UNIKASSEL VERSITÄT
Fachbereich Elektrotechnik/Informatik
Fachgebiet Softwareengineering
Research Group Software Engineering

Lokales Wlan baisertes Multiplayer Spieleframework für Android

BACHELOREARBEIT

zur Erlangung des Grades eines Bachelore-Informatikers
im Fachbereich Elektrotechnik/Informatik
der Universität Kassel

eingereicht am xx.xx.20xx

bei Prof. Dr.-Ing. Albert Zündorf
Prof. Dr.-Ing. Lutz Wegner
Universität Kassel

von Alexander Gerb
Liegnitzerstr. 6
34123 Kassel

Zusammenfassung

Diese Arbeit umfasst die Implementierung eines Rahmenwerks für Mehrspielerspiele auf Basis des P2P-Rahmenwerks Alljoyn und zwei weiteren Spielen, welche mithilfe des Rahmenwerks erstellt worden sind und als praktische Beispiele dienen. Das Rahmenwerk wird auf Basis des Android SDK realisiert und sollte eine Kommunikation zwischen mehreren Geräten soweit vereinfachen, dass man sich bei der Implementierung weiterer Spiele nur um die Spielmechanik Gedanken machen muss und die Lobby-Funktionalität, wie das Erstellen und das Verbinden zum Spiel vom Rahmenwerk übernommen wird.

Als weiteres Kriterium gilt die Fähigkeit, Spiele im lokalen Wlan sichtbar zu machen und so eine Mehrspielerpartie zu ermöglichen ohne einen dedizierten externen Server, sowie einer Internetverbindung zu benötigen. Es sollte zum Spielen nur mindestens zwei Androidgeräte benötigen, die sich im selben Netzwerk befinden.

Schlagwörter: Multiplayer, Android, Lokal, Peer-to-Peer

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig angefertigt und mich fremder Hilfe nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß veröffentlichtem oder unveröffentlichtem Schrifttum entnommen sind, habe ich als solche kenntlich gemacht.

Kassel, xx.xx.20xx

Alexander Gerb

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1 Einleitung

Spiele auf mobilen Endgeräten gibt es unzählige. Jedoch sind die meisten dieser Spiele entweder Einzelspielerspiele oder sind Serverbasierte Mehrspielerspiele, deren Server nur über eine aktive Internetverbindung erreicht werden kann. Trotz der mittlerweile gut ausgebauten Netzabdeckung und guter Internetgeschwindigkeit auf Smartphones, kann das Spielerlebniss durch eine schwankende Internetgeschwindigkeit getrübt werden, vorallem dann wenn eine volle 3G Geschwindigkeit nicht garantiert ist. So sind z.B. Actionspiele meist nur über eine Hotspot Verbindung spielbar. Außerdem gibt es Situationen bei denen man keine oder nur eine beschränkte Internetverbindung hat, z.B. wenn man mit dem Zug unterwegs ist oder sich gerade in Ausland befindet und sich die Roaminggebühren sparen will. Gerade in solchen Situationen ist die Möglichkeit miteinander eine Mehrspielerpartie zu starten ohne eine Internetverbindung zu haben sehr willkommen. Um die Entwicklung solcher Spiele vorran zu treiben und es den Entwickler soweit wie möglich zu erleichtern, sollte im Rahmen dieser Arbeit ein Rahmenwerk entwickelt werden, welches im Folgenden als PTP-Rahmenwerk genannt wird. Das PTP steht als Abkürzung von Peer-to-Peer, welches als Prinzip noch genauer erklärt wird. Es sollen die kommunikatonsbedingen Herausforderungen mit dem PTP-Rahmenwerk gelöst werden, sodass die Entwickler sich nur um die Implementierung der Spiele selbst gedanken machen müssen. Die Entwickler hätten nur die Aufgabe sich um die Synchronisierung der spielbezogenen Daten zu kümmern und die Übertragen der Daten an die einzelnen Gerät würde sozusagen vom PTP-Rahmenwerk übernommen. Damit die Entwicklung solcher Spiele für die Hobbyentwickler keine finanziellen Hürden stellt, wurde das PTP-Rahmenwerk mit dem Android SDK entwickelt. Da das Android SDK kostenlos für jeden Entwickler zur Verfügung steht, kann jeder Entwickler, der über ein Android Gerät verfügt gleich mit der Entwicklung des Spieles loslegen. Natürlich lässt sich das PTP-Rahmenwerk auch für praktische Anwendungen, wie z.B. Textmessanger, verwenden, jedoch wird im Rahmen dieser Ausarbeitung nur auf die Realisierung von Spielen eingegangen.

1.1 Ziel und Aufgabenstellung der Arbeit

Ziel dieser Arbeit ist es die Lokalisierung und Verbindung der Geräte vom PTP-Rahmenwerk übernehmen zu lassen. Somit hat der Entwickler sich nur um die Implementierung der Schnittstellen-Objekte zu kümmern über die die Kommunikation abläuft. Das PTP-Rahmenwerk soll auf dem Android SDK aufbauen, da es sich hierbei um eine kostenlos verfügbaren Entwicklungsframework handelt. Weiterhin sind zwei Spiele als praktische Beispiele mithilfe dieses PTP-Rahmenwerkes zu implementierung um zum einen die Vorgehensweise zu demonstrieren und zum Anderen die Realisierbarkeit zu testen.

1.2 Lösungsweg der Aufgabenstellung

Um sich die ganzen Herausforderung bei der Realisierung Endgerät-zu-Endgerät Kommunikation zu ersparen wurde ein weiteres Rahmenwerk namens AllJoyn verwendet, welches die die Verbindung und die Kommunikation zum größten Teil schon übernimmt. Somit blieb zum Einen eine geschickte Integration von AllJoyn in das PTP-Rahmenwerk, weiterhin mussten die vielen, gerade auf den ersten Blick komplizierten, Konfigurationen des AllJoyn Rahmenwerks durch eine einfache Schnittstelle erweitert werden. Außerdem musste ein Multithreaded Handler-System verwendet werden um UI-Prozesse des Spiels nicht durch die Hintergrundprozesse der Gerätekommunikation zu behindern. Als praktische Anwendungsbeispiele wurde ein Echtzeitpuzzlespiel und ein Turnbased Kartenspiel namens MauMau implementiert.

1.3 Gliederung

Die Arbeit beinhaltet zu Anfang die Erklärung der verwendeten Rahmenwerke AndroidSDK und AllJoyn. Dabei werden die Konzepte dieser Rahmenwerke grob beschrieben. Als nächstes wird das Konzept und die Impelementierung des Multiplayer Spieleframeworks, welches das Hauptaugenmerk dieser Arbeit ist, beschrieben. Daraufhin wird beschrieben wie die zwei Spielebeispiele realisiert wurden, sowie Herausforderung, die während der Implementierung entstanden. Zum Schluss gibt es noch eine theoretische Auseinandersetzung mit dem Prinzip der Wlan basierten P2P-Spiele.

2 Grundlagen

2.1 Konzepte

Das PTP-Rahmenwerk basiert auf dem Peer-to-Peer Prinzip und benötigt deswegen keinen weiteren Server. Weil durch einen Serverbasierten Ansatz der Zustand des Spiels nur auf dem Server organisiert wird ist es eine gewisse Herausforderung bei einem Peer-to-Peer Ansatz alle Gerät synchron zu halten. Der Entwickler hat nur Objekte, über die er die Kommunikation realisieren muss. Dabei handelt es sich zum einen um Objekte, die die eingehenden Signale behandeln und zum Anderen Objekte um Nachrichten an die anderen Spieler zu senden.

Jedes der Geräte hat zum einen Handler sowie einen Emitter. Der Emitter ist das Objekt welches die Signale an andere Geräte rausschickt und der Handler ist somit das Objekt, welches auf jedem Gerät die eingehende Nachricht behandelt. In den meisten Anwendungsfällen haben die Emitter und Handler auf allen Geräten die selbe Impelementierung. Zusätzlich wird eine Session verwendung um die einzelnen Spielsitzungen oder Spielrunden in sich geschlossen zu handhaben. Somit erstellt ein Host eine Session zu der sich die die Klienten verbinden können um an der Spielrunde teilnehmen zu können. Dadurch wird außerdem die Möglichkeit gegeben mehrere Spiele simultan im selben Netzwerk zu spielen.

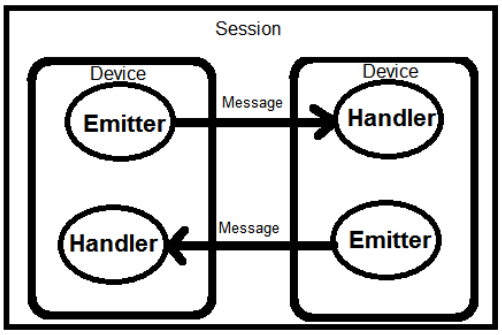

Abbildung 2.1: Kommunications-Konzept

2.2 Peer-to-Peer

Peer ist das englische Wort für Gleichstehender oder Gleichberechtigter, somit ist das Prinzip eines Peer-to-Peer Netzwerk, dass alle Teilnehmer gleichberechtigt sind. Da es bei dem Peer-to-Peer Netzwerk im Grunde keinen Server gibt, welcher sonst die Steuerung der Prozesse und der Ressourcen übernimmt, müssen die einzelnen Peers sich selbst darum kümmern und sich entsprechend selbst organisieren.

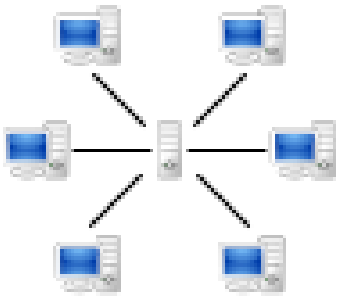

Abbildung 2.2: Server basiertes Netzwerk

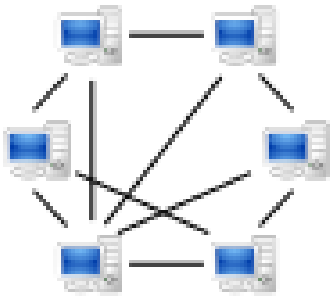

Abbildung 2.3: PtP basiertes Netzwerk

Die Bilder verdeutlichen nochmal das Gesagte. Durch das Weglassen eines fest zugeordneten Servers, erhält man eine Flexibilität, sodass nur Peers benötigt werden um das Spiel zu spielen. Es ergeben sich auch Probleme bei dieser Art der realisierung, die man mit einem Serverbasiertem system kaum hätte. Normalerweise bei einem Serverbasiertem Spiel wird der Zustand des Spiels auf dem Server geregelt und alle zu dem Server verbundenen Geräte brauchen den Server nur nach diesen Zustand zu fragen. Da die Berechnungen meistens auf dem Server liegen werden die verbundenen Geräte entlastet. Bei einem Peer-to-Peer System hingegen muss der Zustand von jedem einzelnen Gerät ermittelt werden und mit den anderen Geräten synchonisiert werden. Dies hat einen höheren Ressourcenaufwand und limitiert somit die Anzahl der möglichen Teilnehmer, da die Informationen mit allen Peers ausgetauscht werden um zu garantieren, dass alle den selben Zustand haben. Jedoch ist es auch sehr vom Anwendungfall abhängig, denn ein entsprechendes Gegenbeispiel für eine hohe Peersanzahl zeigten Tauschbörsen wie Torrent oder Napster um nur zwei bekannte Beispiele zu nennen. Somit liegt die Herausforderung bei den Entwicklern, da sie zum einen Entscheiden müssen wie viele Spieler maximal zulässig sind und somit auch die Anforderung für die Komplexität des Spiel setzen.

2.3 Android SDK

Android ist das Linux-basierte Betriebssystem für mobile Endgeräte, welches von Google 2011 offiziell zur Verfügung gestellt wurde. Android selbst gilt als sogenannte frei Software, welche bis auf den System-Kern unter der Apache-Lizens steht. Diese Tatsache unter Anderen ermöglichte eine rasante Verbreitung dieses Betriebsystems auf vielen Geräten unterschiedlicher Hersteller.

Abbildung 2.4: Android

Somit waren im März 2013 etwa 750 Millionen Android End-Geräte aktiviert und man merkt schnell, dass die Popularität dieses Betriebssystem immer mehr zunimmt. Da ein Smartphone Betriebsystem auch von den angeboten Apps lebt, hat Google eine Entwicklungswerkzeugsammlung zur Verfügung gestellt, welche die Entwicklung von Applikationen für Android möglichst einfach ermöglichen soll. Bei dieser Werkzeugensammlung handelt es sich um das Android SDK, welches auch als das Android Developer Tool kurz ADT verfügbar ist. ADT ist ein Plugin für das mittlerweile sehr weit verbreitete Entwicklungsumgebung Eclipse, welche die Entwicklung und die Übertragung der Applikation auf das Gerät mühelos ermöglicht. Weiterhin bringt das Android SDK einen Emulator mit sich, welches das Testen von Apps unter unteschiedlichen Konfigurationen ermöglicht ohne dass man ein Android-Gerät benötigt. Vorallem ermöglicht das Android SDK die Entwicklung der Apps in der Programmiersprache Java, welche sich immer höherer Beliebtheit erfreut und dank Eclipse zu einer höheren Produktivität beiträgt. Als nächstes gehe ich auf die einzelne Grundlagen von Android SDK ein um die Funktionalität dieser zu beschreiben.

2.3.1 Activity

Eine Activity ist eine Klasse, welche die Erstellung von einzelnen UI-Fenstern übernimmt. Somit bestehen die meisten Apps in Android aus mehreren Activities, welche mit einander verbunden sind. Ein Use-Case bei dem man mehrere Fenster hat, würde sozusagen mehrere Activities nacheinander aufrufen. Um eine Activity zu erstellen muss seine Klasse eine Unterklasse von Activity sein. Zusätzlich muss man die Methode ***onCreate()*** in seiner Klasse überschreiben. Diese Methode wird jedesmal aufgerufen wenn die Activity erstellt wird, also auf dem Bildschirm erscheinen soll. In diese Methode kommen Aufrufe von Fenstern, die die UI beinhalten, oder andere

Operationen die beim Start notwendig sind.

2.3.2 Service

Ein Service ist eine Komponente, welche dazu gedacht ist Hintergrundprozesse zu übernehmen. Ein Service wird von z.B. einer Activity gestarted und läuft dabei im Hintergrund, selbst wenn die Activity nicht mehr existiert. Somit bietet sich ein Service gut an um z.B. die Netzwerkkommunikation im Hintergrund zu behandeln ohne die Applikation selbst zu behindern. Weiterhin lässt sich ein Service auch an z.B eine Applikation binden, sodass der Service auch beendet wird wenn die Applikation geschlossen wird.

2.3.3 Application

Application bietet zusätzlich zu den Activities die Möglichkeit während der ganzen Laufzeit der Applikation eine feste Instanz zu haben, welche den Zustand bestimmter Daten beinhaltet. So kann man es im Prinzip mit einem Singleton vergleichen der den Status der Applikation hält. Um an die Instanz zu kommen muss man aus dem Kontext heraus die Methode ***Context.getApplicationContext()*** aufrufen.

2.3.4 Context

Der Context beinhaltet Informationen über die Applikationsumgebung und lässt verschiedene Aktionen zu, wie z.B. das Aufrufen von weiteren Activities. Eine Activity ist eine Unterklasse vom Context und wird bei der Erstellung von z.B. einer View an diese übergeben.

2.3.5 View

Eine View representiert eine Sammlung von UI-Elementen auf einem Bildschirm. Die Elemente werden meist über das XML-Layout erstellt und darüber referenziert. Activities haben so die Möglichkeit eine Unterklasse der View als Instanz aufzurufen, welche das UI-Fenster representiert oder können die UI-Elemente direkt über das XML-Layout laden.

2.3.6 UI-Thread

Der UI Thread ist der Hauptthread, welcher beim Start der Applikation gestartet wird. Dieser Thread ist für die Darstellung von UI-Elemente verantwortlich sowie auf Benutzeraktivitäten wie Touchevents zu reagieren und sie zu verarbeiten.

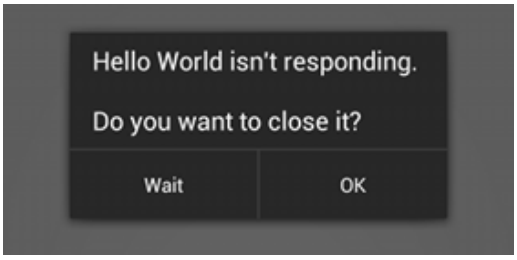

Abbildung 2.5: ANR

Somit ist es wichtig langwierige und blockende Aufgaben wie die Netzwerkkommunikation in ein seperaten Thread auszulagern um den UI Thread nicht zu überlasten Denn ein blockierter UI Thread kann schnell zu einer sogenannten ANR-Meldung führen oder auf deutschen Geräten 'Anwendung reagiert nicht mehr'. Außerdem muss man bei Verwendung von mehreren Thread darauf achten, dass die UI Komponenten nur vom UI Thread angefasst werden dürfen. Somit muss eventuell bei Background Thread ***runOnUiThread()*** Methode verwendet werden, welche die auszuführende Aufgabe an den UI Thread übergibt.

2.4 AllJoyn

Abbildung 2.6: AllJoyn

AllJoyn ist ein Open-Source Peer-to-Peer Rahmenwerk welches es erlaubt Verbindungen zwischen verschiedenen Geräten herzustellen. AllJoyn wird von Qualcomm Innovation Center Inc. entwickelt und steht unter der Apache v.2 Licenz. Besonderer Augenmerk dieses Framework besteht in der Tatsache, dass es verschiedene Betriebssysteme unterstützt und eine Vielzahl von Programmiersprachen, darunter C♯, C++, Java sowie Objective C. Somit lassen sich Anwendungen auf Peer-to-Peer Basis auf Windows,MacOS,Linux,Android und iOS entwickeln und mit einander verbinden. Es bietet weiterhin die Unterstützung für Bluetooth, Wifi und Ethernet. Vorallem ermöglicht AllJoyn die Verbindung von Mobilen-Endgeräten im Wifi-Netz ohne sich selbst um das Finden und Verbinden kümmern zu müssen. Außerdem bietet AllJoyn für jedes Betriebssystem ein SDK an, welches alle notwendigen Bibliotheken sowie einige Beispielanwendungen beinhaltet.

2.4.1 Bussystem

Das Prinzip von AllJoyn basiert auf einem Bussystem zu dem sich einzelne Anwendung verbinden können. Jedes dieser Anwendung muss einen sogenannten BusAttachement erstellen und den entsprechenden EventHandler implementieren über den dann die Nachrichten von anderen Anwendungen behandelt werden. Über einen ProxyObjekt lassen sich dann entweder Methoden auf einem bestimmten Gerät aufrufen oder auf allen Geräten, die zu dem Bus verbunden sind und das entsprechende Interface implementieren.

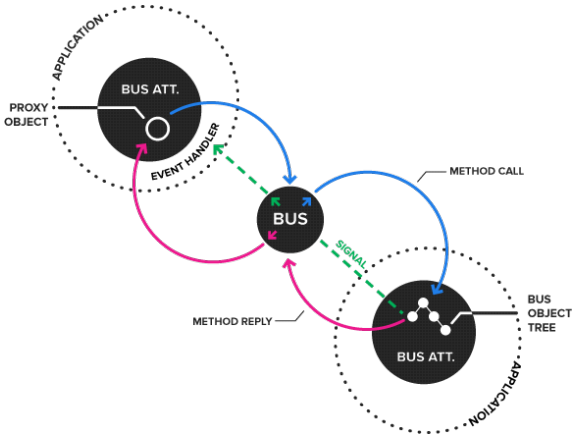

Abbildung 2.7: Bussystem

2.4.2 BusAttachement

Um mit dem Bussystem zu kommunizieren, erstellt man einen BusAttachement:

```
1 bus = new BusAttachment("name",BusAttachment.RemoteMessage.Receive)
```

Der “applicationName” Parameter ist notwendig um verschiedene BusAttachements zu den entsprechenden Anwendungen zu zuordnet, falls mehrere Anwendungen über den selben Bus kommunizieren. Als Zweitparameter lässt sich noch angeben ob das BusAttachement eingehende Nachrichten verwerfen oder behandeln soll.

2.4.3 Interface

Die Kommunikation bei AllJoyn funktioniert über Interfaces, also eine konkrete Beschreibung der Methoden über die Nachrichten verschickt werden. So muss zum einen über dem Interfacenamen die folgende Annotation stehen:

```
1 @BusInterface (name = "de.package.MyInterface")
2 public interface MyInterface{}
```

Das Interface wird benötigt um über das BusAttachement an die ProxyObjekt zu kommen. Weiterhin muss das Inteface auch die Methoden entsprechend deklarieren.

```
1 @BusMethod
2 public void MyMethod() throws BusException;
```

2.4.4 ProxyObjecte

Bei Proxy Objekten handelt es sich um Objekte die dazu benutzt werden um Methoden auf Objekten anderer Geräte oder Anwendungen aufzurufen. Sie implementieren das zuvor definierte Interface, worüber dann die Kommunikation realisiert wird.

2.4.5 BusObject

Ein BusObject ist ein Objekt welches das definierte Inteface implementiert und unter einem bestimmten Pfad abgespeichert wird.

```
bus.registerBusObject(busObject, "/objectPath");
```

Somit können alle Anwendungen über den Pfad auf das Bus-Objekt zugreifen und darauf Methoden aufrufen oder Properties abfragen.

2.4.6 SignalHandler

Ein SignalHandler ist ein Object welches die Behandlung von eingehenden Nachrichten implementiert. Der Unterschied von einem BusObject und einem SignalHandler ist, dass beim BusObject man nur die Methode auf dem entsprechenden Object ausführt, welcher unter einem bestimmten Pfad gespeichert ist und bei SignalHandler handelt es sich um Objecte die auf Nachrichten reagieren die an alle Geräte geschickt werden wie bei einem Broadcast. Falls es sich bei dem Inteface um einen SignalHandler handelt müssen die Methoden @BusSignal als Annotationen beinhalten

```
@BusSignal
public int MyHandler() throws BusException;
```

Zusätzlich muss man bei der Implementierung eine weitere Annotation hinzufügen um den Nachrichtentyp explizit zu definieren.

```
@Override
@BusSignalHandler(iface = "de.packege.MyInterface", signal = "MyHandler")
public void MyHandler() throws BusException {}
```

2.4.7 SignalEmitter

Ein SignalEmitter ist das Objekt das dazu verwendet wird Nachrichten an alle Teilnehmer zu senden. Der SignalEmitter kann dazu verwendet werden um an das Inteface zu kommen welches auch von den SignalHandlern implementiert wird. Es ist im Prinzip der Proxy zu allen Teilnehmern. Den SignalEmitter bekommen man durch den folgenden Aufruf:

```
emitter = new SignalEmitter(busObject, id, SignalEmitter.GlobalBroadcast.Off);
myInterface = (MyInterface) emitter.getInterface(MyInterface.class);
```

Bei der id handelt es sich um die SessionID, welche beim Verbinden zu einer Session mitgeteilt wird. Zusätzlich lässt sich noch konfigurieren ob das Signal auch über den Bus hinaus weitergeleitet wird, falls jemand zu mehreren Bussystemen verbunden ist. Als Standartkonfiguration ist die Signalweiterleitung aus.

2.4.8 Session

Eine Session ermöglicht es mehrere Teilnehmer zu einer gemeinsamen Einheit zusammenzufassen um verschiedene Anwendungsabläufe wie z.B. eine Spielrunde von einander getrennt zu handhaben. Dazu muss z.B. der Host einen sogenannten Channel erstellen, wohin sich alle anderen Teilnehmen verbinden können. Dazu benötigen alle Geräte zum einen den Namen des Channels und zum anderen die Channelportnummer. Dazu kann der Host den erstellten Channelname den anderen Teilnehmern, welche sich im Bussystem befinden,über das sogennante Advertising mitteilen. Um diese Mitteilungen auch zu empfangen, müssen die Teilnehmer einen BusListener implementieren, welcher dann auf solche Nachrichten horcht. Somit hat einer der Teilnehmer die Aufgabe des Hosts und erstellt einen Channel und teilt diesen dann den Anderen mit:

```
1 bus.bindSessionPort(contactPort, sessionOpts, sessionPortListener);
2 bus.requestName("myChannelName");
3 bus.advertiseName("myChannelName");
```

Das Erstellen der Session benötigt unter einigen Einstellungen, wie die Portnummer,Transportprotokol usw, auch den SessionPortListener, welcher z.B. das Verbinden von anderen Teilnehmern behandelt und bestimmte Aktionen dann ausführen kann. Typische Methoden eines SessionPortListeners wären z.B.

```
1 public boolean acceptSessionJoiner(short sessionPort,
2                String joiner, SessionOpts sessionOpts) {}
3
4 public void sessionJoined(short sessionPort, int id, String joiner) {}
```

Der Host braucht dann die Implementierung der Methoden um das Verbinden zu der Session zu Kontrollieren.

3 Implementierung

Es gibt viele verschiedene Spielprinzipien und daher sollte soweit es geht mit dem PTPRahmenwerk den späteren Entwicklern so viel Freiheit gelassen werden wie möglich. Jedoch gibt es viele Abläufe, die sich immer wieder wiederholen. So hat man den typschen Ablauf, dass man das Spiel startet und man sich in einer Lobby befindet. Danach kann man entweder ein Spiel erstellen und sich zu einem bereits erstellten Spiel verbinden. Diesen Ablauf wird durch das PTPRahmenwerk übernommen, sodass man sich hinterher darum nicht mehr kümmern muss. Der Entwickler hätte dann nur noch die Aufgabe sich um die Activities zu kümmern, die das Spiel representieren und die Verbindung und Vermittlung würde das PTPRahmenwerk übernehmen. Um das Framework benutzen zu können müsste der Entwickler dann nur noch eine Jar-Datei als Bibliothek in sein Projekt einbinden und er könnte loslegen.

3.1 PTPHelper

Nach Abwägen mehrerer Möglichkeiten wie man den Service, welcher sich um die Verbindungstechnischen Abläufe kümmert, in die Applikation integrieren kann, ist die Wahl schließlich auf die Benutzung eines Singletons gefallen. Zwar ließe sich eine Abstrakte Application benutzen, die der Entwickler in seinem Projekt implementieren müsste, aber dies würde dem Entwickler auch eine Architekturentscheidung aufzwingen. Außerdem ist eine zu enge Bindung an das Android Rahmenwerk unvorteilhaft, da sich dessen API recht oft ändert und die Entwicklung in der Zukunft erschweren kann. Durch das Singleton kann der Entwickler entscheiden, wann er den Service startet und wie er damit interagieren will. So kann er z.B. den Einzelspielermodus vollkommen ohne den Helper realisieren und erst bei Multiplayermodus den Service starten. Um den PTPHelper zu initialisieren muss der Entwickler in seinem Code den Folgenden Befehl ausführen:

```
1 PTPHelper.initHelper(MyInterface.class, context, proxyObject, signalHandler,
2 MyService.class, MyLobby.class);
```

Daraufhin kann er über den Getter an die initialisierte Instanz kommen.

```
PTPHelper.getInstance();
```

Die P2PHelper Klasse ist eine generische Klasse und muss deshalb bei der Instanziierung den entsprechenden Typ bekommen. Daher auch die etwas umfangreiche Init-Methode. Es wird zum einen die Interface Klasse benötigt über das die Kommunikation mit den anderen Peers realisiert wird. Bei der Initialisierung wird der P2PService im Hintergrund gestartet welcher die Verbindung zum Bussystem aufbaut. Das PTPRahmenwerk ist dazu ausgelegt nur im WLAN zu funkionieren, somit wird bei der Initialisierung geprüft ob das Gerät eine WiFi Verbindung momentan hat. Dies wird über den WiFiManager realisiert.

```
WifiManager wifiManager =(WifiManager)context.getSystemService(Context.
    WIFI_SERVICE);
WifiInfo currentWifi = wifiManager.getConnectionInfo();
if((currentWifi==null || currentWifi.getSSID()== null || currentWifi.getSSID().
    isEmpty()){
  //Show Message there is no WiFi-Connection
}
```

Im Falle dessen, dass es keine WiFi-Verbindung gibt, wird die Initialisierung abgebrochen und der Service wird nicht gestartet. Android ermöglicht es zusätzlich ein AccessPoint zu erstellen, worüber sich andere Geräte verbinden können. Die Information über diesen Zustand wird jedoch nicht ohne weiteres über den WiFi-Mananger herausgegeben, sodass man an diese Information nur über Java-Reflection gelangt:

```

Method method = wifiManager.getClass().getMethod("isWifiApEnabled");
state = (Boolean) method.invoke(wifiManager);
```

Nach dem jedoch der Helper initialisiert ist und der Service gestartet, kann man über den Helper den Service mitteilen sich z.B. zu einem Channel zu verbinden oder einen Channel zu erstellen. Dies geschickt über einfache Methodenaufrufe:

```
PTPHelper.getInstance().setHostChannelName(name)
PTPHelper.getInstance().hostStartChannel()
...
PTPHelper.getInstance().setClientChannelName("channelName");
PTPHelper.getInstance().joinChannel();
```

3.2 Graphenspiel

3.3 MauMau

4 Praktisches Anwendungsbeispiel: Graphenspiel

5 Praktisches Anwendungsbeispiel: MauMau

6 Herausforderungen und Probleme

7 Fazit

A Abkürzungsverzeichnis

B Literaturverzeichnis

[Mus09] Mustermann, Max: *Titel. Untertitel.* Auflage. Verlagsort: Verlag, Jahreszahl (= Reihe).

[Lor09] Ebers, Robin (2009): „Lorem Ipsum“. URL: `http://de.lipsum.com` [Stand: 11.10.2009]