Detailed Design Report for Conqi, the Mobile Conqueror

Prepared by Tolga AKIN, Çiğdem AVCI, Güliz Seray TUNCAY and Hüseyin ULUSOY

Middle East Technical University
Table of Contents

1. Introduction ........................................................................................................... 3
   1.1 Purpose ............................................................................................................... 3
   1.2 Scope .................................................................................................................. 3
   1.3 Definitions, Acronyms and Abbreviations .......................................................... 3
   1.4 References ......................................................................................................... 3
   1.5 Team .................................................................................................................... 3
      1.5.1 Organizational Structure ............................................................................. 3
      1.5.2 Member Roles ............................................................................................... 4
      1.5.3 Process Model .............................................................................................. 4
   1.6 Overview ............................................................................................................ 4
   1.7 Product Perspective ............................................................................................ 5

2. System Design .......................................................................................................... 5
   2.1 Architectural Design ........................................................................................... 5
      2.1.1 Composition Decomposition ....................................................................... 5
      2.1.2 Module Decomposition ................................................................................ 7
      2.1.3 Data Decomposition ..................................................................................... 20
         2.1.3.1 Data Design .......................................................................................... 20
            2.1.3.1.1 ClientData Package ....................................................................... 20
            2.1.3.1.1.1 ClientData Class ......................................................................... 20
            2.1.3.1.2 GPSInfo Class ............................................................................... 20
            2.1.3.1.3 Compass Class ............................................................................... 21
            2.1.3.1.4 Picture Class .................................................................................. 23
      2.1.3.2 Database Design ....................................................................................... 26
         2.1.3.2.1 Entities .............................................................................................. 26
         2.1.3.2.2 Relations ............................................................................................ 26

3. Procedural Design .................................................................................................. 28
   3.1 Install Function ................................................................................................... 28
   3.2 Save Picture Function ........................................................................................ 28
   3.3 Send Picture Function ....................................................................................... 30
   3.4 Take Picture Function ....................................................................................... 31
   3.5 Get More Information Function ......................................................................... 31

4. Interfaces ................................................................................................................. 34
   4.1 Graphical User Interface Design ......................................................................... 34
      4.1.1 Application Entrance Point ........................................................................ 34
      4.1.2 Main Window ............................................................................................... 35
      4.1.3 Result Window ......................................................................................... 35
      4.1.4 Get More Information Menu .................................................................... 36
   4.2 Hardware Interfaces ........................................................................................... 36
   4.3 Software Interfaces ............................................................................................ 36

5. Libraries and Tool .................................................................................................. 37
   5.1 Languages and platforms .................................................................................... 37
      5.1.1 Java .......................................................................................................... 37
      5.1.2 C++ ........................................................................................................... 37
      5.1.3 Eclipse ...................................................................................................... 37
   5.2 Database Connectivity Systems .......................................................................... 38
      5.2.1 JDBC (Java Database Connectivity) ............................................................. 38
   5.3 Web Service Technologies ................................................................................... 38
      5.3.1 Apache Axis ............................................................................................... 38
   5.4 Operating Systems and Relational Technologies ................................................. 39
      5.4.1 Android Emulator ....................................................................................... 39
      5.4.2 DroidDraw ................................................................................................. 39
1. Introduction

1.1 Purpose

This document includes the detailed design of Conqi, the Mobile Conqueror. The purpose of this document is to provide a detailed design description of Conqi.

1.2 Scope

This detailed design document contains design information for Conqi expressing the demonstration of architecture, modules and their interactions, classes, use cases by showing how it interacts with the outer world, functionalities, features, database model, graphical user interface, tools, special technologies and the implementation details. It describes in detail how Conqi works properly and in a safe manner.

1.3 Definitions, Acronyms and Abbreviations

See Appendix A, the Glossary

1.4 References

See Appendix B, References

1.5 Team

1.5.1 Organizational Structure

Our team does not have a team leader to lead and control the flow of development. Instead, every member of the group has equal right to state his/her thoughts, put forward an opinion, make a recommendation or object to a decision. All decisions are made on an agreement in the judgment or opinion reached by the group as a whole.
1.5.2 Member Roles

- Çağdem Avcı              Initiator, Optimist
- Hüseyin Ulusoy           Time Keeper
- Tolga AKIN               Devil's Advocate
- Güliz Seray Tuncay       Devil's Advocate, Time Keeper

1.5.3 Process Model

Our project combines different topics into one. We will need to use a lot of different technologies such as GPS, Image Processing, Mobile Communication and so on. This makes the project harder and more prone to errors. Because of that, we will need to identify requirements, do prototyping, coding and testing again and again. The process model that best fits our requirements is the “Spiral Model”. The spiral model starts with defining system requirements, creating a preliminary design and constructing a first prototype. The following prototypes are constructed by evaluating the previous prototype, defining the requirements of the next prototype, planning and designing the next prototype and building and testing the next prototype.

1.6 Overview

This detailed design description document contains design information about Conqi and The rest of the document is divided into chapters for better understanding.

- In chapter 2 the architectural design is discussed.
- In chapter 3 the procedural design is analyzed.
- In chapter 4 graphical user, hardware and software interfaces are discussed.
- In chapter 5 libraries and tools are specified.
- In chapter 6 schedule of the project is presented.

This document is intended for

Developers: in order to be sure they are developing the right project that implements the design described in this documentation.
**Testers:** in order to have an exact list of the features, functions, models, diagrams and tools that have to respond according to design and provided diagrams.

### 1.7 Product Perspective

Conqi is a self-contained product that is, it is not a component of a larger system. Apart from using GPS information to determine the location of the user, it will use information extracted from the DB which makes it a unique product in its way of supplying the user detailed information by using image processing techniques within the DB provided by the administrator of the system.

### 2. System Design

#### 2.1 Architectural Design

##### 2.1.1 Composition Decomposition

*The Conqi* consists of three main components namely Web Server, Client and Database Server. The application connects to J2EE Server and this web server connects to the Oracle Database via JDBC interface.

![Deployment Diagram](image)
Figure 2.2 – Component Diagram of Conqi
As stated in the deployment diagram (Figure 2.1), there are three main modules of the product: Web Server, Client and Database Server.

- User Application component is the main component of the Client Module containing Server Connector and Location Info Extractor components.
  - Location extractor component is responsible for extracting GPS and compass information.
  - User Application component mainly serves to connect the client to the J2EE server via SOAP interface.

- J2EE server component is the main component of the Web Server module which contains Image Processing, Location Finder and User Informer components inside.
  - Image Processing component serves to find a match between the images in the database and the picture taken by the user. JDBC interface is used to connect to Oracle database.
  - Location Finder component is responsible for extracting information from Google Maps using Google Maps API. The information can be name of the location or some images.
  - User Informer component connects to Wikipedia by using MediaWiki API. Related landmark information is extracted and shown to the user.

- Database Server module contains Oracle Image DB component.

### 2.1.2 Module Decomposition

The Conqi consists of several packages namely **Server, Client, UserData, DatabaseModel, Users, Installer** Packages.

- **Server Package** is responsible for handling the operations taken by the client.
- **Client Package** is responsible for extracting related user information, sending them to the server and supplying the user of the application related information sent by the server.
- **UserData Package** holds the information of GPS location, compass and the picture itself.
- **DatabaseModel Package** holds the information of the database tables as objects.
- **Users Package** handles the user actions and passes them to the server.
- **Installer Package** is responsible for the install and uninstall actions taken by the user.
Figure 2.3 – State Diagram of Conqi
In Figure 2.3, the states of Conqi are presented. When the application starts, it waits for an input from the user. The user either takes a picture or exits from the application. As soon as the user submits the image to the application, GPS and compass information is collected by Conqi. Immediately after, the input (image, GPS, compass) is delivered to the server. Server completes his processes and returns the result to the client as the name of the location. While presenting this result to the user, detailed information about the landmark will be searched through web. Thereafter, the user decides what to do next out of the options provided which are viewing more information, saving picture and sending picture as message.
Figure 2.5 - Client Package and its subpackages

Figure 2.6 - Server Package and its subpackages
Client Package

MachineCommunication Class
This class is responsible for obtaining GPS location and compass information from the smartphone itself.

ClientPosition Class
This is the data class for keeping GPS information and compass information.

MainController Class
This is the class for enabling the application to return the GPS information and compass information. (Later on, MachineCommunication Class helps to get the related information from the phone.)

SCommunication Class
The responsibility of this class is to connect to the database and sending user information to the server.

ServerResponse Class
This is the data class for saving the resulting landmark information.

DisplayMessage Class
The message that will be displayed to the user will be kept in this class.

Parser
The MediaWiki API returns wikicode format as result and this result must be parsed. This class is responsible for parsing the wikicode formatted result.

MediaWikiConnector
This class enables the application to connect to the MediaWiki API.

Server Package

S2GMCommunication Class
This class is the main class which communicates with the Google Maps API.

MapInfo Class
This is the data class for representing the Map information returned by Google Maps API.
**S2CCommunication class**

This is the main class which handles the user action. The user information is sent to this class.

**LandMarkInfo**

This class is a data class representing the information about the landmark.

**ImageProcessor Class**

This is the main class handling the image processing process. The class is mainly responsible for finding the correct match in the database, given the user picture.

**RangedImage Class**

Features of images in the range of given GPS location are kept in this class.

**ImageProcessorInfo Class**

ImageProcessorInfo class is created to hold the data such as image, GPS and Compass information which are obtained from the Database.

**Users Package**

**User Class**

The user actor will call the operation of that class.

**Admin Class**

Admin class is capable of handling the database operations.

**Installer Package**

**InstallationWizard**

This class is responsible for installation process.

**UninstallationWizard**

This class is responsible for uninstallation process.
**Phone Package**

**PhoneController**

This class is responsible for accessing the features of phone like sending image as a short message and saving pictures to the file directory.

![Figure 2.7 Client Package Classes](image-url)
Figure 2.8 Server Package Classes
Detailed Design Report for Conqi, the Mobile Conqueror

Figure 2.9 Users Package Classes (Top View)

Figure 2.10 Users Package Classes and Database Model Package Classes (Part 1)
Detailed Design Report for Conqi, the Mobile Conqueror

DatabaseModel Package

- **Location**
  - Attributes
    - latitude : float
    - longitude : float
    - direction : float
    - altitude : float
  - Operations

- **User**
  - Attributes
    - name : String
    - lastName : String
    - IMEI : String
    - productNo : int
  - Operations

- **HasLocation**
  - Attributes
    - imagInfo : ImagInfo
    - location : Location
  - Operations

- **Features**
  - Attributes
    - featureID : int
    - regionBased : int
    - edge : int
    - histogram : int
    - invariant : float
    - gaborFeatures : int[*]
    - weightParameter : int
    - interestPoint : int
  - Operations

- **Admin**
  - Attributes
    - ID : int
    - name : String
    - lastName : String
    - password : String
  - Operations

- **SentBy**
  - Attributes
    - imagInfo : ImagInfo
    - user : User
  - Operations

- **ImagInfo**
  - Attributes
    - image : IPLImage
    - ID : int
    - locationName : String
    - metaData : String
  - Operations

- **AddedBy**
  - Attributes
    - admin : Admin
    - imagInfo : ImagInfo
  - Operations

Figure 2.11 Users Package Classes and Database Model Package Classes (Part 2)
Detailed Design Report for Conqi, the Mobile Conqueror

**Figure 2.12 - Data Flow Diagram - Level 0**

**Figure 2.13 - Data Flow Diagram - Level 1**
Detailed Design Report for Conqi, the Mobile Conqueror

Figure 2.14 - Data Flow Diagram - Level 2 (Top View)

Figure 2.15 - Data Flow Diagram - Level 2 (Part 1)
The application sends image along with GPS and compass information to the J2EE server in order to find the name of the landmark by using Google Maps API. GPS information consists of three float values named longitude, latitude and altitude and compass information is a float value indicating the Azimuth angle between the north. Google Maps API returns us the landmark name and success information indicating a successful or unsuccessful match for the location. If Google
Maps API is not able to identify the landmark name, image processing module will be invoked. Image along with GPS information will be sent to the Image Processing module and the name of image that is matched with user's picture will be returned. Then, this landmark information will be sent to the client and a query is sent to MediaWiki API. A wikitext is returned and parsed by the WebCommunication module. Related information is shown to the user with an appropriate GUI.

2.1.3 Data Decomposition

2.1.3.1 Data Design

2.1.3.1.1 UserData Package

2.1.3.1.1.1 ClientData Class

This is the data class for keeping user location information and picture itself.

2.1.3.1.1.2 GPSInfo Class

This is the class keeping the longitude, latitude and altitude.
The Android platform supports location-based services for obtaining location information. Location-based services makes it possible to obtain geographical location information using techniques such as GPS and Networking.

android.location Package

Classes and interfaces that organize Android location-based and related services are defined in the android.location package.

Classes

Classes defined in android.location that we are going to use for obtaining geographical location information are as follows:

• Location

• LocationManager
Location
This is the class representing a geographic location obtained at a particular time. A location object includes a latitude and longitude and other data.[1]

LocationManager
This class is used to access the system location services. By using these services, we get periodic updates of the device's geographical location.[2] MachineCommunication class will request location updates using the requestLocationUpdates() function.

2.1.3.1.3 Compass Class
This class saves the compass information.

A mobile device might have numerous hardware sensors for detecting a variety of physical quantities. Orientation, acceleration, light, magnetic field, proximity and temperature are some of them. In the past, being unable to access hardware features such as these was a huge source of frustration for mobile developers. Android Platform, however, bridges the gap between the hardware and software, bringing much of the hardware's capability to the surface.

android.hardware Package
Sensors on the mobile device are accessible through the android.hardware package.

Classes
Classes defined in android.hardware that we are going to use for obtaining compass information are as follows:

• Sensor
• SensorEvent
• SensorManager
Sensor
Sensor is the class representing a physical sensor on the mobile device. The constant TYPE_ORIENTATION (0x00000003) describes the sensor type that is used for getting orientation info.[3]

SensorManager
This is the class that enables accessing the device's sensors. An instance of this class is obtained by calling Context.getSystemService() with an argument of SENSOR_SERVICE.[4]

SensorEvent
This class represents a sensor event and contains the sensor type, the time-stamp, accuracy and the sensor's data.[5] The field “values[]” will contain the yaw, pitch and roll data.

![Figure 2.17 - Roll, Pitch, and Yaw planes of motion.][6]

Interfaces
We are going to use the interface “SensorEventListener” defined in android.hardware.

SensorEventListener
This interface is used for receiving notifications from the SensorManager when a sensor value has changed. Following a change, the function onSensorChanged(SensorEvent event) will be called, passing the sensor event.[7]
2.1.3.1.4 Picture Class

The class keeps the pictures.
Figure 2.19 - UserData Package Classes (Part 1)
Figure 2.20 - UserData Package Classes (Part 2)
2.1.3.2 Database Design

2.1.3.2.1 Entities

Admin

Admin entity holds the name, id, last name and password information of the administrator of the server.

Image_info

Image_info entity holds the images to be processed during picture matching with the picture taken by the mobile phone together with the metadata and location info of the images.

Users

Users entity holds the information of the user on the client side such as name, last name and IMEI of the client.

Features

Features hold the important features which are going to be used during image processing part of the application such as region based features, edge, histogram, invariant_features, gabor_features, weight_parameter, interest_points.

Location

Location holds the location information of each image such as latitude, longitude, altitude and compass.

2.1.3.2.2 Relations

Added_by

Added_by relation relates images added by administrators

Sent_by

Sent by relation relates users with images sent by users

Has_features

Has_features relation relates images with features it has
Has_location

Has_location relation relates the location information with images.

Figure 2.21 - Entity Relationship Diagram
3. Procedural Design

3.1 Install Function

In order to be able to use the application, the user must be first purchased and installed to the smart phone. This process is handled by the InstallationWizard.

![Install Sequence Diagram](image)

*Figure 3.1 - Install Sequence Diagram*

3.2 Save Picture Function

This function helps the user to save the tagged image on the phone.

The images will be saved in the mobile phone in native format and related information will be saved in an XML file. Android supports some image formats such as JPG, PNG and GIF. Therefore, Conqi’s mobile application will save images in these formats. The user can access previously captured images in a folder, indicated when the application installed. The images can be viewed and changed by other Android applications. Moreover, image related data will be saved in an XML file. The file will be in the following XML format.
Figure 3.2 - Save Picture Sequence Diagram
3.3 Send Picture Function

This option enables user to send the tagged image to other mobile phone users by multimedia messaging.

![Send Picture Sequence Diagram](image)

*Figure 3.3 - Send Picture Sequence Diagram*

The user will have the option to send the image to a friend. The action of sending the image to somebody else will be carried out using the Multimedia Messaging Service (MMS) protocol. This way, the user will be able to send the image to anybody, even if the recipient does not have the Conqi application, or even a device that runs the Android Platform. The only requirement is that the user have a mobile device that supports the MMS protocol.

The application will send the MMS using the Intent class.

**Intent Class**

Defined in the android.content package, the Intent class represents an abstract description of an operation to be performed. It can be used to generate bindings between different applications and launch different activities.\(^8\) The application will generate a new Intent with action Intent.ACTION_SEND. The message body and the picture will be passed to the Intent object and
the send activity will be started via a call to startActivity(). This will open up the messaging application with the image attached. The user will modify the message if he/she wishes to do so, and enter the recipient's phone number. Then, the MMS will be sent to the recipient.

The Android platform does not provide an interface to send an MMS message programmatically (i.e. without firing up the messaging application), because this would be a security flaw.

### 3.4 Take Picture Function

By this option, the users takes the photograph and it is sent to the server to be processed for obtaining more information. At the end of this processes user will be supplied with more information related to the landmark.

### 3.5 Get More Information Function

If the user desires to get more information related to the landmark, a browser will be opened and she will be directed to the Wikipedia web page.

![Sequence Diagram](image)

*Figure 3.4 - Take Picture & Get More Information Sequence Diagram (Top View)*
Detailed Design Report for Conqi, the Mobile Conqueror

Figure 3.5.1 - Take Picture & Get More Information Sequence Diagram Part 1
Detailed Design Report for Conqi, the Mobile Conqueror

Figure 3.5.2 - Take Picture & Get More Information Sequence Diagram Part 2

**PSEUDO-CODE**

```java
takeImage()
{
    boolean result
    GPSCompassInfo := getLocation()
    result := processInfo()
    if(result != "unknown place")
        then return result
    else
        result := matchImage() // using image processing aspects
        return result
}
```
4. Interfaces

4.1 Graphical User Interface Design

During graphical user interface design, simplicity should be strictly taken into account. Since functionality is the main purpose of the GUI design, developing simple functions for user is what has been done so far.

An easy to use and clear GUI should be;

- **Consistent**
  
  A consistent color scheme should be used together with consistently placed tools like buttons.

- **Easy to navigate between windows**
  
  Not to bother user with solving the structure of how pages are connected, the flow between screens should be coinciding with the work which user is trying to accomplish.

- **Effectively labeled**
  
  Since the main information sources are the texts that are going to be presented on the screen to the user, they shouldn't be poorly worded and they should be easily understood.

4.1.1 Application Entrance Point

The application will start by double clicking the icon presented to the user on the main menu and “Conqi” will be the user friendly abbreviation of the application name “Conqueror”.

*Figure 3.6 - Application Entrance Point*
4.1.2 Main Window

This is the menu of our application. The user will be able to take a picture, send a previously-tagged picture to other users and uninstall the application through this menu.

*Figure 3.7 - Menu of the application*

4.1.3 Result Window

This is the screen the user will encounter, when she has already chosen the “Take Picture” option. After taking the picture, she will see the still image of the landmark with information given below it. Also, she may want to disable tagging by using the check box given below the picture. Additionally, she will be supplied with more options and she will see those options if she presses “More Options” button.

*Figure 3.8 - Result of Take Picture Action*
4.1.4 Get More Information Menu

If the user wishes to see other options and she has already pressed “More Options” button, she will encounter with this screen. Through this screen, she will be able to get more information about the landmark that she has previously viewed. Furthermore, she may send the tagged picture to other mobile phone users. Additionally she may save the picture to her phone if she chooses “Save” option.

Figure 3.9 - Get More Information Menu

4.2 Hardware Interfaces

Machine interfacing is normally a very specific thing. The manufacturer normally provides an API (Application programmer interface) dll for interfacing with the hardware. Simply, main hardware interfaces of Conqi will be listed as touch interface of the phone and ports of the server.

4.3 Software Interfaces

Firstly, JDBC is going to be used as the software interface used to reach the database. In addition to that, WSDL is and XML format for describing network services as a set of endpoints operating on messages containing either document oriented or procedure oriented information. The operations and messages are described abstractly, end then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL[9] is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.
5. Libraries and Tools

5.1 Languages and platforms

5.1.1 Java

Java is a programming language originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture.[10]

5.1.2 C++

C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language. It is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features. It was developed by Bjarne Stroustrup starting in 1979 at Bell Labs as an enhancement to the C programming language and originally named "C with Classes". It was renamed to C++ in 1983.

Some of its application domains include systems software, application software, device drivers, embedded software, high-performance server and client applications, and entertainment software such as video games. Therefore, C++ is used for image processing part of mobile conquerer because of the speed constraints.[11]

5.1.3 Eclipse

Eclipse is a multi-language software development environment comprising an IDE and a plug-in system to extend it. It is written primarily in Java and can be used to develop applications in Java and, by means of the various plug-ins, in other languages as well, including C, C++, COBOL, Python, Perl, PHP, and others. The IDE is often called Eclipse ADT for Ada, Eclipse CDT for C, Eclipse JDT for Java and Eclipse PDT for PHP.[12]
5.2 Database Connectivity Systems

5.2.1 JDBC (Java Database Connectivity)

JDBC is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases.

JDBC allows multiple implementations to exist and be used by the same application. The API provides a mechanism for dynamically loading the correct Java packages and registering them with the JDBC Driver Manager. The Driver Manager is used as a connection factory for creating JDBC connections.

JDBC connections support creating and executing statements. These may be update statements such as SQL's CREATE, INSERT, UPDATE and DELETE, or they may be query statements such as SELECT. Additionally, stored procedures may be invoked through a JDBC connection. [13]

5.3 Web Service Technologies

5.3.1 Apache Axis

Apache Axis is an open source, XML based Web service framework. It consists of a Java and a C++ implementation of the SOAP server, and various utilities and APIs for generating and deploying Web service applications. Using Apache Axis, developers can create interoperable, distributed computing applications. Axis is developed under the auspices of the Apache Software Foundation.[14]

Besides, as a new version Apache Axis2 is a core engine for Web services. It is a complete redesign and re-write of the widely used Apache Axis SOAP stack. Implementations of Axis2 are available in Java and C.[15] Axis2 not only provides the capability to add Web services interfaces to Web applications, but can also function as a standalone server application. Axis2 has support for REST by just removing the SOAP headers both on the client and on the server.
5.4 Operating Systems and Relational Technologies

5.4.1 Android Emulator

The Android SDK includes a mobile device emulator -- a virtual mobile device that runs on your computer. The emulator lets you prototype, develop, and test Android applications without using a physical device. The Android emulator mimics all of the typical hardware and software features of a typical mobile device, except that it can not receive or place actual phone calls. It provides a variety of navigation and control keys, which you can "press" using your mouse or keyboard to generate events for your application. It also provides a screen in which your application is displayed, together with any other Android applications running. [16]

5.4.2 DroidDraw

DroidDraw is a graphical user interface (GUI) builder for the Android platform. DroidDraw has licensed under the GNU General Public License v2. [17]

Figure 3.10 - Interface of DroidDraw
5.5 Additional APIs and Libraries

5.5.1 MediaWiki API

The goal of MediaWiki API (Application Programming Interface) is to provide direct, high-level access to the data contained in the MediaWiki databases. Client programs can use the API to login, get data, and post changes. The API supports thin web-based JavaScript clients, such as Navigation popups or LiveRC, end-user applications (such as vandal fighter), and can be accessed by another web site (tool server's utilities).

We are planning to use MediaWiki API to retrieve more information about the place in the photo taken by the mobile application. The databases of MediaWiki can be accessed by some libraries in different type of programming languages. JavaWikiBotFramework (JWBF) is one of these libraries written in Java. The JWBF provides access to many functions defined in the MediWiki API. We can request lists of articles (category, backlinks ...) or other lists in Iterator / Iterable form, with the additional feature that only a limited number of items will be downloaded instantly. JWBF will autonomously retrieve more items if they are needed. This helps to reduce traffic, especially in big wikis like Wikipedia. Besides, JWBF works since MediaWiki 1.9.*. Some simple JWBF methods are explained below.

Login:

void login(java.lang.String username, java.lang.String passwd) //Performs a Login

Read Content:

Article readContent(java.lang.String name) // Reads a page’s content

Read Data:

SimpleArticle readData(java.lang.String name)
A sample code to perform a login operation and read a page:

```java
import net.sourceforge.jwbf.bots.MediaWikiBot;
import net.sourceforge.jwbf.contentRep.mw.SimpleArticle;

public class Basic {   public static void main(String[] args) throws Exception
         b.login(“Zephyr”, “*********”);      SimpleArticle sa = new
         SimpleArticle(b.readContent(“Main Page”));
         System.out.println(sa.getText());
    }
}
```

MediaWikiBot is a class to perform simple operations such as login and reading the content of pages. Sample code logsins to MediaWiki and reads the content of the main page.

### 5.5.2 Google Maps API

Google created the Google Maps API to allow developers to integrate Google Maps into their websites with their own data points. It is a free service, and currently does not contain ads, but Google states in their terms of use that they reserve the right to display ads in the future.

By using the Google Maps API, it is possible to embed the full Google Maps site into an external website. Developers are required to request an API key, which is bound to the website and directory entered when creating the key. The Google Maps API key is no longer required for API version 3. Creating a customized map interface requires adding the Google JavaScript code to a page, and then using Javascript functions to add points to the map.

When the API first launched, it lacked the ability to geocode addresses, requiring users to manually add points in (latitude, longitude) format. This feature has since been added for premier.
Geocoding

Geocoding is the process of converting addresses (like "1600 Amphitheatre Parkway, Mountain View, CA") into geographic coordinates (like latitude 37.423021 and longitude -122.083739), which you can use to place markers or position the map. The Google Maps API Geocoding Service provides a direct way to access a geocoder via an HTTP request or by using a GClientGeocoder object.

The Geocoding Object

The Google Maps API geocoding service can be accessed via the GClientGeocoder object. GClientGeocoder.getLatLng() is used to convert a string address into a GLatLng. This method takes as parameters a string address to convert, and a callback function to execute upon retrieval of the address. The callback function is necessary since geocoding involves sending a request to Google's servers and can take some time.

In this example, an address is geocoded, a marker is added at that point, and an info window is opened displaying the address.

```javascript
var map = new GMap2(document.getElementById("map_canvas"));
var geocoder = new GClientGeocoder();

function showAddress(address) {
  geocoder.getLatLng(
    address,
    function(point) {
      if (!point) {
        alert(address + " not found");
      } else {
        map.setCenter(point, 13);
        var marker = new GMarker(point);
        map.addOverlay(marker);
        marker.openInfoWindowHtml(address);
      }
    }
 );
}
```

The Maps API geocoder can also be modified if results within a given viewport (expressed as a bounding box of type GLatLngBounds) through the GClientGeocoder.setViewport() method are preferred. Results tailored to a particular domain (country) using the GClientGeocoder.setBaseCountryCode() method can be returned. Geocoding requests can be
sent for every domain in which the main Google Maps application offers geocoding. For example, searches for "Toledo" will return different results within the domain of Spain (http://maps.google.es) specified by a country code of "es" than within the default domain within the United States.

**Extracting Structured Addresses**

If one would like to access structured information about an address, `GClientGeocoder` also provides a `getLocations()` method that returns a JSON object consisting of the following information:

- **Status**
  - request -- The request type. In this case, it is always geocode.
  - code -- A response code indicating whether the geocode request was successful or not.

- **Placemark** -- Multiple placemarks may be returned if the geocoder finds multiple matches.
  - address -- A nicely formatted and properly capitalized version of the address.
  - AddressDetails -- The address formatted as xAL, or eXtensible Address Language, an international standard for address formatting.
    - Accuracy -- An attribute indicating how accurately we were able to geocode the given address.
  - Point -- A point in 3D space.
    - coordinates -- The longitude, latitude, and altitude of the address. In this case, the altitude is always set to 0.
Here we show the JSON object returned by the geocoder for the address of Google's headquarters:

```json
{
  "name": "1600 Amphitheatre Parkway, Mountain View, CA, USA",
  "Status": {
    "code": 200,
    "request": "geocode"
  },
  "Placemark": [
    {
      "address": "1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA",
      "AddressDetails": {
        "Country": {
          "CountryNameCode": "US",
          "AdministrativeArea": {
            "AdministrativeAreaName": "CA",
            "SubAdministrativeArea": {
              "SubAdministrativeAreaName": "Santa Clara",
              "Locality": {
                "LocalityName": "Mountain View",
                "Thoroughfare": {
                  "ThoroughfareName": "1600 Amphitheatre Pkwy"
                },
                "PostalCode": {
                  "PostalCodeNumber": "94043"
                }
              }
            }
          }
        },
        "Accuracy": 8
      },
      "Point": {
        "coordinates": [-122.083739, 37.423021, 0]
      }
    }
  ]
}
```
In this example, *getLocations()* method is used to geocode addresses and extract the nicely formatted version of the address and the two-letter country from the JSON. The it will be displayed in the info window.

```javascript
var map;
var geocoder;

function addAddressToMap(response) {
  map.clearOverlays();
  if (!response || response.Status.code != 200) {
    alert("" + address + " not found");
  } else {
    place = response.Placemark[0];
    point = new GLatLng(place.Point.coordinates[1],
                     place.Point.coordinates[0]);
    marker = new GMarker(point);
    map.addOverlay(marker);
    marker.openInfoWindowHtml(place.address + '<br>' +
                            '<b>Country code:</b> ' + place.AddressDetails.Country.CountryNameCode);
  }
}
```

Reverse Geocoding (Address Lookup)

As explained above, the term geocoding generally refers to translating a human-readable address into a location on a map. The process of doing the converse, translating a location on the map into a human-readable address, is known as reverse geocoding.[21] This is what we want to achieve in our product. After extracting GPS information (i.e. latitude, longitude and altitude), we will determine the name of the location by means of the Google Maps API.

The *GClientGeocoder.getLocations()* method supports both standard and reverse geocoding. If a *GLatLng* object is passed to this method rather than a String address, the geocoder will perform a reverse lookup and return a structured JSON object of the closest addressable location. Note that the closest addressable location may be some distance from the original latitude and longitude values of the query, if the supplied *GLatLng* is not an exact match for any addressable locations.

➤ The geocoder will attempt to find the closest addressable location within a certain tolerance; if no match is found, the geocoder will return a G_GEO_UNKNOWN_ADDRESS (602) status code.
An example to reverse geocoding is given in the following code segment.

```javascript
var map;
var geocoder;
var address;

function initialize() {
    map = new GMap2(document.getElementById("map_canvas"));
    map.setCenter(new GLatLng(40.730885,-73.997383), 15);
    map.addControl(new GLargeMapControl);
    GEvent.addListener(map, "click", getAddress);
    geocoder = new GClientGeocoder();
}

function getAddress(overlay, latlng) {
    if (latlng != null) {
        address = latlng;
        geocoder.getLocations(latlng, showAddress);
    }
}

function showAddress(response) {
    map.clearOverlays();
    if (!response || response.Status.code != 200) {
        alert("Status Code:" + response.Status.code);
    } else {
        place = response.Placemark[0];
        point = new GLatLng(place.Point.coordinates[1],place.Point.coordinates[0]);
        marker = new GMarker(point);
        map.addOverlay(marker);
        marker.openInfoWindowHtml(
            '<b>orig latlng:</b>' + response.name + '<br/>' +
            '<b>latlng:</b>' + place.Point.coordinates[1] +"," + place.Point.coordinates[0] + '<br/>' +
            '<b>Status Code:</b>' + response.Status.code + '<br/>' +
            '<b>Status Request:</b>' + response.Status.request + '<br/>' +
            '<b>Address:</b>' + place.address + '<br/>' +
            '<b>Accuracy:</b>' + place.AddressDetails.Accuracy + '<br/>' +
            '<b>Country code:</b>' + place.AddressDetails.Country.CountryNameCode);
    }
}
```
6. Implementation Details

6.1 Object Matching

One of the most critical issues in Conqi is the “Object Matching” part. The field research results drew our team to the conclusion that the SURF (Speed Up Robust Features) method is suitable for our application because it is a new method which is presented in 2006 and the standard version of SURF is several times faster than SIFT and claimed by its authors to be more robust against different image transformations than SIFT which is its counterpart.

Actually, the database of Conqi will include the images, their location information (GPS and compass) and their interest points which are extracted as their features. When the user provides an image to our system, first, the interest points of that image are obtained. Immediately after, the correspondence between the user image and the set of images in the database coinciding with the location information of it, will be measured using their interest points. The database image having highest number of matches with the user image will be accepted as the result image.

The following parts will briefly explain SURF method.\[22,23\]

6.1.1 Detection of Interest Points

Basically, the Hessian matrix is used in SURF feature detector. Given a point $x=(x,y)^T$ in an image, the Hessian matrix in $x$ at scale $\sigma$ is defined as;

$$
\mathcal{H}(x, \sigma) = \begin{bmatrix}
L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\
L_{yx}(x, \sigma) & L_{yy}(x, \sigma)
\end{bmatrix}
$$

(1)

where $L_{xx}(x, \sigma)$ is the convolution of the Gaussian second order derivative $\partial^2/\partial x^2 g(\sigma)$ with the image $I$ in point $x$, and similarly for $L_{xy}(x, \sigma)$ and $L_{yy}(x, \sigma)$. SURF approximates second order Gaussian derivatives with box filters (Figure 6.1).
Image convolutions with these box filters can be computed rapidly by using integral images as defined in \cite{24}. The entry of an integral image $I_x(x)$ at location $x = (x, y)^T$ represents the sum of all pixels in the base image $I$ of a rectangular region formed by the origin and $x$.

\[
I_x(x) = \sum_{i=0}^{x} \sum_{j=0}^{y} I(i, j)
\]  

(2)

Once we have computed the integral image, it is straightforward to calculate the sum of the intensities of pixels over any upright, rectangular area.

The location and scale of interest points are selected by relying on the determinant of the Hessian. Hereby, the approximation of the second order derivatives is denoted as $D_{xx}$, $D_{yy}$, and $D_{xy}$. By choosing the weights for the box filters adequately, we find as approximation for the Hessian’s determinant

\[
\text{det}(H_{\text{approx}}) = D_{xx} D_{yy} - (0.9 D_{xy})^2.
\]  

(3)

For more details, see \cite{22}. Interest points are localized in scale and image space by applying a non-maximum suppression in a $3 \times 3 \times 3$ neighborhood. Finally, the found maxima of the determinant of the approximated Hessian matrix are interpolated in scale and image space.
6.1.2 Interest Point Descriptor

In a first step, SURF constructs a circular region around the detected interest points in order to assign a unique orientation to the former and thus gain invariance to image rotations. The orientation is computed using Haar wavelet responses in both x and y direction as shown in the middle of Figure 6.2. The Haar wavelets can be easily computed via integral images, similar to the Gaussian second order approximated box filters. Once the Haar wavelet responses are computed, they are weighted with a Gaussian with $\sigma = 2.5s$ centered at the interest points. In a next step the dominant orientation is estimated by summing the horizontal and vertical wavelet responses within a rotating wedge, covering an angle of $\pi/3$ in the wavelet response space. The resulting maximum is then chosen to describe the orientation of the interest point descriptor.

![Figure 6.2](image)

*Figure 6.2* Left: Detected interest points for a Sunflower field. This kind of scenes show clearly the nature of the features obtained from Hessian-based detectors. Middle: Haar wavelet filters used with SURF. Right: Detail of the Graffiti scene showing the size of the descriptor window at different scales.

In a second step, the SURF descriptors are constructed by extracting square regions around the interest points. These are oriented in the directions assigned in the previous step. The windows are split up in $4 \times 4$ sub-regions in order to retain some spatial information. In each sub-region, Haar wavelets are extracted at regularly spaced sample points. In order to increase robustness to geometric deformations and localization errors, the responses of the Haar wavelets are weighted with a Gaussian, centered at the interest point. Finally, the wavelet responses in horizontal $d_x$ and vertical directions $d_y$ are summed up over each sub-region. Furthermore, the absolute values $|d_x|$ and $|d_y|$ are summed in order to obtain information about the polarity of the image intensity changes. Hence, the underlying intensity pattern of each sub-region is described by a vector:
\[ v = (\Sigma d_x, \Sigma d_y, \Sigma |d_x|, \Sigma |d_y|). \] (4)

The resulting descriptor vector for all \(4 \times 4\) sub-regions is of length 64. Notice that the Haar wavelets are invariant to illumination bias and additional invariance to contrast is achieved by normalizing the descriptor vector to unit length.

### 6.1.3 Object Recognition

Traditional object recognition methods rely on model images, each representing a single object in isolation. In practice, however, the necessary segmentation is not always affordable or even possible. For our object recognition application, we use images where the objects are not separated from the background. Thus, the background also provides features for the matching task. In any given test image, only one object or object group that belongs together is assumed. Hence, object recognition is achieved by image matching. Extracted interest points of the input image are compared to the interest points of database images. In order to create a set of interest point correspondences \(M\), we will use the nearest neighbor ratio matching strategy [4, 5, 6]. This states that a matching pair is detected if its Euclidean distance in descriptor space is closer than 0.8 times the distance to the second nearest neighbor.

The selected object is the one figuring in the database image with the highest recognition score \(SR\). This score is traditionally the number of total matches in \(M\). However, the presence of mismatches often lead to false detections. This can be avoided with the help of the following new alternative for the estimation of the recognition score. We will calculate the mean Euclidean distance to the individual nearest neighbors for each image pair. This value is typically smaller for corresponding image pairs than for non-corresponding ones, and it does not depend on the number of extracted features in the individual images. Hence, we will maximize the following recognition score

\[
S_{Ri} = \arg\max_i \left( \frac{N_i}{\sqrt{\sum_{j=1}^{N_i} d_{ij}^2}} \right)
\]

and chose the object for which the mean distance of its matches is smallest. \(N_i\) denotes the number of matches in image \(i\). Furthermore, \(d_{ij}\) is the Euclidean distance in the descriptor space.
between a matching pair of key points. The matching criteria is that this distance is closer than 0.8 times the distance to the second nearest neighbor.

### 6.1.4 OpenCV – SURF

To speed up implementation process of our application, we will use the SURF class implemented in OpenCV 2.0.\(^{[28]}\)

SURF is a class for extracting Speed Up Robust Features from an image.

```cpp
class SURF : public CvSURFParams
{
    public:
        // default constructor
        SURF();
        // constructor that initializes all the algorithm parameters
        SURF(double _hessianThreshold, int _nOctaves=4,
            int _nOctaveLayers=2, bool _extended=false);
        // returns the number of elements in each descriptor (64 or 128)
        int descriptorSize() const;
        // detects keypoints using fast multi-scale Hessian detector
        void operator()(const Mat& img, const Mat& mask,
                        vector<KeyPoint>& keypoints) const;
        // detects keypoints and computes the SURF descriptors for them
        void operator()(const Mat& img, const Mat& mask,
                        vector<KeyPoint>& keypoints,
                        vector<float>& descriptors,
                        bool useProvidedKeypoints=false) const;
};
```

The class SURF implements Speed Up Robust Features descriptor. There is fast multi-scale Hessian key point detector that can be used to find the key points (which is the default option), but the descriptors can be also computed for the user-specified key points. The function can be used for object tracking and localization, image stitching etc.
7. Design and Implementation Constraints

The limitations of designing an application using mobile device can be listed as:

- Limited processing speed
- Very limited memory for application memory and heap memory
- Small screen size
- Cost of mobile/smart phone
- Issues in Data Security.
- The limit of the availability of wireless hot-spots within the field

Additionally, the system must provide a capacity for parallel operation. The end system should also allow for seamless recovery, without data loss, from individual device failure. If the system is down, then users must not notice, or notice that the system recovers quickly (expected to be in seconds).

7.1 Memory Constraints

7.1.1 Server

7.1.1.1 RAM

J2EE Server is responsible for meeting the requirements caused by image processing computations. Since a limited number of clients are allowed to be connected to the server at the same time (up to 50 clients), the memory that will be spent for their requests will be again limited in parallel to that. Taking that into consideration, a minimum of 15 GB memory will serve the purpose for server side.

7.1.1.2 Primary Storage

Actually, J2EE Server will present memory up to 1TB and it will meet our needs.
7.1.2 Client

7.1.2.1 RAM
The memory usage of Conqi will obviously be constrained by the device it is intended to run on. Client side will be an ordinary mobile device with GPS and compass support and such mobile devices have memory amounts approximately 128 MB which will fulfill our needs. We intend to use Google G1 and G2 in order to test our application.

7.1.2.2 Primary Storage
Google G1 and G2 will present memory from 3 to 8 GB which will be enough to meet our needs.
8. Project Schedule

8.1 Gantt Chart

Detailed Design Report for Conqi, the Mobile Conqueror
Appendix A, the Glossary

<table>
<thead>
<tr>
<th>DB</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>OS</td>
<td>Operating Systems</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>Admin</td>
<td>Administrator</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>

Appendix B, References

[1] Location Class Definition
http://developer.android.com/reference/android/location/Location.html

[2] LocationManager Class Definition
http://developer.android.com/reference/android/location/LocationManager.html

[3] Sensor Class Definition

[4] SensorManager Class Definition

[5] SensorEvent Class Definition


[7] SensorEventListener Class Definition

[8] Intent Class Definition
http://developer.android.com/reference/android/content/Intent.html

[9] Web Services Description Language (WSDL) 1.1
http://www.w3.org/TR/wsd1


[12] Eclipse
http://www.eclipse.org

[13] JDBC
http://java.sun.com/javase/technologies/database

http://en.wikipedia.org/wiki/Apache_Axis

http://en.wikipedia.org/wiki/Apache_Axis2

[16] Android Emulator

[17] DroidDraw
http://www.droiddraw.org

[18] MediaWiki API
http://www.mediawiki.org

[19] Google Maps API
http://code.google.com/apis/maps/

[20] Geocoding
http://code.google.com/apis/maps/documentation/services.html#Geocoding_Object

[21] Reverse Geocoding
http://code.google.com/apis/maps/documentation/services.html#ReverseGeocoding

ECCV. (2006)

Recognition of Museum Objects. In:
    In: Computer Vision and Pattern Recognition. (2001)
[26] Lowe, D.G.: Distinctive image features from scale-invariant keypoints, cascade filtering  
[28] OpenCV 2.0 C++ Reference, Image Processing and Computer Vision  
    http://opencv.willowgarage.com/documentation/cpp/feature_detection.html