

1 Worum geht's?

Im Folgenden zoomen wir von der Welt der unbegrenzten Möglichkeiten zum begrenzten Ausschnitt der Medienverarbeitung mit dem Java Media Framework und dem Mobile Media API. Zunächst streifen wir kurz die allgemeinen Grundlagen der Medienverarbeitung (Was ist ein Medium? Welchem Zweck dienen Medien? Was kann man mit Medien anstellen? Warum ist das faszinierend?), kommen von dort zur Bearbeitung elektronischer Medien mit Java und beschäftigen uns dann eingehend mit dem Java Media Framework. Den Abschluss bildet ein Ausblick auf mögliche zukünftige Entwicklungen in diesem Kontext.

1.1 Medien, Information, Kommunikation

Was ist ein Medium? Ein Medium ist alles, was in unserer vierdimensionalen Lebenswelt existiert. Zum Beispiel ein Stuhl, dessen räumliche Erscheinung in allen seismatischen Varianten sehr markant ist und der im Zeitraum seiner Existenz zu jedem Zeitpunkt existiert. Das ist der Medienbegriff, wie ihn der Pionier der Medientheorie Marshall McLuhan geprägt hat [McLuhan64]. Er spricht von Medien als jede Erweiterung unseres Körpers. Das schließt den Stuhl mit ein. Besonders interessant sind für ihn elektronische Medien, die er Erweiterungen des Nervensystems nennt. Elektronische Medien sind zum Beispiel das Telefon, Fernsehen, der Computer und das World Wide Web.

McLuhan (von ihm stammt der Satz: »Das Medium ist die Botschaft«) hat sich einige sehr sinnvolle Kategorisierungen für Medien ausgedacht. Für uns relevant ist seine Erkenntnis, dass jedes Medium andere Medien enthalten kann und normalerweise wird. Umgekehrt wird ein Medium wiederum von einem anderen Medium umschlossen werden. Das trifft recht gut zu: Das Medium Web enthält eine Menge audiovisueller Medien (Videos, Musikdateien etc.), die erneut aus

Was ist ein Medium?

Medien enthalten Medien

mehreren Einzelmedien bestehen können (etwa Bildspuren und Tonspuren) und geht seinerseits wiederum vollständig im Meta-Medium (oder: der Idee) Computer auf.

Komposition von Medien

Ein ähnliches, aber nicht zu verwechselndes Konzept bezeichnet der Begriff Komposition. Darunter versteht man, dass sich jedes Medium aus einer Folge oder Hierarchie von Versatzstücken zusammensetzt. Komposition ist ein ganz wesentlicher Aspekt der Gestaltung elektronischer Medien (mehr dazu im Abschnitt 1.2). Die Versatzstücke sind selbstverständlich nichts anderes als wiederum Medien. Medien *beinhalten* also Medien (hierarchische Beziehung) und *setzen sich aus* Medien *zusammen* (sequentielle Beziehung).

*Räumliche und zeitliche
Komposition*

Medien können räumlich und zeitlich komponiert (oder: zusammengeklebt) werden. Räumlich zusammenkleben kann man etwa mehrere Bilder zu einem breiten Panoramabild oder indem man ein Video im Fenster eines anderen Videos einblendet, was zum Beispiel in Nachrichtensendungen gern gemacht wird. Zeitlich verkleben kann man beispielsweise mehrere Szenen zu einem Film. Zeitliche Komposition (*Synchronisation*) ist die zeitliche Relation mehrerer Medien. Zwei Videos etwa können so synchronisiert sein, dass sie zur selben Zeit beginnen, gleich lange dauern oder lippensynchron mit einem gesprochenen Text ablaufen. Aus dem Blickwinkel der Komposition kann man sagen, dass Medien so charakterisiert werden können, dass sie »etwas« sind, das zeitlich und räumlich gestaltbar ist. Das heißt, dass sie in allen vier Dimensionen existieren und sich ausgehend von einem Ursprung zeitlich und räumlich ausbreiten (lassen).

Synchronisation

Zweck von Medien

Das bringt uns zur Antwort auf folgende zentrale Frage: *Welchem Zweck dienen Medien?* Der Information oder Kommunikation. Unter Information (Definitionen für diesen Begriff gibt es wie Sand am Meer und einige kommen von mächtig bedeutenden Leuten) wollen wir das Weitergeben von Daten von einem A an einen B verstehen, das heißt einen *gerichteten Datenfluss*. A heißt gemeinhin die Datenquelle und B die Datensenke. Während »Datenfluss« meist den gesamten Prozess der Information bezeichnet, versteht man unter »Datenstrom« die tatsächlich ausgetauschten Daten. In manchen Definitionen wird auch noch verlangt, dass diese Daten nicht beliebig sind und/oder einen – nicht näher spezifizierten – Neuigkeitswert haben. Kommunikation ist, wenn A und B sich wechselseitig informieren und die ausgetauschten Informationen in einem zeitlichen und/oder inhaltlichen Bezug zueinander stehen. Was hat das mit uns zu tun? Die ausgetauschten Daten sind genau genommen immer Medien. Sie haben keine Möglichkeit, etwas anderes zu sein. Daraus folgt, dass A und B, während sie informieren oder kommunizieren, Medienverarbeitung betreiben.

Das bringt uns zur dritten Frage in diesem hinführenden Abschnitt: *Wie ist eigentlich das Verhältnis von Daten zu Medien?* (Diese Frage ist relevant, weil die Datentheorie einer der ausgereiftesten und bekanntesten Bereiche der Informatik ist und daher angenommen werden kann, dass jeder Leser in ihr bewandert ist.) Daten sind das letzte, sozusagen reine Medium. Weder die räumliche noch die zeitliche Dimension des Mediums sind bei Daten noch besonders gut sichtbar. Die Zeitabhängigkeit erkennt man noch an der wechselnden Aktualität von Daten (weshalb es unter anderem notwendig ist, Softwaresysteme zu ihrer Pflege zu entwickeln), die räumliche Dimension aber nur noch durch eine begriffliche Verkettung über viele Metaebenen. Daten sind also das letzte Medium, das keine anderen Medien mehr enthält.

Verhältnis von Daten zu Medien

Warum aber will man in Informations- und Kommunikationssystemen nicht nur Daten, sondern bunte, tönende, die Sinne ansprechende Medien austauschen? Dafür scheint es zwei wesentliche Gründe zu geben.

1. Wir sind das als Menschen durch die Verwendung unserer Sinne zur Kommunikation so gewohnt und übernehmen diese Gewohnheit für die Kommunikation mit elektronischen Medien.
2. Wir wollen mehr als nur Daten kommunizieren: wir wollen beeinflussen. Dass das Medium die Botschaft ist, wird oft als grundsätzlich richtig, aber übertrieben beurteilt. Überlegt man jedoch genau und berücksichtigt die oben angeführten Tatsachen, dann ergibt sich, dass es *exakt* so ist. Das Medium (die enthaltenen Daten, das gewünschte Drumherum) ist die Botschaft, der Datenstrom.

So viel zum Medium an sich. Im nächsten Abschnitt wollen wir uns die Grundlagen der Verarbeitung *elektronischer* Medien ansehen. In der Folge sind, wenn von »Medien« gesprochen wird, immer solche gemeint.

1.2 Medienverarbeitung

Die Verarbeitung elektronischer Medien lässt sich in drei Bereiche einteilen:

- Kodierung
- Komposition
- Verpackung und Transport

- Kodierung* Zum Bereich der Kodierung gehören die Probleme, die sich aus den großen Datenmengen ergeben, die bei elektronischen Medien üblicherweise anfallen. Dazu gehört die Beseitigung von Redundanz, um die Datenmengen zu verkleinern und das Einflechten von Redundanz, um die Qualität der Darstellung sowie die Sicherheit des Transportes zu erhöhen. Wir werden uns mit Kodierung detailliert in den Abschnitten 3.4 und 4.2 beschäftigen.
- Komposition* Komposition bezeichnet – wie bereits oben skizziert – das zeitliche und räumliche Zusammenkleben von Medien. Dazu gehören die Bereiche Synchronisation von Teilmedien und die Anwendung von Effekten. Damit beschäftigen wir uns insbesondere in den Abschnitten 3.2 und 4.3.
- Verpackung und Transport* Mit Verpackung und Transport ist jener Bereich gemeint, der sich mit dem Zusammenfassen mehrerer, voneinander unabhängiger Medien (so genanntes Multiplexing) zum gemeinsamen Transport (Streaming) beschäftigt. Das ist Thema der Abschnitte 4.4, 4.5 und 7.
- Die Medienverarbeitung kann nun auf zwei Arten erfolgen:
- deklarativ
 - prozedural
- Deklarative Medienverarbeitung* Bei der deklarativen Medienverarbeitung wird angegeben, welche Medienobjekte benutzt werden sollen und wie das Endergebnis (wiederum ein Medium) aussehen soll. Es wird also definiert, *was* getan werden soll, aber nicht, *wie* es getan werden soll. Dazu werden üblicherweise Beschreibungssprachen benutzt. Eine der bekanntesten und aktuellsten ist SMIL (Synchronized Multimedia Integration Language), das – abgeleitet von XML – die Deklaration auf elektronischen Medien basierender Kommunikation zwischen einem Benutzer und einem Computer (beziehungsweise seiner Datenbasis) ermöglicht. Für das *Wie* ist bei SMIL ein so genannter User Agent zuständig. Eine typische SMIL-Beschreibung könnte zum Beispiel folgendermaßen aussehen:
- ```
<smil>
 <body>
 <par>
 <seq>
 <video src="film.mpg"/>
 <text src="nachspann.html"/>
 </seq>
 <audio src="hintergrund.mp3"/>
 </par>
 </body>
</smil>
```

Diese Beschreibung teilt dem User Agent mit, dass er dem Benutzer nacheinander (seq steht für sequentiell) das Video `film.mpg` und die Webseite `nachspann.html` zeigen soll. Während er das tut (par steht für parallel), wird die Musik `hintergrund.mp3` abgespielt. Das ist ein sehr einfaches Beispiel, der User Agent muss für seine Umsetzung aber bereits einiges können.

Um einen solchen User Agent implementieren zu können, werden die Methoden der prozeduralen Medienverarbeitung benötigt. Dabei wird festgelegt, *wie* die Medienverarbeitung erfolgt. Welche Medien verwendet werden und welche erzeugt werden, wird aber nur implizit definiert. Zu den Methoden der prozeduralen Medienverarbeitung gehören Verarbeitungsketten, Formate etc. Sie sind der Hauptinhalt dieses Buches. Wenn wir also in der Folge von Medienverarbeitung sprechen, meinen wir eigentlich die *prozedurale Verarbeitung digitaler Medien*. Im nächsten Abschnitt sehen wir uns an, welche Möglichkeiten es dafür in Java gibt. Dabei wird nicht nur auf die zeitabhängigen, sondern alle elektronischen Medien eingegangen.

Prozedurale  
Medienverarbeitung

### 1.3 Die Java Media APIs

Sun (und andere Anbieter) stellen im Rahmen des Java SDK und als Zusatzkomponenten unter anderem APIs für folgende Medientypen zur Verfügung (zusammengefasst als »Java Media APIs«:

- XML Text: Java API for XML Processing (JAXP) etc.
- 2D-Grafik: Java2D, Java Advanced Imaging (JAI)
- 3D-Grafik: Java3D, GL4Java
- Video: Java Media Framework, Mobile Media API (MMAPI)
- Audio: Java Media Framework, JavaSound, JavaSpeech, MMAPi

Als Business-Umgebung bietet Java (im Paket JAXP) natürlich ausgefeilte Methoden zur XML-Verarbeitung an. Verarbeitung umfasst das Lesen ( Parsen) und das Manipulieren/Schreiben von XML-Dateien. Zum Parsen gibt es grundsätzlich zwei Ansätze:

XML-Verarbeitung

- sequentielles Lesen und Auswerten des XML-Dokumentes
- Interpretation des Dokumentes als Baumstruktur

Ersterer Ansatz wird im Simple API for XML Parsing (SAX) implementiert. Dabei wird jede XML-Datei zeilenweise durchgearbeitet und für jedes gefundene Element (drei Arten: Start, Ende, Text) eine so genannte Handler-Funktion aufgerufen. Diese führt eine spezifische

SAX

Verarbeitung durch. Ein einfacher SAX-Parser könnte folgendermaßen erzeugt werden:

```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();
InputSource is = new InputSource(<FILE>);
parser.parse(is, new SaxHandler());
```

In den ersten beiden Zeilen wird ein Parser erzeugt. Daraus ersieht man, dass JAXP das Factory-Entwurfsmuster implementiert [Gamma et al. 00]. Das heißt, der Programmierer weiß letztlich nicht, welcher Parser erzeugt wird. Wichtig ist nur, dass das API feststeht. In der dritten Zeile wird eine Datenquelle erzeugt und in der vierten das Parsen gestartet. Werden Start-, Text- oder Endelemente gefunden, werden Methoden von SAX-Parser aufgerufen:

```
public class SaxHandler extends DefaultHandler {
 public void characters(...) {}
 public void startElement(...) {}
 public void endElement(...) {}
}
```

Daraus wird ersichtlich, dass sich der Programmierer um die Speicherung der XML-Daten selbst kümmern muss. Der Vorteil von SAX ist, dass das API klein und leicht zu benutzen ist. Der wesentliche Nachteil ist, dass keine Prüfung der Korrektheit des XML-Dokuments erfolgt.

DOM

Dies ist möglich mit dem Document Object Model Parser (DOM). Dieser analysiert ein XML-Dokument als Ganzes und erzeugt eine Struktur im Speicher. In JAXP kann ein DOM-Parser folgendermaßen erzeugt werden:

```
DocumentBuilderFactory factory =
 DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
InputSource is = new InputSource(<FILE>);
Document document = builder.parse(is);
```

Der Aufruf von `parse` führt dazu, dass die gesamte Datei ausgewertet wird und ausgehend vom Wurzelknoten `document` ein Baum im Speicher erzeugt wird. Auf die einzelnen Elemente dieses Baumes kann folgendermaßen zugegriffen werden:

```
NodeList smil = document.getElementsByTagName("smil");
Node element = smil.item(0);
Node child = element.getFirstChild();
String text = child.getNodeValue();
```

Zunächst werden alle Knoten mit dem Namen »smil« gelesen. Von diesen wird das erste genommen. Vom ersten Kind wird dann der Wert

gelesen. Wenn es sich dabei um einen Text handelt, wird er in text abgelegt. Die wesentlichen Vorteile von DOM sind also, dass ein Dokument als Ganzes im Speicher abgelegt wird, dort manipuliert werden kann und in der Folge mit einem einzigen Methodenaufruf wieder gespeichert werden kann. Der Nachteil von DOM ist die komplexere Implementierung, die mehr Ressourcen als SAX benötigt. Soviele zur XML-Verarbeitung in Java. Mehr zu SAX und DOM gibt es in [JAXP].

Zur zweidimensionalen Bildverarbeitung gibt es die Pakete Java2D und Java Advanced Imaging (JAI). Während Java2D im Java SDK enthalten ist und einfache Methoden zum Zeichnen enthält, ist JAI optional und bietet mächtige Funktionen zur Bilderzeugung und vor allem Bildanalyse an. Java2D lässt sich auch sehr gut zur Verarbeitung von Videos einsetzen. Ein Beispiel dazu findet sich am Ende von Abschnitt 4.3 (Seite 86).

*2D-Grafik: Java2D und JAI*

Für 3D-Grafik gibt es zur Zeit zwei leistungsstarke Implementierungsmöglichkeiten: Java3D und GL4Java. Das optionale Paket Java3D zeichnet sich durch komfortable Methoden zum Erstellen von Modellen mit Interaktion aus (Szenengraphen etc.), hat aber leider auch schwerwiegende Nachteile: Es unterstützt viele neue Grafikkarten nicht, ist daher schwierig zu installieren und hat eine vergleichsweise schlechte Performance. Außerdem unterliegt man beim Erstellen von Szenengraphen Einschränkungen, sodass zum Beispiel einer aktiven (kompilierten) Szene keine weiteren Elemente hinzugefügt werden können. GL4Java andererseits ist ein Paket, das nicht von Sun zur Verfügung gestellt wird, sondern von der Firma Jausoft entwickelt wurde. GL4Java ist im Wesentlichen ein Java Wrapper für OpenGL. OpenGL ist der De-facto-Standard für 3D-Grafik. Da es sich jedoch um ein schlichtes C-API handelt, ist es mit GL4Java möglich, die umfassenden Möglichkeiten und die natürlich besonders gute Performance von OpenGL in einem objektorientierten Umfeld zu nutzen. Dass die Zukunft eher GL4Java als Java3D gehören wird, ist unter anderem auch daran ersichtlich, dass Sun mittlerweile selbst GL4Java bewirbt. Mehr zu GL4Java gibt es in [GL4Java].

*3D-Grafik: Java3D und GL4Java*

Zur Verarbeitung von Audio gibt es im Java SDK schon seit längerem JavaSound. JavaSound stellt Methoden zum Lesen, Verarbeiten und Schreiben von Audiodateien zur Verfügung. Audiodateien können dabei entweder in Wave-Format oder als MIDI-Dateien vorliegen. JavaSound ist ein direkter Konkurrent des in diesem Buch behandelten Java Media Frameworks. Im Vergleich zum Java Media Framework hat JavaSound aber folgende Nachteile:

*JavaSound*

- weniger unterstützte Audioformate
- ein für Audioverarbeitung spezifisches API
- sehr beschränkte Möglichkeiten zum Erfassen und Streamen von Mediendateien

Allerdings kann JavaSound – im Gegensatz zum Java Media Framework – MIDI-Daten nicht nur lesen, sondern auch schreiben. Übrigens verwendet das Java Media Framework JavaSound für die Tonausgabe. Ein Beispiel für JavaSound gibt es im Abschnitt 2.3.

*JavaSpeech*

Sprachverarbeitung kann in Java basierend auf JavaSpeech implementiert werden. Dabei handelt es sich um ein API, das von IBM und Sun im VoiceXML-Server implementiert wurde. Damit ist es möglich, Sprache zu erfassen und zu analysieren sowie Sprache synthetisch zu erzeugen [JavaSpeech]. Die Steuerung erfolgt über standardisierte XML-Dokumente. Die Grundlage der Sprachverarbeitung im VoiceXML-Server ist übrigens nicht JavaSound, sondern das Java Media Framework (JMF).

*Java Media Framework*

Das JMF schließlich ist ein Plattform-unabhängiges API zur Verarbeitung audiovisueller Medien in Java und das zentrale Thema dieses Buches. Eine abgespeckte Version des JMF gibt es auch für mobile Geräte: das Mobile Media API (MMAPI). Die Struktur des MMAPI entspricht weitgehend der des JMF. Der nächste Abschnitt bietet einen allgemeinen Überblick über das JMF.

*MMAPI*

## 1.4 Das Java Media Framework

Das Java Media Framework (JMF) von Sun ist derzeit das einzige kostenlose, Plattform-unabhängige und objektorientierte Medienverarbeitungs-Framework. Das JMF ist eine Erweiterung zum Java SDK zur Verarbeitung multimedialer zeitabhängiger Daten (im Wesentlichen Audio und Video). Es umfasst folgende Hauptfunktionen:

- Abspielen von Medien
- Verarbeitung von Mediendaten in Echtzeit
- Erfassung von Datenströmen
- Speichern von Mediendaten
- Streaming von Mediendaten

Der Quellcode des JMF ist frei verfügbar. Optimierte Implementierungen existieren für Windows, Linux, MacOS-X und Solaris sowie (in abgespekter Form) als Java-Package ohne plattformabhängige Bibliotheken.

Das JMF kann eine Vielzahl von Datenformaten lesen und schreiben. Dazu gehören die Audioformate MP3, GSM, G.723, ULAW (steht für  $\mu$ -Law), DVI sowie die Videoformate MPEG-1 (nur lesen), MJPEG, H.263, Shockwave (nur lesen) und MPEG-4 (nur lesen; implementiert von IBM [MPEG-4Codec]). An Dateiformaten unterstützt das JMF unter anderem AVI, WAV (Microsoft Windows), MOV (QuickTime), AU (Solaris) und MPEG. Diese Formate werden im Abschnitt 3.4 beschrieben.

*Viele unterstützte Audio- und Videoformate*

Die erste JMF-Version wurde von Sun, SGI und Intel entwickelt und 1998 freigegeben. Version 1.0 beschränkte sich auf das Abspielen von Mediendateien und war verfügbar für Windows und Solaris. In Version 1.1 wurde eine reine Java-Version hinzugefügt. Für Version 2.0 (1999), von Sun und IBM gemeinsam entwickelt, wurde das API neu gestaltet und Funktionen für Medienerfassung und -verarbeitung hinzugefügt. JMF 2.0 war wiederum für Windows, Solaris und als Java-Version verfügbar. Mit Version 2.1 kam erstmals eine Linux-Variante von Blackdown hinzu. Außerdem wurden erstmals RTP-Streaming sowie die Streaming Server von Apple und Sun unterstützt (mehr zu Streaming: s. u.). In Version 2.1.1 schließlich (2001) wurde das RTP-API verbessert und das Videoformat H.263 eingeführt. Dadurch kann JMF mit dem Darwin Streaming Server zusammenarbeiten und als RTSP-Client agieren.

Das MMAPI wurde – basierend auf dem JMF – von Sun im Jahr 2001 für die Java2 Micro Edition entwickelt. Seit Sommer 2002 gibt es eine fertige Spezifikation und erste mobile Geräte, die das MMAPI implementieren (unter anderem Mobiltelefone von Nokia, siehe [MMAPI]). Das MMAPI kann Mediendateien abspielen, erfassen und von einem Streaming Server lesen. Es bietet also nur einen Teil der Funktionalität des JMF an. Daher gibt es in diesem Buch kein eigenes Kapitel zum MMAPI, sondern an den entsprechenden Stellen Absätze mit Hinweisen, wie das jeweilige Konzept im MMAPI realisiert ist. Außerdem gibt es im Anhang einen dokumentierten Pfad durch alle Hinweise zum MMAPI in diesem Buch.

*MMAPI*

Während das MMAPI in seinem Bereich zur Zeit noch einzigartig ist, gibt es für das JMF sehr wohl Alternativen. Diese werden im folgenden Abschnitt kurz besprochen.

## 1.5 Alternativen zum JMF

Derzeit gibt es im Wesentlichen drei Alternativen zum JMF:

- OpenML von der Khronos Group

- DirectX / DirectShow von Microsoft
- QuickTime (for Java) von Apple

*OpenML* OpenML [OpenML] ist in der Spezifikation 1.0 von 2001 ein hardwarenahes C-API für Medienverarbeitungshardware und -software. Darin liegt gleichzeitig sein größter Vorteil und sein größter Nachteil: durch die Hardwarenähe werden OpenML-Implementierungen sehr schnell sein (derzeit existiert noch keine), andererseits ist ein C-API ungeeignet für moderne Software-Entwicklung. Zur Khronos Group gehören viele wichtige Hardwarehersteller (3DLabs, SGI etc.) und Softwarehersteller (unter anderem auch Sun). OpenML soll – ähnlich wie OpenGL für Computergrafik – ein Industriestandard für Basisfunktionalitäten von Medienverarbeitungs-Komponenten werden. Positiv an OpenML sind die professionelle Architektur (wesentlich basierend auf den Erfahrungen und Softwarebibliotheken von SGI), die Integration von OpenGL sowie die Unterstützung professioneller Datenformate. Außerdem ist die OpenML-Spezifikation frei verfügbar. Allerdings ist anzunehmen, dass OpenML-Implementierungen nicht frei erhältlich und hardwareabhängig sein werden. Die Entwicklung einer reinen OpenML-Softwarebibliothek für alle gängigen Plattformen ist derzeit noch nicht abzusehen.

Streng genommen gehört OpenML aber gar nicht zu den in unserem Kontext interessanten Frameworks. Es ist nicht erweiterbar und bietet keine Möglichkeiten für Streaming etc. Vielmehr kann es als technische Grundlage von Medienverarbeitungs-Frameworks angesehen werden. So ist es durchaus vorstellbar (und Aussagen von Vertretern der Khronos Group unterstützen diese Idee), dass das JMF in Zukunft bei gleichbleibendem API auf OpenML basiert wird (eine ähnliche Vorgehensweise wie bei Java3D und OpenGL).

*DirectShow* DirectShow ist seit der Version 8.0 Teil von DirectX [DirectShow]. Dabei handelt es sich um eine wesentlich erweiterte und verbesserte Version des früheren Video for Windows (VfW). DirectShow ist ein echtes Medienverarbeitungs-Framework: es enthält Methoden zur Erfassung, Verarbeitung und zum Transport von Medien. Die Architektur ist durchdacht: jede Komponente der Verarbeitung ist ein Filter. Es gibt drei Arten: *Source Filter* zum Lesen von Mediendaten, *Transform Filter* zum Verarbeiten von Mediendaten und *Rendering Filter* zum Ausgeben von Daten. Die Kommunikation zwischen Filtern erfolgt über so genannte *Pins*.

Die wesentlichen Vorteile von DirectShow sind gute Performance, die Unterstützung modernster Datenformate (MPEG-4 etc.) sowie die Unterstützung durch eine Vielzahl von Geräteherstellern, die ihre

Hardware mit DirectShow-Treibern liefern. Außerdem enthält DirectShow mit dem »Graphbuilder« ein Werkzeug für die interaktive Programmierung von Medienverarbeitungs-Systemen sowie viele Effektklassen für Standardverfahren der Medienverarbeitung (Schnitte etc.).

Diesen Vorteilen stehen aber auch eine Reihe von Nachteilen gegenüber. DirectShow ist Windows-spezifisch. Implementierungen für andere Betriebssysteme existieren nicht. Aufgrund der Hardwarenähe von DirectX ist es auch nicht Teil der .NET-Klassenbibliothek, wird also auch in Zukunft nicht für andere Systeme zur Verfügung stehen. DirectShow basiert auf Microsofts Component Object Model (COM). Wer also mit DirectShow programmieren will, muss COM zumindest in den Grundzügen kennen. Dadurch und durch das teilweise unnötig komplizierte Softwaredesign (zum Beispiel zum Ansprechen von Pins) wird die Programmierung von Standardaufgaben sehr umständlich. Außerdem integriert DirectShow nicht die Verarbeitung von Audiodaten und bietet selbstverständlich nicht die Ein- und Ausgabeformate der Konkurrenz an (QuickTime etc.). Schließlich ist DirectShow kostenpflichtig: Für jeden Client, auf dem eine DirectShow-Anwendung läuft, muss eine Lizenzgebühr an Microsoft bezahlt werden.

QuickTime ist zweifellos der interessanteste Konkurrent des JMF. Seit 1991 bietet Apple diese renommierte Plattform für die Entwicklung von Medienverarbeitungs-Anwendungen an. Mit dem QuickTime for Java API (bei dem es sich im Wesentlichen um einen Wrapper für die nativen C-Bibliotheken handelt) ist es nun sogar möglich, QuickTime-Anwendungen mit Java zu erstellen [Quicktime]. QuickTime umfasst nicht nur Methoden zum Verarbeiten von Audio und Video, sondern auch für Bildverarbeitung, Animation, 3D-Modellierung etc. Wir wollen uns im Folgenden aber auf die Funktionen zur Verarbeitung zeitabhängiger Medien beschränken.

Die Stärken von QuickTime for Java (QfJ) sind der große Funktionsumfang sowie die bewährte Qualität der Implementierung (mit ausgezeichneter Dokumentation!). Dem stehen an Nachteilen gegenüber, dass QuickTime nicht alle gängigen Datenformate unterstützt, für jeden Client lizenziert werden muss und nur für Windows- und Mac-Betriebssysteme implementiert ist. Ein weiterer Nachteil von QfJ ist seine etwas unübersichtliche, altmodisch anmutende Architektur. Sehen wir uns dazu ein Beispiel an.

Das nachfolgende Applet spielt eine Videodatei ab.

```
public class VideoPlayer extends Applet {
 private QTCanvas canvas = null;
 private Drawable video;
```

*QuickTime (for Java)*

*Beispiel: VideoPlayer*

Zunächst werden zwei Ressourcen erzeugt: `canvas` ist der grafische Ausgabebereich für das Video. `video` ist »Drawable«, das heißt, irgend etwas, das ausgegeben werden kann. Eine Audiodatei wäre auch Drawable. In der `init`-Methode des Applets wird eine QuickTime-Session gestartet, die Videodatei geöffnet und dann `canvas` ausgegeben.

```
public void init () {
 try {
 QTSession.open();

 video = QTFactory.makeDrawable (<FILE>);
 canvas = new QTCanvas (QTCanvas.kInitialSize, 0.5F, 0F);
 add(canvas);
 } catch (Exception e1) {}
}
```

Die Parameter `QTCanvas.kInitialSize`, `0.5F` und `0F` zeigen an, dass der Ausgabebereich zu Beginn ganz oben (»`0F`«) mittig (»`0.5F`«) ausgegeben werden soll. In der `start`-Methode wird das Video dem Ausgabebereich zugewiesen und gestartet (`setClient`).

```
public void start () {
 try {
 if (canvas != null) canvas.setClient (video, true);
 } catch (QTException e2) {}
}
```

`stop` und `destroy` beenden das Abspielen der Videodatei und schließen die QuickTime-Session.

```
public void stop () {
 if (canvas != null) canvas.removeClient();
}

public void destroy () {
 QTSession.close();
}
}
```

Einige Dinge in diesem Programm sind sonderbar: Warum muss eine Session geöffnet werden? Wenn das fehlschlägt, kann QuickTime nicht benutzt werden. Warum sind die Parameter des Canvas-Konstruktors so ungewöhnlich anzugeben? Warum wird nicht überhaupt ein normales Java-Canvas benutzt? Warum heißt die entscheidende Methode zum Abspielen `setClient`? Die Antwort auf diese Fragen ist die gewachsene Struktur von QuickTime, über die ein Java API nur drübergestülpt wurde. QuickTime for Java ist deshalb kein schlechtes API: es ist umfangreich und effizient. Was die Programmierung und

die Integration mit dem Java SDK anbetrifft, ist es aber nicht auf dem neuesten Stand der Technik.

Zusammenfassend ist festzustellen, dass das JMF derzeit das einzige kostenfreie Medienverarbeitungs-Framework ist, das wirklich Plattform-unabhängig ist. Zudem existiert mit dem MMAPi ein JMF-Klon für den zukunftssträchtigen Markt der Embedded Systems (mobile Geräte etc.). Im nächsten Abschnitt werden die Zukunftschancen des JMF bewertet.

## 1.6 Ausblick: Zukunft des Java Media Frameworks

Die zentrale Frage dieses Abschnittes ist: *Hat es Sinn, sich in das JMF einzuarbeiten?* Wird es nicht ohnehin in zwei Jahren veraltet sein oder gar nicht mehr existieren? Zur Beurteilung seiner Überlebensfähigkeit stellen wir im Folgenden positive und negative Aspekte gegenüber.

Positive Aspekte des JMF sind die vorzügliche Architektur der Komponenten und das gut dokumentierte API. Außerdem ist der Quellcode des JMF frei und kann von der Sun Website heruntergeladen werden. Deshalb haben sich zahlreiche Gruppen gebildet, die das JMF für die Entwicklung von Multimedia-Anwendungen nutzen. Darüber hinaus gibt es mittlerweile eine Community Website mit Dokumentation, Tutorials, Beispielprogrammen etc. [JMFCCommunity].

*Positive Aspekte*

Wesentliche Probleme des JMF sind die teilweise schlampige Programmierung (zahlreiche Bugs erschweren die Arbeit) sowie das langsame grafische Rendering, wenn die Ausgabe in ein AWT/Swing-Panel eingebunden wird und nicht mehr der Originalgröße des Mediums entspricht. Zusätzlich ist die Programmier-Dokumentation vor allem in Bezug auf Streaming mittlerweile veraltet.

*Negative Aspekte*

Mit der Einführung des MMAPi hat sich Sun aber ganz deutlich zum JMF API bekannt. Da die ersten Geräte mit MMAPi bereits existieren, kann angenommen werden, dass Sun auch das JMF (als Mutter des MMAPi) weiter unterstützen wird. Zumal man durch das Erlernen des JMF die Programmierung mit dem MMAPi sozusagen umsonst dazubekommt. Es ist auch durchaus denkbar, dass in Zukunft das JMF neu implementiert wird (z. B. auf Basis von OpenML), das bewährte API aber unverändert bleibt.

## 1.7 Überblick über das Folgende

Im folgenden Kapitel 2 springen wir ins kalte Wasser und sehen uns an, wie ein Audioplayer implementiert werden kann: zuerst mit dem

JMF (zwei Varianten) und dann zum Vergleich mit JavaSound. In Kapitel 3 werden die einführenden Gedanken dieses Kapitels zu elektronischen Medien vertieft und erweitert. Kapitel 4 – das Herzstück des Buches – zeigt, wie die Konzepte von Kapitel 3 im JMF (und MMAPI) implementiert sind. Kapitel 5 beschäftigt sich mit den Komponenten, die Sun zum JMF mitliefert, sowie grundlegenden Algorithmen der Medienverarbeitung. Nach dem Studium der Kapitel 4 und 5 besitzt man bereits einen umfassenden Werkzeugkasten für die prozedurale Medienverarbeitung. Kapitel 6 beschäftigt sich mit der Erfassung und Kapitel 7 mit dem Transport von Medien im JMF. Kapitel 8 fasst die wesentlichen Aussagen zusammen. Im Anhang gibt es vertiefende Informationen zur Architektur des JMF sowie zur Installation und Verteilung des Frameworks. Außerdem enthält er einen Pfad, der alle Verweise auf das MMAPI in diesem Buch übersichtlich zusammenfasst.