“Ci sono voluti mille anni alla natura per creare una testa come quella di Lavoisier e cinque secondi all’uomo per decidere di tagliarla...”

Pierre-Simon, marquis de Laplace (8 Maggio 1794)

(alla notizia della condanna a morte di Antoine Lavoisier)

Simulating a Post Disaster Scenario Through a Collaborative Peer-to-Peer App for Mobile Devices

Albijon Hoxhaj

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Laurea Specialistica
of the
Universita' degli Studi di Padova.

Department of Computer Engineering

October 15, 2012

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: October 15, 2012

Abstract

In the wake of major disasters, the failure of existing communications infrastructure and the subsequent lack of an effective communication solution results in increased risk, inefficiencies, and damages to the people. One way to address this problem is to develop a distributed peer-to-peer system for mobile devices that relies on local communication such as Bluetooth technology. The special requirements of mobile devices and networks necessitate the elaboration and the adoption of different solutions in order to fulfill the expectations which arise with the use of mobile peer-to-peer technology.

In this project we simulate a post-disaster scenario via a game app for mobile devices, and, we present a peer-to-peer collaboration algorithm through the Bluetooth network to help players in such a scenario.

Acknowledgements

Firstly, I would like to thank my family and my fiancee for all their support during these 7 months in Aberdeen.

Secondly, I would like to thank my supervisors Prof. Carlo Ferrari and Prof. Wamberto Vasconcelos for their support over the last months.

Finally, I would like to thank Prof. Roly Lishman and Dr. Shona Pots for their help to extend my Erasmus period in Aberdeen.

Contents

List of Figures

Chapter 1

Introduction

In the past years centralized client-server networks were transformed to distributed peer-to-peer networks. Nowadays, mobile devices are everywhere and lessons learned from fixed networks are being applied in mobile networks. These personal devices are used for interpersonal communications (phone call, SMS) but they are also increasingly used to access the internet or to share contents they are now able to produce: contextual information, multimedia documents, etc. Indeed, mobile devices can now be considered as multimedia content production endpoints because they have a camera and microphone.

These new features provided by the last mobile devices make them suitable to help in rescue operations after a disaster event. An example is the Haiti Earthquake disaster that occurred back in 2010 with an approximate death toll of over 300,000 people. It is clear that these types of natural disasters can cause chaos on the ground with devastating knock on effects hampering rescue attempts.

Figure 1.1: Haiti after the earthquake.

However, the mobile internet world is a much more controlled and constrained environment. Limitations come from the capacity of the terminal, from the connectivity technologies available on the device or from restriction policies applied by mobile network operators. In order to circumvent these constraints, one can imagine developing an alternative to centralized network models in order to give the user the possibility to collaborate with other users nearby. The best way to do it is to set up a peer-to-peer platform over Bluetooth.

In this project we model a post-disaster scenario in a simple way and propose a collaborative P2P game for Android devices in order to incentive the distribuiton of information between players. It involves tackle challenges due to constraints present in mobile devices and wireless networks such as:

- Memory
- Processing power
- Network accessibility such as the problems related with low bit rates, high latency, packet losses, temporal disconnections, etc.
- Battery consumption
- Mobility issues

1.0.1 Beneficiary Survay

Due to the nature of the project, there were no specific potential users that could be targeted in order to obtain potential feedback. Instead, it was decided to contact users who would become a beneficiary of the application, reaping benefits of what could be provided in the future.

According to [Buchan] details on the project were sent to a number of different police forces and fire rescue services around the UK to ask for feedback on what was being proposed. A response was received from an Inspector within the Police Service of Northern Ireland (PSNI) 4, as follows:

"It seems a useful topic to explore particularly as over and above the situations you illustrate, there are geographical considerations also, where perhaps the signal in a particular area is weak, thereby increasing the chances of a momen- tary breakdown in communication. With the system you propose, this risk is likely to be minimised. Any system that will aim to enhance the service we provide and lessens risk to staff using the system is worthy of consideration."

They also highlighted that they would be willing to assist further with the project if they could do so.

Chapter 2

Related Work

In the past years, we have seen an explosion of new collaborative work supporting systems for PDAs, smartphones and other mobile devices. Applications allowing users to collaborate in real time using wireless connected mobile devices building ad-hoc networks have attracted the attention of many authors. Some of the scenarios for which these applications have been developed are the following

- Educational activities involving group of students and teachers in collaborative room environments
- Mobile devices have been used to mediate between healthcare personal or to support the relationship between therapists and patients in a hospital
- Group of people attending a meeting can share ideas and data by means of their mobile devices
- Field survey operations in remote areas with no fixed infrastructure can be easily facilitated.
- Field survey operations in remote areas with no fixed infrastructure can be easily facilitated.
- Members of an organization can register and share their knowledge about important processes in a more flexible and informal way with a mobile knowledge management system.

2.1 Mobile Peer-to-Peer Content Sharing Application

Mobile Peer to Peer Content Sharing Application[Matuszewski et al.(2006)] is an innovative proposal of an architecture of mobile peer to peer content sharing services in cellular networks developed by the Nokia Research Center and Helsinki University of Technology. This approach uses the SIP protocol as a basis for the deployment of mobile P2P services. The implementation consists of a peer to peer client application in the mobile phone and an application server in the network. The mobile peer to peer client was implemented on the Nokia Series 60 (Symbian platform). This solution presents a hybrid architecture with peers and super-peers.

2.2 Mobile Chedar

Mobile Chedar[Wang et al.(2007)] is an extension to the Chedar peer-to-peer network allowing mobile devices to access the Chedar network and also to communicate with other Mobile Chedar peers. Chedar (CHEap Distributed ARchitecture) is a peer-to-peer middleware designed for peer-to-peer applications. In this project Chedar has been extended to the mobile platform as Mobile Chedar. Mobile Chedar is implemented using Java 2 Micro Edition (J2ME), and uses Bluetooth as a transmission technology for connecting to other peers. Current Bluetooth implementations have a restriction that nodes can be connected to only one piconet at a time. Therefore the only topology available for constructing Bluetooth network is starshaped. One device functions as a master and others as slaves.

2.3 Sony Playstation Vita

The Playstation Vita is a handheld device that was released by Sony6 in early 2012 which brings a new slant on sharing by integrating a number of different wireless protocols into the system, such as Bluetooth, 3G and Wi-Fi introducing new peer-to-peer style functionality to provide a number of new features to the users. Using these wireless technologies, the application allows players to find other players near to them, play against them or even share their latest scores with them to create a localised leaderboard using what is in essence a peer-to-peer to network having no central server.

Figure 2.1: Sony Playstation Vita

2.4 WORKPAD Project Overview

The WORKPAD project[1]aims at designing and developing an innovative software infrastructure (software, models, services, etc.) for supporting collaborative work of human operators in emergency/disaster scenarios. In such scenarios, different teams, belonging to different organizations, need to collaborate with one other to reach a common goal; each team member is equipped with handheld devices (PDAs) and communication technologies, and should carry on specific tasks. In such a case we can consider the whole team as carrying on a process, and the different teams (of the different organizations) collaborate through the exchange of integrated data and content provided by some back-end centres, thus supporting inter-organizational coordination (macro-processes).

The project investigates a 2-level framework for such scenarios: a back-end peer-to-peer community, providing advanced services, data knowledge content integration, and a set of front-end peer-to-peer communities, that provide services to human workers, mainly by adaptively enacting processes on mobile ad-hoc networks.

2.4.1 WORKPAD Architecture

Two classes of users were identified: Back-end and Front-end users. The identification is based on the Consortium's understanding of how Civil Protection works in Italy and other countries and on the collected user requirements. From an organizational perspective, front-end includes several teams of rescuers that are sent to area in order to manage an emergency, whereas back-end includes the control rooms/headquarters of the diverse organizations that have rescuers involved at front-end. These control rooms provide instructions and information to front-end teams to support their work.

Typically, control rooms are provided with servers whose data need to be integrated in order to provide a unified view over the available information. At front end, every team is headed by a "leader operator", who coordinates the intervention of the other team members. [Catarci et al.(2008)]

[1] `http://www.dis.uniroma1.it/~workpad/index.html`

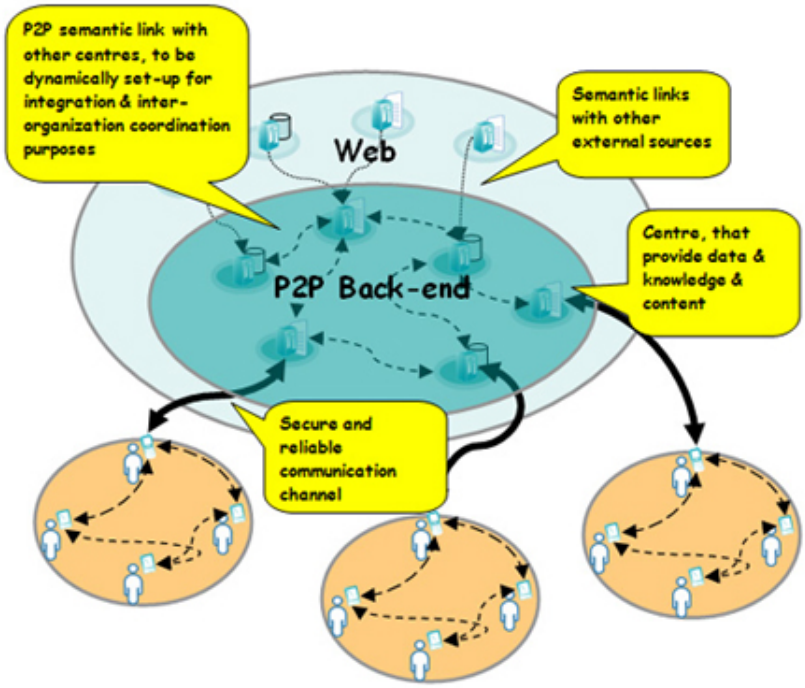

Figure 2.2: WORKPAD architecture.

2.5 LifeNet Project

In the wake of major disasters, the failure of existing communications infrastructure and the subsequent lack of an effective communication solution results in increased risk, inefficiencies, damage and casualties. Current options such as satellite communication are expensive and have limited functionality. A robust communication solution should be affordable, easy-to-deploy, require low-to-zero infrastructure, consume little power and facilitate Internet access.

LifeNet [2]is a WiFi-based data communication solution designed for post-disaster scenarios. It is open-source software and designed to run on consumer devices such as laptops, smart-phones and wireless routers. LifeNet is an ad hoc networking platform over which critical software applications including chat, voice messaging, MIS systems, etc. can be easily deployed. LifeNet can grow incrementally, is robust to node failures and enables Internet sharing. A novel multi-path ad-hoc routing protocol present at its core, enables LifeNet to achieve these features.

In disasters situations, infrastructure is prone to failures either by direct destruction or indirectly by factors such as failure of power supply. Additionally, because of these infrastructure requirements, it is infeasible to deploy such communication solutions rapidly.

LifeNet exploits multihop communication to provide coverage over comparable areas with minimal infrastructure. Every device functions both as a host and as a router. Two devices close to each other communicate with each other directly, whereas communication between two far off devices can be relayed by low-power intermediate nodes in a multihop fashion. Infrastructure such as large mounting structures and power suppplies are not required. This facilitates rapid on field roll-out.

[2]`http://thelifenetwork.org/index.html`

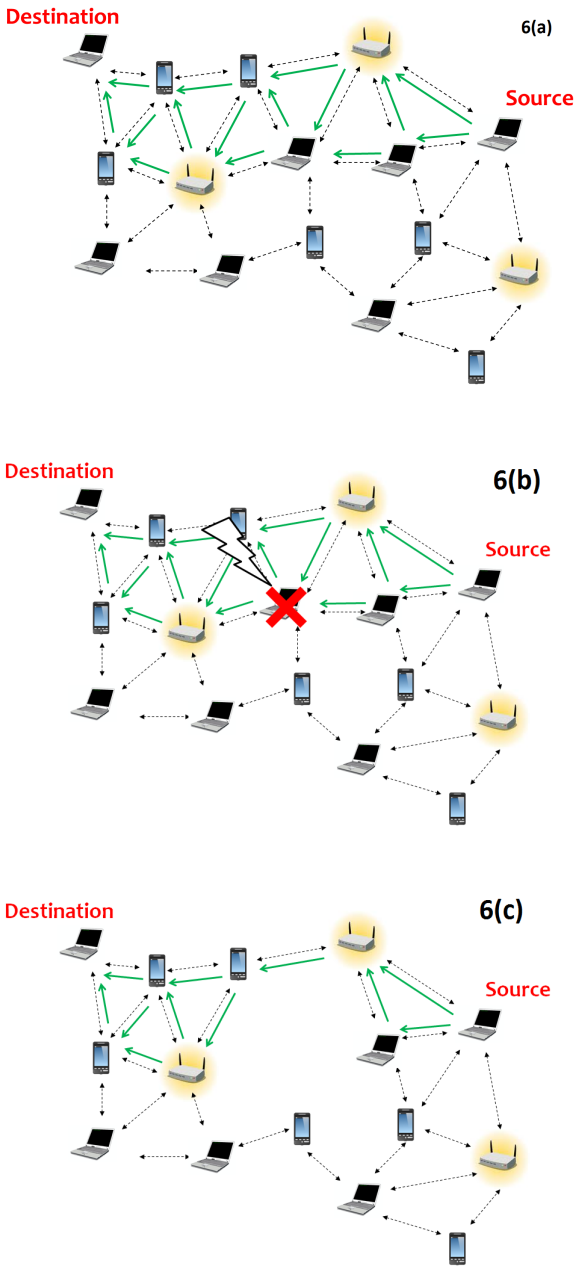

Figure 2.3: Multipath routing algorithm used by LifeNet

As shown in the above figure it is designed to use a multipath routing protocol for communication between devices. This protocol, called “Flexible Routing”, lies at the heart of LifeNet and makes it useful for transient environments. By transience we refer to devices moving in the network, device failures, dynamic network traffic conditions, changing physical obstructions, interference, etc. Disaster relief operations, wireless sensor networks are all highly transient environments. Since the routing protocol, called as ’Flexible Routing’ is capable of delivering packets under varying degrees of transience, it makes LifeNet a promising solution for transient environments. Images 6(b) and 6(c) show the importance of multipath routing in handling node failures.

Chapter 3

Goals, Requirements and Proposed Architecture

In this chapter, we discuss the goals of the project, the requirements and the architecture. We also gave a brief description of each architecture component.

3.1 Goals

The project aim is to implement a peer-to-peer game which simulates an emergency scenario, where people have to reach a safe place (final position) moving through an unknown environment that contains dangerous places (obstacles). The only available help during the route will be offered by neighbor devices, through a peer-to-peer mechanism over Bluetooth. As we already said, the "Game" helps us to attract players for the experiment.

We decided to propose an Implementation for Android mobile devices but it can be easily extended to other operating systems such Apple iOS or Microsoft Windows Phone. The following is the list of the goals:

- Simulate a "post-disaster scenario" via a Game in which the classic telephone network collapse so you can't communicate with a central server in order to ask for, or give information . You don't have complete information about the surrounding environment but you have to move from a "Start Point" to an "End Point" avoiding possible obstacles/dangers. In such a scenario the only available help is to communicate via the Bluetooth network in order to ask/offer information to other devices around you;
- Propose an algorithm for "Information sharing" which will manage the requests and responses generated/received. It should take in account the specific requirements of mobile app development;
- Offer data about each player performance on the game;

3.2 Functional Requirements

After a long analysis on the needed features in order to implement an effective solution to the problem, we identified the following requirements:

1. The App should has an initial GUI utilities screen containing general commands like Start, Continue, About, Exit in which the player chose an option.
2. A graphical user interface in order to play the Game. This will be our representation of the environment in a post-catastrophe scenario. The player shouldn't be able to have a complete view of the surrounding world.
3. both the Graphical User Interfaces should be touch sensitive (implement touch control). The user will guess the next step just by touching one of the neighbors cell.
4. Generate random missions of the same distance for each player.

5. The only way to communicate is via the local Bluetooth network throuugh a peer-to-peer mechanism.

6. Propose an intelligent algorithm to handle incoming and outgoing messages.

7. Provide data collection features. It means provide information about the players performance on the game.

8. The App will be easy accessible in order to involve as many players as possible. This will help to evaluate the game effectiveness.

3.3 Architecture and Components

Figure 3.1 shows ultimate version of the architecture implemented. It's based on specific Android best practices and patterns that can differ from the classical consolidated model of multi-tier architecture.

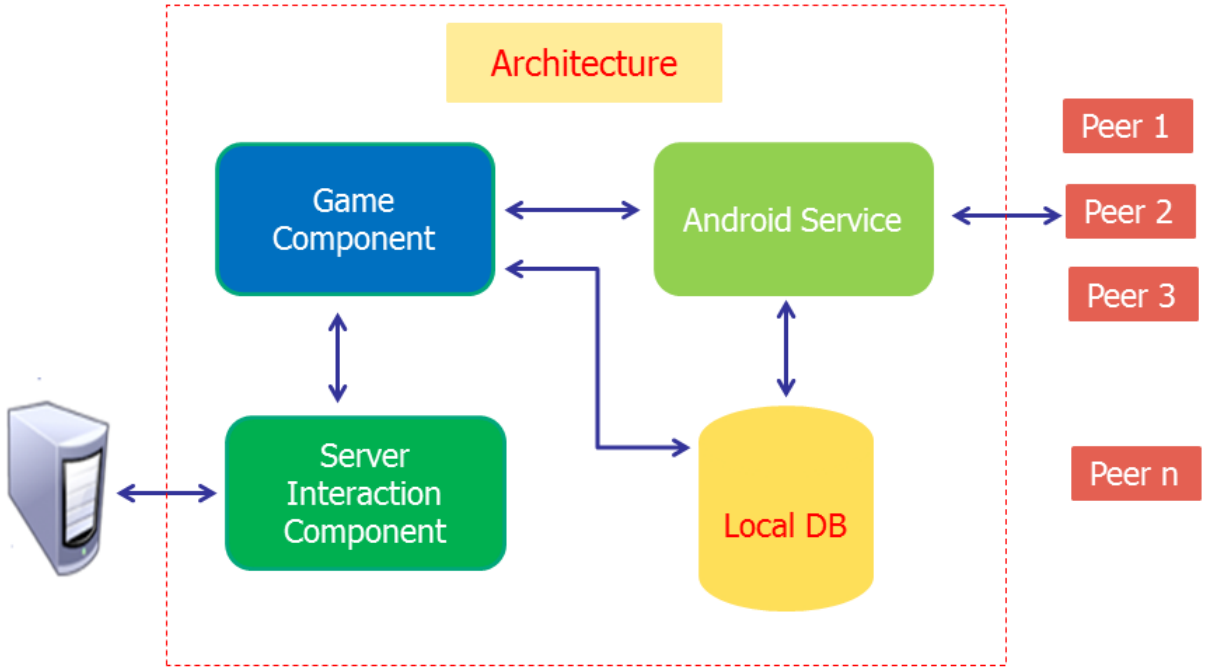

Figure 3.1: Game Architecture

Here follows a brief explanation of each component, further details will be provided in **Chapter 5** "Implementation".

3.3.1 Game Component

The Game component manages all the local game evolving. It is composed by two main parts:

- the **Graphical User Interface**(GUI) which manages the interaction with the player.
- the **Game Logic** which implements our rules and interacts with the Service to request relevant information

With this component we address the first fourth requirements explained above. It provides the two GUI's, implements touch controls and the game's logic.

3.3.2 Game Core Service

The Service (interacting with the Peer to Peer Component) addresses the fifth and the sixth requirements. It runs on a dedicated process and executes in background the following tasks:

- autonomously interacts with other devices through the peer to peer component
- notifies relevant information received to the game component

3.3.3 Peer to Peer Component

It implements the communication with the other peer devices. Its main tasks are:

- discover new devices
- pairs with discovered devices and updates the peers list
- interacts with the Game Core Service

As we said above in team with the Service component they address the fourth and fifth requirements.

3.3.4 Server Interaction Component

It manages the communication between the Game and the Server in a secure way. It offers an easy interface to both Game component and Server component.

3.3.5 DB Layer(Interface)

It offers to Game component and Service component a unique interface to manipulate database data. It manages also concurrent accesses through a locking policy. Together with the Server component, address the seventh requirement, which is:

- keep statistical data in order to evaluate players performance in the game

3.3.6 Server Component

It is responsible to communicate with the game through the Server Interaction Component. Its main tasks are:

- send initialization data to the game which are set by the administrator.
- store on the server database player's performance in each game.

3.4 Database Architecture

To address the need to keep statistical data and offer a consistent user experience, the application needs a server side database and light local database.

The two databases have been designed keeping in mind the two different contexts. As you can easily intuit the local database (Game local DB), has to be light and efficient. No statistical data are kept in it about precedent games played. It contains just the necessary data to the current game in order to be played in a consistent way. The Server database is designed with the purpose to be generic and as any well-designed database it should:

- Eliminate Data Redundancy: the same piece of data shall not be stored in more than one place to avoid inconsistences.
- Ensure Data Integrity and Accuracy

Figure 3.2 shows both database schemas.

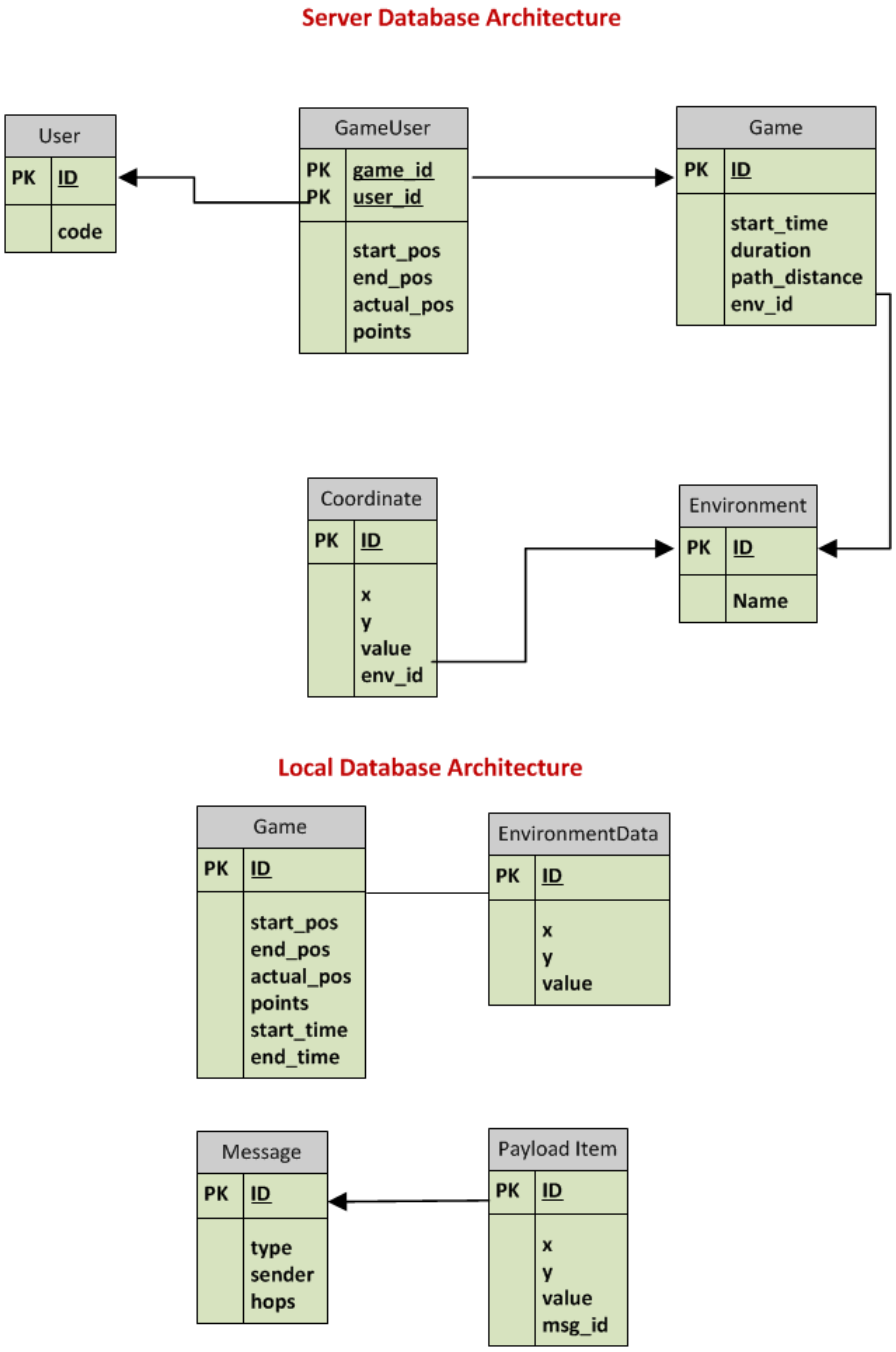

Figure 3.2: Database Architecture

Benefits of Relational Model

The benefits of a database that has been designed according to the relational model are numerous. Some of them are:

- Data entry, updates and deletions will be efficient.
- Data retrieval, summarization and reporting will also be efficient.
- Since the database follows a well-formulated model, it shall behaves predictably.

- Since much of the information is stored in the database rather than in the application, the database is somewhat self-documenting.
- Changes to the database schema are easy to make.

Chapter 4

Developing for Mobile Devices

There are several factors to take in account when you're writing an application for mobile devices. These factors are related either to hardware or software restrictions and best practices. This chapter introduces some techniques and best practices in order to write efficient and easy to use applications for mobile and embedded devices.[Wrox(2012)][Deitel et al.(2012)]

4.1 Hardware-Imposed Design Considerations

Small and portable, mobile devices offer exciting opportunities for software development. Their limited screen size and reduced memory, storage, and processor power are far less exciting, and instead present some unique challenges. Compared to desktop or notebook computers, mobile devices have relatively:

- Low processing power
- Limited RAM
- Limited permanent storage capacity
- Small screens
- High costs associated with data transfer
- Intermittent connectivity, slow data transfer rates, and high latency
- Unreliable data connections
- Limited battery life

Each new generation of phones improves many of these restrictions. In particular, newer phones have dramatically improved screen resolutions and significantly cheaper data costs. The introduction of tablet devices and Android-powered televisions has expanded the range of devices on which your application may be running and eliminating some of these restrictions. However, given the range of devices available, its always good practice to design to accommodate the worst-case scenario to ensure your application provides a great user experience no matter what the hardware platform its installed on.

4.1.1 Be Efficient

Manufacturers of embedded devices, particularly mobile devices, generally value small size and long battery life over potential improvements in processor speed. For developers, that means losing the head start traditionally afforded thanks to Moores law (the doubling of the number of transistors placed on an integrated circuit every two years). In desktop and server hardware, this usually results directly in processor performance improvements; for mobile devices, it instead means thinner, more power-efficient mobiles, with brighter, higher resolution screens. By comparison, improvements in processor power take a back seat.

In practice, this means that you always need to optimize your code so that it runs quickly and responsively, assuming that hardware improvements over the lifetime of your software are unlikely to do you any favors. Code efficiency is a big topic in software engineering, here we explain just some main concepts.

4.1.2 Expect Limited Capacity

Advances in flash memory and solid-state disks have led to a dramatic increase in mobile-device storage capacities. (MP3 collections still tend to expand to fill the available storage.) Although an 8GB flash drive or SD card is no longer uncommon in mobile devices, optical disks offer more than 32GB, and terabyte drives are now commonly available for PCs. Given that most of the available storage on a mobile device is likely to be used to store music and movies, many devices offer relatively limited storage space for your applications.

Android lets you specify that your application can be installed on the SD card as an alternative to using internal memory, but there are significant restrictions to this approach and it isnt suitable for all applications. As a result, the compiled size of your application is an important consideration, though more important is ensuring that your application is polite in its use of system resources.

You should carefully consider how you store your application data. To make life easier, you can use the Android databases and Content Providers to persist, reuse, and share large quantities of data. For smaller data storage, such as preferences or state settings, Android provides an optimized framework. Of course, these mechanisms wont stop you from writing directly to the file system when you want or need to, but in those circumstances always consider how youre structuring these files, and ensure that yours is an efficient solution. Part of being polite is cleaning up after yourself. Techniques such as caching, pre-fetching, and lazy loading are useful for limiting repetitive network lookups and improving application responsiveness, but dont leave files on the file system or records in a database when theyre no longer needed.

4.1.3 Design for Different Screens

The small size and portability of mobiles are a challenge for creating good interfaces, particularly when users are demanding an increasingly striking and information-rich graphical user experience. Combined with the wide range of screen sizes that make up the Android device ecosystem, creating consistent, intuitive, and pleasing user interfaces can be a significant challenge. Write your applications knowing that users will often only glance at the screen. Make your applications intuitive and easy to use by reducing the number of controls and putting the most important information front and center.

Graphical controls are an excellent means of displaying a lot of information in a way thats easy to understand. Rather than a screen full of text with a lot of buttons and text-entry boxes, use colors, shapes, and graphics to convey information. Youll also need to consider how touch input is going to affect your interface design. The time of the stylus has passed; now its all about finger input, so make sure your Views are big enough to support interaction using a finger on the screen. To support accessibility and non-touch screen devices, you need to ensure your application is navigable without relying purely on touch.

Android devices are now available with a variety of screen sizes, from small-screen QVGA phones to 10.1” tablets. As display technology advances and new Android devices are released, screen sizes and resolutions will be increasingly varied. To ensure that your application looks good and behaves well on all the possible host devices, you need to design and test your application on a variety of screens, optimizing for small screens and tablets, but also ensuring that your UIs scale well on any display.

4.1.4 Expect Low Speeds, High Latency

The ability to incorporate some of the wealth of online information within your applications is incredibly powerful. Unfortunately, the mobile Web isnt as fast, reliable, or readily available as we would like; so, when youre developing your Internet-based applications, its best to assume that

the network connection will be slow, intermittent, and expensive. With unlimited 4G data plans and citywide Wi-Fi, this is changing, but designing for the worst case ensures that you always deliver a high-standard user experience.

This also means making sure that your applications can handle losing (or not finding) a data connection. The Android Emulator enables you to control the speed and latency of your network connection.

4.1.5 At What Cost?

If youre a mobile device owner, you know all too well that some of your devices functionality can literally come at a price. Services such as SMS and data transfer can incur additional fees from your service provider.

Its obvious why any costs associated with functionality in your applications should be minimized, and that users should be made aware when an action they perform might result in their being charged.

Its a good approach to assume that theres a cost associated with any action involving an interaction with the outside world. In some cases (such as with GPS and data transfer), the user can toggle Android settings to disable a potentially costly action. As a developer, its important that you use and respect those settings within your application. In any case, its important to minimize interaction costs by doing the following:

- Transferring as little data as possible
- Caching data and geocoding results to eliminate redundant or repetitive lookups
- Stopping all data transfers and GPS updates when your Activity is not visible in the foreground (provided theyre only used to update the UI)
- Keeping the refresh/update rates for data transfers (and location lookups) as low as practicable
- Scheduling big updates or transfers at off-peak times or when connected via Wi-Fi by using Alarms and Broadcast Receivers
- Respecting the users preferences for background data transfers

Rather than enforcing a particular technique based on which seems cheaper, consider letting your users choose. For example, when users are downloading data from the Internet, ask them if they want to use any network available or limit their transfers to times when theyre connected via Wi-Fi.

4.2 Considering the User Environment

Its also important to consider when and how your users will use your applications. People use their mobiles all the time on the train, walking down the street, or even while driving their cars. You cant make people use their phones appropriately, but you can make sure that your applications dont distract them any more than necessary.

What does this mean in terms of software design? Make sure that your application:

- Is predictable and well behaved Start by ensuring that your Activities suspend when theyre not in the foreground. Android fi res event handlers when your Activity is paused or resumed, so you can pause UI updates and network lookups when your application isnt visible theres no point updating your UI if no one can see it. If you need to continue updating or processing in the background, Android provides a Service class designed for this purpose, without the UI overheads.
- Switches seamlessly from the background to the foreground With the multitasking nature of mobile devices, its likely that your applications will regularly move into and out of the

background. Its important that they come to life quickly and seamlessly. Androids nondeterministic process management means that if your application is in the background, theres every chance it will get killed to free resources. This should be invisible to the user. You can ensure seamlessness by saving the application state and queuing updates so that your users dont notice a difference between restarting and resuming your application. Switching back to it should be seamless, with users being shown the UI and application state they last saw.

- Is polite Your application should never steal focus or interrupt a users current Activity. Instead, use Notifications to request your users attention when your application isnt in the foreground. There are several ways to alert users for example, incoming calls are announced by a ringtone and/or vibration; when you have unread messages, the LED flashes; and when you have new voice mail, a small unread mail icon appears in the status bar. All these techniques and more are available to your application using the Notifications mechanism.

- Presents an attractive and intuitive UI Your application is likely to be one of several in use at any time, so its important that the UI you present is easy to use. Spend the time and resources necessary to produce a UI that is as attractive as it is functional, and dont force users to interpret and relearn your application every time they load it. Using it should be simple, easy, and obvious particularly given the limited screen space and distracting user environment.

- Is responsive Responsiveness is one of the most critical design considerations on a mobile device. Youve no doubt experienced the frustration of a frozen piece of software; the multifunctional nature of a mobile makes this even more annoying. With the possibility of delays caused by slow and unreliable data connections, its important that your application use worker threads and background Services to keep your Activities responsive and, more important, to stop them from preventing other applications from responding promptly.

4.3 What is Android?

Android[1] is a mobile operating system that is based on a modified version of Linux. It was originally developed by a startup of the same name, Android, Inc. In 2005, as part of its strategy to enter the mobile space, Google purchased Android and took over its development work (as well as its development team).

Google wanted Android to be open and free; hence, most of the Android code was released under the open-source Apache License, which means that anyone who wants to use Android can do so by downloading the full Android source code. Moreover, vendors (typically hardware manufacturers) can add their own proprietary extensions to Android and customize Android to differentiate their products from others.

This simple development model makes Android very attractive and has thus piqued the interest of many vendors. This has been especially true for companies affected by the phenomenon of Apples iPhone, a hugely successful product that revolutionized the smartphone industry. Such companies include Motorola and Sony Ericsson, which for many years have been developing their own mobile operating systems. When the iPhone was launched, many of these manufacturers had to scramble to find new ways of revitalizing their products. These manufacturers see Android as a solution they will continue to design their own hardware and use Android as the operating system that powers it.

The main advantage of adopting Android is that it offers a unified approach to application development. Developers need only develop for Android, and their applications should be able to run on numerous different devices, as long as the devices are powered using Android. In the world of smartphones, applications are the most important part of the success chain. Device

[1] `http://www.android.com/about/`

manufacturers therefore see Android as their best hope to challenge the onslaught of the iPhone, which already commands a large base of applications.

4.3.1 Android Architecture

In order to understand how Android works, take a look at Figure 4.1, which shows the various layers that make up the Android operating system (OS).

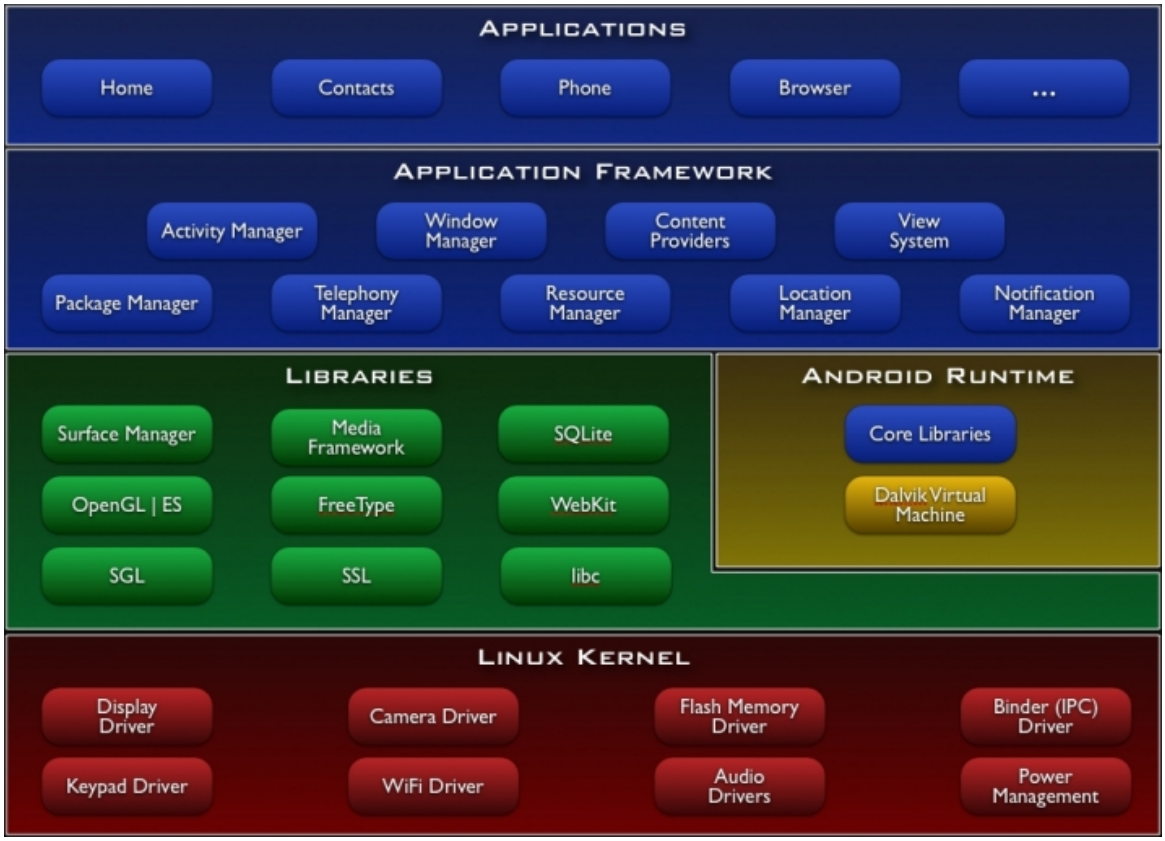

Figure 4.1: Android OS Architecture

- Linux kernel This is the kernel on which Android is based. This layer contains all the lowlevel device drivers for the various hardware components of an Android device.

- Libraries These contain all the code that provides the main features of an Android OS. For example, the SQLite library provides database support so that an application can use it for data storage. The WebKit library provides functionalities for web browsing.

- Android runtime At the same layer as the libraries, the Android runtime provides a set of core libraries that enable developers to write Android apps using the Java programming language. The Android runtime also includes the Dalvik virtual machine, which enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine (Android applications are compiled into the Dalvik executables). Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.

- Application framework Exposes the various capabilities of the Android OS to application developers so that they can make use of them in their applications.

- Applications At this top layer, you will find applications that ship with the Android device (such as Phone, Contacts, Browser, etc.), as well as applications that you download and install from the Android Market. Any applications that you write are located at this layer.

4.3.2 Android Features

As Android is open source and freely available to manufacturers for customization, there are no fixed hardware and software configurations. However, Android itself supports the following features:

- Storage Uses SQLite, a lightweight relational database, for data storage.
- Connectivity Supports GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth (includes A2DP and AVRCP), WiFi, LTE, and WiMAX.
- Messaging Supports both SMS and MMS.
- Web browser Based on the open-source WebKit, together with Chromes V8 JavaScript engine
- Media support Includes support for the following media: H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC (in MP4 or 3GP container), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP
- Hardware support Accelerometer Sensor, Camera, Digital Compass, Proximity Sensor, and GPS
- Multi-touch Supports multi-touch screens
- Multi-tasking Supports multi-tasking applications
- Flash support Android 2.3 supports Flash 10.1.
- Tethering Supports sharing of Internet connections as a wired/wireless hotspot

4.4 Developing for Android

Nothing covered so far is specific to Android; the preceding design considerations are just as important in developing applications for any mobile device. In addition to these general guidelines, Android has some particular considerations and design best practices included in Googles Android Dev Guide [2]. The Android design philosophy demands that applications be designed for:

- Performance
- Responsiveness
- Freshness
- Security
- Seamlessness
- Accessibility

[2] `http://developer.android.com/guide/index.html`

4.4.1 Being Fast and Efficient

In a resource-constrained environment, being fast means being efficient. A lot of what you already know about writing efficient code will be applicable to Android, but the limitations of embedded systems and the use of the Dalvik VM mean you cant take things for granted.

The smart bet for advice is to go to the source. The Android team has published some specific guidance on writing efficient code for Android [3]

Some of these performance suggestions contradict established design practices. For example, avoiding the use of internal setters and getters or preferring virtual classes over using interfaces. When writing software for resource-constrained systems such as embedded devices, theres often a compromise between conventional design principles and the demand for greater efficiency.

One of the keys to writing efficient Android code is not to carry over assumptions from desktop and server environments to embedded devices

4.4.2 Being Responsive

Android takes responsiveness very seriously. Android enforces responsiveness with the Activity Manager and Window Manager. If either service detects an unresponsive application, it will display an "Application is not responding" dialog previously described as a force close error, as shown in Figure 4.2

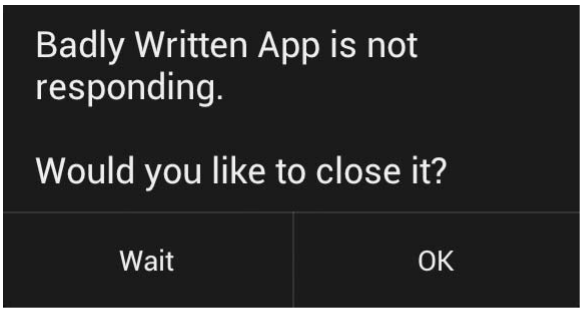

Figure 4.2: Example of a Non responsive app

This alert is modal, steals focus, and wont go away until you press a button. Its pretty much the last thing you ever want to confront a user with. Android monitors two conditions to determine responsiveness:

- An application must respond to any user action, such as a key press or screen touch, within five seconds.
- A Broadcast Receiver must return from its onReceive handler within 10 seconds.

The most likely culprit in cases of unresponsiveness is a lengthy task being performed on the main application thread. Network or database lookups, complex processing (such as the calculating of game moves), and fi le I/O should all be moved off the main thread to ensure responsiveness. There are a number of ways to ensure that these actions dont exceed the responsiveness conditions, in particular by using Services and worker threads. Android 2.3 (API level 9) introduced Strict Mode an API that makes it easier for you to discover file I/O and network transfers being performed on the main application thread.

[3] `http://developer.android.com/guide/practices/design/performance.html`

4.4.3 Ensuring Data Freshness

The ability to multitask is a key feature in Android. One of the most important use cases for background Services is to keep your application updated while its not in use. Where a responsive application reacts quickly to user interaction, a fresh application quickly displays the data users want to see and interact with. From a usability perspective, the right time to update your application is immediately before the user plans to use it.

In practice, you need to weigh the update frequency against its effect on the battery and data usage. When designing your application, its critical that you consider how often you will update the data it uses, minimizing the time users are waiting for refreshes or updates, while limiting the effect of these background updates on the battery life.

4.4.4 Developing Secure Applications

Android applications have access to networks and hardware, can be distributed independently, and are built on an open-source platform featuring open communication, so it shouldnt be surprising that security is a significant consideration.

For the most part, users need to take responsibility for the applications they install and the permissions requests they accept. The Android security model sandboxes each application and restricts access to services and functionality by requiring applications to declare the permissions they require. During installation users are shown the applications required permissions before they commit to installing it.

This doesnt get you off the hook. You not only need to make sure your application is secure for its own sake, but you also need to ensure that it doesnt leak permissions and hardware access to compromise the device. You can use several techniques to help maintain device security, and theyll be covered in more detail as you learn the technologies involved. In particular, you should do the following:

- Require permissions for any Services you publish or Intents you broadcast. Take special care when broadcasting an Intent that you arent leaking secure information, such as location data.
- Take special care when accepting input to your application from external sources, such as the Internet, Bluetooth, NFC, Wi-Fi Direct, SMS messages, or instant messaging (IM).
- Be cautious when your application may expose access to lower-level hardware to third-party applications.
- Minimize the data your application uses and which permissions it requires.

4.4.5 Ensuring a Seamless User Experience

The idea of a seamless user experience is an important concept. It means to ensure a consistent user experience in which applications start, stop, and transition instantly and without perceptible delays or jarring transitions.

The speed and responsiveness of a mobile device shouldnt degrade the longer its on. Androids process management helps by acting as a silent assassin, killing background applications to free resources as required. Knowing this, your applications should always present a consistent interface, regardless of whether theyre being restarted or resumed.

With an Android device typically running several third-party applications written by different developers, its particularly important that these applications interact seamlessly. Using Intents, applications can provide functionality for each other. Knowing your application may provide, or consume, third-party Activities provides additional incentive to maintain a consistent look and feel.

Use a consistent and intuitive approach to usability. You can create applications that are revolutionary and unfamiliar, but even these should integrate cleanly with the wider Android environment. Persist data between sessions, and when the application isnt visible, suspend tasks

that use processor cycles, network bandwidth, or battery life. If your application has processes that need to continue running while your Activities are out of sight, use a Service, but hide these implementation decisions from your users.

When your application is brought back to the front, or restarted, it should seamlessly return to its last visible state. As far as your users are concerned, each application should be sitting silently, ready to be used but just out of sight.

You should also follow the best-practice guidelines for using Notifications and use generic UI elements and themes to maintain consistency among applications.

4.4.6 Providing Accessibility

When designing and developing your applications, its important not to assume that every user will be exactly like you. This has implications for internationalization and usability but is critical for providing accessible support for users with disabilities that require them to interact with their Android devices in different ways.

Android provides facilities to help these users navigate their devices more easily using text-to-speech, haptic feedback, and trackball or D-pad navigation. To provide a good user experience for everyone including people with visual, physical, or age-related disabilities that prevent them from fully using or seeing a touchscreen you can leverage Androids accessibility layer.

As a bonus, the same steps required to help make your touchscreen applications useful for users with disabilities will also make your applications easier to use on non-touch screen devices.

Chapter 5

Implementation

The chapter discusses the implementation details for the main components that were to be implemented in the mobile application, namely the Game component, the Service component, the Database layer component, the Server component and the Peer interface component.

5.1 Application Fundamentals

Android applications are written in the Java programming language. The Android SDK tools compile the codealong with any data and resource filesinto an Android package, an archive file with an .apk suffix. All the code in a single .apk file is considered to be one application and is the file that Android-powered devices use to install the application.

Once installed on a device, each Android application lives in its own security sandbox:

- The Android operating system is a multi-user Linux system in which each application is a different user.

- By default, the system assigns each application a unique Linux user ID (the ID is used only by the system and is unknown to the application). The system sets permissions for all the files in an application so that only the user ID assigned to that application can access them.

- Each process has its own virtual machine (VM), so an application's code runs in isolation from other applications. By default, every application runs in its own Linux process. Android starts the process when any of the application's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other applications.

In this way, the Android system implements the principle of least privilege. That is, each application, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an application cannot access parts of the system for which it is not given permission.

However, there are ways for an application to share data with other applications and for an application to access system services:

- It's possible to arrange for two applications to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, applications with the same user ID can also arrange to run in the same Linux process and share the same VM (the applications must also be signed with the same certificate).

- An application can request permission to access device data such as the user's contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and more. All application permissions must be granted by the user at install time.

5.1.1 Application components

Application components are the essential building blocks of an Android application. Each component is a different point through which the system can enter your application. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific roleeach one is a unique building block that helps define your application's overall behavior.There are four different types of application components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

Here are the four types of application components:

- Activities[1] - An activity is implemented as a subclass of Activity[2] and it represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email application, each one is independent of the others. As such, a different application can start any one of these activities (if the email application allows it). For example, a camera application can start the activity in the email application that composes new mail, in order for the user to share a picture.

- Services[3] -A service is implemented as a subclass of Service[4] and it is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.

- Content providers[5] - A content provider manages a shared set of application data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your application can access. Through the content provider, other applications can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any application with the proper permissions can query part of the content provider (such as ContactsContract.Data) to read and write information about a particular person.

  Content providers are also useful for reading and writing data that is private to your application and not shared. For example, the Note Pad sample application uses a content provider to save notes.

  A content provider is implemented as a subclass of ContentProvider and must implement a standard set of APIs that enable other applications to perform transactions.

- Broadcast receivers[6] A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the systemfor example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Applications can also initiate broadcastsfor example, to let other applications know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver

[1] http://developer.android.com/guide/components/activities.html
[2] http://developer.android.com/reference/android/app/Activity.html
[3] http://developer.android.com/guide/components/services.html
[4] http://developer.android.com/reference/android/app/Service.html
[5] http://developer.android.com/guide/topics/providers/content-providers.html
[6]

is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.

A broadcast receiver is implemented as a subclass of BroadcastReceiver[7] and each broadcast is delivered as an Intent object.

5.1.2 The Manifest File

Before the Android system can start an application component, the system must know that the component exists by reading the application's AndroidManifest.xml file (the "manifest" file). Your application must declare all its components in this file, which must be at the root of the application project directory.

The manifest does a number of things in addition to declaring the application's components, such as:

- Identify any user permissions the application requires, such as Internet access or read-access to the user's contacts.
- Declare the minimum API Level required by the application, based on which APIs the application uses.
- Declare hardware and software features used or required by the application, such as a camera, bluetooth services, or a multitouch screen.
- API libraries the application needs to be linked against (other than the Android framework APIs), such as the Google Maps library.

The primary task of the manifest is to inform the system about the application's components. For example, the manifest file of our application is the following:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mob.p2p.game"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="7" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <application android:debuggable="true"
        android:icon="@drawable/p2pgame"
        android:label="@string/app_name">
        <activity
            android:name=".P2PGame"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity
            android:name=".About"
            android:label="@string/about_title"
            android:theme="@android:style/Theme.Dialog">
        </activity>
        <activity
            android:name=".Game"
            android:label="@string/game_title">
        </activity>
```

[7] http://developer.android.com/reference/android/content/BroadcastReceiver.html

```
        <!-- Declare the Game Core Service -->
        <service android:name=".core.GameCoreService"/>
    </application>
</manifest>
```

As you can see we have three different activities, P2PGame, About and Game. The first one P2PGame is our Main activity, it means that this one will be launched for first when we click on the application icon. You can see it as our entry point to the application and from this point we will use Intent objects to start other acitivities directly from java code.Another important component that you can note is the Service component, namely GameCoreService which will contain our peer-to-peer algorithm.

5.2 Game Component

The game component is the entry point of the application. It is composed by three parts, two Activities and a View, namely P2Pgame, Game, GridView. Figure 5.1 shows the lifecycle of an activity:

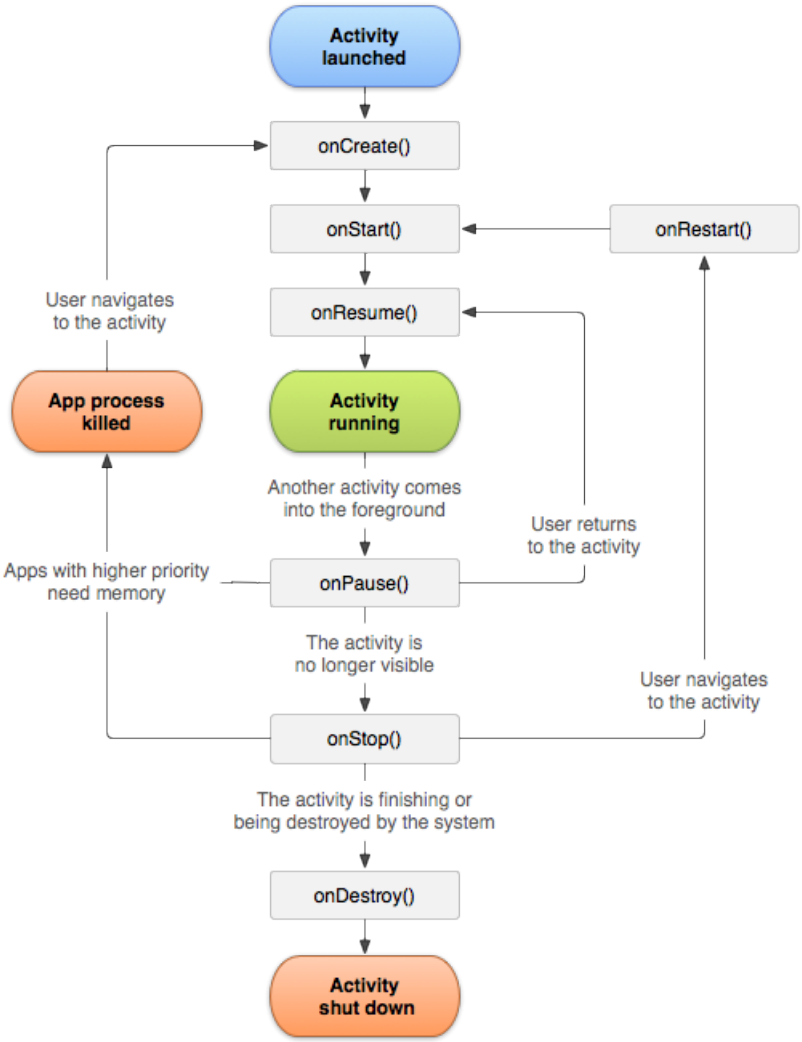

Figure 5.1: The lifecycle of an activity

5.2.1 P2PGame Activity

P2PGame.java is the first component to be executed, once the User click on the game icon. What it does is to render the Menu view in which the user can choose between different options like:

- Start a New Game
- Continue an old Game
- Read the Game rules
- Exit the Game

Every part of this component is configured in xml file, this makes it easly configurable without the need to change java code. Each menu option performs a different action. However the main choices are the first two, they both instantiate a new Intent object which is used to activate the second activity Game.java

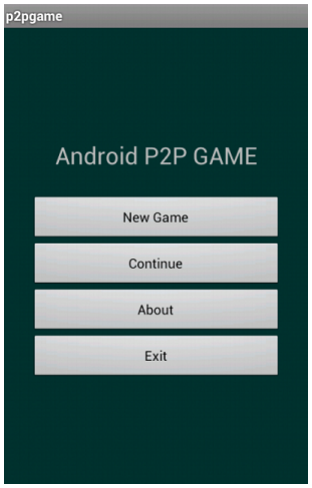

Figure 5.2: Menu screen

5.2.2 GridView

The view can be seen as the *"Presentation Layer"* while the *Game* activity as the business logic. Consequently the GridView contains all the UI design details. Its main functionalities are:

- designs the UI
- manages touch control
- manages KeyPad control to devices without touchscreen (like D-Pad)

Our User Interface(which simulates an environment) consists essentially in a 2D-Grid. The grid is composed by *n* cells, each of them represents a step. Three different colors are used to show Safe, Dangerous and *Unknown* steps, respectively green, red and yellow. Half of the grid is composed by *Unknown* steps, which are randomly chosen.

The dimension of the grid(number of cells), colors used, and other configuration data are set on a *XML* configuration file. The cells are simple rectangles of the same dimension with a color. To support different size screens, we divide the screen dimension by the number of rectangles it should contains in order to get the dimensions of the rectangles. By supporting different screens and different controls(touch and D-pad) we make sure to meet the *"Provide Accessibility"* requirement.

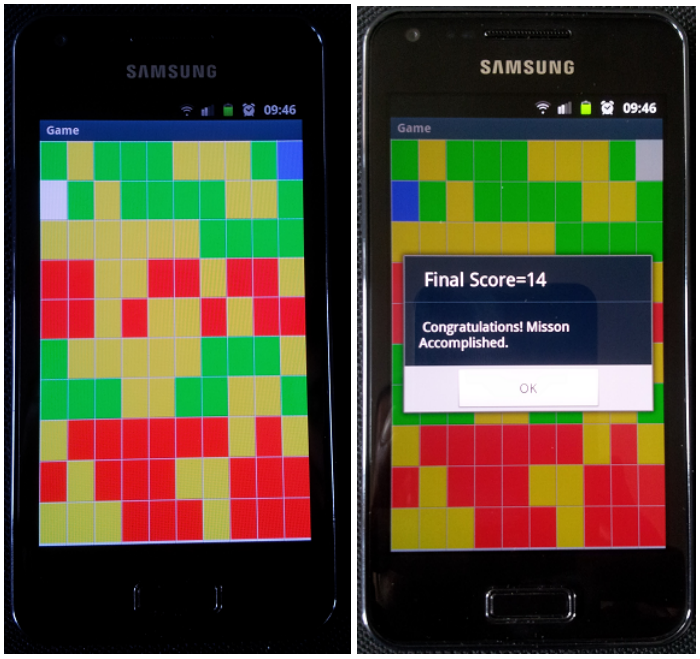

Figure 5.3: The Grid View is our simple model of the post disaster scenario

5.2.3 Game Activity

Game activity is responsible for all the game logic except, the managing of requests from peers which is delegated to the service component. As you can easly intuitiate watching the figure, the following three tasks are executed in the onCreate() method.

- Instantiate a View class namely GridView.java which is responsible for the user interface
- It launches the Server Interaction component in a new background work thread.
- Creates the known Environment data (Grid Data) or load them from the database depending on whether the user is playing a new game or loading an old one. Both operations are done within a new background worker thread.

After the environment is created, during the game palying the component is responsible also for the following activities:

- After each step (user moves) it checks whether the neighbors are all known, if not it creates a new Intent object containing all the information needed and send it to the peer-to-peer service (activating the service if it not active) in order to ask the others peer.
- It contains a Broadcast Receiver listener able to capture replies from the other peers which has been broadcasted to the activity by the Service component. Once the new information is captured it sends it to the View in order to update the known environment data.
- It validates if the step is valid and updates the score consequently.
- If the player reaches the destination coordinate than it activates the Server Interaction Components in order to send the Game data to the Server Component.
- instantiate the database component (within a new background worker thread) in order to save the game data, if for any reason the exit the game. Keeping in mind always our activity lifecycle we can intuie that the right place to do this is inside the onPause() or onExit() method.

The BroadcastReceiver

Inside the Game activity we use another core component of Android, a BroadCast Receiver. The intent is to capture the information sent by our P2P Service. We said before that any Android component has to be declared in the manifest file, but if you look at it you'll not see any BroadcastReceiver. Is this an exception of the general rule? here we explain our choice.

We said before in our best practices section; one important thing to keep in mind is that mobile devices have a limited time battery. We don't want to do computation that is not necessary. When you declare a BroadcastReceiver in the Manifest.xml file, no matter whether your app is running or not, the BroadcastReceiver will be awaken and thus the onReceive method will be called, thus consumes battery dramatically. However, when you declare it in the code, the broadcastReceiver will only be effective when the activity is running and thus avoid extreme battery consumption.

Listing 5.1: onResume() method registers the receiver while onPause() unregister it

```
@Override
protected void onResume() {
       super.onResume();
       // Register the broadcaster receiver with the service
       registerReceiver(onBroadcast, new IntentFilter(Constants.NOTIFY_GAME));
}

@Override
protected void onPause() {
       super.onPause();
       Log.d(TAG, "onPause");
       /* Unregister the receiver */
       unregisterReceiver(onBroadcast);

}
```

Listing 5.2: onReceive() method of our Broadcast Receiver

```
/**
 * The broadcast receiver able to capture messages sent from our Peer to
 * peer service
*/
private BroadcastReceiver onBroadcast = new BroadcastReceiver() {
@Override
public void onReceive(Context ctxt, Intent intent) {
  if (Constants.NOTIFY_GAME.equals(intent.getAction())) {
       Message msg = (Message)
           intent.getSerializableExtra(Constants.RESPONSE_MSG);
       // Updates the view showing the information received from peers
       updateGridValues(msg.getCoordinates());
   }
}
```

You can also note that the component makes extensively use of multithreading, it is used to keep the View (user interface) always responsive by running it on a dedicated process. Reassuming, we address some of the most important requirements every mobile application should have:

- Switches seamlessly from the background to the foreground
- Be Responsive
- Avoid extreme battery consumption

5.3 Service Component

The Service Component can be considered the core of the Application since it encapsulates the peer-to-peer algorithm. There are several reasons that made us decide to build a service in order to manage the communication with peers:

- it offers an easy integration with other components or applications
- the service abstraction is provably scalable
- it is suitable to perform long-running operations in the background

The peer-to-peer communication component was built in a precedent project without knowing the algorithm to call in order to manage outgoing and upcoming messages. Encapsulating the algorithm within an Android Service makes communication with it easy and application independent. It means that a Service can be called by any component of the application or by another application in the same way. It consists on:

- Instantiating a java object call Intent.
- Put all the information needed to the service inside the Intent object that incorporates also as a bundle.
- Call the startService() method with the Intent upon created as a parameter.

Services are scalable, some of the best examples of stateless service-oriented interactions can be seen in certain P2P technologies such as Gnutella.[Taylor & Harrison(2009)] It is suitable to perform long-running operations in the background because it runs on a separate process and executes all the work within a dedicated worker thread.

Implementation details are hidden behind the service interface. To communicate with the service we use messages, and the structure of the message and its content are defined by the interface. Like distributed object systems that use an IDL, Android services describe this interface in a description language called AIDL (Android IDL). Its the only way to provide IPC (Inter Process Communication) in an Android application.

Service Implementation

You can implement a Service in Android in different ways, our choice was to implement it like an IntentService[8].To understand our choice, let's give a look at what the IntentService offers:

- Creates a default worker thread that executes all intents delivered to onStartCommand() separate from your application's main thread. *So we have our main thread dedicated to the view, this address the "Be Responsive" requirement.*
- Creates a work queue that passes one intent at a time to your onHandleIntent() implementation, so you never have to worry about multi-threading. *Well, it manages for us multithreading and makes use of a FIFO(First In First Out) policy to manage incoming messages.*
- Stops the service after all start requests have been handled, so you never have to call stopSelf(). *We don't want to keep the service running if it is not necessary, otherwise it consumes the battery*
- Provides a default implementation of onStartCommand() that sends the intent to the work queue and then to your onHandleIntent() implementation. *As we said before, in order to start the service a component/application sends an Intent object. To manage these objects we have to implement our logic inside the onHandleIntent() mthod.*

```
@Override
protected void onHandleIntent(Intent intent) {

Log.d(TAG, "onHandleIntent" + intent.toString());

synchronized (this) {
      try {

            gamedb = new GameDB(getApplicationContext());

            Bundle bundle = intent.getExtras();

            if (intent.getAction().equals(Constants.CALL_SERVICE)) {
                  Message msg = (Message) bundle
                        .getSerializable(Constants.MSG_DATA);

                  if (msg != null) {
                        Log.d(TAG, "Message to Manage " + msg.toString());

                        /* Call the Algorithm to manage incoming message */
                        manageMsg(msg);
                  }
            }

      } catch (Exception e) {
            e.printStackTrace();
      }
}
}
```

Figure 5.4: Implementation of onHandleIntent()

Figure 5.4 shows our simple implementation of the method onHandleIntent():

You can see that all the logic is delagated to the method *manageMsg()* which will be explained in the next section. Notice how the synchronized construct takes an object in parantheses. In our case "this" is used, which is the instance the P2P algorithm is executed on. The object taken in the parantheses by the synchronized construct is called a monitor object. The code is said to be synchronized on the monitor object. A synchronized method uses the object it belongs to as monitor object, and most importantly, only one thread can execute inside a code block synchronized on the same monitor object.

5.3.1 P2P Algorithm

Our service has basically these main functions:

- handles the communication with the peers autonomously
- notifies the game view whenever a "Response" message is received (if it contains unknown information)
- handles the communication with the peers on Game behalf.

The algorithm takes as input a **queue** of messages (sent by the peers or the game). A **Message** object has the following main properties (instance variables):

- *Sender* - the ID of the sender who created the message. This field never changes during the lifecycle of the message ,the sender is also the final destinator.

[8] http://developer.android.com/reference/android/app/IntentService.html

- *Type* - this field has two possible values “Request” or “Response”.
- *Hops* - the remaining number of hops. The message is forwarded to peers until the hopes becomes equal to zero.
- *Payload* - the content of the message. It contains all the asked/answered informations.

A Simple Use Case

To understand how the state of the Message changes during his life let’s do the following simple example:

A needs some information, so he creates a new Message
sets as *sender* himself, as *type* “Request” and as *payload* the needed informations.
A forwards the message to his peers.
Let’s suppose peer **B** has the informations,
then he updates the *payload* and changes the *type* to “Response”
B decreases *hops* and forward the message.

In poor words the only variable that does not changes is the *Sender*. To explain how it implements these functionalities let’s take a look at the algorithm used to manage incoming messages.

P2P Algorithm

```
1:  Q ← getMessages()
2:  while Q is not empty do
3:      message ← Q.pop()
4:      hops ← message.getHops()
5:      knownData ← getKnownEnvironment()
6:      if is Request message then
7:          if Sender is me then
8:              forwardToPeers(message);
9:          else if hops ≥ 1 then
10:             hops ← hops − 1;
11:             match ← message ∩ knownData;
12:             forwardToPeers(match);
13:         end if
14:     else if is Response message then
15:         notifyGame(message);
16:         updateDB(message);
17:         if I am Not the Sender then
18:             forwardToPeers(message);
19:         end if
20:     end if
21: end while
```

The first thing to note is that the same algorithm(block of code) is applied to:

- *Request* messages
- *Response* messages
- Incoming messages
- Outgoing messages

Steps 6-8
If the type is Request and the sender is the game itself
then it just forwards the messages to peers. No more steps are necessary.
Steps 9-12

If the type is Request and the sender is not me,
then checks the hops number is greater than zero.
If true, **then** checks if it has the information requested.
If Yes (partial or complete) it forwards a response message
Steps 14-16
Otherwise If was a Response
notifies the game with the new information if needed
update the database with the new information if needed
Steps 17-19
If the Sender is not me (in other words; I am not the final destinator)
it also forwards the message to peers.

5.4 Database Layer Component

One of the key requirement of the project is to provide data about the player performances. Our game will last more than a day so we can't expect users to play it without an interruption. It would also be impossible because of the battery. However, even if the period would have been shorter, you have to avoid accidental loss of data. To address this requirement we need a way to persist the game data.

Android provides several options for you to save application data. The solution choosen depends on your specific needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires. Android data storage options are the following:

- **Shared Preferences** *Store private primitive data in key-value pairs.*
- **Internal Storage** *Store private data on the device memory.*
- **External Storage** *Store public data on the shared external storage.*
- **SQLite Databases** *Store structured data in a private database.*
- **Network Connection** *Store data on the web with your own network server.*

After an investigation of the possible scenarios(use cases) our choice was to use a SqLite database to persist our key data.

5.4.1 SQLite

[9] Most database systems are large server-based applications. For example many web applications use multiple servers and clusters of databases on the server side. SQLite is often used within applications to manage local data. Apple OS X, DropBox, Firefox, and Chrome all use it, as do many other applications and products.

SQLite uses the Structured Query Language (SQL), as its name implies, to allow you to create and maintain tables and to insert and select data. Though SQLite uses SQL, it isnt meant to replace the large server offerings that Oracle, Microsoft, IBM, and others supply. Instead, its designed to be small, fast, and easy to use for in-process data.

Even though its small and fast, SQLite is powerful. It supports transactions (which are atomic even after system crashes), foreign keys, functions, triggers, and more. In addition, although SQLite has many features other SQL systems have, it doesnt have them all. SQLite doesnt support certain join types (right outer, full outer), some alter statements, and it treats data types more loosely than other systems.

However we don't need enterprise features, after all, our App runs on Smartphones.

[9] `http://www.sqlite.org/`

5.4.2 Database layer (Interface)

The data persisted on the database are used by our two main components, the **Game** activity and the **Service**. When you do professional application development you have to avoid code duplication, and offer to different components a common interface to resources. To implement these rules we created the **GameDB** layer that is an interface to our database with all the utilities methods needed to the app.

As you can intuit its main tasks are:

- Create the tables at runtime the first time it is called. Of course this operation is executed just once, the first time you open the game.
- Offer methods to Load/Insert/Update/Delete table records.
- Encrypt sensible data before saving on the database. This feature is necessary because we don't want users to manually change the data on the database.

5.5 Server Component

The Server componenet (or interface) manages the interaction with the central server. It has two basic functionalities:

- when the Game starts it communicates with the server to get initialization data
- when the game ends, it sends the game's data to the server

In the current implementation it uses the *HTTP* protocol to communicate with the server. Here again, before sent, data are encrypted to protect from unauthorized interception.

5.6 Messages

Several options are available for the encoding of messages in order for them to be passed around and still remain machine processable. The main three are Java Serialised Objects, XML and JSON. Each of them are discussed below in order to determine the most suitable option to choose and implement in the application.

5.6.1 Serialised Objects

As Android applications are written using the Java language, the most obvious choice of formats to transmit messages would be a Serialised Object. Messages will most likely be represented as an object within the application, so adding serialisation to the class would be an easy option.

The process of serialising and deserialising an object is seen to be quite CPU intensive, even by Oracle the developers of Java. When on a mobile device running on battery, using these high-cost CPU operations can decrease battery life unnecessarily.

Also, serialised objects do not allow for any future expansion to other platforms by not being cross-compatible and restricted to Java based systems only. These two reasons were enough to decide to look for alternative options available.

5.6.2 XML

XML is a markup language which can be used to interchange data between devices or services, usually over the internet. By using custom tags, it allows complete flexibility in describing what is within the message being sent.

Listing 5.3: Example XML Message (164 characters)

```
<?xml version="1.0"?>
<message>
        <from>Sender</from>
        <type>REQUEST</type>
        <hops>1</hops>
```

```
        <timestamp>1335621871</timestamp>
        <payload>
                <request>1</request>
        </payload>
</message
```

5.6.3 JSON

JSON is an alternative markup language for data interchange, designed to be more lightweight than XML but still provide the same level of functionality that is required. When communication times are potentially restricted, keeping message size down is an important key factor to take into account to ensure as much data as possible is sent within the available timeframe.

The example JSON message shown is approximately 53than the exact same message denoted in XML.

Listing 5.4: Example JSON Message (88 characters))

```
{
    message:{
                from:‘Sender’,
                type:‘REQUEST’,
                hops:1,
                timestamp:1335621871,
                payload:{
                        request:1
                }
   }
}
```

5.7 Security

The option to encrypt messages was one of the key requirements to be implemented, so choosing the correct method was important particularly due to being on a mobile handset. RSA[10] was chosen initially as the method used to encrypt and decrypt messages. RSA was devised in 197719 and to date is the “most widely used public-key cryptosystem in the world”.

RSA works by encrypting the message with a public key and decrypts it at the other end with a private key. The strength of the encryption depends on the RSA modulus size of the keys, typically measured in bits. Naturally, a message encrypted with a 4096-bit key will be more difficult to break than a message encrypted with a 512-bit key. However, the higher the key modulus size is, the more CPU intensive and time consuming the process is to both encrypt and decrypt the messages.

RSA encryption also has a maximum message length, again depending on key size. As users could be walking in opposite directions, time is really of the essence. With the time taken increasing exponentially (5.5), a balance between security and time had to be found.

[10] http://www.rsa.com/rsalabs/node.asp?id=2146

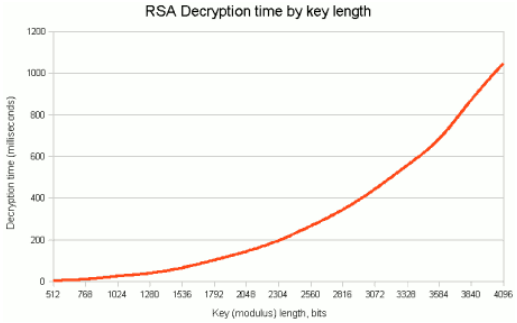

Figure 5.5: RSA decryption time by key length

Based on the times shown in 5.5 , the optimum key length would be 2048 bits which allows for messages of up to 245 bytes (245 characters) in length to be encrypted and transmitted in less than 0.2 seconds.

For the purposes of the project, a 2048-bit pub- lic/private key pair was generated for use with the RSA algorithm and stored on the handset. The encryption could be further improved by combining it with a second symmetric cryptographic algorithm, like AES[11], to alleviate some of the issues incurred with using RSA on its own.

Listing 5.5: (RSA Encryption using the public key)

```
public static String encrypt(Context ctx, String input) {
               PublicKey pkPublic = null;
               InputStream instream;
               byte[] encodedKey;

               try {

                       instream = ctx.getAssets().open(PUBLIC_KEY);
                       encodedKey = new byte[instream.available()];
                       instream.read(encodedKey);
                       X509EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(
                                       encodedKey);
                       KeyFactory kf = KeyFactory.getInstance("RSA");
                       pkPublic = kf.generatePublic(publicKeySpec);

                       Cipher pkCipher =
                           Cipher.getInstance("RSA/ECB/PKCS1PADDING");
                       pkCipher.init(Cipher.ENCRYPT_MODE, pkPublic);
                       byte[] encryptedInByte =
                           pkCipher.doFinal(input.getBytes());

                       String encryptedInString = new String(
                                       Base64Coder.encode(encryptedInByte));
                       return encryptedInString;

               } catch (FileNotFoundException e) {
                       e.printStackTrace();
               } catch (IOException e) {
                       e.printStackTrace();
               } catch (NoSuchAlgorithmException e) {
                       e.printStackTrace();
```

[11] url{http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

```
            } catch (InvalidKeySpecException e) {
                  e.printStackTrace();
            } catch (IllegalBlockSizeException e) {
                  e.printStackTrace();
            } catch (BadPaddingException e) {
                  e.printStackTrace();
            } catch (NoSuchPaddingException e) {
                  e.printStackTrace();
            } catch (InvalidKeyException e) {
                  e.printStackTrace();
            }

            return input;

      }
```

This would increase overall size of the message, but would allow for an unlimited message length to be processed independent on the RSA key size. The hybrid RSA+AES method of encryption is more complicated and is best described when broken down into multiple steps:

1. Encrypt the message using the **AES** algorithm
2. Encrypt the AES key, which will be used to decrypt the previously encoded message, using the RSA algorithm
3. Send both parts together onward to the recipient

Regardless of method chosen, if the private key used to decrypt messages is stored on the mobile device, the security of the messages cannot be guaranteed. Android does make reasonable attempts to block access to these files, however if a more knowledgeable user applies root access to the device they will have the entire device file system at their disposal which in turn would allow them to retrieve the private keys stored.

Chapter 6

Testing and Evaluation

This chapter discusses both the various methods of testing covered through the project de- velopment and subsequent evaluation on the code to determine its effectiveness.

6.1 Devices used and techniques

The application has been tested on two different devices:

1. *Samsung Galaxy S Advance*, display 4,0”, OS Android 4.0 Ice Cream Sandwich

2. *hTC Wildefire S*, display 3,2”, OS Android 2.3 Gingerbread

Figure 6.1: Samsung Galaxy S Advance and hTC Wildefire used for testing

We’ve used the following methods to test and evaluate the app:

- End-User tests on the Android Emulator and Smartphones

- Mocking code to send **Requests** and **Responses** messages to the Service

- Android *StrictMode*[1] for monitoring and evaluation

[1] `http://developer.android.com/reference/android/os/StrictMode.html`

6.2 Bluetooth Testing

A number of tests were carried out to determine the limits of Bluetooth range. Each of these involved using two mobile handsets, with one connected to a laptop to view the console output generated by the application. The primary device was set to send 1000 messages using a loop, while the second device was set to listen for the messages and echo them back to the primary handset. Using such a large number of messages allowed for plenty time to carry out the tests in each case.

- Range Test - Walls: The first test carried out aimed to determine the capability of us- ing Bluetooth through a wall, between rooms. Despite being under 10m straight line distance, a wall (plasterboard, not concrete) prevented a connection from initiating. This would be expected, as Bluetooth is targeted as a short range communication tool.

- Range Test - Straight Corridor: The second test carried out was to test the range of Bluetooth within an environment that had no obstacles blocking the signal. Also as expected, achieving a 10m range was possible.

- Range Test - Crowded Environment: The third and final test carried out was to test the range of Bluetooth within an environment that did have obstacles that could affect the signal, which in this case were people. This was completed at '"The Hub (University Restaurant) during the peak lunchtime to gauge as realistic a scenario as possible for a semi-crowded environment. On completion, it was found that approx 10m would still be possible in most cases as the space is still fairly open, as people are walking about and are not a permanent obstacle. If crowds of people were to be packed together, it would be expected that this would shorten the range overall, but unfortunately it was not possible to realistically test this scenario.

In all cases, the Bluetooth range met the expected book-value range of 10m, which was suitable for the purpose required. It was also found that, as expected, when outwith range or with major obstacles placed, no connection could be established. However, if the connection was established while in-range then moved to a location that would not normally provide a connection, communication was still possible at a slower speed for a longer distance before eventually disconnecting entirely. Higher powered Bluetooth transmitters are available that would increase the range, in some cases up to 100m, but the side effect is the undesired higher consumption on battery resources.

6.3 Component Testing and Integration

Before explaining how we tested the integration between the different components let's give a look to the data flow:

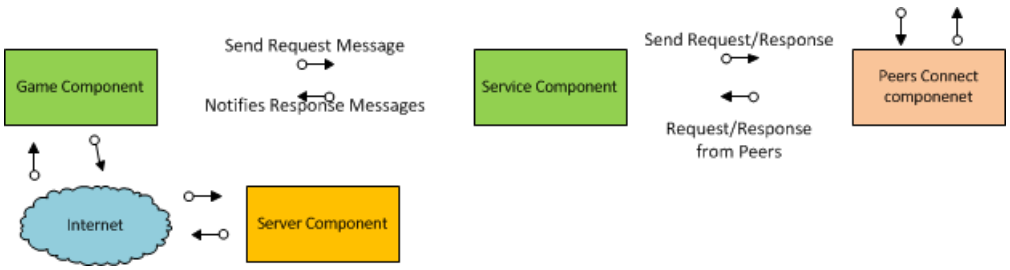

Figure 6.2: Data flow between componenets

As you can see the *Service* is independent from the local *Game*. It is not able to distinguish between messages received from peers or from the local game.

This implementation allowed us to use the local Game component as a generator of Requests and Responses and send them to the Service. The Service implements the P2P algorithm which applied to those messages forward them to the local game, to the peers or both. Let's explain it with a asimple example:

1. We move to cell (1,1)

2. then the game checks for unknown neighbours cells, creates a Request message with them and sends it to the Service

3. after executing some code lines the game componenet call our Mock method which creates and sends a Response message with destinator itself. Here through the game componenet we simulate the receiving of a Response message by other peers.

4. the Service process this Response message and once realized which is the destinator sends it to the local game.

5. the broadcast receiver in the Game capture this message from the service and updates the View. In the view(screen) the colors of uknown celles is updated from yellow to green or red.

Figures 6.3 and 6.4 illustrates the above example. The first one is the initial situation of the game, while the second is the situation after the first step to cell (1,1). Our move to cell (1,1) created a request message because as you can note there were some neighbors of (1,1) markes as "Unknown". After that, the game received a Response message and shows the new situation in the second figure.

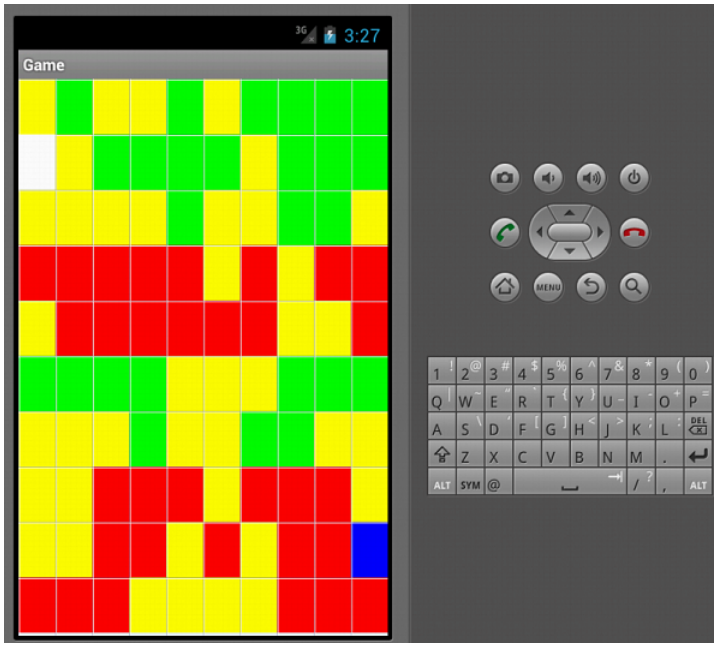

Figure 6.3: Game started. The current position is the initial position(1,0)

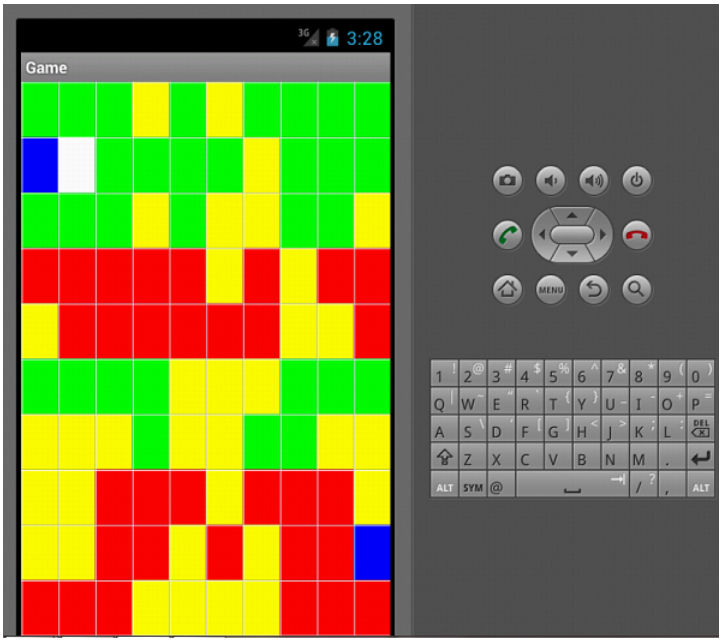

Figure 6.4: After the first step, the game receives a Response message which indicates that all neighbor cells are safe.

We repeated this mechanism in many points creating many Requests and Responses and each time the result was always the desired one. This way of testing was more useful than just simple JUnit tests in which you test the P2P algorithm, because allowed us to control the entire cycle of communication between the the Server and the Game.

6.4 Evaluation

The evaluation of the app has been done assessing; the robustness, the scalability, requirements meets and responsiveness of the UI.

6.4.1 Robustness

In order to evaluate the robustness of the app the following key tests were executed:

1. stress the Service by sending 50 messages (Requests and Responses)
2. test the Server interface component
3. test the access to the database in order to manipulate data

Service evaluation

As we explained it runs on a separate process with a dedicated thread, and uses a queue (with a FIFO policy) in order to manage messages. The response of the Service to the 50 messages was almost immediate. It takes just a few milliseconds to answer to one request and the most time consuming operation is the first access to the database in order to load the environment data.

To evaluate if 50 messages is a realistic scenario we have to take in account that Bluetooth gives you the possibility to interact with at most 7 peers. After this consideration, naturally follows than 50 messages in less than 1 second can be considered as a stress test because it will be difficult to receive so many messages in a real scenario with the constraints of the 7 peers.

Interaction with the central server

Since we dont have a central server, what weve done was to install the apache server in our local machine. All the connection configuration data such as host, protocol, port and path have been

implemented on a XML file. It means that the only thing to change when the central server will be operative is the xml file. Theres no need to change java code.

Here again; the communication with the server is handled by a dedicated background thread. It follows that the processing of the information is very fast, the bottleneck stands on the Http connection to the server. In our case it was very fast because was a connection to the local machine but in real scenario this is not true. The performance of this part depends on factors that are not dependant on our application such as the mobile device used and the connection bandwidth granted by the proper ISP.

Database access

Here the things become a bit more complicated. Even if we use dedicated thread to access the database due to concurrent accesses sometimes the communication can takes longer time than others. Why? To understand why we should turn back to the architecture and note that Service and Game interact with the database independently. It means that concurrent access can occurs. For this reason the methods to update or insert data are written as **synchronized**.

In java when a method is declared as synchronized it can be executed by one thread at time. If there are more threads calling the method, theyll be executed one after the other with a first in first out policy. So when concurrent accesses occur the last thread takes longer time to complete its job. Is this a problem? No. The local database is kept light (just the strictly necessary data are stored) and the access minimized and optimized to make it still very fast.

6.4.2 Requirements

The initial requirements of this project have been implemented and tested. However, during this period we tried also to integrate the app with the communication layer and implement some functionality on the Server component but for a lack of time it was not possible.

As we explain better in the next chapter the Server part is enough long to require an entire project in order to be implemented. What we have done is to implement the interface to the server and to test the connection to it passing and receiving some simple data. In the server part we have implemented a simple PHP script which receives the data and return a string as response. In poor words the communication has been tested, but all the logic on the server is to be implemented.

6.4.3 Scalability

The game is perfectly scalable in terms of users; this is a characteristic of all P2P architectures. Since the communication are local to a maximum of 7 peers we can add as many peers as we want, the system remains stable.

6.4.4 Responsiveness

We kept this requirement in mind since the very beginning of the project. Thats why ours is a multithreading application. This guarantees the responsiveness of the user interface and makes the app more attractive to the players.

6.4.5 Android Strict Mode

In addition to the all the above testing and evaluatio techniques, we have used also this new tool offered by the Android framework. StrictMode is a developer tool which detects things you might be doing by accident and brings them to your attention so you can fix them.

StrictMode is most commonly used to catch accidental disk or network access on the application's main thread, where UI operations are received and animations take place. Keeping disk and network operations off the main thread makes for much smoother, more responsive applications. The following is an example code to enable from early in the application component's onCreate() method:

Listing 6.1: Example of StrictMode use inside the onCreate() method

```
public void onCreate() {
    if (DEVELOPER_MODE) {
```

```
        StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder()
                .detectDiskReads()
                .detectDiskWrites()
                .detectNetwork() // or .detectAll() for all detectable problems
                .penaltyLog()
                .build());
        StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder()
                .detectLeakedSqlLiteObjects()
                .detectLeakedClosableObjects()
                .penaltyLog()
                .penaltyDeath()
                .build());
    }
    super.onCreate();
}
```

Thanks to this tool and to the Android emulator we ensured that our code doesn't have any accidental disk access or network access. We kept the strict mode on during all the development and testing period without noting anything strange.

Chapter 7

Conclusions and Future Work

This chapter summarizes what I've learnt, my conclusions, and introduces some examples of possible future works.

7.1 Conclusions

With the rapid technological advancements in Artificial Intelligence, Integrated Circuitry and increases in Computer Processor speeds, the present and the future of mobile computing looks increasingly exciting.

Thanks to this project, I was introduced to the current areas of research regarding mobile computing. It was worthy to understand and learn where research is moving on and the potentiality of distributed P2P architectures to improve our life. First, peer-to-peer systems can be used to set up real-time collaborative applications. Communication can either be the main purpose of these applications, like in Skype, or an enabler serving another purpose.

For instance, mobile peer-to-peer systems may be used for multiplayer gaming in order to allow players involved in the same game to exchange information together on their positioning and actions. In those applications, the use of peer-to-peer systems compared to a classical client-server approach significantly reduces the load on the server, and also helps reducing the message transmission delay, which is critical in real-time collaboration applications. For those applications, peer-to-peer systems prove being more scalable as they adapt to the number of connected peers by design.

On the other side I came into touch with the Android world. Android is the most used OS for mobile devices and it is open source. My personal experience with open source software never reached a higher level of satisfaction so far. Android makes development funny and easy, to me it is the better designed framework that Ive ever used. I was also impressed by its documentation; usually the documentation of open source software never satisfied me rendering necessary the buying of professional books. This is not the case if you're going to develop Android applications.

Over the course of the project, I have successfully been able to implement a distributed P2P game for the Android platform. It involved to use multithreading and manages concurrent accesses to the database in order to meet the project requirements, learn new programming techniques to meet the Android requirements.

7.2 Future Work

In order to become operative the project needs all its parts to be completed. The first project provided the communication layer through the Bluetooth network. This project (the second), provides the Game application and the Service which implements the collaborative p2p protocol. The third project should provide the central server system which will interact with the game to define initial parameters and get the final result.

7.2.1 Server application

The Database architecture proposed in **Chapter 3** illustrates the data to store on the central server and the relations between them. After the creation of the database the second step should be to

implement a GUI in order to facilitate the data manipulation by the administrator (who can be a non-technical person). Every table contained in the database should have its corresponding View, and all the Views together with statistical tools will form the Admin Control Panel. This is a key feature which gives the possibility to easily change the environment in which the next game will be played, and also set other game's data like start time, duration etc. Once this is done the next steps should be:

- implement the Server Component which listen for requests by peers, takes the data from the server DB and sends to them.
- extract useful information from the database in order to evaluate the game effectiveness, and present them in a friendly way through graphics and charts.
- implement a server side algorithm which lists the final ranking in order to reward the first three players.

Then the next step should be to integrate all together, this means:

- integrate the Communication layer component with the Service component
- integrate the new Server component with the Server Interface component on the game.
- turning back to the Server component, the environment can be created randomly. Why randomly?. Because it has been demonstrated to be the best way for testing real scenarios.

7.2.2 Further Communication Protocols and Operating Systems

he implementation so far has been restricted to work solely over Bluetooth, and no others.This because Android offers a very good support to Bluetooth, and for a lack of time, it would have been impossible to deliver the communication layer builded through WiFi.

In **Chapter 2** we introduced the LifeNet project which has the same motivation as ours. They are trying to offer the WiFi infrastructure, we are trying to offer the collaboration algorithm. Consequently, it would be profitable to extend our implementation of the communication layer also to WiFi network.

A final future work, in the case of big success, could be to implement the project for other operating systems like Apple iOS or Microsoft Windows Phone 8.

7.2.3 How to improve the game further?

The game can be further improved by adding new features in order to better simulate the scenario. Some ideas are the following:

- Add the Role field in the Player. It means for example that an ambulance which reaches its destination maybe its more helpful than a single person.
- Add the concept of "Group, to model a group of people/machines which move together.
- Add different types/levels of dangers. Of course, a unique danger is not realistic, so this feature brings the game closer to reality.

Many other features can be added, it's up to the next graduates find and implement them. I bet it will be funny.

Appendix A

Maintenance Manual

The Maintenance Manual provides instructions for building and running the P2P Game app android application,

A.1 Compiling and Running the System

Running on Handset Directly To run the application directly in the mobile phone you have to execute one of the following alternative steps:

1. Copy the compiled Android .APK file to the mobile device from your computer. Then, locate the file on the device storage, and run.

2. Send you .APK file to your email, then access your email and simply click on the .APK, this will start the installation process

3. Publish the .APK to the Android Market and then install it directly from Google Play.

The installer will then start - Accept the prompt. Once installed, the application will be available from the main device menu.

A.2 Running/Building from Source

To run from source, the phone must be connected to the computer using a USB cable. The PC must have Eclipse installed (available for all platforms) with the Android SDK installed. The Android 2.3.3 SDK should be selected. First, copy the project to your Eclipse workspace, import the Existing Project, then Open.

With the project open, right click on the project name, select “Run As and choose “Android Application. This will automatically re-compile the Android application, trans- fer it to the handset, then install and launch the application.

A.3 Requirements

The application requirements are explained in the following table:

Description	Minimum Requirements	Notes
Hardware	Mobile device with support for Bluetooth v2.1	Bluetooth standards higher than v2.1 are suitable
Operating System	Android 2.3.3 (Gingerbread)	
Disk Space	1MB	More disk space will allow for a larger message history to be stored internally
Memory	5MB	More RAM will all for the device to support the larger numbers of concurrent connections with ease.
Development IDE	This application was developed using Eclipse IDE along with the required Android plugins	.

Figure A.1: Requirements in order to be able to run the application

Bibliography

[Catarci et al.(2008)] Catarci, T. Sapienza-Univ. di Roma, Rome de Leoni, M. ; Marrella, A. ; Mecella, M. ; Salvatore, B. ; Vetere, G. ; Dustdar, S. ; Juszczyk, L. ; Manzoor, A. ; Hong-Linn Truong "Pervasive Software Environments for Supporting Disaster Responses", Internet Computing, IEEE, Jan.-Feb. 2008

[Sapateiro et al.(2009)] Sapateiro, C. Syst. Inf. Dept., Polytech. Inst. of Setubal, Setubal Baloian, N. ; Antunes, P. ; Zurita, G. "Developing collaborative peer-to-peer applications on mobile devices", 22-24 April 2009

[Blake et al.(2011)] Balke, T., De Vos, M. and Padget, J., 2011. "Analysing energy incentivized cooperation in next generation mobile networks using normative frameworks and an agent-based simulation." Future Generation Computer Systems, 27 (8), pp. 1092-1102.

[Almudena et al.(2007)] Almudena Daz, Pedro Merino Laura Panizo, and A lvaro M. Recio,"A Survey on Mobile Peer-to-Peer Technology", JCSD 2007, Torremolinos, Mlaga

[Buchan(2012)] Benjamin J. Buchan, "Peer-to-Peer Networks using Mobile Devices and Bluetooth", 2012 University of Aberdeen

[Matuszewski et al.(2006)] M. Matuszewski, N. Beijar, J. Lehtinen, and T. Hyyrylainen, "Mobile Peer-to-Peer content sharing application, in Consumer Communications and Networking Conference, 2006. CCNC 2006. 2006 3rd IEEE, vol. 2, 8-10 Jan. 2006, pp. 13241325.

[Wang et al.(2007)] A. I. Wang, T. Bjrnsgard, and K. Saxlund,"Peer2Me - rapid application framework for mobile peer-to-peer applications, in The 2007 International Symposium on Collaborative Technologies and Systems (CTS 2007), May 21-25 2007.

[Taylor & Harrison(2009)] Peer-2-peer environments. In Ian J. Taylor and Andrew B. Harrison, editors, "From P2P and Grids to Services on the Web", Computer Communications and Networks, pages 107125. Springer London, 2009.

[Wrox(2012)] Reto Meier, "Professional Android 4 Application Development" (Wrox Professional Guides), May 1, 2012

[Deitel et al.(2012)] P. Deitel, H. Deitel, A. Deitel,M Morgano, "Android for Programmers An App-Driven Approach", 2012