# Java Spiele Schnittstellen OEM-Erweiterung der Standard Java 2 Micro Edition

Michael Becker

Siemens AG
Corporate Technology (CT SE2 / Java Task Force)
Email: Michael.Becker@mchp.siemens.de

**Kurzfassung**: In Zukunft werden Mobiltelefone nicht mehr mit wenigen fest installierten Applikationen ausgeliefert, welche danach nicht erweitert oder gelöscht werden können. Es werden über das Internet viele verschiedene Programme und Spiele zum herunterladen angeboten, welche auf verschiedenen Mobiltelefonen installiert werden können. Dazu eignet sich besonders die plattformunabhängige Programmiersprache Java, speziell die von Sun spezifizierte Java 2 Micro Edition, hier die Java „Connected Limited Device Configuration (CLDC)“ mit dem „Mobile Information Device Profile (MIDP)“. Das MIDP bietet grundlegende Funktionalität um auf mobilen Endgeräten, hier speziell auf Mobiltelefonen, einfache Applikationen und Spiele zu entwickeln. Auf den folgenden Seiten wird eine OEM-Erweiterung der Standard Java 2 Micro Edition um eine Java Spiele API aufgezeigt, welche auf Siemens Mobiltelefonen getestet und eingesetzt wird und momentan bei Sun Microsystems als Java Specification Request bearbeitet wird. Diese Schnittstelle beinhaltet viele zusätzliche grafischen Funktionen, wie z.B. Spriteunterstützung, Alpha-Blending, Vergrösserung, Verzerrung, Drehung und generischen Spiele-Komponenten, sowie Funktionen für Ton-, Licht-, und Vibratorsteuerung. Diese ermöglichen es, Spiele besonders einfach zu entwickeln und performant ablaufen zu lassen.

## 1 Einleitung

### 1.1 Hintergrund und Motivation

Die Entwicklung von Applikationen für neue Internet-fähige, mobile Produkte, wie z.B. Mobiltelefone, stellt Software Entwickler und Benutzerinterface-Designer (GUI) vor neue Herausforderungen. Viele Entwickler haben Erfahrung mit der Entwicklung von Java Applikationen für Desktop und Server Systeme. Können diese Erfahrungen aber auch im mobilen Sektor angewandt werden?

Bei mobilen Produkten, hier speziell Mobiltelefonen, handelt es sich um Geräte mit sehr limitierten Ressourcen, etwa kleines Display, wenig Speicher und vor allem geringe Rechenleistung. Um auf solchen eingeschränkten Geräten Applikationen zu entwickeln ist es schon einige Genialität nötig. Zu den technischen Einschränkungen sind auch noch die Erwartungen der End-Benutzer, dem Verbraucher, zu beachten. Computer-Anwender sind meist gewohnt, mit großen Displays, viel Speicher und relativ komplexen Applikationen auf Desktop-Computern umzugehen. Benutzer von Mobiltelefonen fordern jedoch einfache, intuitive und vor allem zuverlässige Geräte. Dies bedeutet, dass es nicht annehmbar wäre, während eines Telefonats oder einer Bank-Transaktion das Mobiltelefon zu rebooten oder verschiedene Tasten-Kombinationen auszuführen. Es ist auch erforderlich, dass das Gerät unter eingeschränkten Bedingungen bedient werden kann, in denen nur eine Hand zur Verfügung steht und nicht die volle Aufmerksamkeit auf das Gerät gesetzten werden kann [MIDApp].

## 1.2 Problemstellung

Mobiltelefone werden in Zukunft nicht mehr mit wenigen fest installierten Applikationen ausgeliefert, welche danach nicht erweitert oder gelöscht werden können. Es werden über das Internet viele verschiedene Programme und Spiele zum Herunterladen angeboten, welche auf verschiedenen Mobiltelefonen installiert werden können. Dazu eignet sich besonders die plattformunabhängige Programmiersprache Java, speziell die von Sun spezifizierte Java 2 Micro Edition, hier die Java „Connected Limited Device Configuration (CLDC)“ mit dem „Mobile Information Device Profile (MIDP)“. Das MIDP bietet grundlegende Funktionalität, um auf mobilen Endgeräten, hier speziell auf Mobiltelefonen, einfache Applikationen und Spiele zu entwickeln.
Vor gerade einmal 15 Jahren hatten wir eine ähnliche Situation. Da gab es Computer wie z.B. den Commodore 64 und den ZX Spectrum. Diese hatten bis auf das Display, ähnliche Beschränkungen von Speicher und Rechengeschwindigkeit. Trotzdem wurden tausende von Applikationen und vor allem Spiele für diese Rechner entwickelt. Diese Spiele-Ideen werden heute wieder benutzt und auf mobile Plattformen, wie Mobiltelefonen, umgesetzt.
Um diese Spiele auf Mobiltelefonen ablaufen zu lassen, werde spezielle und performante Schnittstellen für Grafik und Ton benötigt.

# 2 Anforderung und Konzept

## 2.1 Aufbau und Anforderungen der Java 2 Micro Edition

Die Java 2 Micro Edition (J2ME) ermöglicht es, die Java Technologie auf Geräten mit stark begrenzten Ressourcen zugänglich zu machen. Eine J2ME-Plattform baut auf der Java-Programmiersprache auf und unterstützt die Funktionalität, die auf solchen leistungsbeschränkten Geräten möglich ist. Um dies zu ermöglichen, ist die J2ME horizontal in sogenannte Konfigurationen aufgeteilt. Diese definiert das Minimum an Java Schnittstellen, die einem Entwickler für eine Gruppe von Geräten mit ähnlichen Ressourcen zur Verfügung gestellt wird. Allgemein gesagt, eine J2ME-Konfiguration gilt für Geräte der unterschiedlichsten Bereiche, die den gleichen Hardware-Anforderungen genügen.
Eine J2ME Konfiguration ist die Connected Limited Device Configuration (CLDC). Die CLDC, welche durch das Java Specification Request 30 (JSR-000030) spezifiziert wurde, gilt für mobile Endgeräte, die folgenden Anforderungen genügen:

- 128 kB - 512 kB Gesamtspeicher
- eingeschränkter Energiezufuhr (meist aufladbare Akkus)
- Netzwerkanschluss
- relativ eingeschränkten Benutzerschnittstellen

Auf den Konfigurationen der Java 2 Micro Edition setzen die sogenannten Profile auf. Profile werden jeweils für bestimmte Klassen von Zielgeräten spezifiziert. Bestandteile der Profile sind speziell für diese Geräteklasse zugeschnittene Klassen, u.a. für die Benutzerschnittstelle und die persistente Datenhaltung.
Derzeit liegt das Mobile Information Device Profil (MIDP) für die Connected Limited Device Configuration vor. Für dieses Profil ist eine Referenzimplementierung inklusive Emulator von Sun Microsystems erhältlich. Die Zielgruppe des MID Profils sind mobile drahtlose Geräte, wie Mobiltelefone, Two-Way Pager und PDAs mit Funkverbindung, also die Geräteklasse, die auch für das WAP Protokoll prädestiniert ist. Das Final Release der MIDP Spezifikation 1.0 wurde im September 2000 fertiggestellt.
MIDP eignet sich hervorragend für die Entwicklung von einfachen Applikationen und Spielen für Mobiltelefone. Es stellt folgende Programmierschnittstellen zur Verfügung [MIDP]:

- Anwendungssteuerung (Semantik und Kontrolle einer MIDP Anwendung)
- Benutzerinterface (UI) (Anzeige und Eingabe)
- Persistenter Speicher
- Netzwerk
- Timer

Das MIDP stellt folgende Software- und Hardware-Anforderungen:

| | | |
|---|---|---|
| **Betriebssystem Anforderungen** | • Einfacher Kernel, um eine Java VM auszuführen<br>• r/w auf einen nichtflüchtigen Speicher<br>• Timerfunktionalität<br>• Zugriff auf grafisches Display<br>• Entgegennahme von Benutzereingaben<br>• Integrierter Applikations-Manager | |
| **Hardware Anforderungen** | Display | Größe: mind. 96 x 54<br>Farbtiefe: mind. 1 bpp<br>Seitenverhältnis: ca. 1 : 1 |
| | Eingabegeräte (alternativ) | ITU-T Telefontastatur<br>QWERTY- Tastatur<br>Touchscreen |
| | Speicher (zusätzlich zu CLDC) | 128 kB nichtflüchtig (Komponenten)<br>8 kB nichtflüchtig (Daten)<br>32 kB flüchtig (Heap) |
| | Netzwerk | Two-way<br>Wireless<br>Keine ständ. Verbindung<br>Eingeschränkte Bandbreite |

Abbildung 1 MIDP Anforderungen (ergänzend zu CLDC)

Die Laufzeitumgebung für die MIDP-Anwendung setzt sich aus mehreren Komponenten zusammen (vergleiche Abbildung 2). Auf unterster Ebene steht die Hardware des mobilen Endgerätes, auf welcher die native Systemsoftware aufsetzt, die zum einen aus dem Betriebssystem und zum anderen aus speziellen auf das Gerät abgestimmten Bibliotheken besteht. Auf der Systemsoftware setzt die CLDC auf, die sowohl die KVM, als auch die entsprechenden Bibliotheken enthält, die durch die CLDC spezifiziert werden. Oberhalb der CLDC kommen die APIs, die durch das MIDP spezifiziert werden. Gleichzeitig können aber von den Hardwareherstellern auch noch OEM APIs implementiert werden, die den Zugriff auf gerätespezifische Funktionen ermöglichen. Diese OEM APIs können entweder auf den Funktionen der CLDC bzw. MIDP aufbauen, oder direkt auf native Funktionen zugreifen. In beiden Fällen besteht jedoch das Problem, dass Anwendungen, die auf diese OEM APIs zurückgreifen, eventuell nicht portabel zwischen verschiedenen Gerätetypen oder Geräten verschiedener Hardwarehersteller sind [CLDC]. Durch einen „Java Spezification Request“ (JSR) kann eine Standardisierung von OEM-Schnittstellenerweiterungen angefordert werden.

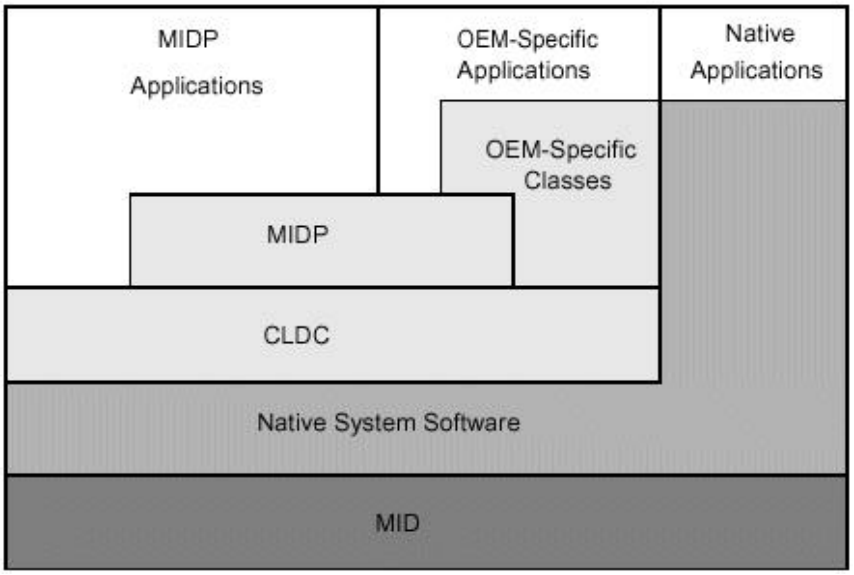



Abbildung 2 MIDP Architektur

### 2.2 Konzept Java Spiele Schnittstellenerweiterung

Die Standard Java 2 Micro Edition mit CLDC und MIDP bietet nur grundlegende Funktionalität, wie Bilder (Images) und einfache Grafische Funktionen an, mit welchen sehr einfache Spiele entwickelt werden können. Für komplexe und schnell ablaufende Spiele sind diese Funktionen nicht geeignet.
Die Standard J2ME soll um eine OEM Java Spiele Schnittstelle erweitert werden, die es ermöglicht, Spiele für Mobiltelefone schnell und einfach zu entwickeln und performant ablaufen zu lassen. Hierbei müssen die Beschränkungen eines Mobiltelefons beachtet werden. Diese sind insbesondere wenig Speicher, kleines Display mit eingeschränkter Farbtiefe (evtl. nur schwarz/weiß) und die geringe Rechenleistung des Prozessors.
In Gesprächen mit großen Spiele-Herstellern, die viel Erfahrung mit der Spieleentwicklung für kleine Geräte wie dem GameBoy haben, wurden Schnittstellen definiert, welche die gewünschte Funktionalität zur Verfügung stellen.

## 3 OEM Java Spiele Schnittstellenerweiterung

Die hier entwickelte OEM Java Spiele Schnittstelle ergänzt die Java 2 Micro Edition um Schnittstellen, welche zur Spieleentwicklung für mobile Endgeräte sinnvoll sind.

Diese OEM Java Spiele API ist in zwei Gruppen gegliedert:

- Die erste, die „OEM Game API“ (siehe Abbildung 3) enthält low-level Funktionen, welche eine direkte Schnittstelle zu nativen Funktionen des Mobiltelefons bietet.
- Die zweite, die „Java Game API“ (siehe Abbildung 3) ist komponentenbasiert, d.h. hier werden generische Java Komponenten (Objekte) angeboten, welche in den meisten Spielen wiederverwendet werden können, z.B. Highscore-Verwalter und die für den Spielablauf verantwortliche GameMachine.

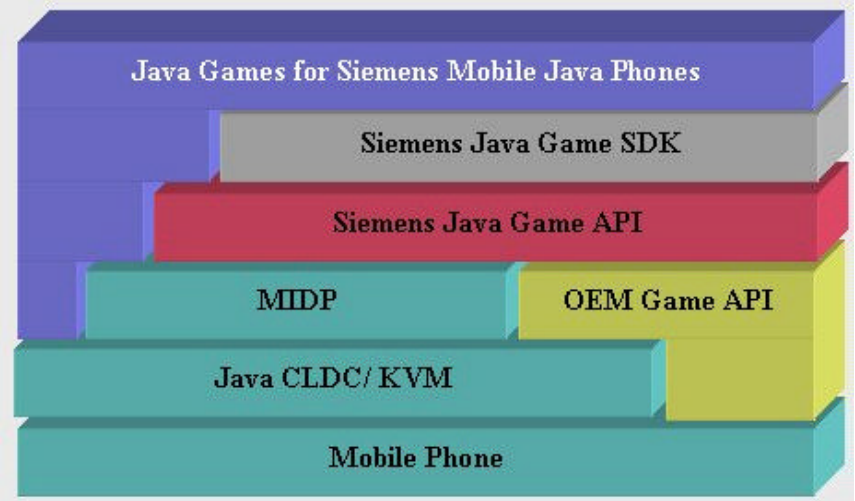



Abbildung 3 Java Game API Architektur

### 3.1 OEM Game API

Die OEM Game API ist eine Programmierschnittstelle, welche low-level Funktionen für die Spieleentwicklung zur Verfügung stellt.

#### 3.1.1 OEM Game API Funktionen

Die low-level Game API besteht aus folgenden Funktionen:

- **GraphicObject**
  Ein GraphicObject ist eine abstrakte Klasse für grafische Objekte. Diese Klasse stellt grundlegende Funktionalität zur Verfügung, welche für das Darstellen und Verwalten von grafischen Objekten benötigt wird.

- **Hardwarenahe Sprites und SpriteManager (GraphicObjectManager)**
  Sprites erweitern die Klasse GraphicObject. Sprites sind grafische Elemente (Bitmaps), welche unabhängig von Hintergrund und anderen Sprites bewegt und dargestellt werden können. Des weiteren können Kollisionen mit anderen Sprites erkannt werden. Ein Sprite besteht aus einem Bitmap und kann eine Maske enthalten, die es ermöglicht, einzelne Bereiche des Bitmaps auszublenden (ähnlich einer transparenten Farbe).
  Die Sprites werden von einem GraphicObjectManager verwaltet und gezeichnet (siehe auch Java-Native Anbindung).
- **TiledBackground**
  Diese Klasse erweitert die Klasse GraphicObject. Ein TiledBackground Objekt besteht aus einem Bild (Image) mit vielen einzelnen, meist 8x8 Pixel großen Kacheln und einer Karte (Map). Die Karte enthält Informationen darüber, wo jede einzelne Kachel gezeichnet werden soll. Dadurch können einzelne Bildkomponenten, sogenannte Tiles (Kacheln), für Spiel-hintergründe oder Spielfiguren immer wieder verwendet und gezeichnet werden, obwohl diese nur einmal im Speicher im TiledBackground Objekt vorhanden ist. Dies ist eine klassische Methode der Spieleentwicklung, um Speicher zu sparen.
- **Rasterizer**
  Die Klasse Rasterizer bietet grafische Funktionen an, die es ermöglichen, Bilder zu vergrössern, verkleinern, drehen, kippen und verzerren.
- **ExtendedImage**
  Diese Klasse bietet erweiterte Funktionen, wie setPixel, setPixelArray, getPixelArray, clearPixelArray, mit welchen ein normales MIDP Image manipuliert werden kann.
- **Alpha-Blending (Transparente Farbe)**
  (siehe auch Beispiel Alpha Blending)
  Alpha-Blending ermöglicht es, bestimmte Bereiche von Bildern auszublenden.
  Bei der Klasse Image ist eine spezielle Farbe definiert, welche beim Zeichnen nicht dargestellt wird, d.h. der Hintergrund bleibt an dieser Stelle erhalten.
  Bei der Klasse Sprite wird Alpha-Blending durch Masken realisiert. (siehe Sprite)
- **Clipping**
  Clipping ermöglicht es, außerhalb des sichtbaren Displays zu zeichnen (Virtual Screen).
- **Sound**
  In der Sound Klasse werden Funktionen zum Abspielen von Klängen angeboten, welche sich aus Frequenz und Dauer zusammensetzen.
- **Melody-Composer und Melody**
  Mit der Klasse MelodyComposer können Melodien durch Angabe von Noten und Notenlängen komponiert und zum Spielablauf abgespielt werden.

- **Lichtsteuerung**
  Hiermit kann die LCD-Hintergrundbeleuchtung des Mobiltelefons gesteuert werden.
- **Vibratorsteuerung**
  Hiermit kann der Vibrator des Mobiltelefons gesteuert werden.

**Beispiel Alpha-Blending:**

Angenommen, es liegen zwei einfache Bilder (Ufos) vor (Sprites, vergleiche Abbildung 4):

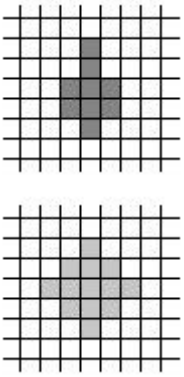


Abbildung 4 Sprites

Wenn diese beiden Ufos kollidieren, sollte folgendes Bild zu sehen sein (vergleiche Abbildung 5):

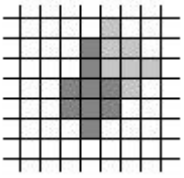


Abbildung 5 Sprite Kollision mit Transparenz

Leider ist es in Wirklichkeit nicht so. Das zum Schluss gezeichnete Ufo überzeichnet das andere, da ein Java Image durch ein Rechteck beschrieben wird (vergleiche Abbildung 6).

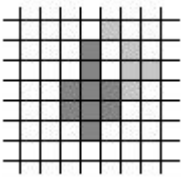


Abbildung 6 Sprite Kollision ohne Transparenz

Durch Clipping (Rechtecken) könnten hier die Regionen ausgeblendet werden, in welche nicht gezeichnet werden darf. Dies könnte aber bei komplexeren Bildern recht aufwendig sein und die Performance auf so kleinen Geräten würde dafür auch nicht ausreichen.
Hier wurde die MIDP Image Klasse erweitert. Es kann zu den normalen Farben auch eine transparente Farbe angegeben werden, welche dann wie bei GIF-Bildern einfach ausgeblendet wird, wodurch ein übermalen verhindert wird.

### 3.1.2 Java-Native Anbindung

Um hoch performante grafische Funktionen anbieten zu können, wurden alle Funktionen nativ implementiert. Ein Java-Wrapper bietet die nativen Funktionen in der Java Game API an. Dieser ruft direkt die nativen Funktionen auf und benutzt auch den nativen Speicher (siehe Abbildung 7).
Um den nativen Speicher vor dem Garbage-Collector zu schützen, sind sämtliche Java Klassen, die native Funktionen besitzen, von NativeMem abgeleitet. NativeMem implementiert eine Art verkettete Liste, in welche die nativen Objekte eingehängt werden. Dadurch wird dieser Speicher erst freigegeben, wenn die Referenz auf das Java-Wrapper-Objekt freigegeben wird.

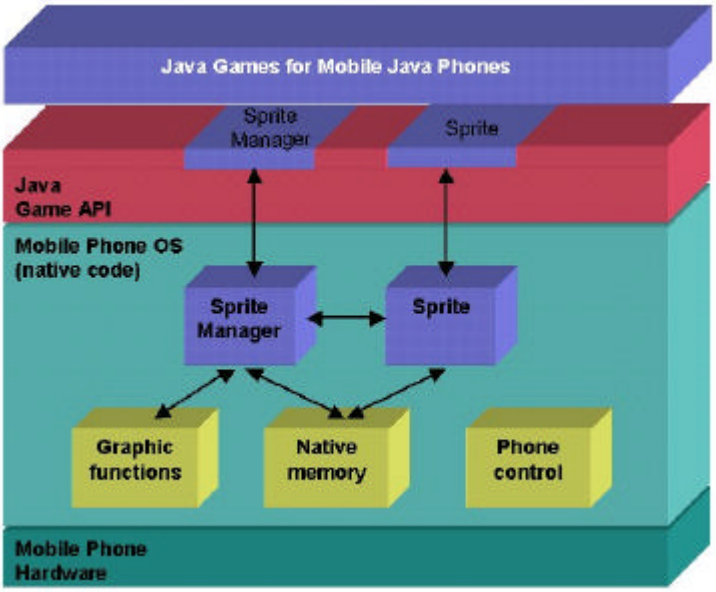



Abbildung 7 Java-Native Architektur

**Beispiel Native-Anbindung anhand der Sprite Klasse:**

- Java-Code des Sprite Wrappers (nicht gekürzt):

Java Konstruktor

```
public Sprite(Image pixels,
              Image mask,
              int width,
              int height,
              int numFrames)
{
       createNativeSprite(pixels, mask, width,
                                height, numFrames);
}
```

Aufruf native Konstruktor

```
public native void createNativeSprite(Image pixels,
                                       Image mask,
                                       int width,
                                       int height,
                                       int numFrames);
```

- Java-Code kompilieren

- Java-Klassen mit nativen Methoden werden mit der von Sun gelieferten Klasse JavaCodeCompact in eine C-Datei umgewandelt, welche zu den nativen Sourcen dazugebunden wird.

- Nativer-Code:

Nativer Konstruktor

Java Instanzen und Variablen vom Stack holen

Erzeugen des nativen Sprite-Objektes

```
void createNativeSprite(void)
{
   TP_Sprite nativeSpriteInst;

   //java int
   INT32 numFrames = popStackAsType(INT32);
   //java int
   INT32 height = popStackAsType(INT32);
   //java int
   INT32 width = popStackAsType(INT32);
   //Java Image with mask
   INSTANCE mask  = popStackAsType(INSTANCE);
   //Java Image with pixels
   INSTANCE pixels  = popStackAsType(INSTANCE);
   //Java Object
   INSTANCE jSpriteInst = popStackAsType(INSTANCE);

   nativeSpriteInst=Sprite_New(Malloc, Mfree);

   . . .
}
```

### 3.2 Siemens Java Game API

Die Siemens Java Game API kapselt alle low-level Funktionen der OEM Game API (vergleiche Abbildung 3) und bietet einige zusätzliche high-level Java Komponenten an, die in verschiedenen Spielen immer wieder verwendet werden können.

### 3.3 Siemens Java Game SDK

Das Siemens Java Game SDK erweitert die schon besprochenen Schnittstellen und Komponenten um einen Simulator mit Debugger und einen grafischen GameBuilder.

### 3.4 Ergebnisse und Zusammenfassung

Das standard Java 2 Micro Edition wurde um zusätzliche Schnittstellen für die Spieleentwicklung erweitert. Diese Schnittstellen ermöglichen es, Spiele einfach zu entwickeln und hoch performant ablaufen zu lassen. Erste Spiele bestätigen das Konzept. Diese Schnittstelle wird durch einen Java Spezification Request (JSR) standardisiert.

## 4 Ausblick und Schlußbemerkung

### 4.1 Netzwerkfähige Spiele Schnittstellenerweiterung

Ericsson, Motorola und Siemens haben sich in einem Konsortium zusammengeschlossen, um eine Spezifikation einer „Universal Mobile Games Platform“ auszuarbeiten. Ziel dieses Konsortium ist es, Service-Anbietern und Spiele-Entwicklern eine standardisierte Plattform für Netzwerkspiele anzubieten.
Die erste Spezifikation wird im Q3 2001 erwartet.
Die besprochene Game API kann dann an die „Universal Mobile Games Platform“ durch zusätzliche netzwerkfähige Klassen angebunden werden

### 4.2 Standardisierung

Die von Siemens entwickelte GameAPI ist noch proprietär. Jedoch wird diese Schnittstelle eine Grundlage für die Standardisierung anhand eines Sun Microsystems „Java Spezification Request“ (JSR) bieten.

### 4.3 Hardware Spiele Unterstützung

Für kommende Mobiltelefon-Generationen können aufgrund der gewonnenen Erfahrungen dieser Game API neue Hardware Komponenten festgelegt werden, welche den Spielablauf besser unterstützen und evtl. Funktionen in Hardware anbieten können.

### 4.4 MIDP NG

MIDP Next Generation wurde als JSR-000118 angemeldet und stellt folgende Anforderungen:

- Kompatibel mit MIDP 1.0
- M-Commerce Focus
- Sicherheitsmodell (Signierung und Verifizierung), HTTPS
- Netzwerk über Sockets und Datagram
- Push-Architektur
- Low-level LCDUI Erweiterungen für Spieleentwicklung und Unterstützung größerer Displays
- XML Parser
- Einfaches Sound API

# 5 Literatur


[Sun] Sun Microsystems, http://java.sun.com

[CLDC] JSR-000030 J2ME Connected Limited Device Configuration

[MIDP] Mobile Information Device Profile (JSR-37), JCP Specification Java 2 Platform, Micro Edition, 1.0

[MIDP_NG] Mobile Information Device Next Generation (MIDP_NG), JSR-000118 http://java.sun.com/aboutJava/communityprocess/jsr.html

[MIDApp] Applications for Mobile Information Devices, White-paper

[UMGP] Ericsson, Motorola, Siemens ; “Universal Mobile Games Platform”, http://www.ericsson.com/press/20010321-0054.html, 21.03.2001


# 6 Abbildungen


Abbildung 1 MIDP Anforderungen (ergänzend zu CLDC) ..... 4
Abbildung 2 MIDP Architektur ..... 5
Abbildung 3 Java Game API Architektur ..... 6
Abbildung 4 Sprites ..... 8
Abbildung 5 Sprite Kollision mit Transparenz ..... 8
Abbildung 6 Sprite Kollision ohne Transparenz ..... 9
Abbildung 7 Java-Native Architektur ..... 10