

RED RIVER FLOOD MONITOR iOS APPLICATION

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By
Saumya Singh

In Partial Fulfillment
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

May 2013

Fargo, North Dakota

North Dakota State University
Graduate School

Title

Red River Flood Monitor iOS Application

By

Saumya Singh

The Supervisory Committee certifies that this *disquisition* complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Anne Denton

Chair

Dr. Kendall E. Nygard

Dr. Wei Jin

Dr. Jagdish Singh

Approved:

07/02/2013

Date

Dr. Brain M. Slator

Department Chair

ABSTRACT

The Flood Monitor system is designed to help people share photos and information in case of a flood in the basin of the Red River. This work is part of the redesign of an application that has the goal of enabling mobile and web-based interactions. My contribution to this project is to develop the iOS interface of the flood monitor system.

The main components of the iOS application consist of an interface to view existing data and another interface to upload new markers. The information exchange between the server and application happens via XML communication. The paper discusses features of application, problems encountered while developing and their solutions. It also presents the user interface design and software design/development procedures.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my advisor, Dr. Anne Denton. I appreciate that she gave me the valuable opportunity to work with her on the Red River Flood Monitor Project. She always encouraged and helped me to do my best. I really thank her for her support and advice from my heart. Secondly, I would like to express my sincere gratitude to my committee members Dr. Kendall Nygard, Dr. Wei Jin and Dr. Jagdish Singh (Chair and professor at Department of Pharmaceutical Sciences at North Dakota State University) for their guidance. Thirdly, I would like to thank my team members Cesar Ramirez and Justin Anderson for helping me out with my paper.

Huge thanks for my sister Neha and my family for their constant support and encouragement throughout my graduate studies.

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGEMENTS	iv
LIST OF TABLES	vii
LIST OF FIGURES	viii
CHAPTER 1. INTRODUCTION	1
1.1. Overview	1
1.2. iOS Application	3
1.3. Problem Statement and Requirement Analysis	5
1.4. Application.....	7
1.4.1. Overview.....	7
1.4.2. Logic	10
1.5. Organization of the Paper	12
CHAPTER 2. BACKGROUND.....	13
2.1. Overview	13
2.2. Existing and Ongoing Work	15
CHAPTER 3. APPLICATION DESIGN	17
3.1. Overview	17
3.2. Challenges in Mobile UI Design.....	17
3.3. Wireframes.....	20
3.4. UML 2 Use Case Diagrams	25
CHAPTER 4. IMPLEMENTATION/DEVELOPMENT	28

4.1. Development Overview	28
4.2. XML Communication	28
4.3. Data (Response) Parsing	38
CHAPTER 5. APP PERFORMANCE ANALYSIS.....	40
5.1. Loading Time.....	40
5.2. Testing.....	41
5.2.1. Unit Testing	41
5.2.2. Integration Testing	42
CHAPTER 6. CONCLUSION	43
6.1. Conclusion	43
6.2. Future Work.....	44
REFERENCES.....	46

LIST OF TABLES

<u>Table</u>	<u>Page</u>
1. Main problem areas in mobile designing [11]	19

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1. Working of a web application.....	2
2. iOS Storyboard.....	9
3. XML communication.....	10
4. Example XML commands.....	11
5. One region and two boundaries.....	13
6. Bounds from map points.....	14
7. General architecture of Flood Monitor project.....	15
8. Wireframe 1: Screen showing a tab-based structure.....	20
9. Wireframe 2: Initial view and generated response to user query.....	21
10. Wireframe 3: Screen showing markers with different severity levels.....	22
11. Wireframe 4: Marker detail view.....	23
12. Wireframe 5: Marker submission action initiated by the user.....	24
13. Wireframe 6: Marker submission form.....	25
14. Use Case Diagram 1: Interaction between user and server.....	26
15. Use Case Diagram 2: Detail diagram view markers action.....	26
16. Use Case Diagram 3: Detail diagram generating and receiving XML response.....	27
17. GetRegions XML format.....	32
18. GetEvents XML format.....	33
19. UITableView showing events for Fargo boundary.....	33
20. GetMarker XML format.....	34
21. KML file format.....	36

22. SubmitMarker XML format.....	37
23. Loading time comparison	41

CHAPTER 1. INTRODUCTION

1.1. Overview

The World Wide Web (www), as we know it today, serves myriads of global users via an array of different web applications. What makes the implementation of these applications quick and easy is the highly programmable nature of the www. The modern website consists of two modules: flexible web browsers and web applications. [1]

A web browser is a software application allowing users to access data located on web servers through websites and interact with it. Today's web is not just a collection of static pages but of documents generated on the go as a response to client's request. Web applications present a person surfing website with dynamically generated webpages via interaction and querying of the content server/content repository database. The documents generated have a standard format (language) that is supported by all browsers (e.g., Hyper Text Markup Language (HTML) or Extensible Hyper Text Markup Language (XHTML)). The task of the web browser is to interpret and run all the scripts along with displaying the requested pages and content. The figure 1 below shows the three-layered web application model. The first layer is the one presented to the user, typically a web browser or the user interface; the second layer is responsible for generating the content dynamically using technology such as Java servlets (JSP) or Active Server Pages (ASP), and the third layer is that of the database [1].

The native application for mobile platforms, in this case for iPhone, follow the same principle but website functionality (webserver communication) is implemented in the code. This

lets the developer choose any means of displaying the front-end to the user, not necessarily a web browser. The data entered by the user is collected in the same fashion as it would have been collected when the front-end would is a website. The rest is the same as shown in the below figure.

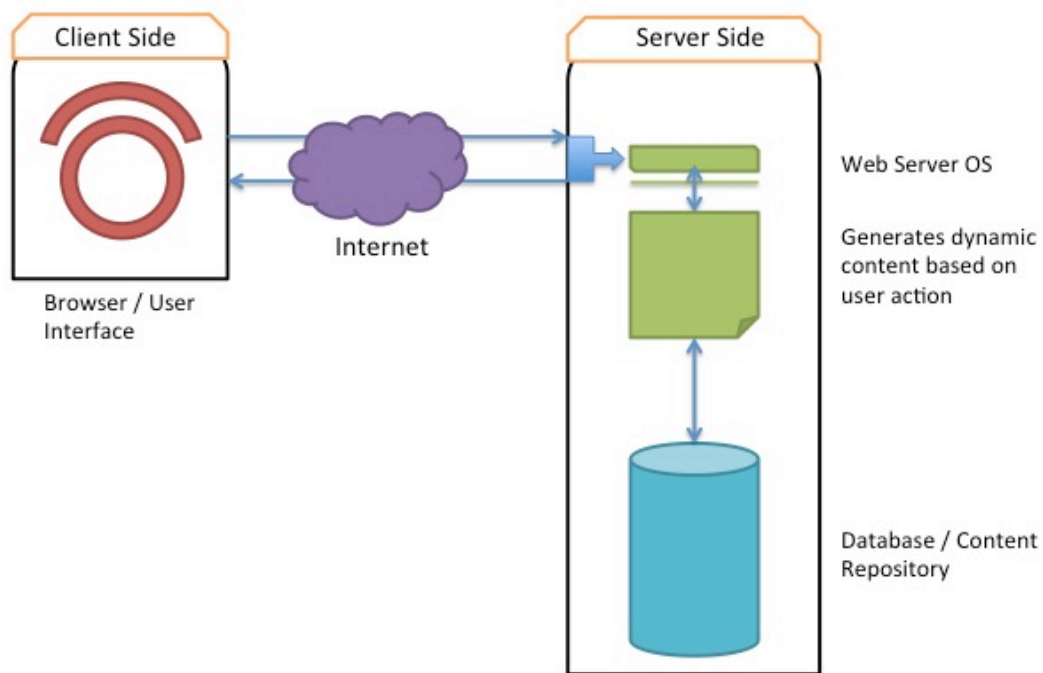


Figure 1. Working of a web application

This project helps people share information, regarding floods and spread awareness benefitting others. The project aims at providing an interface for an easy, hassle-free and efficient way to share this information. The project has been implemented for three platforms:

- Web – User Interface/front end is a website. The user interacts with the system via browser.

- Android – The user here is using an Android phone and has installed our application as a native application. This implementation utilizes the native aspects of Android OS to render the front-end to the user and interacts with the webserver programmatically.
- iOS – The user is on an iPhone, and has natively installed the application from the app store. The front-end is comprised of the native elements of iOS and interaction with the webserver is done programmatically.

This paper consists of an explanation of the functionality of the implementation for iOS. The project under consideration involves building a native application for the Red River Flood Monitoring System. It is intended for use by iPhone/iPad users to see and share flood information.

1.2. iOS Application

iOS is a mobile operating system developed and distributed by Apple Inc. for use with its products (iPhone/iPad/iPod). The user interface in iOS uses multi-touch gestures to implement the concept of direct manipulation. The direct manipulation is a human computer interaction concept allowing the users to directly manipulate the objects in view via gestures/actions that would correspond, at least loosely, to how they manipulate things in real world. Multi-touch gestures are standardized motions used to interact with multi touch devices [2]. In iOS, there are four abstraction layers:

1. **Core OS layer** – This layer implements the low-level features of the iOS. The user does not normally interact with this layer but other frameworks may use it. This may be used directly when interacting with other external devices (eg. Bluetooth).

2. **Core Services layer** – This layer implements the most fundamental system services used by all the iOS applications. If not used directly then other frameworks (utilized in the application) use them.
3. **Media layer** – This layer implements graphics, audio, and video technologies that are used to create best multimedia experience on a mobile device.
4. **Cocoa Touch layer** – This layer implements the basic application infrastructure and support for key technologies such as multitasking, touch-based input, push notifications, and many high-level system services.

The iOS SDK delivers the means to create a native iOS app. It is comprised of tools and interfaces essential for developing, installing, running and testing native applications [3]. To develop iOS apps, one uses Xcode, Apple's integrated development environment (IDE). Xcode provides all the tools needed to design an app's user interface and write the code that implements it. In earlier versions of iOS (and < 5), for making a functional or even an accurate looking prototype, some coding was required. This is because Apple's engineers adhere very closely to the Model-View-Controller (MVC) design paradigm. UIViews that represent a view (having a physical representation on screen) really just provide visual information. In iOS (and < 5) to navigate across various views one had to write a controller [4]. But with the introduction of iOS 5 (and > 5), the need for coding, to even accomplish little tasks, has been eliminated to some extent. In particular, the code for navigation is auto generated and the developers just have to make connections in the Storyboards.

The Xcode interface is a seamless integration of a code editor, user interface (UI) design with Storyboards, testing, and debugging, all within a single platform. The embedded Apple LLVM compiler underlines coding mistakes as we type, and fixes the problems automatically [5]. **Objective-C** is a general-purpose, high-level, object-oriented programming language that adds Smalltalk-style messaging to the C programming language. It is the main programming language used by Apple for all its operating systems (the OS X and iOS). The iOS SDK was used in the Xcode IDE, to create this native application. The language used is Objective C.

1.3. Problem Statement and Requirement Analysis

With the increase in hand-held media, addressing Internet users is not limited to web front ends (browser dependent). When considering a project that involves the web audience (interacting with the system using a browser), it is necessary to think about other media through which the data could be accessible. The most important consideration in developing a user interface is that how beneficial it could be for the end user. iOS is one of the major mobile operating system used by a wide variety of people throughout the world. Developing a native app for the Red River Flood Monitor system had the intention of allowing users to interact with the system easily on their iOS devices. Particularly, in our case it facilitate uploading markers on the go using mobile devices and not waiting until users get a chance to use their desktop. The inclusion of mobile interface in the project serves mainly three purposes:

1. The wait time between event observed and data uploaded is minimized. The user has the option of uploading the data from point of observation. As stated earlier the user does not have to wait until he gets a chance to use his/her desktop.

2. Developing an array of interfaces that allow users to interact with would address a larger number of audiences, thereby increasing the user base of the project.
3. The above two factors help increase accuracy of the data displayed within the application. Since, the user can now upload the data from his mobile (in this case an iOS device), the probability that the uploaded data is more accurate, compared to the scenario where the user waits to reach his desktop, increases. Elaborating on the previous statement, all mobile devices have an inbuilt GPS location tracker system, which captures the user's exact location; this is the default location the application uses for the uploaded data, unless otherwise stated by the user. Desktop don't have this feature, so in case the user uploads a marker from web interface without specifying his/her location, the location grabbed by the system may not be the exact one. Also, since the application would be available to a larger audience the amount of marker uploads would also be high. The larger the amount of data, the more accurate is the reflection of flood situation.

The main issues considered were user wait time and interoperability of heterogeneous platforms. Caching was introduced to address the issue of user wait time. When the user requests flood data for a region more than once, he/she is presented with a cached version, reducing his wait time considerably. The cache is updated regularly so that the user sees up-to-date information. XML communication solves the latter problem. Since communication with the server is central to the application, XML is used as a single unified data representation that is transferred from the server to the clients, irrespective of their platform. All the mobile devices have the capability to parse XML easily.

The objective for the user interface was to present to the user a map with region overlays for which the flood data is available. Each overlay represents a collection of structures (regions, events, markers). The user is then free to select the region for which he wants to view the data. The information exchange happens through a KML file. This file holds all the relevant data that is to be presented to a user. The file gets its data from a centralized database and is updated in regular intervals. Any user interface (view) that presents the flood data interacts with its controller that reads the data from the file and presents it to the view in a humanly readable form. The iOS frontend shows the points for which the data is available as an annotated point. When the user taps on the marker, information related to that marker is presented to him or her. A PHP script updates the database in case of a new submission. The key is the XML Communication between the user's device (in this case an iOS device) and the server. The data returned by the server for various queries is an XML response, which is parsed at the user's end and displayed in a readable format.

1.4. Application

1.4.1. Overview

In [6], a three-component archetype is presented for mobile applications. It includes cloud computing, a RESTful Web Service and the smart phone. This allows extensive information be presented to the user within the context of his/her activity. In our Flood Monitor Application, we exploit a similar scenario, where the database is stored remotely and the handheld smart phone device interacts with it through an XML interface, representing the RESTful web service of the above case. The advantages of using RESTful web services are that

it is easy to invoke, produces a discretely formatted response and is easily parsed using event-driven XML parsing. Using this archetype in the application makes sense as both J2ME and Cocoa Touch are C language based derivatives having a large developer community; provide for a high-level approach to TCP/IP connectivity, streaming XML Parser with a rich framework for user interface development [6].

Technical specification of the iOS app: The native app in consideration is targeted for Apple's handheld devices (iPhone/iPad/iPod). The base SDK targeted was iOS 4 (the devices that implement iOS 4 or versions above are able to run this program). As is typical with all iOS applications, the delivery method is via the iTunes App Store. The application currently supports portrait orientation only. The application was built using Xcode 4 IDE. The graphics used in the application were built using Adobe Photoshop CS5. The application requires Internet connectivity to function properly. All iPhone applications are developed using Objective C as their primary language. Objective C is a variant of ANSI C with OO extensions based on Smalltalk syntax. It is an easy to learn language as some problematic C constructs (arrays, string and pointers) replaced by fully object-oriented types NSArray and NSString. Pointers are only required to declare objects and are never explicitly dereferenced. The parameter call is different in the parameters being labeled and object messaging uses square brackets rather than dots. [4]

In Objective C classes are divided in interface and implementation, .h and .m files respectively. The variables are declared inside curly braces in the interface section, followed by constructor, methods and properties definition. The compiler directive @interface and @end mark the start and end of interface file and similarly @implementation and @end contain the

implementation file. The user interface of the application is implemented in Xcode's Storyboard. Memory management is manual, and memory leaks have been tested using Instruments v4.5. The current app does not utilize Automatic Reference Counting (ARC).

Figure 2 below shows a screen capture of the application's storyboard from Xcode. Arrows depict the flow of the navigation across the UINavigationController. As explained earlier, UINavigationController act as the controller in the MVC design paradigm. The implementation of code to make the application function is written in the UINavigationController. The UINavigationController contain the UIView, as is visible in the figure. UIViews are the collection of all the user interface elements (like the buttons, textfields, labels etc.).

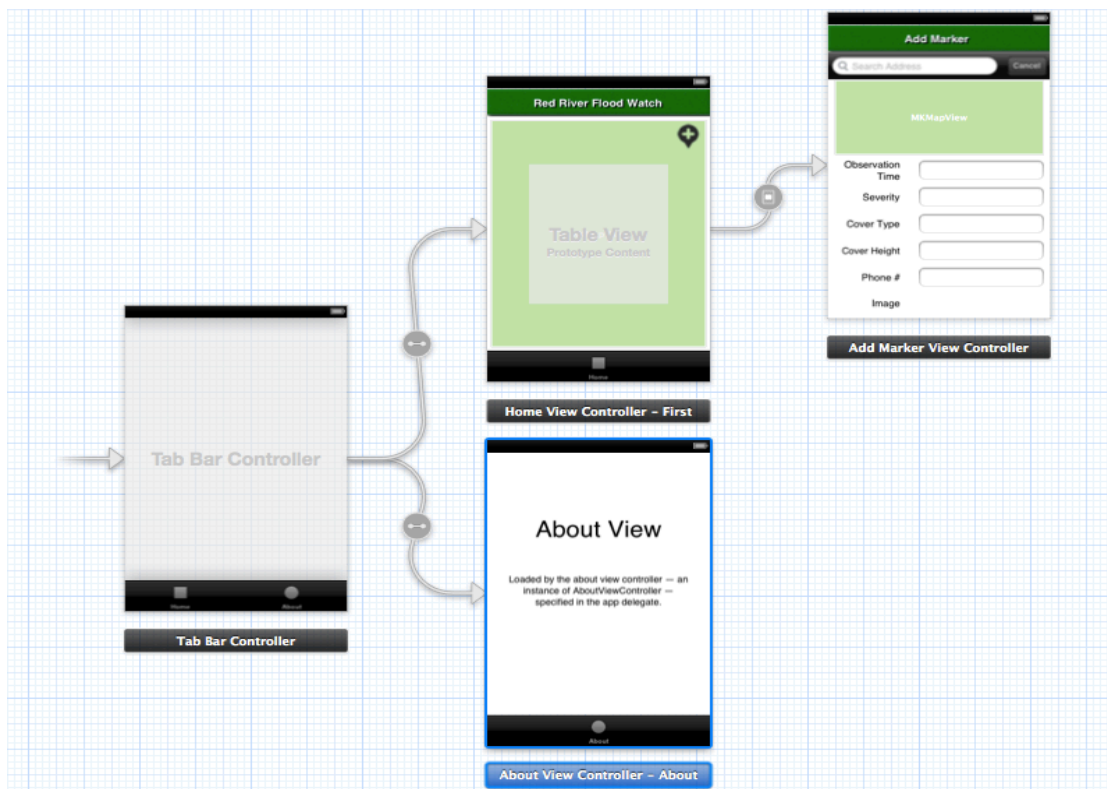


Figure 2. iOS Storyboard

A third party library ASIHTTPRequest has been used to send HTTP post request to the server. The data returned by the server is XML Response, which is parsed using built-in NSXMLParser.

1.4.2. Logic

The app's core functionality and responses to user interaction depend heavily on user selection and XML communication with the server. All requests of the device are posted over HTTP Post to the following URL:

<http://flood.cs.ndsu.nodak.edu/index.php>

Figure 3, shows that all the interaction between the server and the device is done via the XML Communication. The mobile device sends in a HTTP Post request in XML to which the server sends a valid response, also in XML.

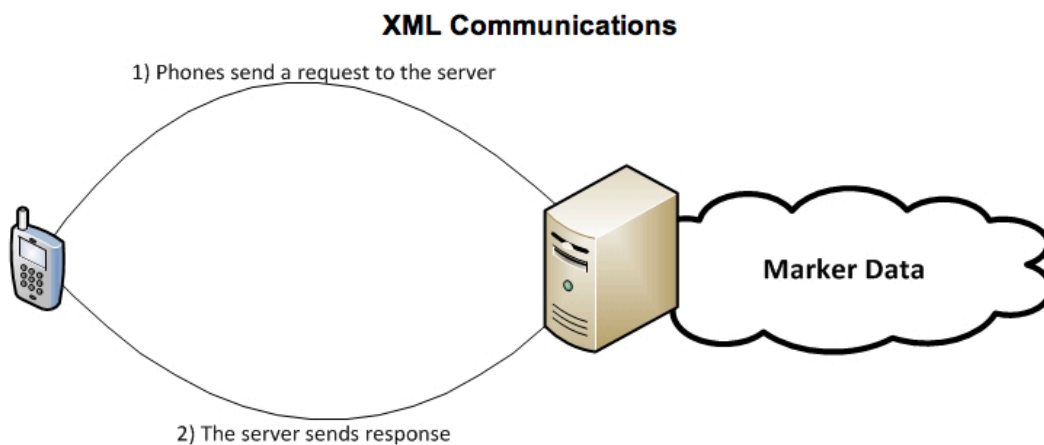


Figure 3. XML communication

There are several sets of phone XML request commands and server XML response commands. It is essential to have a method to quickly parse this data into user readable format. The built-in NSXMLParser Class in iOS SDK accomplishes this. The XML documents along with DTD declarations are parsed in an event-driven manner by the NSXMLParser Class instance. The delegate is notified about the encountered items (elements, attributes, CDATA blocks, etc.) during the processing of an XML document. It only reports the elements (does not modify them) and also reports any parsing error that may have occurred. [7]

The figure 4 gives example formats of an XML request send by phone over HTTP. The second following image is a format for the server's response to this request.

GetEventsByBoundaryID

```
<phone>
  <command>GetEventsByBoundaryID</command>
  <params>
    <BoundaryID>##</BoundaryID>
  </params>
</phone>
```

This will get a list of events that share a single boundary.

GetEventsByBoundaryID_Response

```
<server>
  <command>GetEventsByBoundaryID</command>
  <response>
    <events>
      <event>
        <id>##</id>
        <name>$$$</name>
        <begindate>YY-MM-DD</begindate>
        <enddate>YY-MM-DD</enddate>
        <markercount>##</markercount>
        <active>##</active>
      </event>
      ...
    </events>
  </response>
</server>
```

This will return a list of events that share a given boundary.

Figure 4. Example XML commands

1.5. Organization of the Paper

The remainder of the paper is as follows: Chapter 2 provides a literature review of the existing work and also the other platforms on which the project has been implemented. Chapter 3 explains details of application design by documenting various diagrams. Chapter 4 gives an overview of implementation of the developed application and its functionalities. Chapter 5 shows results of various tests performed to analyze app's performance on a user's device. Chapter 6 summarizes the work done, and documents conclusions, limitations of the current system and suggestions for future work.

CHAPTER 2. BACKGROUND

2.1. Overview

In many years the Red River floods and each flooding event should be uniquely identified in the system. One way of partitioning the data in small chunks and also keeping it unique is to divide it based on time interval. That is why each unique event has its own start date and end date. Hence, we can say an event is an instance of flooding dictated by a start date, an end date, a region and a type.

A region in this system could be thought of as a collection of one or more boundaries. A boundary is a geographical area having a rectangular shape. The concept of boundary is used to increase the granularity of the data. The figure 5 below shows North Dakota region and the highlighted rectangular areas show the two boundaries of Fargo and Minot.



Figure 5. One region and two boundaries

Each event is constrained by a boundary. In case a marker does not belong to any of the pre-defined boundaries, it belongs to “un-clustered”.

As mentioned earlier, a boundary depicts a geographical rectangle. It is defined by two points on a map, the north-west and south-east coordinates. Each map point is an ordered pair of latitude and longitude value. For example, map point = (latitude value, longitude value). Given these two points, we can construct a boundary as shown in the following figure 6:

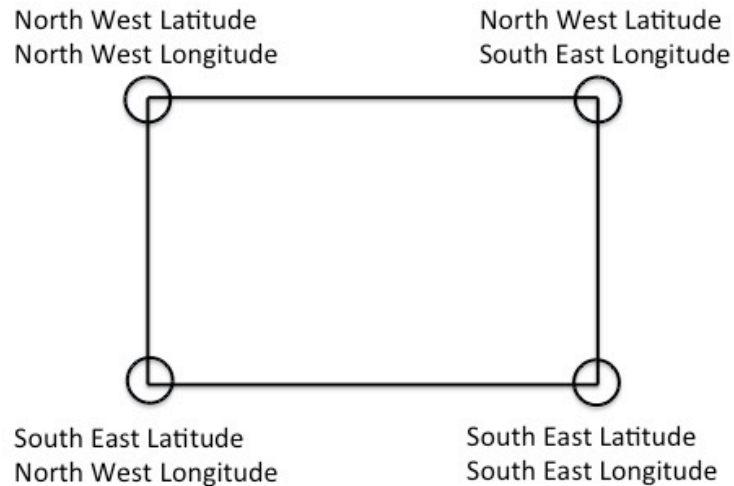


Figure 6. Bound from map points

An event type is depended on the severity level of markers inside the event. Severity is an integer that shows the impact of flood in an area. The user who uploaded the marker gives it a severity level.

The core component of the system is the KML file. This file holds all the relevant data that is to be presented to a user. This file is recreated periodically from a centralized database and updated in shorter intervals. Any user interface would read its data from the file and present it in an appropriate form. A php script would update the database in case of a new submission. The structure at a coarse level is shown in figure 7.

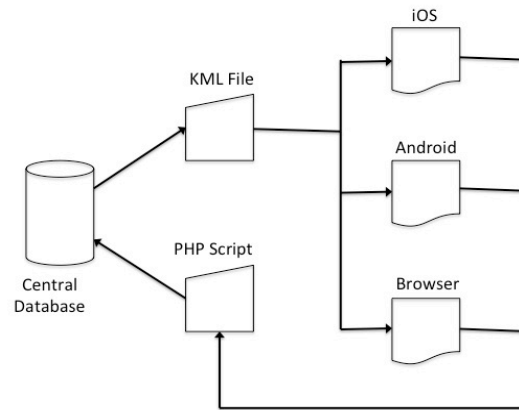


Figure 7. General architecture of Flood Monitor project

2.2. Existing and Ongoing Work

The current website for the project is hosted on <http://flood.cs.ndsu.nodak.edu/index.php>. The concept came out of a course project in North Dakota State University's course CSCi 372 – Comparative Programming Languages. As a part of extending the project, the website has been remodeled and functionalities revisited and made more efficient and the project has been extended to the Android and iOS platforms. Although, all the frontends differ a lot in presentation and interfaces, the core functionality and basic application elements across each has been kept almost similar.

Related work in the field includes [8], research done by Simon R., & Fröhlich P.. In the paper they have presented an application framework that displays geospatial content on the WWW by enabling new methods of interaction and different types of user interfaces on advanced mobile phones and PDAs. It allows developers to create innovative geospatial applications on high-end devices maintaining interoperability with more conventional devices.

They also have implemented an XML data exchange format for the query result returned by the server.

CHAPTER 3. APPLICATION DESIGN

3.1. Overview

There are many types of application design paradigms that help in understanding the overall guidelines of a project, its functionality, data involved and logic. They help us envision the form of the final deliverable, logic for achieving that and hence identify the areas that pose potential problem.

For example, the flowcharts and activity diagrams act as a storyboard for the entire application. They determine the navigational flow of the application. The wireframes define the ‘rough look’ of an application, placing main emphasis on user interaction and the data presented as a result of that interaction. Unified Modeling Language (UML) as the name suggests, is a modeling language, very popular in the field of object-oriented software engineering. It includes a set of graphic notation techniques to create visual models of the software system. It is used to specify, visualize, modify, construct and document the concepts of an object-oriented software system under development [9].

Because of the limited resources, the user experience is critical and there is a greater need to create design prototypes, mockups [10]. With the limited screen space, wireframes and other design paradigms have high importance in mobile UI design process.

3.2. Challenges in Mobile UI Design

Unlike designing for PC users or static traditional platforms, designing UIs for mobile platforms can pose some challenges. The difference appears due to screen sizes and computing

capabilities of both the platforms. As pointed in [11], major areas that need special attention while designing for mobile devices are:

1. Optimal use of screen space.
2. User Interaction mechanism.
3. Design at large.

While designing a user interface for mobile devices, uniformity with other application running on the system must be maintained. All mobile platforms have their own UI libraries, so that the native application for the platform would have a common “look and feel” [10]. User wait time is a key concern in all consumer applications [6]. It is defined as the time a user has to wait until he sees a valid response of his query on screen or an error message if that request failed. These common elements are part of the user interface guidelines published by the company.

The below table from [11] summarizes the problem areas encountered in mobile designing.

While presenting the wireframe design in next section, the major design concerns and their solutions will be discussed.

Table 1. Main problem areas in mobile designing [11]

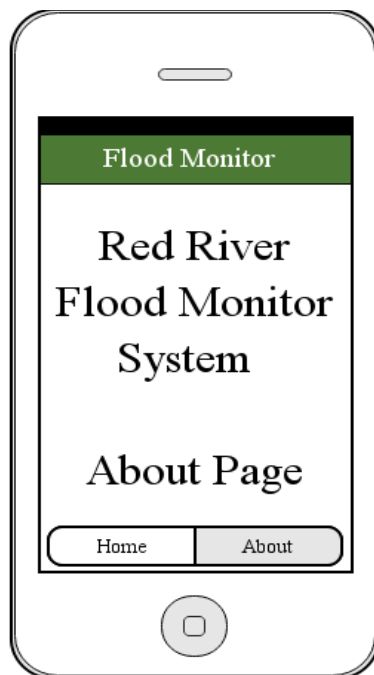
Main problem areas and problem areas structuring the patterns collection with their connected problems.	
Main problem area	Description and individual problems (with connected UI design patterns)
Utilizing screen space	<p>Focus on problems connected to the limitations regarding screen space on mobile equipment</p> <p>Addresses problems connected to different layout challenges on small screens</p> <ul style="list-style-type: none"> • Presenting elements in lists • Principles for grouping information • Mechanisms for grouping information • Mechanisms for packing information • Horizontal scrolling <p>Addresses problems connected changing the layout dynamically at runtime because the either the information that should be presented and/or the environment in which to present the information change</p> <ul style="list-style-type: none"> • Presentation based on models or data – how to do this on a small screen • Handling crowded dialogs when software keyboard is shown and hidden • Versions and variants – dynamic and configurable user interfaces on small screens • User interfaces that facilitate switching between portrait and landscape mode • User interfaces that are able to run on equipment with different screen size
Flexible user interfaces	<p>Focus on problems connected to the limitations regarding interaction mechanism on mobile equipment</p> <p>Addresses problems connected to entering information more efficient and/or with less probability for entering incorrect information</p> <ul style="list-style-type: none"> • Mechanisms for entering text • Order entry • Mechanisms for entering numerical data • Multimodal input • Controlling input cursor from an application
Interaction mechanisms	<p>Addresses problems connected to entering information in situation when it is not possible/convenient/desired for the user to use a stylus</p> <ul style="list-style-type: none"> • Interacting with applications without using stylus • Retrieving data from a database without using keyboard
Not using the stylus	<p>Focus on problems connected to design principles for user interfaces on mobile equipment</p> <p>Addresses design guidelines on different levels of generality</p> <ul style="list-style-type: none"> • Design guidelines for data base applications, including automatically generated user interfaces • Standard features that should be available in an automatically generated prototype • Design that supports branding, is aesthetic, and utilize screen space optimally • Solutions for searching large amounts of data • Visually coding of entry fields to mark editability (must, may, may not) • Standard solutions vs. usable tailored solutions
Design at large	<p>Addresses problems connected to providing understanding of what is happening when a mobile application performs functionality that can be difficult to understand for the end user – usually functionality that is specific for mobile applications</p> <ul style="list-style-type: none"> • User interaction during synchronization • User interaction for log-on/log-off • User interaction during waiting for long-lasting operations to complete
Guidelines	<p>“Difficult to understand”</p>

3.3. Wireframes

A design wireframe is a basic visual diagram depicting interface structure and the relationships between its pages. They are blue prints that define each screen's structure, content and functionality. They are usually defined before any design work starts so that the focus is on layout, functionality and navigation without the detailing of finer elements. [12]

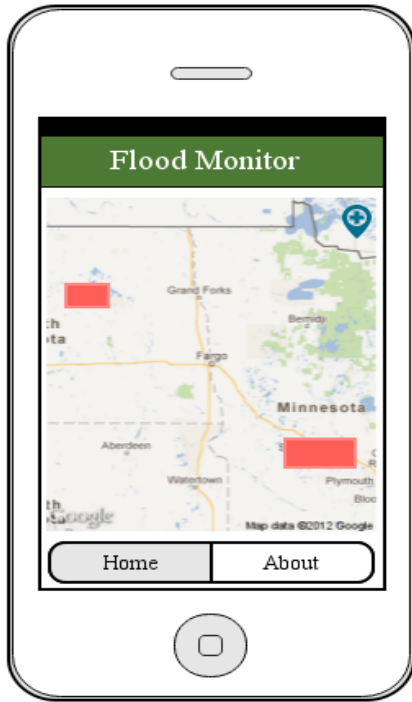
Functional wireframes are used in building web/mobile applications. It shows structuring of each screen, information about each widget, button, field, each piece of content, and the navigational flow. [12]

Figures 8 through 13 show the functional wireframes of the application along with an explanation of the design.





The about tab section of the Flood Monitor system displays about page of the website. It utilizes the UIWebView and displays the project data.

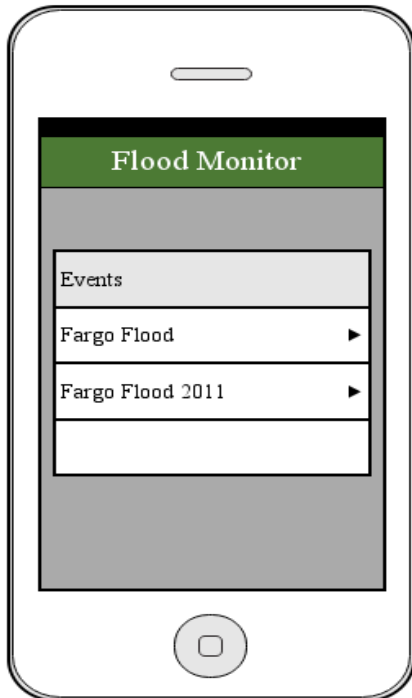
Figure 8. Wireframe 1: Screen showing a tab-based structure



When the user opens the document, a map view is displayed. The regions for which the data is available are marked as Map Overlays. The user can interact with map overlays using the `UILongPressGestureRecognizer`.

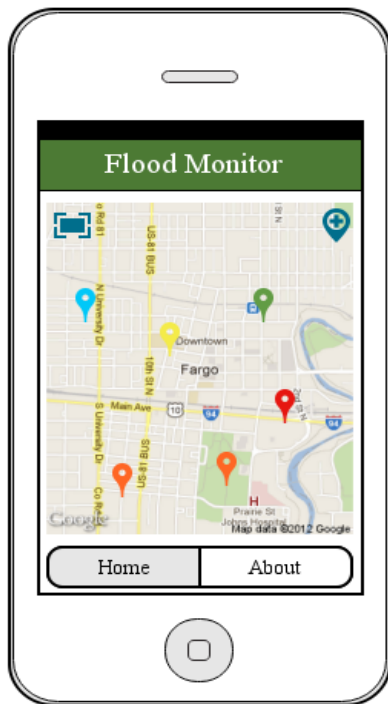
 button in place to let the user upload a marker into the system. When clicked presents the user with a simple form to fill out the details in and then submit.

 indicates the region overlays. Each overlay represents a collection of structures (regions, events, markers).




When the user clicks on the region overlay he wants to interact with, a `UITableView` containing the names of related events pops up. The user can then click on any event he wishes to see.

Figure 9. Wireframe 2: Initial view and generated response to user query



After selecting an event, a KML file is retrieved as a response from server. The KML file is parsed for information at the user's end and then the markers are displayed on the map. The markers correspond to the locations for which the data is in the system. Markers are customized and displayed using MKAnnotation. The user can normally interact with the markers (UITapGestureRecognizer). The markers are color coded based on the recorded severity level of the flood event.

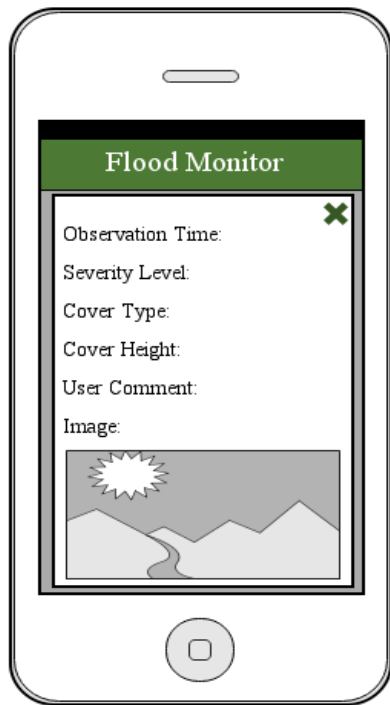
 button indicates that the user can go back to the initial map zoom level, on clicking this.

The severity level is indicated by the color coding on the marker.

Figure 10. Wireframe 3: Screen showing markers with different severity levels

This operation of showing the markers involves reading and displaying large amount of data, depending on user's choice. The delay here must be handled carefully as to avoid user dissatisfaction. Either a good feedback mechanism or no delay in providing information is a plausible solution. In [8], researches have considered *thematic consistency* in accordance with Mobile Web Best Practices (MWBP) guidelines, under which the designed framework provided a single unified data output format that is suitable for all mobile devices. In the same sense our server sends geo data and other information in form of KML and XML that are platform independent languages and easily interpreted for both Android and iPhone.

In this case the solution was to use caches. In the event where there is a need to transfer large amounts of data in response to user action, caching the data is a possible solution [11]. When the user access an event, data from his cache is read and is very much faster than downloading the marker file each time.





Once the user clicks on the marker, a pop up appears showing the relevant details of the marker. This data has already parsed and is a part of the KML file returned by the server.


Figure 11. Wireframe 4: Marker detail view



We are trying to display a large amount of marker-related data. Very often this data is larger than what can be effectively presented on the iPhone screen. A design concern could be how to display this information with two options being, vertical scroll or horizontal scroll. Vertical scroll is always preferred to horizontal scroll. The reason being that when scrolling horizontally, it is easier to lose the information context than when scrolling vertically.

Information on the same line is viewed as more closely related compared to information on different lines [11].



On Clicking  an MKAnnotation pin is dropped to the center of the visible map view. The  is draggable and the subtitle for MKAnnotation object corresponds to the street address of the pin point. The street address id acquired through reverse geo coding.

Since the number of upload pins that is to be displayed to the user is just one at a time, the  is hidden unless the user has completed or canceled the data upload action.

 is used to cancel the data upload event. When the button is selected, the  is removed from the map and view returns to original.


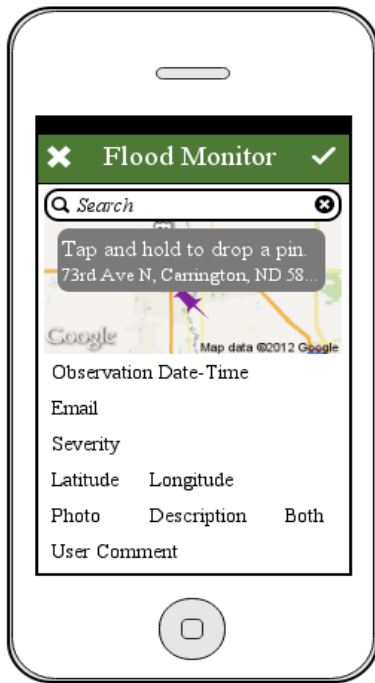
 is used to confirm the data upload event. When the button is selected, it implies the user is willing to upload an event data and id presented with a form.

Figure 12. Wireframe 5: Marker submission action initiated by the user

While filling out the form shown in figure 13, a software keyboard is present on the screen at all times. That obscures the information on the screen. Vertical scroll bar is added to prevent this problem [11]. When the user taps on a field that needs input, a scroll bar appears so that, he can scroll and see all the information on the screen.



The map view and MKAnnotation view has same functionality as on the parent view. Apart from dragging and dropping the annotation pin to reach the desired location, the user can search the address as well. The search is implemented using UISearchBar and its delegate classes.

There are other fields that the user would need to fill up, the observation time and severity are mandatory fields. Apart from that, the latitude and longitude fields are automatically filled, depending on the annotation pin location. After the user filled in information he can submit the data or cancel the process any time.

Figure 13. Wireframe 6: Marker submission form

3.4. UML 2 Use Case Diagrams

The best way to describe the functionality of a software-intensive system in a horizontal way is to use a UML Use Case Diagrams [13]. Use Case Diagrams (UCDs) represent the details of individual features of a system and all of its available functionality. The UCDs are essentially different from sequence diagrams or flow charts as they do not specify the order or number of times that the systems actions and sub-actions are to be executed [13]. Figures 14 through 16 show the UCDs of the application along with an explanation of the design.

In Figure 14, a very high level design at an initial stage shows the basic interaction between user and the server via the application.

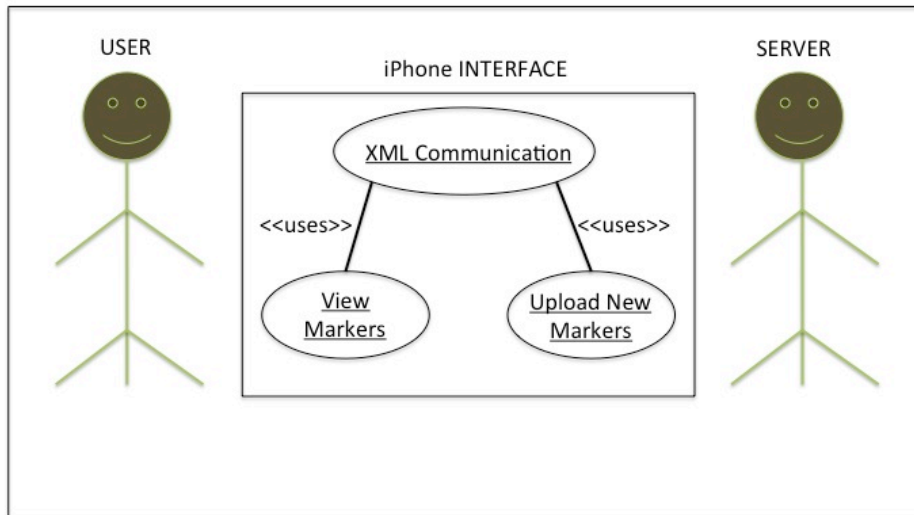


Figure 14. Use Case Diagram 1: Interaction between user and server

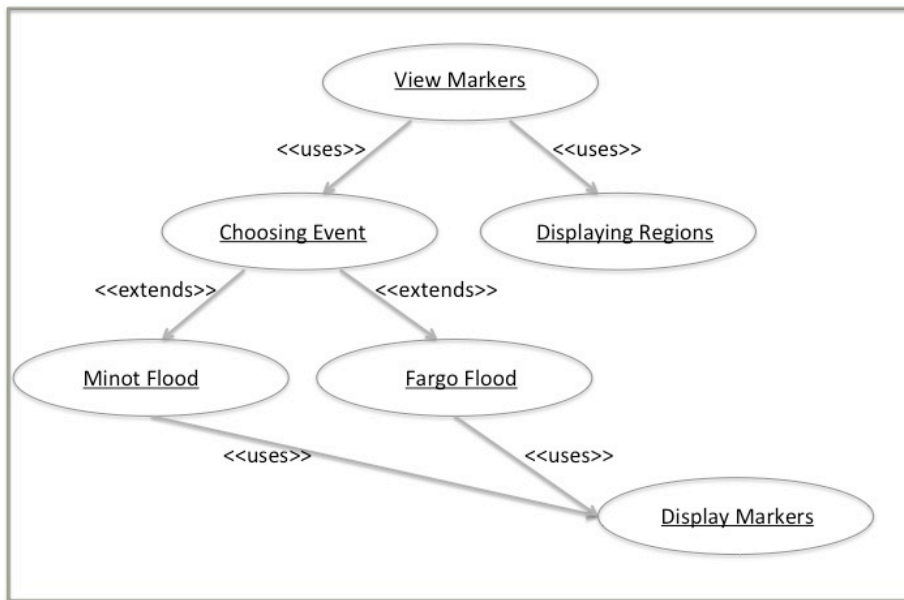


Figure 15. Use Case Diagram 2: Detail diagram view markers action

In Figure 15, detailed design for explaining the view markers action is shown. It incorporates 5 new actions. A very high level design at an initial stage shows the basic interaction between user and the server via the application. The view markers has to be done after the regions are displayed and the user has chosen an event to view, that is why the uses edge from view markers to choosing event and displaying regions. Currently we take example regions as Minot and Fargo. The user has to choose either Minot or Fargo event. When the user chooses an event the markers belonging to that event is displayed to him/her.

The figure 16 below shows a detail diagram for generating and receiving XML response from the server. The server generates the send response action when an HTTP Post request is received from the iOS device. This request can be generated as a result of user queries. Once the server has the request it generates a valid XML response and sends it to the phone.

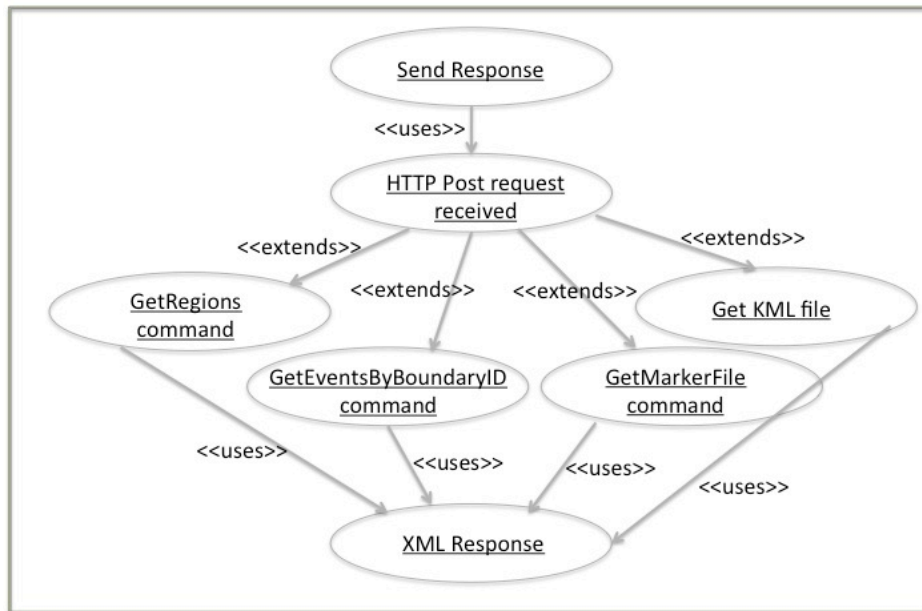


Figure 16. Use Case Diagram 3: Detail diagram generating and receiving XML response

CHAPTER 4. IMPLEMENTATION/DEVELOPMENT

4.1. Development Overview

Before diving into the development on any mobile platform, one should be aware that the limited resources like available screen size and various forms of user interaction impact the design and are also major influences on application development. As stated in [10], mobile user interface design process is centered on widgets, touch, physical motion, and keyboards and is not the same as the “window, icon, menu, pointing device” WIMP interface style of popular desktop systems.

A developer has to consider various tradeoff scenarios and decide the optimal path. For example, which processes should be done on the client (mobile, in this case iPhone)? Once that incur greater processing are delegated to the server. Other possible questions such as to maintain a cache or depend on network communication all the time and the frequency of cache updates, need to be addressed.

4.2. XML Communication

As stated in [14], one of the problem areas while addressing the development of a single application in a distributed manner is that the parts that run independently on different devices have to be identified. The parts that require heavy processing are delegated to a proxy. Usually it is a stationary host, running without user interaction. The clients access the proxy via a proxy interface [14]. In this application iPhone is taken to be the client and our server plays the part of the proxy with XML communication protocols as the interface between the two.

Extensible Markup Language (XML) is a markup language that encodes a document based on a certain set of rules understandable by both human and machine. The design goals of XML stress simplicity, generality, and usability over the Internet [15]. It was designed to store and communicate data, without modifying it. It is now widely implemented to accomplish above process and aid the communication between all sorts of applications [16]. XML is described as a semi-human readable stream of tagged data items. It has a schema describing the format of the message and name of the tags. Multiple schemas can describe one document; the tags have their own namespaces removing the ambiguity as to what tags belong to what schema. The result is self-descriptive structured document. XML is extensible, allowing addition of new tags and schemas without changing the meaning of the older ones. This makes XML favorable for use on multiple platforms. [17]

In [17] researchers have explored the possibility of using XML as the language of embedded systems communication over a heterogeneous network. They have argued that if multiple devices could interact with each other using the same language, it could allow for easy creation of seamless applications. This interoperation adds value to the whole system. In [18] researchers have pointed out advantages of using XML as communication language in heterogeneous systems:

- XML has the ability to be efficiently and easily generated, parsed, edited and translated, compared to other encoding formats like Lisp and plain ASCII. XML has a strict and extremely consistent syntax and a large number of XML tools available for all the above functions.

- XML has a schema that describes the information contained within the document giving developers the flexibility over modification of the document structure and vocabularies to specify different information. Hence, it is easily modifiable.
- The parsing standards of XML document and the document itself is open. This openness makes it interchangeable, satisfying the interoperability criteria present in most heterogeneous systems. Information can be transmitted from a source capable of generating XML and received by the system that is capable of parsing it. (This can be clearly seen in this paper).
- XML's hierarchical structure provides for representing various sorts of information in the real world, such as documents, databases and objects. This allows it to form a uniform information base. This information base is easily manageable in files/databases and is programmable.
- Most importantly, XML is platform independent making it very suitable for heterogeneous environment.

In our flood monitor system, as pointed out earlier there are three interfaces: Web, Android and iPhone. All these platforms have their own languages of implementation. For these platforms to interact with one single server that provides data, it would be inefficient to create an interface in three languages suited to each one. This also constitutes a scenario similar to a heterogeneous network. Hence, a preferred way to communicate was through XML. The backbone of the app is the XML communication between the device and the server. The

application talks with the server through the XML communication API. Commands should be sent via a HTTP POST and device will receive the response as an XML string.

When the app launches, the home screen shows two boundaries, each for Fargo and Minot. The class responsible for implementing the home screen uses MapKit framework and implements the MKMapViewDelegate. The MKMapViewDelegate protocol specifies methods that are available to developers to be used to receive map-related update messages. Many times as performance criteria, applications require the MKMapView class to load data asynchronously. Its methods then come handy in notifying the application when specific operations are completed [19]. These methods are responsible for generating annotation and overlay views and also provide for interaction management with those views [19]. The figure 17 shows the HTTP Post request sent to the server and response received for loading the boundaries when the application launches.

When the user interacts with any of the overlays and fires the UILongPressGestureRecognizer event. UILongPressGestureRecognizer is a concrete subclass of UIGestureRecognizer that is invoked after a long-press gesture. The user must press one or more fingers on the view for at least a specified period of time for the recognizer to be invoked [20]. The region being pressed is recognized and other HTTP Post request sent to the server. The request includes the ID of the boundary that was interacted with.

GetRegions

```
<phone>
  <command>GetRegions</command>
</phone>
```

This will get a list of all event regions.

GetRegions_Response

```
<server>
  <command>GetRegions</command>
  <response>
    <regions>
      <region>
        <id>##</id>
        <name>$$</name>
        <boundaries>
          <boundary>
            <id>##</id>
            <name>$$</name>
            <northwest>LAT, LONG</northwest>
            <southeast>LAT, LONG</southeast>
          </boundary>
          ...
        </boundaries>
      </region>
      ...
    </regions>
  </response>
</server>
```

This will return all the regions and their boundaries

Figure 17. GetRegions XML format

As the response in figure 18 shows that the XML is a list of events (events are clustered using a start date and an end date) that belong to the boundary whose ID was sent in the request. The event object includes the event ID, name, start date, end date and other details of the event. When this data is parsed, it is showed to the user in a UITableView as shown in figure 19.

GetEventsByBoundaryID

```
<phone>
  <command>GetEventsByBoundaryID</command>
  <params>
    <BoundaryID>##</BoundaryID>
  </params>
</phone>
```

This will get a list of events that share a single boundary.

GetEventsByBoundaryID_Response

```
<server>
  <command>GetEventsByBoundaryID</command>
  <response>
    <events>
      <event>
        <id>##</id>
        <name>$$$</name>
        <begindate>YY-MM-DD</begindate>
        <enddate>YY-MM-DD</enddate>
        <markercount>##</markercount>
        <active>##</active>
      </event>
      ...
    </events>
  </response>
</server>
```

This will return a list of events that share a given boundary.

Figure 18. GetEvents XML format

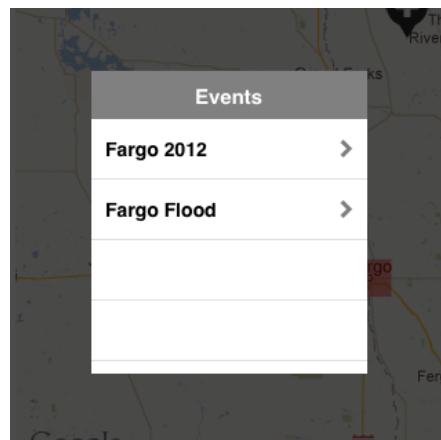


Figure 19. UITableView showing events for Fargo boundary

The user is allowed to dismiss the table view by clicking on the black translucent overlay or he selects an event. Once the user selects an event, another post request is sent to the user with the event ID selected. The boundary ID to which the event belongs is also sent.

GetMarkerFile

Method 1

```
<phone>
  <command>GetMarkerFile</command>
  <params>
    <boundaryid>##</boundaryid>
    <eventid>##</eventid>
  </params>
</phone>
```

This will return info for a KML file with all markers belonging to the event and boundary.

GetMarkerFile_Response

Method 1

```
<server>
  <command>GetMarkerFile</command>
  <response>
    <kmlfile>
      <id>##</id>
      <file>$$$</file>
      <version>##</version>
    </kmlfile>
  </response>
</server>
```

This will return the URL to a KML file with all markers belonging to the event and boundary.

Figure 20. GetMarker XML format

The response shown in figure 20 contains the URL to the relevant KML file that has the marker data. The file data is retrieved from the specified URL. The file and the ID number are both cached, to hasten any subsequent access to the same data. KML stands for Keyhole markup Language and is used to display geographic data on an Earth browser such as Google Earth, Google Maps, and Google Maps for mobile [21]. KML is based on XML standards and has a

tagged set of data items having nested elements and attributes. The KML reference contains all the knowledge on what tags to display, if they are optional or required, and what is the order of their appearance. The tags, in the document, are case-sensitive and must be appear exactly as specified in the KML Reference. The Reference indicates which tags are optional. Within a given element, tags must appear in the order shown in the Reference [21].

Scientists, researchers and KML developers have been utilizing the KML format to display scientific data and explain scientific phenomenon. When considering possible KML creation methods, a developer must be familiar with data set attributes, data volume and styling requirements and intended audience [22]. There are two ways to go about creation of a KML file.

1. Script to generate the KML file – This method involves manually writing the opening section, KML block for each data element followed by a closing section [22]. The advantages include easy modification to suit one’s need offering complete control over functionality and styling. The term “styling” is generally used for coloring and classification that differentiates an area visually.
2. Using open source geospatial software to generate KML. They offer flexibility in terms of use and can easily be incorporated into various computing scenarios. The downside can be the high learning curve [22].

We have adopted the first approach using a php random access file writer. With that our team member was able to set the writer at a location near the end and then append data to the file. The format for the KML file is shown in the below figure 21:

The KML Format

The Style Elements

There are 5 style elements. These are used in the google maps api to determine the image of the marker.

For Example:

```
<Style id="sever_5">
  <IconStyle>
    <Icon>
      <href>http://flood.cs.ndsu.nodak.edu/~ander773/flood/images/okay.png</href>
    </Icon>
  </IconStyle>
</Style>
```

Style	This defines a style tag used in the KML standard. Not: the id="" is used for by the placemark elements.
IconStyle?	This says we're defining a style for an icon placemark
href	This is the url to the icon style

Placemark Elements

These are the actual markers.

For Example:

```
<Placemark id="0">
  <styleUrl>#sever_4</styleUrl>
  <Point>
    <coordinates>-96.79801900000000,46.90281100000000</coordinates>
  </Point>
  <ExtendedData xmlns:mark="http://flood.cs.ndsu.nodak.edu">
    <mark:ObservationTime>2010-03-11 15:53:41</mark:ObservationTime>
    <mark:UploadTime>2010-03-11 15:53:41</mark:UploadTime>
    <mark:Email_</mark:Email>
    <mark:PhoneNumber>555555</mark:PhoneNumber>
    <mark:Valid>t</mark:Valid>
    <mark:UserComment>Puddles on the road, nothing serious yet.</mark:UserComment>
    <mark:ImageUrl> http://flood.cs.ndsu.nodak.edu/pictures/35.jpg</mark:ImageUrl>
    <mark:CoverType>_</mark:CoverType>
    <mark:CoverHeight>1</mark:CoverHeight>
  </ExtendedData>
</Placemark>
```

Placemark	This defines that we putting some object on a google map
StyleUrl	The ID of the style we defined above to apply to the marker
Point	We are placing a marker point
Coordinates	The coordinates of a marker. Format <i>Lat, Long, elevation</i>
ExtendedData	Extra data that is ignored by google maps, but used by the programmer.
mark:ObservationTime	The date and time the marker was observed
mark:UploadTime	The time the marker was uploaded or added to the database
mark:Email	The email address of the person who uploaded the marker
mark:PhoneNumber	The phone number of the person who uploaded the marker
mark:Valid	Is the marker valid or not? <i>t</i> for yes, <i>f</i> for no
mark:UserComment	The user comment left by user
mark:ImageUrl	The url of the image of the marker
CoverType	The cover type of the marker
CoverHeight	The height of the cover type

Figure 21. KML file format

The KML file is parsed to get an array of MKAnnotation objects. These are displayed on the map as markers. This file is cached in the device to accelerate any subsequent access of the same data. When the user tries to access the KML file for an event that has already been cached a different request is sent to the server and in case the file is not having the latest data, only the data that is not present in the file, is returned by the server.

The next main function is that of uploading the marker data. Once the user has filled in all the data for an upload, the data is formatted in an XML and in case there is an image, it is processed to byte code. This data is now sent to the server via an HTTP post request as shown in figure 22:

SubmitMarker

```
<phone>
  <command>SubmitMarker</command>
  <params>
    <latitude>##.###</latitude>
    <longitude>##.###</longitude>
    <observationTime>MM/DD/YYYY HH:MM</observationTime>
    <phoneNumber>###-###-###</phoneNumber>
    <severity>##</severity>
    <coverType>$$$</coverType>
    <coverHeight>##</coverheight>
    <uploadTime>MM/DD/YYYY HH:MM</uploadTime>
    <pictureData>$$$</pictureData>
  </params>
</phone>
```

SubmitMarker_Response

```
<server>
  <command>SubmitMarker</command>
  <response>
    <msg>$$$</msg>
  </response>
</server>
```

*This will return a string with a status message.
It'll either tell you if you added the marker successfully, marker and image successfully, or just added the marker successfully.*

Figure 22. SubmitMarker XML format

Depending upon the response format for which is shown figure 22, corresponding user message is displayed on the device. The message either conforms a successful upload or asks the user to try again, in case the upload failed.

4.3. Data (Response) Parsing

Since all the data we receive from the server is in XML format, we cannot present it as to the user as it is. Hence, we need to parse the data and categorize it into unique objects. Each object has its own set of identifying data and is presented as and when required.

As explained earlier, the built-in class NSXMLParser is used for parsing an XML document including DTD declarations in an event-driven manner. Again, an NSXMLParser calls its delegate methods about the items (elements, attributes, CDATA etc.) that it encounters as it parses the document. It just reports the found elements and does nothing with those parsed items [7]. The classes that are used to parse the data implement the NSXMLParserDelegate protocol. The NSXMLParserDelegate protocol defines the optional methods implemented by delegates of NSXMLParser objects [10]. The three main methods of NSXMLParserDelegate implemented and utilized in the project are:

- **parser:didStartElement:namespaceURI:qualifiedName:attributes:** -- The parser object notifies its delegate that it has encountered a start tag for a given element [23].
- **parser:didEndElement:namespaceURI:qualifiedName:** -- The parser object notifies its delegate that it has encountered an end tag for a given element [23].

- **parser:foundCharacters:** -- The parser object to gives its delegate a string representing all/part of the characters of the current element [23].

As the characters are found, the data inside the opening and closing tags is stored corresponding to the object to which it belongs. The KML parser treats each marker as a unique object. So, when the user clicks on a specific marker, only the data related to that maker is accessed.

As pointed out in [6], out of the two available approaches to parse an XML document:

- Using a Document Object Model that creates a memory based tree structure and loads entire document in the memory.
- Event-driven model; generates events as the document streams through memory.

The latter is a better approach. In parsing of the XML and KML documents we follow the latter approach, where events are generated when a tag of application's interest is encountered (**parser:foundCharacters**). To this method we pass the tag name that is of interest and it generates events when that tag is encountered.

CHAPTER 5. APP PERFORMANCE ANALYSIS

All the Software Development Kits (SDKs) for mobile development provide an emulator/simulator on which the developer can test their application without needing to actually test it on a device. The Apple family of mobile devices is small and hence a simulator testing usually suffices for most applications but testing becomes quite challenging for larger product families, such as Android devices. It's insufficient to test an Android application just on an emulator; it has to be tested across many different Android devices running various versions of Android OS [10]. In this case we are dealing with an iPhone scenario so the major issues are memory leaks and loading time since the application gets its information from a server.

5.1. Loading Time

As discussed above, the user wait time or loading time is a very important criterion in measuring an app's performance, especially for mobile devices. In our application, we have used cache as an effective solution to improve the loading time of large amount of data sets. The cache is implemented in SQL Lite. The user wait time is negligible since the data is read from cache buffer that does not require parsing of the KML file; those overheads are handled in the background. The KML file is downloaded, parsed and stored in the database, when required, without the user noticing any delay in information presented to him. Hence the application's performance in this regard is acceptable. Figure 23 shows the loading time comparison, with and without, caching. The time may be dependent on the actual speed of Internet connection at the time of recording test data. But, the chart is a good approximation to see the staggering difference between the load time when the caching is implemented to when its not.

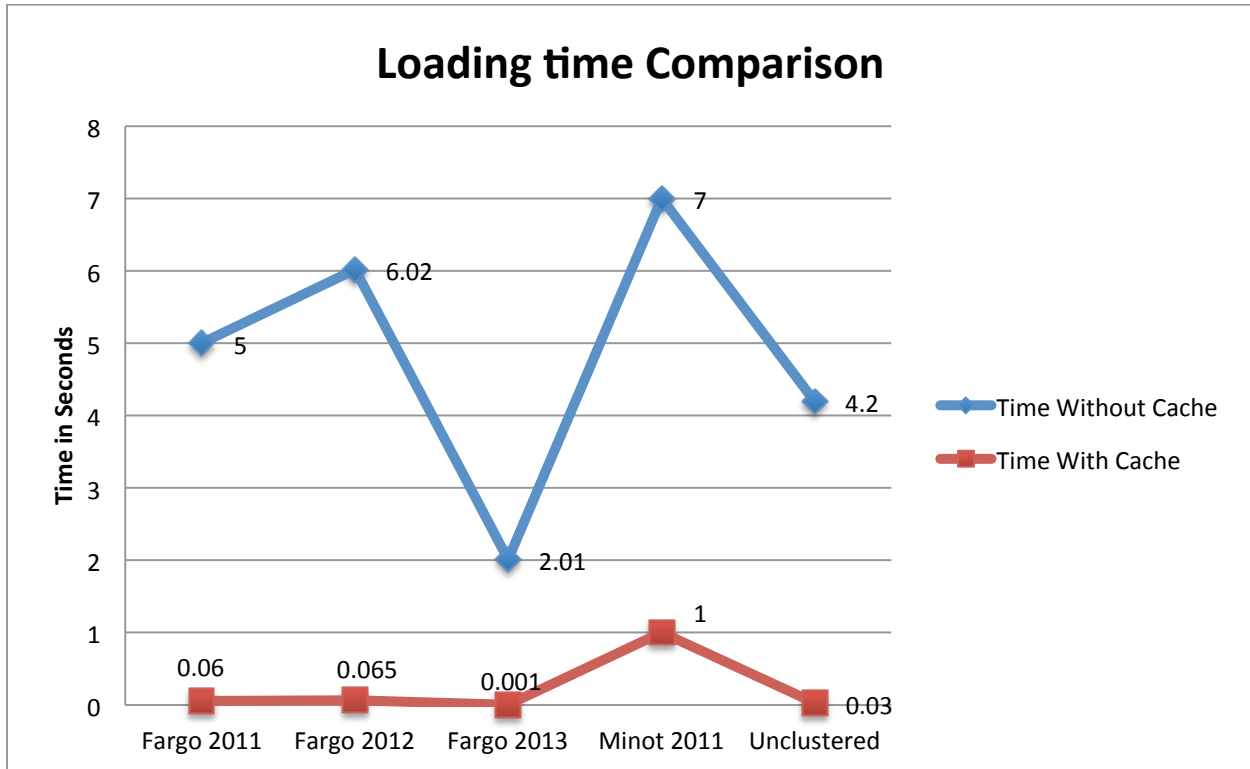


Figure 23. Loading time comparison

5.2. Testing

Two types of testing were performed on the application: Unit testing and Integration testing.

5.2.1. Unit Testing

This testing was conducted at completion of every single feature of the application. It is conducted by testing each module/functional unit independent of the system. The advantage of this sort of testing is that bugs are revealed at a very early stage in development. Some unit test case scenarios are:

- Testing the navigation between views: the most general cause of an application crash during navigation is due to mismanagement in memory; i.e. releasing the view from the memory before it is removed from view hierarchy.
- The data being displayed is correct: The parsing of the data is done correctly and is displayed appropriately.

5.2.1. Integration Testing

It involves testing the whole application as a single unit working cohesively to produce the desired result. This is done after unit testing phase, and hence most functionality bugs are fixed. This is done at the end of development phase. Some integration test case scenarios are:

1. Whole application is working correctly, without any crashes.
2. Data synchronized between client and server; no data losses

CHAPTER 6. CONCLUSION

6.1. Conclusion

Natural disaster such as floods cause heavy disruption in day-to-day life. Awareness about flood and areas influenced by flood is both necessary for people living in that area and others, in order to be informed, protect and prepare themselves for any emergency situations. The Red River floods in many years and disrupt life in the neighboring regions. This project is designed to make information sharing regarding flood events easy and efficient. The flood monitor system is designed to help people keep themselves and others updated in case of a flood in the Red River basin. The project is designed with the purpose of letting people see the details of an existing or a previous flooded area and submit new or update existing information.

Nowadays, many people have an iOS device at their disposal. Hence, sharing this data is easier, with the project available natively on the device. Keeping this idea in mind, we have developed this application that can be downloaded from iTunes app store and installed natively on user's device. With this app, users can share the data on the go. Thereby, the wait time between the observation of the event and submission of its marker data is reduced. The above factor also improves the accuracy of an uploaded event, as the user can (in most of the cases) upload an image or a piece of data within a very short span of time from when it was first observed.

The design of the application has been kept simple. There are only two views implemented: one showing the relevant markers based on user choice and the other shows a

form, which is used to submit marker data. This makes the application easy to comprehend and use.

The problem of user wait time was solved with the introduction of caching. The data is cached on the user device making the next access to this information faster. The issue of interoperability of heterogeneous systems was solved by XML communication. The devices (each interface) interact with the server using XML communication. This makes the communication uniform irrespective of the platform.

Hence, with this application, people living in Red River basin can easily share information and be alerted for any emergency situation.

6.2. Future Work

In any application, there is always a scope of further improvement. Since floods are events that have a social impact and this project is designed to spread as much awareness as possible, integration of social networking can be explored. The application can have a new section that is dedicated to the purpose of displaying this data. For example, considering Facebook integration, to access this part of the app, the user is prompted to enter their FB username and password. Once they are logged in, through a query to FB the data regarding the flood monitor could be gathered and presented to the user in a fashion similar to timeline. This could be a more user-friendly way to see the data and also since almost everyone is on one or the other social networking site, they are easily able to access this.

Another important integration could be to use push notification available on iOS devices. The user could opt for being notified via pushed notification whenever a new marker is placed or an existing marker is updated within a certain radius of his current location. These notifications would be generated via the push notification feature available on the iOS and the user could always come back and disable it. Explaining on a very coarse level, the application would read the KML file at certain intervals and whenever the KML file is updated, it would fire a notification invocation method in the application.

With these features implemented, the iOS application would be more powerful than the current one in terms of its additional features of social networking and push notification.

REFERENCES

- [1] Web Applications: What are They? What of Them?,
<http://www.acunetix.com/websecurity/web-applications/>
(Last accessed on 15 March 2013)
- [2] Touch Topics: Touch Terminology: What is Multi-touch?,
http://solutions.3m.com/wps/portal/3M/en_US/TouchTopics/Home/Terminology/WhatIsMultitouch/
(Last accessed on 16 March 2013)
- [3] iOS Technology Overview: About the iOS Technologies,
<http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html>
(Last accessed on 15 March 2013)
- [4] Rogers M. P., Wrong Number: Avoiding the hidden perils in iPhone development, *Journal of Computing Sciences in Colleges*, 2010;25(5):300-305
- [5] Xcode 4 Downloads & Resources – Apple Developer, <https://developer.apple.com/xcode/>
(Last accessed on 16 March 2013)
- [6] Christensen J. H., Using RESTful web-services and cloud computing to create next generation mobile applications, *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, 2009:627-634
- [7] NSXMLParser Class Reference,
https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSXMLParser_Class/Reference/Reference.html
(Last accessed on 17 March 2013)
- [8] Simon, R., & Fröhlich, P., A mobile application framework for the geospatial web, *Proceedings of the 16th international conference on World Wide Web*, 2007:381-390
- [9] Unified Modeling Language from FOLDOC, <http://foldoc.org/Unified+Modeling+Language>
(Last accessed on 1 January 2013)

- [10] Wasserman A. I., Software engineering issues for mobile application development, *Proceedings of the FSE/SDP workshop on Future of software engineering research - FoSER '10*, 2010:397-400
- [11] Nilsson E. G., Design patterns for user interface for mobile applications, *Advances in Engineering Software*, 2009;40(12):1318-1328
- [12] Wireframes, <http://www.usability.gov/templates/wireframes.pdf>
(Last accessed on 12 February 2013)
- [13] UML Use Case Diagrams: Tips, <http://www.andrew.cmu.edu/course/90-754/umlucdfaq.html>
(Last accessed on 12 February 2013)
- [14] Roth, J., Patterns of mobile interaction, *Personal and Ubiquitous Computing*, 2002:6(4);282-289
- [15] XML 1.0 Origin and Goals, <http://www.w3.org/TR/REC-xml/#sec-origin-goals>
(Last accessed on 12 February 2013)
- [16] XML Introduction – What is XML?, http://www.w3schools.com/xml/xml_what_is.asp
(Last accessed on 12 March 2013)
- [17] Helander J., Deeply embedded XML communication: towards an interoperable and seamless world, *Proceedings of the 5th ACM international conference on Embedded software*, 2005:62-67
- [18] Chen B., Linz D. D., Cheng H. H., XML-based agent communication, migration and computation in mobile agent systems, *Journal of Systems and Software*, 2008:81(8);1364-1376
- [19] MKMapViewDelegate Protocol Reference,
http://developer.apple.com/library/ios/#documentation/MapKit/Reference/MKMapViewDelegate_Protocol/MKMapViewDelegate/MKMapViewDelegate.html
(Last accessed on 16 March 2013)
- [20] UILongPressGestureRecognizer Class Reference,
http://developer.apple.com/library/ios/#documentation/uikit/reference/UILongPressGestureRecognizer_Class/Reference/Reference.html (Last accessed on 12 March 2013)

- [21] KML Tutorial – Keyhole Markup Language – Google Developer,
https://developers.google.com/kml/documentation/kml_tut
(Last accessed on 17 March 2013)
- [22] Ballagh, L.M., Raup, B.H., Duerr, R.E., Khalsa, S.J.S., Helm, C., Fowler, D., Gupte, A.,
Representing scientific data sets in KML: Methods and challenges, *Computers &
Geosciences*, 2011;37(1):57-64
- [23] NSXMLParserDelegate Protocol Reference,
http://developer.apple.com/library/ios/#documentation/cocoa/reference/NSXMLParserDelegate_Protocol/Reference/Reference.html (Last accessed on 17 March 2013)
- [24] Wireframes and Activity Diagrams are drawn using, <https://moqups.com>
(Last accessed on 10 June 2013)
- [25] UML 2 Diagrams Tutorial, http://ima.udg.edu/~sellares/EINF-ES2/uml2_diagrams.pdf
(Last accessed on 12 February 2013)
- [26] Introduction to UML 2 Use Case Diagrams,
<http://www.agilemodeling.com/artifacts/useCaseDiagram.htm>
(Last accessed on 12 February 2013)
- [27] Application Design Process,
http://www.columbia.edu/itc/visualarts/r4110/f2000/week09/09_02_Application_Design.pdf
(Last accessed on 16 March 2013)
- [28] Static Maps API V2 Developer Guide – Google Maps Image APIs – Google Developer,
<https://developers.google.com/maps/documentation/staticmaps/index>
(Last accessed on 17 March 2013)