Final Architecture Document

Confide Instant Messaging Application

Table of Contents

2 Definitions and Acronyms

Definition/Acronym	Description
API	Application Programming Interface
CIMA	Confide Instant Messaging Application
CCRD use case	Critical, Core, Risky Difficult use case
OpenFire	Open source XMPP-based RTC server
Pinnacle	Pinnacle Corporation Ltd
RTC	Real Time Collaboration
XEP	XMPP Extension Protocol
XMPP	Extensible Messaging and Presence Protocol

3 Introduction

This document describes the final architecture that has been determined over the course of the elaboration iterations for the Confide Instant Messaging Application. This architecture has been demonstrated to provide complete end-to-end support for the critical, core, risky and difficult use cases (CCRD).

4 Project Vision and Requirements

4.1 Vision

Project vision is covered separately in the document *Project Vision.docx*. A brief summary of Pinnacle Corporation’s vision for the Confide application is:

> “
>
> An instant messaging tool using best-of-breed standards-based technology to deliver a communication solution which is clearly Pinnacle’s own. Actively-developed open source software will be used to ensure that Pinnacle retains control of the product and can continue to develop and innovate it to meet the evolving needs of the business.
>
> ”

4.2 Functional Requirements

Full details of the functional requirements are provided in *Requirement Model.docx*. The project’s architecture is explained in terms of how it solves the CCRD use cases.

4.2.1 CCRD Use Cases

The CCRD use cases for this project are the logistical requirements for two (or more) users to be ready to initiate or receive a chat message, then to actually send messages. This involves the following features:

- Create an account
- Add a user to contacts
- View contacts
- Chat with contacts
- View chats with contacts

4.2.1.1 Client/Server Architecture

We chose to use a client/server architecture. For a corporate software product, this architecture is particularly beneficial for a number of reasons:

- Administration and corporate data is centralised. This is suitable for typical IT management and can be scaled up or down as required.

- If using a standards-based communication protocol, either the server or client can be replaced or upgraded with excellent chances of success

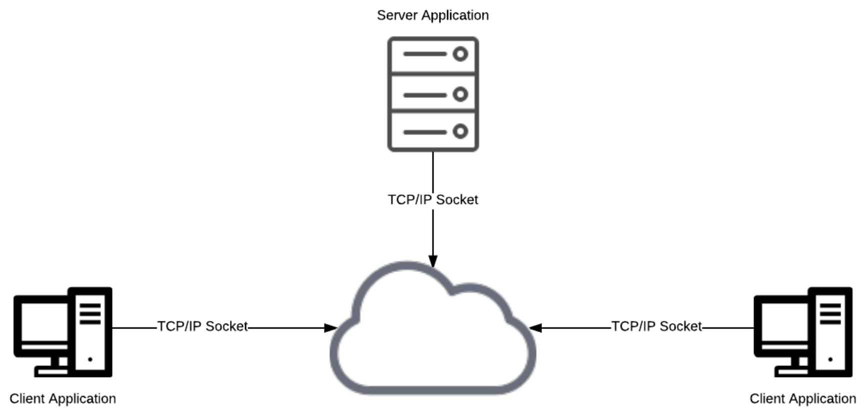

Figure 1

4.3 Core Communication Protocol

The Messaging Service requires a common communication framework to meet the requirements of the project. XMPP is a long standing and popular communication protocol for messaging and presence services. Development on the protocol starting in 1999, it has long term support and is used on significant Platforms such as Google Talk and Atlassian HipChat. The XMPP protocol also relies upon a large base of similarly mature and well-maintained Open Source projects. **Table 1** shows how the protocol satisfies various non-functional project requirements.

Table 1 – XMPP Protocol NFR Suitability

Key Mechanisms	**Non-functional Requirement**[1]	**Priority**
IEEE International Standard, Open Source Platform	3.3	Medium
Middleware-oriented Architecture	4.2, 4.3, 3.4	High
P2P Encryption	4.1	Low
Asynchronous Messaging Relaying.	3.2	High
User Presence Relaying.	2.7	High

Centralised Database	4.4	Low
Architectural Neutral	3.3	Medium

Note: 1. Refer to Project Vision for details of these non-functional requirements.

5 Data Model

The applications main(String) is contained within the App.java, the core structure of the application is built around this. The core software architecture is implemented with the model-view-controller (MVC) pattern. This pattern is organised with the following Java Packages:

Java Package	**Description**
controllers	MVC controlling functions
lambda	Lambda interfaces for event handling
models	MVC data models
views	MVC GUIs
xmpp	Base XMPP functionality
exceptions	Exception interfaces for custom events
resources	Image components

Team Orange has prepared two diagrams to illustrate the data model used. Links to these diagrams are provided below:

Component Diagram

Deployment Diagram

6 Technical Implementation

Team Orange has chosen the Java-based OpenFire XMPP server for this project. OpenFire is a mature project which enjoys a large user base and ongoing development.

6.1 Data storage

OpenFire can use any relational database system which supports JDBC 2.0. This includes the following databases:

- MySQL
- Oracle
- Microsoft SQLServer
- PostgreSQL
- IBM DB2
- HSQLDB

The Open Fire servers will be located in a data centre (or platform as a service) with a minimum guaranteed up time of 99.9% (i.e. less than 45 minutes offline per month)

XMPP implementations support the use of directory services. This means it is possible for users to have single sign on authentication to use the service. As with corporate email, Pinnacle will make it clear to users that in the event of an investigation, user's chat data may be accessed. This will be possible because all chat communications will be stored on the database which is controlled by Pinnacle.

For alpha and beta testing, the simple embedded Java relational database provided by OpenFire will be used.

6.2 Other non-functional requirements

Lists other important non-functional requirements which the OpenFire XMPP server supports.

Table 2 – OpenFire XMPP Server NFR Suitability

Key Mechanisms	**Non-functional Requirement**[1]	**Priority**
Central Administration	4.2, 4.3	Medium
P2P Encryption	4.1	Low
Native Data Storage	1.3	High

Platform Independent	3.3	Medium
Can support large user base >10,000	1.1	High
Archiving messages	1.2	Low
Offline message retrieval	1.3	High
Multi User Chat	1.9	High
User Administration	1.4, 1.5, 1.6, 1.7, 1.8	High

Note: 1. Refer to Project Vision for details of these non-functional requirements.

7 XMPP Client Implementation

7.1 Client API Language

We have decided to use Java for the client side, as we all know java quite well. Java is also quite high level and therefore fast and easy to work in (NFR 3.5), tried and tested, and secure (NFR 3.6). Java being strictly object orientated will be helpful as we are designing and modelling this system in an object-oriented way. Finally Java's cross platform capabilities will make it easy for us to have our client side application be compatible with windows, os x, and linux (NFR 2.2). Therefore, only Java APIs will be considered.

7.2 Client API

Babbler was chosen as the XMPP client API for the project. Babbler is relatively new and is based on learning and improvement from older frameworks such as Smack. In terms of the XMPP features supported, both Babbler and Smack were quite evenly matched, however after initial testing of simple use cases, Babbler proved to be easier to use.
Table 3 shows the basis for the evaluation of the client APIs.

Table 3 – Babbler API NFR Suitability

Key Mechanisms	**Non-functional Requirement**[1]	**Priority**
Architecture Neutral Pure Java Implementation	3.5	Low
Opensource	2.6, 3.1	High
Object Oriented	3.1, 3.5	High
Abstraction from XMPP protocol, XML, and XMPP XML format.	3.5	Medium
Great documentation	3.5	Medium
Works well with OpenFire server		High
Gradle/Maven support		Medium
Well tried and tested (Stable)	3.5, 3.6	High

Multi-user chat	FR 2.1	High
Presence of other users	FR 4.5	High
Contact Lists		Medium
Instant Messaging	3.2	High
Multi-platform	2.2	High

Note: 1. Refer to Project Vision for details of these non-functional requirements. FR is functional requirement.