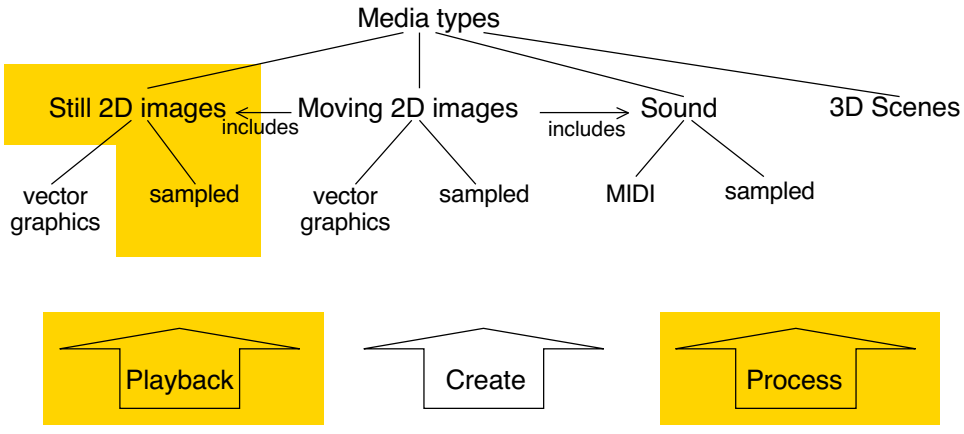
4 Fundamental Issues in Multimedia Programming

Literature:

J. Knudsen, Java 2D Graphics, O'Reilly 1999

Image Processing Programs



Bildbearbeitung in Java

- Frühe Java-Versionen:
  - In AWT Einlesung und Anzeigen von Bildern unterstützt
  - Noch keine Funktionen zur Modifikation von Bildern
- Java 2D:
  - Bild als Bestandteil der Rendering-Kette
  - Begrenzter Satz von Bildbearbeitungsfunktionen
- Java Advanced Imaging (JAI):
  - November 1999, Portierung auf diverse Plattformen noch im Gang
  - Erweiterung von Java 2D
  - Ausgefeilte, hochleistungsfähige Bildbearbeitungsfunktionen
  - Folgt konsequent dem Java-Prinzip "Write once, run everywhere"
- Performance:
  - In diesem Bereich nach wie vor das Hauptproblem der Java-Plattform
  - C- und C++-Programme deutlich überlegen

Java 2D: Verwendung vordefinierter Operationen

- Beispiel: Konversion in Graustufen

```
public static BufferedImage convertToGrayscale
  (BufferedImage source) {
  BufferedImageOp op =
    new ColorConvertOp(
      ColorSpace.getInstance(ColorSpace.CS_GRAY), null);
    return op.filter(source, null);
}
```

- Operationen werden als Objekte erzeugt
  - Entwurfsmuster *"Strategy"* (Gamma et al.)
  - Ausführung:
    - » Entweder bei Übergabe an **`drawImage()`**
    - » oder durch Aufruf von **`filter()`** aus dem Objekt

Vordefinierte Operationen in Java 2D

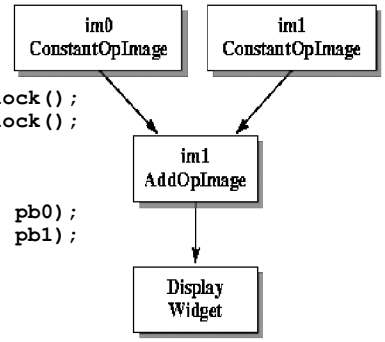
Klasse	Hilfsklassen	Effekte	"in place"? (src = dst)
`ConvolveOp`	`Kernel`	Weich- und Scharfzeichnen, Kantenerkennung	nein
`Affine TransformOp`	`java.awt.geom. AffineTransform`	Geometrische Transformationen	nein
`LookupOp`	`LookupTable, ByteLookupTable, ShortLookupTable`	Inversion, Farbtrennung, Aufhellung, Thresholding	ja
`RescaleOp`		Aufhellen, Abdunkeln	ja
`Color ConvertOp`	`java.awt.Color. ColorSpace`	Farbraum-konversion	ja

Java Advanced Imaging (JAI): Programmiermodell

- Operationen werden nicht direkt ausgeführt, sondern als Datenstruktur aufgebaut
  - Entwurfsmuster *"Command"* (Gamma et al.)
  - *Rendergraph* ist eine Baumstruktur mit den auszuführenden Operationen
  - *Parameterblöcke* werden in Rendergraph eingefügt

- Beispiel:

```
ParameterBlock pb0 = new ParameterBlock();
ParameterBlock pb1 = new ParameterBlock();
... // hier werden in pbX Daten
... // zur Manipulationen
... // für die Bilder geschrieben
RenderedOp im0 = JAI.create("const", pb0);
RenderedOp im1 = JAI.create("const", pb1);
im1 = JAI.create("add", im0, im1);
```

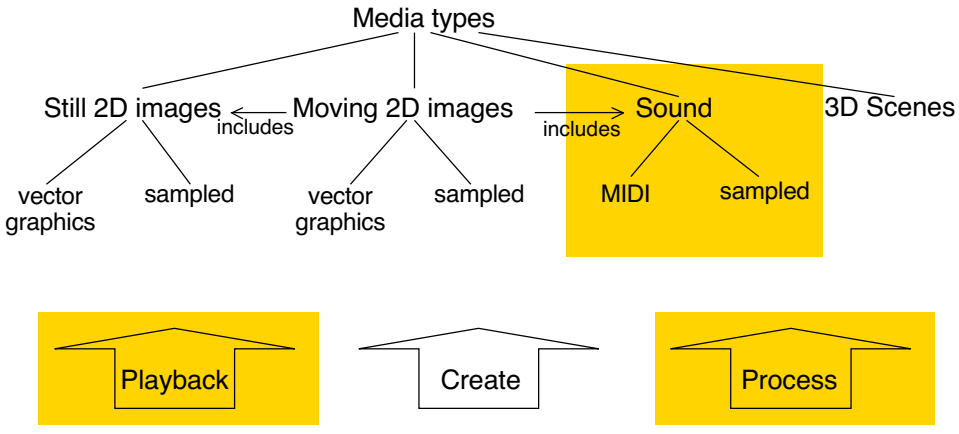


4 Fundamental Issues in Multimedia Programming

Literature:

- http://java.sun.com/products/java-media/sound/
- http://www.jsresources.org
- http://www.tritonus.org

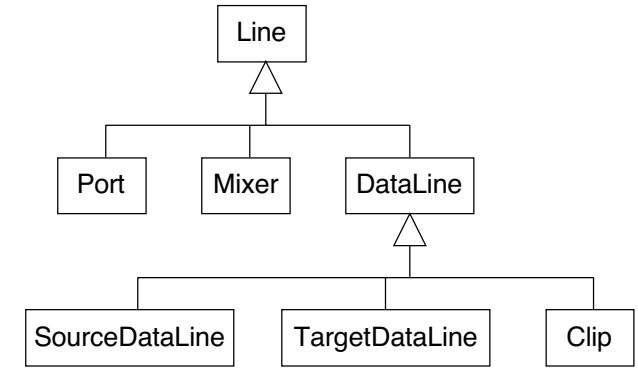
Sound Processing Programs



Java Sound API

- "Low-level" API
  - zur Steuerung der Ein- und Ausgabe von Tonmedien
  - umfasst Funktionen für digitale Audioinformation und für MIDI-Daten
  - erweiterbare Basis, keine ausgefeilten Editor-Funktionen o.ä.
- Verwandte Java-Technologien:
  - Java Media Framework (JMF)
    - » auf höherer Ebene angesiedelt
    - » einfachere Lösung für Abspielen von Tonmedien
    - » Synchronisation mit anderen Medien (v.a. Video)
- Pakete des Java Sound APIs (in Standard-Java-Installation enthalten):
  - **`javax.sound.sampled`**
  - **`javax.sound.midi`**

Line-Klassenhierarchie



Gepufferte Wiedergabe einer Audioquelle

Gepufferte Aufnahme einer Audioquelle

Wiedergabe einer Audioquelle, die komplett in den Speicher vorgeladen wird

Beispiel: Öffnen einer Audio-Line zur Wiedergabe

```
public static void main(String[] args) {
  String strFilename = args[0];
  File soundFile = new File(strFilename);
  AudioInputStream audioInputStream = null;
  try {
    audioInputStream =
      AudioSystem.getAudioInputStream(soundFile);
  }
  catch (Exception e) {};
  AudioFormat audioFormat = audioInputStream.getFormat();
  SourceDataLine line = null;
  DataLine.Info info =
    new DataLine.Info(SourceDataLine.class, audioFormat);
  try {
    line = (SourceDataLine) AudioSystem.getLine(info);
    line.open(audioFormat);
  }
  catch (Exception e) {};
  line.start();
  ...
}
```

Ressourcenabfrage
Ressourcenreservierung

Beispiel: Audiowiedergabe aus Datei

```
...
  int nBytesRead = 0;
  byte[] abData = new byte[EXTERNAL_BUFFER_SIZE];
  while (nBytesRead != -1) {
    try {
      nBytesRead =
        audioInputStream.read(abData, 0, abData.length);
    }
    catch (Exception e) {};
    if (nBytesRead >= 0) {
      int nBytesWritten = line.write(abData, 0, nBytesRead);
    }
  }
...
```

Beispiel: Direkte Bearbeitung von Samples

```
public class SineOscillator extends AudioInputStream {

  public SineOscillator
      (float fSignalFrequency, float fAmplitude,
      AudioFormat audioFormat, long lLength) {
    super(new ByteArrayInputStream(new byte[0]),
      new AudioFormat(AudioFormat.Encoding.PCM_SIGNED,...), lLength);
    ...
    m_abData = new byte[nBufferLength];
    for (int nFrame = 0; nFrame < nPeriodLengthInFrames; nFrame++) {
      float   fPeriodPosition =
        (float) nFrame / (float) nPeriodLengthInFrames;
      float   fValue =
        (float) Math.sin(fPeriodPosition * 2.0 * Math.PI);
      int nValue = Math.round(fValue * fAmplitude);
      int nBaseAddr = (nFrame) * getFormat().getFrameSize();
      m_abData[nBaseAddr + 0] = (byte) (nValue & 0xFF);
      m_abData[nBaseAddr + 1] = (byte) ((nValue >>> 8) & 0xFF);
      m_abData[nBaseAddr + 2] = (byte) (nValue & 0xFF);
      m_abData[nBaseAddr + 3] = (byte) ((nValue >>> 8) & 0xFF);
    }
    m_nBufferPosition = 0;
  }
```


MIDI-Synthesizer: Abspielen einer Note

```
import javax.sound.midi.*;

public class SynthNote1 {

  public static void main(String[] args) {
    int     nNoteNumber = 0;        // MIDI key number
    int     nVelocity = 0;
    int     nDuration = 0;
    ...
    Synthesizer     synth = null;
    try {
      synth = MidiSystem.getSynthesizer();
      synth.open();
    }
    catch (Exception e) {};
    MidiChannel[] channels = synth.getChannels();
    channels[0].noteOn(nNoteNumber, nVelocity);
    try {
      Thread.sleep(nDuration);
    }
    catch (InterruptedException e) {}
    channels[0].noteOff(nNoteNumber);
    System.exit(0);
  }
}
```

4 Fundamental Issues in Multimedia Programming

4.1 Multimedia Programming in Context

4.2 History of Multimedia Programming

4.3 A Radically Alternative Approach: Squeak

4.4 The Programmers' Way: Multimedia Frameworks for Java

- Java 2D + Advanced Imaging
- Java Sound
- Java Media Framework

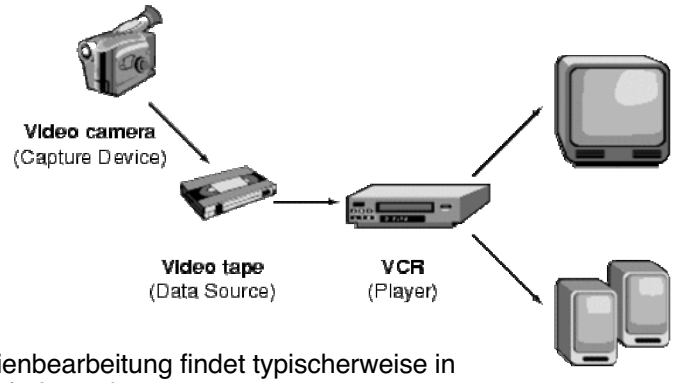
4.5 Trends and Visions

Literatur:

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

http://www.jmfapi.org

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

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)

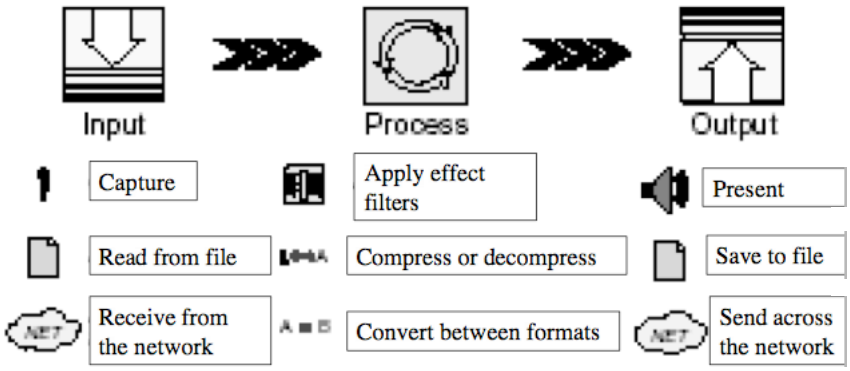
Klassifikation von Verarbeitungsketten (2)

- 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

Java Media Framework

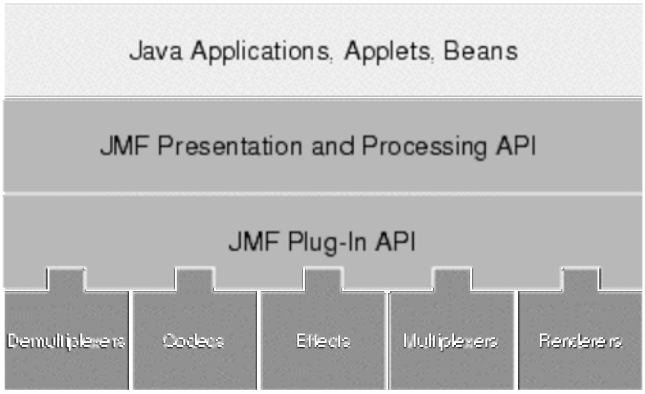
- Derzeit das einzige kostenlose, plattformunabhängige und objektorientierte Medien-Framework
- Hauptfunktionen:
  - Abspielen von Medien
  - Verarbeitung von Mediendaten in Echtzeit
  - Erfassung von Datenströmen
  - Speichern von Mediendaten
  - Strombasierte Übertragung *(streaming)* von Mediendaten
- 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“

Verarbeitungsketten-Modell in JMF



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

High-Level-Architektur von JMF



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

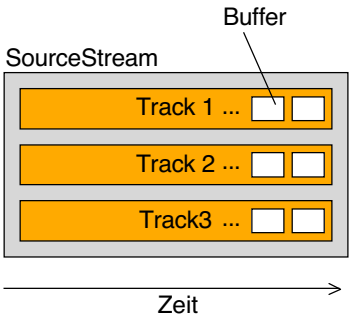
Unterstützte Dateiformate

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

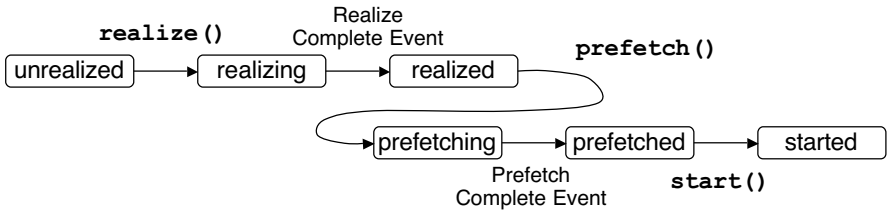
Videoformate	Audioformate
AVI	AIFF
MPEG	GSM
Quicktime	WAVE
	Sun Audio
	MIDI
	RMF

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, ...`



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`**

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: Trivialer Audio/Video-Player (1)

```
import javax.media.*;

class SimplePlayerFrame extends JFrame {

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

  public SimplePlayerFrame(String file) {
    setTitle("Simple JMF Player");
    ...
    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);
    }
  }
...
```

Beispiel: Trivialer Audio/Video-Player (2)

```
...
  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]);
  }
}
```


Streaming in JMF

- Datenquellen und -senken mit Netzübertragung
  - **`RTPManager`**
  - Standardmässige Unterstützung für RTP und RTSP
  - Unterstützung für Datenformate MPEG-1 und H.263
    - » Hohe Plattformunabängigkeit
    - » Realisierung von Diensten wie *Video on Demand* oder *Video/AudioConference*
- Nutzung des Java-Ereignisdelegations-Modells für
  - Ereignisse beim Sitzungsaufbau
  - Änderung der aktuellen Bedingungen
- Standard-Programm zum Senden und Empfangen von Medienströmen:
  - **`JMStudio`**

Java Mobile Media API

- Abgespeckte und vereinfachte Version des JMF
- 2001 für die Java2 Micro Edition entwickelt
- Seit 2002 in einzelnen mobilen Geräten unterstützt
  - Z.B. Nokia-Mobiltelefone
- Unterstützt z.B.:
  - **`Manager, DataSource, Control, Player,`**
  - Aber z.B. kein **`MediaLocator`**
  - Streaming-Funktionen
- Zusätzlich:
  - MIDI-Tonerzeugung
- Möglicherweise die wichtigste Entwicklung zur Verankerung von JMF als Basistechnologie
  - Z.B. UMTS-Mobilgeräte

Alternativen zum JMF

- OpenML (Khronos Group = SDLabs, SGI u.a.)
  - Hardwarenahes C-API
  - Als Äquivalent zum OpenGL-Grafikstandard intendiert
  - Implementierungen (noch nicht existent) sollen sehr schnell werden
  - 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.

Summary on Java Media APIs

- Main application areas:
  - Creation of media creation and editing software
  - Not targeted for individual creation of multimedia applications
- Architectural principles:
  - Processing chains
  - Prefabricated components for dealing with complex media types (e.g. video)
  - Realized by various software design patterns
    - » Strategy objects encapsulating e.g. a single filter function
    - » Pipeline architectures
    - » Event handling for synchronisation
- Programming style:
  - Low-level, rather tedious, many technical details
- Expressive power:
  - Very high power when using very low level description (e.g. sound synthesis)
  - Limited power when using pre-fabricated media-processing components

4 Fundamental Issues in Multimedia Programming

Trend: Steady Increase of Multimedia Development

- Presentations: Multimedia usage steadily increasing
- Web sites:
  - Presentations required to differ from competitors, elegance
  - Work environment replacing desktop
- Mobility:
  - WLAN, UMTS enable powerful interactive applications for small portable devices
  - User interface required to be simple and independent of keyboard input
- Innovative user interfaces:
  - E.g. VR and AR partially based on multimedia technology
- Visualization of results of complex measurements, simulations, etc.

- There is no “multimedia revolution” but multimedia elements are slowly entering many traditional areas of computing

Trend: Increasing Level of Abstraction in Programming

- Machine language, assembler, high-level programming languages
  - What is the next step?
- Authoring tools?
  - Program development environments are similar to “GUI authoring tools”
  - Pure multimedia authoring tools (e.g. Flash) include compiler, debugger etc.
- Component systems and frameworks?
  - Programming e.g. in JMF is mainly “wiring” between prefabricated components
  - “Authoring tool” for processing chains still missing

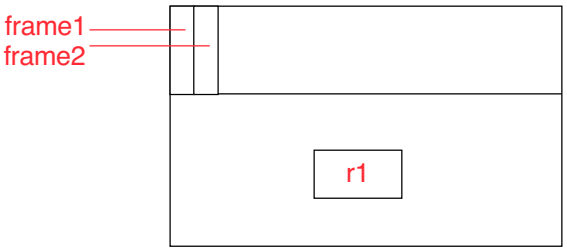
Various Representations of Same Concept

```
<layout>
  <region id="r1" ...>
</layout>
<body>
  <seq>
      ...frame1
      ...frame2
  </seq>
</body>
```

XML

```
Component r1 = ...;
Animation frame1 = ...;
Animation frame2 = ...;
Animation all =
  Animations.sequential(
    new Animation[]{
      frame1, frame2});
```

Java



Authoring Tool (Flash-like)

Visions: Provocative Questions

- What is special about multimedia programming?
  - Are there special language concepts?
  - Can multimedia make programming simpler (cf. the Squeak/EToys idea)?
- Will a future multimedia development tool still provide support for a classical, text-based programming language?
  - Is there a way for fully graphic “programming”?
  - If yes, will it be really helpful?
- Will new paradigms supersede the object-oriented one?
  - E.g. “aspects”?
  - Is there a better, more abstract replacement for event handling?
- Which role will be played by abstract models of the underlying application and of the user interaction itself?
  - Will it ever be possible to develop a multimedia application in a platform-independent way?