblattes. Wird ein Tag im aktuellen Monat mit der linken Maustaste angeklickt, so wird das entsprechende Datum grau hinterlegt. Wird ein Tag im vorhergehenden oder folgenden Monat angeklickt, so wird der entsprechende Monat vollständig eingeblendet.

August 1998    August    1998

Sun	Mon	Tue	Wed	Thu	Fri	Sat
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Abbildung 12: Ein Microsoft Calendar Control als Beispiel für ein Component

Abbildung 13 zeigt die Ausgabe des Microsoft Spy++ V6.00 für das in Abbildung 12 dargestellte Component. Das Calendar Control besteht demzufolge aus einem Fenster, das zwei Auswahlfelder (Comboboxes) enthält, die Standardelemente unter Windows sind. Bei dem eigentlichen Kalenderblatt, das eindeutig auch ein Element der grafischen Benutzungsschnittstelle ist, handelt es sich also um ein individuelles Bedienelement, wie unter b) beschrieben.

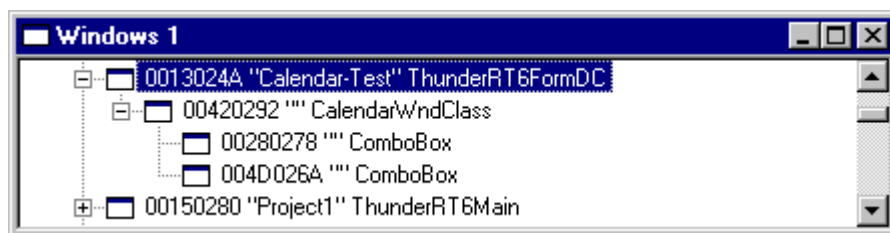


Abbildung 13: Das Calendar Control im Microsoft Spy++

Im Hinblick auf die Aufzeichnung stellen Components eine besondere Herausforderung dar. Ihre Programmierschnittstelle ist in der Regel auf die anwendungsspezifische Aufgabe ausgelegt, die das Component erfüllen soll. Die einzelnen Elemente, aus denen das Component besteht, werden dabei gekapselt und sind nach außen nicht mehr direkt zugänglich.

<sup>13</sup> Anmerkung: Auch bei Standardelementen erfolgt die Programmierung natürlich über eine exakt definierte Schnittstelle. Jedoch ist diese nicht anwendungsspezifisch im Hinblick auf den Endbenutzer. Durch die Schnittstelle für die Programmierung eines Standardelements vom Typ Listenfeld wird im Allgemeinen nicht festgelegt werden, was in einem solchen Listenfeld angezeigt wird, jedenfalls nicht im Hinblick auf eine bestimmte Anwendung. Es werden einfach „irgendwelche“ Texte angezeigt.

Ein Component für den Zugriff auf Datenbanken könnte dagegen bspw. ein Bedienelement vom Typ Listenfeld zur Verfügung stellen, welches das Ergebnis einer Datenbankabfrage in Tabellenform anzeigt. Die Schnittstelle des Components könnte eine Methode enthalten, mit welcher durch eine einzige Anweisung die Verbindung zu einer entfernten Datenbank hergestellt, eine Abfrage ausgeführt und das Ergebnis angezeigt werden kann.

Bezogen auf das oben genannte Beispiel eines Calendar Controls erhält die Anwendung, in die das Component integriert ist, z.B. eine Nachricht, wenn der Monat geändert wurde. Die Anwendung und damit auch eine zu entwickelnde Aufzeichnungskomponente kann jedoch nicht mit Sicherheit feststellen, ob diese Änderung durch die Verwendung des Auswahlfeldes am oberen Rand des Components herbeigeführt wurde oder auf einem anderen Weg.

Eine realitätsnahe Aufzeichnung der Benutzerinteraktion sollte jedoch evtl. sogar beinhalten, wie das Auswahlfeld zunächst geöffnet wurde, wie die Einträge in der aufklappbaren Liste durchgeblättert wurden und wie dann ein Eintrag ausgewählt wurde.

Im Allgemeinen verhalten sich Components also wie Anwendungsprogramme, die sich nicht „bewußt“ sind, daß sie aufgezeichnet oder wiedergegeben werden und die dementsprechend auch nicht mit den Aufzeichnungs- oder Wiedergabekomponenten kooperieren können. Der Anwendungsprogrammierer kann praktisch keine Annahmen über den internen Aufbau eines Components machen. Selbst wenn solche Informationen mit Hilfe entsprechender Tools (Disassembler, Debugger, Spy etc.) gewonnen werden können, kann eine neue Version des Components eine komplett neue Implementierung besitzen. Wie weit ein Component aufgezeichnet werden kann, wird also jeweils im Einzelfall zu untersuchen sein. Eine allgemeine Aufzeichnungsunterstützung könnte hier evtl. folgende Mechanismen, mit unterschiedlicher Effektivität und Effizienz zur Verfügung stellen. Abbildung 14 skizziert ein entsprechendes Aufzeichnungskonzept.

- Es wird versucht, untergeordnete Standardelemente des Components zu identifizieren. Soweit der Zugriff auf diese Objekte möglich ist, werden Zustandsänderungen und Nachrichten wie unter a) beschrieben, aufgezeichnet.
- Systemereignisse, bspw. Maus- und Tastatureingaben, die im Bereich des Components erfolgen, werden wie unter b) für selbst programmierte Bedienelemente dargestellt, aufgezeichnet.
- Die Darstellung des Components wird evtl. mittels Screen-Grabbing aufgezeichnet.

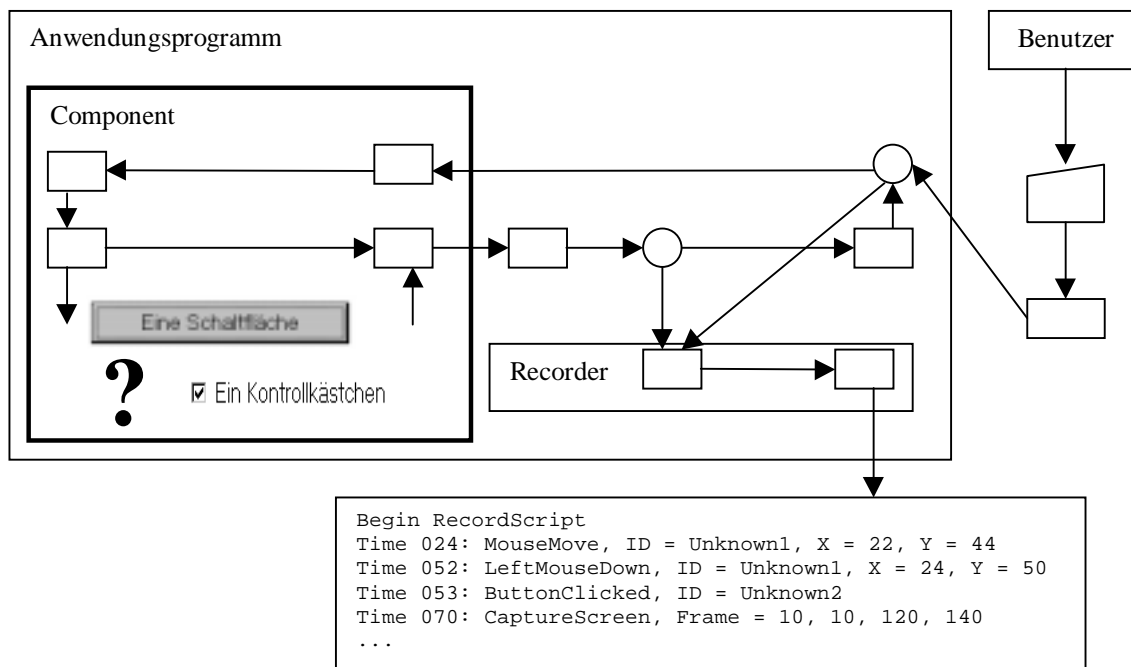


Abbildung 14: Ein Aufzeichnungskonzept für Components

### 2.2.3 Funktionsbereiche von Elementen grafischer Benutzungsschnittstellen

Den Elementen einer grafischen Benutzungsschnittstelle können verschiedene Funktionsbereiche zugeordnet werden, wobei ein bestimmtes Element, z.B. eine Schaltfläche oder eine Bildlaufleiste, mehrere der nachfolgend beschriebenen Funktionsbereiche ausfüllen kann. Bei der Aufzeichnung und Wiedergabe können bzw. müssen in Abhängigkeit von den gewünschten Verarbeitungsmöglichkeiten nur einzelne oder auch alle Funktionsbereiche in dem Sinne abgedeckt werden, daß die entsprechende Funktion bei der Wiedergabe mit ausreichender Qualität (wieder) erfüllt werden kann.

#### a) Benutzerinteraktion

Durch Benutzerinteraktion erfolgt der Informationsfluß vom Benutzer hin zur Anwendung. Interaktionselemente sind dabei diejenigen Elemente der grafischen Benutzungsschnittstelle, die auf Aktionen, die durch den Benutzer über das Zeigeelement oder die Tastatur ausgelöst werden, unmittelbar reagieren.

In diesem Sinne stellen die meisten Elemente der Benutzungsschnittstelle zu jedem Zeitpunkt Interaktionselemente dar. Nicht nur solche Elemente, die eine konkrete Aufgabe im Sinne der Anwendungslogik übernehmen (Schaltflächen, Kontrollkästchen, Textfelder usw.), sondern auch solche, die zur Organisation der grafischen Benutzungsschnittstelle selbst dienen, wie z.B. die Titelleiste zum Verschieben eines Fensters oder der Rahmen, über den die Fenstergröße aufgezoogen werden kann.

Ob ein Element zu einem bestimmten Zeitpunkt als Interaktionselement dienen kann oder nicht, hängt vom aktuellen Zustand der Anwendung und der Benutzungsschnittstelle ab. So ist der Hintergrund eines Dialogfeldes aus der Sicht des Benutzers auch ein Interaktionselement, wenn das Dialogfeld, das zur Zeit nicht das aktive Fenster ist, durch einen Mausklick auf den Hintergrund aktiviert werden kann. Keine Interaktionselemente sind dagegen z.B. Elemente die unsichtbar oder deaktiviert sind.

Die Aufzeichnung von Benutzerinteraktionen kommt im Wesentlichen der Aufzeichnung der Systemnachrichten gleich, mit deren Hilfe die Aktionen des Benutzers an die Anwendung übermittelt werden. Diese Nachrichten können in einem anwendungsunabhängigen Format aufgezeichnet, gespeichert und bei der Wiedergabe erneut eingespielt werden.

#### b) Visualisierung

Der Informationsfluß von der Anwendung zum Benutzer erfolgt im Rahmen von grafischen Benutzungsschnittstellen durch Visualisierung. Dies betrifft nicht nur Daten, die von der Anwendung durch Verarbeitung erzeugt und ausgegeben werden, z.B. über Statuszeilen oder Ausgabebereiche, sondern auch Informationen über den Zustand der Benutzungsschnittstelle selbst, z.B. Fenstergrößen und Positionen oder die farbliche Hervorhebung der Titelleiste des aktiven Fensters, sowie Rückmeldungen auf Benutzerinteraktionen, z.B. eine Schaltfläche, die durch einen grafischen Effekt vertieft dargestellt wird, wenn sie mit der Maus angeklickt wird, und erhöht, sobald die Maustaste losgelassen wird.

Im Hinblick auf die Wiedergabe der grafischen Benutzungsschnittstelle werden Daten, die für die Wiederherstellung des visuellen Eindrucks notwendig sind, zum Teil implizit durch die Daten geliefert, die bei der Aufzeichnung der Benutzerinteraktion gewonnen werden. Darüber hinaus müssen u.U. weitere Daten anwendungsspezifisch ermittelt werden.

In die erste Kategorie fallen bspw. unmittelbare Rückmeldungen auf Aktionen des Benutzers, wie die hervorgehobene oder invertierte Darstellung von Schaltflächen beim Anklicken mit der Maus, die wieder umgekehrt wird, sobald die Maustaste losgelassen wird. Dieser und ähnliche visuelle Effekte können u.U. bei der Wiedergabe nur wieder hergestellt werden, indem entsprechende Systemereignisse wieder an ein entsprechendes Bedienelement übermittelt werden. (Von der Verwendung von Bitmaps soll hier abgesehen werden.)

Zusätzliche Daten für die visuelle Wiedergabe müssen dann anwendungsspezifisch ermittelt werden, wenn Anzeigen durch die Verarbeitung der Anwendung ausgelöst werden, die nicht oder nur mittelbar auf Benutzerinteraktionen zurückgeführt werden können und wenn die Wiedergabe nicht einer nochmaligen Ausführung gleichkommen soll. Dies betrifft bspw. die Initialisierung von Listen oder Textbereichen.

### **c) Datenspeicherung**

Bestimmte Elemente der Benutzungsschnittstelle können direkt anwendungsbezogene Daten enthalten. Dies betrifft z.B. Daten, die in Textfeldern, Listenfeldern, Auswahlfeldern oder Baumansichten enthalten sind. Diese können sich mit den unter b) beschriebenen Daten, die für die Visualisierung herangezogen werden, überschneiden. Es kann sich jedoch hier auch um Daten handeln, die während einer Anwendungssitzung niemals sichtbar werden. Bspw. Text außerhalb des sichtbaren Bereichs eines Textfeldes oder verborgene Einträge einer Liste oder einer Baumansicht.

Bei der Aufzeichnung der grafischen Benutzungsschnittstelle müssen anwendungsbezogene Daten, die in Bedienelementen gespeichert sind, dann mit aufgezeichnet werden, wenn eine Wiedergabe erfolgen soll, die einer erneuten Ausführung entspricht. Darüber hinaus können solche Daten mittelbar sichtbar sein, bspw. in der Form der Größe des Schiebers einer Bildlaufleiste, was ebenfalls die Aufzeichnung erfordert, wenn eine optisch korrekte Wiedergabe erfolgen soll.

### **d) Gruppierung**

Verschiedene Elemente der grafischen Benutzungsschnittstelle können zur Gruppierung oder als Behälter für untergeordnete Elemente dienen. So dienen bspw. Rahmenfenster als Behälter für Dokumentenfenster oder Dialogfelder als Behälter für Bedienelemente oder Menüleisten als Behälter für Menüs.

Zwischen übergeordneten und den enthaltenen Elementen kann dabei sowohl eine statische als auch eine dynamische Beziehung bestehen. Die Beziehung zwischen einem Dialogfeld und den darauf liegenden Schaltflächen wird in der Regel statisch sein, da sie durch das Anwendungsprogramm fest vorgegeben ist. Die Beziehung zwischen einem Rahmenfenster und einem darin enthaltenen Dokumentenfenster ist jedoch in dem Ausmaß dynamisch, wie die Entstehung des Dokumentenfensters vom Verlauf der Anwendungssitzung und damit vom Benutzer bestimmt wird.

Dynamische Beziehungen zwischen Elementen der grafischen Benutzungsschnittstelle müssen bei der Aufzeichnung erfaßt und verfolgt werden, bspw. die Erzeugung und die relative Positionierung eines Dokumentenfensters im Verhältnis zu seinem Rahmenfenster.

Wenn die Wiedergabe nicht durch die aufgezeichnete Anwendung selbst, sondern durch einen Runtime-Player erfolgen soll, müssen auch statische Beziehungen explizit aufgezeichnet werden, um die entsprechenden Elemente nachbilden zu können. (Von der Verwendung von Bitmaps soll wiederum abgesehen werden.)

## 2.3 Beispiele für grafische Benutzungsschnittstellen

In diesem Kapitel werden drei real existierende grafische Benutzungsschnittstellen vorgestellt und anhand dieser Beispiele allgemeine oder auch unterschiedliche Anforderungen für die Aufzeichnung von Benutzungsschnittstellen aufgezeigt.<sup>14</sup>

1. Das X Window-System als Beispiel für eine grafische Benutzungsschnittstelle, die von Grund auf für den Einsatz in einer verteilten Umgebung eingerichtet ist.
2. Microsoft Windows als Beispiel für eine Benutzungsschnittstelle, die integraler Bestandteil eines Betriebssystems für Desktop- und Personalcomputer (PCs) ist.
3. Das Abstract Windowing Toolkit (AWT) und die Java Foundation Classes (JFC) mit der Swing-Bibliothek als Beispiel für eine Benutzungsschnittstelle, die in die objektorientierte Programmiersprache Java und die zugehörigen Klassenbibliotheken integriert ist und bei deren Konzeption Plattformunabhängigkeit eine zentrale Rolle spielt.

### 2.3.1 Das X Window-System

#### a) Ursprung und Einsatzbereiche

Das oft nur kurz als „X“ bezeichnete X Window-System ist eine Gemeinschaftsentwicklung des Massachusetts Institute of Technology (MIT), der Firma Digital Equipment Corporation (DEC) sowie anderer Firmen und Organisationen und stellt einen weit verbreiteten, nicht proprietären Standard für fensterbasierte Benutzungsschnittstellen dar, auf dem Produkte vieler kommerzieller Anbieter aufbauen.

[NY95] bietet auf S.3ff. einen kurzen Abriss über die Entstehung des X Window-Systems. Dementsprechend wurden die frühen Versionen hauptsächlich in der Forschung eingesetzt. Die im Jahre 1986 erschienene Version 10, Release 4 (kurz X10R4) bildete die Grundlage für erste kommerzielle Entwicklungen. Mit der Version 11 erreichte das System im Wesentlichen den heute noch gültigen Funktionsumfang und bildete faktisch einen Industriestandard als Basis für die Entwicklung von grafischen Benutzungsschnittstellen für Workstations. Die Kontrolle über die Fortentwicklung des X-Standards wird seit 1988 und Version X11R2 von dem X-Consortium ausgeübt, einem Zusammenschluß zahlreicher Anbieter von Hard- und Software für Workstations. Eine Version, auf der auch heute noch zahlreiche Produkte aufbauen, ist X11R5, das 1991 veröffentlicht wurde.

Der X-Standard definiert eine Architektur für ein portables, netzwerkbasiertes System für die Entwicklung von grafischen Benutzungsschnittstellen und stellt Bibliotheken mit grundlegenden Funktionen für deren Realisierung zur Verfügung. X ist als offenes System konzipiert und setzt kein bestimmtes Betriebssystem oder eine bestimmte Hardwarearchitektur voraus. Ebenso wird durch X weder ein konkretes Look and Feel noch eine Menge von Bedienelementen definiert.

---

<sup>14</sup> Anmerkung: In diesem Rahmen wird keine umfassende Beschreibung der Möglichkeiten erfolgen, welche die verschiedenen Systeme dem Anwender bieten. Die Programmierschnittstellen, mit denen sich Entwickler konfrontiert sehen, werden nur in der Übersicht behandelt. Es wird vorausgesetzt, daß der Leser mit graphischen Oberflächen im Allgemeinen gut und mit den genannten Systemen wenigstens oberflächlich vertraut ist. Im Hinblick auf technische Konzepte und die Programmierung wird auf weiterführende Literatur verwiesen.

In der Praxis stellt X die Grundlage für grafische Benutzungsschnittstellen für UNIX-artige Betriebssysteme dar. Verschiedene Hersteller bieten Betriebssysteme an, die auf UNIX bzw. dem POSIX-Standard basieren und dazu eine grafische Benutzungsschnittstelle auf der Grundlage von X. (Bspw. Sun mit Solaris, IBM mit AIX etc.) Die Portabilität des Quellcodes ist damit für zumindest solche Anwendungen gewährleistet, die keine herstellerspezifischen Funktionen verwenden.

Da X zuerst einen technischen Standard aus Sicht des Systementwicklers definiert, werden für die Entwicklung von Anwendungsprogrammen in der Regel Toolkits eingesetzt, die auf einer höheren Abstraktionsebene ein konkretes Look and Feel und Standardelemente zur Verfügung stellen. Die Open Software Foundation (OSF), eine Organisation zahlreicher Hard- und Softwarehersteller, bemüht sich um die Definition von Standards, welche die Interoperabilität zwischen Systemen verschiedener Hersteller erleichtern [HM94]. Motif ist ein OSF-Standard für grafische Benutzungsschnittstellen und definiert sowohl ein konkretes Look and Feel als auch eine Schnittstelle für die Anwendungsentwicklung. Das OSF/Motif-Toolkit ist eine aus diesem Standard hervorgegangene Bibliothek für das X Window-System.

Aufgrund der Konzeption als Client/Server-System und mit UNIX als Betriebssystem wird X heute vielfach auf Workstations eingesetzt, die als Server in Abteilungsnetzwerken oder im Internet dienen, im universitären Bereich sowie, in Form von X-Terminals, an Großrechnern oder an Systemen der mittleren Datenverarbeitung. Im Rahmen der individuellen Datenverarbeitung auf PCs oder Desktopsystemen ebenso wie im Serverbereich ist in den letzten Jahren außerdem eine zunehmende Zahl von X-Installationen unter LINUX, einem frei verfügbaren UNIX-Derivat, zu verzeichnen (vgl. [CT199] S.32).

## b) Das Grundkonzept des X Window-Systems<sup>15</sup>

Das X Window-System ist grundsätzlich auf Netzwerkfähigkeit ausgelegt und basiert auf dem Client/Server-Modell. Abbildung 15, die Figure 1-3, S.9 in [NY95] nachempfunden ist, skizziert den grundlegenden Aufbau.

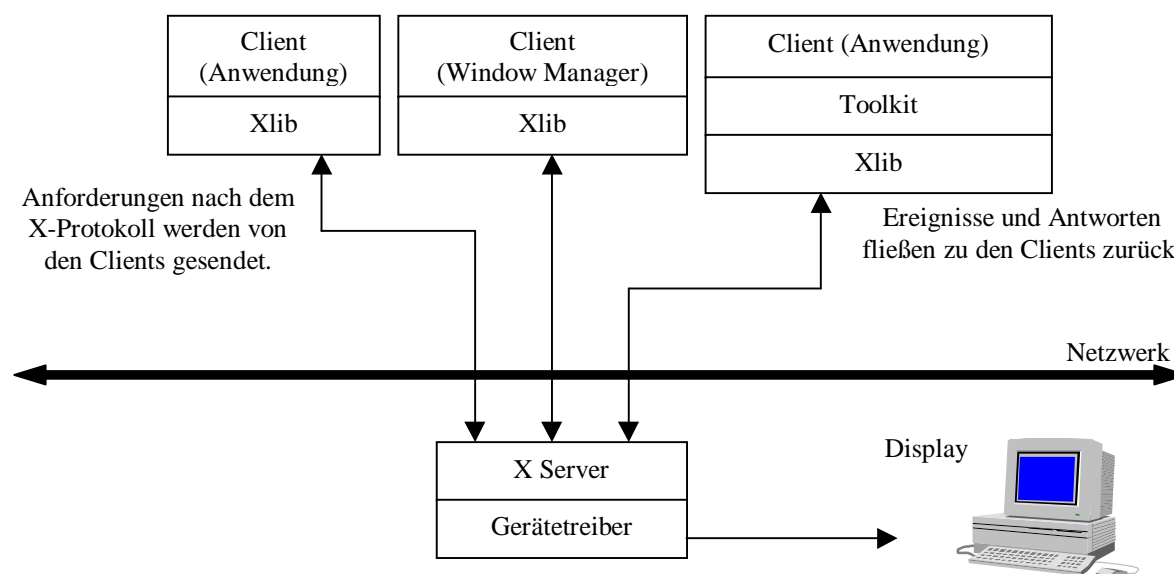


Abbildung 15: Das Grundkonzept des X Window-Systems (Quelle: Figure 1-3, S.9, [NY95])

<sup>15</sup> Anmerkung: Die Aussagen in diesem Abschnitt sind eine kurze Zusammenfassung der einführenden Kapitel aus [NY95] und [NO93].

Ein Display ist in der Terminologie von X ein System, das wenigstens einen grafikfähigen Bildschirm (Screen), eine Tastatur (Keyboard) und ein Zeigegerät (Pointing Device) umfaßt. Das Programm, welches das Display verwaltet und anderen Programmen mit Hilfe der Geräte die Interaktion mit dem Benutzer über eine grafische Benutzungsschnittstelle ermöglicht, ist der X-Server. Anwendungsprogramme, die ein solches Display für die Darstellung ihrer Oberfläche nutzen, sind Clients. Client und Server müssen sich nicht auf demselben Rechner befinden. Die Rolle des Servers kann z.B. ein X-Terminal übernehmen, ein minimal ausgestatteter Rechner, auf dem ausschließlich der Serverprozeß läuft. Der Client kann z.B. ein Programm auf einem Großrechner sein.

Die Kommunikation zwischen Client und Server erfolgt nach dem X-Protokoll, das definiert, wie Anforderungen eines Clients an den Server (Requests), synchrone Antworten des Servers auf Anforderungen (Replies) und asynchrone Nachrichten des Servers an den Client (Events) als Folge von Anforderungen oder von Ereignissen, die der Benutzer auslöst, aussehen.

Die Ein- und Ausgabe von Daten erfolgt über sog. Windows, die durch den X-Server auf dem Bildschirm dargestellt werden. X selbst macht keine genauen Vorgaben, ob und wie Windows automatisch angeordnet werden und welche Standardelemente, bspw. zum Verschieben oder zum Verändern der Größe, sie besitzen. Ein ausgezeichneter Client im X Window-System ist daher der sog. „Window Manager“, der in einem bestimmten Rahmen das Look and Feel der Oberfläche sowie die Organisation der Windows bestimmt.

### c) Die Programmierschnittstellen von X

Der Anwendungsentwickler sieht sich unter X mit wenigstens zwei, in der Regel sogar drei unterschiedlichen Programmierschnittstellen konfrontiert. Abbildung 16, die Figure 1-5, S. 8 in [NO93] nachempfunden ist, illustriert die Softwarehierarchie unter X. Die verschiedenen Schichten übernehmen dabei jeweils folgende Aufgaben:

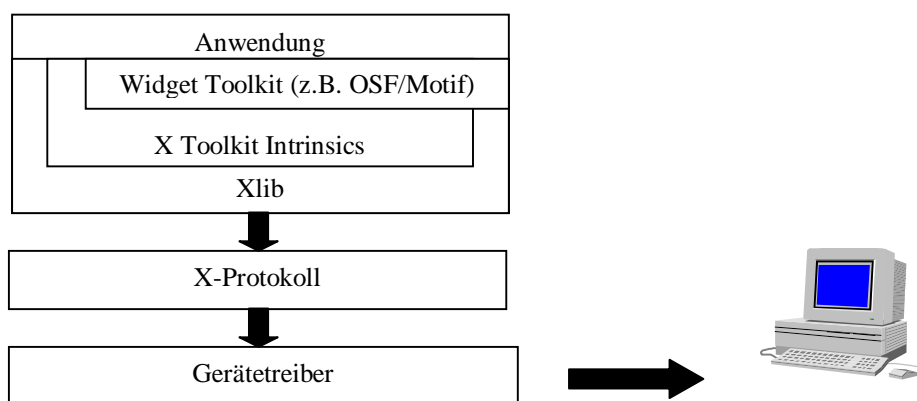


Abbildung 16: Die Softwarehierarchie unter X (Quelle: Figure 1-5, S.8, [NO93])

- **Xlib:** Die Xlib ist die grundlegende Bibliothek des X Window-Systems. Sie stellt Mechanismen für die Client/Server-Kommunikation nach dem X-Protokoll zur Verfügung und enthält Funktionen für die Programmierung von Windows, die auf dieser Ebene praktisch abstrakte Objekte sind, die Bereiche für die Ein- und Ausgabe darstellen. Ereignisse, die im Bereich des Windows mit Hilfe eines Zeigegeräts oder der Tastatur ausgelöst werden, werden in Nachrichten an das entsprechende Programm übersetzt. Die Xlib stellt die Funktionen für die Auswahl, den Empfang und die Verarbeitung dieser Nachrichten zur Verfügung. Ausgaben in Windows erfolgen über Graphic Contexts (GCs), die Windows

zugeordnet werden können. Funktionen für die Verwaltung von GCs sowie die Ausgabe von grafischen Primitiven wie Bitmaps, Linien und Schriften, werden ebenfalls durch die Xlib zur Verfügung gestellt. Detaillierte Informationen zur Xlib finden sich z.B. in [NY95] und [NY92].

- **X Toolkit Intrinsics (Xt):** Die Windows, welche die Xlib zur Verfügung stellt, werden auf einer höheren Abstraktionsebene dazu verwendet, Standardelemente für grafische Oberflächen zu implementieren. Das Hauptfenster einer Anwendung wird ebenso durch ein Window realisiert, wie die Bedienelemente, die auf ihm angeordnet werden. Verschiedene Klassen von Elementen, z.B. Schaltflächen, Textfelder oder Bildlaufleisten, werden in zahlreichen Anwendungen benötigt und daher nicht für jede Anwendung mit Hilfe der Funktionen der Xlib neu implementiert. Solche Windowklassen, die unter X als Widgets bezeichnet werden, stellen für den Anwendungsentwickler die elementaren Einheiten dar, aus denen eine Oberfläche aufgebaut wird. Die Xt-Bibliothek implementiert selbst keine konkreten Widgets, sondern stellt Mechanismen zur Verfügung, mit deren Hilfe Widgets auf der Grundlage der Xlib realisiert werden können. Ein Beispiel für einen solchen Mechanismus sind Callback-Funktionen, mit deren Hilfe Events, die auf der Ebene der Xlib in einer globalen Schleife der Anwendung verarbeitet werden, zu Aktionen auf der Ebene einzelner Bedienelemente zusammengefaßt werden können, die an die Callback-Funktion der jeweiligen Widgetklasse weitergeleitet werden. Bspw. könnten Events vom Typ „Maustaste gedrückt“ und „Maustaste losgelassen“ zu einem Aufruf einer Callback-Funktion vom Typ „Angeklickt“ einer Schaltflächenklasse übersetzt werden. Detaillierte Informationen zu Xt finden sich z.B. in [NO93], [FL93].
- **Widget Toolkits:** Widget Toolkits, wie das Athena Toolkit des MIT, OPEN LOOK von AT&T oder OSF/Motif, bauen auf den X Toolkit Intrinsics auf, um einen konkreten Satz von Standardelementen mit einem durchgängigen Look and Feel zur Verfügung zu stellen. Der X-Standard enthält selbst kein konkretes Toolkit und definiert keine bestimmte Menge von Bedienelementen oder eine bestimmte Programmierschnittstelle aus der Sicht des Anwendungsentwicklers. Der Anwendungsentwickler hat damit die Wahl, auf welchem standardisierten Toolkit die Oberfläche seiner Anwendung aufbauen soll.

#### d) Das Grundgerüst eines systemnahen X-Programms

Abbildung 17 zeigt ein Beispiel für X-Programm, das in dem Sinne systemnah ist, als nur Xlib-Funktionen verwendet werden. Das Programm besteht im Wesentlichen aus folgenden Schritten:

1. Zunächst wird die Verbindung zu einem Display hergestellt, ein Screen ausgewählt und das Hauptfenster der Anwendung geöffnet.
2. Anschließend werden dem Window Manager Daten für die gewünschte Darstellung des Fensters mitgeteilt, z.B. Titel, Symbol für die Minimierung, Größe.
3. Für Ausgaben in das Fenster wird ein Graphics Context erzeugt und verschiedene Darstellungsattribute wie Farben, Schriftarten, Ausgabemodi initialisiert.
4. Anschließend werden die Events maskiert, die über das Fenster empfangen werden sollen.
5. Schließlich werden in einer Schleife Events bearbeitet, die durch Requests der Anwendung oder Aktionen des Benutzers ausgelöst werden.

```

#include <X11/Xlib.h>
#include <X11/Xutil.h>

// Grundgerüst eines Xlib basierten X Window-Programms. (Der Kürze wegen keinerlei
// Fehlerbehandlung. Voraussetzung: Alle Ressourcen können allokiert werden.)
void main(int argc, char **argv)
{
    Display *MyDisplay;
    int MyScreen;
    Window MyWin;
    XTextProperty MyTitle;
    char *Title="Testfenster";
    GC MyGc;
    XGCValues GCValues;
    XFontStruct *Font;
    XEvent Msg;
    unsigned int w,h;

    // Standarddisplay öffnen, Standardscreen ermitteln und ein Fenster erzeugen
    MyDisplay=XOpenDisplay(NULL);
    MyScreen=DefaultScreen(MyDisplay);
    MyWin=XCreateSimpleWindow(MyDisplay,RootWindow(MyDisplay,MyScreen),10,10,300,200,4,
        BlackPixel(MyDisplay,MyScreen),WhitePixel(MyDisplay,MyScreen));
    // Verhandlung mit dem Window Manager (hier nur minimal).
    XStringListToTextProperty(&Title,1,&MyTitle);
    XSetWMProperties(MyDisplay,MyWin,&MyTitle,NULL,argv,argc,NULL,NULL,NULL);
    // Graphics Context erzeugen und Attribute initialisieren.
    MyGc=XCreateGC(MyDisplay,MyWin,0,&GCValues);
    Font=XLoadQueryFont(MyDisplay,"9x15");
    XSetFont(MyDisplay,MyGc,Font->fid);
    XSetForeground(MyDisplay,MyGc,BlackPixel(MyDisplay,MyScreen));
    XSetLineAttributes(MyDisplay,MyGc,15,LineSolid,CapRound,JoinRound);
    // Events, die bearbeitet werden sollen, maskieren und Fenster anzeigen.
    XSelectInput(MyDisplay,MyWin,ExposureMask|ButtonPressMask|StructureNotifyMask);
    XMapWindow(MyDisplay,MyWin);
    while(1) // Hauptschleife
    {
        XNextEvent(MyDisplay,&Msg);
        switch(Msg.type)
        {
            case Expose: // Der Fensterinhalt muß neu gezeichnet werden.
                XDrawRectangle(MyDisplay,MyWin,MyGc,20,20,w-40,h-40);
                XDrawString(MyDisplay,MyWin,MyGc,80,80,"Hallo!",strlen("Hallo!"));
                break;
            case ConfigureNotify: // Konfiguration des Fensters hat sich geändert.
                w=Msg.xconfigure.width;
                h=Msg.xconfigure.height;
                break;
            case ButtonPress: // Maustaste wurde gedrückt.
                XUnloadFont(MyDisplay,Font->fid);
                XFreeGC(MyDisplay,MyGc);
                XCloseDisplay(MyDisplay);
                exit(1);
        }
    }
}

```

Abbildung 17: Das Grundgerüst eines systemnahen X-Programms

### e) Ansatzpunkte für die Aufzeichnung der Benutzungsschnittstelle

Reale Anwendungen bauen, wie in c) geschildert, meist nicht direkt auf den Funktionen der Xlib auf, sondern nutzen Toolkits, die ihrerseits die Funktionen der Xt-Bibliothek verwenden und Abstraktionen für die unter d) genannten Abschnitte eines Programms anbieten. Bei der Aufzeichnung der Benutzungsschnittstelle müssen daher ggf. sowohl Low-Level-Daten, z.B. Events nach dem X-Protokoll, als auch Informationen auf der Ebene der Widgets berücksichtigt werden. Ansatzpunkte sind demzufolge der X-Nachrichtenstrom selbst und, abhängig vom verwendeten Toolkit, Callback-Funktionen.

## 2.3.2 Microsoft Windows

### a) Ursprung und Einsatzbereiche

Das von der Firma Microsoft entwickelte Microsoft Windows ist die grafische Benutzungsschnittstelle mit den meisten Installationen und Marktführer im Bereich der Desktop- und Personalcomputer. Umgangssprachlich wird oft schlicht „Windows“ gesagt, wenn Microsoft Windows gemeint ist. Microsoft Windows V1.0 erschien im Jahre 1983 [GA95].

Windows wird hauptsächlich auf Rechnern eingesetzt, die Prozessoren verwenden, die auf der INTEL 8086 Prozessorfamilie basieren. Die frühen Versionen, bis Windows 3.11, die bis Mitte der 90er angeboten wurden, stellten eine grafische Benutzungsschnittstelle zunächst nur in Form eines Aufsatzes für das eigentliche Betriebssystem MS-DOS zur Verfügung. Im Verlauf der Entwicklung wurden mehr und mehr Betriebssystemdienste integriert. Parallel zu Windows als Betriebssystem für Desktop- und Einzelplatzrechner steht seit den frühen 90er Jahren mit Microsoft Windows NT auch ein Betriebssystem für Workstations zur Verfügung, von dem auch Implementierungen für Alpha und MIPS Prozessoren existieren. Die aktuellen Versionen Windows 95/98 und NT 4.0 integrierten eine grafische Benutzungsschnittstelle vollständig in das Betriebssystem und führten das aktuelle Look and Feel und die Programmierschnittstelle Win32 API ein, die es bei Berücksichtigung einiger Besonderheiten erlaubt, Programme zu schreiben, die unter beiden Systemen, Windows 95/98 und NT lauffähig sind. Bei den folgenden Betrachtungen werden die älteren Versionen nicht mehr berücksichtigt.

Im Rahmen der zukünftigen Entwicklung sollen grafische Oberflächen mit dem für Microsoft Windows typischen Look and Feel auch vermehrt für embedded Systems<sup>16</sup> zur Verfügung gestellt werden. Windows CE soll dem Anwendungsentwickler hier die Möglichkeit bieten, die Entwicklungszeiten durch die Nutzung bekannter Schnittstellen und Entwicklungsumgebungen zu verkürzen. Dem Benutzer soll die intuitive Bedienung unterschiedlicher Geräte durch eine einheitliche und dann jedem bekannte Benutzungsschnittstelle ermöglicht werden. Eine übersichtliche Einführung in Windows CE bietet [RI97].

### b) Die Integration der grafischen Benutzungsschnittstelle in das Betriebssystem

Die Tatsache, daß Microsoft Windows ursprünglich für Personalcomputer entwickelt wurde, spiegelt sich in der Systemarchitektur wieder. Mit dem PC kam im Verlauf der 80er Jahre eine Form der individuellen Datenverarbeitung auf, bei der sich Daten, Programme, eine grafische Benutzungsschnittstelle und ein Betriebssystem, das diese Komponenten integriert, auf demselben Rechner, einem Einzelplatzsystem, befinden. Die Vernetzung in einem LAN ist ein Konzept, das prinzipiell durch dasselbe Betriebssystem geleistet werden kann, hier jedoch, im Gegensatz zum X Window-System, unabhängig von der grafischen Benutzungsschnittstelle zu sehen ist.

Bei Windows 95/98 und NT ist die grafische Benutzungsschnittstelle fest integriert. Das Fenstermanagement sowie die Funktionen für die grafische Ausgabe sind keine austauschbaren Dienste und werden über die allgemeine Programmierschnittstelle, das Application Programming Interface (API) des Betriebssystems angesprochen. Abbildung 18 skizziert den Aufbau. Die einzelnen Komponenten werden im Folgenden näher beschrieben.

---

<sup>16</sup> Anmerkung: Unter einem embedded System ist in diesem Zusammenhang ein Computersystem zu verstehen, das in ein Gerät des täglichen Gebrauchs integriert ist. Typische Beispiele sind Videorecorder oder TV-Geräte, die heute in der Regel über Bildschirmmenüs programmiert werden können. Ambitionierte Prognosen sprechen von Mikrowellengeräten und Waschmaschinen mit Internetzugang.

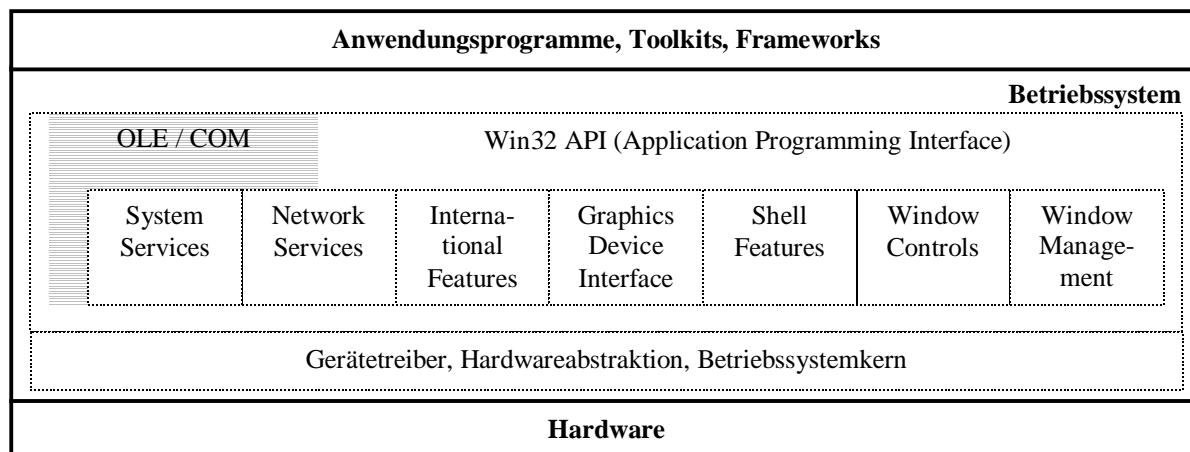


Abbildung 18: Die Integration verschiedener Dienste im Betriebssystem Microsoft Windows

### c) Die Programmierschnittstelle Win32 API

Die originäre Schnittstelle für die Programmierung der grafischen Oberfläche unter Microsoft Windows ist die Win32 API, die gleichzeitig auch die Schnittstelle für die Programmierung der übrigen Betriebssystemdienste bildet. [SDK-A] zufolge kann die Win32 API in folgende Funktionsbereiche untergliedert werden. Die Funktionen welche die grafische Oberfläche betreffen, verteilen sich auf mehrere dieser Bereiche.

- **Window Management:** Funktionen für die Erzeugung und Verwaltung von Fenstern, die für die Interaktion mit dem Benutzer verwendet werden.
- **Window Controls:** Funktionen für die Programmierung bestimmter Standardelemente der grafischen Benutzungsschnittstelle, z.B. von Baumansichten, Listenansichten usw..
- **Shell Features:** Funktionen für eine abstrakte, auf einem Objektbegriff basierende Sicht auf Dokumente, Geräte und Dienste.
- **Graphics Device Interface:** Funktionen für die grafische Ausgabe, die von konkreten Ausgabegeräten wie Bildschirmen und Druckern abstrahieren.
- **System Services:** Funktionen zum Zugriff auf Speicher, Prozesse, Dateien und andere Systemressourcen.
- **International Features:** Funktionen zur Berücksichtigung landesspezifischer Formate, bspw. bei Datumsangaben, Währungen, Zeichensätzen etc.
- **Network Services:** Funktionen für die Verwaltung gemeinsamer Ressourcen und von Verbindungen im Netzwerk.

Aufgrund der verwendeten Datentypen und der Entwicklerdokumentation kann C/C++ als die „natürliche“ Sprache für die Programmierung von Microsoft Windows auf der Systemebene betrachtet werden, wobei für die Programmierung der Win32 API unmittelbar keine objektorientierten Konzepte dieser Sprache zum Einsatz kommen. Für die Anwendungsentwicklung werden darüber hinaus häufig Klassenbibliotheken und Toolkits, z.B. Microsoft Foundation Classes, Visual Basic, Borland Delphi, eingesetzt, die objektorientierte oder anwendungsnahe Abstraktionen von der systemnahen Win32 API anbieten.

Eine weitere in das System integrierte Schnittstelle für die objektorientierte Programmierung wird durch das Component Objekt Model (COM) und Objekt Linking and Embedding (OLE) zur Verfügung gestellt. COM definiert eine allgemeine Architektur für die Verwendung von Components. Der COM-Standard wird in [MD95] beschrieben. OLE stellt unter Windows die konkreten Mechanismen für die Realisierung von Components sowie eine auf dem Objektbegriff basierende Interprozeßkommunikation zur Verfügung. Für die direkte Programmierung der grafischen Benutzungsschnittstelle müssen OLE und COM nicht unbedingt näher betrachtet werden.<sup>17</sup> Eine Ausführliche Beschreibung des OLE Konzeptes liefert [BR95].<sup>18</sup>

#### **d) Das Grundgerüst einer fensterbasierten Anwendung unter Microsoft Windows**

Die grundlegende Einheit der Win32 API für die Erstellung der Oberfläche eines Anwendungsprogramms ist das Window. Jedes Window besitzt eine Reihe von Attributen, deren Werte teilweise bei der Erzeugung, teilweise explizit durch das Anwendungsprogramm und teilweise implizit durch den Programmverlauf und die Interaktion mit dem Benutzer gesetzt und verändert werden. Die Attribute beschreiben unter anderem

- den Zustand des Fensters, ob es sichtbar oder unsichtbar, aktiv oder im Hintergrund ist.
- die Geometrie des Fensters, bestehend aus der Größe und der Position relativ zu einem übergeordneten Fenster oder zum Ursprung des Bildschirms.
- Standardelemente und das grundlegende Aussehen, wie z.B. Rahmen, Titelleiste und Schaltflächen, über die der Benutzer die Geometrie des Fensters verändern kann.

Bei der Erzeugung eines Windows kann neben den Ausgangswerten für einige Attribute, z.B. die Geometrie des Fensters, auch ein übergeordnetes Fenster (Parent) angegeben werden, so daß im Programmverlauf eine Hierarchie von Fenstern entsteht. Jedes Window gehört darüber hinaus zu einer Klasse, die ebenfalls bei der Erzeugung angegeben wird. Die Klasse bestimmt verschiedene Ausgangswerte für die Attribute des Windows sowie die Windowprozedur, eine Callback-Funktion, in der an das Window gesendete Nachrichten verarbeitet werden.

Abbildung 19 zeigt das Grundgerüst einer fensterbasierten Anwendung unter Microsoft Windows, das im Wesentlichen aus den folgenden Elementen besteht:

1. Der Registrierung einer Windowklasse für das anwendungsspezifische Hauptfenster.
2. Der Erzeugung eines entsprechenden Windows, auf dem untergeordnete Windows angelegt werden, die zu Standardklassen von Bedienelementen gehören.
3. Einer Schleife zum Abholen der anstehenden Nachrichten.
4. Den Windowprozeduren, welche die eigentliche Verarbeitung der Programmlogik aufgrund von Nachrichten realisieren.

---

<sup>17</sup> Anmerkung: Bemerkenswert mag in diesem Zusammenhang sein, daß neuere Dienste, die unter dem Begriff DirectX zusammengefaßt werden, nur über COM-Objekte und nicht über „normale“ Win32 API Funktionen programmiert werden können. DirectX ermöglicht unter Umgehung der Standardoberfläche über eine neue bzw. zusätzliche Abstraktionsschicht den Zugriff auf die Sound- und Grafikhardware, z.B. für Unterhaltungssoftware oder Multimediaanwendungen.

<sup>18</sup> Anmerkung: Die Begriffe OLE und COM werden in der Literatur zu Microsoft Windows meist nicht scharf getrennt. Entsprechend [BR95] wurde die erste OLE Version 1991 für die Realisierung von Verbunddokumenten entworfen. COM wurde erst später und eher in Form eines formalen Standards definiert. Im Rahmen der Erweiterung der OLE-Architektur wurden dann die COM-Konzepte durch OLE implementiert.

```

#include <windows.h>

HWND hMyWnd,hMyButton; // Window-Handles

// Windowprozedur für die Verarbeitung von Nachrichten, die an Windows
// vom Typ der anwendungsspezifischen Klasse 'MyWndClass' gesendet werden.
LRESULT CALLBACK MyWindowProc(HWND hwnd,UINT uMsg,WPARAM wParam,LPARAM lParam)
{
    switch(uMsg)
    {
        case WM_COMMAND: // Nachricht von einem untergeordneten Control?
            if(HIWORD(wParam)==BN_CLICKED && (HWND)lParam==hMyButton)
                MessageBox(hwnd,"Hello World!","Angeklickt:",MB_OK);
            return 0;
        case WM_DESTROY: // Das Fenster wurde geschlossen?
            PostQuitMessage(0); // Führt zur Beendigung der GetMessage-Schleife in WinMain.
            return 0;
        default: // Übrige Nachrichten vom Betriebssystem bearbeiten lassen.
            return DefWindowProc(hwnd,uMsg,wParam,lParam);
    }
}

// Hauptprozedur des Programms
int APIENTRY WinMain(HINSTANCE hInst,HINSTANCE hPrevInst,LPSTR lpCmdLine,int nCmdShow)
{
    WNDCLASS MyClass;
    MSG Msg;

    // Initialisierung und Registrierung der anwendungsspezifischen Windowklasse.
    memset(&MyClass,0,sizeof(WNDCLASS));
    MyClass.lpfWndProc=MyWindowProc;
    MyClass.hInstance=hInst;
    MyClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
    MyClass.hCursor=LoadCursor(NULL,IDC_ARROW);
    MyClass.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
    MyClass.lpszClassName="MyWndClass";
    if(RegisterClass(&MyClass))
    {
        // Erzeugung des Hauptfensters und einer Schaltfläche
        if((hMyWnd=CreateWindow("MyWndClass","Testfenster",WS_VISIBLE|WS_OVERLAPPEDWINDOW,
            10,10,480,200,NULL,NULL,hInst,NULL))!=NULL)
        {
            hMyButton=CreateWindow("BUTTON","Hier klicken",WS_VISIBLE|WS_CHILD|BS_PUSHBUTTON,
                10,10,100,40,hMyWnd,NULL,hInst,NULL);

            // GetMessage-Schleife, die ausgeführt wird, bis das Hauptfenster geschlossen wird.
            while(GetMessage(&Msg,NULL,0,0))
            {
                TranslateMessage(&Msg);
                DispatchMessage(&Msg);
            }
        }
    }
    return 0;
}

```

Abbildung 19: Das Grundgerüst einer fensterbasierten Anwendung unter Microsoft Windows

Das Betriebssystem stellt selbst bereits eine Reihe von Windowklassen und entsprechende Windowprozeduren für Standardelemente zur Verfügung. Letztendlich wird das Verhalten und damit der Einsatzbereich jeder Windowklasse durch die Windowprozedur bestimmt, die den Code für die Verarbeitung der Nachrichten enthält. Nachrichten werden unter anderem

- vom Betriebssystem an ein Window gesendet, wenn bestimmte Ereignisse durch den Benutzer ausgelöst werden. Bspw. wenn die Maustaste gedrückt wird, während sich der Mauszeiger über dem Window befindet.
- von der Anwendung an sich selbst gesendet, um den Zustand eines Windows aufgrund der Programmlogik zu ändern, z.B. um eine Schaltfläche sichtbar/unsichtbar zu machen.

- von untergeordneten Windows an ihre Parents gesendet, wenn eine bestimmte Folge von Ereignissen von dem untergeordneten Fenster verarbeitet wurde. Bspw. sendet ein Window der Klasse „BUTTON“ eine Nachricht vom Typ „BN\_CLICKED“ an sein Parent, wenn die Maustaste über dem Window gedrückt und wieder losgelassen wurde.
- als kaskadierende Folge ausgelöst, wenn komplexere Vorgänge, die in Standardpakete zerlegt werden können, verarbeitet werden müssen. Bspw. zieht die Erzeugung eines Windows unter anderem Nachrichten zur Änderung der Größe sowie zum Neuzeichnen des Windows nach sich. Beide Nachrichten könnten jedoch auch in einem anderen Kontext einzeln, bspw. durch Aktionen des Benutzers, ausgelöst werden.

### e) Ansatzpunkte für die Aufzeichnung der Benutzungsschnittstelle

Die Aufzeichnung der Benutzungsschnittstelle kann im Wesentlichen auf den Prozeduren zur Nachrichtenverarbeitung aufsetzen, wobei das Betriebssystem bereits zwei Mechanismen zur Unterstützung anbietet:

- **Subclassing:** Für Windows, die zu Standardklassen oder zu Toolkits gehören, kann die Technik des Subclassings verwendet werden, um eine zusätzliche Windowprozedur zu installieren, die anstatt der original Windowprozedur aufgerufen wird. In dieser Prozedur könnte die Auswahl und Aufzeichnung relevanter Nachrichten erfolgen. Anschließend werden die Nachrichten zur Verarbeitung an die ursprüngliche Windowprozedur weitergeleitet. Eine Detaillierte Beschreibung dieses Mechanismus liefert [MA94].
- **Hooks:** Mit Hilfe eines Hooks können Prozeduren zur fensterübergreifenden Überwachung von Nachrichten installiert werden. Microsoft Windows stellt verschiedene Typen von Hooks zur Verfügung. Für die Aufzeichnung von Anwendungssitzungen geeignet ist z.B. der JournalRecordHook, der die Maus- und Tastaturereignisse einer Anwendung überwacht sowie dessen Pendant, der JournalPlaybackhook, der verwendet werden kann, um solche Nachrichten in einer Form wieder abzuspielen, die es dem Betriebssystem erlaubt, sie so zu verarbeiten, als würden sie vom Benutzer ausgelöst. Die beiden genannten, ebenso wie weitere Typen von Hooks werden in [SDK-H] ausführlich beschrieben.

Ein auf diesen Techniken basierendes Konzept für die Aufzeichnung der grafischen Benutzungsschnittstelle unter Microsoft Windows könnte mit folgenden drei wesentlichen Punkten für die Datenaufnahme realisiert werden:

1. Bei der Erzeugung von Windows werden Daten aufgezeichnet, die bspw. von einem Runtime-Player verwendet werden können, um die Oberfläche zu rekonstruieren.
2. Ein Hook zeichnet global für die Anwendung Low-Level-Ereignisse auf, z.B. Bewegungen der Maus oder Tastaturereignisse.
3. Als Reaktion auf spezifische oder anwendungsnahe Nachrichten werden bei Bedarf in den entsprechenden Windowprozeduren zusätzliche semantische Informationen aufgezeichnet. Für Standardelemente von Windows oder Bedienelemente die durch Toolkits zur Verfügung gestellt werden, kann Subclassing eingesetzt werden.<sup>19</sup>

---

<sup>19</sup> Anmerkung: Im Gegensatz zum X Window-System (vgl. Abschnitt 2.3.1) wird durch die Win32 API ein Look and Feel vorgegeben. Toolkits kapseln Standardelemente, ermöglichen deren vereinfachte oder objektorientierte Verwendung oder fügen ihnen erweiterte Funktionen hinzu. Sie ersetzen jedoch häufig die Standardelemente nicht, sondern verwenden sie. Der Zugriff auf die Windows solcher erweiterter Bedienelemente kann entweder durch das Toolkit selbst ermöglicht werden. (Bei den Microsoft Foundation Classes und Visual Basic ist dies z.B. der Fall.) Andernfalls kann die Hierarchie der Windows mit Hilfe entsprechender Win32 API Funktionen analysiert werden, um das Subclassing der Bedienelemente zu ermöglichen.

### 2.3.3 Java, das Java-AWT und die Java Foundation Classes

#### a) Ursprung und Einsatzbereiche

[NE+97] zufolge begann die Entwicklung, die schließlich zu Java führte, Anfang der 90er Jahre bei der Firma Sun Microsystems mit dem Ziel, die Entwicklung von embedded Systems zu vereinfachen.<sup>20</sup> Seit 1995 steht mit Java ein System zur Verfügung, das entsprechend [NE+97] S.28ff. folgendermaßen skizziert werden kann:<sup>21</sup>

- Java stellt eine objektorientierte Programmiersprache dar, deren Syntax an C++ angelehnt ist und die damit für eine große Zahl von Programmierern einfach zu erlernen ist. Java ist jedoch einfacher als C++ gehalten und vermeidet einige Nachteile dieser Sprache.<sup>22</sup>
- Durch das Konzept einer virtuellen Maschine (Java Virtual Machine), die den Bytecode interpretiert, zu dem Java-Programme kompiliert werden, sind Java-Programme portierbar und architekturunabhängig.
- Durch die umfangreichen, standardmäßig vorhandenen Bibliotheken, die durch die Firma Sun mit den Java Development Kits (JDKs) ausgeliefert werden, steht von Anfang an eine leistungsfähige, standardisierte API für die Anwendungsentwicklung zur Verfügung.

Eine kommandozeilenorientierte Entwicklungsumgebung für die Betriebssysteme Microsoft Windows und Sun Solaris, die Standardbibliotheken und umfangreiches Informationsmaterial werden von Sun im Internet unter <http://java.sun.com> angeboten.

Ein wesentlicher Einsatzbereich für Java ist das World Wide Web. Java-Applets, d.h. in HTML-Seiten referenzierte Java-Klassen, werden hier verwendet, um interaktive Elemente in Webseiten zu integrieren und verteilte Anwendungen zu realisieren. Beispiele für den Einsatz von Java-Applets finden sich im Bereich Home-Banking und E-Commerce.

Weitere, von Sun geförderte Einsatzbereiche sind bspw. sog. Netzcomputer, die ähnlich wie klassische Terminals von Großrechnern, als Terminals in Intranets (Firmennetzwerken, die auf Internettechnologie basieren) betrieben werden und bei denen Java als Betriebssystem fungieren kann [SUN98], oder embedded Systems, die Java-Technologie nutzen [SUN99].

#### b) Konzepte für eine plattformunabhängige Benutzungsschnittstelle

Wie unter a) bereits geschildert, ist Java grundsätzlich auf Plattformunabhängigkeit ausgelegt. Mit Bezug auf die grafische Benutzungsschnittstelle wurden die Techniken, mit deren Hilfe dies erreicht wird, mit den verschiedenen Java Development Kits (JDKs) weiter entwickelt.

Bereits seit dem JDK 1.0 steht mit dem Abstract Windowing Toolkit (AWT) eine Bibliothek zur Verfügung, mit der Benutzungsschnittstellen erzeugt werden können. Das AWT enthält eine Reihe von Klassen, die Standardelemente wie Schaltflächen, Kontrollkästchen, Listen, Rahmenfenster usw. implementieren, und dabei die unter dem jeweiligen Betriebssystem vorhandenen Bedienelemente kapseln.

<sup>20</sup> Anmerkung: [NE+97] liefert auf S.18ff. eine recht launige Abhandlung über die Geschichte von Java. Bspw. wird dort dementiert, daß JAVA für „Just Another Vague Acronym“ steht.

<sup>21</sup> Anmerkung: An dieser Stelle soll keine ausführliche Erklärung der genannten Konzepte erfolgen. Weitere, einführende Informationen bieten bspw. [NE+97] und [STE98].

<sup>22</sup> Anmerkung: Als Nachteile von C++ werden in [NE+97] bspw. genannt: Mehrfachvererbung, Überladung von Operatoren, ausgedehnte automatische Konvertierung von Datentypen, Zeigerarithmetik.

Für jedes Element, das als AWT-Komponente erzeugt wird, existiert ein entsprechendes „echtes“ Element, ein sog. „Peer“ des Betriebssystems unter dem die Java Virtual Machine gerade läuft. Methodenaufrufe der AWT-Klassen werden in Funktionen des Betriebssystems, die auf den Peer angewandt werden, übersetzt. AWT-Komponenten gehorchen demzufolge dem Look and Feel der Oberfläche des Betriebssystems. Eine Schaltfläche sieht unter Windows wie eine Windows-Schaltfläche und unter Solaris wie eine Motif-Schaltfläche aus.

Ein anderer Weg wird mit den sog. „Swing Components“ der Java Foundation Classes (JFC) beschritten, die zunächst als Zusatzbibliothek zum JDK 1.1 zur Verfügung standen und die seit dem JDK 1.2 ein fester Bestandteil der Standardbibliotheken sind. Swing-Komponenten werden als sog. „Lightweight Components“ bezeichnet. Damit wird zum Ausdruck gebracht, daß sie kein gegenüber in einem Bedienelement des Betriebssystems, keinen Peer besitzen, sondern vollständig in Java implementiert sind. Demzufolge wird das Look and Feel von Swing-Komponenten nicht durch das Betriebssystem, unter dem die Java Virtual Machine läuft, bestimmt. Diese Freiheit wird an die Anwendungsprogramme weitergegeben, indem ein sog. „plugable Look and Feel“ zur Verfügung gestellt wird. D.h. Anwendungsprogramme können ihr Look and Feel frei bestimmen. Es kann unabhängig vom Betriebssystem eines der vorgefertigten Look and Feels verwendet werden (Windows, Motif oder Metal). Es kann das Look and Feel verwendet werden, das dem Betriebssystem entspricht, unter dem die Java Virtual Machine gerade läuft, oder es kann ein eigenes Look and Feel realisiert werden, indem die entsprechenden Klassen überschrieben oder neu implementiert werden.

Beide Konzepte, AWT-Komponenten und Swing, existieren im JDK 1.2 parallel. Komplexere Elemente wie Listenansichten (Tables) oder Baumansichten (Trees) stehen jedoch nur durch die Swing-Bibliothek zur Verfügung. Ein Problem stellt dabei zur Zeit noch dar, daß Internet-Browser wie der Microsoft Internet Explorer und der Netscape Navigator in ihren aktuellen Versionen Swing nicht direkt unterstützen. Der Anwender muß sich hier mit sog. Plug-ins behelfen, die von Sun zur Verfügung gestellt werden.<sup>23</sup>

### c) Die API zur Programmierung der Benutzungsschnittstelle

Von der umfangreichen API zur Programmierung der Benutzungsschnittstelle sollen hier zwei Teilbereiche kurz skizziert werden: Die Klassenhierarchie der Bedienelemente und das Modell der Nachrichtenverarbeitung. Eine Beschreibung aller Klassen und Methoden enthält die Dokumentation des JDK, die unter <http://java.sun.com> im Internet zur Verfügung steht. Eine übersichtliche Einführung in die Java-Programmierung und die grundlegenden Konzepte bietet [CW98]. Eine tabellarische Übersicht der Klassen und Methoden des JDK 1.2 sowie eine Liste mit Querverweisen befindet sich in [CH98]. Die Programmierung der JFC/Swing-Komponenten wird sehr anschaulich und anhand vieler Beispiele in [WA98] beschrieben.

AWT- und Swing-Komponenten werden im Folgenden gemeinsam behandelt, wobei die Swing-Bibliothek teilweise auf AWT-Klassen aufbaut oder diese verwendet. Die Unterteilung von Bibliotheken bzw. die Gruppierung von thematisch zusammengehörigen Klassen erfolgt in Java in Form von Paketen. Die Klassen des AWT befinden sich im Paket `java.awt` und dazu untergeordneten Paketen. Die Swing-Klassen befinden sich im Paket `com.sun.java.swing` bzw. in `javax.swing` und den jeweils dazu untergeordneten Paketen.<sup>24</sup>

<sup>23</sup> Anmerkung: In diesem Zusammenhang kann man entweder mit Humor oder mit Bedauern sehen, daß z.T. Plattformabhängigkeit gegen Versionsabhängigkeit eingetauscht werden muß und sich Versionen noch schneller ändern als die Plattformen in früheren Zeiten. Dieses Problem besteht leider bei vielen der sich schnell entwickelnden „Standards“ im Umfeld des World Wide Web und des Internet.

<sup>24</sup> Anmerkung: Der Pfad der Pakete der Swing-Bibliothek wurde im Verlauf der Entwicklung von Sun geändert. Wird Swing als Zusatzbibliothek zum JDK 1.1 verwendet, werden die Klassen über `com.sun.java.swing` referenziert, im JDK 1.2 über `javax.swing`.

## Die Klassenhierarchie der Bedienelemente

Abbildung 20 zeigt einen Ausschnitt der Klassenhierarchie der Komponenten, die unter Java für die Erzeugung von Bedienelementen, Dialogfeldern, Fenstern usw. zur Verfügung stehen.<sup>25</sup> Die Swing-Klassen, deren Bezeichnungen mit einem „J“ beginnen (abgesehen von AbstractButton) sind z.T. von den entsprechenden AWT-Komponenten abgeleitet, wie bspw. JFrame von Frame, ansonsten handelt es sich um Unterklassen von JComponent.

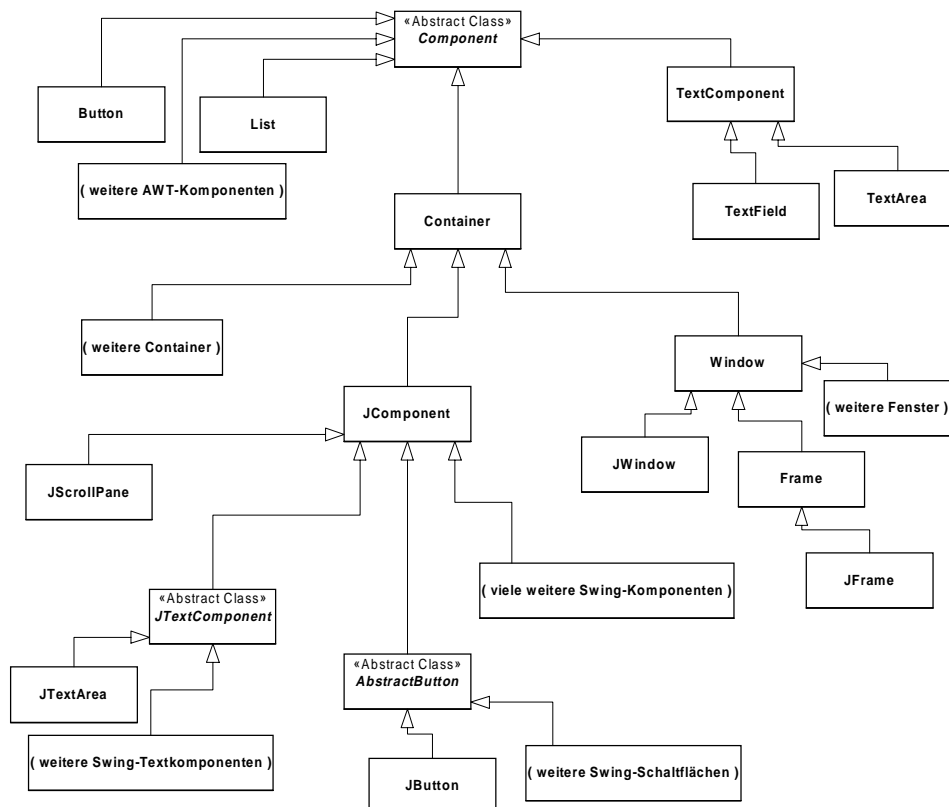


Abbildung 20: Die Klassenhierarchie der Java-Komponenten

Die Verwendung der Klassen erfolgt in Anwendungsprogrammen entweder direkt, d.h. eine Anwendung erzeugt bspw. einen Frame und fügt diesem Schaltflächen und andere Elemente hinzu, oder durch die Erweiterung und Ableitung anwendungsspezifischer Klassen.

## Das Modell der Nachrichtenverarbeitung unter Java

Nachrichten an Komponenten können sowohl durch den Benutzer über Eingabegeräte (z.B. die Maus oder die Tastatur) ausgelöst werden, als auch durch bestimmte programminterne Verarbeitungsschritte, z.B. das Erzeugen neuer Schaltflächen auf einem Fenster.

Nachrichten werden in Form von Ereignisobjekten an diejenige Komponente gemeldet, über der das auslösende Ereignis stattfand. Unterschiedliche Ereignistypen werden dabei durch verschiedene Unterklassen der Klasse EventObject beschrieben. Bspw. beschreiben Objekte vom Typ MouseEvent Bewegungen der Maus oder KeyEvent Tastaturereignisse. Programme, die über einen bestimmten Nachrichtentyp informiert werden möchten, implementieren einen sog. „Listener“ und registrieren diesen bei den betroffenen Komponenten.

<sup>25</sup> Anmerkung: Die Darstellung basiert auf dem JDK 1.2. Im JDK 1.1 ist bspw. die Klasse Container noch eine abstrakte Klasse. Als Quelle dienten die Quellcodes, die mit den jeweiligen JDKs von Sun ausgeliefert wurden.

Für die verschiedenen Typen von Nachrichten sind durch die API Listener-Schnittstellen vorgegeben, die Methoden definieren, die von den Klassen implementiert werden müssen, die als Listener fungieren sollen. Die Verarbeitung von Nachrichten erfolgt, indem Ereignisobjekte als Parameter an die zu dem jeweiligen Ereignis gehörende Methode derjenigen Listener-Objekte übergeben werden, die für den Nachrichtentyp registriert sind. Dieses Modell der Nachrichtenverarbeitung wird als „Delegation Event Model“ bezeichnet.<sup>26</sup> Eine ausführliche Beschreibung erfolgt bspw. in [WA98] S.19ff..

#### d) Das Grundgerüst einer fensterbasierten Java-Anwendung

```
// Grundgerüst einer fensterbasierten Java-Anwendung. Der Kürze wegen keine Fehlerbehandlung.
import java.awt.*;
import java.awt.event.*;

// Hauptklasse der Anwendung
public class SimpleApp extends Frame implements ActionListener {

    // Konstruktor-Methode
    public SimpleApp() {
        // Schaltfläche erzeugen und SimpleApp-Objekt als ActionListener registrieren.
        Button HalloButton=new Button("Hallo Welt!");
        HalloButton.addActionListener(this);
        // Anonyme, von WindowAdapter abgeleitete Klasse als WindowListener registrieren.
        addWindowListener(new WindowAdapter() {
            // Wird aufgerufen, wenn das Fenster geschlossen werden soll.
            public void windowClosing(WindowEvent event) {
                System.exit(0); // Beendet das Programm.
            }
        });
        // MyListener-Objekt erzeugen und als MouseMotionListener registrieren.
        addMouseMotionListener(new MyListener());
        // Attribute des Frames setzen, Schaltfläche hinzufügen und Frame anzeigen.
        this.setLayout(new FlowLayout());
        add(HalloButton);
        setTitle("Testanwendung");
        setBounds(10,10,200,60);
        setVisible(true);
    }

    // Implementiert das ActionListener-Interface. Wird aufgerufen, wenn ein
    // Bedienelement vom Benutzer "angeklickt" wurde.
    public void actionPerformed(ActionEvent event) {
        System.out.println(event.getActionCommand()+ " angeklickt");
    }

    // Klassenmethode als Hauptprozedur der Anwendung
    public static void main(String[] args) {
        new SimpleApp();
    }
}

// Anwendungsspezifische Listener-Klasse, die MouseMotionListener implementiert.
class MyListener implements MouseMotionListener {
    // Wird aufgerufen, wenn die Maus mit gedrückter Taste bewegt wurde.
    public void mouseDragged(MouseEvent event) {
        System.out.println("Dragged, X: "+event.getX()+" Y: "+event.getY());
    }
    // Wird aufgerufen, wenn die Maus ohne gedrückte Taste bewegt wurde.
    public void mouseMoved(MouseEvent event) {
        System.out.println("Moved, X: "+event.getX()+" Y: "+event.getY());
    }
}
```

Abbildung 21: Das Grundgerüst einer fensterbasierten Java-Anwendung

<sup>26</sup> Anmerkung: Das Delegation Event Model steht erst seit JDK 1.1 zur Verfügung. Das Ereignismodell des JDK 1.0, bei dem die Nachrichtenverarbeitung nur über Methoden der Komponenten möglich war, gilt als veraltet und sollte nicht mehr verwendet werden. Die Swing-Komponenten erfordern das Delegation Event Model.

Abbildung 21 zeigt das Grundgerüst einer fensterbasierten Java-Anwendung, das aus folgenden Abschnitten besteht:

1. Die Hauptklasse der Anwendung, SimpleApp erweitert die Klasse Frame und stellt damit ein anwendungsspezifisches Rahmenfenster zur Verfügung.
2. Die Klassenmethode „main“ dient als Startpunkt für die Ausführung des Programms. Beim Start der Anwendung wird ein SimpleApp-Objekt erzeugt.
3. Im Konstruktor der SimpleApp-Klasse wird die Benutzungsschnittstelle initialisiert. Es wird eine Schaltfläche erzeugt, der das SimpleApp-Objekt als ActionListener übergeben wird. Eine anonyme, von WindowAdapter abgeleitete Klasse wird als WindowListener verwendet, um die Anwendung zu beenden, wenn das Hauptfenster geschlossen wird.<sup>27</sup> Außerdem wird ein MyListener-Objekt als MouseMotionListener an das SimpleApp-Objekt übergeben. Schließlich werden weitere Attribute des Fensters, wie das Layout der Schaltflächen, die Größe und der Titel gesetzt und das Fenster angezeigt.
4. Durch die Methode actionPerformed wird das ActionListener-Interface implementiert. Da im Konstruktor das SimpleApp-Objekt selbst als ActionListener an die Schaltfläche übergeben wird, erfolgt ein Aufruf dieser Methode, wenn die Schaltfläche angeklickt wurde.
5. Die Klasse MyListener stellt eine Implementierung des MouseMotionListener-Interfaces zur Verfügung. Da im Konstruktor ein Objekt vom Typ MyListener als Listener für den Frame, der durch die Klasse SimpleApp implementiert wird, registriert wurde, werden die Methoden mouseDragged und mouseMoved jedesmal dann aufgerufen, wenn die Maus über dem Hauptfenster der Anwendung bewegt wird.

#### e) Ansatzpunkte für die Aufzeichnung der Benutzungsschnittstelle

Für die Aufzeichnung der Benutzungsschnittstelle können die oben geschilderten Mechanismen zur Nachrichtenverarbeitung direkt genutzt werden. Ein Aufzeichnungssystem kann nach dem folgenden Schema realisiert werden:

- Die aufzuzeichnenden Komponenten werden von der Anwendung an ein Aufzeichnungsmodul übergeben, das Informationen über den Anfangszustand der Komponenten ermittelt und aufzeichnet.
- Das Aufzeichnungssystem stellt Listener-Klassen zur Verfügung, die aus den jeweils übergebenen Ereignisobjekten die Daten extrahieren können, die für die Aufzeichnung der Anwendung notwendig sind. Für alle aufzuzeichnenden Komponenten werden durch das Aufzeichnungsmodul automatisch entsprechende Listener-Objekte registriert.
- Daten oder Ereignisse, die nicht durch Listener überwacht werden können, müssen von der Anwendung explizit an das Aufzeichnungssystem zur Sicherung übergeben werden.

Ein auf diesem Schema basierendes Aufzeichnungs- und Wiedergabesystem für Java-Anwendungen wird im zweiten Teil dieser Arbeit im Detail entworfen und implementiert.

---

<sup>27</sup> Anmerkung: Für bestimmte Listener stehen sog. Adapterklassen zur Verfügung. Diese enthalten leere Funktionsrümpfe für alle Methoden der Listener-Schnittstelle und können verwendet werden, um Listener-Objekte in Form von abgeleiteten Klassen zu erzeugen, bei denen nicht alle Methoden des Listeners implementiert werden müssen, sondern nur die notwendigen Methoden überschrieben werden.

## 2.4 Standardelemente grafischer Benutzungsschnittstellen

Betriebssysteme mit integrierter grafischer Benutzungsschnittstelle sowie Toolkits für die Implementierung von grafischen Benutzungsschnittstellen bieten in der Regel eine Reihe von Standardelementen für die Entwicklung von Anwendungsprogrammen an. Im Folgenden werden solche gebräuchlichen Bedienelemente im Überblick dargestellt und ihre Realisierung im Rahmen der in Abschnitt 2.3 genannten Systeme beschrieben.<sup>28</sup>

Da solche Standardelemente letztendlich ein zentraler Gegenstand bei der Aufzeichnung einer grafischen Benutzungsschnittstelle sind, werden anschließend die einzelnen Schritte bei der Interaktion mit einem Standardelement exemplarisch für das Element „Kontrollkästchen“ im Detail betrachtet.

### 2.4.1 Gebräuchliche Bedienelemente im Überblick



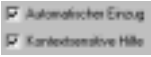




Die nachfolgende Tabelle bietet auf der Basis der in Abschnitt 2.3 beschriebenen grafischen Benutzungsschnittstellen einen Überblick über gebräuchliche Bedienelemente. Die Auswahl ist insofern willkürlich, als nur solche Elemente aufgenommen wurden, die bei wenigstens zwei der drei Systeme zur Verfügung stehen. Die Spalten der Tabelle sind wie folgt zu interpretieren:








- **Bedienelement...** enthält eine gebräuchliche deutsche und eine englische Bezeichnung für das Bedienelement sowie eine typische Ansicht basierend auf dem Look and Feel von Microsoft Windows NT 4.0.
- **Standardfunktion...** beschreibt den Einsatzbereich oder die Funktion, die das Element unabhängig von einer konkreten Anwendung übernimmt.
- **X mit OSF/Motif 1.2...** nennt die Widgetklasse oder die Funktion durch die das Element unter dem X Window-System mit dem OSF/Motif 1.2 Toolkit implementiert wird bzw. erzeugt werden kann und beschreibt Besonderheiten oder zusätzliche Funktionen zu der in Spalte 2 genannten Standardfunktion.
- **Microsoft Windows...** nennt die Windowklasse (und wenn nötig den Style), die für die Erzeugung des Bedienelements unter Microsoft Windows angegeben werden muß, und beschreibt Besonderheiten oder zusätzliche Funktionen zur der in Spalte 2 genannten Standardfunktion. Mit „(comctl32.dll)“ gekennzeichnete Elemente waren bei der ersten Version von Microsoft Windows 95 noch nicht verfügbar. Es handelt sich um sog. „Common Controls“, die durch eine zusätzliche Bibliothek zur Verfügung gestellt werden, die z.B. mit dem Microsoft Internet Explorer installiert wird und die bei aktuellen Versionen von Windows standardmäßig vorhanden ist.

---

<sup>28</sup> Anmerkung: Unter Standardelementen sollen in diesem Zusammenhang nur solche Elemente verstanden werden, die von Anwendungsentwicklern individuell für anwendungsspezifische Aufgaben erzeugt werden können, und nicht Elemente die aus der Sicht des Benutzers eine Standardfunktion, unabhängig von der Anwendung, übernehmen. Bspw. könnte die Systemschaltfläche zum Schließen eines Fensters vom Benutzer eher als ein Standardelement empfunden werden, als Schaltflächen, die unterschiedliche Aufgaben innerhalb einer Anwendung erfüllen. Aus der Sicht des Programmierers handelt es sich bei Elementen vom Typ „Schaltfläche“ jedoch um einen Standardmechanismus für die Interaktion mit dem Benutzer.

- **Java-AWT/JFC:** Nennt die JFC/Swing-Klasse (und soweit vorhanden die AWT-Klasse) durch die das Bedienelement implementiert wird und beschreibt Besonderheiten oder zusätzliche Funktionen zu der in Spalte 2 genannten Standardfunktion.

Bedienelement	Standardfunktion	X mit OSF/Motif 1.2	Microsoft Windows	Java-AWT/JFC
Schaltfläche (Button) 	Löst Aktionen aus. Entspricht einem Taster, der 2 Zustände annehmen kann: aktiviert und nicht aktiviert, wobei der aktivierte Zustand nicht permanent beibehalten wird.	PushButton Widget  Ist eine Unterklasse von Label Widget und besitzt dementsprechend dieselben Möglichkeiten, z.B. mehrzeiligen Text oder Symbol statt Text.	Class "BUTTON" Style BS_PUSHBUTTON  Kann auch mehrzeiligen Text oder ein Symbol darstellen.	Class Button (AWT) Class JButton (JFC)  JButton kann gleichzeitig einen Text und ein Symbol darstellen.
Statischer Text (Label) 	Dient zur Ausgabe von Texten, meist als Bezeichnung für andere Elemente. Bietet keine Interaktionsmöglichkeit für den Benutzer.	Label Widget  Kann auch zur Darstellung mehrzeiliger Texte oder eines Symbols verwendet werden.	Class "STATIC"  Kann auch zur Darstellung mehrzeiliger Texte oder eines Symbols verwendet werden.	Class Label (AWT) Class JLabel (JFC)  JLabel kann gleichzeitig einen Text und ein Symbol darstellen.
Kontrollkästchen (Checkbox) 	Repräsentiert mit den Zuständen „markiert“ und „nicht markiert“ eine boolesche Variable.	ToggleButton Widget  Ist eine Unterklasse von Label Widget und besitzt dementsprechend dieselben Möglichkeiten, z.B. mehrzeiligen Text oder Symbol statt Text.	Class "BUTTON" Style BS_CHECKBOX  Kann auch mehrzeiligen Text oder ein Symbol (zusätzlich zu dem Symbol, das den Zustand anzeigt) darstellen.	Class Checkbox (AWT) Class JCheckBox (JFC)  Bei JCheckBox können die Symbole zur Anzeige der Zustände frei gewählt werden.
Optionsfeld (Radiobutton) 	Repräsentiert mit den Zuständen „markiert“ und „nicht markiert“ eine boolesche Variable, wobei innerhalb einer Gruppe von Optionsfeldern nur eines markiert sein kann.	Keine separate Klasse. Wird durch ToggleButtons in Verbindung mit einem RowColumn Widget realisiert, das die Aktivierung mehrerer ToggleButtons in einer Gruppe verhindert.	Class "BUTTON" Style BS_RADIOBUTTON  Kann auch mehrzeiligen Text oder ein Symbol (zusätzlich zu dem Symbol, das den Zustand anzeigt) darstellen. Keine automatische Verwaltung von Gruppen von Optionsfeldern.	Class Checkbox (AWT) Class JRadioButton (JFC)  Bei JRadioButton können die Symbole zur Anzeige der Zustände frei gewählt werden. Die Verwaltung einer Gruppe von Optionsfeldern erfolgt durch die Klasse ButtonGroup.
Textfeld (Edit) 	Dient zum Editieren eines Textes.	TextField Widget Text Widget  Text Widgets erlauben mehrzeilige Texte. In Verbindung mit einem Widget vom Typ ScrolledWindow kann automatischer Bildlauf für mehrzeiligen Text realisiert werden.	Class "EDIT"  Erlaubt sowohl einzeligen als auch mehrzeiligen Text. Kann mit und ohne Bildlaufleisten erzeugt werden.	Class TextField (AWT) Class TextArea (AWT) Class JTextField (JFC) Class JTextArea (JFC)  TextArea und JTextArea erlauben auch mehrzeilige Texte. TextArea kann wahlweise mit Bildlaufleisten erzeugt werden. Für JTextArea werden diese durch die Klasse JScrollPane realisiert.
Listenfeld (Listbox) 	Zeigt eine Liste von Einträgen an und erlaubt die Auswahl eines Eintrags oder die Markierung mehrerer Einträge aus der Liste.	List Widget  In Verbindung mit einem ScrolledWindow Widget kann automatischer Bildlauf für die Liste realisiert werden.	Class "LISTBOX"  Stellt automatisch Bildlaufleisten zur Verfügung.	Class List (AWT) Class JList (JFC)  List stellt automatisch Bildlaufleisten zur Verfügung. Für JList werden diese durch die Klasse JScrollPane realisiert.
Auswahlfeld (Combobox) 	Besteht aus einer Liste und einem Textfeld, das einen ausgewählten Eintrag aus der aufklappbaren Liste zeigt.	OptionMenu  Keine Möglichkeit, einen Eintrag zu editieren. Die Liste ist grundsätzlich aufklappbar. Wird wie ein Pulldown Menu (s.u.) realisiert, das an ein CascadedButton Widget angehängt ist. Für die Erzeugung sind einfache Funktionen vorhanden	Class "COMBOBOX"  Die Liste kann sowohl aufklappbar als auch permanent sichtbar sein. Der Inhalt des Textfelds kann bei Bedarf editiert werden.	Class Choice (AWT) Class JComboBox (JFC)  Choice bietet im Gegensatz zu JComboBox keine Möglichkeit, den Eintrag des Textfeldes zu editieren. Die Liste ist sowohl bei Choice als auch bei JComboBox grundsätzlich aufklappbar.

Bedienelement	Standardfunktion	X mit OSF/Motif 1.2	Microsoft Windows	Java-AWT/JFC
Bildlaufleiste (Scrollbar) 	Ermöglicht die Auswahl eines Unterbereichs aus einem Wertebereich. Die Bildlaufleiste repräsentiert den Wertebereich. Ein Schieber innerhalb der Leiste repräsentiert den Unterbereich und seine Position innerhalb des Wertebereichs.	Scrollbar Widget  Soll laut [HF94] S.307 nur in Verbindung mit ScrolledWindows verwendet werden. Separate Verwendung widerspricht dem Motif Style Guide.	Class "SCROLLBAR"  Kann für bestimmte vordefinierte Fensterklassen (z.B. Textfelder, Listenfelder) und anwendungsspezifische Fensterklassen automatisch erzeugt werden. Ist jedoch auch als separates Element einsetzbar.	Class Scrollbar (AWT) Class JScrollbar (JFC)  JScrollbar wird standardmäßig automatisch mit JScrollPane erzeugt. Ist jedoch auch als separates Element einsetzbar.
Menüleiste (Menu) 	Eine horizontale Reihe von Schaltflächen. Jede Schaltfläche repräsentiert ein Menü, das geöffnet wird, wenn die Schaltfläche aktiviert wird.	MenuBar  Wird durch die Kombination eines RowColumn Widgets mit mehreren CascadedButton Widgets realisiert. Für die Erzeugung sind einfache Funktionen vorhanden	CreateMenu() LoadMenu()  Wird inkl. der Menüs durch die Funktion LoadMenu als Ressource geladen oder mit Hilfe von CreateMenu als leerer Container für PopupMenus erstellt, die mit InsertMenuItem hinzugefügt werden..	Class MenuBar (AWT) Class JMenuBar (JFC)  Sowohl MenuBar als auch JMenuBar sind zunächst leere Container für Menüs.
Menü (Pull-down Menu) 	Eine vertikale Reihe von Schaltflächen, die an eine Schaltfläche in einer Menüleiste angehängt ist. Die Einträge des Menüs repräsentieren Aktionen, boolesche Variablen oder untergeordnete Menüs.	Pull-down Menu  Wird durch ein RowColumn Widget realisiert, das Pushbuttons, Labels und Togglebuttons enthalten kann. Für die Erzeugung sind einfache Funktionen vorhanden.	CreatePopupMenu() LoadMenu()  Wird inkl. der Einträge und Untermenüs als Ressource geladen oder mit CreatePopupMenu als leerer Container für MenuItem erstellt, die mit InsertMenuItem eingefügt werden.	Class Menu (AWT) Class JMenu (JFC)  Ein JMenu kann außer JMenuItem auch beliebige andere Bedienelemente (z.B. Optionsfelder und Kontrollkästchen) enthalten.
Kontextmenü (Popup Menu) 	Eine vertikale Reihe von Schaltflächen, die Aktionen, boolesche Variablen oder untergeordnete Kontextmenüs repräsentieren.	Popup Menu  Wird durch ein RowColumn Widget realisiert, das Pushbuttons, Labels und Togglebuttons enthält. Für die Erzeugung sind einfache Funktionen vorhanden.	CreatePopupMenu() LoadMenu()  Erstellung s.o. bei Menü. (Microsoft Windows unterscheidet nicht zwischen Pull-down und Popup Menüs.) Anzeige erfolgt mit TrackPopupMenu.	Class PopupMenu (AWT) Class JPopupMenu (JFC)  Ein JPopupMenu kann außer JMenuItem auch beliebige andere Bedienelemente (z.B. Optionsfelder und Kontrollkästchen) enthalten.
Schieberegler (Slider) 	Realisiert einen Schieberegler zur Einstellung eines Wertes innerhalb eines Wertebereichs.	Scale Widget	Class TRACKBAR_CLASS (comctl32.dll)  Es existieren verschiedene Typen, die z.B. mit unterschiedlichen Versionen des Microsoft Internet Explorers ausgeliefert wurden.	Class JSlider (JFC)
Listenansicht (Listview) 	Zeigt eine Tabelle an und erlaubt die Auswahl einer oder mehrerer Zeilen der Tabelle.	Nicht vorhanden	Class WC_LISTVIEW (comctl32.dll)  Erlaubt neben tabellarischer Ansicht auch Symbole und mehrspaltige Listen. Es existieren verschiedene Typen, die z.B. mit unterschiedlichen Versionen des Microsoft Internet Explorers ausgeliefert wurden.	Class JTable (JFC)  Erlaubt auch die Auswahl von Spalten und Zellen und das Editieren von Zellen. Bildlaufleisten werden durch die Klasse JScrollPane realisiert.
Baumansicht (Treeview) 	Dient zur Visualisierung hierarchischer, baumartiger Datenstrukturen und ermöglicht die Auswahl eines Eintrags im Baum.	Nicht vorhanden	Class WC_TREEVIEW (comctl32.dll)  Erlaubt das Editieren von Einträgen. Es existieren verschiedene Typen, die z.B. mit unterschiedlichen Versionen des Microsoft Internet Explorers ausgeliefert wurden.	Class JTree (JFC)  Erlaubt das Editieren von Einträgen. Bildlaufleisten werden durch die Klasse JScrollPane realisiert.

## 2.4.2 Detaillierte Betrachtung der Interaktion mit einem Bedienelement

In diesem Kapitel wird anhand eines konkreten Beispiels dargestellt, daß bei der Aufzeichnung der Interaktion des Benutzers mit einem Bedienelement auch oder gerade Ereignisse berücksichtigt werden müssen, die in Anwendungsprogrammen in der Regel nicht bearbeitet werden, wenn ein Bedienelement nur entsprechend der „normalen“ anwendungsspezifischen Funktion verwendet wird, bspw. um dem Benutzer die Möglichkeit zu geben, ein Kommando auszulösen oder eine Auswahl vorzunehmen. Wenn eine Interaktion mit einem Bedienelement im Detail betrachtet wird, können dabei zwei Ebenen unterschieden werden:

1. Eine anwendungsnahe Ebene, auf welcher der Benutzer mit Hilfe des Bedienelements eine Aktion auslöst, die ggf. eine Verarbeitung oder Zustandsänderung im Sinne der Anwendungslogik nach sich zieht.
2. Die systemnahe Ebene, auf der das Bedienelement zunächst nur eine Datenstruktur des Betriebssystems, ein Window unter Microsoft Windows oder ein Component unter Java darstellt. Erst eine Folge von Ereignissen auf dieser Ebene entspricht einer Aktion auf der anwendungsnahen Ebene.

Die Vorgänge auf beiden Ebenen werden nachfolgend am Beispiel eines Kontrollkästchens betrachtet, wobei die folgende Anwendungssituation durchgespielt wird: Ein Benutzer klickt ein Kontrollkästchen mit der linken Maustaste an, um dessen Zustand von „markiert“ auf „nicht markiert“ zu ändern.

	Kontrollkästchenszustand	Interaktionsschritt	Ereignis (System)	Ereignis (Anwendung)
1	markiert (nicht aktiviert) <input checked="" type="checkbox"/> markiert	Der Mauszeiger betritt den Bereich des Kontrollkästchens.	MouseEnter	
2	markiert (nicht aktiviert) <input checked="" type="checkbox"/> markiert	Der Mauszeiger wird über dem Kontrollkästchen bewegt.	MouseMove	
3	markiert (nicht aktiviert) <input checked="" type="checkbox"/> markiert	Die linke Maustaste wird über dem Kontrollkästchen gedrückt.	ButtonDown	
4	markiert (aktiviert) <input checked="" type="checkbox"/> markiert	Der Mauszeiger wird über dem Kontrollkästchen bewegt.	MouseMove	
5	markiert (aktiviert) <input checked="" type="checkbox"/> markiert	Die linke Maustaste wird über dem Kontrollkästchen losgelassen.	ButtonUp	
6	nicht markiert (nicht aktiviert) <input type="checkbox"/> nicht markiert			ClickAction
7	nicht markiert (nicht aktiviert) <input type="checkbox"/> nicht markiert	Der Mauszeiger wird über dem Kontrollkästchen bewegt.	MouseMove	
8	nicht markiert (nicht aktiviert) <input type="checkbox"/> nicht markiert	Der Mauszeiger verläßt den Bereich des Kontrollkästchens.	MouseLeave	

Abbildung 22: Interaktion mit einem Kontrollkästchen.

Abbildung 22 zeigt 8 Interaktionsschritte im Detail. In der ersten Spalte wird der Zustand des Kontrollkästchens vor einem Interaktionsschritt beschrieben und durch eine Abbildung, die dem Look and Feel von Microsoft Windows NT 4.0 entspricht, darstellt. Die zweite Spalte beschreibt den Interaktionsschritt. In der dritten Spalte werden Ereignisse, die sich aus dem Interaktionsschritt auf der systemnahen Ebene ergeben, beschrieben. In der vierten Spalte Ereignisse auf der Anwendungsebene. (Die Bezeichnung der Ereignisse ist hier symbolisch und entspricht keinem konkreten Betriebssystem.) Aus der Betrachtung des Vorgangs können folgende Schlußfolgerungen gezogen werden:

- Auf der anwendungsnahen Ebene wird nur ein einziges Ereignis ausgelöst, das die Aktion des Benutzers beschreibt. Wenn das Anwendungsprogramm auf diese Aktion unmittelbar reagieren möchte (und nicht nur den Zustand des Kontrollkästchens zu einem späteren Zeitpunkt abfragt), so erfolgt diese Reaktion auf die entsprechende Nachricht, die bspw. unter Java durch einen ActionListener, der für das Kontrollkästchen registriert wurde, abgefangen werden kann.
- Auf der systemnahen Ebene des Bedienelements treten zahlreiche Ereignisse auf,<sup>29</sup> die zu einer Reihe von Nachrichten führen. Die Reaktion des Kontrollkästchens auf diese Nachrichten ist von der konkreten Anwendung unabhängig und solche Nachrichten müssen von Anwendungsprogrammen nicht explizit verarbeitet werden. Anschaulich gesprochen: Ein Java Programm wird für ein Kontrollkästchen keinen MouseMotionListener registrieren, der Mausebewegungen mitteilt, wenn es mit dem Kontrollkästchen „nichts besonderes“ vorhat, außer dem Benutzer die Auswahl einer Option zu ermöglichen.
- Das Kontrollkästchen, das aufgrund seiner Funktion im Sinne der Anwendungslogik nur zwei Zustände kennt: „markiert“ und „nicht markiert“, besitzt ein internes Attribut, das durch seinen Wert, „aktiviert“ oder „nicht aktiviert“, anzeigt, ob gerade eine Interaktion stattfindet. Der Wert dieses Attributs bestimmt die Darstellung des Kontrollkästchens. Der Innenbereich des Symbols, das den Zustand „markiert“ oder „nicht markiert“ anzeigt, wird grau unterlegt, wenn das Kontrollkästchen aktiviert ist. Darüber hinaus zeigt ein gepunkteter Rahmen um den Text des Kontrollkästchens an, daß das Bedienelement zur Zeit den Tastaturfokus besitzt. Das Kontrollkästchen wird damit im Verlauf des Interaktionsvorgangs selbst auch zum aktuellen Wert eines globalen Attributs des Systems.

Verallgemeinert man diese Schlußfolgerungen, so ergeben sich folgende Konsequenzen für die Aufzeichnung und die Wiedergabe von Bedienelementen:

- Bedienelemente können interne Attribute besitzen, die sich auf das Erscheinungsbild des Elements auswirken. Um eine wirklichkeitsnahe Wiedergabe einer Benutzerinteraktion zu erreichen, müssen systemnahe Ereignisse erfaßt werden, die Werte dieser Attribute verändern und die von der Anwendung sonst, d.h. ohne Aufzeichnung, evtl. nicht erfaßt werden würden. Bei der Wiedergabe muß der interne Zustand eines Bedienelements so wiederhergestellt werden, wie dies bei der Aufzeichnung der Fall war. Wenn Attribute von außen nicht zugänglich sind, müssen Nachrichten, die zu der gewünschten Zustandsänderung führen, (wieder) an das Bedienelement geschickt werden.<sup>30</sup>
- Wenn Ereignisse sowohl auf der systemnahen Ebene als auch auf der Anwendungsebene aufgezeichnet werden, muß bei der Wiedergabe gewährleistet sein, daß bestimmte Folgen von systemnahen Ereignissen nicht zu doppelten anwendungsnahen Aktionen führen, falls die Wiedergabe einer nochmaligen Ausführung gleichkommen soll. Alternativ kann auf die Aufzeichnung von anwendungsnahen Ereignissen verzichtet werden, sofern gewährleistet ist, daß systemnahe Ereignisse so wiedergegeben werden können, daß sie wie „echte“ Ereignisse, die durch den Benutzer ausgelöst wurden, verarbeitet werden.

Mit Bezug auf die Implementierung des im Rahmen dieser Arbeit entworfenen Systems wird in Kapitel 4.2.5 noch detailliert geschildert werden, daß eine wirklichkeitsgetreue Wiedergabe in Folge der hier genannten Anforderungen in der Realität nicht in jeder Situation möglich ist.

---

<sup>29</sup> Anmerkung: Tatsächlich werden zwischen den in Abbildung 22 gezeigten Ereignissen meist noch weitere Ereignisse vom Typ MouseMove auftreten, die hier der Kürze wegen nicht gezeigt werden.

<sup>30</sup> Anmerkung: Von der Möglichkeit, das Erscheinungsbild eines Bedienelements bei jeder Zustandsänderung als Bitmap zu sichern und diese Bitmaps später wiederzugeben, soll hier abgesehen werden.

### 3. Design eines Recorder/Viewers für das Java-AWT

Nachdem in den vorangegangenen Kapiteln Einsatzbereiche für die Aufzeichnung von Benutzungsschnittstellen gezeigt, mit der Aufgabenstellung zusammenhängende Fragestellungen betrachtet sowie drei Benutzungsschnittstellen vorgestellt und Ansatzpunkte für deren Aufzeichnung dargestellt wurden, wird nachfolgend auf dieser Grundlage das JRV-System, ein Recorder/Viewer für Java-AWT- und JFC/Swing-Komponenten entworfen.

In diesem Kapitel wird zunächst die Funktion des JRV-Systems in einer Übersicht vorgestellt sowie das Design des Systems von der Modul- bzw. Paketebene bis zur Spezifikation der Abläufe in typischen Anwendungssituationen dokumentiert.

Kapitel 4 enthält einige Anmerkungen zur Implementierung. Kapitel 5 zeigt das prototypisch implementierte System im Einsatz und bietet anhand verschiedener Anwendungsbeispiele eine Anleitung für die Benutzung des Systems. Eine kritische Analyse sowie Ansatzpunkte für Verbesserungen und Erweiterungen werden schließlich in Kapitel 6 vorgestellt.

#### 3.1 Begriffe und Schreibweisen

Zur Beschreibung des Systemdesigns werden Begriffe und Konzepte der objektorientierten Softwareentwicklung wie Klasse, Objekt, Instanz, Methode, Polymorphismus, Vererbung usw. verwendet, die an dieser Stelle nicht explizit definiert werden sollen. Eine Übersicht bieten bspw. [JCJO92] S.42ff. und [RSC97].

Die Abbildungen zur Dokumentation des Designs orientieren sich an der Notation der UML (Unified Modeling Language), wie sie in [RSC97] beschrieben ist. Die Benennung von Datentypen, Klassen, Objekten, Variablen usw. erfolgt nach dem folgenden Schema:

- **Allgemeines...** Die in dieser Aufzählung beschriebenen Objekte sowie alle Bezeichner, die auch im Programmquellcode vorkommen können, werden mit englischen Begriffen bezeichnet.
- **Pakete, Schnittstellen und Klassen...** beginnen mit einem Großbuchstaben. Wenn der Bezeichner aus mehreren Worten besteht, beginnt jedes Wort mit einem Großbuchstaben. Zwischen den Worten steht kein Trennzeichen. Beispiel: **RecordController**.
- **Methoden und Operationen...** beginnen mit einem Kleinbuchstaben, ausgenommen sind Konstruktormethoden, die wie die entsprechende Klasse benannt werden. Wenn der Bezeichner aus mehreren Worten besteht, beginnt jedes Wort außer dem ersten mit einem Großbuchstaben. Zwischen den Worten steht kein Trennzeichen. Beispiel: **openTrack**.
- **Attribute, Parameter und Variablen...** werden durchgehend klein geschrieben. Beispiel: **objectcount**.
- **Statische konstante Werte...** werden durchgehend groß geschrieben. Wenn der Bezeichner aus mehreren Worten besteht, werden die Worte durch einen Unterstrich getrennt. Beispiel: **STATE\_PLAYING**.
- **Einfache Datentypen...** werden nach dem entsprechenden Datentyp unter Java benannt. Beispiel: **int**.

Zur Beschreibung der Funktionalität des JRV-Systems werden darüber hinaus die nachfolgend aufgezählten Begriffe mit den folgenden Bedeutungen verwendet:

- **Vorgangstyp...** Ein Vorgangstyp ist eine Folge von Arbeitsschritten, die mit einem konkreten Anwendungsprogramm durchgeführt werden, um eine bestimmte Aufgabe aus der Sicht des Benutzers zu bewältigen. Beispiele für Vorgangstypen sind das Kopieren einer Datei mit Hilfe eines Dateimanagers, das Ausfüllen eines Formulars auf einer Seite im Internet oder das Schreiben eines Briefes mit Hilfe eines Textprogramms.
- **Vorgang...** Ein Vorgang ist eine Instanz eines Vorgangstyps. Bspw. das Kopieren einer Datei A aus dem Verzeichnis X in das Verzeichnis Y oder das Ausfüllen eines Formulars im Internet mit den Daten des Autors dieser Arbeit, um bei der Universitätsbibliothek ein Buch zu reservieren.
- **Aufzeichnung...** Eine Aufzeichnung beschreibt einen Vorgang.
- **Wiedergabe...** Eine Wiedergabe ist ein Vorgang, bei dem eine Aufzeichnung verwendet wird, um das optische Erscheinungsbild eines Vorgangs oder den Vorgang selbst zu wiederholen.
- **Aufzeichnungsklassen...** Aufzeichnungsklassen sind Datentypen, die in Anwendungsprogrammen verwendet werden und deren Funktion und Arbeitsweise für die Erstellung von Aufzeichnungen in Betracht gezogen werden muß. In der Regel handelt es sich bei den hier betrachteten Klassen um Elemente der Benutzungsschnittstelle, z.B. um Fenster, Schaltflächen oder Dialogfelder.
- **Aufzeichnungsobjekte...** Aufzeichnungsobjekte sind Instanzen von Aufzeichnungsklassen. Für die Erzeugung von Aufzeichnungen werden die Zustände und die Änderungen der Zustände von Aufzeichnungsobjekten ermittelt.
- **Wiedergabeklassen...** Wiedergabeklassen sind Datentypen, die von einem Programm verwendet werden, um die Wiedergabe von Aufzeichnungsklassen zu ermöglichen.
- **Wiedergabeobjekte...** Wiedergabeobjekte sind Instanzen von Wiedergabeklassen.
- **Standardklassen...** Standardklassen sind Aufzeichnungsklassen die im Java-AWT oder in der JFC/Swing-Bibliothek enthalten sind und die durch das JRV-System standardmäßig bei der Erzeugung von Aufzeichnungen und deren Wiedergabe unterstützt werden.
- **Standardobjekte...** Standardobjekte sind Instanzen von Standardklassen.
- **Ereignisse oder Events...** Ereignisse oder Events sind Informationseinheiten, aus denen sich eine Aufzeichnung zusammensetzt. In der Regel beschreiben Ereignisse den Zustand oder eine Zustandsänderung eines Aufzeichnungsobjektes.
- **Spuren oder Tracks...** Eine Spur oder ein Track ist ein Datentyp durch den eine Aufzeichnung innerhalb des JRV-Systems repräsentiert wird.
- **Einträge oder Entries...** Einträge oder Entries sind Datenpakete, aus denen sich eine Spur zusammensetzt. Ein Eintrag repräsentiert ein Ereignis innerhalb der Aufzeichnung, die durch die Spur repräsentiert wird, in welcher der Eintrag enthalten ist.
- **Kopf oder Header...** Ein Kopf oder Header ist derjenige Teil eines Eintrags, der strukturierte, beschreibende Informationen über den weiteren Inhalt des Eintrags, enthält.
- **Player...** Ein Player ist ein Programm zur Wiedergabe einer Aufzeichnung.

## **3.2 Die Funktionalität des Java Recorder/Viewer-Systems**

### **3.2.1 Anwendungsbereiche**

Das Java Recorder/Viewer-System, JRV, stellt eine Klassenbibliothek zur Verfügung, die in aufzuzeichnende Anwendungsprogramme integriert werden kann und für eine große Bandbreite von Anwendungen einsetzbar ist. Dabei können die beiden folgenden grundlegenden Anwendungsbereiche unterschieden werden:

#### **Aufzeichnungen zur optischen Wiedergabe**

Das Ergebnis der Aufzeichnung eines Anwendungsprogramms zur optischen Wiedergabe ist eine Art Video oder Diashow, die bspw. zur Erstellung von Präsentationen, von Unterrichts- oder Anleitungsmaterial oder zur Gewinnung von Daten zur Analyse des Benutzerverhaltens genutzt werden kann.

#### **Aufzeichnungen zur wiederholten Ausführung**

Das Ergebnis der Aufzeichnung zur wiederholten Ausführung eines Anwendungsprogramms ist ein Art Ausführungsskript. Einsatzmöglichkeiten sind bspw. die Unterstützung von Makro-funktionalität oder die zeitgesteuerte oder wiederholte Ausführung von Vorgängen.

Diese Anwendungsbereiche weisen die nachfolgend beschriebenen Merkmale und Anforderungen auf, die durch das JRV-System im Besonderen unterstützt werden sollen.

### **3.2.2 Aufzeichnung zur optischen Wiedergabe**

Um das optische Erscheinungsbild eines Vorgangs aufzeichnen und wiedergeben zu können, stellt das JRV-System die folgenden Mechanismen zur Verfügung, die unabhängig von der Semantik des Vorgangs verwendet werden können und nur wenige Eingriffe in das aufgezeichnete Anwendungsprogramm erfordern, wenn die Benutzungsschnittstelle des Programms aus Standardobjekten besteht:

- Automatische Aufzeichnung und Wiedergabe des Erscheinungsbildes der Standardklassen des Java-AWT und der JFC/Swing-Bibliothek, die durch das JRV-System direkt unterstützt werden.
- Automatische Aufzeichnung von Fenstern, die Komponenten von Standardklassen enthalten. So bewirkt bspw. die Übergabe eines Rahmenfensters (Frames) die automatische Aufzeichnung der enthaltenen Schaltflächen.
- Die Steuerung des Aufzeichnungs- und Wiedergabeprozesses durch den Endbenutzer (Start, Stop, Pause) kann über eine in das JRV-System integrierte Standardoberfläche erfolgen.
- Ein in das JRV-System integrierter Runtime-Player ermöglicht die Wiedergabe ohne das aufgezeichnete Programm.

### 3.2.3 Aufzeichnung zur wiederholten Ausführung

Die Aufzeichnung zur wiederholten Ausführung erfordert in der Regel eine Anpassung des Aufzeichnungs- und Wiedergabeprozesses an die Semantik des aufzuzeichnenden Vorgangs. Das JRV-System stellt zu diesem Zweck einen Rahmen zur Verfügung, der es Anwendungsentwicklern ermöglicht, vorhandene Ressourcen zu nutzen und auf eine allgemeine Struktur zurückgreifen zu können, der jedoch flexibel an die Erfordernisse verschiedener Einsatzbereiche angepaßt und erweitert werden kann.

- Einfache Erweiterbarkeit um zusätzliche Aufzeichnungsklassen, bspw. AWT- oder JFC/Swing-Komponenten, die in der prototypischen Implementierung des JRV-Systems nicht realisiert wurden. Zusätzliche Aufzeichnungsklassen können zur Laufzeit hinzugefügt werden. Vorhandene Standardkomponenten können zur Laufzeit ersetzt werden.
- Die Steuerung des Aufzeichnungs- und Wiedergabeprozesses kann vollständig durch die Anwendung erfolgen. Die konsequente Trennung der Kontrollmechanismen von der Benutzungsschnittstelle ermöglicht sowohl anwendungsspezifische Benutzungsschnittstellen als auch eine „unsichtbare“ Integration in Anwendungsprogramme.
- Sowohl Aufzeichnung als auch Wiedergabe können vollständig auf der Ereignisebene kontrolliert werden. Anwendungsspezifische Ereignisse und Daten können durch das generische Aufzeichnungssystem verarbeitet werden.
- Ein konsequenter objektorientierter Ansatz ermöglicht sowohl die Erweiterung als auch den skalierten Einsatz vorhandener Komponenten, von der kompletten Nutzung bis zum Einsatz als unterstützendes Speichermanagement- oder Timingsystem.

### 3.3 Die Paketarchitektur des JRV-Systems

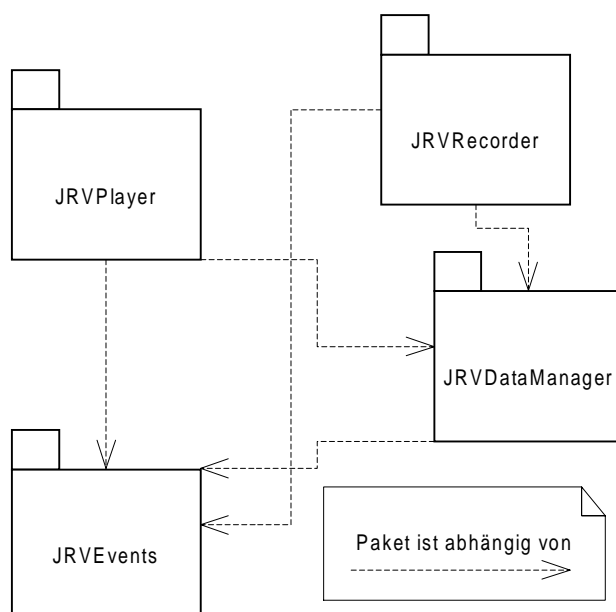


Abbildung 23: Unterteilung des JRV-Systems in Pakete

Abbildung 23 zeigt die Unterteilung des JRV-Systems in folgende vier Pakete:

#### **a) JRVRecorder**

Das JRVRecorder-Paket enthält Klassen für die Aufzeichnung von AWT- und JFC/Swing-Komponenten, zur Kontrolle des Aufzeichnungsprozesses und die Kommunikation mit dem aufgezeichneten Programm sowie eine Standardoberfläche zur Steuerung des Aufzeichnungsprozesses.

#### **b) JRVPlayer**

Das JRVPlayer-Paket enthält neben Klassen für die Wiedergabe von AWT- und JFC/Swing-Komponenten einen Runtime-Player zur Wiedergabe ohne die aufgezeichnete Anwendung, eine Standardoberfläche für die Steuerung der Wiedergabe sowie Klassen für die Kontrolle des Wiedergabeprozesses und die Kommunikation mit einem wiedergebenden Anwendungsprogramm.

#### **c) JRVDatamanager**

Das JRVDatamanager-Paket enthält Klassen für die Organisation von aufgezeichneten Daten im Hauptspeicher sowie für die permanente Sicherung (Serialisierung) auf Sekundär Speichermedien.

#### **d) JRVEvents**

Im JRVEvents-Paket werden Klassen und Konstanten definiert, die von den übrigen Paketen gleichermaßen benötigt werden, um Ereignisse darzustellen.

### ***3.4 Aufzeichnen mit dem JRVRecorder***

In diesem Kapitel werden die Mechanismen des JRV-Systems zur Aufzeichnung beschrieben und die grundlegenden Konzepte dargestellt. Eine detaillierte Beschreibung des Verhaltens der Klassen und ihrer Methoden liefert der dokumentierte Quellcode des Systems. Kapitel 5 bietet konkrete Anwendungsbeispiele.

#### **3.4.1 Die Klassen des JRVRecorder-Pakets**

Abbildung 24 zeigt eine Übersicht der Klassen des JRVRecorder-Pakets, die im Einzelnen folgende Funktionen übernehmen:

##### **a) RecordSourceManager**

RecordSourceManager sind Objekte, welche die Aufzeichnung einer Aufzeichnungsklasse, bspw. eines bestimmten Typs von Bedienelementen oder Fenstern, oder einer Menge von Aufzeichnungsklassen übernehmen.

RecordSourceManager werden zur Laufzeit bei einem RecordController (s.u.) registriert und zeichnen Daten, die für die Wiedergabe notwendig sind auf, so daß ein korrespondierendes PlayTargetManager-Objekt (vgl. Abschnitt 3.5.1) die Darstellung oder das Verhalten der Aufzeichnungsobjekte wiedergeben kann.

Durch das RecordSourceManager-Interface werden die Signaturen der Methoden definiert, über die ein RecordController mit RecordSourceManager-Objekten kommuniziert. Das JRVRRecorder-Paket enthält Implementierungen des RecordSourceManager-Interfaces für die Aufzeichnung ausgewählter Klassen der AWT- und der JFC/Swing-Bibliothek.

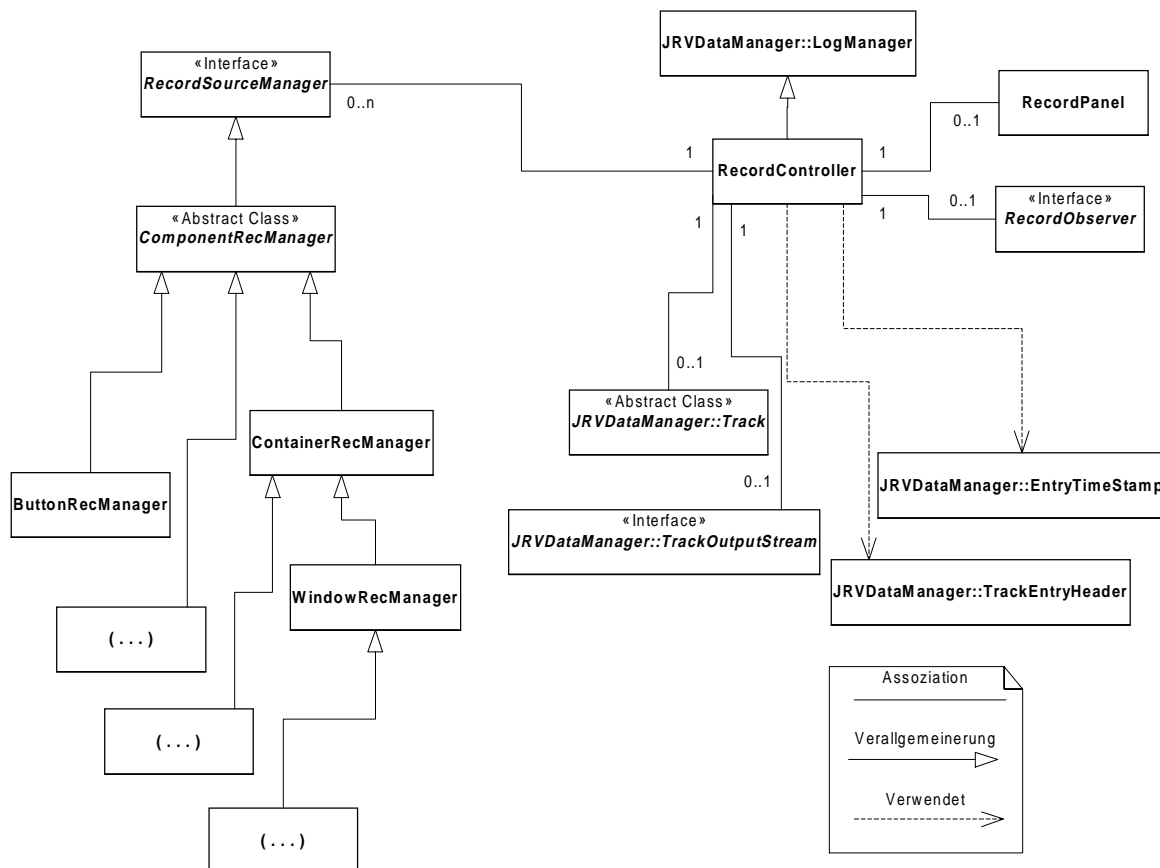


Abbildung 24: Die Klassen des JRVRRecorder-Pakets

## b) RecordController

Der RecordController ist die zentrale Komponente zur Durchführung von Aufzeichnungsprozessen. Er verwaltet RecordSourceManager und Aufzeichnungsobjekte und übernimmt die Kommunikation mit dem aufgezeichneten Anwendungsprogramm sowie mit den Klassen des JRVRDataManger-Pakets, das die Datentypen Track und TrackOutputStream zur Verwaltung und Sicherung von aufgezeichneten Daten zur Verfügung stellt.

Darüber hinaus bietet der RecordController Methoden, über welche wahlweise das aufgezeichnete Anwendungsprogramm oder der Endbenutzer mit Hilfe eines RecordPanels (s.u.) den Aufzeichnungsprozeß steuern, d.h. Aufzeichnungen starten, anhalten und beenden kann.

## c) RecordPanel

Das RecordPanel stellt ein Dialogfeld zur Verfügung, über das der Endbenutzer den Aufzeichnungsprozeß steuern, d.h. starten, anhalten oder beenden kann.

## d) RecordObserver

Das RecordObserver-Interface definiert eine Reihe von Callback-Funktionen, die durch den RecordController beim Eintreten bestimmter Ereignisse während des Aufzeichnungsprozesses aufgerufen werden, z.B. beim Hinzufügen oder Entfernen von Aufzeichnungsobjekten oder vor der Sicherung von Ereignissen. Durch ein RecordObserver-Objekt, das durch das aufgezeichnete Anwendungsprogramm zur Verfügung gestellt wird, kann die Aufzeichnung auf der Ereignisebene kontrolliert werden.

### 3.4.2 Das Zusammenspiel der Komponenten des JRVRecorder-Pakets

Abbildung 25 zeigt das Zusammenspiel der Komponenten des JRVRecorder-Pakets sowohl untereinander als auch in der Wechselwirkung mit einer aufgezeichneten Anwendung sowie mit dem JRVDataManager-Paket.

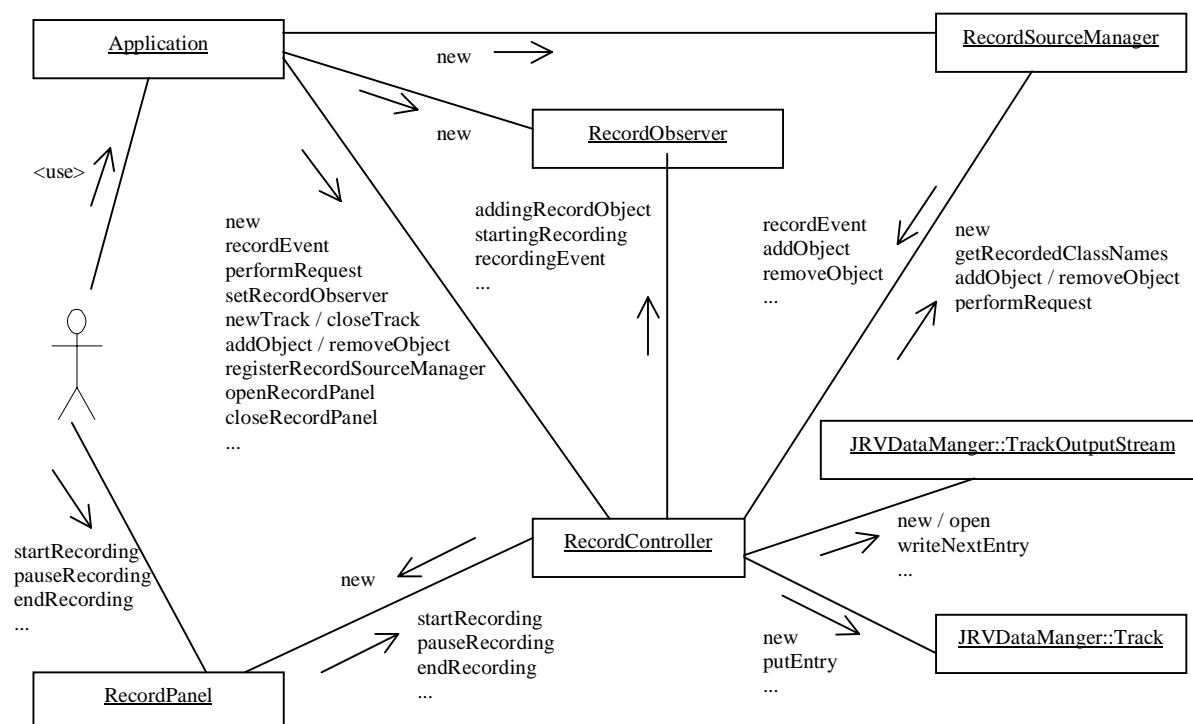


Abbildung 25: Zusammenarbeit der Komponenten des JRVRecorder-Pakets

Die folgenden Abschnitte und die Abbildungen 26 bis 28 demonstrieren den zeitlichen Ablauf beim Zusammenspiel der oben beschriebenen Klassen in den prototypischen Situationen:

- Initialisierung des Aufzeichnungssystems
- Initialisierung einer Aufzeichnung
- Aufzeichnung von Ereignissen

### 3.4.3 Die Initialisierung des Aufzeichnungssystems

Zu Beginn einer Aufzeichnungssitzung erfolgt, wie beispielhaft in Abbildung 26 gezeigt, die Initialisierung des JRVRecorder-Systems, mit den unten näher beschriebenen Schritten. Die im Text in Klammern, in der Form (X:) angegebenen Verweise entsprechen den Teilschritten des in der Abbildung dargestellten Sequenzdiagramms.

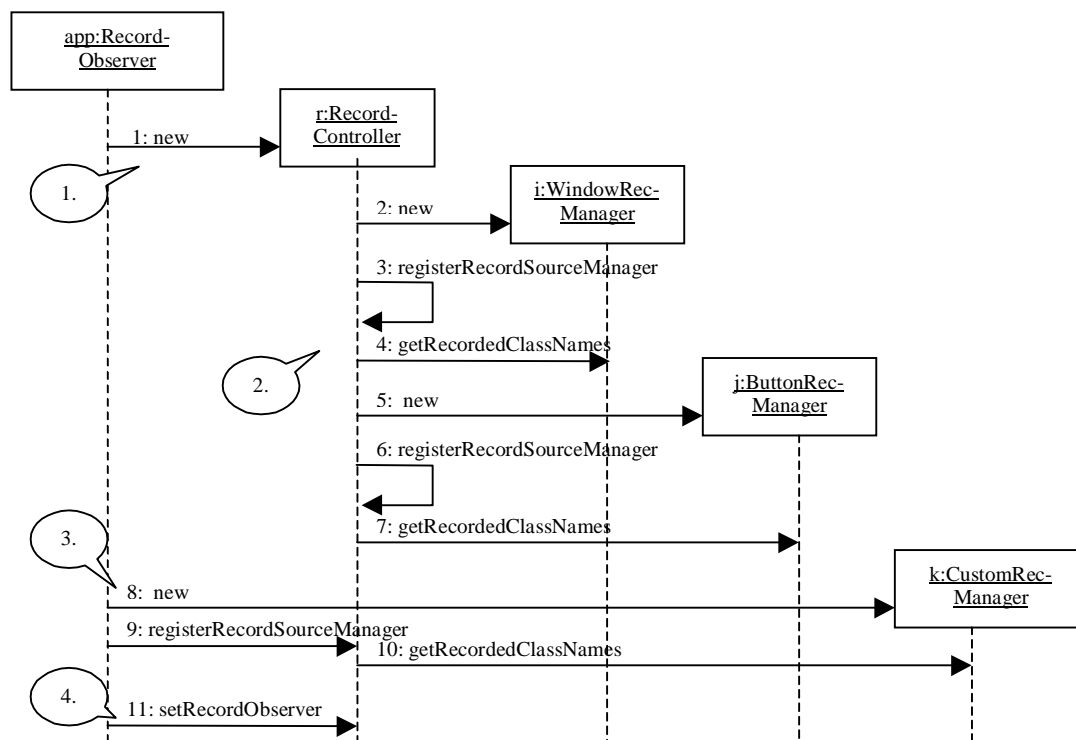


Abbildung 26: Initialisierung des Aufzeichnungssystems

#### 1. Erzeugung eines RecordControllers (1:)

#### 2. Automatische Registrierung standardmäßig vorhandener RecordSourceManager

Der RecordController erzeugt automatisch RecordSourceManager für Aufzeichnungsklassen, die standardmäßig durch das JRV-System unterstützt werden (2: und 5:), in dem in Abbildung 26 dargestellten Beispiel für Fenster (Windows) und Schaltflächen (Buttons). Er registriert diese bei sich selbst (3: und 6:) und fragt die Klassen der Aufzeichnungsobjekte ab, die durch den jeweiligen RecordSourceManager aufgezeichnet werden können. (4: und 7:)

#### 3. Registrierung anwendungsspezifischer RecordSourceManager

Wenn das Anwendungsprogramm zusätzliche RecordSourceManager zur Verfügung stellt, werden diese erzeugt und ebenfalls bei dem RecordController registriert. (8: bis 10:) Falls mehrere RecordSourceManager dieselbe Aufzeichnungsklasse aufzeichnen können, wird der jeweils zuletzt registrierte RecordSourceManager verwendet, so daß Anwendungsprogramme standardmäßige RecordSourceManager durch anwendungsspezifische ersetzen können.

## 4. Registrierung eines RecordObservers

Falls der Aufzeichnungsprozeß durch das Anwendungsprogramm auch auf Ereignisebene kontrolliert werden soll, registriert es einen RecordObserver beim RecordController. (11:)

### Anmerkungen

Das in Abbildung 26 gezeigte Beispiel beschreibt ein Szenario, in dem das aufzuzeichnende Programm zusätzliche RecordSourceManager zur Verfügung stellt und den Aufzeichnungsprozeß mit einem RecordObserver auf Ereignisebene kontrolliert. In Anwendungssituationen, in denen eine Benutzungsschnittstelle, die nur aus Standardklassen aufgebaut ist, aufgezeichnet werden soll, kann die Initialisierung auf die Schritte 1. und 2. reduziert werden. In dem hier gezeigten Beispiel, ebenso wie in den folgenden Abschnitten, wird vorausgesetzt, daß die Klasse, die den Kern des Anwendungsprogramms repräsentiert, auch das RecordObserver-Interface implementiert. In der Realität wird es sich dabei oft um unterschiedliche Klassen handeln. Die erste Spalte der hier gezeigten Sequenzdiagramme ist in diesem Sinne als Platzhalter für „sonstige Objekte“ der Anwendung zu verstehen, die nicht eigens benannt werden.

### 3.4.4 Die Initialisierung einer Aufzeichnung

Die Initialisierung einer Aufzeichnung erfolgt entsprechend dem in Abbildung 27 gezeigten Schema in den nachfolgend beschriebenen Schritten. Die im Text in Klammern, in der Form (X:) angegebenen Verweise entsprechen den Teilschritten des in der Abbildung dargestellten Sequenzdiagramms.

#### 1. Erzeugung der Aufzeichnungsobjekte

Zunächst werden Aufzeichnungsobjekte erzeugt. In dem in Abbildung 26 gezeigten Beispiel handelt es sich um eine Benutzungsschnittstelle, die nur aus einem Rahmenfenster (Frame) besteht, dem eine Schaltfläche (Button) hinzugefügt wird. (1: bis 3:)

#### 2. Erzeugung eines Tracks

Anschließend wird ein Track zur Sicherung der Aufzeichnung erzeugt. (4: und 5:) Die Track-Klasse und die sonstigen Funktionen des JRVDatamanager-Pakets werden in Abschnitt 3.5.1 näher beschrieben.

#### 3. Start des Aufzeichnungsprozesses

Der RecordController wird angewiesen, eintreffende Ereignisse aufzuzeichnen (6:). Der RecordObserver wird informiert, daß eine Aufzeichnung gestartet wurde, und kann an dieser Stelle gegen die Aufzeichnung votieren. (7:) Beim Start der Aufzeichnung stehen folgende Modi zur Verfügung:

- **Zeitlose Aufzeichnung...** Alle Einträge erhalten dieselbe Zeitmarke, nämlich die des jeweils vorhergehenden Eintrags, und werden bei der Wiedergabe so schnell als möglich hintereinander abgespielt.
- **Aufzeichnung in Echtzeit...** Die jeweils seit dem Start der Aufzeichnung vergangene Zeit bestimmt die Zeitmarke der aufgezeichneten Einträge.

Zeitlose Aufzeichnung kann verwendet werden, um bspw. einen Konfigurationsprozeß durch den Benutzer aufzuzeichnen, der längere Zeit in Anspruch nimmt, jedoch bei der Wiedergabe adhoc ablaufen soll, z.B. das Öffnen mehrerer Fenster oder die Initialisierung der Bedienelemente. Anschließend kann die Aufzeichnung angehalten und später im Echtzeit-Modus erneut gestartet werden, wenn der eigentliche Vorgang, der aufgezeichnet werden soll, beginnt.

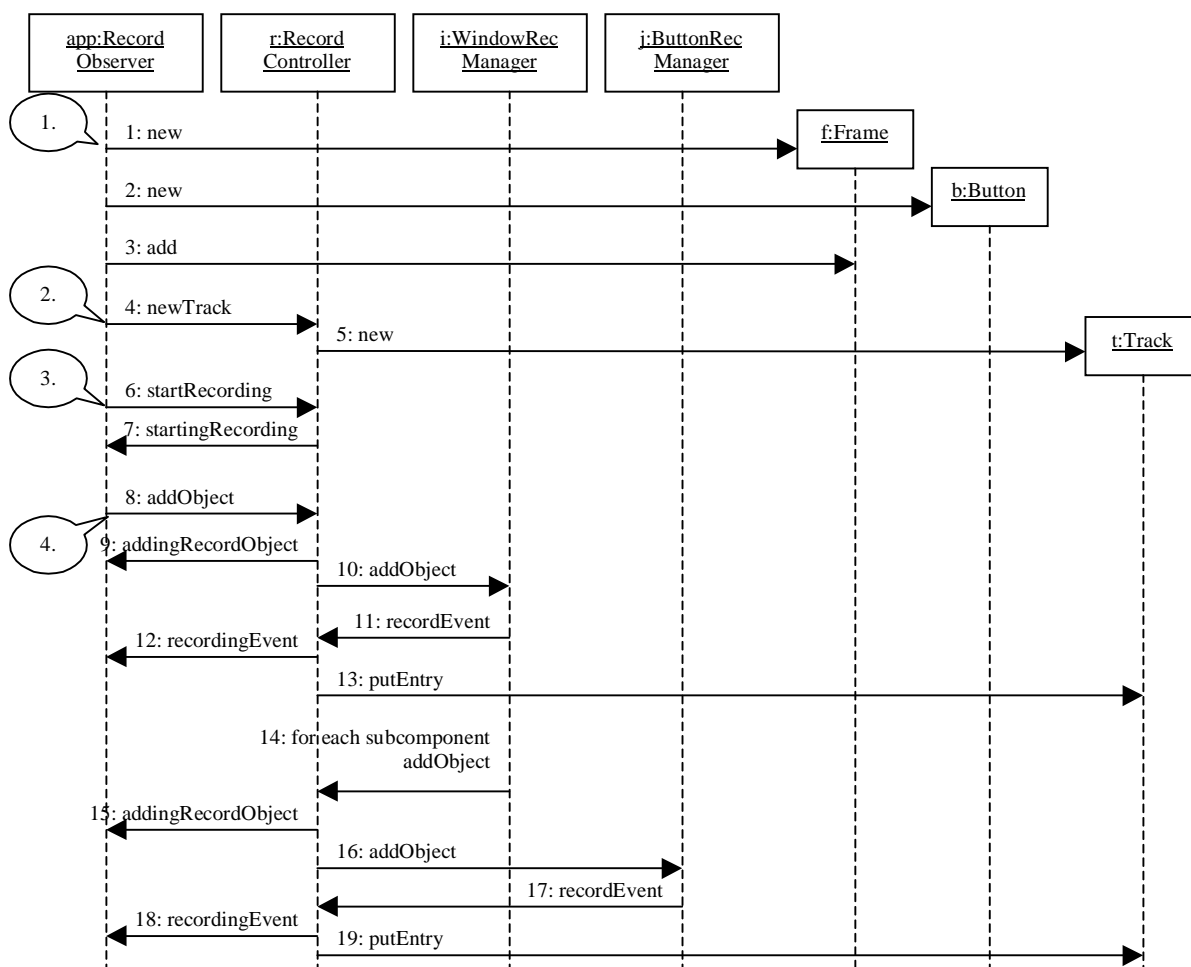


Abbildung 27: Initialisierung einer Aufzeichnung

#### 4. Übergabe der Aufzeichnungsobjekte an den RecordController

Anschließend erfolgt die Übergabe der Aufzeichnungsobjekte an den RecordController. In dem in Abbildung 27 dargestellten Fall übergibt die Anwendung ein Rahmenfenster. (8:) Der RecordController vergibt eine ID für das Objekt, die im Rahmen der aktuellen Aufzeichnung eindeutig ist, und teilt diese dem RecordObserver zusammen mit der Nachricht, daß ein neues Aufzeichnungsobjekt hinzugefügt werden soll, mit. (9:) Der RecordObserver kann an dieser Stelle gegen das Hinzufügen des Objektes votieren. Wenn das Objekt akzeptiert wird, wählt der RecordController anhand der Klasse des Aufzeichnungsobjektes einen RecordSourceManager aus und gibt das Objekt weiter. (10:) Falls für die Klasse des Aufzeichnungsobjektes kein RecordSourceManager existiert, wird versucht einen RecordSourceManager für eine Superklasse des Objektes zu finden.

Der ausgewählte RecordSourceManager ermittelt daraufhin die Daten, die notwendig sind, um bei der Wiedergabe ein passendes Wiedergabeobjekt erzeugen zu können, und gibt ein entsprechendes Ereignis zur Aufzeichnung an den RecordController weiter. (11:)

Der RecordController reicht das Ereignis an den RecordObserver durch (12:), der wiederum die Möglichkeit hat, gegen die Aufzeichnung zu votieren. Falls das Votum positiv ausfällt, wird das entsprechende Ereignis im aktuellen Track gespeichert. (13:)

Anschließend ermittelt der RecordSourceManager ggf. vorhandene Unterobjekte, in diesem Beispiel die Schaltfläche, die sich auf dem Rahmenfenster befindet, und übergibt diese an den RecordController. Die hier beschriebene Prozedur wird entsprechend rekursiv wiederholt. (14: bis 19:)

## **Anmerkungen**

Die ID des Aufzeichnungsobjektes spielt eine besondere Rolle, falls zur Wiedergabe nicht der integrierte Runtime-Player des JRV-Systems sondern die aufgezeichnete Anwendung oder ein anwendungsspezifischer Player verwendet werden soll. In diesem Fall müssen diese bei der Wiedergabe in der Lage sein, die aufgezeichneten Objekte anhand der eindeutigen ID zu identifizieren, um diese Objekte wieder zur Verfügung stellen oder anwendungsspezifische Informationen mit den Wiedergabeobjekten verbinden zu können. Zu diesem Zweck hat die aufgezeichnete Anwendung die Möglichkeit, ID's für Aufzeichnungsobjekte vorzuschlagen. Automatisch vergebene ID's und anwendungsspezifische ID's werden durch getrennte Wertebereiche repräsentiert.

### **3.4.5 Das Aufzeichnen von Ereignissen**

In Abbildung 28 werden die nachfolgend beschriebenen, beispielhaften Situationen, in denen Ereignisse aufgezeichnet werden können, dargestellt. Die im Text in Klammern, in der Form (X:) angegebenen Verweise entsprechen den Teilschritten des in der Abbildung dargestellten Sequenzdiagramms.

#### **1. Aufzeichnung von Benutzerinteraktionen**

Es findet eine Interaktion des Benutzers mit einem Aufzeichnungsobjekt statt. Bspw. wird eine Maustaste gedrückt, während sich der Mauszeiger über der Schaltfläche befindet. (1:) Der zugehörige RecordSourceManager besitzt einen Mechanismus (im konkreten Fall wurde bspw. ein MouseListener für die Schaltfläche registriert), um über solche Ereignisse informiert zu werden. Er verarbeitet das Ereignis und leitet eine Aufforderung zur Aufzeichnung an den RecordController weiter. (2:)

Der RecordController informiert den RecordObserver, der gegen die Aufzeichnung votieren kann. (3:) Falls der RecordObserver der Aufzeichnung zustimmt, wird ein entsprechender Eintrag zur Sicherung des Ereignisses an den Track übergeben. (4:)

#### **2. Durchführung von Requests**

Das Anwendungsprogramm fordert die Durchführung eines sog. „Requests“ an. Im Rahmen eines Requests können RecordSourceManager aufgefordert werden, auf eine bestimmte Situation zu reagieren oder bestimmte Aufgaben für einzelne Aufzeichnungsobjekte oder mit allen Objekten eines bestimmten Typs zu erledigen. Denkbare Einsatzbereiche für Requests sind bspw.:

- **Außerordentliche Ereignisse...** Es tritt eine Zustandsänderung bei einem Aufzeichnungsobjekt ein, die der entsprechende RecordSourceManager aus technischen Gründen nicht automatisch registrieren kann, z.B. weil kein entsprechender Listener zur Verfügung steht. Falls die Wiedergabe des Objektes wirklichkeitsnah erfolgen soll, kann ein entsprechender Request definiert werden, der vom Anwendungsprogramm explizit ausgelöst wird, wenn die Zustandsänderung eingetreten ist.
- **Displaypoints...** Mit Hilfe von sog. „Displaypoints“ könnten Aufzeichnungsobjekte aufgezeichnet werden, die lediglich zur Ausgabe von Daten dienen und nicht auf Benutzerinteraktion reagieren. Jedesmal, wenn ein entsprechendes Objekt neu dargestellt wird, könnte ein Request an den zugehörigen RecordSourceManager geschickt werden, der daraufhin das Erscheinungsbild des Aufzeichnungsobjektes als Bitmap sichert.<sup>31</sup>
- **Savepoints...** RecordSourceManager könnten in regelmäßigen Intervallen aufgefordert werden, sog. „Savepoints“ auszuführen, um die unmittelbare Wiedergabe von bestimmten Punkt der Aufzeichnung oder schnelles Vor- und Zurückspulen bei der Wiedergabe zu erleichtern. Im Rahmen eines Savepoints sollte ein RecordSourceManager Daten für die Aufzeichnungsobjekte in der Form sichern, daß die Wiedergabe von diesem Punkt an möglich ist, ohne daß weitere, vorhergehende Ereignisse verarbeitet werden müssen.<sup>31</sup>

Mit der Aufforderung wird der Typ des Requests sowie das betroffene Aufzeichnungsobjekt oder dessen ID an den RecordController (5:) übermittelt, der anschließend den Request an den RecordSourceManager weiterleitet, der für das entsprechende Aufzeichnungsobjekt zuständig ist. (6:) Der RecordSourceManager verarbeitet den Request, gibt ein entsprechendes Ereignis an den RecordController zur Aufzeichnung weiter, das gesichert wird, nachdem es an den RecordObserver durchgereicht wurde. (7: bis 9:) RecordSourceManager, die nicht in der Lage sind, auf einen Request zu reagieren, ignorieren den Request.

### 3. Entfernen von Aufzeichnungsobjekten

In dem in Abbildung 28 dargestellten Beispiel wird ein Rahmenfenster durch den Benutzer geschlossen. (10:) Der entsprechende RecordSourceManager wird, z.B. durch einen Listener, informiert. An dieser Stelle darf das Rahmenfenster nicht endgültig aus der Menge der Aufzeichnungsobjekte entfernt werden, da dasselbe Fenster im weiteren Verlauf wieder geöffnet werden könnte. Vielmehr gibt der RecordSourceManager ein Ereignis zum Schließen des Fensters an den RecordController weiter, der es nach der Zustimmung des RecordObservers sichert. (11: bis 13:).

Falls ein Aufzeichnungsobjekt endgültig aus der Menge der aufzuzeichnenden Objekte entfernt werden soll, teilt die Anwendung dies dem RecordController mit (14:), der, nachdem die Zustimmung des RecordObservers vorliegt (15:), den entsprechenden RecordSourceManager ermittelt und ihn zum Entfernen des Objektes auffordert. (16:)

Der RecordSourceManager ermittelt ggf. vorhandene untergeordnete Aufzeichnungsobjekte, die als Folge der Entfernung des Elternobjektes ebenfalls entfernt werden müssen, und gibt entsprechende Aufforderungen an den RecordController durch (17:). Dieser wiederum leitet die Aufforderungen, jeweils nach der Zustimmung durch den RecordObserver (18:), an die entsprechenden RecordSourceManager weiter. (19:) Bei der Entfernung der Objekte können Ereignisse in der zuvor bereits geschilderten Art und Weise (Zustimmung durch den RecordObserver, Sicherung durch den Track) aufgezeichnet werden. (20: bis 22:) Nachdem alle untergeordneten Aufzeichnungsobjekte entfernt wurden, zeichnet der RecordSourceManager des Elternobjektes selbst noch ein Ereignis für die Entfernung auf. (23: bis 25:)

---

<sup>31</sup> Anmerkung: Displaypoints und Savepoints sind in der prototypischen Realisierung des JRV-Systems, die im Rahmen dieser Arbeit vorgenommen wurde, nicht implementiert.

#### 4. Aufzeichnung zusätzlicher Ereignisse

Das Anwendungsprogramm fordert die Aufzeichnung eines zusätzlichen Ereignisses an, das mit keinem bestimmten Objekt in Beziehung steht. (26:) Das Ereignis wird, die Zustimmung des RecordObservers vorausgesetzt, durch den RecordController direkt zur Sicherung an den Track weitergeleitet. (27: und 28:)

#### 5. Beenden der Aufzeichnung

Die Aufzeichnung wird beendet. (29:) Der RecordObserver wird hier lediglich informiert. Ein Votum gegen das Ende der Aufzeichnung kann nicht erfolgen. (30:) Der Track wird geschlossen (31:) und, falls gefordert, auf einen Sekundärspeicher geschrieben. (32: und 33:)

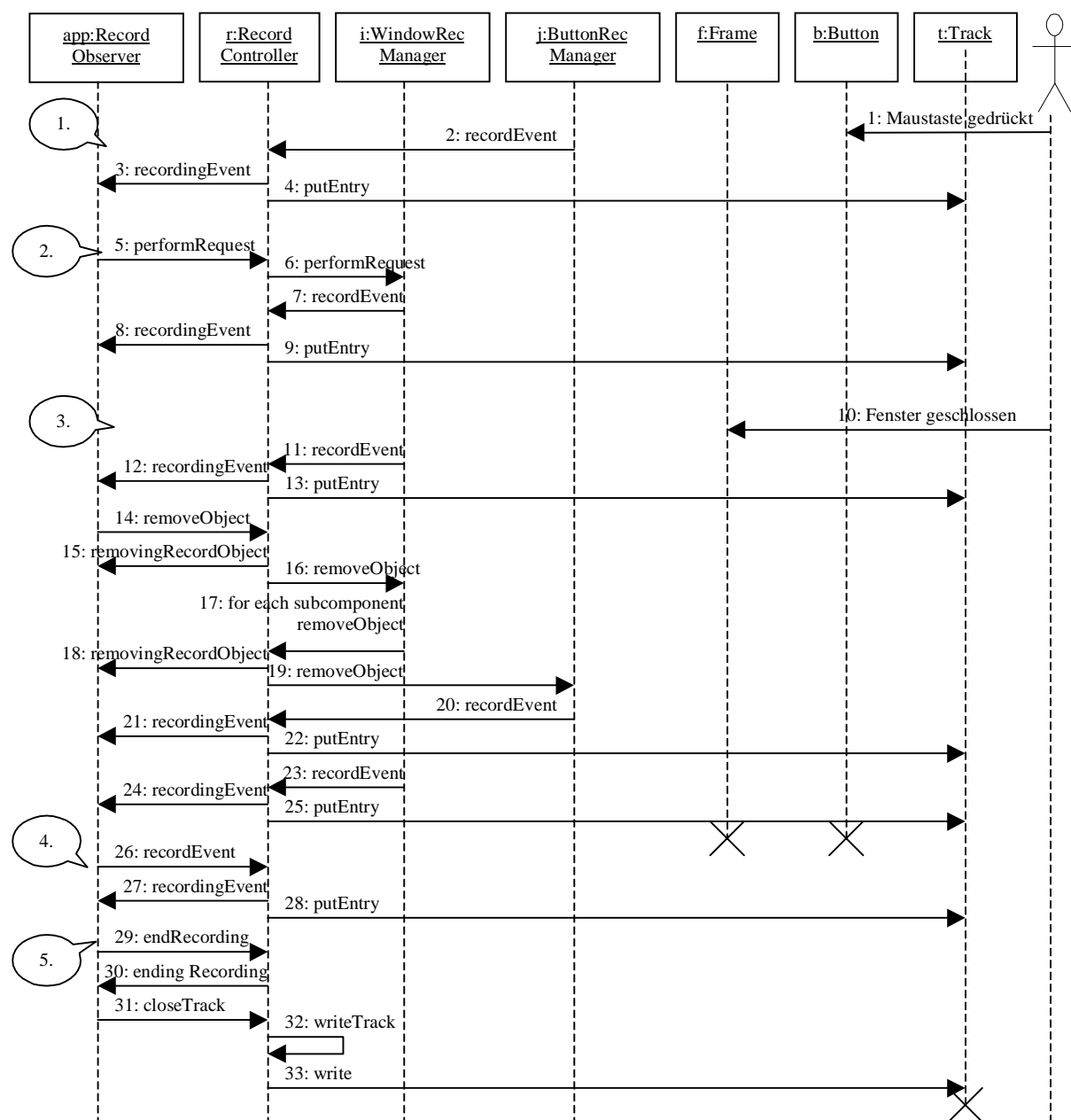


Abbildung 28: Aufzeichnen von Ereignissen

### 3.5 Wiedergabe mit dem JRVPlayer

In diesem Kapitel werden die Mechanismen des JRV-Systems zur Wiedergabe beschrieben und die grundlegenden Konzepte dargestellt. Eine detaillierte Beschreibung des Verhaltens der Klassen und ihrer Methoden liefert der dokumentierte Quellcode des Systems. Kapitel 5 zeigt konkrete Anwendungsbeispiele.

#### 3.5.1 Die Klassen des JRVPlayer-Pakets

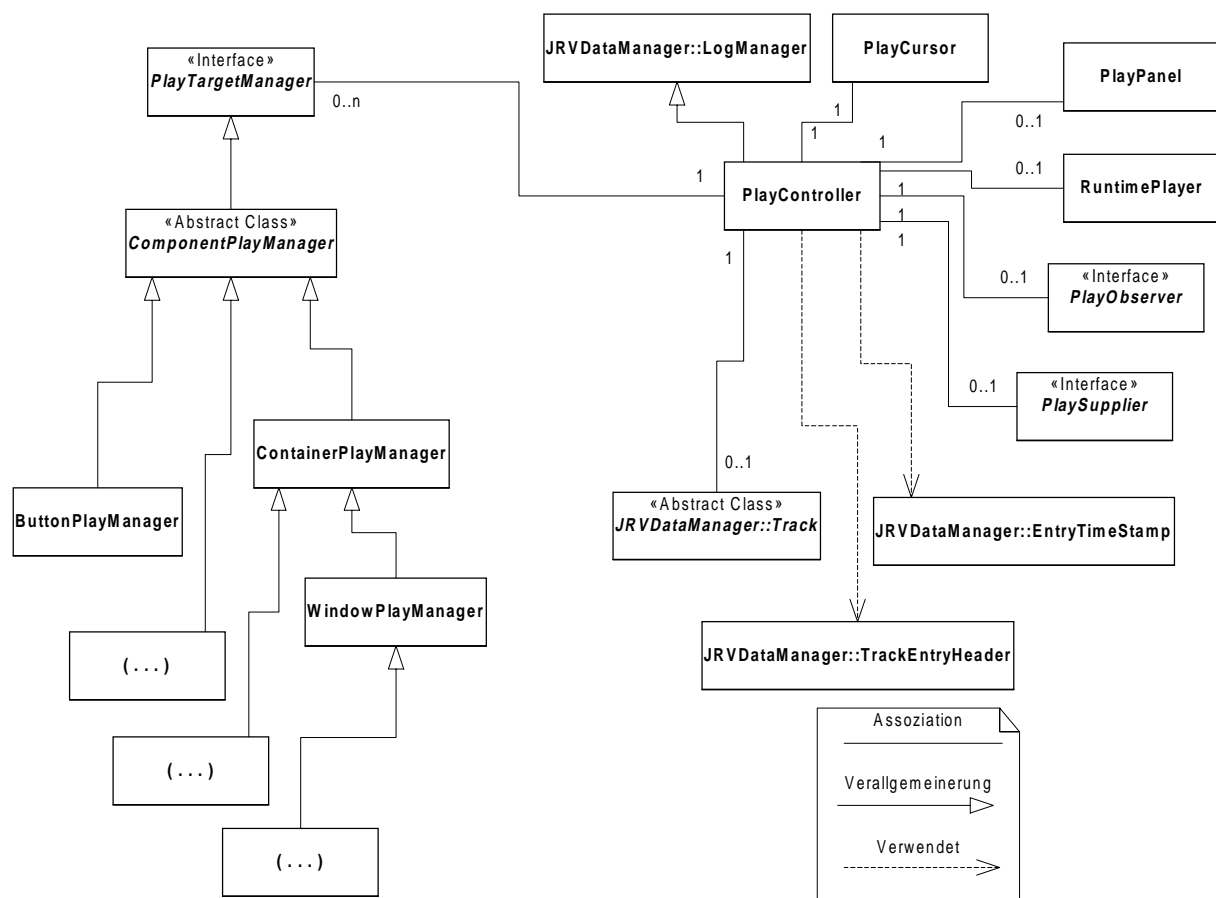


Abbildung 29: Klassen des JRVPlayer-Pakets

Abbildung 29 zeigt eine Übersicht der Klassen des JRVPlayer-Pakets, die im Einzelnen folgende Funktionen übernehmen:

#### a) PlayTargetManager

PlayTargetManager sind Objekte, welche die Wiedergabe einer Aufzeichnungsklasse, z.B. eines bestimmten Typs von Bedienelementen oder Fenster, oder einer Menge von Aufzeichnungsklassen übernehmen. PlayTargetManager werden zur Laufzeit bei einem PlayController (s.u.) registriert und müssen fähig sein, Daten, die durch korrespondierende RecordSource-Manager aufgezeichnet wurden, zu interpretieren.

Mit Hilfe dieser Daten rekonstruieren die PlayTargetManager das optische Erscheinungsbild oder das Verhalten der Aufzeichnungsobjekte entsprechend der Anforderungen, die an den Wiedergabevorgang gestellt werden.

Durch das PlayTargetManager-Interface werden die Signaturen der Methoden definiert, über die ein PlayController mit PlayTargetManager-Objekten kommuniziert. Das JRVPlayer-Paket enthält Implementierungen des PlayTargetManager-Interfaces zur Wiedergabe ausgewählter Klassen der AWT- und der JFC/Swing-Bibliothek.

### **b) PlayController**

Der PlayController ist die zentrale Komponente zur Kontrolle des Wiedergabeprozesses. Er verwaltet PlayTargetManager und Wiedergabeobjekte und übernimmt die Kommunikation mit dem RuntimePlayer (s.u.) oder dem wiedergebenden Anwendungsprogramm sowie mit den Komponenten des JRVDatamanager-Pakets, das den Datentyp Track für die Verwaltung von aufgezeichneten Daten zur Verfügung stellt. Darüber hinaus bietet der RecordController Methoden, über die wahlweise das wiedergebende Programm oder der Endbenutzer mit Hilfe eines PlayPanels (s.u.) den Wiedergabeprozess steuern, d.h. die Wiedergabe starten, anhalten und beenden kann.

### **c) PlayCursor**

Die PlayCursor-Klasse stellt Methoden zur Verfügung, um einen Mauszeiger willkürlich, d.h. entsprechend einer Aufzeichnung, bewegen zu können.

### **d) PlayPanel**

Das PlayPanel stellt in Form eines Dialogfeldes eine Benutzungsschnittstelle zur Verfügung, über die der Endbenutzer die Wiedergabe steuern, d.h. starten, anhalten und beenden kann.

### **e) RuntimePlayer**

Der RuntimePlayer ist eine ausführbare Klasse, die ein Programm zur optischen Wiedergabe solcher Aufzeichnungen implementiert, für die nur Standardklassen notwendig sind.

### **f) PlayObserver**

Das PlayObserver-Interface definiert eine Reihe von Callback-Funktionen, die durch den PlayController während des Wiedergabeprozesses beim Eintreten bestimmter Ereignisse, z.B. beim Hinzufügen oder Entfernen von Wiedergabeobjekten sowie vor der Wiedergabe von Ereignissen aufgerufen werden. Ein wiedergebendes Anwendungsprogramm kann einen PlayObserver zur Verfügung stellen, um auf einzelne Ereignisse gezielt reagieren zu können.

### **g) PlaySupplier**

Das PlaySupplier-Interface definiert eine Callback-Funktion, die durch den PlayController aufgerufen wird, wenn die Wiedergabe nicht über den RuntimePlayer (s.o.) sondern über die aufgezeichnete Anwendung oder einen anwendungsspezifischen Player erfolgt. Mit Hilfe der Callback-Funktion werden ID's von Aufzeichnungsobjekten mit solchen Wiedergabeobjekten verbunden, die nicht durch das JRV-System sondern durch die wiedergebende Anwendung erzeugt werden.

### 3.5.2 Das Zusammenspiel der Komponenten des JRVPlayer-Pakets

Abbildung 30 zeigt das Zusammenspiel der Komponenten des JRVPlayer-Pakets sowohl untereinander als auch in der Wechselwirkung mit der wiedergebenden Anwendung sowie mit dem JRVDatamanager-Paket. Wenn es sich bei der wiedergebenden Anwendung um den RuntimePlayer des JRVPlayer-Pakets handelt, entfällt das PlaySupplier-Objekt.

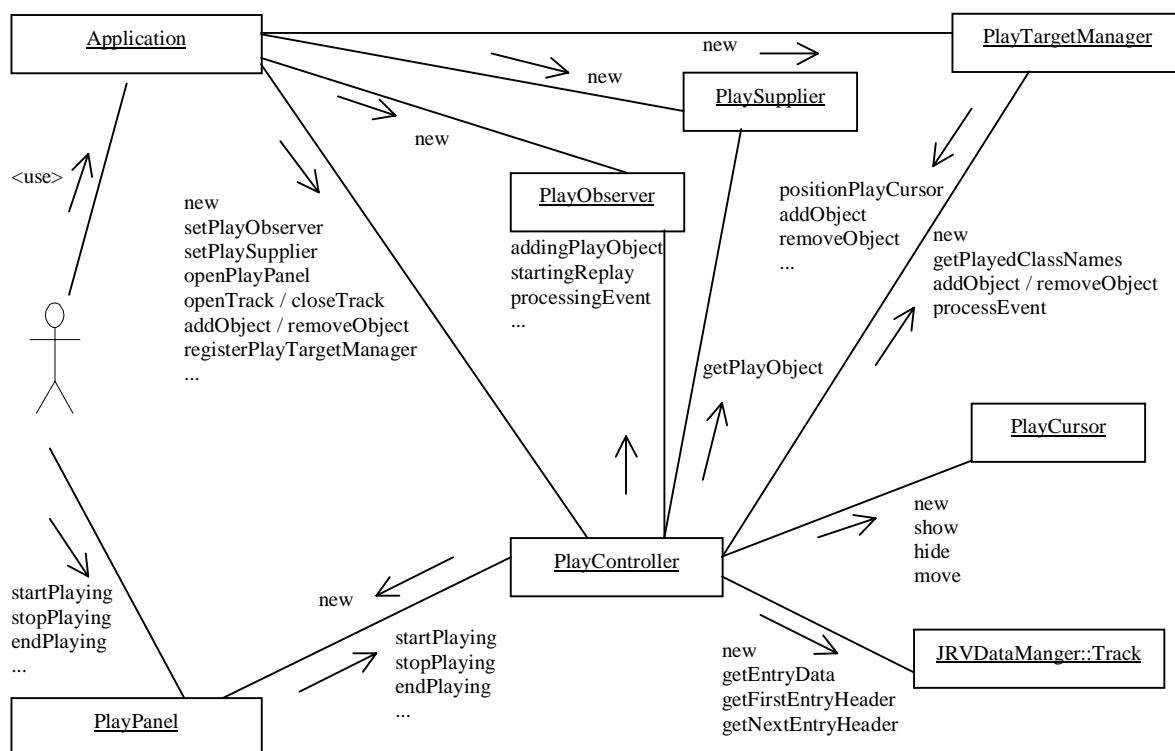


Abbildung 30: Das Zusammenspiel der Wiedergabekomponenten mit einem Anwendungsprogramm

### 3.5.3 Die Initialisierung des Wiedergabesystems

Abbildung 31 zeigt die nachfolgend beschriebenen Schritte zur Initialisierung des JRVPlayer-Systems. Die im Text in Klammern, in der Form (X:) angegebenen Verweise entsprechen den Teilschritten des in der Abbildung dargestellten Sequenzdiagramms. Falls die RuntimePlayer-Klasse zur Wiedergabe verwendet wird, werden nur die Schritte 1., 2. und 4. ausgeführt.

#### 1. Erzeugung eines PlayControllers (1:)

#### 2. Automatische Registrierung standardmäßig vorhandener PlayTargetManager

Der PlayController erzeugt automatisch PlayTargetManager für die Wiedergabeklassen, die standardmäßig durch das JRV-System unterstützt werden (2: und 5:). In dem in Abbildung 31 gezeigten Beispiel für Fenster (Windows) und Schaltflächen (Buttons). Er registriert diese bei sich selbst (3: und 6:) und fragt die Klassen der Aufzeichnungsobjekte ab, die durch den jeweiligen PlayTargetManager wiedergegeben werden können. (4: und 7:) Außerdem wird ein Timer erzeugt, bei dem sich der RecordController als ActionListener registriert. (8: und 9:)

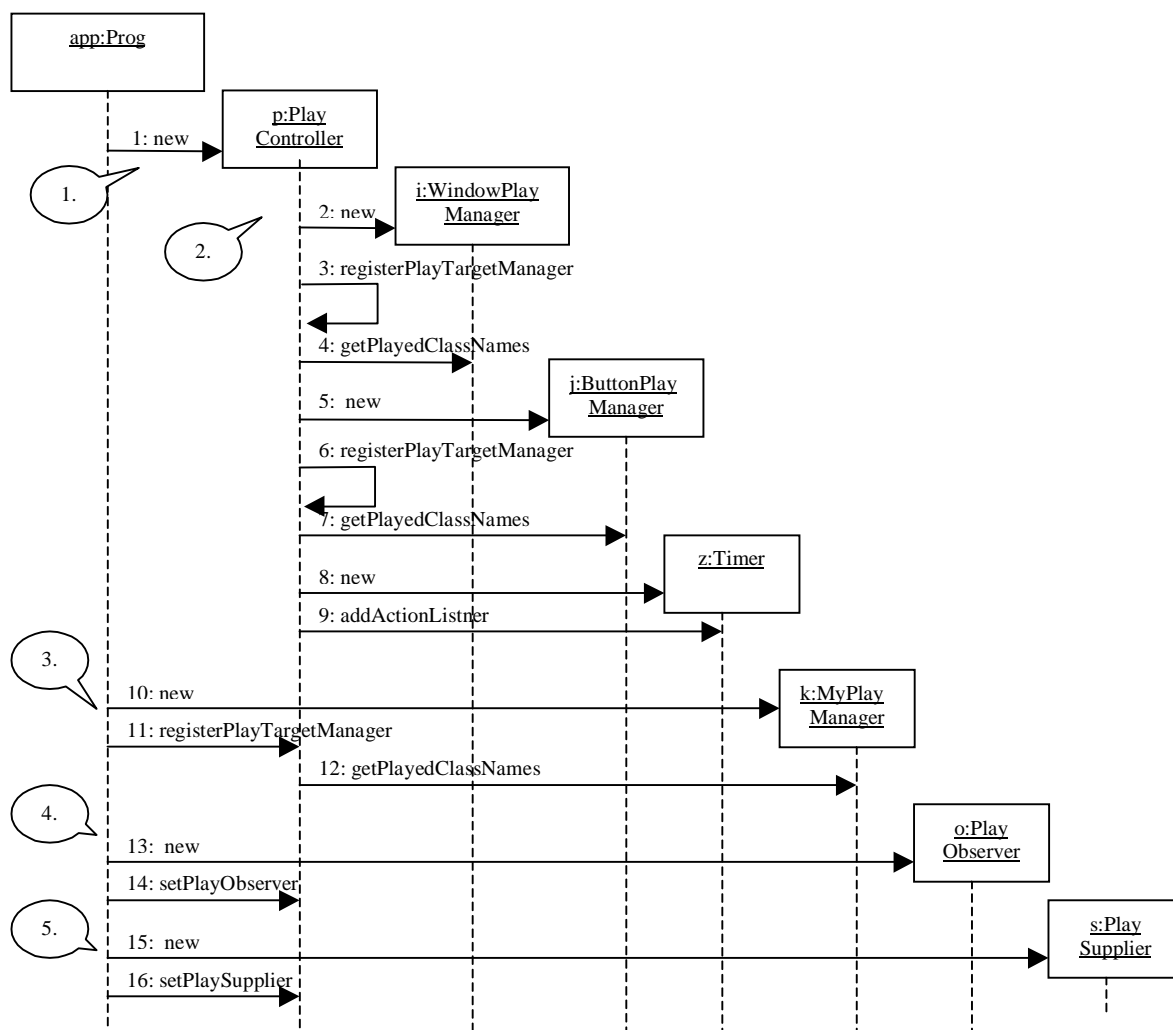


Abbildung 31: Initialisierung des Wiedergabesystems

### 3. Registrierung anwendungsspezifischer PlayTargetManager

Wenn die wiedergebende Anwendung weitere PlayTargetManager zur Verfügung stellt, werden diese ebenfalls erzeugt und beim PlayController registriert. (10: bis 12:) Falls dieselbe Aufzeichnungsklasse durch mehrere PlayTargetManager wiedergegeben werden kann, wird der zuletzt registrierte PlayTargetManager verwendet, so daß die standardmäßig vorhandenen PlayTargetManager durch anwendungsspezifische ersetzt werden können.

### 4. Registrierung eines PlayObservers

Falls die Wiedergabe auf Ereignisebene kontrolliert werden soll, wird ein PlayObserver an den PlayController übergeben. (13: und 14:)

### 5. Registrierung eines PlaySuppliers

Falls bestimmte Wiedergabeobjekte nicht durch PlayTargetManager erzeugt, sondern durch die wiedergebende Anwendung zur Verfügung gestellt bzw. mit existierenden Objekten verbunden werden sollen, wird ein PlaySupplier registriert. (15: und 16:)

### 3.5.4 Die Wiedergabe mit dem Runtime-Player

Abbildung 32 zeigt die nachfolgend beschriebenen, beispielhaften Situationen bei der Wiedergabe von Standardklassen und mit Hilfe des Runtime-Players. Die im Text in Klammern, in der Form (X:) angegebenen Verweise entsprechen den Teilschritten des in der Abbildung dargestellten Sequenzdiagramms.

#### 1. Öffnen eines Tracks

Der Track, der die gewünschte Aufzeichnung enthält, wird geöffnet. (1: und 2:)

#### 2. Start der Wiedergabe

Der Start der Wiedergabe wird dem RecordController mitgeteilt. (3:) Da der RuntimePlayer sich selbst als PlayObserver registriert hat, erfolgt daraufhin ein Aufruf dieses PlayObservers (4:), der an dieser Stelle gegen den Start der Wiedergabe votieren kann. Der RuntimePlayer implementiert das PlayObserver-Interface nur, um am Ende das Dialogfeld zur Kontrolle der Wiedergabe automatisch schließen zu können und beantwortet daher alle Aufrufe von PlayObserver-Methoden positiv.

Anschließend wird der Header des ersten Eintrags des Tracks gelesen (5:) und der Zeitgeber gestartet. (6:) Der Header des Eintrags wird nun so lange für die Verarbeitung vorgehalten, bis durch den Zeitgeber ein Ereignis ausgelöst wird, dessen Zeitpunkt, bezogen auf den Start der Wiedergabe, größer oder gleich der Zeitmarke des Eintrags ist.

#### 3. Erzeugen von Wiedergabeobjekten

Sobald eine Nachricht des Zeitgebers beim PlayController eingeht, die anzeigt, daß die Zeit für die Wiedergabe des nächsten Eintrags erreicht ist (7:), werden die Daten des Eintrags gelesen (8:). Der Eintrag wird an den PlayObserver durchgereicht (9:) und nach einem positiven Votum an denjenigen PlayTargetManager weitergeleitet, der für die Wiedergabe der Aufzeichnungsklasse, die im Header des Eintrags angegeben ist, registriert wurde. (10:)

Wenn es sich bei dem Ereignis, das durch den Eintrag beschrieben wird, um die Erzeugung eines Objektes handelte, wird ein entsprechendes Wiedergabeobjekt durch den PlayTargetManager erzeugt (11:) und an den PlayController übergeben. (12:) Sobald der PlayObserver dem Hinzufügen des Objektes zugestimmt hat (13), stellen der PlayTargetManager und der PlayController eine feste Beziehung zwischen dem Wiedergabeobjekt und der ID des Aufzeichnungsobjektes her, die im Header des Eintrags enthalten ist. Alle folgenden Ereignisse mit der gleichen ID werden mit demselben Wiedergabeobjekt verbunden.<sup>32</sup>

Nach der Bearbeitung des Ereignisses wird der Header des nächsten Ereignisses gelesen (14:) und, wie in diesem Beispiel vorausgesetzt, unmittelbar verarbeitet, wenn seine Zeitmarke kleiner oder gleich der Zeit ist, die seit dem Start der Wiedergabe vergangen ist. Die Daten des Eintrags werden gelesen (15:) und der Eintrag an den PlayObserver weitergereicht. (16:) Die Zustimmung des PlayObservers vorausgesetzt, erfolgt anschließend die Verarbeitung der Eintrags. (17:)

---

<sup>32</sup> Anmerkung: Bei der Aufzeichnungsklasse und der Wiedergabeklasse muß es sich nicht zwangsläufig um dieselbe Klasse handeln. Die Wiedergabeklasse muß lediglich das Verhalten von Objekten der Aufzeichnungsklasse in dem Maße imitieren können, wie es der Anwendungszweck der Players erfordert.

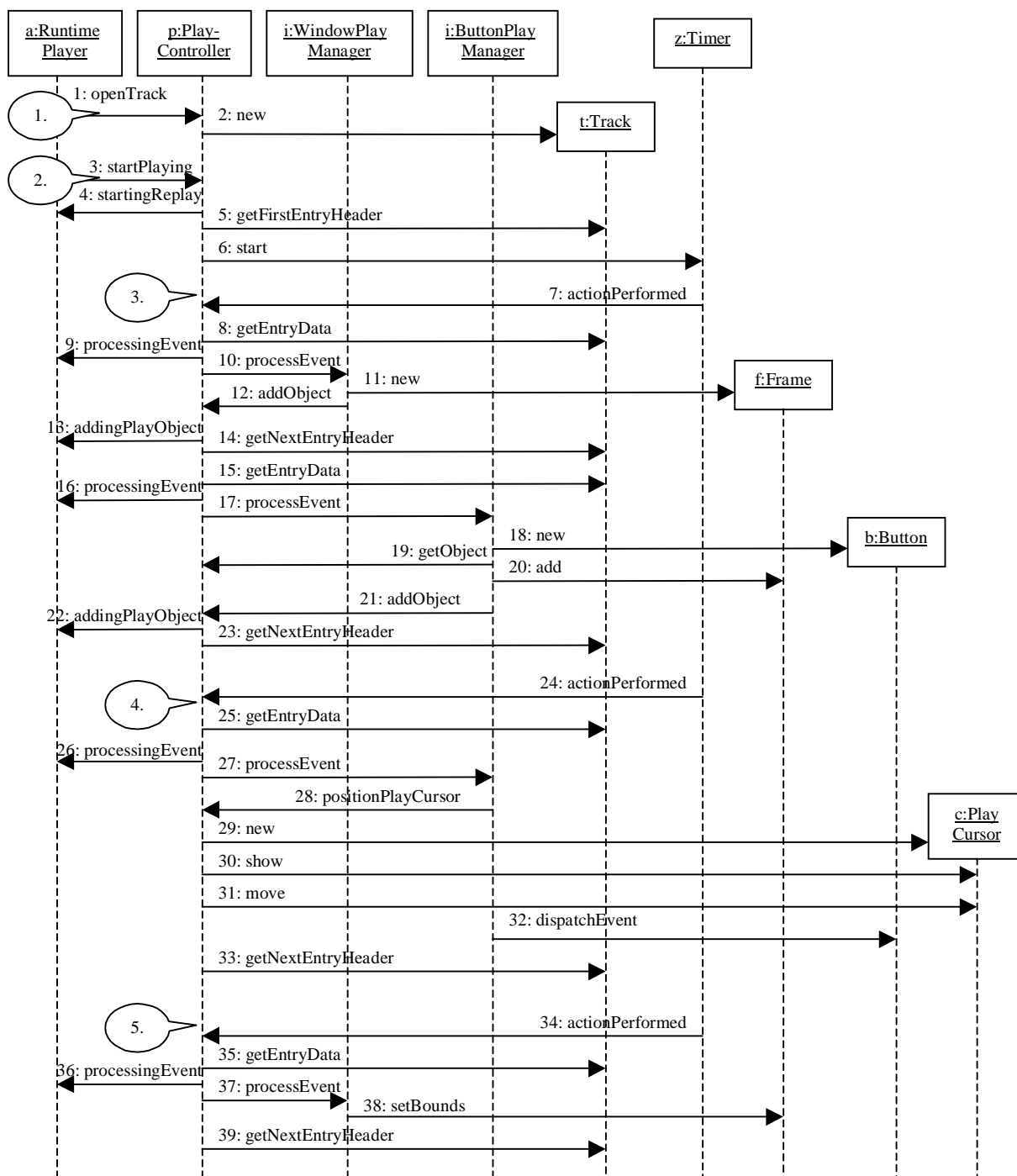


Abbildung 32: Wiedergabe mit dem Runtime-Player

Bei der Erzeugung von untergeordneten Wiedergabeobjekten, in dem in Abbildung 32 gezeigten Beispiel handelt es sich um einen Button (18:), erfragt der PlayTargetManager das zugehörige übergeordnete Wiedergabeobjekt, dessen ID in dem Eintrag angegeben ist, bei dem dem PlayController (19:) und stellt die Verbindung zwischen den beiden Objekten her. (20:) Anschließend wird das neu erzeugte Wiedergabeobjekt an den PlayController zur Registrierung übergeben und nach der Zustimmung des PlayObservers der Header des nächsten Eintrags gelesen (21: bis 23:)

#### 4. Wiedergabe von Mausereignissen

Wenn eine Aktion des Mauszeigers wiedergegeben werden soll, wird der entsprechende Eintrag zunächst an den PlayTargetManager weitergeleitet, der für die Wiedergabe der Klasse des Aufzeichnungsobjektes zuständig ist, über dem das Ereignis aufgezeichnet wurde. (24: bis 27:) Der PlayTargetManager berechnet anhand des Ereignisses und der Position des zugehörigen Wiedergabeobjektes die Bildschirmposition, an welche der Mauszeiger bewegt werden soll und gibt eine entsprechende Anweisung an den PlayController weiter. (28:)

Wenn der PlayController die erste Anweisung zur Positionierung des Mauszeigers erhält, wird ein PlayCursor erzeugt und angezeigt. (29: und 30:) Im übrigen wird der PlayCursor an die entsprechende Position bewegt. (31:) Anschließend erzeugt der PlayTargetManager ein Ereignisobjekt, das wie ein „echtes“, d.h. durch den Benutzer ausgelöstes Mausereignis an das betreffende Wiedergabeobjekt übermittelt wird.<sup>33</sup> (32:) Die Verarbeitung des Mausereignisses ist damit abgeschlossen. Der Header des nächsten Eintrags wird gelesen (33:)

#### 5. Wiedergabe von sonstigen Ereignissen

Alle übrigen Ereignisse werden nach demselben Schema verarbeitet: Sobald die Zeit für die Wiedergabe eines Eintrags gekommen ist (34:) wird der Eintrag durch den PlayController gelesen (35:), an den PlayObserver durchgereicht (36) und anschließend an den PlayTargetManager weitergeleitet, der die Klasse, die im Header des Eintrags vermerkt ist, wiedergeben kann. (37:). Der PlayTargetManager verarbeitet das Ereignis. In dem letzten in Abbildung 32 gezeigten Beispiel handelt es sich um die Anpassung der Größe des Rahmenfensters. (38:) Anschließend wird wieder der Header des nächsten Ereignisses gelesen. (39:)

### 3.5.5 Die Wiedergabe durch einen anwendungsspezifischen Player

Die Wiedergabe durch einen anwendungsspezifischen Player oder durch die aufgezeichnete Anwendung selbst erfolgt prinzipiell nach demselben Schema, wie im vorherigen Abschnitt beschrieben. Jedoch kann der Player mit Hilfe eines PlayObservers aufgrund von Ereignissen zusätzliche Aktionen auslösen oder gegen die Wiedergabe von Ereignissen votieren. Darüber hinaus kann ein PlaySupplier verwendet werden, um bestehende Objekte als Wiedergabeobjekte mit den ID's zu verknüpfen, die bestimmte Aufzeichnungsobjekte referenzieren.

Dies wird bspw. notwendig sein, wenn Wiedergabeobjekte vor dem Start der Wiedergabe existieren müssen, um ihre normale Funktion im Rahmen der Anwendungslogik zu erfüllen oder weil für die Erzeugung der Objekte Daten notwendig sind, die bei der Aufzeichnung nicht aufgezeichnet werden konnten, bspw. Daten, welche die Systemumgebung oder den Standort des Rechners bei der Wiedergabe betreffen.

Abbildung 33 demonstriert diese Situation. Die im Text in Klammern, in der Form (X:) angegebenen Verweise entsprechen wieder den Teilschritten des in der Abbildung dargestellten Sequenzdiagramms.

---

<sup>33</sup> Anmerkung: Das Weiterleiten von Mausereignissen an die Bedienelemente in solch einer Form, daß diese wie „echte“, d.h. wie durch den Benutzer ausgelöste Ereignisse verarbeitet werden, dient dazu, die Bedienelemente zu veranlassen, grafische Effekte, die mit bestimmten Aktionen der Maus verbunden sind, darzustellen. Bspw. wird eine Schaltfläche vertieft dargestellt, wenn sie mit der linken Maustaste angeklickt wird. Eine Diskussion, wann und inwieweit dies immer möglich oder sinnvoll ist, erfolgt in Abschnitt 4.2.5.

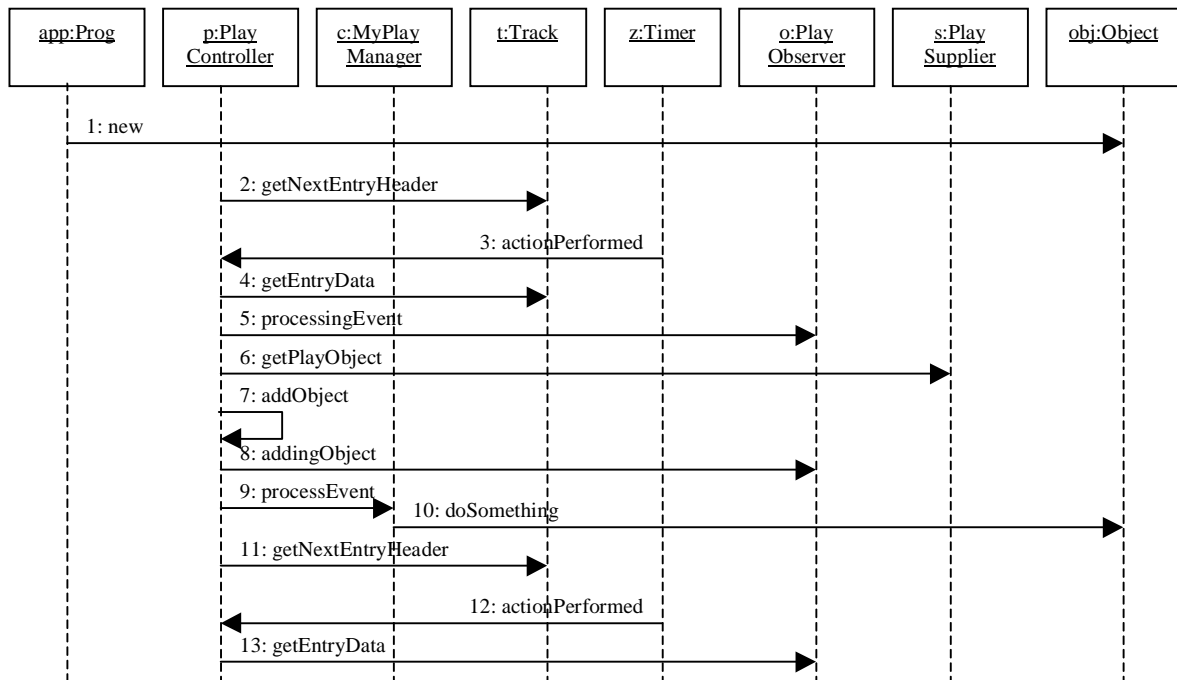


Abbildung 33: Wiedergabe mit einem anwendungsspezifischen Player

### Verbinden von beliebigen Objekten mit der Wiedergabe

In Abbildung 33 wird zunächst ein Objekt, das später als Wiedergabeobjekt fungieren soll, von der wiedergebenden Anwendung erzeugt. (1:) Die weiteren Schritte zur Initialisierung des Wiedergabesystems, wie das Erzeugen der PlayTargetManager, das Öffnen eines Tracks usw. werden in Abbildung 33 nicht gezeigt. Die entsprechenden Abläufe wurden in den Abschnitten 3.5.3 und 3.5.4 beschrieben.

Wenn der Zeitpunkt für die Bearbeitung eines Ereignisses erreicht ist (2: und 3:), werden die Daten des Eintrags gelesen (4:) und das Ereignis an den PlayObserver weitergereicht (5:), der gegen die Wiedergabe votieren kann. Falls der PlayController noch kein Wiedergabeobjekt für die ID, die im Header des Eintrags angegeben ist, zur Verfügung hat, wird das Objekt bei dem PlaySupplier angefragt. (6:) Das Objekt, das vom PlaySupplier geliefert wurde, wird anschließend bei dem RecordController selbst registriert. (7:) Nach der Mitteilung an den PlayObserver (8:) wird das Ereignis dann an den PlayTargetManager weitergeleitet, der für die Wiedergabe von Objekten der entsprechenden Aufzeichnungsklasse registriert ist. (9:) Der PlayTargetManager verarbeitet das Ereignis (10:) Der Header des nächsten Eintrags wird gelesen und nach Erreichen der entsprechenden Zeitmarke verarbeitet. (11: bis 13:)

### 3.6 Datenorganisation mit dem JRVDataManager

Im Folgenden werden die Mechanismen des JRV-Systems zur Verwaltung von Aufzeichnungen in Form von Tracks sowie zur Serialisierung auf Sekundär Speichermedien beschrieben. Dabei werden hier die grundlegenden Konzepte dargestellt. Eine detaillierte Beschreibung des Verhaltens der Klassen und Ihrer Methoden liefert der dokumentierte Quellcode des Systems.

#### 3.6.1 Die Klassen des JRVDataManager-Pakets

Abbildung 34 zeigt die Klassen des JRVDataManager-Pakets, die im Einzelnen die nachfolgend beschriebenen Funktionen übernehmen.

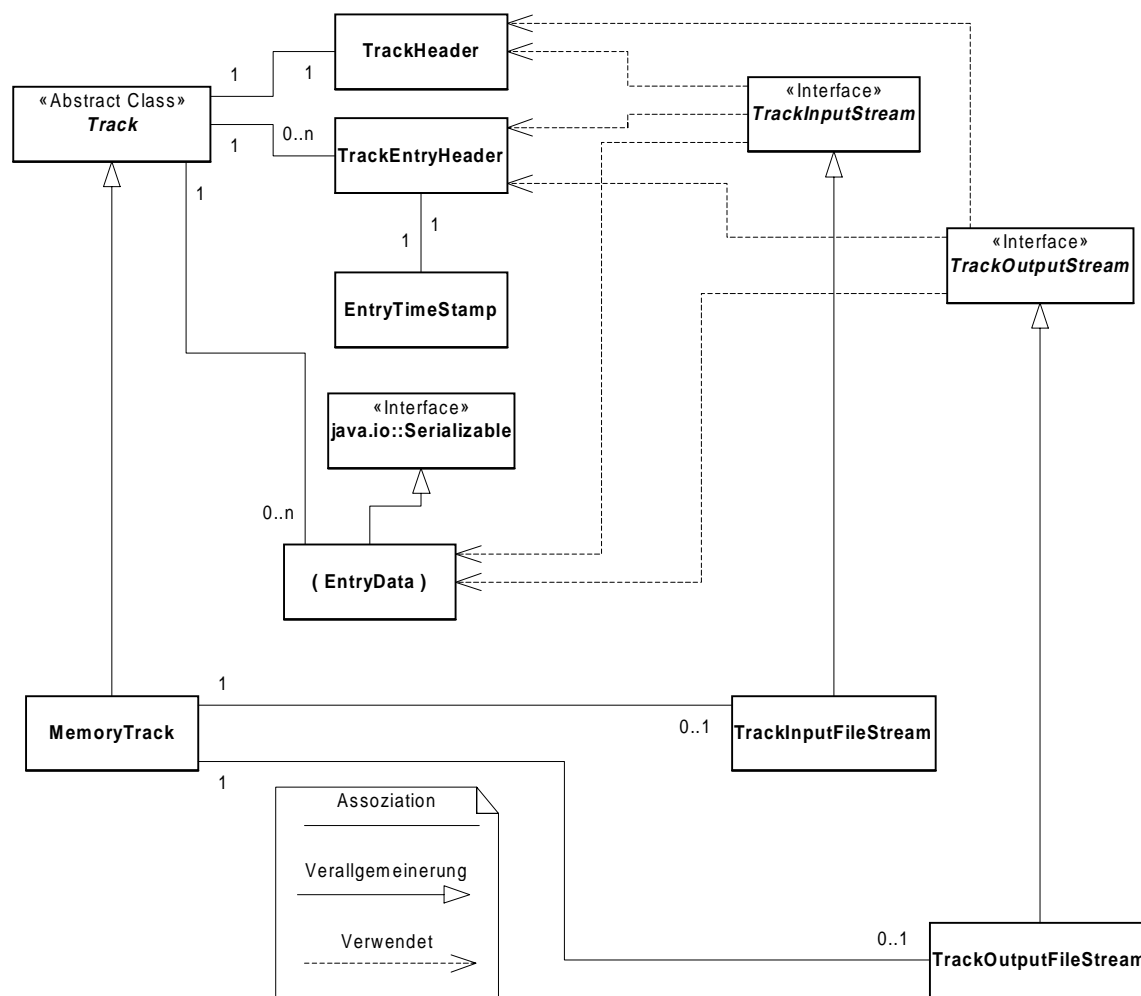


Abbildung 34: Datentypen und Klassen des JRVDataManager-Pakets

### a) **Track / MemoryTrack**

Die abstrakte Klasse `Track` definiert die Signaturen der Methoden, über die `RecordController` und `PlayController` Daten einer Aufzeichnung abrufen und übergeben können. Die Klasse `MemoryTrack` implementiert diese Methoden für Aufzeichnungen, die sich (wenigstens) vorübergehend komplett im Hauptspeicher befinden können. Die `Track`-Klasse ermöglicht einen wahlfreien, lesenden und schreiben Zugriff auf den Inhalt eines Tracks, im Gegensatz zu den Klassen `TrackInputStream` und `TrackOutputStream` (s.u.) die ausschließlich sequentiellen Zugriff erlauben.

### b) **TrackHeader**

`TrackHeader` beschreiben globale Eigenschaften eines Tracks, wie die Bezeichnung oder den Zeitpunkt der Erzeugung.

### c) **TrackEntryHeader**

`TrackEntryHeader` beschreiben einen einzelnen Eintrag eines Tracks. Der `TrackEntryHeader` enthält die Bezeichnung der Aufzeichnungs-klasse und die ID des Aufzeichnungsobjektes, über dem das Ereignis, das durch den Eintrag repräsentiert wird, aufgezeichnet wurde, sowie eine Zeitmarke in Form eines `EntryTimeStamps` (s.u.) und die Bezeichnung des Ereignisses, das durch die Daten des Eintrags repräsentiert wird.

### d) **EntryTimeStamp**

Durch `EntryTimeStamps` wird eine Ordnung auf den Einträgen eines Tracks definiert, welche die Reihenfolge bei der Wiedergabe bestimmt. Die Zeitmarke eines Eintrags setzt sich aus dem Zeitpunkt der Aufzeichnung, relativ zum Beginn der Aufzeichnung, und einer Sequenznummer zusammen, die zur Unterscheidung von Ereignissen dient, die zu demselben Zeitpunkt aufgezeichnet wurden.

### e) **EntryData / Serializable**

Die Daten eines jeden Eintrags bestehen aus einem beliebigen, serialisierbaren Objekt.<sup>34</sup>

### f) **TrackInputStream / TrackInputFileStream**

Das `TrackInputStream`-Interface definiert die Signaturen der Methoden, die verwendet werden, um den Inhalt eines Tracks in Form eines Datenstroms zu lesen. Die Klasse `TrackInputFileStream` stellt eine Implementierung dieses Interfaces zur Verfügung, mit deren Hilfe Tracks aus Dateien gelesen werden können.

### g) **TrackOutputStream / TrackOutputFileStream**

Das `TrackOutputStream`-Interface definiert die Signaturen der Methoden, die verwendet werden, um den Inhalt eines Tracks in Form eines Datenstroms zu schreiben. Die Klasse `TrackOutputFileStream` stellt eine Implementierung dieses Interfaces zur Verfügung, mit der Tracks in Dateien geschrieben werden können.

---

<sup>34</sup> Anmerkung: Nähere Angaben zur Serialisierung von Objekten unter Java erfolgen in Abschnitt 4.2.2.

### 3.6.2 Das Zusammenspiel der Komponenten des JRVDatamanager-Pakets

Abbildung 35 zeigt das Zusammenspiel der Komponenten des JRVDatamanager-Pakets untereinander und in der Wechselwirkung mit den Klassen RecordController und PlayController als Benutzer.

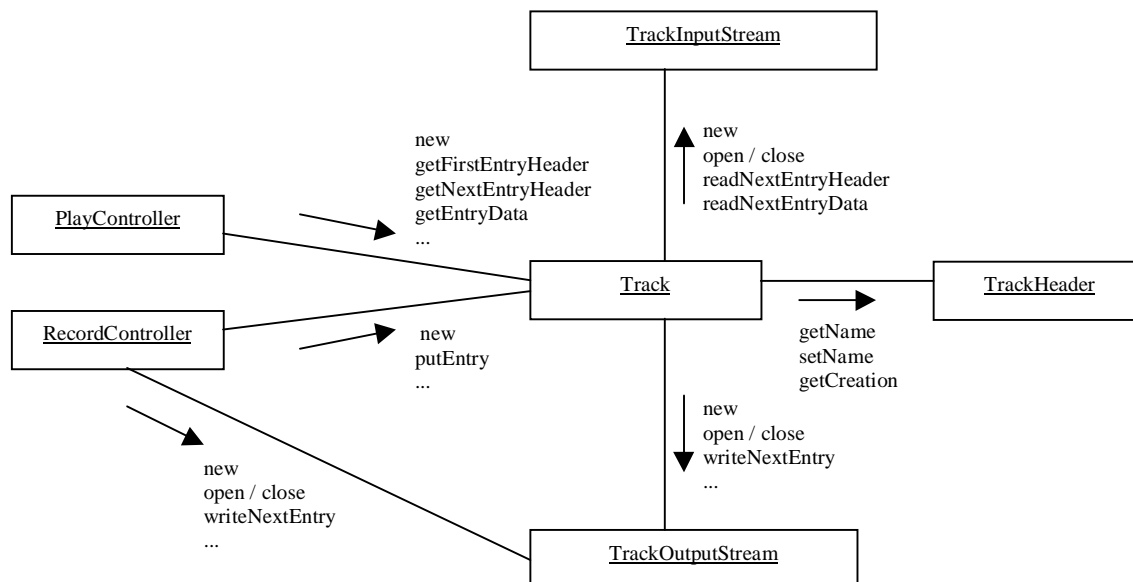


Abbildung 35: Das Zusammenspiel der Komponenten des JRVDatamanager-Pakets

### 3.7 Hilfsklassen und Konstanten im JRVEvents-Paket

Das JRVEvents-Paket enthält Hilfsklassen, die in den Paketen JRVRecorder, JRVPlayer und z.T. auch JRVDatamanager gleichermaßen benötigt werden. Diese Klassen übernehmen bspw. folgende Funktionen:

- Die Definition von serialisierbaren Datenstrukturen zur Beschreibung von Ereignissen bei der Aufzeichnung und Wiedergabe, die nicht durch standardmäßig verfügbare Java-Klassen beschrieben werden können.
- Die Definition von Bezeichnungen, konstanten Werten und Aufzählungen.

Es handelt sich hierbei um relativ einfache Klassen, die zur Gruppierung von strukturierten, zusammengesetzten Werten dienen und deren Funktion sich im einzelnen aus dem Kontext ergibt, in dem sie verwendet werden. Eine detaillierte Beschreibung erfolgt im dokumentierten Quellcode des JRV-Systems.

## 4. Die Implementierung des JRV-Systems

In diesem Kapitel werden ausgewählte Aspekte der Implementierung des JRV-Systems beschrieben, mögliche Alternativen für einige Vorgehensweisen genannt und begründet, warum die eine oder die andere Alternative gewählt wurde. Eine detailliertere Beschreibung der Implementierung kann dem dokumentierten Quellcode entnommen werden.<sup>35</sup>

### 4.1 Entwicklungs- und Systemumgebung

Auch wenn Java, wie in Abschnitt 2.3.3 geschildert, grundsätzlich auf Plattformunabhängigkeit ausgelegt ist, konnten, bzw. mußten bei der Implementierung des JRV-Systems einige Entscheidungen im Hinblick auf die Entwicklungsumgebung und in Abhängigkeit von dem zugrundeliegenden Betriebssystem getroffen werden. Teils aus pragmatischen Gründen, teils aufgrund von technischen Anforderungen. Diese Entscheidungen betrafen im Wesentlichen die nachfolgend beschriebenen Aspekte.

#### 4.1.1 Das Java Development Kit

Die prototypische Implementierung des JRV-Systems, die im Rahmen dieser Arbeit vorgenommen wurde, verwendet das JDK 1.1.6 und die Swing-Bibliothek in der Version 1.0.1. Auf eine Umstellung auf das JDK 1.2 bzw. die „Java 2 Platform“, die in ihrer Endversion ca. vier Wochen vor Abschluß der Arbeit zur Verfügung stand, wurde sowohl aus Zeitgründen, als auch wegen der besseren, vollständigen Integration des JDK 1.1.6 in die zu diesem Zeitpunkt verfügbare Version der verwendeten Entwicklungsumgebung (siehe Abschnitt 4.1.2) verzichtet.

#### 4.1.2 Die JBuilder 2 Entwicklungsumgebung von Inprise

Das von der Firma Sun unter <http://java.sun.com> angebotene JDK enthält eine kommandozeilenorientierte Entwicklungsumgebung. Für die Entwicklung von Anwendungen, die aus zahlreichen Klassen bestehen sowie für die Fehlersuche und den Programmtest bietet eine visuelle, integrierte Entwicklungsumgebung Vorteile. Insbesondere um das Verhalten der einzelnen AWT- und JFC/Swing-Komponenten analysieren zu können, war z.T. intensives Debugging notwendig. Aufgrund dieser Vorteile und zur Beschleunigung der Entwicklung wurde als Entwicklungsumgebung JBuilder 2 Standard der Firma Inprise (ehemals Borland) verwendet. Informationen zu JBuilder 2 sind unter <http://www.inprise.com> erhältlich.

---

<sup>35</sup> Anmerkung: Beispielprogramme, die bestimmte, in diesem Kapitel beschriebene Effekte dokumentieren, sind dem Quellcode des JRV-Systems ebenfalls beigelegt.

### 4.1.3 Der Mauszeiger als betriebssystemabhängige Komponente

Die Implementierung und der Test des JRV-Systems erfolgte unter Microsoft Windows. Die Frage des Betriebssystems spielte in erster Linie bei der Klasse `PlayCursor` eine Rolle, die dazu dient bei der Wiedergabe einen (zusätzlichen) Mauszeiger zur Verfügung zu stellen, der entsprechend den aufgezeichneten Daten bewegt wird. Dabei konnten verschiedene Ansätze in Betracht gezogen werden:

#### a) Der „echte“ Mauszeiger wird für die Wiedergabe verwendet

Nachteilig wirkt sich in diesem Fall aus, daß der Benutzer während der Wiedergabe nur noch über die Tastatur eingreifen kann, bspw. um die Wiedergabe anzuhalten oder zu beenden, da ihm der Mauszeiger „aus der Hand“ genommen wird. Darüber hinaus besteht unter Java keine unmittelbare Möglichkeit, den Mauszeiger programmgesteuert zu bewegen. Zu diesem Zweck müßte das Betriebssystem mit Hilfe des Java Native Interface (JNI) angesprochen werden.

#### b) Der Mauszeiger wird in die Wiedergabefenster gezeichnet

Wenn ein simulierter Mauszeiger mit Hilfe der Grafikfunktionen, die das JDK standardmäßig zur Verfügung stellt, bspw. als `Image`, in die Wiedergabefenster gezeichnet werden soll, muß dafür eine transparente Ebene, eine sog. `GlassPane` verwendet werden, die den gesamten Bereich des Fensters abdeckt und über allen Bedienelementen liegt. Dieser Mechanismus steht jedoch nur für Fensterklassen der JFC/Swing-Bibliothek zur Verfügung.

#### c) Simulation eines zweiten Mauszeigers mit Hilfe eines transparenten Fensters

Eine realitätsnahe Simulation eines zweiten Mauszeigers läßt sich durch ein transparentes Fenster realisieren, das ständig im Vordergrund (`topmost`) gehalten wird und in das ein Abbild eines Mauszeigers gezeichnet wird. Die Bewegung des Mauszeigers wird durch das Verschieben des Fensters realisiert. Transparente Fenster sind, im Gegensatz zu transparenten Bedienelementen, weder im AWT noch in der JFC/Swing-Bibliothek vorgesehen. Microsoft Windows bietet jedoch eine gewisse Unterstützung für transparenter Fenster,<sup>36</sup> die in dem hier beschriebenen Sinne genutzt werden kann. Für das JRV-System wurde eine `Dynamic Link Library (DLL)` in C/C++ realisiert, die entsprechende Funktionen für zusätzliche Mauszeiger zur Verfügung stellt und die über das Java Native Interface angesprochen werden kann.

## 4.2 Implementierung der `RecordSource`- und `PlayTargetManager`

Alle wesentlichen Funktionen für die Aufzeichnung und Wiedergabe der Komponenten des AWT und der JFC/Swing-Bibliothek werden durch diejenigen Klassen zur Verfügung gestellt, welche die entsprechenden `RecordSource`- und `PlayTargetManager` implementieren. Im Folgenden werden grundsätzliche Überlegungen, die bei der Realisierung dieser Klassen eine Rolle gespielt haben, dargestellt.

---

<sup>36</sup> Anmerkung: Die Unterstützung für transparente Fenster ist unter Microsoft Windows insofern unvollkommen, als der Hintergrund des Fensters nicht automatisch aktualisiert wird, wenn sich der Untergrund, bspw. aufgrund einer Bewegung des Fensters, ändert. Die Aktualisierung muß daher „von Hand“ angestoßen werden.

#### 4.2.1 Die Klassenhierarchien der RecordSource- und PlayTargetManager

Die AWT- und Swing-Komponenten sind in Form einer Klassenhierarchie realisiert, wie in Abschnitt 2.3.3, Abbildung 20 skizziert. Diese Hierarchie wurde für die Aufzeichnung durch entsprechende RecordSourceManager- und für die Wiedergabe durch PlayTargetManager-Implementierungen nachgebildet. Die Abbildung 24 in Abschnitt 3.4.1 bzw. die Abbildung 29 in Abschnitt 3.5.1 skizzieren den Aufbau dieser Klassenhierarchien.

Auf eine Spezialisierung von RecordSource- und PlayTargetManager-Klassen wurde immer dann verzichtet, wenn mehrere Unterklassen einer AWT- oder Swing-Klasse durch dieselbe Managerklasse bearbeitet werden konnten. Die Klassenhierarchie der AWT- und Swing-Komponenten wird also nicht vollständig nachgebildet. Die zu den jeweiligen AWT- und Swing-Komponenten gehörenden Klassen des JRV-Systems können durch ihre Benennung identifiziert werden, die nach dem folgenden Schema erfolgt:

- <Name der AWT/Swing-Komponente>RecManager
- <Name der AWT/Swing-Komponente>PlayManager

Die Aufzeichnung und Wiedergabe der Eigenschaften von Komponenten und die Registrierung der Listener, die zur Überwachung von Ereignissen eingesetzt werden, sind innerhalb der Managerklassen allgemein auf der gleichen Ebene der Klassenhierarchie angesiedelt, auf der sich die entsprechende AWT- bzw. JFC/Swing-Komponente befindet. So kann bspw. für jede Komponente, die von `java.awt.Component` abgeleitet ist, ein Listener zur Überwachung der Maus installiert werden. Dementsprechend erfolgt die Aufzeichnung und Wiedergabe solcher Ereignisse in den Klassen `ComponentRecManager` und `ComponentPlayManager`.

Wenn zukünftig weitere RecordSourceManager und PlayTargetManager implementiert oder die bestehenden erweitert werden sollen, müssen die entsprechenden Methoden der Superklassen explizit aufgerufen werden, wenn die Grundfunktionalität erhalten bleiben soll. Die zugehörigen Stellen sind im Quellcode gekennzeichnet.

#### 4.2.2 Die Serialisierung von Objekten

Um aufgezeichnete Daten dauerhaft zu sichern und wieder zu lesen, wird innerhalb des JRV-Systems grundsätzlich die Möglichkeit zur Serialisierung und Deserialisierung von Objekten genutzt, die das JDK standardmäßig zur Verfügung stellt. Mit Hilfe des Serialisierungsmechanismus können Objekte von solchen Klassen, welche die Schnittstelle `java.io.Serializable` implementieren, in Ausgabedatenströme geschrieben und aus Eingabedatenströmen gelesen werden.<sup>37</sup> Bei der Anwendung dieses Mechanismus sind jedoch folgende Aspekte zu berücksichtigen:

---

<sup>37</sup> Anmerkung: Detaillierte Informationen zur Serialisierung und Deserialisierung von Objekten enthält die Dokumentation des JDK bei der Beschreibung der Schnittstelle `java.io.Serializable` sowie bei der Beschreibung der Klassen `java.io.ObjectOutputStream` und `java.io.ObjectInputStream`. Viele Klassen der Standardbibliotheken des JDK sind serialisierbar. Eine Liste dieser Klassen kann den Querverweisen in [CH98] entnommen werden. Da die Schnittstelle `java.io.Serializable` keine Methoden enthält, genügt es bei anwendungsspezifischen Klassen den Zusatz „implements `Serializable`“ zur Definition der Klasse hinzuzufügen. Falls eine Klasse besondere Aktionen im Rahmen der Serialisierung durchführen muß, können dazu optionale Methoden zu der Klasse hinzugefügt werden. Die erforderlichen Signaturen sind ebenfalls in der Dokumentation des JDK beschrieben.

- Im Rahmen der Serialisierung eines Objektes wird eine transitive Hülle über alle Objekte gebildet, die durch Mitgliedsvariablen des zur serialisierenden Objektes referenziert werden. Alle so erreichbaren Objekte werden ebenfalls serialisiert. So genügt es bspw., die Wurzel einer baumartigen Datenstruktur zu serialisieren, wenn der gesamte Baum serialisiert werden soll. Demzufolge müssen jedoch alle in die Serialisierung eingeschlossenen Objekte zu serialisierbaren Klassen gehören. Ansonsten wird eine Ausnahme ausgelöst.<sup>38</sup>
- Es werden Objekte serialisiert, nicht Werte. Wenn dasselbe Objekt bspw. zweimal über denselben Ausgabedatenstrom in eine Datei geschrieben wird, so kann das Objekt zwar auch zweimal aus der Datei gelesen werden. Die Mitgliedsvariablen des Objektes weisen jedoch anschließend beides mal die Werte auf, die bei der ersten Serialisierung galten. Es handelt sich tatsächlich um ein einziges Objekt. Sollen zwei Zustände ein und desselben Objektes serialisiert werden, muß dazu entweder eine Kopie des Objektes erzeugt oder eine Klasse implementiert werden, die Objekte für die Aufnahme der zu serialisierenden Werte zur Verfügung stellt.

#### 4.2.3 Ermitteln, Sichern und Wiederherstellen des Zustands von Komponenten

Wenn die Aufzeichnung mit dem Ziel erfolgen soll, daß ein von der Anwendung unabhängiger Runtime-Player das visuelle Erscheinungsbild des Verlaufs einer Anwendungssitzung wiedergibt, so muß das Aufzeichnungssystem in der Lage sein, Informationen über die verwendeten Fenster, Dialogfelder, Bedienelemente etc. in dem Maße zu erfassen, daß bei der Wiedergabe entsprechende Komponenten erstellt werden können. Dabei müssen folgende Problemfälle berücksichtigt werden.

##### a) Das Ermitteln von nicht gesetzten oder transitiv bestimmten Attributen

Komponenten können Attribute besitzen, die, so lange nicht ein anwendungsspezifischer Wert zugewiesen wird, null sind oder durch übergeordnete Komponenten oder standardisierte Vorgaben initialisiert werden. Wie das folgende Beispiel zeigt, muß der ermittelbare Wert eines Attributs dann nicht mit dem tatsächlich verwendeten Wert übereinstimmen.

Die Funktion der Methode `getBackground` der Klasse `java.awt.Component` wird in der API-Dokumentation des JDK 1.1 sowie des JDK 1.2 von Sun wie folgt beschrieben:

```
public Color getBackground()
```

Gets the background color of this component.

Returns:

This component's background color. If this component does not have a background color, the background color of its parent is returned.

Mit Hilfe eines Testprogramms kann gezeigt werden, daß unter Microsoft Windows für eine Schaltfläche, d.h. eine Komponente vom Typ `java.awt.Button`, die sich auf einem Rahmenfenster, d.h. einer Komponente vom Typ `java.awt.Frame`, befindet, ein Wert zurückgeliefert wird, welcher der Hintergrundfarbe des Rahmenfensters (standardmäßig Weiß) entspricht, während die Schaltfläche jedoch grau dargestellt wird. Grau entspricht der Standardfarbe von Schaltflächen unter Windows und damit der Farbe des Peers des Buttons.

---

<sup>38</sup> Anmerkung: Mitgliedsvariablen, die Objekte referenzieren, die nicht serialisiert werden sollen, können durch das Schlüsselwort „transient“ gekennzeichnet werden.

Wird also während der Aufzeichnung mit Hilfe der Methode `getBackground` die Farbe von Schaltflächen ermittelt, so erhalten diese Schaltflächen bei der Wiedergabe u.U. eine falsche Hintergrundfarbe, nämlich die des Fensters, auf dem sich die Schaltfläche befindet. Es besteht keine Möglichkeit, festzustellen, ob die Farbe einer Schaltfläche nicht gesetzt ist und daher das Element mit der Standardfarbe des Peers dargestellt wird, oder ob die Schaltfläche und das übergeordnete Fenster tatsächlich dieselbe Farbe besitzen sollen.

## b) Das Ermitteln von Attributen, die aus komplexen Objekten bestehen

Einige Attribute, die das visuelle Erscheinungsbild von Komponenten bestimmen, bestehen selbst aus komplexen zusammengesetzten Objekten, im Gegensatz bspw. zu dem unter a) beschriebenen Attribut „Hintergrundfarbe“, das durch ein Farbobjekt bestimmt wird, das sich durch eine einfache Ganzzahl, den RGB-Wert der Farbe, beschreiben läßt.

JFC/Swing-Komponenten, die bei Verwendung des JDK 1.1 mit separater Swing-Bibliothek von der Klasse `com.sun.java.swing.JComponent` abgeleitet sind, können bspw. einen Rahmen (Border) besitzen, der mit Hilfe der Methoden `setBorder` und `getBorder` gesetzt bzw. ermittelt werden kann.

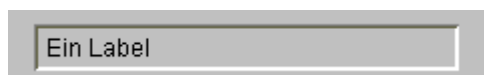


Abbildung 36: Ein Label mit einem Rahmen

In Abbildung 36 wird eine Komponente vom Typ `com.sun.java.swing.JLabel` dargestellt, die mit einem Rahmen versehen wurde. Wenn dieser Rahmen mit der Methode `getBorder` ermittelt wird, so kann das Ergebnis ein Objekt von jeder beliebigen Klasse sein, welche die Schnittstelle `com.sun.java.swing.border.Border` implementiert.

Um einen identischen Rahmen bei der Wiedergabe erzeugen zu können, kann das Aufzeichnungssystem den Typ des Borderobjektes ermitteln, das Objekt analysieren und schließlich die Attribute, die das Erscheinungsbild des Rahmens bestimmen, sichern. Abbildung 37 zeigt die konkrete Zusammensetzung des Rahmens aus Abbildung 36, wobei hier die Operationen dargestellt sind, die nötig sind, um den Rahmen auf elementare Datentypen zurückzuführen, die für die Erzeugung eines entsprechenden Objektes an die Konstruktoren der Unterobjekte übergeben werden könnten.

Der konkrete Typ des Rahmens ist hier `CompoundBorder`, ein zusammengesetzter Rahmen, der aus einem `BevelBorder` besteht, der einen vertieften Reliefrahmen darstellt, sowie einem `EmptyBorder`, der gewährleistet, daß der Text, der in dem `JLabel` dargestellt wird, das Relief nicht berührt. Tiefer verschachtelte Strukturen sind ohne weiteres denkbar.

Wenn der Zustand zusammengesetzter Objekte bis hinab zu den elementaren Datentypen ermittelt werden soll, müssen Klassen implementiert werden, die es ermöglichen, die gewonnenen Daten im Rahmen der Aufzeichnung zu sichern und bei der Wiedergabe identische Objekte zu erzeugen. Solche Klassen müssen bei Bedarf zusammengesetzte und potentiell beliebig tief verschachtelte Strukturen repräsentieren können.

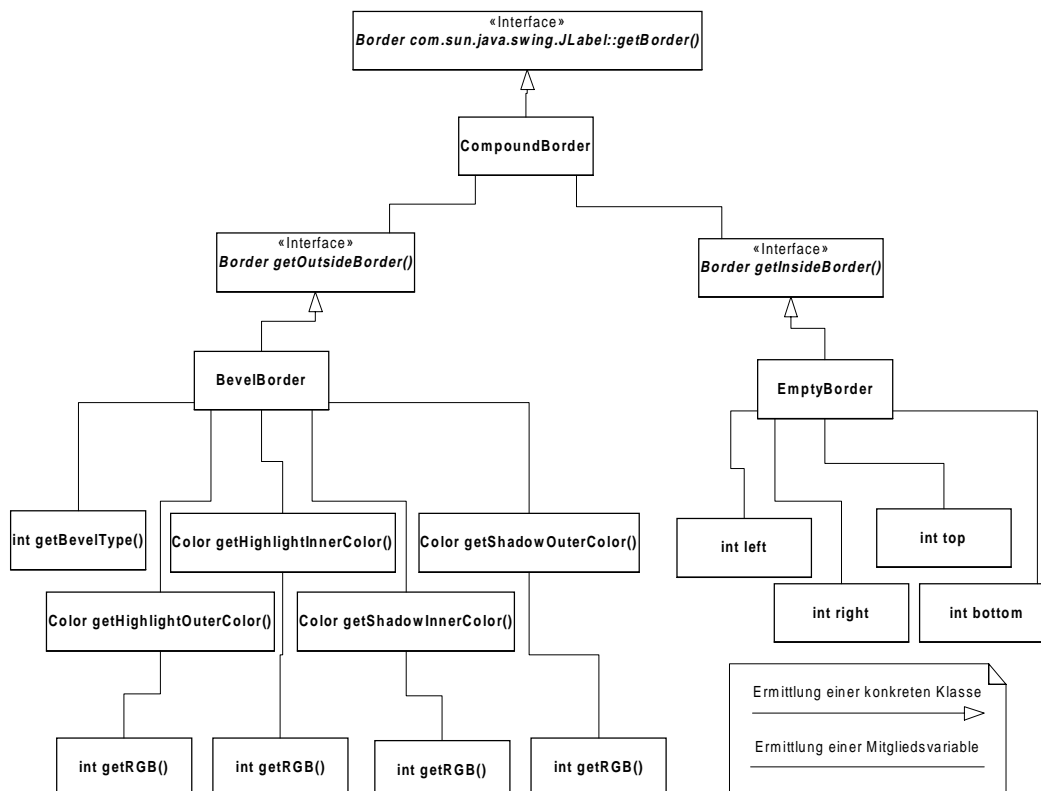


Abbildung 37: Der Aufbau eines Borderobjektes

### c) Direkte Serialisierung von aufzuzeichnenden Objekten

Die Ermittlung der Attribute von Komponenten kann, wie unter a) geschildert, mit Fehlern behaftet oder, wie unter b) gezeigt, verhältnismäßig aufwendig zu implementieren sein. Eine Alternative ist, den Zustand von Objekten nicht vollständig zu ermitteln. Da die hier betrachteten Klassen der AWT- und der JFC/Swing-Bibliothek durchweg serialisierbar sind, können Attribute, wie bspw. der unter b) beschriebene Rahmen, direkt als Objekt gesichert werden. Wenn jedoch einzelne Attribute serialisiert werden, wie bspw. Rahmen, Farbobjekte oder Schriftarten, warum dann nicht das ganze Objekt, die gesamte Schaltfläche, oder besser noch das Fenster, das aufgezeichnet werden soll, mit allen Komponenten darauf, zu Beginn einer Aufzeichnung serialisieren?

Tatsächlich wurden bei der prototypischen Implementierung des JRV-Systems beide Ansätze getestet. Zunächst wurden für einige AWT-Komponenten Klassen realisiert, die z.B. den Zustand einer Schaltfläche vom Typ `java.awt.Button` bis hinab zu den elementaren Datentypen, `int`, `boolean`, `String` etc. ermitteln und wiederherstellen. Dennoch werden in der aktuellen Version des JRV-Systems alle aufzuzeichnenden Komponenten serialisiert, um ihren Anfangszustand zu sichern und eine identische Anordnung von Fenstern und Bedienelementen mit Hilfe des Runtime-Players wieder herstellen zu können.

### d) Serialisierung versus Zustandsermittlung

Wird die Objektserialisierung im großen Rahmen, d.h. für ganze Fenster oder Dialogfelder eingesetzt, so hat dies weitreichende Auswirkungen, sowohl für die Realisierung des Aufzeichnungs- und Wiedergabesystems als auch für die Benutzung in Anwendungsprogrammen. Vor- und Nachteile dieser Methode gegenüber der Ermittlung des Zustands von Komponenten sollen im Folgenden gegeneinander abgewogen werden.

- **Behandlung von privaten untergeordneten Komponenten:** Eine Komponente vom Typ JScrollBar besitzt bei Verwendung des Look and Feel von Windows unter Swing Version 1.0.1 immer zwei untergeordnete Elemente vom Typ BasicArrowButton, die bei der Aufzeichnung berücksichtigt werden müssen, um bspw. die Bewegung der Maus über den Pfeilen der Bildlaufleiste nachvollziehen zu können. Wenn nicht serialisiert, sondern der Zustand der Komponenten ermittelt werden soll, kann dabei zunächst wie bei „normalen“ Komponenten des Programms vorgegangen werden: Das JScrollBar wird als Objekt von der Klasse Container betrachtet. Die untergeordneten Komponenten werden nummeriert und analysiert. Bei der Wiedergabe muß jedoch berücksichtigt werden, daß BasicArrowButtons nicht explizit erzeugt werden, sondern die aufgezeichneten Daten mit denjenigen Elementen verbunden werden, die bei der Erzeugung des übergeordneten Elements, des JScrollBars, automatisch erzeugt werden. Solche Zusammenhänge müssen für jeden Komponententyp einzeln behandelt werden.<sup>39</sup> Werden Fenster dagegen komplett serialisiert, enthalten alle Komponenten automatisch wieder die untergeordneten Elemente, die auch bei der Aufzeichnung vorhanden waren.
- **Robustheit der Routinen zur Zustandsermittlung:** Die im vorhergehenden Abschnitt beschriebenen BasicArrowButtons sind private Elemente der JScrollBar-Klasse in dem Sinne, daß die öffentliche Schnittstelle dieser Klasse nicht offenbart, daß diese Elemente existieren. Es gibt keine Gewähr, daß sie in derselben Form bei jedem Look and Feel vorhanden sind oder daß eine andere Version der Swing-Bibliothek nicht eine andere Form der Implementierung der JScrollBar-Klasse zur Verfügung stellt. Wird der Zustand von Komponenten detailliert ermittelt, so ist das Aufzeichnungs- und Wiedergabesystem in solchen Fällen von Details der Implementierung der AWT- und Swing-Bibliothek abhängig, die durch einen objektorientierten Ansatz eigentlich verborgen werden sollen. Wenn Komponenten komplett serialisiert werden, muß bei einem Versionswechsel der Bibliothek lediglich die Aufzeichnung wiederholt werden, eine Modifikation des Quellcodes des Systems ist in solchen Situationen nicht notwendig.
- **Analyse und Rekonstruktion der Komponentenhierarchie:** Wenn ein Fenster komplett mit den darauf befindlichen Komponenten serialisiert wird, ist eine Wiederherstellung der Hierarchie der Elemente nicht notwendig und die Zuordnung der aufgezeichneten Daten zu den Elementen einfach. Jeder Komponente wird bei der Übergabe zur Aufzeichnung eine ID zugeordnet, die zusammen mit dem Objekt in Form eines Erzeugungsereignisses gesichert wird. Untergeordnete Objekte und ihre Hierarchie werden in diesem Moment automatisch festgehalten. Anschließend werden die Unterkomponenten nummeriert und rekursiv zusammen mit den ihnen zugeordneten IDs gespeichert, wobei hier automatisch nur noch Verweise auf die vorher bereits einmal serialisierten Objekte gesichert werden. Die Wiederherstellung der Objekte und die Zuordnung der IDs bei der Wiedergabe erfolgt einfach durch das Deserialisieren der Erzeugungsereignisse in derselben Reihenfolge. Wird dagegen die Hierarchie der Komponenten explizit analysiert und Daten für deren Erzeugung gespeichert, muß nicht nur zu jeder Komponente gesichert werden, welches das übergeordnete Element ist und in welcher Form die Komponente zu diesem Element hinzugefügt werden muß. Bei Elementen, wie den oben genannten BasicArrowButtons dürfen keine neuen Objekte erzeugt werden. Bei anderen Elementen besteht u.U. Wahlfreiheit, ob ein neues Objekt erzeugt wird, oder ob Objekte, die automatisch mit übergeordneten Elementen erzeugt wurden, direkt für die Wiedergabe verwendet werden.<sup>40</sup>

---

<sup>39</sup> Anmerkung: Die Klasse JScrollBar ist hier noch ein harmloser Fall. Mit Hilfe eines Beispielprogramms kann gezeigt werden, daß eine JScrollPane grundsätzlich 2 JScrollBars, 4 JPanels und 1 JViewport enthält.

<sup>40</sup> Anmerkung: Dies betrifft z.B. JRootPanels, JLayeredPanels oder Komponenten, die als ContentPane in JFrame dienen, und die sowohl automatisch erzeugt werden, als auch explizit gesetzt werden können.

- **Begrenzung der serialisierten Daten:** Ein besonderes Problem bei der Serialisierung ist die Frage, wie die Daten, die serialisiert werden, begrenzt werden können. Wie bereits in Abschnitt 4.2.2 beschrieben, wird grundsätzlich ein Graph von Objekten serialisiert, der durch die Referenzen in den Mitgliedsvariablen der Objekte aufgebaut wird. Genau diese Eigenschaft ist es ja, die bei der Serialisierung eines Fensters mit all seinen Komponenten genutzt werden soll. Problematisch sind in diesem Zusammenhang jedoch Referenzen, die aus der Hierarchie der Komponenten heraus, auf anwendungsspezifische Datentypen zeigen. Als ein herausragendes Beispiel sind Listener zu nennen, die bei unterschiedlichen Bedienelementen registriert werden können, um auf Aktionen des Benutzers zu reagieren. Die Klassen, die diese Listener implementieren, sind anwendungsspezifisch und führen entweder dazu, daß die Serialisierung fehlschlägt, wenn eine dieser Klassen nicht als serialisierbar gekennzeichnet ist, oder daß Daten der Anwendung serialisiert werden, die nicht serialisiert werden sollen oder dürfen. Aus diesem Grund muß für die Serialisierung gefordert werden, daß Listener, ebenso wie andere anwendungsspezifische Klassen, erst dann zu den Bedienelementen hinzugefügt werden, nachdem diese zur Aufzeichnung übergeben wurden.
- **Anwendungsspezifische Klassen:** Wenn der Zustand von Aufzeichnungsobjekten nicht ermittelt, sondern serialisiert wird, stellen anwendungsspezifische Klassen immer dann ein Problem dar, wenn die Aufzeichnung durch einen Runtime-Player wiedergegeben werden soll und dabei die anwendungsspezifischen Klassen, bspw. an einem anderen Ort, nicht zur Verfügung stehen. Dies geht über das oben geschilderte Problem der Listener hinaus. So ist es z.B. nicht unüblich anwendungsspezifische Fenster von der Klasse JFrame abzuleiten, um Fenster mit bestimmten Funktionen zu implementieren. Die Deserialisierung solcher Fenster wird fehlschlagen, wenn die entsprechende Klasse bei Verwendung des Runtime-Players nicht zur Verfügung steht. Während für Listener gefordert werden kann, daß sie den Bedienelementen erst hinzugefügt werden, nachdem diese zur Aufzeichnung übergeben wurden, läßt sich das Problem bei der Serialisierung anwendungsspezifischer Komponentenklassen nicht mit vertretbarem Aufwand vermeiden. (Eine Ansatz wäre z.B. Klassen zu fordern, die Komponenten nicht erweitern, sondern lediglich verwenden.) Wird auf die Serialisierung verzichtet und statt dessen eine Analyse der aufzuzeichnenden Komponenten betrieben, können dagegen anwendungsspezifische Klassen durch die nächstliegende Superklasse, die noch zu den AWT- oder Swing-Komponenten gehört, ersetzt werden. Wo die Anwendung z.B. eine Klasse MyFrame verwendet, die von JFrame abgeleitet ist, könnte das Wiedergabesystem direkt einen JFrame verwenden.
- **Flexibilität:** Ein weiterer Vorteil der vollständigen Analyse von Komponenten gegenüber der Serialisierung ist eine höhere Flexibilität bei der Wiedergabe oder Bearbeitung der analysierten Daten. Eine Bearbeitung der Daten wird bei der Serialisierung weitgehend ausgeschlossen sein, da die Objekte in einer binär kodierten Form vorliegen, die für das Aufzeichnungs- und Wiedergabesystem nicht zugänglich ist und da auf die Analyse der einzelnen Unterkomponenten ja gerade verzichtet wurde. Bei der Wiedergabe ergeben sich Einschränkungen insofern, als die Objekte für die Wiedergabe nicht willkürlich neu erzeugt werden können, ohne den Eingabestrom neu zu lesen, wenn bspw. die Wiedergabe wiederholt werden soll. Einmal deserialisierte Daten, d.h. hier die Komponenten selbst, werden durch die Wiedergabe verändert. Werden Analysedaten verwendet, um die Komponenten zu erzeugen, können diese Daten im Speicher behalten und für die wiederholte Wiedergabe immer wieder verwendet werden.

Nach Abwägung der hier geschilderten Vor- und Nachteile der Serialisierung komplexer Komponenten wurde bei der prototypischen Implementierung des JRV-Systems die Serialisierung gewählt.

#### 4.2.4 Grenzen der Ereignisaufzeichnung durch Listener

Die Aufzeichnung der Benutzerinteraktion sowie von bestimmten Zustandsänderungen, die durch Anwendungsprogramme explizit ausgelöst werden können (bspw. die Änderung der Fenstergröße), erfolgt mit Hilfe der Listener, die konzeptionell in Abschnitt 2.3.3 beschrieben wurden. Listener werden im JRV-System soweit nur irgend möglich eingesetzt, da auf diese Weise Daten ohne weitere Eingriffe in die aufzuzeichnende Anwendung gewonnen werden können. Die Anwendung übergibt eine Komponente zur Aufzeichnung. Das JRV-System installiert die notwendigen Listener und verwaltet automatisch die gewonnenen Daten. Grenzen einer solchermaßen automatisierten Aufzeichnung bestehen jedoch in folgenden Bereichen, insbesondere bei der Aufzeichnung von AWT-Komponenten.

##### a) Untergeordnete Elemente, die keine Komponenten sind

Bei einigen AWT- und JFC/Swing-Komponenten existieren Bereiche oder untergeordnete Bedienelemente, die grundsätzlich durch bestimmte Listener nicht überwacht werden können. Das Aufzeichnungssystem kann über Aktionen in diesen Bereichen zum Teil nur mittelbar, d.h. durch resultierende Zustandsänderungen, informiert werden. Zum Teil ist das Aufzeichnungssystem für bestimmte Aktionen vollständig blind, wie folgende Beispiele zeigen.

Bewegungen der Maus werden bspw. über dem Außenbereich eines Fensters, d.h. über der Titelleiste und dem Rahmen, von einem MouseMotionListener nicht erfaßt. Zwar können Aktionen, die über die dort befindlichen Bedienelemente ausgelöst werden, mittelbar durch andere Listener erfaßt werden, bspw. die Veränderungen der Größe oder der Position durch einen ComponentListener oder das Schließen eines Fensters durch einen WindowListener. Jedoch kann nicht im Detail aufgezeichnet werden, wie der Benutzer die Aktion auslöst.

Weitreichender wirkt sich jedoch das Fehlen des Zugangs zu untergeordneten Elementen bei Komponenten wie dem Listenfeld `java.awt.List` aus, bei dem wesentliche Aspekte der Benutzung des Elements weder unmittelbar, d.h. in Form der Benutzerinteraktion, noch mittelbar in Form des resultierenden Ergebnisses aufgezeichnet werden können. Abbildung 38 zeigt ein solches Listenfeld.

Mit Hilfe eines Testprogramms kann gezeigt werden, daß die Bildlaufleiste am rechten Rand des Listenfelds für keinen Listener zugänglich ist. Aktionen der Maus über der Bildlaufleiste werden weder durch MouseMotionListener noch durch MouseListener registriert. Keiner der Listener, die unter dem JDK 1.1.6 für Klassen vom Typ `java.awt.List` selbst oder für Klassen vom Typ `java.awt.Component` registriert werden können, ist in der Lage, den Bildlauf zu überwachen. Lediglich Änderungen der Selektion von Listeneinträgen oder aber Doppelklicks auf einen Eintrag können mit Hilfe eines ItemListeners bzw. eines ActionListener registriert werden. Weitere Superklassen zu `java.awt.List`, für die Listener registriert werden könnten, existieren nicht.

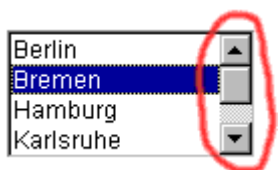


Abbildung 38: Eine `java.awt.List`-Komponente

Darüber hinaus stehen auch keine Methoden in der Klasse `java.awt.List` zur Verfügung, um die gerade sichtbaren Elemente der Liste oder den Index des ersten sichtbaren Elements zu ermitteln. Es ist lediglich möglich, den Index desjenigen Elements zu ermitteln, das zuletzt explizit sichtbar gemacht wurde. Mit diesen Informationen ist es nicht möglich, die Benutzerinteraktion mit der Liste realitätsnah aufzuzeichnen und wiederzugeben. Die Bildlaufleiste, die offensichtlich ein Element des betriebssystemspezifischen Peers der Liste ist, existiert für das Java-Programm aus Sicht des Programmentwicklers praktisch nicht.

## b) Elemente, die außerhalb der Klassenhierarchie von `java.awt.Component` liegen

Ebenfalls unerreichbar für die Aufzeichnung durch nahezu alle Listener sind die Menüs der AWT-Bibliothek. Sämtliche Listener für die Aufzeichnung der Maus, der Tastatur sowie von Veränderungen der Größe und der Position von Bedienelementen können über Methoden der Klasse `java.awt.Component` registriert werden. Komponenten, die Menüs implementieren bilden im AWT eine separate Klassenhierarchie, wie in Abbildung 39 dargestellt.

Demzufolge können Interaktionsschritte mit AWT-Menüs nicht aufgezeichnet werden. Für Menüelemente stehen lediglich `ActionListener` zur Verfügung, mit deren Hilfe ermittelt werden kann, wann eine Option aus einem Menü gewählt wurde.<sup>41</sup>

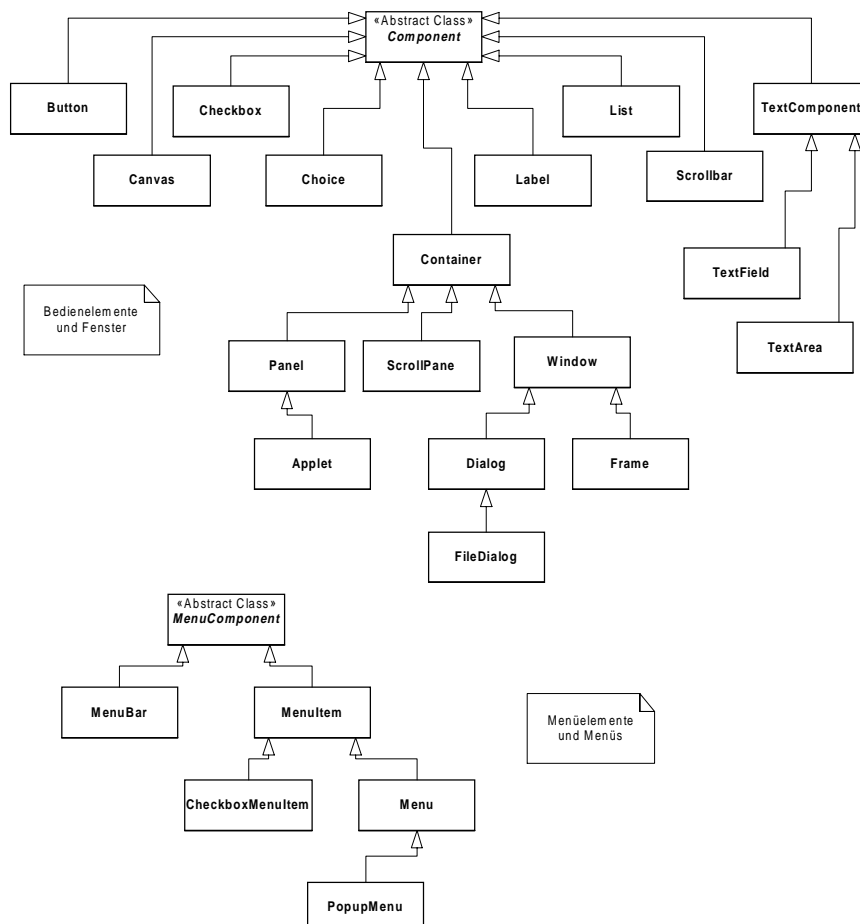


Abbildung 39: Die Klassenhierarchie der AWT-Komponenten

<sup>41</sup> Anmerkung: Für die Menüs der JFC/Swing-Bibliothek existieren die hier genannten Einschränkungen nicht. Die Swing-Menükomponenten sind in die Klassenhierarchie der „normalen“ Komponenten eingeordnet, für die entsprechende Listener registriert werden können.

### c) Zustandsänderungen, für die keine Listener zur Verfügung stehen

Wenn Änderungen bestimmter Eigenschaften von Komponenten, wie bspw. der Vorder- und Hintergrundfarbe oder der Schriftart, aufgezeichnet werden sollen, muß die aufgezeichnete Anwendung entsprechende Ereignisse explizit mitteilen, indem Requests, wie in Abschnitt 3.4.5, Absatz 2 beschrieben, implementiert werden.

Zwar können unter dem JDK 1.1 mit separater JFC/Swing-Bibliothek für Elemente, die von `com.sun.java.swing.JComponent` abgeleitet sind, `PropertyChangeListener` installiert werden, die für die Mitteilung derartiger Ereignisse vorgesehen sind. Allerdings werden hier nur Änderungen von Attributen mitgeteilt, die erst auf dieser Ebene der Klassenhierarchie eingeführt wurden, wie bspw. der Rahmen (`Border`) einer Komponente.<sup>42</sup>

### d) Schlußfolgerungen

Aufgrund der hier geschilderten Einschränkungen wurde der Schwerpunkt bei der prototypischen Implementierung des JRV-Systems auf `RecordSourceManager` und `PlayTargetManager` für JFC/Swing-Komponenten gelegt. AWT-Komponenten wurden hintan gestellt.

## 4.2.5 Low-Level-Ereignisse und High-Level-Ereignisse

In Abschnitt 2.4.2 wurde anhand eines Kontrollkästchens geschildert, daß bei der Betrachtung der Interaktion eines Benutzers mit einem Bedienelement Ereignisse auf unterschiedlichen Ebenen auftreten. Einerseits systemnahe oder sog. Low-Level-Ereignisse, die elementare Interaktionsschritte repräsentieren. Bspw. eine Bewegung des Mauszeigers, die nicht mehr in kleinere Bewegungsschritte zerlegt wird, das Herunterdrücken einer Maustaste, das Loslassen einer Maustaste. Andererseits anwendungsnahe oder sog. High-Level-Ereignisse, die oftmals aus einer bestimmten Folge von Low-Level-Ereignissen resultieren und eine Aktion aus der Sicht der Anwendung beschreiben. Bspw. wird die Aktion „Schaltfläche angeklickt“ ausgelöst, wenn die linke Maustaste über der Schaltfläche gedrückt und wieder losgelassen wurde, wobei zwischen diesen beiden Ereignissen beliebig viele Bewegungen der Maus mit gedrückter Maustaste erfolgen können.

Bei der Wiedergabe zur wiederholten Ausführung muß in jedem Fall berücksichtigt werden, daß bestimmte Low- und High-Level-Ereignisse nicht gemeinsam wiedergegeben werden dürfen. In dem oben genannten Fall würde die Anwendung z.B. doppelt auf das Anklicken der Schaltfläche reagieren, wenn sowohl die Mausereignisse, als auch ein Ereignis „Angeklickt“ wiedergegeben werden würden. Wenn die Wiedergabe des optischen Erscheinungsbildes gefordert wird, kann in bestimmten Situationen abgewägt werden, ob die Wiedergabe der Low- oder der High-Level-Ereignisse dem Anwendungsziel besser dient.

Bei der prototypischen Implementierung des JRV-Systems werden sowohl Low- als auch High-Level-Ereignisse aufgezeichnet. Bei der Wiedergabe kann für einige Bedienelemente durch Parameter zur Laufzeit bestimmt werden, welche Art von Ereignissen bearbeitet wird. Einige Aspekte, die in diesem Zusammenhang berücksichtigt werden müssen, werden im Folgenden kurz beschreiben:

---

<sup>42</sup> Anmerkung: Mit dem JDK 1.2 ist die Möglichkeit, `PropertyChangeListener` zu verwenden von der Klasse `JComponent` in die Klasse `Component` gewandert, welche die Wurzel der Klassenhierarchie der Komponenten bildet. Der Quellcode, der mit dem JDK ausgeliefert wird, zeigt, daß damit auch Änderungen von Attributen wie Vorder-, Hintergrundfarbe und Schriftart mit einem `PropertyChangeListener` registriert werden können.

- **Low-Level-Ereignisse sind „schöner“:** Die Wiedergabe von Low-Level-Ereignissen gewährleistet eine höhere Realitätsnähe. Diese Behauptung wird dadurch begründet, daß grafische Effekte, die eine Rückmeldung über den Fortschritt einer Interaktion liefern, durch Low-Level-Ereignisse ausgelöst werden. Solche Effekte sind bspw. Rahmen, die erhöht oder vertieft dargestellt werden, oder Flächen, die farblich hinterlegt werden, während ein Bedienelement angeklickt wird. Diese grafischen Effekte ermöglichen es dem Benutzer, zu erkennen, ob eine begonnene Aktion abgebrochen oder ausgelöst wird, wenn z.B. die Maustaste losgelassen wird. Eine Schaltfläche, die beim Niederdrücken der Maustaste vertieft dargestellt wird, kann beliebig oft dazu veranlaßt werden, ihren Zustand zwischen der normalen Darstellung (über den Hintergrund erhaben) und der aktivierten Darstellung (in den Hintergrund eingelassen) zu wechseln, indem die Maus mit gedrückter Maustaste aus dem Bereich der Schaltfläche hinaus und wieder hineinbewegt wird. Nur wenn die Maustaste losgelassen wird, während die Schaltfläche vertieft dargestellt ist, wird auch die Aktion „Schaltfläche angeklickt“ ausgelöst. Da solche Effekte bei der Wiedergabe tatsächlich schön anzusehen sind und bessere Rückschlüsse auf das Verhalten des Benutzers bei der Aufzeichnung erlauben, werden sie im JRV-System immer dann wiedergegeben, wenn nicht einer der folgenden Gründe für High-Level-Ereignisse spricht.
- **Auswählen und Umschalten mit Hilfe von High-Level-Ereignissen:** Bei Komponenten, die das Markieren oder eine Auswahl ermöglichen, z.B. Kontrollkästchen (JCheckBox), Optionsfelder (JRadioButton) oder Registerdialoge (JTabbedPane), signalisieren High-Level-Ereignisse Zustandsänderungen. Wenn diese auch bei der Wiedergabe verwendet werden, können auch Änderungen automatisch erfaßt werden, die durch die Anwendung aufgrund einer Verarbeitung und nicht durch den Benutzer ausgelöst werden. Auf die Wiedergabe des einen oder des anderen Ereignistyps muß verzichtet werden, wenn durch die Wiedergabe beider Typen eine Zustandsänderung wieder rückgängig gemacht wird. Mit Hilfe eines Beispielprogramms kann gezeigt werden, daß bspw. bei Objekten vom Typ JCheckBox und JTabbedPane Low- und High-Level-Ereignisse bei der Aufzeichnung in einer solchen Reihenfolge eintreffen, daß nicht ohne aufwendige ereignisübergreifende Analyse entschieden werden kann, ob eine Zustandsänderung mit oder ohne Benutzerinteraktion ausgelöst wurde.<sup>43</sup> Entscheidet man sich gegen High-Level-Ereignisse müssen Änderungen, die durch die Anwendung ausgelöst werden, dem Aufzeichnungssystem explizit mitgeteilt werden. Entscheidet man sich gegen Low-Level-Ereignisse, verliert man die im vorhergehenden Abschnitt beschriebenen grafischen Effekte.
- **Low-Level-Ereignisse mit Folgen:** Bei der Wiedergabe können Low-Level-Ereignisse im Zusammenhang mit bestimmten Komponenten Folgen nach sich ziehen, die nicht mehr vollständig unter der Kontrolle des Wiedergabesystems stehen und die sich mit anderen aufgezeichneten Ereignissen überschneiden. Wird bspw. ein Auswahlfeld (JComboBox) angeklickt, so erscheint ein Popup-Menü, das eine Liste der wählbaren Einträge enthält. Dieses Popup-Menü wird aufgezeichnet und bei der Wiedergabe automatisch dargestellt. Werden jedoch gleichzeitig Low-Level-Ereignisse an das Auswahlfeld geschickt, wird dieses bei der Wiedergabe wieder ein eigenes Popup-Menü erstellen. Die Wiedergabe der Benutzerinteraktion erfolgt jedoch über das Popup-Menü, welches das Wiedergabesystem explizit erstellt hat. Derartige Effekte können durch das Auslassen von Low-Level-Ereignissen am einfachsten vermieden werden.

---

<sup>43</sup> Anmerkung: Unter Swing 1.0.1 mit Windows Look and Feel wird bei Komponenten vom Typ JTabbedPane der Wechsel der angezeigten Seite durch einen ChangeEvent signalisiert, bevor die zugehörigen MouseEvents mitgeteilt werden. Bei JCheckBox-Komponenten trifft der ItemEvent, der die Änderung der Selektion anzeigt, zwischen den MouseEvents ein. Diese Reihenfolge könnte bei anderen Versionen der Bibliotheken oder unter einem anderen Look and Feel unterschiedlich sein.

## 5. Das JRV-System im Einsatz

Um das JRV-System zu testen, seine Eignung für den realen Einsatz zu überprüfen und mögliche Defizite identifizieren zu können, wurden mehrere Beispielprogramme implementiert. Die in diesem Kapitel beschriebenen Beispiele decken verschiedene Funktionsbereiche des Systems ab und vermitteln in Form der ausführbaren Programme einen Eindruck aus der Sicht des Benutzers und in Form der dokumentierten Quellcodes eine konkrete Anleitung für den Anwendungsentwickler, der das System in andere Programme integrieren möchte.

### 5.1 Komponententests

Drei Beispielprogramme mit den Bezeichnungen SwingTest, MDITest und AWTTest dienen dazu, Aufzeichnungen von AWT- und Swing-Komponenten zu erzeugen, die anschließend mit dem Runtime-Player (siehe Abschnitt 5.2) wiedergegeben werden können. Das aufgezeichnete Anwendungsprogramm wird dabei für die Wiedergabe nicht benötigt. Im Folgenden wird kurz skizziert, wie diese Programme ausgeführt werden können, welche Komponenten und Operationen dadurch abgedeckt werden und welche Schritte nötig waren, um das JRV-System auf der Ebene des Quellcodes in die Testprogramme zu integrieren.

#### 5.1.1 Ausführen der Komponententests

Alle drei Komponententestprogramme sind identisch aufgebaut. Nach dem Start der jeweiligen ausführbaren Klasse: SwingTest, MDITest oder AWTTest erscheint das in Abbildung 40 dargestellte Dialogfeld.

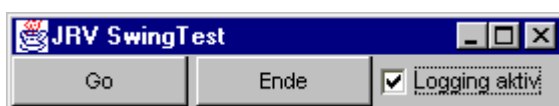


Abbildung 40: Das Dialogfeld zum Start der Komponententests

Nachdem die Schaltfläche „Go“ gewählt wurde, wird ein Dateiauswahldialog geöffnet, in dem der Dateiname angegeben werden kann, unter dem die Aufzeichnung gesichert werden soll. Darüber hinaus wird das aufzuzeichnende Fenster und das in Abbildung 41 dargestellte Dialogfeld zur Steuerung der Aufzeichnung geöffnet. Unmittelbar im Anschluß daran beginnt die Aufzeichnung der Operationen, die mit dem aufzuzeichnenden Fenster durchgeführt werden können.



Abbildung 41: Das Record-Panel-Dialogfeld zur Steuerung des Aufzeichnungsprozesses

Abbildung 41 zeigt das Dialogfeld „Record-Panel“, das durch das JRV-System standardmäßig zur Steuerung des Aufzeichnungsprozesses durch den Benutzer zur Verfügung gestellt wird. Über die Schaltflächen „Pause“, „Aufzeichnen“ und „Ende“ kann die Aufzeichnung angehalten, fortgesetzt und beendet werden. Mit Hilfe des Kontrollkästchens „Logging aktiv“ wird die Erstellung eines textuellen Logs der aufgezeichneten Ereignisse an- und ausgeschaltet. Die Schaltflächen „Log anzeigen“ und „Log schließen“ dienen zum Anzeigen bzw. Ausblenden des Fensters, in dem dieses Log angezeigt wird.

### 5.1.2 SwingTest

Nachdem das SwingTest-Beispiel, wie in Abschnitt 5.1.1 beschrieben gestartet wurde, wird ein Rahmenfenster geöffnet, auf dem sich ein Registerdialog befindet. Die Seiten dieses Registerdialogs enthalten die Bedienelemente der Swing-Bibliothek, die durch das JRV-System unterstützt werden. Abbildung 42 zeigt die 4 Seiten dieses Registerdialogs.

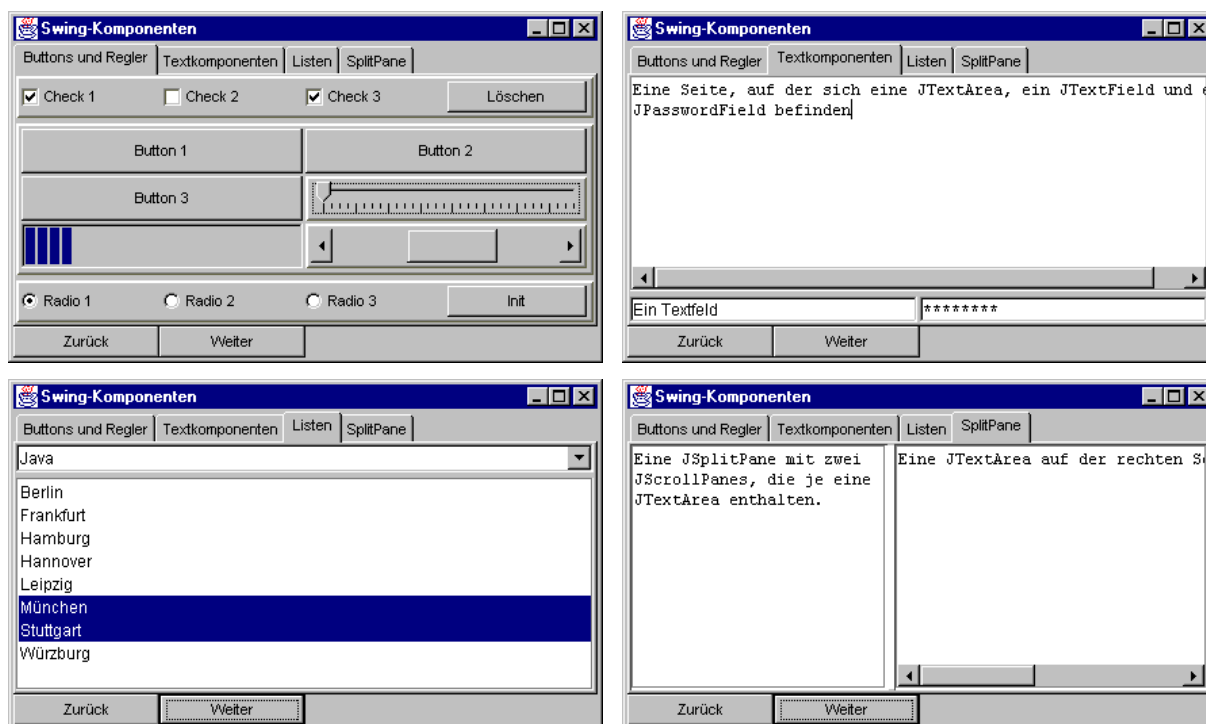


Abbildung 42: Die Dialogseiten der SwingTest-Anwendung

Die Aufzeichnung des SwingTest-Programms durch das JRV-System umfaßt sowohl Operationen, die das gesamte Rahmenfenster betreffen, wie bspw. das Verschieben des Fensters auf dem Bildschirm oder die Veränderung der Fenstergröße mit der Maus, als auch Operationen, die mit Hilfe der Maus oder über die Tastatur mit den Bedienelementen auf den verschiedenen Seiten des Registerdialog durchgeführt werden.

Die Komponenten auf den Seiten „Textkomponenten“, „Listen“ und „SplitPane“ können entsprechend ihrer „natürlichen“ Funktion mit der Maus oder der Tastatur benutzt werden. Die erste Dialogseite, „Buttons und Regler“, enthält einige Bedienelemente, mit deren Hilfe eine Verarbeitung durch die Anwendung simuliert werden kann, die sich in den Elementen der Benutzungsschnittstelle niederschlägt und deren visuelle Effekte auch durch das JRV-System aufgezeichnet und wiedergegeben werden können. Dabei handelt es sich im einzelnen um folgende Funktionen:

- Die Animation der Fortschrittsleiste (JProgressBar), die eine fortlaufende Verarbeitung simuliert und in der SwingTest-Anwendung durch einen Timer gesteuert wird, wird durch das JRV-System automatisch registriert und bei der Wiedergabe wiederhergestellt.
- Über die Schaltflächen „Löschen“ und „Init“ wird die Markierung der Kontrollkästchen am oberen Rand der Dialogseite gelöscht, bzw. das Optionsfeld „Radio 1“ aktiviert. Diese Funktionen, die beispielhaft für eine programm- und nicht benutzergesteuerte Änderung des Zustands von Bedienelementen stehen, werden durch das JRV-System in Form von High-Level-Ereignissen registriert und bei der Wiedergabe entsprechend nachvollzogen.
- Mit Hilfe des Schiebereglers (JSlider) kann die Textfarbe der Schaltflächen „Button 1“, „Button 2“ und „Button 3“ kontinuierlich von schwarz nach weiß verändert werden. Dieser visuelle Effekt kann jedoch im Gegensatz zu Maus- und Tastatureingaben und zu den zuvor geschilderten Zustandsänderungen von Bedienelementen durch das JRV-System nicht automatisch registriert werden, da für Änderungen der Vordergrundfarbe unter Swing 1.0.1 keine Listener zur Verfügung stehen. Das SwingTest-Programm löst daher explizit Ereignisse in Form von Requests aus, um diesen Effekt aufzuzeichnen.

### 5.1.3 MDITest

Die Beispielanwendung MDITest, die ebenfalls wie unter 5.1.1 geschildert gestartet werden kann, simuliert eine Anwendung, die mit mehreren Dokumentenfenstern in einem Rahmenfenster arbeitet, und damit ein sog. „Multi Document Interface“ (MDI) zur Verfügung stellt.

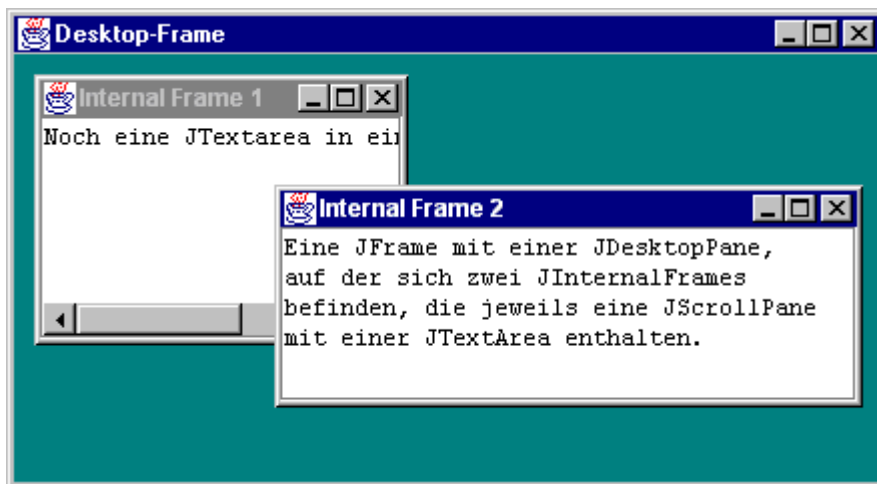


Abbildung 43: Das Desktop-Rahmenfenster der MDITest-Anwendung

Abbildung 43 zeigt das Desktop-Rahmenfenster (JDesktopPane) mit zwei internen Fenstern (JInternalFrame) die jeweils einen Textbereich enthalten. Durch das JRV-System werden sowohl Operationen wie das Verschieben oder Größenänderungen aufgezeichnet, die auf das Rahmenfenster oder die Dokumentenfenster angewandt werden, als auch Texteingaben.

### 5.1.4 AWTTest

Das dritte Komponententestprogramm, AWTTest, entspricht bezüglich Ausführung und Handhabung den beiden zuvor beschriebenen, verwendet jedoch keine JFC/Swing-Elemente sondern lediglich ein Rahmenfenster (Frame) und Komponenten der AWT-Bibliothek.

Das in Abbildung 44 dargestellte Fenster enthält eine Schaltfläche vom Typ Button, ein Listenfeld vom Typ List und ein Texteingabefeld vom Typ TextField. Bei diesen Elementen handelt es sich um die einzigen AWT-Komponenten, die in der vorliegenden Version des JRV-Systems implementiert wurden. Aus den in Abschnitt 4.2.4 dargelegten Gründen, wurde der Schwerpunkt auf die Unterstützung von JFC/Swing-Komponenten gelegt.

Die Aufzeichnung erstreckt sich auch bei diesem Beispielprogramm wieder sowohl auf die Manipulation des gesamten Fensters, als auch auf die Interaktion mit den einzelnen Bedienelementen.

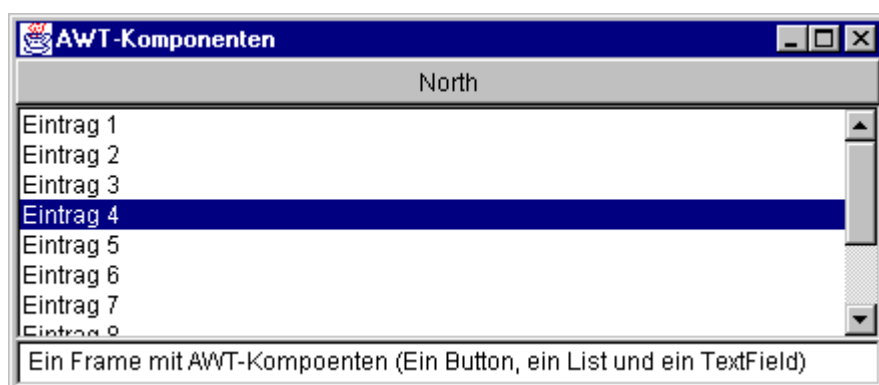


Abbildung 44: Das AWTTest-Fenster

### 5.1.5 Der Aufbau der Testprogramme

Alle drei zuvor besprochenen Testprogramme sind identisch aufgebaut und es sind jeweils nur einige Programmzeilen notwendig, um die hier dargestellten Testfenster aufzeichnenbar zu machen. Die entsprechenden Anweisungen sind im dokumentierten Quellcode der Programme im Detail beschrieben. Im wesentlichen folgt der Ablauf folgendem Schema:

- Das Anwendungsprogramm erzeugt ein Objekt vom Typ `JRVRecorder.RecordController`.
- Das Fenster, das aufgezeichnet werden soll, wird komplett, mit allen darauf befindlichen Bedienelementen zusammengebaut.
- Das Anwendungsprogramm öffnet über die Methode `RecordController.newTrack` einen Track für die Aufzeichnung.
- Die Aufzeichnung wird durch die Methode `RecordController.startRecording` gestartet und das aufzuzeichnende Fenster an die Methode `RecordController.addObject` übergeben.
- Das Anwendungsprogramm fügt ggf. notwendige Listener zu den aufgezeichneten Bedienelementen hinzu, um selbst auf Benutzerinteraktionen reagieren zu können.<sup>44</sup>

<sup>44</sup> Anmerkung: In 4.2.2 und 4.2.3 wurde begründet, daß Listener zu den aufgezeichneten Bedienelementen erst hinzugefügt werden dürfen, nachdem die Objekte zur Aufzeichnung übergeben wurden, um zu verhindern, daß anwendungsspezifische Klassen serialisiert werden. Die Reihenfolge muß hier also eingehalten werden.

- Die Anwendung öffnet über die Methode `RecordController.openRecordPanel` ein Dialogfeld, über das der Benutzer den Aufzeichnungsprozeß anhalten, fortsetzen oder beenden kann.
- Die Anwendung implementiert das Interface `JRVRecorder.RecordObserver` und gibt dort in der Methode `endingRecording`, die aufgerufen wird, wenn der Benutzer die Aufzeichnung beendet hat, die Ressourcen wieder frei, schließt den Track und das Record-Panel.

## 5.2 Der Runtime-Player

Die Aufzeichnung der in Abschnitt 5.1 beschriebenen Testanwendungen erfolgt mit dem Ziel, das visuelle Erscheinungsbild der Anwendungsprogramme und der Benutzerinteraktion wiedergeben zu können.

Für die Wiedergabe solcher Aufzeichnungen stellt das JRV-System die ausführbare Klasse `JRVPlayer.RuntimePlayer` zur Verfügung, die immer dann genutzt werden kann, wenn die Aufzeichnung keine anwendungsspezifischen Klassen oder Komponenten enthält, die am Ort der Wiedergabe nicht zur Verfügung stehen.

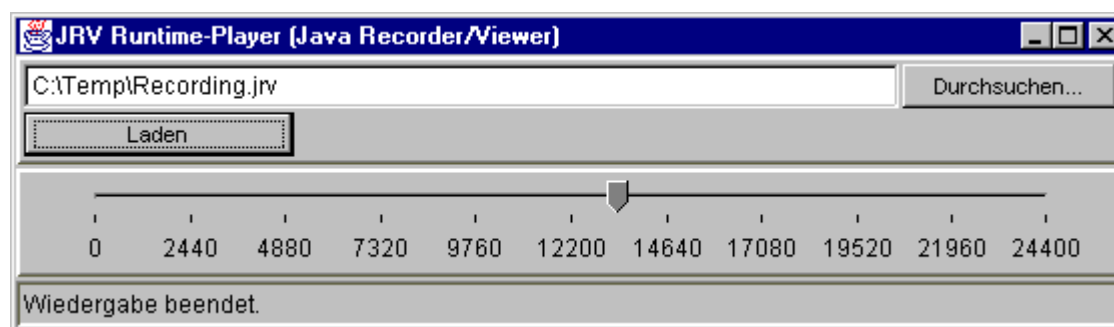


Abbildung 45: Das Dialogfeld des Runtime-Players

Nach dem Start der Klasse `RuntimePlayer` wird zunächst das in Abbildung 45 dargestellte Dialogfeld geöffnet. In dem Textfeld am oberen Rand des Fensters muß der Dateiname der zu ladenden Aufzeichnung angegeben werden. Über die Schaltfläche „Durchsuchen...“ kann ein Dateiauswahldialog geöffnet werden, um die Verzeichnishierarchie nach der gewünschten Datei zu durchsuchen. Durch die Wahl der Schaltfläche „Laden“ wird die angegebene Datei gelesen.

Sobald die Aufzeichnung vollständig geladen ist, wird unterhalb des Schiebereglers in der Mitte des Dialogfeldes eine Skala eingeblendet, welche die Länge der Aufzeichnung in ms angibt. Mit Hilfe des Schiebereglers kann jetzt der Zeitpunkt, ab dem die Wiedergabe in Echtzeit erfolgen soll, eingestellt werden.<sup>45</sup> Darüber hinaus wird das Dialogfeld zur Steuerung der Wiedergabe, das Play-Panel-Dialogfeld geöffnet, das in Abbildung 46 dargestellt ist.

<sup>45</sup> Anmerkung: Die aktuelle Implementierung des JRV-Systems erlaubt lediglich die sequentielle Wiedergabe von Aufzeichnungen und arbeitet in dem Sinne zustandslos, daß nicht irgendein beliebiger Punkt innerhalb der Aufzeichnung angesprungen werden kann, um die Wiedergabe an dieser Stelle aufzunehmen oder fortzusetzen. Der Start der Wiedergabe ab einem bestimmten Zeitpunkt erfolgt dementsprechend dadurch, daß alle vor diesem Zeitpunkt liegenden Ereignisse zeitlos, d.h. so schnell als möglich, wiedergegeben werden, um den Zustand der Wiedergabeobjekte zu dem geforderten Zeitpunkt herzustellen. Anschließend wird die Wiedergabe von diesem Punkt an in Echtzeit fortgesetzt. Diese Einschränkung sowie andere Erweiterungsmöglichkeiten des JRV-Systems werden in Kapitel 6 skizziert.



Abbildung 46: Das Play-Panel-Dialogfeld zur Steuerung des Wiedergabeprozesses

Über die Schaltflächen „Abspielen“, „Pause“ und „Ende“ kann die Wiedergabe gestartet bzw. fortgesetzt, angehalten und beendet werden. Mit Hilfe des Kontrollkästchens „Logging aktiv“ wird die Erstellung eines textuellen Logs der wiedergegebenen Ereignisse an- oder ausgeschaltet. Die Schaltflächen „Log anzeigen“ und „Log schließen“ dienen zum Anzeigen bzw. Ausblenden des Fensters, in dem dieses Log angezeigt wird.

Sobald die Aufzeichnung vollständig wiedergegeben oder durch den Benutzer beendet wurde, wird das Play-Panel-Dialogfeld geschlossen. Wiedergabeobjekte, Fenster und Dialogfelder, die bei der Aufzeichnung nicht geschlossen wurden, bleiben auch am Ende der Wiedergabe erhalten, können also anschließend noch wie „normale“ Fenster betrachtet und untersucht werden, bis eine neue Aufzeichnung geladen wird.<sup>46</sup>

### 5.3 Die DrawLine-Beispielanwendung

Im Gegensatz zu den in Kapitel 5.1 beschriebenen Testprogrammen steht bei der DrawLine-Beispielanwendung nicht die Aufzeichnung mit dem Ziel der visuellen Wiedergabe sondern die wiederholte Ausführung von Aktionen im Vordergrund.

Die ausführbare Klasse DrawLine stellt ein sehr einfaches Malprogramm dar (es können nur Linienzüge in roter Farbe gezeichnet werden), das demonstriert, wie das JRV-System dazu verwendet werden kann, makroähnliche Funktionen in Anwendungen zu integrieren. Dabei werden folgende Techniken angewandt:

- Das Anwendungsprogramm wird nicht nur für die Aufzeichnung, sondern auch für die Wiedergabe verwendet, die einer nochmaligen Ausführung der aufgezeichneten Aktionen gleichkommt. (Die Wiedergabe durch den Runtime-Player des JRV-Systems liefert demzufolge keine sinnvollen Ergebnisse.)
- Durch die Implementierung der RecordObserver, der PlayObserver und der PlaySupplier-Schnittstellen des JRV-Systems kontrolliert das Anwendungsprogramm die Aufzeichnung und Wiedergabe detailliert auf der Ereignisebene. Es werden nur solche Ereignisse aufgezeichnet, die notwendig sind, um die Aktionen des Benutzers semantisch wiederholen zu können, d.h. um dasselbe anwendungsspezifische Resultat zu erzielen.
- Die Benutzungsschnittstelle, über die Aufzeichnung und Wiedergabe gesteuert werden, ist in das Anwendungsprogramm integriert. Die Dialogfelder, die das JRV-System standardmäßig zur Verfügung stellt, werden nicht verwendet.

Nachdem die Klasse DrawLine gestartet wurde, wird das in Abbildung 47 dargestellte Rahmenfenster geöffnet. Den oberen Bereich des Fensters nimmt ein zunächst leerer Bereich ein, in dem durch das Anklicken mit der Maus ein Linienzug gezeichnet werden kann. Die Wahl der Schaltfläche „Löschen“ entfernt den gezeichneten Linienzug.

<sup>46</sup> Anmerkung: Wegen der Serialisierung und Deserialisierung kompletter Fenster, die in Abschnitt 4.2.3 diskutiert wurde, muß die Aufzeichnung neu geladen werden, wenn die Wiedergabe wiederholt werden soll.

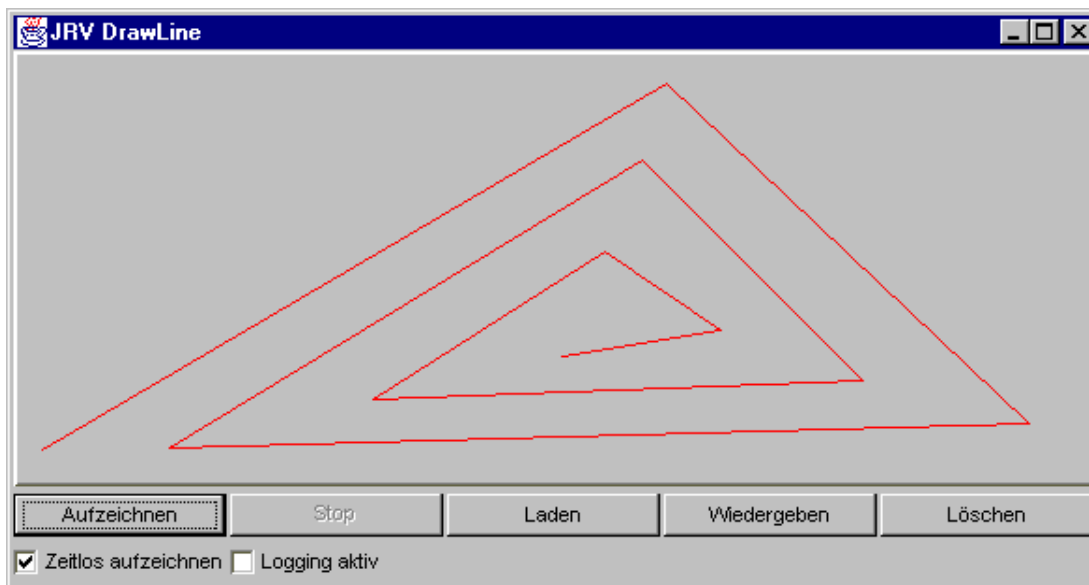


Abbildung 47: Die DrawLine-Beispielanwendung

Wird die Schaltfläche „Aufzeichnen“ gewählt, so werden die anschließend gesetzten Punkte aufgezeichnet, wobei zwei unterschiedliche Modi verwendet werden können, die über das Kontrollkästchen „Zeitlos aufzeichnen“ bestimmt werden:

- **Zeitlose Aufzeichnung:** Wenn das Kontrollkästchen „Zeitlos aufzeichnen“ markiert ist, werden ausschließlich Mausklick-Ereignisse im Zeichenbereich aufgezeichnet. Bei der Wiedergabe einer Aufzeichnung werden diese Ereignisse so schnell als möglich abgespielt. Der aufgezeichnete Linienzug erscheint also praktisch sofort.
- **Aufzeichnung in Echtzeit:** Ist das Kontrollkästchen „Zeitlos aufzeichnen“ nicht markiert, so werden nicht nur Mausklicks, sondern auch Bewegungen der Maus über dem Zeichenbereich in Echtzeit aufgezeichnet. Bei der Wiedergabe können diese Operationen und das Setzen der Punkte dann auch optisch nachvollzogen werden.

Das Filtern der aufzuzeichnenden Ereignisse übernimmt in Abhängigkeit von dem gewählten Aufzeichnungsmodus ein anwendungsspezifischer RecordObserver. Zeitlose Aufzeichnung und Aufzeichnung in Echtzeit sind Mechanismen, die der RecordController des JRV-Systems standardmäßig zur Verfügung stellt.

Um die Aufzeichnung zu beenden, wird die Schaltfläche „Stop“ gewählt. Im Anschluß daran erscheint ein Dateiauswahldialog, über den ein Dateiname angegeben werden kann, unter dem die Aufzeichnung gesichert wird. Das Laden einer Aufzeichnung erfolgt über die gleichnamige Schaltfläche, ebenfalls mit Hilfe eines Dateiauswahldialogs.

Durch die Wahl der Schaltfläche „Wiedergeben“ wird die Aufzeichnung, die sich aktuell im Speicher befindet abgespielt und der entsprechende Linienzug neu erzeugt. Ein durch die DrawLine-Anwendung implementierter PlaySupplier übernimmt dabei die Aufgabe, das vorhandene Wiedergabeobjekt, den Zeichenbereich, mit der Wiedergabe zu verbinden.

Da im Gegensatz zu den in Abschnitt 5.1 beschriebenen Testprogrammen der als Wiedergabeobjekt fungierende Zeichenbereich (es handelt sich hier um eine anwendungsspezifische Klasse, die von JPanel abgeleitet wurde) durch die Anwendung zur Verfügung gestellt wird, erfolgt bei der Aufzeichnung keine Serialisierung, bei der Wiedergabe keine Deserialisierung des Objektes. Die Aufzeichnung wird nur wahlweise in eine Datei geschrieben, kann jedoch auch während der ganzen Zeit im Speicher gehalten und mehrmals wiedergegeben werden, ohne daß ein (erneutes) Laden erforderlich ist. Der Quellcode der DrawLine-Anwendung dokumentiert und erklärt die hier beschriebenen Konzepte im Detail.

## 6. Erweiterungen, Kritik und Fazit

Was kann man sonst noch machen? Was hätte man anders machen können? Und wie ist es gelaufen? Umgangssprachlich formuliert sind dies die Fragen, die in diesem Kapitel kurz angerissen werden sollen.

### 6.1 Ansatzpunkte für Erweiterungen des JRV-Systems

Erweiterungen des JRV-Systems könnten an verschiedenen Punkten ansetzen. Die Frage, warum diese oder jene Funktion nicht in der aktuellen Implementierung realisiert ist, ließe sich allgemein mit dem Hinweis auf die naturgemäß begrenzte Zeit, die zur Implementierung zur Verfügung stand, beantworten.<sup>47</sup> Dennoch sollen im Folgenden einige solche Ansatzpunkte skizziert und begründet werden, warum die Realisierung hintan gestellt wurde.

#### 6.1.1 Unterstützung weiterer Bedienelemente

Die in Kapitel 4 und 5 vorgestellte Implementierung des JRV-Systems unterstützt nicht alle Standardelemente der AWT- und der JFC/Swing-Bibliothek. Eine Erweiterung des Systems könnte RecordSource- und PlayTargetManager für weitere Bedienelemente implementieren. Die „fehlenden“ Komponenten können in folgende Gruppen unterteilt werden:

- **Menüs:** In Kapitel 4.2.5 wurde im Abschnitt „Low-Level-Ereignisse mit Folgen“ kurz skizziert; daß bei der Aufzeichnung einer JComboBox eine detaillierte Unterscheidung zwischen Low- und High-Level-Ereignissen notwendig ist. Dasselbe trifft für Menüs zu. Hier muß z.T. durch Ausprobieren ermittelt werden, welche Ereignisse wie wiedergegeben werden sollten, um ein befriedigendes Ergebnis zu erzielen.
- **Swing-Komponenten mit komplexen „Modellen“:** JFC/Swing-Komponenten sind nach einem Konzept realisiert, das als „Model-View-Controller-Architektur“ bezeichnet wird. Dabei wird die Funktionalität einer Komponente durch das Zusammenspiel dreier Objekte realisiert: Das Model übernimmt die Speicherung der Daten, die in einem Bedienelement enthalten sind, bspw. die Einträge einer Liste. Der Controller definiert das Verhalten des Bedienelements. Wenn von JFC/Swing-Klassen wie JList oder JSlider die Rede ist, handelt es sich dabei im engeren Sinne nur um den Controller. Die View implementiert die Darstellung des Bedienelements am Bildschirm. ([WA98] liefert auf S.8ff. eine detaillierte Beschreibung des Konzepts.) Komponenten, bei denen ein erheblicher Aufwand für die Berücksichtigung des Modells bei der Aufzeichnung und der Wiedergabe getrieben werden muß oder bei denen die Wahrscheinlichkeit groß ist, daß „reale“ Anwendungen ein spezifisches Model implementieren, werden in der aktuellen Version des JRV-Systems nicht berücksichtigt. Im Wesentlichen handelt es sich dabei um die Klassen JEditorPane, JTextPane, JTable und JTree.

---

<sup>47</sup> Anmerkung: Es handelt sich hierbei gewissermaßen tatsächlich um naturgegebene Beschränkungen, die sich aus der dem Typus dieser Arbeit, einer Diplomarbeit ergeben. Es handelt sich um ein 1 Mann Projekt. Es stehen 6 Monate zur Verfügung und die Qualitätsanforderungen erlauben keine Vorgehensweise, bei der lediglich möglichst schnell möglichst viel Code produziert wird.

- **AWT-Komponenten:** In Kapitel 4.2.4 wurde beschrieben, daß insbesondere bei AWT-Komponenten bestimmte Ereignisse nicht so durch Listener ermittelt werden können, wie es für eine realitätsnahe Wiedergabe notwendig wäre. Swing-Komponenten verhalten sich hier in der Regel kooperativer. Aus diesem Grund wurden außer den in Kapitel 5.1.4 genannten AWT-Komponenten keine weiteren implementiert.

Die Unterstützung weiterer Bedienelemente kann durch die Implementierung neuer bzw. die Erweiterung bestehender RecordSource- und PlayTargetManager erfolgen. Der konsequent objektorientierte Aufbau der bestehenden Klassen sollte dies unterstützen und hat sich bereits bei der Realisierung der aktuellen Implementierung bewährt, die sehr gut sukzessive erfolgen konnte, ohne daß an vielen Baustellen gleichzeitig gearbeitet werden mußte.

### 6.1.2 Zustandsermittlung statt Objektserialisierung

In Kapitel 4.2.3 wurde bereits ausführlich diskutiert, daß neben der Serialisierung kompletter Fenster zur Sicherung des Anfangszustands der aufzuzeichnenden Objekte auch eine detaillierte Analyse des Aufbaus der Objekte erfolgen könnte. Eine Erweiterung des JRV-Systems könnte darin bestehen, daß RecordSource- und PlayTargetManager implementiert werden, die diese Analyse vornehmen. Zu Beginn der Implementierung des JRV-Systems wurde auch tatsächlich nach dieser Methode vorgegangen. Nach Ansicht des Autors dieser Arbeit hätte diese Vorgehensweise jedoch in der vorgegebenen Zeit nicht zu einem befriedigenden Ergebnis geführt und letztendlich die in Kapitel 4.2.3 beschriebenen Nachteile der Serialisierung gegen die Gefahr eingetauscht, entweder ein System mit anderen Beschränkungen oder sogar kein funktionierendes System zu haben.

### 6.1.3 Zusätzliche Funktionalität bei der Wiedergabe und ein Editor

Das JRV-System erlaubt aktuell nur die Wiedergabe in eine Richtung, vorwärts entlang der Zeitachse. Die Wiedergabe kann dabei zwar beliebig angehalten und fortgesetzt werden, es ist jedoch nicht möglich, zu einem vorherigen Zeitpunkt zurückzukehren. Eine Simulation der Wiedergabe ab einem beliebigen Punkt ist nur dadurch möglich, daß die Aufzeichnung so schnell als möglich bis zu dem betreffenden Punkt abgespielt wird. Um flexiblere Formen der Wiedergabe zu ermöglichen, könnten folgende Funktionen realisiert werden:

- **Rückwärtswiedergabe:** In die PlayTargetManager könnten Funktionen integriert werden, die es erlauben, den Effekt von Ereignissen rückgängig zu machen. In einigen Fällen wird es dabei jedoch erforderlich sein, daß der Zustand der Objekte in abstrakter Form ständig mitgeführt und aktualisiert wird. Entweder bei der Aufzeichnung, um Undo-Information mit den Ereignissen zu sichern, oder bei der Wiedergabe, um sie adhoc zu generieren. Wird bspw. bei der Aufzeichnung ein Ereignis registriert, das eine Änderung der Fenstergröße mitteilt, ist die alte Größe bereits nicht mehr ermittelbar. Die Fenstergröße müßte also ständig mitgeführt werden.
- **Wiedergabe ab einem beliebigen Punkt:** Wenn die Wiedergabe ab einem beliebigen Zeitpunkt gestartet werden soll, ohne alle vorhergehenden Ereignisse bis zu diesem Punkt abzuspielen, muß (wenigstens in regelmäßigen Intervallen) Information in den aufgezeichneten Daten enthalten sein, die es ermöglicht, die Wiedergabeobjekte in den Zustand zu versetzen, der zu diesem Zeitpunkt gültig ist.

Eine andere Erweiterungsmöglichkeit des JRV-Systems wäre ein Editor, der entweder für die Analyse der aufgezeichneten Daten genutzt werden könnte, mehr als dies mit den textuellen Logs möglich ist, die mit dem aktuellen System erzeugt werden können, oder aber sogar das Bearbeiten von Aufzeichnungen erlauben könnte.

Inwieweit eine Bearbeitung möglich oder sinnvoll ist, hängt dabei natürlich von dem Anwendungsbereich ab, für den die Aufzeichnung konkret erfolgte. Bei Aufzeichnungen, die für die erneute Ausführung vorgenommen werden, sind die Bearbeitungsmöglichkeiten in der Regel anwendungsspezifisch. Bei Aufzeichnungen zur visuellen Wiedergabe, bspw. für Präsentationen, könnte allgemein ein nachträgliches Editieren der Mausbewegungen vorgesehen werden, um Aktionen, die bei der Aufzeichnung stockend durchgeführt wurden, durch ansprechendere Kurven zu ersetzen.

## 6.2 Kritische Anmerkungen

Unter der Überschrift „Kritische Anmerkungen“ sollen abschließend und in der Rückschau, am Ende der Arbeit noch einige Fragen aufgegriffen werden, die während der Durchführung zu Problemen geführt haben, denen in einem zweiten Anlauf mehr Berücksichtigung zuteil werden könnte oder die am Ende offen geblieben sind.<sup>48</sup>

### 6.2.1 JDK 1.1 oder JDK 1.2?

Hätte die Implementierung des JRV-Systems mit dem JDK 1.2 anstatt mit dem JDK 1.1 durchgeführt werden sollen? Das JDK 1.2 integriert nicht nur die Swing-Bibliothek (und andere) in den Grundumfang des Gesamtsystems. Auch bereits im JDK 1.1 existierende Pakete wurden um neue Klassen erweitert, bestehende Klassen ergänzt. Als ein Beispiel sei hier nur das Paket `java.util` genannt, das einige neue und sehr interessante Klassen enthält.

Andererseits lassen bereits einige Anmerkungen in der Dokumentation des JDK 1.1 erahnen, daß eine Bearbeitung des JRV-Systems nötig sein wird, um es an das JDK 1.2 anzupassen. Bspw. heißt es im JDK 1.1 bei der Beschreibung der Methode `addPropertyChangeListener` der Klasse `JComponent`: „This method will migrate to `java.awt.Component` in the next major JDK release“. Wie viele Methoden dieser Art existieren?

Während des Verlaufs dieser Arbeit standen mehrere Betaversionen, außerdem ein sog. „Release Candidate“ und schließlich die (vorläufige?) Endversion des JDK 1.2 zur Verfügung. Vermutlich nicht umsonst lautet die aktuelle Bezeichnung von Sun für das Gesamtsystem „Java 2 Platform“. Der Wechsel des JDKs kann also durchaus mit einem Versionswechsel bei einem Betriebssystem verglichen werden.

Um nicht mit häufig wechselnden Versionen arbeiten zu müssen und aus den in Abschnitt 4.1.1 und 4.1.2 dargelegten Gründen wurde das JRV-System durchgehend auf der Basis des JDK 1.1 und mit separater Swing-Bibliothek 1.0.1 entwickelt. Der Autor würde beim nächsten mal wieder so verfahren.

---

<sup>48</sup> Anmerkung: Die Fragen werden hier hypothetisch und subjektiv aus der Sicht des Autors dieser Arbeit gestellt. Die Antworten geben die persönliche Meinung des Autors wieder, könnten mithin vom Leser also auch als falsch empfunden werden. Es bilde sich jeder eine eigene Meinung.

## 6.2.2 Existieren noch Fehler in den AWT- und JFC/Swing-Bibliotheken?

Oft dürfte der erste Gedanke eines Anwendungsentwicklers beim Auftreten eines Problems sein: „Das Betriebssystem ist schuld. Die Bibliothek funktioniert nicht wie beschrieben. Es handelt sich um einen Compilerfehler.“ Wenn später dann sog. „Service Packs“ oder „Bug Fixes“ erscheinen, waren solche Gedanken manchmal auch nicht ganz unbegründet. Am Ende dieser Arbeit kann man jedoch feststellen, daß sich das JDK 1.1 und die Swing-Bibliothek 1.0.1 durchweg sehr kooperativ verhalten haben. Lediglich bei wenigen Anomalien konnte abschließend der Verdacht aufrecht erhalten werden, daß es sich möglicherweise nicht um anwendungsabhängige Probleme handelt.

- Gelegentlich scheint sich bei JFC/Swing-Komponenten eine Neudarstellung des Bedienelements nicht durchzusetzen. Bereiche in denen der alte Fensterinhalt zurückbleibt sind die Folge.
- Wenn Bildlaufleisten vom Typ JScrollbar deserialisiert werden, erscheint der Schieber der Bildlaufleiste nicht, wenn vorher nicht bereits einmal eine Bildlaufleiste auf normalem Wege, d.h. nicht über Deserialisierung, erzeugt wurde.
- Das in Abschnitt 5.1.3 beschriebene Programm MDITest funktioniert nicht, wenn nicht das Windows Look and Feel verwendet wird. Er tritt ein Fehler beim Deserialisieren der Aufzeichnung auf.

## 6.2.3 Der Einsatz des JRV-Systems in einer „echten“ Anwendung

Gewissermaßen eine zusätzliche Erweiterung des JRV-Systems wäre der Einsatz oder die Integration in eine „echte“ Anwendung, d.h. in ein Umfeld, in dem das System an den konkreten Anforderungen eines Anwenders gemessen wird. Ist das System in der vorliegenden Form bereits für solche Einsätze tauglich oder müssen Anpassungen vorgenommen werden und wenn ja, in welchem Umfang?

Den Hintergrund für eine möglicherweise kritische Einstellung in dieser Hinsicht bietet die Frage, ob die in Kapitel 5 beschriebenen Testanwendungen nur gezielt so entwickelt wurden, daß sie auch funktionieren, oder ob sie tatsächlich repräsentativ sind.

Insbesondere das Design, aber auch die Implementierung des JRV-Systems wurden explizit auf Erweiterbarkeit und Anpassbarkeit ausgerichtet. Die Testanwendungen zeigen Situationen, wie sie in realen Anwendungen auftreten können.

Ein umfangreiches Anwendungsprogramm, in welches das System hätte integriert werden können, stand im vorgegebenen Zeitrahmen nicht zur Verfügung bzw. konnte nicht extra entwickelt werden. Selbst wenn dies der Fall gewesen wäre, ließe sich schließlich jetzt die Frage stellen, ob das System nur auf diese vorgegebene Anwendung ausgerichtet wäre.

### 6.3 Fazit

Im Rahmen dieser Arbeit wurde gezeigt, daß die Aufzeichnung und die Wiedergabe von grafischen Benutzungsschnittstellen im allgemeinen sowie des Java-AWT und der JFC/Swing-Bibliothek im konkreten Fall durch folgende Aspekte gekennzeichnet sind:

- Die Aufzeichnung und Wiedergabe der Benutzungsschnittstelle stellt einen Teilbereich der Anwendungsaufzeichnung im Allgemeinen dar, für den viele Anwendungsbereiche denkbar sind, bspw. Präsentationen, Benutzer Support, Makrofunktionalität.
- Um möglichst viele Anwendungsbereiche abdecken zu können, ist eine Integration des Aufzeichnungssystems in die aufgezeichnete Anwendung notwendig.
- Für die visuelle Wiedergabe ist ein anwendungsunabhängiger Viewer wünschenswert.
- Es existieren standardisierte Bedienelemente, für die auch standardisierte Aufzeichnungs- und Wiedergabemechanismen implementiert werden können.
- Die Ermittlung und Wiederherstellung des Zustands von aufzuzeichnenden Objekten wird durch die Kapselung der Objekte durch das Betriebssystem oder eine Objektbibliothek erschwert.
- Die Aufzeichnung von Ereignissen kommt im Wesentlichen dem Überwachen der Nachrichten gleich, über die sowohl Ereignisse vom Betriebssystem an die Anwendung als auch anwendungsinterne Zustandsänderungen übermittelt werden können.
- Es gibt systemnahe und anwendungsnahe Ereignisse, die beide bei der Aufzeichnung berücksichtigt und bei der Wiedergabe synchronisiert werden müssen.
- Durch die zusätzliche Abstraktion vom Betriebssystem, die Java in Form der virtuellen Maschine bietet, wird die Aufzeichnung und Wiedergabe der Benutzungsschnittstelle, die letztendlich systemnah ist, erschwert.
- Ein optimales Aufzeichnungs- und Wiedergabesystem läßt sich vermutlich nur realisieren, wenn die entsprechende Funktionalität in die Benutzungsschnittstelle selbst integriert ist.

Für das Java-AWT und die JFC/Swing-Komponenten wurde ein Aufzeichnungs- und Wiedergabesystem, das JRV-System, entwickelt, das sich durch folgende Merkmale auszeichnet:

- Das System kann als Bibliothek in beliebige Java-Programme integriert werden.
- Ein objektorientiertes Design gewährleistet einfache Erweiterbarkeit und die Anpassungsfähigkeit an unterschiedliche Anwendungssituationen. Sowohl auf der Quellcode-Ebene, als auch zur Laufzeit.
- Für die Aufzeichnung und Wiedergabe vieler Elemente der JFC/Swing-Bibliothek wurden fertige, „ready-to-use“ Klassen implementiert.
- Für die rein visuelle Wiedergabe steht ein Runtime-Player zur Verfügung.
- Das System wurde anhand von Beispielprogrammen, die unterschiedliche Einsatzbereiche abdecken, erfolgreich getestet.

## Literaturverzeichnis

- BFM97 Hartmut Benz, Steffen Fischer und Rolf Mecklenburg, „Architecture and Implementation of a Distributed Multimedia Annotation Environment: Practical Experiences using Java“. In Hartmut König, Kurt Geihs und Thomas Preuß, Editoren, „Distributed Applications and Interoperable Systems“, S. 49 - 59, Boundary Row, London Se1 8HN, UK, Oktober 1997, Chapman & Hall.
- BFMW97 Hartmut Benz, Steffen Fischer, Rolf Mecklenburg und Andreas Wenger, „Application Output Recording for Instant Authoring in a Distributed Multimedia Annotation Environment“. In Ralf Steinmetz und Lars C. Wolf, Editoren, „Interactive Distributed Multimedia Systems and Telecommunication Services“, 4th International Workshop, IDMS `97, Darmstadt, Deutschland, Nr. 1309 in „Lecture Notes in Computer Science“, S. 220 - 230. Springer Verlag, Berlin, September 1997.
- BHB+97 Sandford Bessler, Michael Hager, Hartmut Benz, Rolf Mecklenburg und Steffen Fischer, „DIANE: A Multimedia Annotation System“. In Serge Fdida und Michele Morganti, Editoren, „Multimedia Applications, Services and Techniques“, ECMAST `97, Nr. 1242 in „Lecture Notes in Computer Science“, S. 183 - 198. Springer Verlag, Berlin, May 1997.
- BR95 Kraig Brockschmidt, „Inside OLE“, 2nd Edition, Microsoft Press, Redmond, USA, 1995.
- CH98 Patrick Chan, „The Java™ Developers Almanac“, Addison-Wesley, USA, 1998.
- CW98 Mary Campione und Kathy Walrath, „The Java™ Tutorial Second Edition“, Addison-Wesley, USA, 1998.
- CT199 Dr. Oliver Diedrich (odi), „Linux erobert den Server-Markt“, c't Magazin für Computertechnik, Ausgabe 1/1999, Verlag Heinz Heise GmbH & Co KG, Hannover 1999.
- DIANE [http://www.kapsch.co.at/forschung/diane/eng/diane\\_home.html](http://www.kapsch.co.at/forschung/diane/eng/diane_home.html)
- DIN44-1 DIN 44300, Teil 1
- DUDEN „DUDEN Informatik - Ein Sachlexikon für Studium und Praxis“, 2.Auflage, Bibliographisches Institut & E.A. Brockhaus AG, Mannheim 1993.
- FL93 David Flanagan, „X Toolkit Intrinsics Reference Manual“, O'Reilly & Associates, USA, 1993.
- GA95 William H. Gates III., „Der Weg nach vorn: Die Zukunft der Informationsgesellschaft“, Hoffmann und Campe Verlag, Hamburg, 1995.
- HF94 Dan Heller und Paula M. Ferguson, „Motif Programming Manual“, O'Reilly & Associates, USA, 1994.
- JCJÖ92 Ivar Jacobsen, Magnus Christerson, Patrik Jonsson, Gunnar Övergaard, "Object-Oriented Software-Engineering - A Use Case Driven Approach", ACM press, 1992
- MA94 Kyle Marsh, „Safe Subclassing in Win32“, MSDN Library Visual Studio 6.0, Technical Articles, Windows Platform, Windows Management, Microsoft, 1998.
- MD95 Microsoft Corporation and Digital Equipment Corporation, „The Component Object Model Specification“, Draft Version 0.9, October 24, 1995

- MP95 Nelson R. Manohar und Atul Prakash, „The Session Capture and Replay Paradigm for Asynchronous Collaboration“. In H. Marmolin, Y. Sundblad und K. Schmidt, Editoren, „Proceedings of the 4th European Conference on Computer-Supported Work“, ECSCW `95, Stockholm, Schweden, S. 149 - 164. Kluwer Academic Publishers, September 10 - 14, 1995.
- MP96 Nelson R. Manohar und Atul Prakash, „Tool Coordination and Media Integration on Asynchronously-Shared Computer-Supported Workspaces“. CSE Division Technical Report CSE-TR-284-96, Department of Electrical Engineering and Computer Science, University of Michigan at Ann Arbor, Februar 1996.
- NE+97 Alexander Newman u.a., „Java: Referenz & Anwendungen“, Que, Markt & Technik Buch- und Software-Verlag GmbH, München, 1997.
- NO93 Adrian Nye und Tim O'Reilly, „X Toolkit Intrinsic Programming Manual“, O'Reilly & Associates, USA 1993.
- NY92 Adrian Nye, „Xlib Reference Manual“, O'Reilly & Associates, USA, 1992.
- NY95 Adrian Nye, „Xlib Programming Manual“, O'Reilly & Associates, USA, 1995.
- RI97 Jeffrey Richter, „Developing Applications for Microsoft Windows CE: An Overview of the Windows CE SDK and Visual C++ for Windows CE“, MSDN Library Visual Studio 6.0, Microsoft 1998.
- RSC97 "Unified Modeling Language Notation Guide version 1.0", <http://www.rational.com>, Rational Software Corporation, Santa Clara, USA, 1997
- SDK-A N.N., „Using the Win32 API“, MSDN Library Visual Studio 6.0, Platform SDK, Windows Programming Guidelines, Win32 Programming, Microsoft 1998.
- SDK-H N.N., „Hooks“, MSDN Library Visual Studio 6.0, Platform SDK, Windows Base Services, Inter-process Communication, Microsoft 1998.
- SUN98 „SUN ANNOUNCES AVAILABILITY OF JAVAPC™ ENGINE 1.0“, <http://java.sun.com/pr/1998/06/pr980630.html>, USA, 1998
- SUN99 „CALLING THE INTERNET Java™ Technology Phone Wins Big Award“, <http://java.sun.com/features/1999/01/alcatel.html>, USA, 1999
- SN95 Ralf Steinmetz, Klara Nahrstedt, „Multimedia: Computing, Communications and Applications“, Prentice-Hall, USA, 1995.
- SH92 Ben Shneiderman, „Designing the User Interface - Strategies for Effective Human-Computer Interaction“, 2. Auflage, Addison-Wesley Publishing Company, Inc., 1992
- SH97 Peter Stahlknecht, Ulrich Hasenkamp, „Einführung in die Wirtschaftsinformatik“, 8.Auflage, Springer-Verlag, Berlin, 1997.
- STA96 Christian Stary, „Interaktive Systeme, Software-Entwicklung und Software-Ergonomie“, 2.Auflage, Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig, Wiesbaden 1996.
- STE98 Ralph Steyer, "JAVA 1.2 KOMPENDIUM", Markt & Technik Buch- und Software-Verlag GmbH, Haar bei München, Deutschland, 1998
- WA98 Scott R. Weiner, Stephen Asbury, „Programming with JFC“, John Wiley & Sons, Inc., USA, 1998.

## Erklärung

Ich versichere, daß ich diese Arbeit selbständig verfaßt und nur die angegebenen Hilfsmittel verwendet habe.

---

Jörg Schließer