Android Walkie Mesh

Architectural Design Documentation

by

Clark Fourie 10243926

Ivan du Toit 29363412

Adriaan Louw 11205637

23 October 2013

Version 1.0

Change Log

Date	Version	Comment
21/06/2013	V0.0.1	Meeting discussing architectural specification.
22/06/2013	V0.0.2	Adding a thorough introduction.
23/06/2013	V0.0.3	Drafting of the documentation structure.
23/06/2013	V0.0.4	Detailed architectural requirements (Functional and nonfunctional requirements).
24/06/2013	V0.0.5	Drawing and inclusion of diagrams and technologies.
25/06/2013	V0.0.6	Architectural patterns, strategies and tactics.
26/06/2013	V0.0.7	Reference architecture discussion and inclusion.
26/06/2013	V0.0.8	Add a glossary, front page and content page.
28/06/2013	V0.0.9	Final changes to prepare for release (spelling, conversion to pdf, layout reconstruction etc.)
30/06/2013	V1.0	Final Version

Table of Contents

1. Introduction

1.1 Purpose

This document provides a high-level abstraction of the architectural strategies/tactics and foundations that are going to be implemented to meet the requirements specified in the requirement specification. The requirement specification also serves as a formal contract between the client and the developers.

NOTE: Terms used in this document that seem vague are described in the glossary.

1.2 Document Conventions

- Documentation formulation: MS Word (converted to pdf via CutePDF).
- Architectural schemas: MS Paint.

1.3 Project Scope

The scope of Android mesh PTT Walkie Talkie can be summarized as a software solution for a specific Wi-Fi enabled domain to enable medium ranged virtual communication and improve in terms of cost and quality. Providing this service should provide increased work capabilities because of the ease of communication the software provides. The long term goal would be for all employees to utilize the software as to ensure up-to-date, real-time communication and broadcasts from every level of the company/organisation. Eventually not just on devices that run Android operating systems but on all platforms.

1.4 References

- Essentials of Software Engineering, Second Edition, Frank Tsui and Orlando Karam.
- Pieter Botha en Alex Terlunen, by means of requirement elicitation and and formal meeting.
- The Serval Project Wiki (http://developer.servalproject.org/dokuwiki/doku.php)
- Serval Project GitHub Repository (https://github.com/servalproject)
- Scott, J. and Kazman, R. (2009) *Realizing and Refining Architectural Tactics: Availability*.

1.5 Related Documents

- Requirement Specification.
- Vision and Scope Document.

1.6 System Description

The goal of the system is to provide a service that will improve communication both horizontally as well as vertically in a task-orientated business environment. The system intends to replace quick phone calls with a simpler, cheaper and situationally more effective means to convey quick but crucial messages to colleagues that are not in the nearby vicinity.

2. Architecture Requirements

2.1 Architectural Scope

This system does not require a persistent or reporting infrastructure. Infrastructure wise all it needs is network to connect to that serves as the foundation (e.g. a WiFi) and other devices that run an Android OS and have the system installed as well. All of this is graphically depicted in the following diagram:

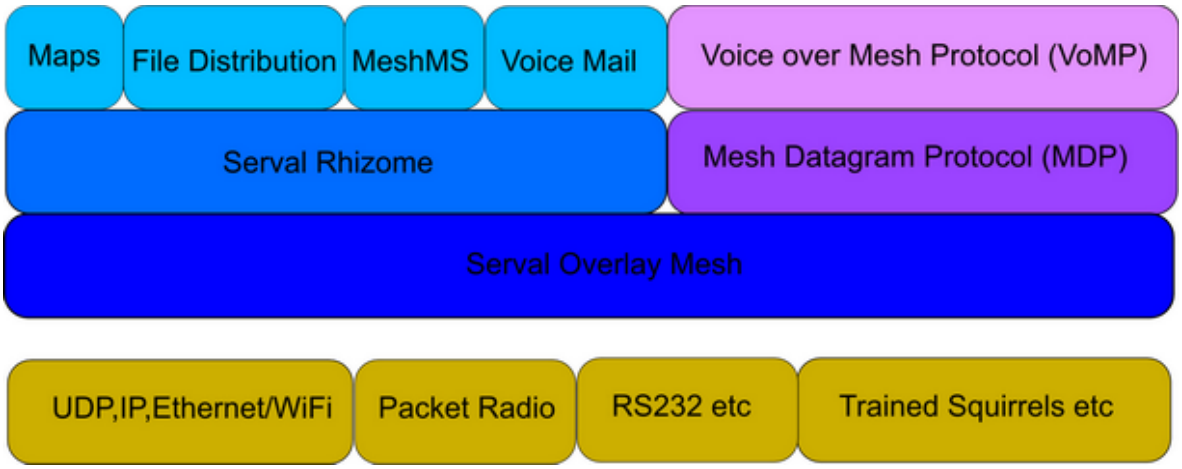

Figure 1 - Mesh overview

2.2 Quality Requirements

- Reliability - Various tests will be performed to ensure the system is reliable (including unit testing, integration testing and system testing) to ensure verification (conforms to requirements and specifications) and validation (the product meets the requirements and specification of the user) of the product.
- Ease of use - The client needs the system to be simplistic in the sense that no training must be necessary for the employees in order to fully utilize the system. Communication via the system must take place with the least amount of actions as humanly possible.
- Scalability - The system must be able to accompany multiple users even though the current user is not talking or texting via the app. The user must still act as a point that other users can “hop” of to reach other users.
- Portability - The system can be run on various Android models [from Android 2.2 (Froyo) to Android 4.2 (Jellybean)] as well on different devices such as tablets. A long term goal (as specified in the project scope) would be to make the system platform independent. The system also caters for multiple concurrent users.
- Flexibility/Reusability - The system is implemented with coding standards that will make code flexible and reusable (an example would be the string.xml file in the values folder of the project. All string values are kept in strings.xml so translation of the system can

easily be done).

- Security - The system is deemed as secure as it uses Native Android which provides intent based security and always uses the lowest possible security privilege. As for the communication section: It makes use of voice data encryption while the message is in transit.
- Performance - No explicit performance increasing measurements are going to be implemented. Optimization will be a “nice to have” after the project is finished but will not be the main goal. Reasonable performance will be provided through a solid architectural foundation (as described in architectural patterns and styles).
- Maintainability - We will later provide a complete user manual as well as a technical manual.

2.3 Integration and Access Channel Requirements

The human interface is a standard android user interface that should be streamlined to allow simple and zero configuration access to communicate with other parties.

The system may allow expose an API to other Android applications to query the mesh network state. The system may allow file sharing and messaging via the mesh network over and above the voice communication. A web server might also be provided to server static files from even if the client does not support the mesh technology.

2.4 Architectural Constraints

- The architecture must only use open source framework
- The system must be deployed on Android.
- Must make use of local standards (developing, coding etc.)

3. Architectural Patterns or Styles

Android applications make use of the Model View Viewmodel (MVVM) architectural pattern. Largely based on the MVC Pattern the MVVM strives to create platforms that supports event driven programming. MVVM makes use of data bindings to separate the GUI from the view layer so markup languages (e.g. XML) can be used to create bindings to the view model. This allows interactive designers to focus on UXD rather than business logic, making it possible for the layers of application to be developed in multiple streams to increase productivity.

The frontend android and the backend JNI code will form two layers that are connected with adapters to allow both of the layers to develop independently.

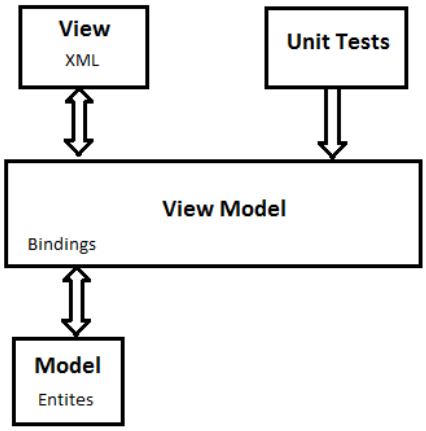

Figure 2 - MVVM

4. Architectural Tactics or Strategies

Tactics used in the system can be summarised as availability tactics. To be more specific **fault detection and fault recovery.** These tactics were chosen to help us release a version of the system as soon as possible - not necessarily a complete or optimal system but one that is in a working condition and serve as a foundation to build upon. The above mentioned tactics provide a means to do exactly that.

Firstly we need to define the model for which we are developing the system. Even though the system runs on tablets as well our main focus is on cellular phones as depicted in *Figure 3*

For **fault detection** we will use the **heartbeat tactic** visually depicted in *Figure 5*

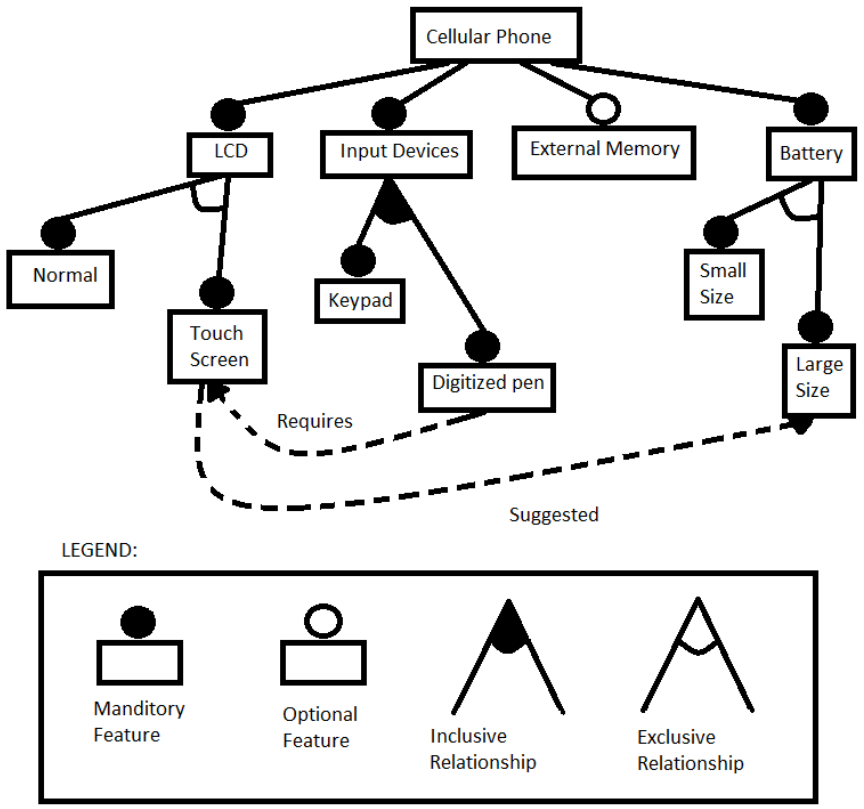

Figure 3 - Cell Phone Model

Fault recovery we will be handled using sparing (aka cold spare [*Figure 4*). This technique is implemented because of its simplicity and it works well for systems that want high reliability (Scott and Kazman: 2009).

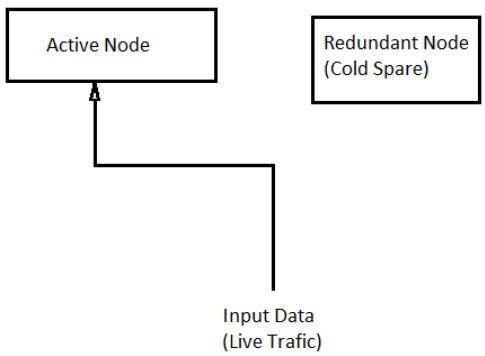

Figure 4 - Cold Sparing

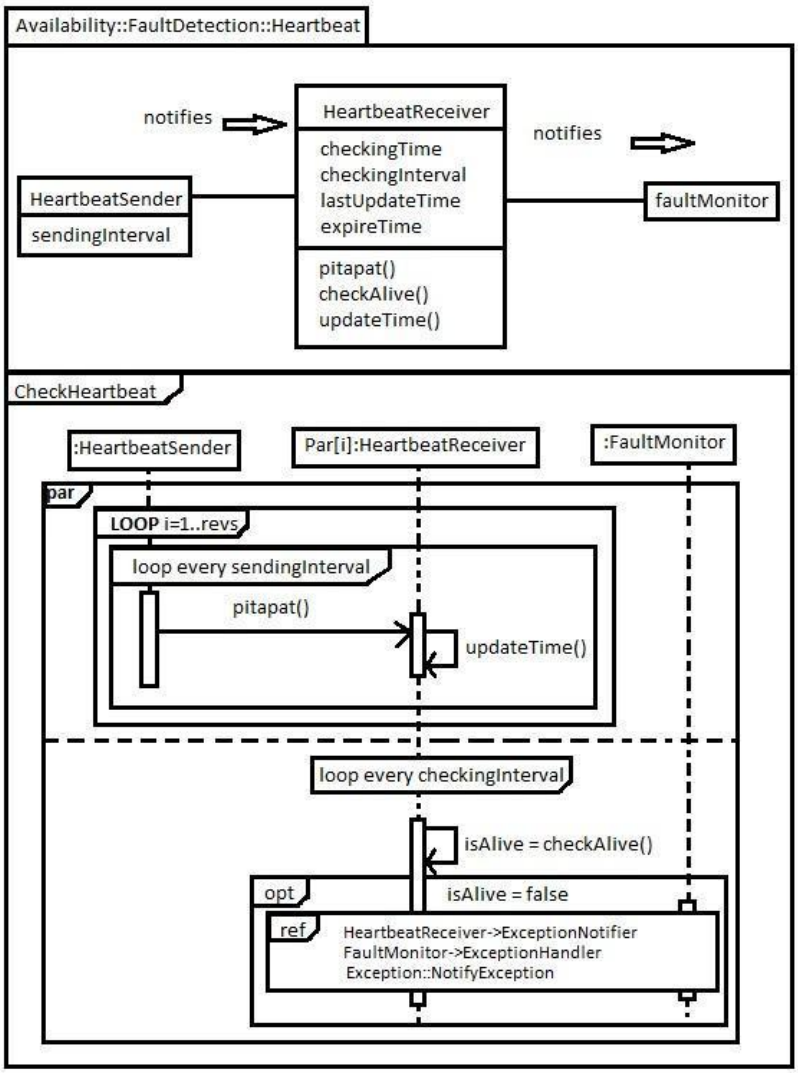

Figure 5 - Heartbeat

5. Use of Reference Architectures and Frameworks

To develop the application we will be using the Android framework for the frontend, at the moment it is not optional to use on an Android device. This framework allows us to skip a lot of boilerplate code and provides security and logging functionality out of the box.

6. Technologies

The project will be developed for Android which is a Linux based operating system backed by Google and other members of the Open Handset Alliance. As such the Java programming language will be used. Java is an object-oriented, high-level programming language similar to C++ but simplified to eliminate language features that cause common programming errors. In

addition to this some of the mesh technology is also written in C and C++ which is compiled and integrated with the project with JNI (Java Native Interface).
To compile the system GNU make is used for the native compilation. This system was built and tested on Linux so it is the suggested build system.

7. Glossary

JNI - Java Native Interface.

PTT - Push to Talk.

MVC - Model View Controller.

OS - Operating System.

GUI – Graphics User Interface.

XML – Extensible Mark-0up Language.

UXD – User Experience Design.

Heartbeat tactic - Consists of the periodic message exchange between the system monitor and the process which will indicate the to the system monitor where the fault is incurred.

Cold spare - refers to a configuration where a component must be manually adjusted in the event of problems or, worst case, system failure. In this situation we will have a redundant node that does not interact with the active node (see *Figure 1.3)* . It simply replaces the active node in the case of failure.

Boilerplate code – Repetitive code that can and should be used in a reusable way.