


B5. Frameworks zur Medieneinbindung

- B5.1 Frameworks 
- B5.2 Anforderungen an Medien-Frameworks
- B5.3 Java Media Framework im Überblick
- B5.4 Videoverarbeitung mit dem Java Media Framework

Literatur:

H. M. Eidenberger, R. Divotkey: Medienverarbeitung in Java,
dpunkt-Verlag 2004

<http://java.sun.com/products/java-media/jmf>

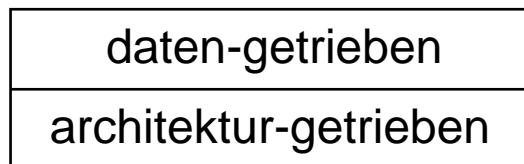
<http://www.jmfapi.org>

Frameworks

- **Definition** (Taligent): „A **framework** is a set of prefabricated software building blocks that programmers can use, extend, or customize for specific computing solutions.“
- **Definition** (nach Pomberger/Blaschek): Ein "**framework**" (Rahmenwerk, Anwendungsgerüst) ist eine Menge von zusammengehörigen Klassen, die einen abstrakten Entwurf für eine Problemfamilie darstellen.
- **Ziele:**
 - Wiederverwendung von Code, Architektur und Entwurfsprinzipien
 - Wiederverwendung von Verhaltensschemata einer Gruppe von Klassen
 - Homogenität unter verschiedenen speziellen Anwendungssystemen für eine Problemfamilie (z.B. ähnliche, ergonomische Bedienung)

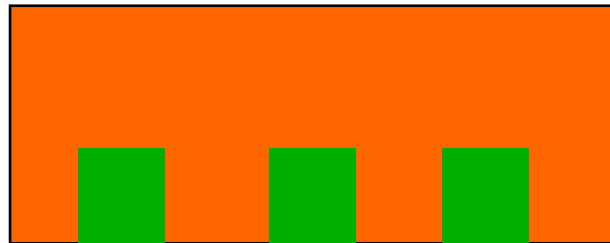
Klassifikation von Frameworks

- Architektur-getriebenes Framework (*architecture-driven*):
 - Anpassung durch Vererbung und Überdefinieren
 - Komplexe Klassenhierarchien und Muster
 - Relativ viel neuer Code zu schreiben
 - Anpassung erfordert sehr hohen Einarbeitungsaufwand
- Daten-getriebenes Framework (*data-driven*):
 - Anpassung durch Objektkonfiguration und Parametereinstellung
 - Weitergeleitete Objekte bestimmen Verhalten (z.B. Ereignis-Objekte)
 - Relativ einfach anzupassen
 - Eingeschränkte Flexibilität
- Möglicher und sinnvoller Kompromiß:
 - Zwei-Schichten-Architektur:



Vergleich Klassenbibliothek-Framework

Klassenbibliothek

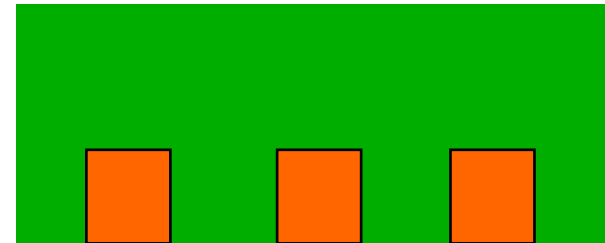


-  Anwendungsspezifische Teile
-  Vorgefertigte Teile

Anpassung
nur durch Instanziierung

Ablaufsteuerung
nicht vordefiniert

Framework



***"Don't call us,
we call you"***

(„Hollywood-Prinzip“)

Anpassung
auch durch Spezialisierung

Ablaufsteuerung
im wesentlichen vordefiniert

Die Grenze ist fließend, siehe z.B. Java AWT !

B5. Frameworks zur Medieneinbindung

B5.1 Frameworks

B5.2 Anforderungen an Medien-Frameworks



B5.3 Java Media Framework im Überblick

B5.4 Videoverarbeitung mit dem Java Media Framework

Literatur:

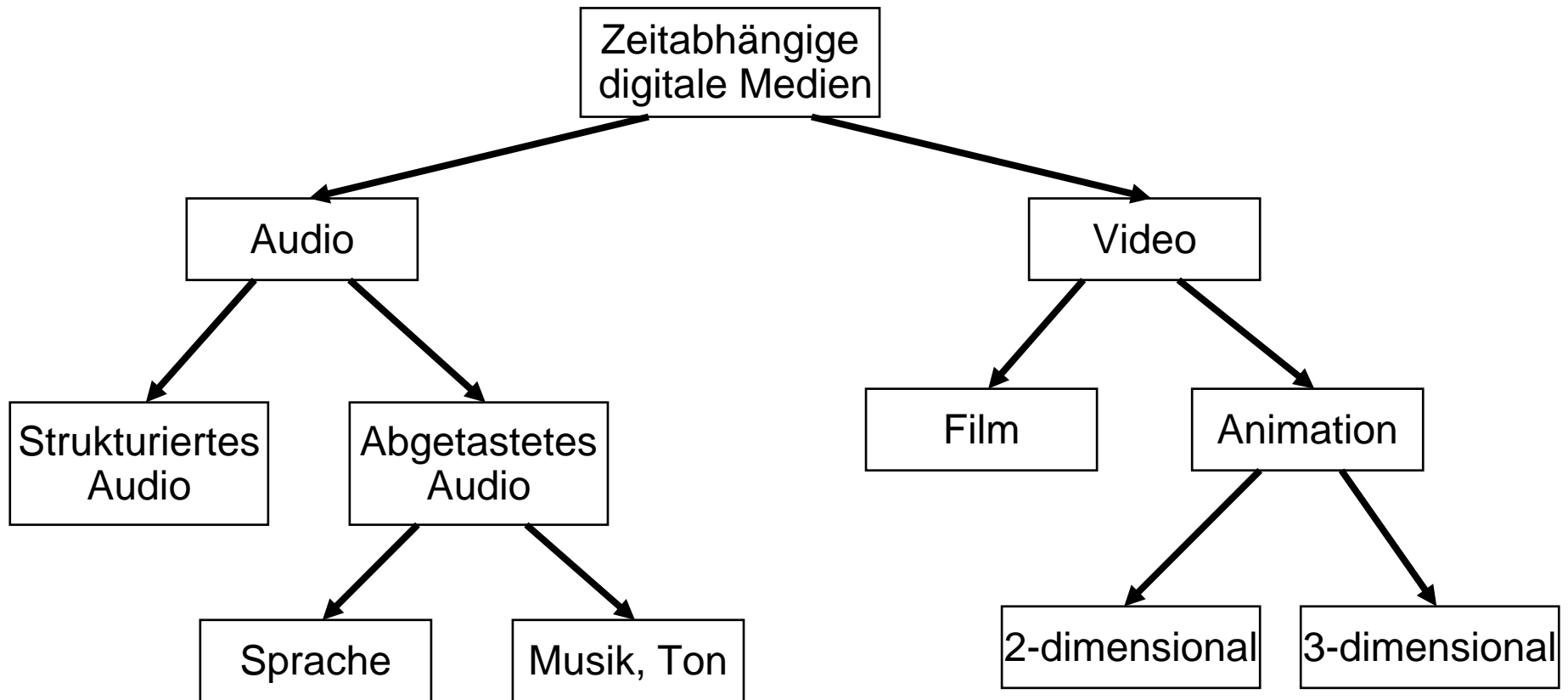
H. M. Eidenberger, R. Divotkey: Medienverarbeitung in Java.
dpunkt.Verlag 2004

<http://www.jmfapi.org>

Medien-Frameworks

- Infrastruktur für folgende Aufgaben:
 - Behandlung von zeitabhängigen Medien unter Beachtung von
 - » Synchronisation
 - » verschiedenen Formaten
 - » verschiedenen Quellen/Senken und deren Protokollen
 - Wiedergabe (Rendering)
 - Verarbeitung (Konversion, Kodierung, Effekte)
 - Aufnahme
- Implementierungsbasis für folgende Arten von Software:
 - „Player“ für dateigebundene und aus einem Netz zugeführte („streaming“) zeitabhängige Medien
 - Server für zeitabhängige Medien im Netz
 - Unterstützungssoftware für Medienproduktion
 - » Audio- und Video-Schnitt und -Bearbeitung

Zeitabhängige digitale Medien: Überblick



Datensichten für zeitabhängige Medien

- Zeitabhängiges Medium als Datenbehälter (*container*):
 - Komplexe Datenstruktur
 - Gesamtdatenmenge in einer Datei
 - Prinzipiell alle Bestandteile beliebig zugreifbar
 - Zeitabhängigkeit nicht relevant
- Zeitabhängiges Medium als Strom (*stream*):
 - Repräsentierende Datenstruktur stellt nur ein „Fenster“, einen „Schnappschuss“ auf die Gesamtdatenmenge dar
 - Aktuell empfangene Daten ändern sich über die Zeit
 - Prinzipiell unendliche Sequenz (d.h. „Strom“) von Dateneinheiten (Frames, Paketen)

Deklarative und prozedurale Medienverarbeitung

- Deklarative Medienverarbeitung:
 - Definiert, *was* getan werden soll, aber nicht *wie*
 - Beispiel für Deklaration von Multimedia-Präsentationen:
 - » SMIL (Synchronized Multimedia Integration Language)
(siehe Vorlesung „Digitale Medien“)
- Prozedurale Medienverarbeitung:
 - Definiert, *wie* die Medienverarbeitung erfolgt
 - Konkret zu verwendende oder erzeugende Medien treten meist nur als Parameter auf
 - Unterstützungssoftware für deklarative Medienverarbeitung (sog. *User agents*) werden durch prozedurale Medienverarbeitung realisiert

Behandlung von Zeitabhängigkeit in Software

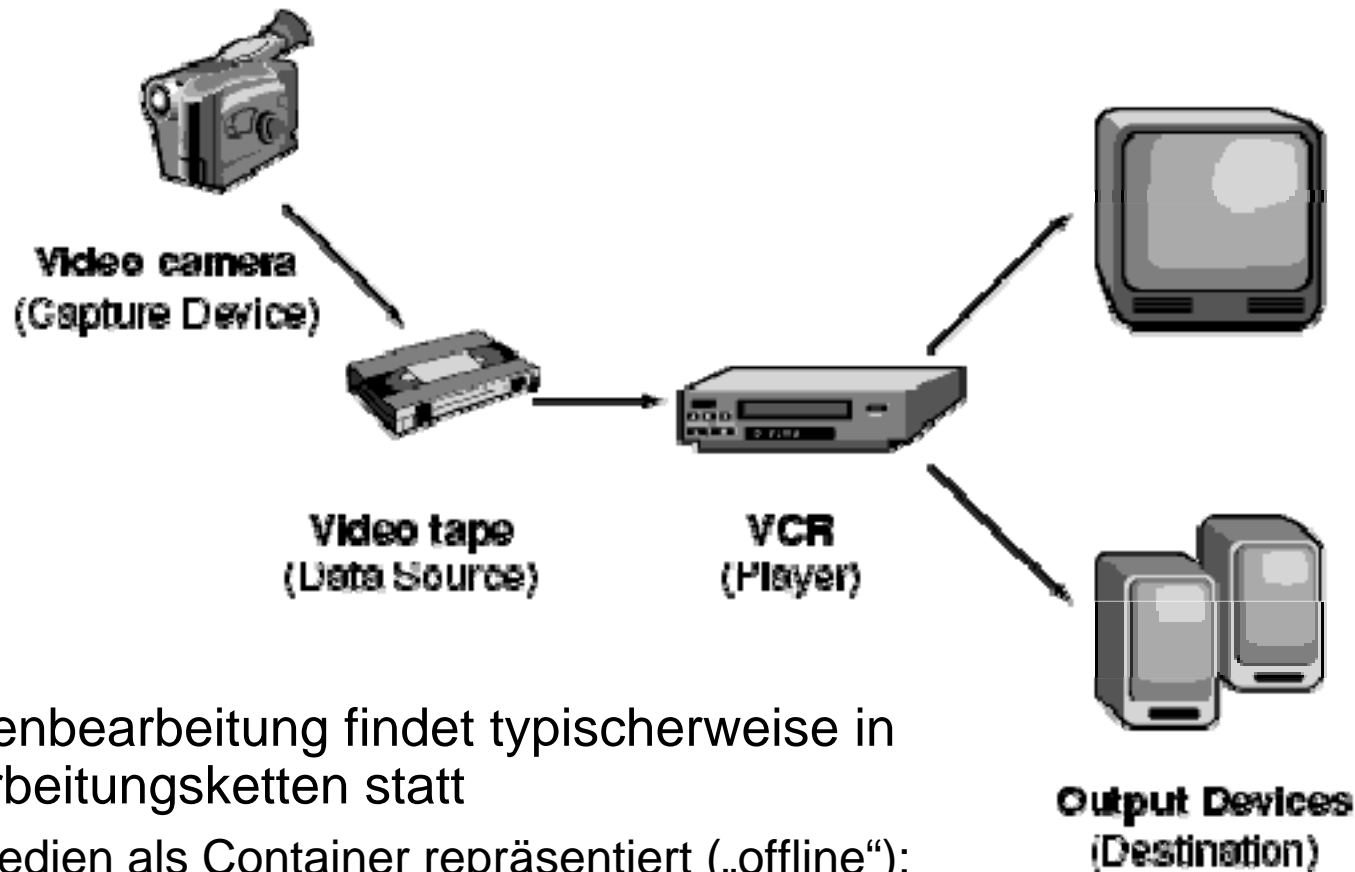
- Starre Koordination entlang einer Zeitleiste
 - Übersichtlich, Synchronisation leicht verständlich
 - Leicht zu visualisieren
(z.B. in Macromedia Flash & Director, Adobe Premiere)
 - Unflexibel: Starrer Ablauf
 - Häufigste Basis für deklarative Medienverarbeitung
- Ereignis-getriebene Koordination
 - Ereignisse: Benutzerinteraktion, Programmzustände, Zeitablauf
 - Schwieriger darzustellen (z.B. als Bedingungs-/Ereignis-Netz)
 - Flexibler und allgemeiner
 - Häufigste Basis für prozedurale Medienverarbeitung

Synchronisation

- Intramedia-Synchronisation:
 - Synchronisation von Elementen eines Datenstroms
 - » Z.B. der Frames eines Video-Datenstroms
 - Meist auf relativ niedrigem Abstraktionsniveau in der Wiedergabesoftware sichergestellt

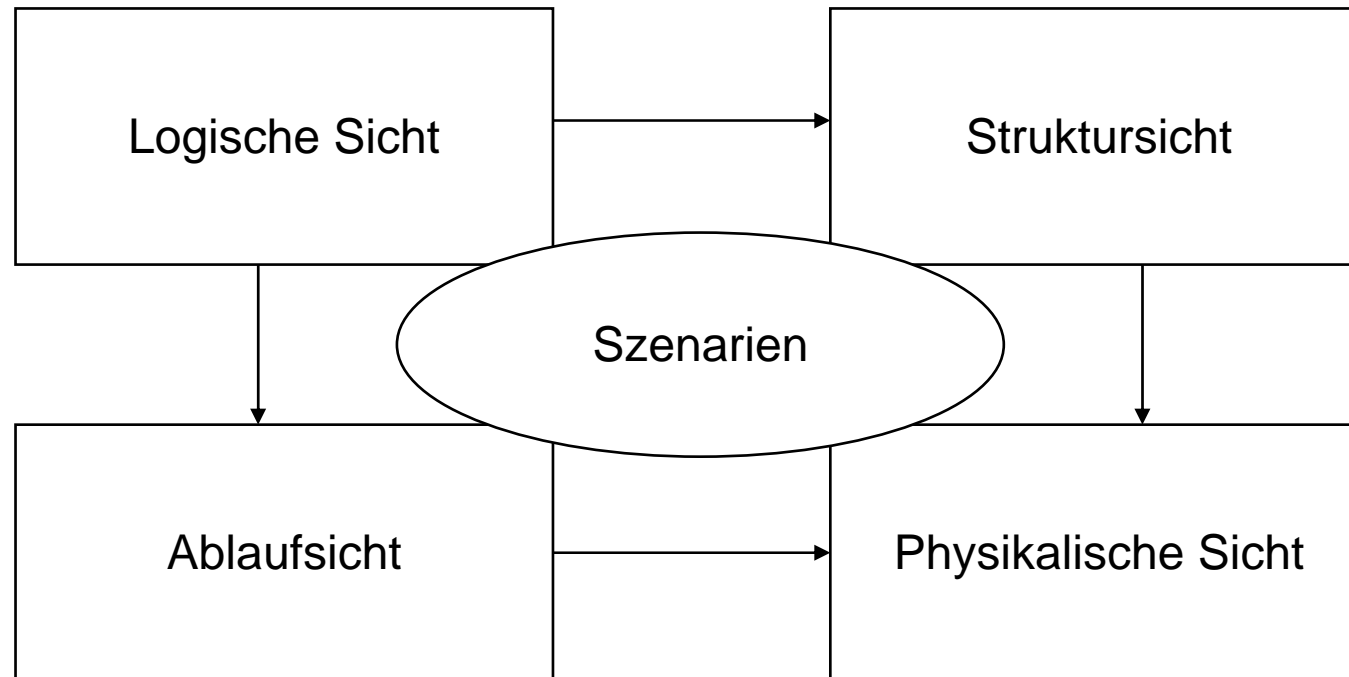
- Intermedia-Synchronisation
 - Synchronisation zwischen verschiedenen Datenströmen
 - » Z.B. Synchronisation zwischen Ton- und Bildinformation
 - Zwei Alternativen zur Definition:
 - » Angaben zu zeitlichen Relationen (z.B. in SMIL)
 - » Angaben in absoluten oder relativen Zeitwerten (z.B. beim Videoschnitt)

Verarbeitungsketten in der Medienbearbeitung



- Medienbearbeitung findet typischerweise in Verarbeitungsketten statt
 - Medien als Container repräsentiert („offline“):
Lange Bearbeitungszeiten, grosse Dateneinheiten
 - Medien als Ströme repräsentiert („online“):
Kurze Bearbeitungszeiten, kleine Dateneinheiten

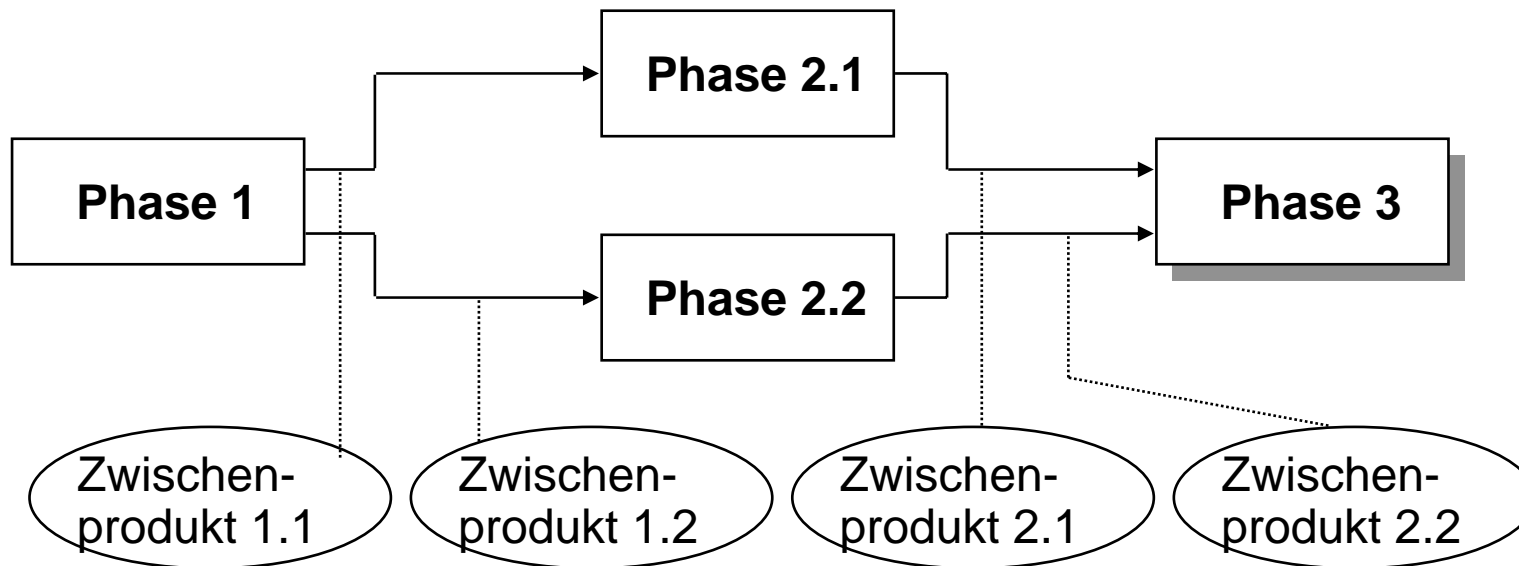
„4+1 Sichten“-Modell der Softwarearchitektur



Philippe Kruchten, The 4+1 view model of architecture, IEEE Software, November 1995, 12(6), pp. 42-50

<http://www.rational.com/support/techpapers/ieee>

Architekturmuster "Kette"



- Inkrementelle oder phasenweise Verarbeitung
- Anderer Name: Pipes & Filters
- Beispiele:
 - UNIX pipes
 - Batch-sequentielle Systeme
 - Compiler-Grundstruktur

Abstraktes Modell von Verarbeitungsketten

- Transformator (*transform*)
 - Verarbeitungseinheit, die Mediendaten aufnehmen und/oder produzieren kann
 - Spezialfälle von Transformatoren:
 - » Eingabe (*capturing*)
 - » Entpacken (*demultiplexing*)
 - » Dekodieren (*decoding*) (codec = coder/decoder)
 - » Verarbeitung (*processing*)
 - » Kodieren (*coding*)
 - » Packen (*multiplexing*)
 - » Ausgabe (*rendering*)
- Anschluss (*port*)
 - Verbindungsmöglichkeit für Transformatoren
 - » Transformator hat mindestens einen Port
 - Eingabe- und Ausgabeports

Klassifikation von Verarbeitungsketten (1)

- Nach der Art der Zeitanforderung:
 - Echtzeit:
 - » Obergrenze für die Dauer der Verarbeitung gegeben
 - » Verarbeitungsdauer ist bestimmt durch die Frequenz des Eingabemediums (bzw. deren Kehrwert)
 - » Strategien bei Nichteinhaltung der Zeitgrenze durch Transformator: Verwerfen (*drop*) oder verzögert weiterleiten
 - Keine Echtzeit:
 - » Keine Obergrenze für Verarbeitungsdauer
- Nach der Pufferung:
 - Ungepuffert:
 - » Ermöglicht höchste Aktualität der Ausgabe (z.B. bei Telefonie)
 - » Grösste Empfindlichkeit gegen Schwankungen der Verarbeitungsdauer
 - Gepuffert:
 - » Begrenzter Ausgleich von Schwankungen der Verarbeitungsdauer
 - » Reduzierte Aktualität des Endergebnisse (z.B. Videowiedergabe)

Klassifikation von Verarbeitungsketten (2)

- Nach dem Kontrollfluss:
 - Wer steuert die Zeitbasis der Verarbeitungskette?
- Aktiver Transformer
 - Arbeitet immer mit strombasierter Repräsentation
 - Sendet nach eigenem Zeittakt Dateneinheiten in die Kette
 - » Z.B. Videokamera
 - *Push mode*
- Passiver Transformer
 - Arbeitet meist mit behälterbasierter Repräsentation
 - Liefert Dateneinheiten auf Nachfrage
 - *Pull mode*
- Separate Steuerung der Verarbeitungskette?
 - Eigenes Steuerungsobjekt bei nur passiven Transformatoren
 - Ein aktiver Transformator bei sonst nur passiven Transformatoren
 - Koordination unter mehreren aktiven Transformatoren

B5. Frameworks zur Medieneinbindung

B5.1 Frameworks

B5.2 Anforderungen an Medien-Frameworks

B5.3 Java Media Framework im Überblick 

B5.4 Videoverarbeitung mit dem Java Media Framework

Literatur:

H. M. Eidenberger, R. Divotkey: Medienverarbeitung in Java.
dpunkt.Verlag 2004

<http://www.jmfapi.org>

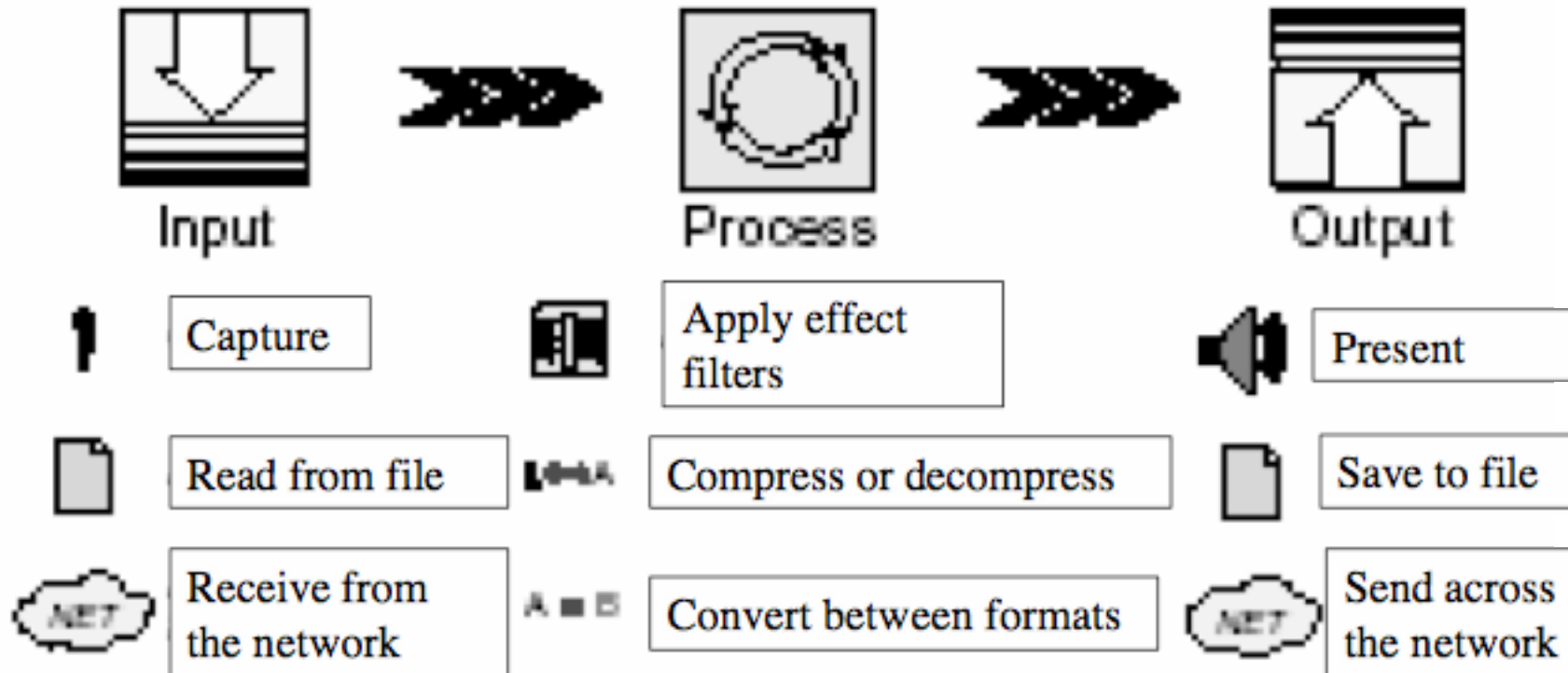
Java Media Framework

- Kostenloses, plattformunabhängiges und objektorientiertes Medien-Framework
- Hauptfunktionen:
 - Abspielen von Medien
 - Verarbeitung von Mediendaten in Echtzeit
 - Erfassung von Datenströmen
 - Speichern von Mediendaten
 - Strombasierte Übertragung (*streaming*) von Mediendaten
- Geschichte:
 - JMF 1.0 (Sun, SGI, Intel): 1998
 - JMF 2.0 (Sun, IBM): 1999
 - Aktuelle Fassung JMF 2.1.1: 2001
 - » realisiert Funktionalität von 2.0
- JMF *nicht* Bestandteil der Standard-Java2-Distribution
 - Cross-Platform-Implementierung und „Performance Packs“

Alternativen zum JMF

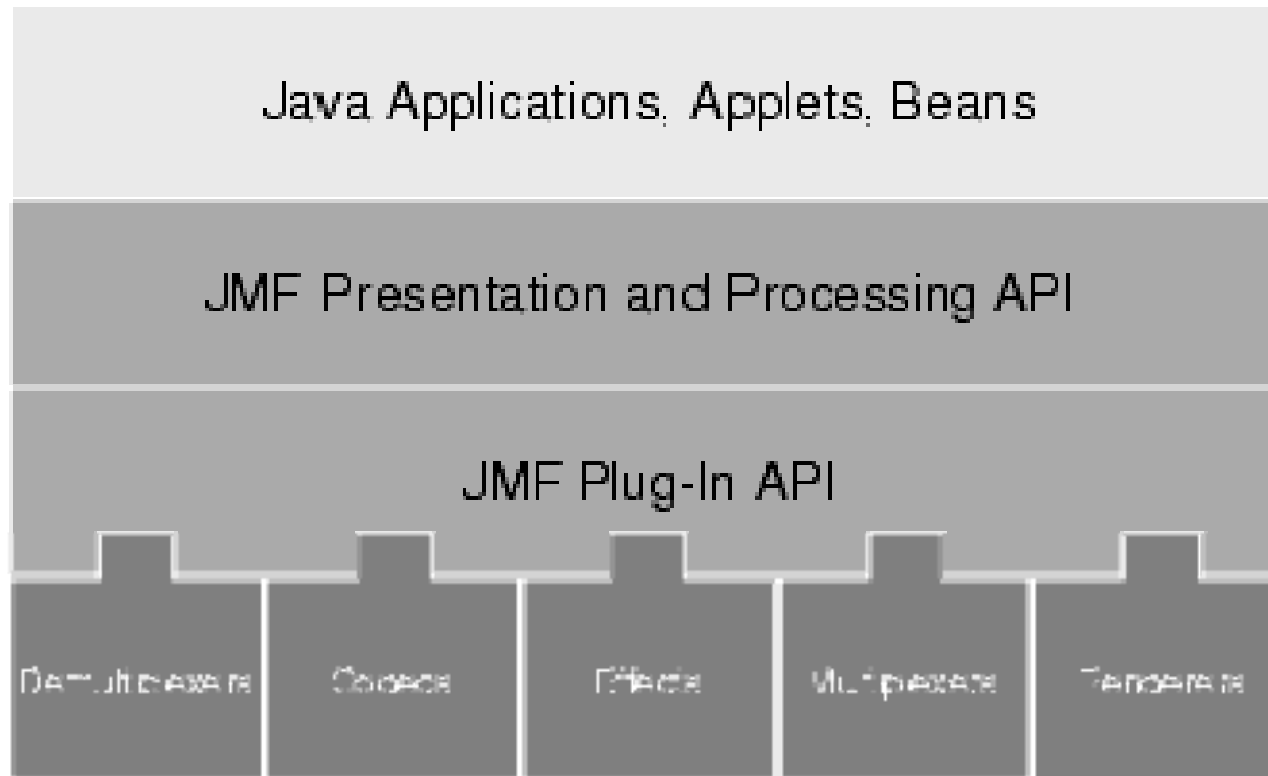
- OpenML (Khronos Group = SDLabs, SGI u.a.)
 - Hardwarenahes C-API
 - Als Äquivalent zum OpenGL-Grafikstandard intendiert
 - Implementierungen sehr schnell
 - Geeignet als Implementierungsbasis für Frameworks höherer Abstraktion
- DirectShow (Microsoft)
 - Bestandteil von DirectX (ab Version 8.0)
 - Erweiterte und verbesserte Fassung von „Video for Windows“
 - Methoden zur Erfassung, Verarbeitung und zum Transport von Medien
 - Filter (Source, Transform, Rendering), verbunden über Pins
 - Gute Dateiformatunterstützung
 - Nachteil: Windows-spezifisch, nicht Bestandteil von .NET
- QuickTime (for Java) (Apple)
 - C-Bibliothek (nun auch mit Java-Wrappern)
 - Umfasst neben Medienstrom-Verarbeitung auch Bildverarbeitung, Animation, 3D-Modellierung u.a.

Verarbeitungsketten-Modell in JMF



- Prinzipiell ist jede Kombination der möglichen Eingabeoptionen, Verarbeitungsschritte und Ausgabeoptionen möglich

High-Level-Architektur von JMF



- Durch „Plug-Ins“ sehr flexibel für die einheitliche Unterstützung verschiedener Medientypen und für nachträgliche Erweiterungen

Medien in JMF

- Medientypen: anhand eines MIME-Typs beschrieben
 - z.B. „video/quicktime“ oder „audio/midi“
 - » siehe Vorlesung „Digitale Medien“
 - Alternativ auch nur Grobspezifikation des Datenstroms:
 - » RAW (nur eine Art von Daten)
 - » MIXED (Tracks mehrerer Arten)
 - » UNKNOWN
- Abstraktion eines Medienstroms in JMF:
 - Objekt der Klasse `ContentDescriptor`
- Abstraktion eines Medienbehälters (Datei) in JMF:
 - Objekt der Klasse `FileTypeDescriptor`
(Unterklasse von `ContentDescriptor`)

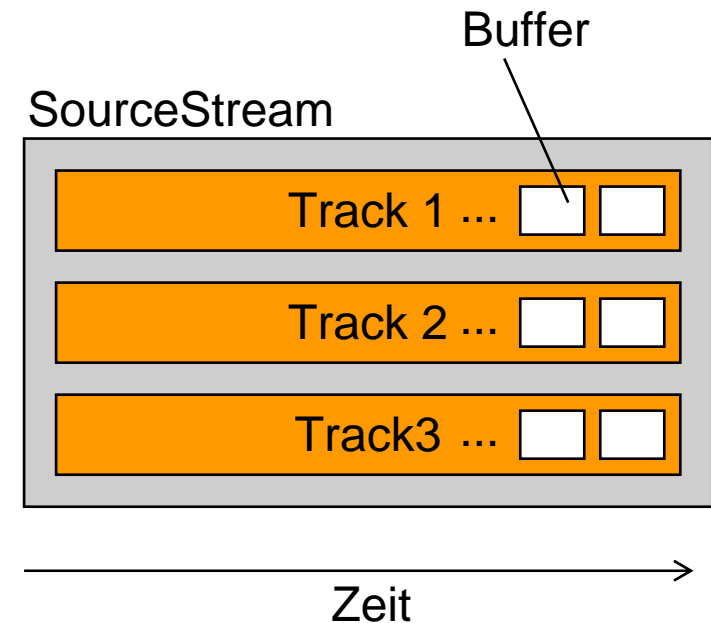
Unterstützte Dateiformate

- Standardausstattung von JMF-Implementierungen
 - jederzeit durch Plugin-Mechanismus erweiterbar!

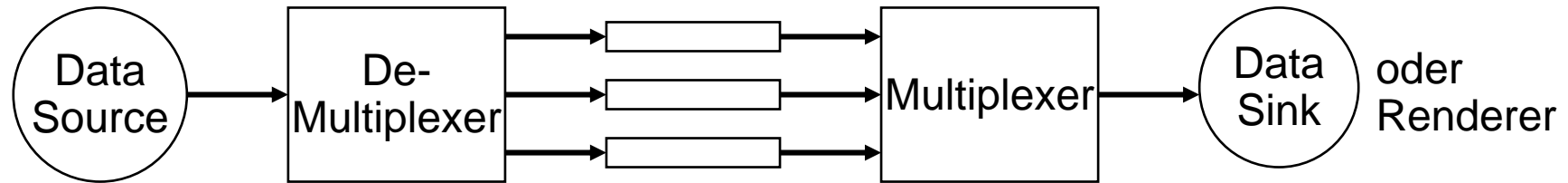
| Videoformate | Audioformate |
|---|---|
| AVI MPEG (nicht bei „cross- platform“““) Quicktime | AIFF GSM WAVE Sun Audio MIDI RMF |

Packungsgrad von Medien

- **SourceStream:**
 - Kapselt Medium
 - Beschrieben durch `ContentDescriptor`
- **Track:**
 - Einzelkomponente eines Stroms (z.B. Video-, Audiospur)
 - Zugriff auf Mediendaten
- **Buffer:**
 - Einzelner Datenblock eines **Track**
 - Wird zur Weitergabe von Daten in Verarbeitungsketten genutzt
 - Detaillierte Beschreibung: **Format-Objekt**
 - » **AudioFormat**
 - » **VideoFormat**
 - **RGB, YUV, JPEG, ...**

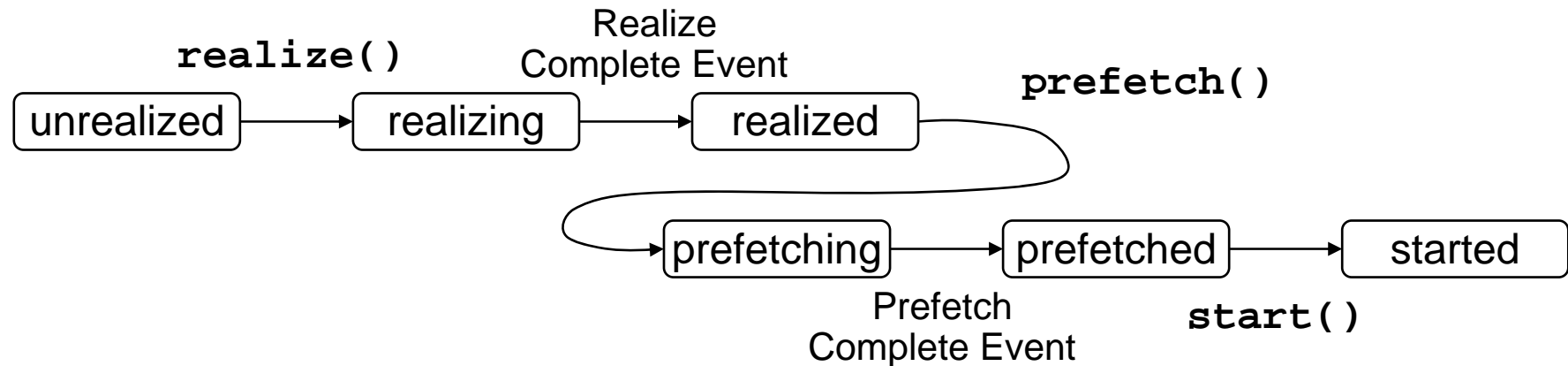


Verarbeitungsketten in JMF



- Kette von Transformatoren in JMF:
 - Echtzeitanforderung: Bei Zeitüberschreitung wird **Buffer** gelöscht
 - Alle Transformatoren passiv
 - Wer steuert die ganze Kette (und gibt z.B. den Zeittakt vor)?
- **Controller-Schnittstelle**
 - Schnittstelle für Steuerung von Verarbeitungsketten
 - Spezialfälle:
 - » **Processor**: Allgemeine Verarbeitungskette
 - » **Player**: Einfacher Fall mit trivialer Transformation und Rendering
 - **DataSink**: Dateiausgabe, ähnlich aber nicht von **Controller** abgeleitet

Zustandsmodell von Player



- *Unrealized:*
 - Anfangszustand
- *Realizing:*
 - Medienabhängige Teile des Players werden bereitgestellt
- *Prefetching:*
 - Eingabestrom wird soweit gelesen wie nötig, um Puffer zu füllen
- *Started:*
 - Verarbeitung läuft

Ereignis-Konzept in JMF

- Ereignisse werden wie in AWT/Swing durch *callback* realisiert
- Bei einem `Player` werden Objekte mit `addControllerListener` registriert, die Controller-Ereignisse interpretieren

```
public interface javax.media.ControllerListener {  
    public void controllerUpdate(ControllerEvent event)  
}
```

- Beispiele für Controller-Ereignisse (Unterklassen von `ControllerEvent`):
 - `RealizeCompleteEvent`
 - `PrefetchCompleteEvent`
 - `StartEvent`
 - `StopAtTimeEvent`
 - `EndOfMediaEvent`
 - `FormatChangeEvent`
 - `RateChangeEvent`
 - `StopTimeChangeEvent`

Zeit in JMF

- **TimeBase:**
 - Schnittstelle, die allgemeine Anforderungen an einen Taktgeber beschreibt
 - implementiert durch **SystemTimeBase** (Zeitbasis des JMF)
- Zeit-Datentypen:
 - **Time:** Zeitpunkte
 - **Duration:** Zeitdifferenzen, Zeitdauern
- **Clock:**
 - Lokale Zeit eines Controllers (enthält eine **TimeBase**)
 - Erlaubt viele Methoden zum Verändern der aktuellen Zeit
 - » **MediaTime:** Aktuelle Position im Medienstrom
 - » **Rate:** Relative Geschwindigkeit zur **TimeBase** (z.B. +2, +0.5, -2)
 - $MediaTime = mst + (tbt - tbst) * Rate$
 - » mst = media start-time
 - » tbt = time-base time
 - » $tbst$ = time-base start-time

Manager-Klassen in JMF

- Vermittlerobjekte zwischen den zahlreichen Interfaces und Implementierungen davon:
 - *Manager*
 - » zur Verwaltung und Konstruktion von Controller-Objekten (*Player*-, *Processor*-, *DataSource*-, und *DataSink*- Objekte)
Beispiel: `Manager.createPlayer(DataSource d);`
 - *PackageManager* (Verwaltung aller Pakete, die die JMF-Dateien enthalten)
 - *CaptureDeviceManager* (Verwaltung aller vorhandenen Eingabegeräte)
 - *PluginManager* (Verwaltung aller verfügbaren Plug-Ins, z.B. Multiplexer, Codecs)
 - *RTPManager* (Verwaltung von Streaming-Sitzungen)
- **MediaLocator**
 - Erwartet Information zu Protokoll, Hostname, Dateiname in URL-ähnlicher Syntax
 - Lokalisiert Medienquelle und richtet notwendige Verwaltung (z.B. Pufferung) ein
 - (Viele JMF-Funktionen erlauben auch direktere Angabe von Medienquellen)

Beispiel 1: Trivialer Audio/Video-Player (1)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.media.*;

class SimplePlayerFrame extends JFrame {

    private Player p = null;
    private SimplePlayerFrame f = this;

    public SimplePlayerFrame(String file) {
        setTitle("Simple JMF Player");
        getContentPane().setLayout(new BorderLayout());
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent event) {
                p.stop();
                p.deallocate();
                System.exit(0);
            }
        });
    }

    ...
}
```

Beispiel 1: Trivialer Audio/Video-Player (2)

```
...
    try {
        p = Manager.createPlayer(new MediaLocator("file:"+file));
        p.addControllerListener(new ContrEventHandler());
        p.realize();
    }
    catch (Exception e) {
        System.out.println("Exception "+e);
        System.exit(-1);
    }
}

public void addControlPanel() {
    getContentPane().
        add(p.getControlPanelComponent(), BorderLayout.SOUTH);
    Component vc = p.getVisualComponent();
    if (vc != null)
        getContentPane().add(vc, BorderLayout.NORTH);
    setSize(400,300);
    setVisible(true);
}
...
```


Beispiel 1: Trivialer Audio/Video-Player (3)

```
...
class ContrEventHandler implements ControllerListener {

    public synchronized void controllerUpdate(ControllerEvent e) {
        if (e instanceof RealizeCompleteEvent) {
            f.addControlPanel();
            p.start();
        }
        else if (e instanceof EndOfMediaEvent) {
            p.stop();
            p.setMediaTime(new Time(0));
            p.start();
        }
    }
}

public class SimplePlayerApp {

    public static void main(String[] argv) {
        SimplePlayerFrame pf = new SimplePlayerFrame(argv[0]);
    }
}
```