Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

Sascha Kluth

# Tätigkeitsbericht im Projekt "Pervasive Gaming Framework"

*Fakultät Technik und Informatik*
*Department Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

Sascha Kluth

# Tätigkeitsbericht im Projekt "Pervasive Gaming Framework"

Bericht eingereicht im Rahmen des Masterstudiums
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg
Betreuender Professor: Prof. Dr. Olaf Zukunft

Abgegeben am 6.März 2009

**Sascha Kluth**
**"Tätigkeitsbericht im Projekt Pervasive Gaming Framework"**

**Stichworte**
Android, Emulator, GPS, Hardwareanbindung, Kamera, Kompass, Künstlicher Horizont, Wii, Wii-Remote

**Kurzzusammenfassung**
Für die Nutzung auf dem Android-Emulator wurden mehrere Komponenten entwickelt, mit denen es möglich ist, Hardware wie Kamera, Kompass und Wii-Remote für die Nutzung in Androidanwendungen verfügbar zu machen. Die einzelnen Komponenten wurden für die Verwendung in einer Rollenspiel-Demoanwendung vorbereitet. Durch die Auswertung von Kompass- und GPS-Daten ist es möglich ein Videobild mit einem künstlichen Horizont zu versehen und die Position von Mitspielern einzublenden.

# Darstellungskonventionen

In dieser Arbeit werden folgende Darstellungskonventionen verwendet:

Ein- und Ausgaben in der Konsole

```
haw@desktop:~$
```

# Inhaltsverzeichnis

# 1 Arbeitsgruppe „Rollenspiel“, Teilprojekt „Hardwareanbindung“

Im Rahmen des Projektes „Pervasive Gaming Framework“ sollte im Rahmen der Projektlaufzeit vom 10.Oktober 2009 bis 23.Januar 2009 ein Framework erstellt werden, mit dessen Hilfe es Entwicklern von Spielen für die Android-Plattform ermöglicht wird, das Android -Framework leicht zu ergänzen. Dazu sollte das „Pervasive Gaming Framework“ modular aufgebaut werden, so dass es leicht möglich ist eigene Servicemodule bereit zu stellen, die dann von den Anwendungen genutzt werden können. Diese Servicemodule sollten zur Laufzeit nachladbar und austauschbar sein. Im Projekt sollten neben dem eigentlichen Framework Servicemodule wie z.B.

- Persistence Modul: Zur persistenten Speicherung beliebiger Anwendungsdaten auf einem Server. [HUTZLER]
- Database Module: Zur transparenten Anbindung von (Serverbasierten) Datenbanken. [HUTZLER]
- Transport Module: Zur transparenten Datenübertragung zwischen Clients untereinander und zum Server. [SALCHOW]
- Data Ascertainment Module: Zur Datenerhebung auf Clients. Diese Module sollten die Erhobenen Daten sowohl der lokalen Anwendung, als auch Serveranwendungen und weiteren Clients zur Verfügung stellen. [SALCHOW]

Als eine Demo-Anwendung sollten eine Kalenderanwendung (s. [ELAYADI],[PLITSCHKA]) und ein Rollenspiel (s. [DEDAJPREISLER]), erstellt werden, in dem sich mehrere Mitspieler mittels einer Android-Anwendung miteinander kämpfen können. Dabei soll die Positionierung der einzelnen Spieler via GPS erfolgen und mit Hilfe eines Kompasses die Ausrichtung der Gegner zueinander errechnet werden. Die Darstellung der Gegner soll in einer Live-Video-Darstellung in Form von Symbolen dargestellt werden. Es soll möglich sein, mittels einer Wii-Remote ein virtuelles Schwert zu schwingen und so die Mitspieler zu bekämpfen.

Das erste Android-Endgerät G1 war zu Beginn des Projektes in Deutschland nicht verfügbar, daher sollte der Android-Emulator V1.0 für die Entwicklung genutzt werden.

Aufgabe des Autors war es, im Teilprojekt Rollenspiel die Anbindung der benötigten Hardware an den Emulator zu gewährleisten. Ebenso war der Autor für die Einrichtung der Entwicklungs- und Testumgebungen verantwortlich. Diese Einrichtungsaufgaben sowie die Evaluation einer Alternativplattform (s. [ANDROIDOPENMOKO]) werden in diesem Bericht nicht betrachtet.

# 2 Recherche

In einem Beitrag zum „Google Android Developer Challenge“ [GOOGLEADC] haben zwei Studenten der Universität Koblenz-Landau, ein Navigationskonzept namens „enkin“ auf Basis des Android-Emulators implementiert, das sich in Teilbereichen mit der Vision dieses Projektes deckt.

Dabei wurde eine USB-Kamera und ein USB-Kompassmodul an den Emulator angebunden, so dass es möglich wurde, in einer Android-Anwendung ein Livebild der Kamera mit Ortsangaben zu anotieren. Dabei wird basierend auf GPS- und Kompassdaten errechnet in welchem Bildbereich des Kamerabildes die Anotationen dargestellt werden sollen. Detailierte Dokumentationen zu diesem Projekt standen dem Autor nicht zur Verfügung. Es konnte auf eine grobe Beschreibung [ENKINPDF] zurückgegriffen werden, in der eine Übersicht des Projektes vermittelt wird. Im Enkin-Projekt wurde ein OceanServer OS5000-US Kompassmodul (s. [OS5000]) verwendet, das in seiner seriellen Variante auch in diesem Projekt Anwendung fand.

Laut [ANDROIDEMULATOR] hat der Emulator folgende für das Projekt relevanten Einschränkungen:

- Keine Unterstützung von USB-Verbindungen
- Keine Unterstützung von Videokameras
- Keine Unterstützung von Bluetoothgeräten

Daraus ergab sich die Notwendigkeit, im Emulatorbetrieb andere Möglichkeiten zu finden um die benötigten Funktionalitäten bereit zu stellen. Dies sollte in Form von Data Ascertainment Modulen geschehen, insbesondere:

- Compass Module: Zur Anbindung externer Kompasshardware
- Wii-Remote Module: Zur Verwendung einer Wii-Remote als Eingabemedium
- Camera Module: Zur Darstellung eines Live-Videobildes

Unter [ANDROIDSENSOREN] stellt Google eine Javaanwendung zur Verfügung mit der es möglich ist, Sensordaten von Beschleunigungssensoren, Kompass und Thermometer für ein Androidgerät zu simulieren. Dieser Sensorsimulator wird dabei als Serveranwendung verwendet, auf dessen Daten Androidanwendungen via Socketverbindungen zugreifen können. Eine Anbindung weiterer Geräte, insbes. Kameras, wird nicht unterstützt.

Unter [ANDROIDSENSOREN] wird ebenfalls erläutert, wie in der Androidanwendung die Standard-Systemservices und Listener so adaptiert werden können, dass die Androidanwendung Hardwaresensoren nutzt, sofern sie zur Verfügung stehen und ansonsten auf den Simulator zurückgreift.

Unter [VIDEOEMULATOR] wird beschrieben, wie es möglich ist, ein via TCP zur Verfügung gestelltes Bitmap in eine Androidanwendung einzubetten. Diese Beschreibung ist

zwar unvollständig und die bereitgestellten Sourcen laufen nur in der älteren Androidemulatorversion 0.9, geben aber ein nutzbares Gerüst für eine eigene Weiterentwicklung vor.

Unter [WIILI] ist ein Entwicklerforum veröffentlicht, das sich mit der Portierung einer Linux-Distribution für die Wii-Spielkonsole beschäftigt. Hier werden auch diverse Bibliotheken beschrieben, die sich mit der Einbindung der Wii-Remote in eigene Anwendungen beschäftigen. Die verschiedenen Bibliotheken wurden evaluiert und WiiRemoteJ [WIIREMOTEJ] als einzige stabile Version ausgewählt. Mit WiiRemoteJ ist es möglich, mittels einer Standard-Javaanwendung eine Wii-Remote anzusteuern und deren Signale auszuwerten.

# 3 Konzeption

Da es zu Projektbeginn möglich erschien, zum Projektende zwei G1-Geräte zur Verfügung zu haben, sollte die Konzeption die Möglichkeit vorsehen, die zu entwickelnde Androidanwendung möglichst ohne Änderungen auch auf der Hardwareplattform laufen zu lassen.

Bei der Recherche hat sich gezeigt, dass die benötigten technischen Voraussetzungen gegeben zu sein scheinen.

Im Gesamtprojekt sollten Dienstleistungen einzelner Module als Frameworkservices einbindbar sein. Da das „Pervasive Gaming Framework“ selbst noch zu entwickeln war, sollten die einzelnen Module im Teilprojekt „Hardwareanbindung“ zunächst als eigenständige Anwendungen erstellt werden und nach Bereitstellung der Modulschnittstellen des Framework möglichst direkt in ein Framework Modul überführbar sein. Hierbei stellt sich jedoch das Problem, dass die Module zur Hardwareanbindung nicht als Android-Projekt implementiert werden konnten, da die benötigten Schnittstellen im Emulator nicht zur Verfügung stehen und nicht bekannt war, ob die Anbindung an die Schnittstellen der Hardware funktionieren würden.

Im Emulator steht die Möglichkeit zur Verfügung Daten via TCP, UDP und im Falle von Kompassdaten über eine proprietäre Schnittstelle auszutauschen. Mit dem Hintergedanken eine größtmögliche Verteilbarkeit zu erreichen sollten daher für jede Hardware eine Serveranwendung erstellt werden, die die jeweilige Hardware verwaltet und die zur Verfügung stehenden Sensordaten über Netzwerkschnittstellen zur Verfügung stellen. Auf Seiten des Frameworks sollten dann einzelne Module zur Verfügung stehen, die die Services für die Entwickler des Rollenspiels transparent zur Verfügung stellen.

# 4 Implementation

Zur besseren Übersicht sei auf Abbildung 3 Systemübersicht, Seite 18 hingewiesen. Die Implementierung der Module wurde in mehrere Arbeitsstufen aufgeteilt. Da zu Projektbeginn eine Wii-Remote zur Verfügung stand, wurde mit der Entwicklung des Data Ascertainment Module „Wii-Remote“ begonnen.

## 4.1 Data Ascertainment Module „Wii-Remote“, „Bluetooth-Schwert“

Unter Nutzung der WiiRemoteJ-Bibliothek (s.[WIIREMOTEJ]) wurde eine Standard-Java-Anwendung implementiert, die die Registrierung einer Wii-Remote über den Bluetoothstack des Betriebssystems ermöglicht. Hierzu mussten auf dem verwendeten IBM-T43-Notebook die Bluetooth-Treiber durch Beta-Versionen des Chip-Herstellers ersetzt werden. Nach dieser Aktualisierung war die Nutzung unter Windows XP SP3 und Kubuntu 8.04 problemlos möglich.

Die registrierte Wii-Remote liefert über einen Eventhandler ca. 20x pro Sekunde aktualisierte Informationen über den Status seiner Sensoren. Da die Daten der Beschleunigungssensoren sehr fein aufgelöst sind und sich mit jeder Aktualisierung ändern (selbst wenn die Wii-Remote ruhig auf einer festen Unterlage liegt) wurden die möglichen Achs-Werte in Bereiche aufgeteilt und auf einer Skala, wie folgt, dargestellt.

```
[]: Skalenbereich
..: Auslösender Bereich
,,: Unausgewerteter Bereich
| : Aktueller Wert

X-Achse:                Y-Achse:                Z-Achse:
[..,,,,|,,,:,,,,,,,,,,] [..,,,,,,,,,:,,|,,,,,,,] [..,,,,|,,,:,,,,,,,,,,]
```

In Experimenten wurden die Skalen der drei Beschleunigungssensoren für die X-,Y- & Z-Achse beobachtet, während mit der Wii-Remote die Schläge eines Schwertes nachgeahmt wurden.

Dabei zeigte sich, eine Aufteilung in 20 Bereiche die sicherste Gestenerkennung ermöglichte und dass ein „Schwerthieb“ dann zu erkennen ist, wenn der X-Wert im auslösenden Bereich und der Y-Wert positiv ist.

Aufgrund der hohen Updaterate der Wii-Remote wurde eine Sperre integriert, so dass maximal alle 2 Sekunden ein Schlag-Event ausgelöst werden kann.

Da die Fertigstellung der Frameworkservices zur Präsentation am 19. Dezember 2008 noch nicht abgeschlossen war, wurde der Schlag-Event über einen Key-Event an den Emulator übermittelt. Die Rollenspielanwendung wertete daher in diesem Stadium den Druck der Taste „X“ als Auslöser eines Schlages. In diesem Stadium war es somit möglich

mittels einer Wii-Remote einen Schwertschlag in der im Emulator laufenden Anwendung auszulösen.

Da das einzige G1-Gerät im Projekt für andere Entwicklungsaufgaben benötigt wurde konnte eine Übertragung des WiiEventMappers auf das Endgerät nicht mehr vorgenommen werden. Somit lässt sich keine Aussage darüber treffen, ob die Anbindung der WiiRemote an das Bluetoothmodul des G1 funktioniert.

## 4.2 Data Ascertainment Module „Kamera“

Wie in Kapitel 2 erläutert, ist die Nutzung einer Kamera im Emulator nicht direkt möglich. Daraus ergab sich die Idee, eine Web-Kamera zu nutzen, die Bilder über TCP ausliefern kann, um aus dem Framework heraus Bilder via TCP anzufordern und der Anwendung zur Verfügung zu stellen, wie unter [VIDEOEMULATOR] beschrieben. Diese Remoteanbindung ist auch auf der realen Hardware sinnvoll nutzbar, wenn es darum geht, Bilddaten externer Kameras zu verwenden. Um später möglichst wenige Berechnungsschritte auf dem Hostsystem auszuführen, auf dem der Emulator läuft, sollte eine eigene Serveranwendung erstellt werden, die die Bilder von einer an einen weiteren PC angeschlossenen USB-Kamera weiterverarbeitet. So war angedacht, dass eine spätere Kopplung an die zu erstellende Kompass-Server-Anwendung die Möglichkeit bieten sollte Informationen, wie einen künstlichen Horizont bereits in das Videobild einzurechnen, bevor es an die Android-Anwendung ausgeliefert wird. Die unter [VIDEOEMULATOR] zur Verfügung stehenden Sourcen und Anleitungen beziehen sich auf den Emulator 0.9 und mussten für eine Nutzung im Emulator 1.0 umgebaut werden. Die ebenfalls unter [VIDEOEMULATOR] bereitgestellten Sourcen für einen WebcamBroadcaster nutzen die von Sun bereitgestellten java.media Bibliotheken. Diese werden von Sun nicht mehr für Windows zur Verfügung gestellt, standen dem Autor dieses Berichtes aber aus einem älteren Projekt zur Verfügung. Diese Anwendung konnte nach geringer Modifikation für das Projekt übernommen werden. So lagen die größten Hürden für dieses Modul in der Bereitstellung des JavaMediaFramworks da dieses nur RGB-Kameras unterstützt, dem Autor zunächst aber nur eine YUV-Kamera zur Verfügung stand und eine Klasse zur Umrechnung von YUV- in RGB-Daten erstellt werden musste.

## 4.3 Data Ascertainment Module „Kompass“

Für die Ermittlung der aktuellen Position im Raum wurde die serielle Version des Ocean-Server OS5000-US Kompassmoduls verwendet (s. [OS5000]). Die Wahl der seriellen Version bietet gegenüber der USB-Version den Vorteil, dass die Anbindung an das Hostsystem potentiell direkt und ohne Treiber möglich ist, bzw. ein USB-zu-R232-Umwandler frei gewählt werden kann. So kann ein USB-zu-R232-Umwandler, für den die jeweilige Hostsystem-Treiber vorliegen, genutzt werden. Das Kompassmodul kann bis zu 40x pro Sekunde aktuelle Daten liefern. Da diese Daten von der Androidanwendung aber in das Videobild eingblendet werden sollen, das nur ca. 4x pro Sekunde aktualisiert wird, wurde die Updaterate des Kompassmoduls auf 12 Aktualisierungen pro Sekunde (gewünschte Framerate des Videobildes) gesetzt. Die Kommunikation mit dem Kompassmodul kann mittels einer einfachen Terminalanwendung geschehen. Sobald das Kompassmodul mit einer Spannung von 3-5V versorgt wird, gibt es seine Sensordaten über die serielle Schnittstelle aus.

Die jeweils aktuellen Daten können dann über eine Socketverbindung aus dem USB-zu-R232-Umwandler ausgelesen werden. Eine Standard-Java-Anwendung übernimmt dieses Auslesen, bereitet die Daten auf und stellt sie auf einer weiteren Socketschnittstelle zur Verfügung, von wo aus sie von der Android-Anwendung abgefragt werden können.

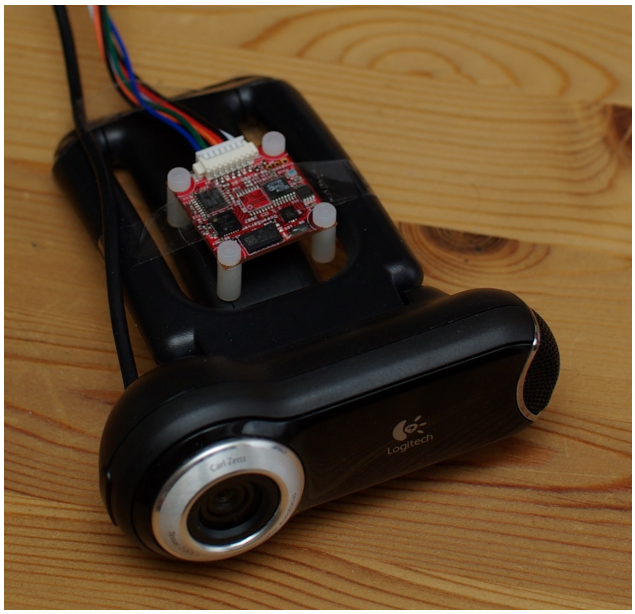


*Abbildung 1: USB-Kamera mit montiertem Kompassmodul*

## 4.4 „Einblendungen im Videobild“

Im laufenden Videobild sollte ein künstlicher Horizont eingeblendet werden, um die Lage im Raum darzustellen. Des weiteren sollten die Gegner im Videobild über Symbole dargestellt werden. Unter Verwendung von GPS- und Kompassdaten können die Lage der Kamera und des Gegners im Raum errechnet und in Beziehung gesetzt werden, so dass die Darstellung eines Kastens als Gegner im Kamerabild möglich ist.

Für die Berechnung der Positionen aus den GPS-Daten wurde eine Klasse „GPSLocation“ erstellt, die GPS-Lokationen eines (realen) Objektes hält und Methoden zur Berechnung von Winkeln zweier Objekte zueinander zur Verfügung stellt. Somit steht der Winkel in Grad, in dem ein Gegner sich befindet zur Verfügung.

Da, aus den Daten des Moduls „Kompass“, die Ausrichtung der Kamera im Raum bekannt ist, kann nun unter Berücksichtigung der Kamera-, der Bildschirmgeometrie und der Bildschirmauflösung die darzustellende Position des Gegners errechnet werden.

Das Zeichnen übernimmt die onDraw-Methode der Activity der Android-Anwendung. Bei der Erprobung des entsprechenden Codes war auffällig, dass die Android-Anwendung bei der Nutzung von Clip-Regionen beim Zeichnen auf Bitmaps die Zeichenopperationen auch in Bildbereichen, die nicht im Clip-Bereich liegen vollständig ausführt, sie aber nicht darstellt. Durch eine Reduzierung der Zeichenbefehle, auf die Befehle, die im Clip-Bereich zu Zeichenopperationen führen, konnte die Zeichengeschwindigkeit um den Faktor 14 verbessert werden. Dieses Verhalten konnte später auch auf der G1-Hardware bestätigt werden.

In Abbildung 2 werden zwei Screenshots des Emulators gezeigt. Im oberen Bereich des Emulator-Displays wird das Videobild gezeichnet, auf dem der künstliche Horizont aufgelegt wird. Abbildung 2 links zeigt dabei die Darstellung, wenn die Kamera ungefähr waagerecht gehalten wird, Abbildung 2 rechts zeigt die Darstellung, wenn die Kamera (bzw. das Mobile Gerät) um ca. 15° geneigt wird. Das rote Quadrat symbolisiert die Position eines Gegners.


*Abbildung 2: Künstlicher Horizont*

# 5 Evaluation Alternativplattform

Da sich die Bestellung der G1-Geräte im Laufe des Projektes mehrfach verzögerte, wurden zwei OpenMoko-Geräte [OPENMOKO] angeschafft. Für diese Geräte sollte bereits eine Portierung des Android-Betriebssystems vorliegen. Unter [OPENMOKOFORUM] gibt es eine aktive Entwicklergemeinde, die sich unter anderem auch mit der Android-Portierung befasst. Zur Projektlaufzeit stellte sich heraus, dass es keine fertige Portierung gab. Es gab jedoch ROM-Images mit dem Betriebssystem, die auf den, dem Autor vorliegenden, OpenMoko-Geräten nicht lauffähig waren. Es standen unvollständige Anleitungen zur Installation der ROM-Images und zur Erstellung eigener Kernel-Images und die Sourcen für das Android-Betriebssystem zur Verfügung. Mit Hilfe dieser Quellen und den Offiziellen Android-Quellen (s. [ANDROIDSOURCEN]) war es dem Autor möglich eigene Kernel-Images auf einem Ubuntu-System zu erstellen. Das reine Erstellen dauerte auf einem Pentium 4, 3,4GHz-System mit Kubuntu 8.04 rund 32 Stunden. Das Android-Betriebsystem konnte nun auf den OpenMoko-Geräten in Betrieb genommen werden.

Das Booten des Systems dauerte ca. 8 Minuten, ließ sich dann aber flüssig nutzen. Die Software des Projektes konnte im Rahmen der Projektzeit nicht mehr auf dieser Plattform getestet werden. Eine ausführliche Anleitung zur Erstellung und Installation von Android-Kernel-Images, zum Booten des Systems und zur Installation eigener Anwendungen soll in Kürze vom Autor [ANDROIDOPENMOKO] veröffentlicht werden.

# 6 Optimierungsmöglichkeiten

## 6.1 Videodarstellung

In der aktuellen Implementierung werden die Teilaufgaben „Neues Videobild anfordern“, „GPS-Daten aktualisieren“, „Kompassdaten aktualisieren“, „Winkel berechnen“, „Horizont zeichnen“ und „Gegner einblenden“ in einem Thread sequentiell ausgeführt. Dabei sind die Datenlieferanten zu „Neues Videobild anfordern“, „GPS-Daten aktualisieren“ und „Kompassdaten aktualisieren“ synchron angebunden und „Neues Videobild anfordern“ ist mit ca. 4 Updates pro Sekunde der langsamste Teil.

Es gibt nun zwei Möglichkeiten:

- „Neues Videobild anfordern“, „GPS-Daten aktualisieren“ und „Kompassdaten aktualisieren“ werden in eigenständigen Threads bearbeitet, wobei „Neues Videobild anfordern“ als Taktgeber dient und GPS- und Kompassdaten nur so häufig aktualisiert werden, wie neue Videobilder verfügbar sind. Es wird dann auf jedes Videobild der künstliche Horizont mit den aktuellen Daten eingezeichnet und, falls sichtbar, Gegner dargestellt.
- In der zweiten Variante werden die Aufgaben ebenfalls in eigenständige Threads ausgegliedert, als Taktgeber wird aber die Updaterate des Displays bzw. die Bearbeitungsgeschwindigkeit der onDraw-Methode genutzt. Dazu wird das letzte zur Verfügung stehende Videobild gepuffert und der künstliche Horizont mit den aktuellen Daten und sichtbare Gegner werden (sobald es hier Aktualisierungen gibt) auf das gepufferte Videobild gezeichnet. Dabei könnte das Videobild in Abhängigkeit der Lokalitätsdaten auch transformiert dargestellt werden, um den Eindruck einer flüssigeren Videodarstellung zu erzeugen. D.h., wenn die Kompassdaten eine Rotation von 10° nach links angeben, wird das Videobild ca. 100 Pixel weiter rechts gezeichent.

Diese Optimierungen sind auch bei der Verwendung der Android-Anwendung auf der realen Hardware grundsätzlich sinnvoll, wenn nur Systemeigene Datenquellen genutzt werden, da jede gesparte Berechnungszeit der Bearbeitung anderen Aufgaben zur Verfügung steht. Es ist jedoch zu beachten, dass die notwendigen Threadwechsel und Fallunterscheidungen ebenfalls Rechenaufwand bedeuten.

## 6.2 Taktung des Kompassmoduls

Das Kompassmodul ist im Bereich von 40 Updates pro Sekunde bis 1 Update alle 10 Sekunden frei konfigurierbar. Diese Konfiguration erfolgt über eine Socketverbindung und könnte somit auch von dem Data Ascertainment Module „Kompass“ vorgenommen werden. In Abhängigkeit dazu, wie schnell die Bearbeitung der Kompassdaten in der Androidanwendung erfolgen kann, könnte das Data Ascertainment Module diese Updaterate des

Kompassmoduls rekonfigurieren. Hierbei ist zu beachten, dass eine Rekonfiguration eine Unterbrechung der Datenlieferung von ca. 2 Sekunden hervorruft.

Experimentell könnte die onDraw-Methode in Abhängigkeit der Bearbeitungszeiten der einzelnen Aufgaben die Rekonfiguration des Kompassmoduls veranlassen. Da die Verarbeitung der Kompassdaten ebenfalls hier geschieht, würde sich rückgekoppelt ein annähernd optimaler Wert einstellen.

# 7 Offene und unbearbeitete Punkte

- Während der Projektlaufzeit wurde dass Ziel, die einzelnen Module über eine Schnittstelle an das Framework anzubinden aus Zeitgründen aufgegeben. Daher konnten die Komponenten des Autors nicht als Module in das Framework integriert werden und nur in der Demoanwendung „Rollenspiel“ genutzt werden.
- Es konnte nicht überprüft werden, ob das Data Ascertainment Module „Wii-Remote“ direkt auf der Hardware lauffähig ist. Hierzu wäre eine Migration der Standard-Java-Anwendung nach Android notwendig.
- Auf den OpenMoko-Geräten wurde das original Betriebssystem erfolgreich durch das Android-Betriebssystem ersetzt. Hier wäre das Android-Betriebssystem zu aktualisieren und die erstellte Software des Projektes auf dieser Plattform zu evaluieren.
- Da nun die Funktionsweisen der Data Ascertainment Module bekannt sind, könnten diese auf das original Betriebssystem von OpenMoko oder z.B. auch das iPhone (s. [IPHONE]) portiert werden.
- Das unter [ANDROIDSENSOREN] erläuterte Vorgehen, wie in der Androidanwendung die Standard-Systemservices und Listener so adaptiert werden können, dass die Androidanwendung Hardwaresensoren nutzt, sofern sie zur Verfügung stehen und ansonsten auf Simulatoren zurückgreift, sollte genutzt werden. Generalisiert betrachtet, sollten die zu den einzelnen Data Ascertainment Modulen gehörigen Serveranwendungen jedoch nicht transparent, sondern durch Auswahl der Anwendung / des Benutzers verwendet werden.
- Es sollte möglich sein, die Sensoren der  Data Ascertainment Module zu priorisieren, damit ausfallende Sensoren automatisch ersetzt werden können.

# 8 Bildnachweis

Sämtliche Bilder wurden vom Autor erstellt.

# 9 Quellen


[ANDROIDSOURCEN] Google Inc.; Android Sources; http://code.google.com Zugriffsdatum: 17.Oktober 2009

[ANDROIDEMULATOR] http://code.google.com/intl/de-DE/android/reference/emulator.html Zugriffsdatum: 17.Oktober 2008

[ANDROIDSENSOREN] http://code.google.com/p/openintents/wiki/SensorSimulator Zugriffsdatum: 17.Oktober 2008

[ANDROIDOPENMOKO] Kluth, Sascha: Installationsanleitung Android auf Openmoko Neo Freerunner, 2008

[DEDAJPREISLER] Dedaj, Dennis; Preisler, Thomas : Pervasive Gaming Framework. Projektbericht, HAW Hamburg. 2008

[ELAYADI] El-Ayadi, Amine: Pervasive Gaming Framework. Projektbericht, HAW Hamburg. 2008

[ENKINPDF] http://www.enkin.net/Enkin.pdf Zugriffsdatum: 17.Oktober 2008

[GOOGLEADC] Google Inc.; Google Android Developer Challenge; http://code.google.com/intl/de-DE/android/adc.html – Zugriffsdatum: 23. Februar 2009

[HUTZLER] Hutzler, Tobias: Pervasive Gaming Framework. Projektbericht, HAW Hamburg. 2008

[OPENMOKO] Openmoko Inc.; Neo Freerunner-Produktseite; http://www.openmoko.com/product.html – Zugriffsdatum: 23. Februar 2009

[OPENMOKOFORUM] http://wiki.openmoko.org/wiki/Android – Zugriffsdatum: 23.02.2009

[OS5000] OceanServer Technology Inc.; Compassmodules product page; OS5000-US; http://www.ocean-server.com/download/Compass_OS5000_Family.pdf – Zugriffsdatum: 23.Februar 2009

[PLITSCHKA] Plischka, Julia: Pervasive Gaming Framework. Projektbericht, HAW Hamburg. 2008

[SALCHOW] Salchow, Peter: Pervasive Gaming Framework. Projektbericht, HAW Hamburg. 2008

[VIDEOEMULATOR] Gibara,Tom; http://www.tomgibara.com/android/camera-source Zugriffsdatum: 17.Oktober 2008

[WIILI] http://www.wiili.org/index.php/Main_Page Zugriffsdatum: 17.Oktober 2008

[WIIREMOTEJ] http://www.wiili.org/WiiremoteJ Zugriffsdatum: 17.Oktober 2008

# 10 Systemübersicht

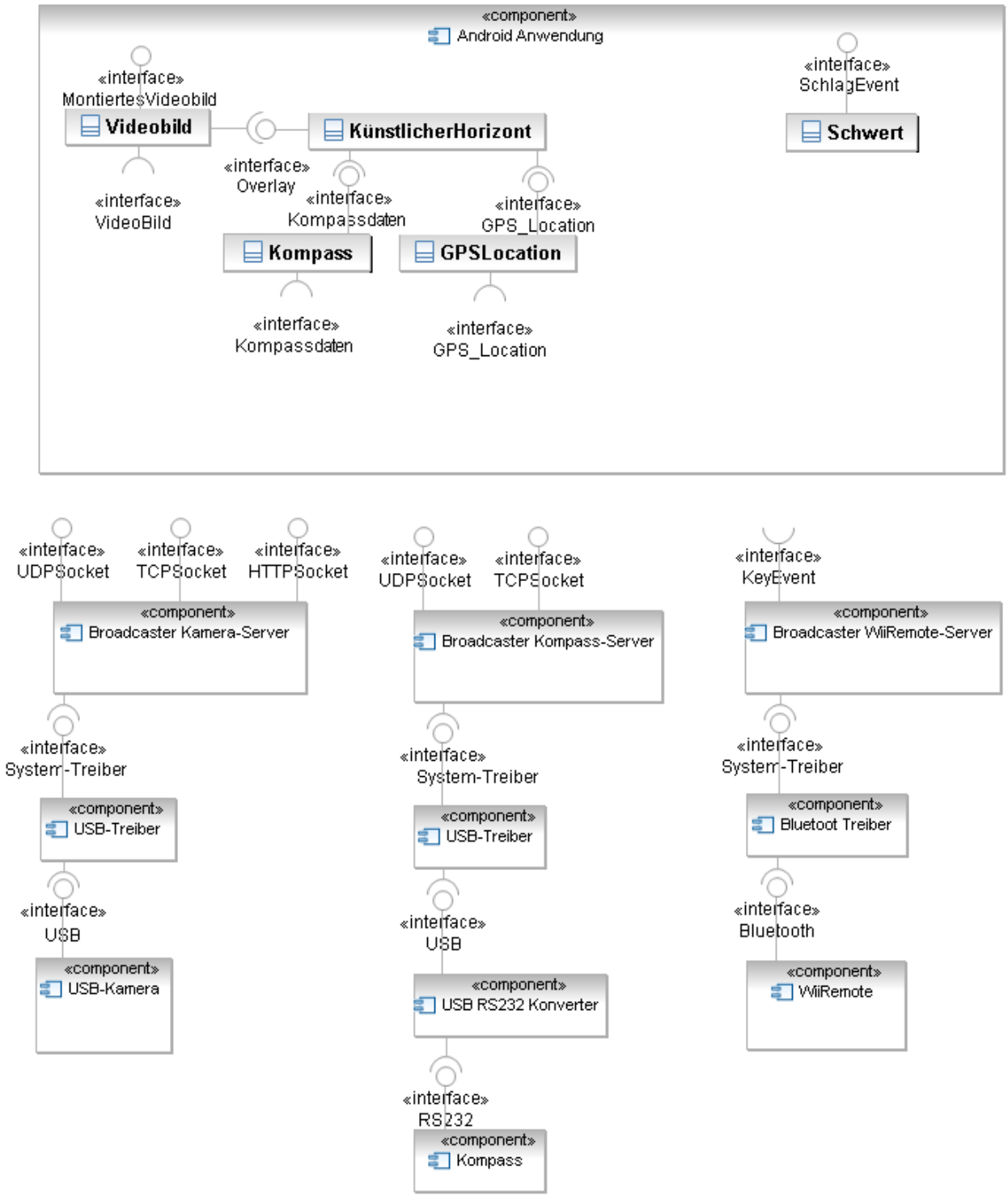


*Abbildung 3: Systemübersicht*