Mobile Media API (JSR-135) & Mobile 3D Graphics (JSR-184)

Ausarbeitung im Rahmen des Proseminar Mobile Java
Universität Ulm
SS 2004

Christian Kempter
christian.kempter@informatik.uni-ulm.de

Zuständige Betreuer: Martin Gumhold, Dr. Frank Kargl

Mobile Media API (JSR-135) & Mobile 3D Graphics (JSR-184)

1. Mobile Media API (JSR-135)

1.1 Einleitung

Zum jetzigen Zeitpunkt sind die meisten Mobiltelefone mit MIDP 1.0 ausgestattet. Dieses Profil bietet eine solide Grundlage für Spiele und interaktive Anwendungen auf mobilen Geräten. Mit dem steigenden Interesse an solchen Anwendungen wurde auch das Verlangen nach Audio- und Videounterstützung größer. Die Mobile Media API, welche ich in dieser Ausarbeitung vorstelle, beschäftigt sich mit diesen Multimedia Typen. Mit ihr ist es möglich Zugang zu den zeitbasierten Multimedia Elementen wie Audio Clips, MIDI Sequenzen, Filmen, Clips sowie Animationen zu erhalten. Da viele dieser Elemente in Zukunft vor allem online verfügbar sein werden, bzw. über das Web verschickt oder empfangen werden, ist es von größter Bedeutung eine solide Webanbindung für Applikationen zu schaffen welche sich mit zeitbasiertem Multimedia beschäftigen um zum Beispiel Streaming zu ermöglichen. Die Benutzer von mobilen Geräten verlangen nach immer mehr Möglichkeiten die vom Desktop Computer bekannten Vorzüge auch mobil zu genießen. Das scheint momentan zwar noch utopisch anzumuten, aber führt man sich vor Augen das heutige High-End PDAs zum Teil schon so leistungsstark sind wie drei Jahre alte Desktop Maschinen lässt es einem dieses Vorhaben schon weniger unrealistisch erscheinen. Die MMAPI (Mobile Media API) Expert Group (unter der Leitung von Nokia) hatte die Intention eine API zu entwickeln welche Hersteller übergreifend eine Plattform zur Verfügung stellt mit der sich Audio- bzw. Videoelemente implementieren lassen. Im Folgenden einige mögliche Einsatzgebiete:

- Erweiterte Nachrichtenübertragung mit Audio- und Videoelementen
- Audio/Video Konferenzen
- Sicherheitsüberwachungen wie Alarmanlagen oder Überwachungskameras
- Einkaufsführer (Wenn man sich einem Laden nähert welcher das gewünschte Produkt anbietet, könnte ein Signal ertönen)
- Mehr Möglichkeiten beim versenden von Kurznachrichten (z.B. Audio, Video und Bilder können zusammen mit einer Textnachricht verschickt werden)
- Interaktives online gaming mit verbessertem Sound und Video
- Streaming von Audio und Video

1.2 Konzept

Die Mobile Media API ist ein optionales Package für J2ME, welches versucht trotz der Limitierungen die sich durch geringe Ressourcen bzw. Leistungsfähigkeit ergeben und den Problemen die durch drahtlose Datenübertragung entstehen ein möglichst großer Spektrum an Audio- und Videowiedergabe bzw. Übertragung zu abzudecken. Deswegen wurde ein Basisgrundgerüst bzw. Konzept entworfen welches reibungslos und relativ unabhängig von Übertragungsart und Datenquelle arbeitet. Dieses Konzept der Mobile Media API gliedert sich in die vier Basistypen DataSource, Player, Controls, Manager welche ich in den folgenden Kapiteln näher vorstellen werde.

1.2.1 DataSource

javax.microedition.media.protocol

Class DataSource

```
java.lang.Object
  |
  +--javax.microedition.media.protocol.DataSource
```

Die Klasse DataSource wird benutzt um die Daten von ihrem Ursprungsort (Datei, Server,...) hin zum Player zu transportieren. Dabei bleiben die Details wie die Daten gelesen werden außen vor. Ferner bleibt es einem frei gestellt wie die Daten an den Player übertragen werden (HTTP, RTP,...) sollen.

Durch diese Klasse besteht auch die Möglichkeit für eine Anwendung ihre eigenen Daten, sprich ihre eigene DataSource zu erzeugen und quasi "on-the-fly" an den Player zu liefern.

1.2.2 Player

javax.microedition.media

Interface Player

Das Abspielen der eigentlichen Daten erledigt das Interface Player. Es besitzt die benötigten Methoden um die Wiedergabe zu starten, stoppen oder an eine beliebige Stelle des Datenflusses zu springen. Der Player spielt sowohl Audio als auch Video Daten ab. Von dem Zeitpunkt des Erschaffens eines Players bis hin zum kompletten Abspielen der entsprechenden Daten durchläuft der Player den so genannten Player Life Cycle. Dieser untergliedert sich in folgende Zustände:

- UNREALIZED:
  Das ist der Zustand in dem sich der Player direkt nach seiner Erschaffung befindet
- REALIZE:
  In diesem Zustand sammelt der Player die benötigten Informationen der abzuspielenden Daten. (z.B. baut Verbindung zu einem Server auf, liest eine Datei, kommuniziert mit anderen Objekten...)
- PREFETCHED:
  Kurz vor dem Abspielen befindet sich der Player im PREFETCHED Zustand. Hier wird der nötige Speicher freigegeben, Puffer mit Media Daten gefüllt und sonstige start-up Prozeduren durchgeführt.
- STARTED:
  Die Daten werden wiedergegeben.
- CLOSED:
  In diesem Zustand gibt der Player fast alle seine Ressourcen frei, und es bleibt dem User überlassen ihn weiter zu benutzen.

Das folgende Schaubild stellt die fünf Zustände sowie deren Übergänge ineinander und die dazu benötigten Methoden noch einmal dar.

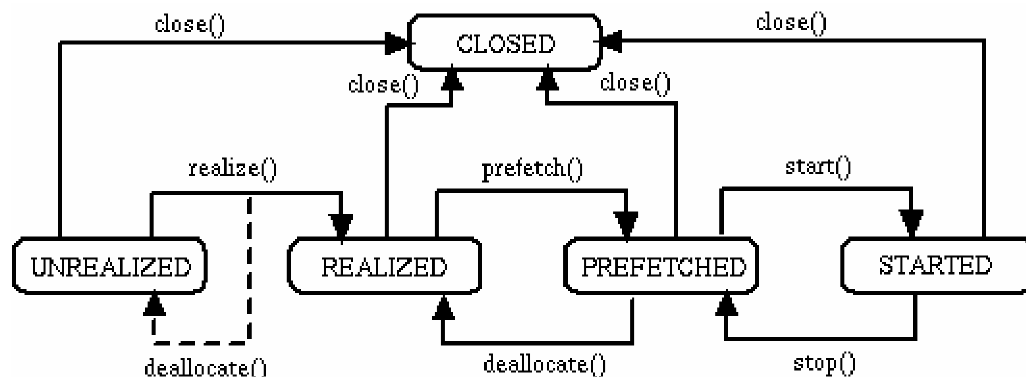

Abb. 1: Player Schaubild

Hier noch ein einfaches Beispiel für einen Player welcher ein MPEG File abspielt:

```
try {
     Player p = Manager.createPlayer("http://abc.mpg");
     p.realize();
     VideoControl vc;
     if ((vc = (VideoControl)p.getControl("VideoControl")) != null)
         add((Component)vc.initDisplayMode(vc.USE_GUI_PRIMITIVE,null));
     p.start();
     } catch (MediaException pe) {
     } catch (IOException ioe) {
     }
```

1.2.3 Controls

javax.microedition.media

Interface Control

Das im letzten Kapitel behandelte Interface Player besitzt Controllable als SuperInterface. Dieses wird benutzt um einem Player die jeweils anwendungsspezifischen Control Interfaces anzuhängen. Bei der Wiedergabe von Audiodaten empfiehlt es sich z.B. jeweils Regler für Lautstärke (VolumeControl), sowie für die Wahl der Endzeit (StopTimeControl) anzuhängen. Die verschiedenen Control Interfaces sind jedoch zu mehr als nur zu Justierungszwecken zu gebrauchen. Um z.B. Daten aufzuzeichnen wird das Interface RecordControl benutzt, oder um Videodaten überhaupt korrekt anzuzeigen wird das Interface VideoControl benötigt. Hier eine vollständige Auflistung mit kurzer Beschreibung aller Control Interfaces der Mobile Media API 1.0:

FramePositioningControl	Allows presice positioning to a video frame
GUIControl	Defined for controls that provide UI components
MetaDataControl	Retrieve metadata informationen from the media
MIDIControl	Provide access to MIDI rendering and transmitting devices
PitchControl	Raises or lowers the playback pitch without changing the playback speed
RateControl	Controls the playback rate
RecordControl	Records what is currently played by the Player
StopTimeControl	Allows preset stop time to be defined for a Player
TempoControl	Controls the tempo of the (MIDI) song
ToneControl	Interface to enable playback of user-defined monotonic tone sequence
VideoControl	Controls the display of visual content
VolumeControl	Controls the volume

Im Folgenden noch ein Beispiel für die Wiedergabe einer Audio Datei, bei der mit Hilfe des Interfaces TempoControl, die Wiedergabegeschwindigkeit verändert wird.

```
Player p;
TempoControl tc;

try {
    p = Manager.createPlayer("http://webserver/tune.mid");
    p.realize();

    // Grab the tempo control.
    tc = (TempoControl)p.getControl("TempoControl");
    tc.setTempo(120000); // 120 beats/min
    p.start();

} catch (IOException ioe) {
} catch (MediaException me) { }
```

Natürlich ist es auch möglich noch weitere, eigene Controls hinzuzufügen, wobei aber bedacht werden sollte, dass jedes zusätzlich implementierte Interface weiteren Speicherplatz benötigt, auch wenn keiner der verwirklichten Player es benutzt.

1.2.4 Manager

javax.microedition.media
Class Manager

```
java.lang.Object
  |
  +--javax.microedition.media.Manager
```

Der Manager hat schlussendlich die Aufgabe die beiden Konzeptgrundtypen DataSource und Player miteinander zu verbinden. Man erstellt mittels der Methode createPlayer(...) einen Player und weist diesem dabei die entsprechende DataSource zu. Möchte man nur einzelne Töne abspielen, so ist dies auch mit dem Manager möglich (playTone(int note, int duration, int volume)) und es ist nicht nötig hierfür den weitaus ressourcenintensiveren Player zu erstellen.
Dazu folgendes Beispiel:

```
try {
    Manager.playTone(ToneControl.C4, 5000 /* milsec */, 100 /* max vol */);
} catch (MediaException e) { }
```

Abschließend füge ich hier noch ein Schaubild an, welches das Zusammenspiel der einzelnen Komponenten noch einmal verdeutlichen soll.

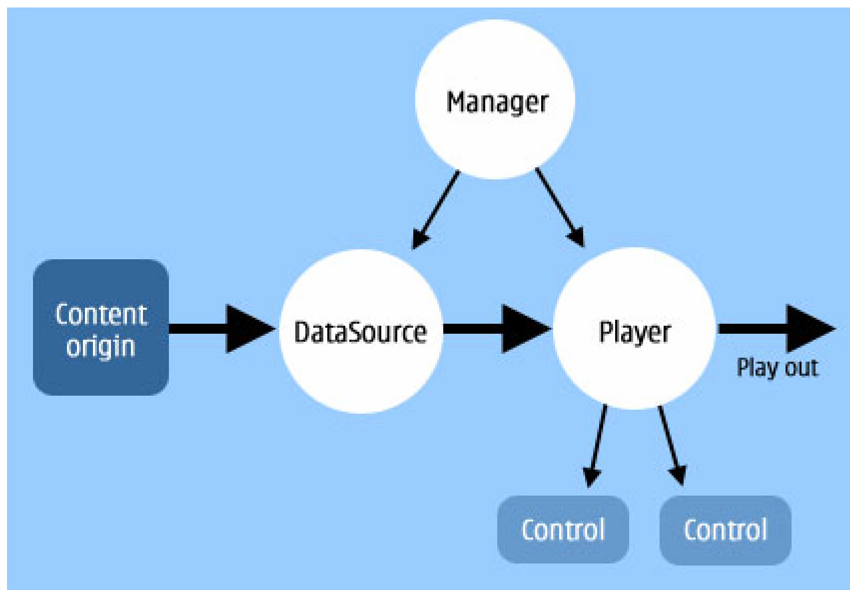

Abb. 2: Konzept der Mobile Media API

1.3 Eigenschaften

1.3.1 Transparenz in Bezug auf Datenquelle und Übertragungsart

Der Player der Mobile Media API wurde so konzipiert, dass er sämtliche Arten zeitbasierter Multimedia abspielen kann. Er ist in der Lage die ihm übermittelten Daten zu lesen, sie anschließend zu dekodieren und dann letztendlich wiederzugeben. Das heißt also, dass der Player selbst keine Formatvorgaben hat welche er akzeptiert und er benötigt auch keine speziellen Codecs um die verschiedenen Quellen wiederzugeben. Das hängt ausschließlich vom jeweils verwendeten Profil ab (welches im Moment bei fast allen Mobiltelefonen das MIDP 1.0 ist).
Die Klasse DataSource kümmert sich um die Art der Übertragung bzw. die Bereitstellung der Daten an sich. Mit ihrer Hilfe ist es möglich die Quelle der Daten auszulesen und die Art der Übertragung festzulegen. Der User besitzt allerdings nicht die Möglichkeit der Einsicht, wie die Daten genau eingelesen werden, aber er hat die Möglichkeit eine eigene DataSource zu erzeugen. Diese

Eigenschaft macht das Zusammenspiel von DataSource und Player äußerst flexibel. So ist es zum Beispiel möglich eine für eine Anwendung speziell erstellte DataSource gleich mit dem Player abzuspielen.

1.3.2 JMF (Java Media Framework) als Vorbild für die Mobile Media API

Die Mobile Media API wurde zu einem sehr großen Teil von dem Java Media Framework beeinflusst. Beide APIs haben die Grundidee der DataSource und des Players gemeinsam und besitzen beide das gleiche Look&Feel. Das macht für den Benutzer ein Wechseln zwischen den beiden sehr einfach. Doch was war der Grund dafür nicht einfach das JMF auf die mobilen Plattformen zu übertragen? Sicherlich in erster Linie die Größe, denn es wäre einfach unmöglich gewesen das JMF mit all seinen Funktionen auf eine Größe von 20 KB zu reduzieren, welche die Mobile Media API besitzt. Darüber hinaus gibt es natürlich noch weitere gewichtige Gründe: Das CLDC unterstützt keine Gleitkomma Zahlen, kein AWT (Abstract Window Toolkit) und es existieren noch eine Menge Objekte im JMF die die mobilen Geräte nicht verwenden können. Es besteht auch keine Möglichkeit zur Bildbearbeitung oder des Mapping. Die Mobile Media API ist auch nicht direkt als API für die Implementierung von Spielen oder anderen hochgradig interaktiven Anwendungen zu verstehen (dafür existiert eine eigene API, nämlich die Mobile Game API (JSR-178)), noch sollte sie in 2D oder gar 3D Anwendungen benutzt werden Das ist auch der Grund warum man das JMF auf jeden Fall der Mobile Media API vorziehen sollte, falls es sich vom verwendeten Gerät her leistungsmäßig anbietet. Überdies nutzt die Mobile Media API auch die gerätspezifische Hardware und Software aus um mit ihr zu interagieren, was die Rechenleistung erheblich herabsetzt und den Prozessor entlastet.

1.3.3 Vielseitigkeit im Bezug auf das Einsatzgebiet

Obwohl die Mobile Media API speziell für mobile Einzelgeräte entwickelt wurde, ist sie auf allen Java fähigen Plattformen einsetzbar, beginnend bei CLDC für Mobiltelefone bis hin zum J2EE, der Java 2 Enterprise Edition. Wobei erwähnt werden sollte, dass es sich für Anwendungen welche auf der J2SE (Java 2 Standard Edition) oder der J2EE laufen empfiehlt das weitaus leistungsstärkere und umfassendere JMF (Java Media Framework) zu benutzen. Hierbei bietet es sich für Programme die eine Server – Client Verbindung verwenden natürlich an auf dem leistungsstärkeren Desktop Computer das Java Media Framework zu benutzen und auf dem mobilen Endgerät die Mobile Media API einzusetzen. Da auch das JMF nicht an ein bestimmtes Übertragungsprotokoll gebunden ist, ermöglicht das auch eine leichte Kommunikation zwischen Server und Client.

1.3.4 Der “Audio-Only“ Block

Für die Konzeption des MIDP 2.0 wurde ein Teil der Mobile Media API als eigener, vom Rest unabhängiger Block konzipiert: der “Audio-Only“ Block. Wie der Name schon sagt unterstützt dieser Block nur die Audiowiedergabe, also sampled audio, tone generation, monotonic tone sequencing und simple WAV playpack. Durch diese Einschränkung nimmt der “Audio-Only“ Block nur eine Größe von 6 KB ein und benötigt damit im MIDP 2.0 Profile relativ wenig Platz. Deswegen müssen aber auch im Audio Bereich einige kleine Einschränkungen gemacht werden. Während der Player komplett erhalten blieb, musste die Klasse DataSource aus Platzgründen weichen. So ist auch die Neutralität gegenüber der Datenquelle nicht mehr gewährleistet und es hängt vom MIDP ab, welche Datentypen wiedergegeben werden. Da der “Audio-Only“ Block genauso wie die Mobile Media API aufwärts kompatibel ist, kann sie natürlich auf allen Javaplattformen auch anstelle der MMAPI verwendet werden, falls nur die Audioeigenschaften benötigt werden.

1.4 Zusammenfassung und Ausblick

Die Intention der Mobile Media API war es, zeitbasiertes Multimedia auf portablen Endgeräten zu ermöglichen. Überdies sollte durch die Schaffung einer einheitlichen API ein höheres Maß an Konformität erreicht werden, so dass Entwicklern untereinander eine gemeinsame Basis zur Verfügung gestellt wird. Dies spiegelt sich im Endeffekt für den Verbraucher in der Anzahl an verschiedenen neuen Applikationen wieder. Es wurde sehr großer Wert auf Flexibilität beim Design der API gelegt. Zum Einen hinsichtlich des Einsatzgebietes, die MMAPI kann auf dem CLDC/MIDP Stack bis hin zu allen höherkompatiblen Java Plattformen eingesetzt werden, als auch zum Anderen bei der Auswahl der verwendeten Elemente die ein möglich großes Spektrum an Daten wiedergeben

bzw. empfangen können. Die enorme Bedeutung für die weitere Entwicklung der Multimedia Technologie bei mobilen Geräten lässt sich vor allem daran erkennen, dass die MMAPI den Audio Block für das neue MIDP 2.0 zur Verfügung stellt und damit wohl in den meisten javafähigen Handys der neuen Generation inbegriffen sein wird.

2. Mobile 3D Graphics (JSR-184)

2.1 Einleitung

Im Zuge der schnell voranschreitenden Handytechnologie und mit dem kürzlichen UMTS Start in Deutschland scheint es nur eine Frage der Zeit bis das Handy mit einer ganzen Flut neuer Multimedia Anwendungen konfrontiert wird. Vom Aspekt der Grafikentwicklung fällt im Hardwarebereich auf, dass hierbei besonderer Wert auf die Entwicklung der Kameras, des Videos (MPEG 4) und hochauflösender neuer Displays gelegt wurde. Schon jetzt werden Spiele immer aufwendiger auf Handys umgesetzt, Wallpaper und DesktopThemes sowie kleinere Animationen und Filme gehören schon längst zum Standard. Diese und viele andere mediale Anwendungen sind natürlich besonders dann attraktiv, wenn sie mit einer überzeugenden Grafik aufwarten können. Um auf mobilen Geräten und den damit verbundenen geringen Speicherressourcen bzw. Prozessorleistungen überzeugen zu können wurde die Entwicklung einer API vorangetrieben (speziell im Hause Nokia), die sich konkret mit der Wiedergabe dreidimensionaler Objekte beschäftigt.

Betrachten wir kurz warum 3D Grafik bei vielen Anwendungen von Desktop Computern bereits ihren berechtigten Platz haben. Stark profitiert haben dürfte vor allem die Spiele, bzw. Entertainment Branche im Allgemeinen. Aber auch in der Industrie sind 3D Programme zum Erstellen von Modellen oder zum Simulieren von Testszenarien alltäglich. Letzteres wird vermutlich nicht oder lediglich in einem geringen Maße auf mobilen Geräten möglich sein. Es ergibt aber auf jeden Fall einen Sinn, die erstgenannten Bereiche auch auf mobilen Geräten ins Auge zu fassen. Vor allem der Markt für Handy Spiele erwartet in den nächsten Jahren Umsatzerlöse in Höhe von mehreren Milliarden Euro. Außerdem stellt es eine ernsthafte Konkurrenz zu tragbaren Spielekonsolen wie dem Gameboy von Nintendo dar, da es ja schließlich auch noch Telefon, Organizer, Kamera und vieles mehr in einem mobilen Gerät verbindet. Und hier ist einer der entscheidenden Ansatzpunkte für die Entwicklung einer 3D API speziell für mobile Geräte, denn schließlich ist ein gutes Spiel auch nur dann wirklich attraktiv, wenn es auch ansprechend aussieht.

Der limitierende Faktor des ganzen ist sicherlich die Leistungsfähigkeit der mobilen Geräte. Trotz enormer Verbesserungen in den letzten Jahren, muss man hier mit geringerer Prozessorleistung, kleineren Displays mit kleineren Auflösungen, geringerer Speicherkapazität und längeren Zugriffszeiten arbeiten. Ein weiterer wichtiger Aspekt ist, dass sämtliche mobilen Geräte zum Großteil über einen Akku betrieben werden, das heißt auch die Zeit in der solche mobilen Anwendungen laufen, begrenzt ist. Hierbei ist besonders zu betonen, dass die Akkutechnologie keine so rasanten Fortschritte macht wie die Mikroprozessortechnologie. Alles in allem sind das weitere gewichtige Gründe die für die Entwicklung einer speziellen 3D Grafik API sprechen.

2.2 Konzept

Da es diese API ermöglichen soll, den verschiedensten Anforderungen gerecht zu werden (Spielen, Navigationstools, animierte Nachrichten, Produkt Visualisierung oder Bildschirmschonern), reicht es nicht eine einfache Scene Graph API oder Immediate Mode API zu kreieren (Erklärungen dazu in den folgenden beiden Kapiteln). Die Mobile 3D Graphics API unterstützt je nach Bedarf eine der beiden Varianten oder beide zur gleichen Zeit. Die folgenden Kapitel beschäftigen sich näher mit diesen beiden Varianten und den Klassen der Mobile Graphics 3D API welche sich für deren Umsetzung verantwortlich zeigen.

2.2.1 World

javax.microedition.m3g
Class World

```
java.lang.Object
  |
  +--javax.microedition.m3g.Object3D
        |
        +--javax.microedition.m3g.Transformable
              |
              +--javax.microedition.m3g.Node
                    |
                    +--javax.microedition.m3g.Group
                          |
                          +--javax.microedition.m3g.World
```

Bei der Klasse World handelt es sich um einen speziellen Knoten in einem Baum welcher der Top-Level-Container für Scene Graphs ist. Der Scene Graph an sich ist eine Hierarchie von Blättern die alle über eine gemeinsame Wurzel miteinander verbunden sind. Diese Wurzel heißt im Fall der Mobile 3D Graphics API World. Um sich diesen Sachverhalt besser vorstellen zu können habe ich folgende Abbildung eingefügt, welche eine solche Baumstruktur einer 3D Grafik darstellt.

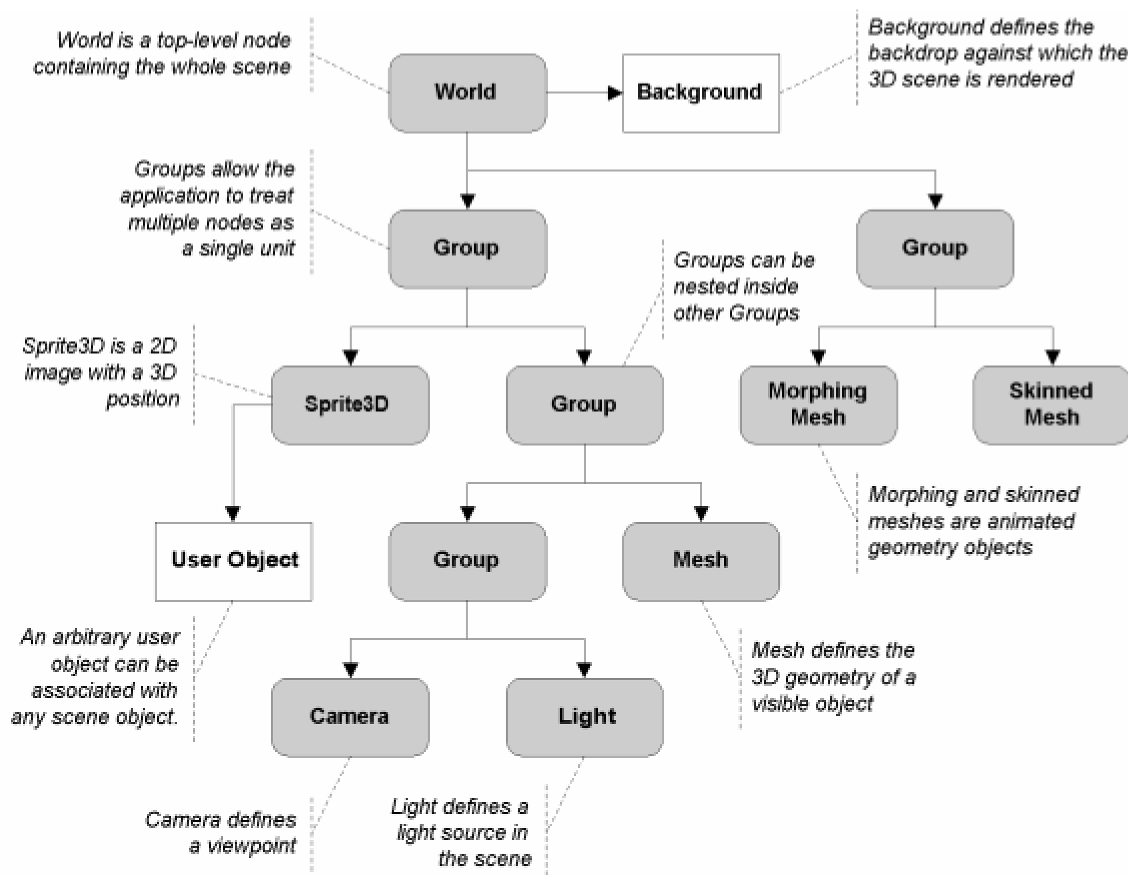

Abb. 3: Class World - Beispiel

An dieser Stelle möchte ich auch noch erwähnen, dass es durchaus auch möglich ist nur einzelne Teile des Scene Graphs darzustellen. Dies wird über eine spezielle Methode der Klasse Graphics3D ermöglicht. Dazu mehr im folgenden Kapitel. Auch möchte ich noch mal darauf hinweisen das World

keine Knoten mehr über sich haben kann, also nicht das Kind eines anderen Knoten sein kann, will heißen, dass es kein “Universe“ gibt wie in vielen anderen 3D APIs.

2.2.2 Graphics3D

javax.microedition.m3g

Class Graphics3D

```
java.lang.Object
  |
  +--javax.microedition.m3g.Graphics3D
```

Hierbei handelt es sich um die grundlegendste Klasse der gesamten API. Sie alleine besitzt die Möglichkeit mittels der Methode render(…) 3D Objekte zu zeichnen und damit am Bildschirm darzustellen. Aus diesem Grund betrachten wir die Methode render(...) etwas näher. Sie existiert vier Mal in der Klasse Graphics3D, wobei sich diese in den übergebenen Parametern unterscheiden. Die erste Möglichkeit besteht darin ein Objekt vom Typ World zu übergeben und dieses damit komplett darzustellen. Man spricht hier auch vom “retained mode“ der API. Die zweite Methode erlaubt es einem einzelne Knoten des Graphen bzw. eine Gruppe von Knoten darzustellen. Die beiden verbliebenen Methoden können dazu eingesetzt werden einzelne submeshs darzustellen. Ein submesh oder einfach mesh ist eine bestimmte geometrische Eigenschaft des 3D Objekts. Wird nicht die erstgenannte Methode ausgeführt, sondern eine der anderen so spricht man vom “immediate mode“ der API. Einer der wesentlichen Unterschiede zwischen diesen beiden Modi ist, dass sie ihre Informationen für Camera (die Perspektive aus der der User das 3D Objekt betrachtet) und Light (die Lichtverhältnisse und der Ort der Lichtquelle) unterschiedlich beziehen. Das Graphics3D Objekt besitzt eine eigene Instanz von Camera und Light, welche vom “immediate mode“ verwendet wird. Dahingegen überschreibt der “retained mode“ diese beiden Instanzen mit denen die er aus der Klasse World erhält. Bevor jedoch überhaupt irgendetwas dargestellt wird, muss erstmal ein Grafikobjekt mittels der Methode bindTarget(…) an das Graphics3D Objekt gebunden werden. Das im Folgenden bearbeitete Objekt wird schließlich wieder mittels der Methode releaseTarget(…) freigegeben. Welche verschiedenen Objekte von Graphics3D gebunden werden können, hängt von der verwendeten Plattform ab und reicht von Objekten des Typs Graphic bis hin zu Image2D. Wird eines der beiden MID Profile verwendet, muss es sich um ein Objekt vom Typ “javax.microedition.lcdui.Graphics“ handeln.
Wie schon einige Male erwähnt spielt möglichst geringer Speicherverbrauch und flüssiger Ablauf der Applikation trotz geringer Prozessorleistung eine große Rolle. Trotzdem kann es sein, dass es bestimmte Situationen gibt in denen höherer Speicherverbrauch und langsamere Darstellung billigend in Kauf genommen werden um eine höhere Qualität zu erzielen. Die Klasse Graphics3D bietet drei verschiedene Möglichkeiten das zu erreichen. Als da wären: Antialiasing, Dithering, True Color Rendering. Es handelt sich dabei um statische Variablen der Klasse Graphics3D, die mittels der Methode bindTarget(…) an- bzw. ausgeschaltet werden können. So kann man je nach Nutzen für die Anwendung die Qualität zugunsten der Geschwindigkeit herabsetzen oder eben umgekehrt. Ein Beispiel soll die erwähnten Fakten noch mal illustrieren.

```
public class MyCanvas extends Canvas
{
    World myWorld;
    int currentTime = 0;

    public MyCanvas() throws IOException {

        // Load an entire World. Proper exception handling is omitted
        // for clarity; see the class description of Loader for a more
        // elaborate example.

    Object3D[] objects = Loader.load("http://www.example.com/myscene.m3g");
    myWorld = (World) objects[0];

    }

    // The paint method is called by MIDP after the application has issued
    // a repaint request. We draw a new frame on this Canvas by binding the
    // current Graphics object as the target, then rendering, and finally
    // releasing the target.

    protected void paint(Graphics g) {

        // Get the singleton Graphics3D instance that is associated
        // with this midlet.

        Graphics3D g3d = Graphics3D.getInstance();

        // Bind the 3D graphics context to the given MIDP Graphics
        // object. The viewport will cover the whole of this Canvas.

        g3d.bindTarget(g);

        // Apply animations, render the scene and release the Graphics.

        myWorld.animate(currentTime);
        g3d.render(myWorld);     // render a view from the active camera
        g3d.releaseTarget();     // flush the rendered image
        currentTime += 50;       // assume we can handle 20 fps

    }
}
```

2.3 Vergleich: SVG Mobile, OpenGL ES, Mobile 3D Graphics

Da momentan mehrere Formate die Absicht haben den mobilen Grafikmarkt zu erobern, werde ich in diesem Abschnitt die drei populärsten Vertreter vorstellen.

Bei Scalable Vector Graphics (SVG) handelt es sich um ein Datei Format für 2D Darstellungen und kleinere 2D Animationen. Die Möglichkeiten welche SVG besitzt reichen von Bezierkurven und den daraus bestehenden Polygonzügen über Linienalgorithmen und deren präzise Einstellungen bis hin zu abschließenden Filtereffekten. Die mobilen Varianten nennen sich SVG Tiny für Mobiltelefone bzw. das etwas umfangreichere SVG Basic für PDAs. SVG Tiny konnte sich sogar den ausgeschriebenen Part für 2D animierte Vektorgrafiken in MMS (Multimedia Messaging System) bei der 3GPP Revision 5 sichern. (Bei 3GPP handelt es sich um das 3rd Generation Partnership Project, eine 1998 gegründete

Vereinigung, die es sich zur Aufgabe gemacht hat global gültige Standards für technische Spezifikationen und technische Reports im Bereich der Telekommunikation festzulegen)
Eine der bedeutendsten 3D Grafik APIs der letzten zehn Jahre ist OpenGL von Khronos. Diese API besticht durch eine riesige Anzahl von Möglichkeiten und Funktionen, welche jedoch im mobilen Bereich oft nicht benötigt werden. Aus dieser Intention heraus entstand OpenGL ES, eine abgespeckte und auf den mobilen Bereich zugeschnittene Version. Diese gilt im Moment als Standard für "mobile low-level immediate mode API", will heißen sie stellt in Kombination mit den gerätespezifischen Software Treibern ein solides Grundgerüst dar, welches je nach verwendeter Hardware unterschiedlichen Anforderungen gerecht wird.
Die Java Mobile 3D Graphics API geht noch einen Schritt weiter. Während die JSR-184 in der Ausführung absolut kompatibel zu OpenGL ES ist, und damit auch die Hardware welche OpenGL ES unterstützt, hervorragend für die JSR-184 geeignet ist, bietet sie einem auch die Möglichkeiten und Vorzüge einer hochentwickelten Scene Graph API. Da weniger Methodenaufrufe nötig sind, kann mehr Zeit in die Graphic Engine selbst, welche eigentlich immer in C oder Assembler verfasst ist, investiert werden, da diese (C, C++, Assembler) sowohl zur Laufzeit als auch in Bezug auf Speicherplatz wesentlich mehr Speicher und Rechenleistung in Anspruch nehmen als die Java Applikationen selbst. Zudem bietet die Scene Graph API mehr Möglichkeiten an, welche die verschiedensten Programme benutzen können, so dass durch diese Vielfalt an Möglichkeiten die Programme an sich kleiner und leistungsfähiger werden.
Zusammenfassend erkennt man das die Mobile 3D Graphics API dem Benutzer, sei er nun Programmierer oder Graphic Designer die meisten Möglichkeiten anbietet. Während SVG nur ein einfaches Datei Format, bzw. ein einfacher Animationsplayer ist, und OpenGL ES eher rudimentäre Arbeit leistet und zudem für Nicht-Programmierer schwer zugänglich ist, wird die JSR-184 allen Anforderungen gerecht. Durch ihr eigenes Dateiformat (m3g) ist es leicht möglich einzelne 3D Objekte in den Scene Graph zu laden, und diese ohne großes Programmieren darzustellen. Da die API aber auch in jedem anderen Kontext der Programmiersprache Java eingesetzt werden kann und der Scene Graph auch viele Möglichkeiten zur Bearbeitung bietet, bleibt es einem freigestellt, auch wesentlich komplexere Arbeiten mit der Mobile 3D Graphics API zu gestalten.

2.4 Ausblick

Auch wenn Kritiker immer die Performance mobiler Geräte gegenüber Desktop Computern bemängeln sollte ihr größter Vorteil, nämlich ihre Mobilität wie der Name schon sagt, nicht außer Acht gelassen werden. Diese bietet einem vor allem Dingen eins und das ist eine große Ersparnis an Zeit. Während man unterwegs ist oder auf eine Verabredung wartet ist man in der Lage bequem im Internet zu surfen und nach seinen Aktienkursen zu sehen. Oder wenn man abends mit seinen Freunden gemütlich in einer Kneipe abhängt und Lust auf ein kleines Spiel bekommt, steht dem nichts mehr im Wege. Ronald Azuma ein Wissenschaftler im Bereich der Virtual Reality Systeme, geht sogar noch weiter und mutmaßt, dass es in zehn Jahren keinesfalls mehr ungewöhnlich sein wird, wenn viele von uns mit ihrem eigenen persönlichen kleinen Holodeck unterwegs sein werden.

3. Abbildungsverzeichnis

Abb. 1: Player Schaubild
Abb. 2: Konzept der Mobile Media API
Abb. 3: Class World - Beispiel

4. Quellen (in alphabetischer Reihenfolge)

3D Graphics On Mobile Phones, Nokia Demos
(URL: http://www.nokia.com/nokia/0,8764,5400,00.html)

A Survey of Augmented Reality. Presence: Teleoperators and Virtual Environments, By Ronald Azuma. 06.04.1997
(URL: http://www.cs.unc.edu/~azuma/ARpresence.pdf)

About 3GPP – Third Generation Partnership Projekt
(URL: http://www.3gpp.org/About/about.htm)

Better tools for creating mobile 3D graphics on the way, Nokia Company
(URL: http://www.nokia.com/nokia/0,,42207,00.html)

Birth Of Mobile Java Multimedia, Nokia Library, By Antti Rantalahti and Jyri Huopaniemi, Nokia Research Center, 19.05.2003
(URL: http://www.nokia.com/nokia/0,,53719,00.html)

Bringing Time-Based Multimedia To Consumers Devices, Java White Paper
(URL: http://java.sun.com/j2me/docs/pdf/mmapiwp.pdf)

Cramming more components onto integrated circuits, By Gordon Moore, Electronics 38, 08.1965
(URL: ftp://download.intel.com/research/silicon/moorespaper.pdf)

JSR-135: Mobile Media API
(URL: http://www.jcp.org/en/jsr/detail?id=135)

JSR-184: Mobile 3D Graphics API for J2ME
(URL: http://www.jcp.org/en/jsr/detail?id=184)

Mobile SVG Profiles: SVG Tiny and SVG Basic
(URL: http://www.w3.org/TR/SVGMobile)

OpenGL ES Overview
(URL: http://www.khronos.org/opengles)

The Java 3D API, Technical White Paper
(URL: http://java.sun.com/products/java-media/3D/collateral/j3d_api/j3d_api_1.html)

The Rise Of Mobile Graphics, Nokia, By Kari Pulli, Nokia Mobile Phones, 05.05.2003
(URL: http://www.nokia.com/nokia/0,,27155,00.html)