Ubiquitous Interoperable Emergency Response System

Matthew J. Passini
Marquette University

UBIQUITOUS INTEROPERABLE EMERGENCY RESPONSE SYSTEM

by

Matthew J. Passini

A Thesis submitted to the Faculty of the Graduate School,
Marquette University,
in Partial Fulfillment of the Requirements for
the Degree of Master of Science

Milwaukee, Wisconsin

May 2014

ABSTRACT
UBIQUITOUS INTEROPERABLE EMERGENCY RESPONSE SYSTEM

Matthew J. Passini

Marquette University, 2014

In the United States, there is an emergency dispatch for fire department services more than once every second - 31,854,000 incidents in 2012. While large scale disasters present enormous response complexity, even the most common emergencies require a better way to communicate information between personnel. Through real-time location and status updates using integrated sensors, this system can significantly decrease emergency response times and improve the overall effectiveness of emergency responses.

Aside from face-to-face communication, radio transmissions are the most common medium for transferring information during emergency incidents. However, this type of information sharing is riddled with issues that are nearly impossible to overcome on a scene. Poor sound quality, the failure to hear transmissions, the inability to reach a radio microphone, and the transient nature of radio messages illustrate just a few of the problems.

Proprietary and closed systems that collect and present response data have been implemented, but lack interoperability and do not provide a full array of necessary services. Furthermore, the software and hardware that run the systems are generally poorly designed for emergency response scenarios. Pervasive devices, which can transmit data without human interaction, and software using open communication standards designed for multiple platforms and form factors are two essential components.

This thesis explores the issues, history, design, and implementation of a ubiquitous interoperable emergency response system by taking advantage of the latest in hardware and software, including Google Glass, Android powered mobile devices, and a cloud based architecture that can automatically scale to 7 billion requests per day. Implementing this pervasive system that transcends physical barriers by allowing disparate devices to communicate and operate harmoniously without human interaction is a step towards a practical solution for emergency response management.

ACKNOWLEDGEMENTS

Matthew J. Passini

I would like to thank my wife, Brittney, for her unwavering support and sacrifices throughout my academic career, especially while writing this thesis. I would also like to thank my daughter, Azaria, for her unconditional love despite my recent inattentiveness. And to my soon to be born son, who unknowingly provided motivation to persevere through the consistent difficulties and finish this project. My family has sacrificed far more than I have by allowing me the chance to complete my academic career, and the completion of this work is dedicated to them.

I would also like to thank the Marquette University faculty members that comprised my committee; Dr. Thomas Kaczmarek, Dr. Sheikh Ahamed, and Dr. Douglas Harris for their expertise and feedback while finishing this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Technology Axiom

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it." So begins the nearly quarter century old seminal paper in ubiquitous computing by Mark Weiser titled, "The Computer for the 21st Century" [1]. Despite its age, I assert no truer words regarding technology have since been uttered. I, and undoubtedly countless others, have taken those prolific words and structured my entire perspective around this axiom in order to create more meaningful technologies. It is with this perspective that I present the following thesis.

1.2 State of the Fire Service

A popular adage in the fire service is "100 years of tradition unimpeded by progress." Although grossly inaccurate, it highlights a fact that the fire service has many traditionalists who value proven customs far more than they value the potential progress made through various changes in their tried-and-true procedures, especially when that change involves bleeding edge technology. However, the fire service has slowly learned to embrace innovative technology, especially when empirical proof demonstrates how a change or addition to procedures can produce superior results or solve common problems.

Problems are something all too common to the fire service, especially considering the main purpose of the fire service is to solve emergency problems. However, the practice of doing so comes with its own issues, of which new ones are

emerging every year. The fire service has evolved into an all-encompassing emergency response department – handling issues including complex conflagrations, hazardous material situations, technical rescues, emergency medical services, and large scale mass causality incidents. While the service has continually taken on more responsibility, it has done so with less funding, less support, and less staffing.

In 1986, there were 1,045,950 firefighters in the United States, which equates to 1 firefighter for every 229 citizens. Since that time, the increase in general population has far outpaced the increase of firefighters. In 2012, there were 1,129,250 firefighters, which equates to just 1 firefighter for every 277 citizens [2]. More staggering is the total call volume increase. In 1986, there were a total of 11,890,000 incidents, which is about 11.36 incidents per firefighter per year. In 2012, there were a total of 31,854,000 incidents, which comes to 28.21 incidents per firefighter per year [3]. The increased call volume and decreased number of resources necessitates a better method of tracking, managing, and analyzing each emergency response.

Year	Firefighters	Population	Firefighters Per 1000 People
1986	1,045,950	240,130,000	4.35
1996	1,081,800	269,390,000	4.01
2006	1,140,900	298,380,000	3.81
2012	1,129,250	313,910,000	3.60

Table 1.1: Number of Firefighters per 100 People in the United States

Year	Firefighters	Incidents	Incidents Per Firefighter
1986	1,045,950	11,890,000	11.37
1996	1,081,800	17,503,000	16.18
2006	1,140,900	24,470,000	21.25
2012	1,129,250	31,854,000	28.21

Table 1.2: Number of Incidents per Firefighter in the United States

Furthering the issues involved with emergency response is the fact that 69% of firefighters are volunteers [2]. These members are responding from their homes, places of work, or anywhere in-between. Worse yet, there is no guarantee who will be available when the next incident occurs. Tracking who is responding on each apparatus is difficult for full-time departments as well. Although commanding officers generally know who is on duty and what apparatus are staffed, it becomes extremely difficult to track the status of each truck at all times of the day, or during large incidents with multiple departments responding.

1.3 Current Systems

Currently, incidents are dispatched and managed almost solely via radio transmission, which are inherently transient. The audio is transmitted once, and then lost. Even simple emergency scenes are hectic and filled with loud and distracting noises, making radio transmissions not only hard to hear, but hard to understand [4]. While wearing personal protective equipment, such as firefighter's turnout gear and self-contained breathing apparatus, it can be extremely difficult to reach the button to initiate the microphone of a radio device [5]. These radio systems are generally created using proprietary equipment, forcing regions to purchase costly infrastructures and specific

brands of radios and accessories. These expensive systems are rarely interoperable with other infrastructure, so neighboring communities often find themselves unable to communicate with each other. Although this century old technology has been a solid solution, far better technologies have become available for communicating between responders.

This thesis illustrates how departments can marry tradition with progress and implement a better way to manage incidents by pushing towards invisibility and aiming to weave itself into the fabric of everyday emergency response. Through utilizing ubiquitous computing concepts, wearable technology, sensor communication, and interoperable software standards, this system can decrease emergency response times and increase the overall effectiveness of emergency responses.

The following chapter provides a background for the hardware and software technologies that are incorporated in the system. The third chapter reviews similar systems and applications currently available. The fourth chapter summarizes the general system requirements, while the fifth chapter details the development and implementation of those requirements. Lastly, this thesis concludes by evaluating the implemented system and outlines the many opportunities for future enhancements.

CHAPTER 2: TECHNICAL BACKGROUND

2.1 Ubiquitous Hardware

Ubiquitous computing could perhaps be succinctly described as the invisibility of computing, occurring anywhere and everywhere. However, that does not mean that human-computer interaction is completely replaced. In fact, when Mark Weiser categorized the three original types of ubiquitous devices, they all had visual displays for users to interact with. Tabs were defined as wearable devices measured in centimeters, Pads were hand-held devices measured in decimeters, and Boards were devices with larger displays measured in meters [1]. However, in 2009, Stefan Poslad proposed additional forms of ubiquitous devices, some requiring no display or human interaction [6]. The first of his forms are Dusts, such as Micro Electro-Mechanical Systems (MEMS), without displays, ranging from nanometer to micrometer in size. Skins, which are devices or MEMS that are flexible in nature and generally interact with light or conductive materials, are intended to be used within clothing, painted on walls, or even within the skin itself through electronic tattoos [7]. Lastly were Clays, which can be larger MEMS or arrays of MEMS, and usually create three dimensional devices that resemble physical objects. When Clays are manipulated, they modify or otherwise interact with digital information representing the physical object.

With or without displays and human interaction, the heart of ubiquitous computing is sensor technology and the ability for devices to communicate. Today's mobile devices are filled with extremely useful sensors and other MEMS that allow users

to knowingly or unknowingly interact with the physical world and communicate their environment to other devices.

Infrared sensors are one such MEMS which measure physical proximity by emitting non-visible light and detecting the amount of infrared light that returns. Accelerometers measure the change in velocity, typically in 2 or 3 axes. Gyroscopes measure rotation and orientation, such as the angle at which a device is being tilted. Touchscreens detect the location of taps and swipes on the screen, while microphones capture audio. However, the most impactful sensor for this thesis is the Global Positioning System (GPS) receiver. GPS receivers detect signals from multiple satellites and measure the time it takes to retrieve a packet from each different satellite. They then calculate the device's location based on the distance from the various satellites.

2.2 Mobile and Wearable Devices

As of the end of 2013, the number of worldwide mobile device subscriptions was 6.8 billion, while the total worldwide population was 7.1 billion. It is estimated that the number of mobile subscriptions will surpass the entire world's population in 2014 [8]. In fact, the number of mobile devices had already surpassed the population in the United States in 2011, and the number of mobile devices per inhabitant continues to rise [9].

While mobile devices are an important part of ubiquitous computing, they are far from invisible. However, wearable devices, which are a subset of mobile devices, are an important step towards truly ubiquitous computing. Wearable computers trend towards invisibility and weave themselves into the fabric of everyday life by allowing users to literally wear the devices in a comfortable and familiar manner, breaking free of the

necessity to dig a mobile device out of a pocket and hold it up with one or two hands to interact with it. While wearable computers have been around for decades, or centuries depending upon the definition used, the last five years have been a sort of renaissance where watches, fitness trackers, and most recently glasses have boomed in popularity.

Most notably, Google has slowly released Google Glass, a wearable computer built into the frame of glasses. The display sits above the user's right eye, just above the standard line of sight. It is intended to be there when users want it, and out of the way when they don't. It allows users to interact with the device hands free through voice commands captured by the device's microphone. It also has a trackpad along the right temple that allows users to tap and swipe to navigate the interface. It is capable of taking 5 megapixel still pictures as well as 720p video. Its main audio is produced from a bone conduction speaker, which uses vibrations against the skull to transmit sound to the inner ear.

Aside from the trackpad, camera, and microphone mentioned above, its sensors also include an accelerometer, gyroscope, magnetometer, and light sensor. Currently, there is no access to onboard GPS. However, when paired via Bluetooth to a separate device, it uses the other device's GPS, if available. It can connect directly to 802.11b/g wireless networks or connect to other devices through Bluetooth, but does not currently have a cellular chip for communicating directly with cellular networks. It runs on the Android operating system and has built-in software for real-time turn-by-turn navigation, web browsing, email and SMS, as well as phone and video calling. The amount of Glassware, which is the term given by Google to applications and services written for

Google Glass, is rapidly increasing as new developers and users are gaining access to the device every day.

Glass's user interface is a series of cards that display in a timeline. A card can be thought of as a static web page that can display a subset of HTML, including text, pictures, audio, and video. The home card displays a clock and can be considered the middle of the timeline. To the left of the home card are cards with information about what is happening now or in the future, such as the weather or upcoming flights from the user's schedule. To the right of the home card are cards that are in the past, such as messages sent to the user or pictures taken by the user. Users navigate the timeline by swiping forward and backward on Glass's trackpad. Tapping on a card brings up the options available for that card.

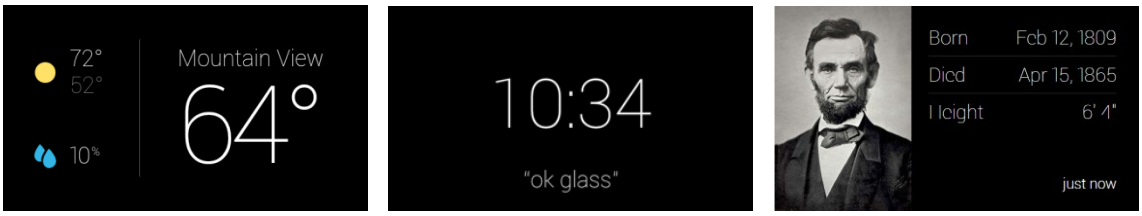

Figure 2.1: Timeline showing something happening now, the home card, and a search result

A key requirement for emergency responders is having hardware that is easy to use and always available. Although it is very common to dig a mobile device out of a pocket and stare down at the screen while using one hand to hold the device and another to interact with it, this is far from a satisfactory user experience during emergencies. Google Glass takes that necessary step towards ubiquitous computing, or rather, takes a step towards disappearing, by creating a device that is there when users need it and out of

the way when users don't. No longer do users need to dig out their mobile devices and use both hands to interact with it, they can simply nod their head upwards and speak to the device. This ease of use and always-present hardware significantly saves time, which is an essential feature for emergency response management. Glass does not attempt to create a virtual reality or even augment reality; it simply intends to enhance reality.

2.3 Ubiquitous Software

Ubiquitous hardware is just one half of the equation. Emergency responders also require the interface and overall user experience to perform the act of disappearing. For this, we turn to a term called geo-fencing. Geo-fencing is the creation of boundaries, usually in the form of a circle around one specific geographic point. The system can then detect when a user enters, leaves, or stays within the geo-fence for a period of time. By dynamically creating geo-fences based on incident location, the system can monitor user location and detect when they arrive and leave different geographic locations, allowing the system to update responders' status without specific user interaction.

The last major technical roadblock is device interoperability. Within emergency management, interoperability generally refers to the ability for one technology or device to communicate and operate with another. Interoperability has been a significant issue and became harrowingly evident on September 11, 2001 [10]. Since that time, substantial research has been completed and resources allocated to increasing interoperability between public safety departments and neighboring regions [11]. By utilizing open standards and a service oriented architecture (SOA), disparate devices and systems can seamlessly interact and share live data.

CHAPTER 3: RELATED WORK

3.1 Response Tracking

There are many commercial applications with similar functionality and goals as this system, but none match the ubiquity and interoperability described in this thesis, which are two keys to a practical emergency response system. One similar commercial system is IamResponding [12]. IamResponding is a cloud based system that focuses on exactly what its name states – the initial response aspect of an incident. The system has two main mediums for responders to reply to an incident, via a phone call, and via a notification sent from a native mobile application. Responders listen for dispatches through their regular medium, which is generally a radio signal sent to pagers. Once alerted, responders can dial a phone number and an automated phone service registers who is calling and sets the responder's status as responding to the incident. Responders can also use a native application - either Android or iOS - to notify the central service that they are responding and also view a map and directions to the scene, among other features. The system then aggregates who is all responding to the incident so it can easily be viewed on one screen. Optionally, incidents can be dispatched through the IamResponding web application.

3.2 Real-Time On Scene Information

While IamResponding focuses on the response to the station, an application being tested by the Philadelphia Fire Department provides firefighters with real-time mobile access to the city's Geographic Information System (GIS) by overlaying data on an interactive map for use while en-route or on a scene [13]. This native mobile application

for iOS and Android, which was written by the city's staff, allows responders to view the dimensions of a building and the occupant capacity. This allows commanders to decide how many resources are needed to safely extinguish the fire and handle transferring inhabitants to another location. Aside from structural building data, the application shows calls coming into the 911 system, the current location of fire trucks and ambulances, licenses and inspections data, storage location of hazardous materials, floor plans for large buildings and the location, size and working condition of fire hydrants, as well as the location of natural gas mains and lines running under streets and into a building.

3.3 Wearable Computing

A firefighter from Rocky Mount, North Carolina has coupled a previously written mobile emergency response application for Android with Google's Mirror API and Google Glass hardware to create a hands free notification system [14]. The system pushes dispatch data to Google Glass so that a map of the location and a text description are visible by simply glancing slightly upwards. A separate native Glass application was written to find the nearest fire hydrant, which can be initiated completely hands free by simply speaking, 'Ok Glass, find the nearest fire hydrant.'

3.4 Physiological Status Monitoring

The increase in availability, practicality, and feasibility of wearable sensors, such as ones that can measure biological signals, has allowed systems to incorporate the management and monitoring of firefighters' health in real-time. The Wearable Advanced Sensor Platform, or WASP, can not only pinpoint firefighters' locations, but can also assess their physical condition, such as heart rate, breathing, and activity levels in real-

time [15]. The physiological status monitoring (PSM) sensor fits into a customized t-shirt and sits on the side of the body just below the heart. The location tracking device attaches to a standard belt, to be worn on the inside of turnout gear. These sensors connect via Bluetooth to either an Android based smartphone or a Motorola APX radio. The Android device can then send the location and PSM data over Wi-Fi or cellular networks, and the Motorola APX can send the location and PSM data over radio waves. The main system then collects that data in a Windows based application and provides tools to rapidly analyze a firefighter's physiological response both in a live stream as well as over a period of time, while also showing the firefighter's current location.

System	This System	IamResponding	Philadelphia	Rocky Mount	WASP
Notification	All phone types Native Android Google Glass Web client	Android iOS Web client	N/A	Android Google Glass	N/A
Status Tracking	Apparatus and responders throughout entire incident	Initial response of responders only	Apparatus only	N/A	Physiological vital signs only
Location Tracking	Entering and exiting geo-fences	N/A	Apparatus only	N/A	Real-time responder coordinates
Ubiquity	GPS and geo-fencing	N/A	N/A	N/A	GPS and physiological sensors
Pre-Plan Access	N/A	N/A	Pre-plans and GIS data	N/A	N/A
Inter-operability	Cloud based with all phone types, native Android, Google Glass, and web clients	Cloud based with iOS, Android, non-smartphone, and web clients	Private servers with iOS and Android clients	Cloud based with Android and Google Glass clients	Proprietary hardware and network connectivity

Table 3.1: Feature comparison of related systems

CHAPTER 4: SYSTEM REQUIREMENTS

4.1 Non-functional Requirements

Designing a system to be used in situations where every second counts requires careful thought and significant industry experience. The nonfunctional requirements enumerated below are every bit as important as the functional requirements. While this system could be used in all facets of emergency response, focus is placed on the firefighting discipline, specifically highlighting the advantages for volunteer fire departments. A volunteer fire department is the epitome of a bring-your-own-device (BYOD) environment. Volunteer departments generally do not have the financial resources to pay their firefighters for their time, let alone to supply each of them with standardized mobile devices packed with sensors and communication technologies.

As such, the system must support standard cellular phones without an advanced operating system or data connection and also provide an advanced solution for those who do own a smartphone with sensors and other communication technologies, all while harmoniously communicating with each other and to a cloud based server. The system must also incorporate a large-format optimized user interface for use at static computing locations by administrative assistants, dispatchers, or command staff in a command vehicle or command post. These positions generally have the luxury of physical screen real-estate instead of being forced to use a diminutive mobile device. Another use for a large-format interface would be with a large screen hung in the apparatus bay so all responders can immediately view the current status of other responders from within the station. This ability for disparate hardware to operate and communicate together highlights the necessary component of interoperability.

In emergency situations, truly every second counts. The system is designed to require minimal input from the user while still providing relevant and useful information with only a quick glance. Emergency responders will simply not use the system if the time it takes to input or retrieve valuable information impedes their ability to respond to the scene and neutralize the problem as quickly as possible. Designing a user interface that is easy to navigate, read, and interact with is imperative to the success of the project. For users without a smart device, the system must allow for a method of interaction with just a click or two. For smart device users, the system must utilize sensors and advanced software to automatically detect contextual and environmental input without requiring a user to explicitly interact with the device.

Until recently, there was no other satisfactory hardware, aside from handheld mobile devices, for which such a system could be written. However, with the advent of Google Glass and Android Wear [16], emergency response systems now have the opportunity to integrate with wearable devices of all types to allow nearly hands-free interaction. A user no longer has to dig a phone out of a pocket, wake it up, unlock it, and open the application before being able to enter or retrieve important information. The combination of advanced hardware, which moves towards invisibility by implementing a computer into everyday wearable objects such as glasses and watches, coupled with advanced software, which allows devices and sensors to communicate without human interaction, progresses this system towards the necessary requirement of ubiquitous computing.

Aside from the requirements of optimizing for mobile, wearable, and large format hardware, as well as eliminating human interaction while still allowing for seamless

communication between devices, the software requirements also include administrative tasks. Administrators must be able to dispatch incidents as well as manage departments, devices, personnel, and apparatus. The system must also permit smart devices to easily register and unregister for notifications without administrative intervention.

Security and privacy are a concern for every application. Specifically for this system, we must ensure that the right devices are receiving the correct information in a secure manner. The system must also ensure that all communication with the central data store is originating from authenticated and authorized sources. Lastly, we must secure the administration of the site to a limited user group.

Performance, reliability, and scalability are vital to an emergency response system. When every second matters, a system cannot be sluggish to respond. Whether it is simple user interface navigation or updating a responder's status, all interaction and updates must be nearly immediate. Similarly, the system as a whole must be responsive and available 24 hours a day, 7 days a week, and 365 days a year, just like the responders who use the system. Lastly, the architecture supporting the system must have the ability to shrink and grow on demand to keep costs down when utilization is minimal, but be able to immediately increase to meet the demands of a large scale incident, or numerous concurrent incidents.

4.2 Functional Requirements

The above non-functional requirements build a solid foundation for any emergency response application. Although the system could easily be expanded to

include all facets of emergency response, the following functional requirements illustrate a system focused on the response aspect of fire department incidents.

The main functions of the system are comprised of alerting responders of an incident, allowing responders to acknowledge and update their status, and aggregating the responders' statuses to show the overall state of the incident. To achieve this, there is a web based application programming interface (API), a web based client application, a native Android application, and web services that interact with the Mirror API.

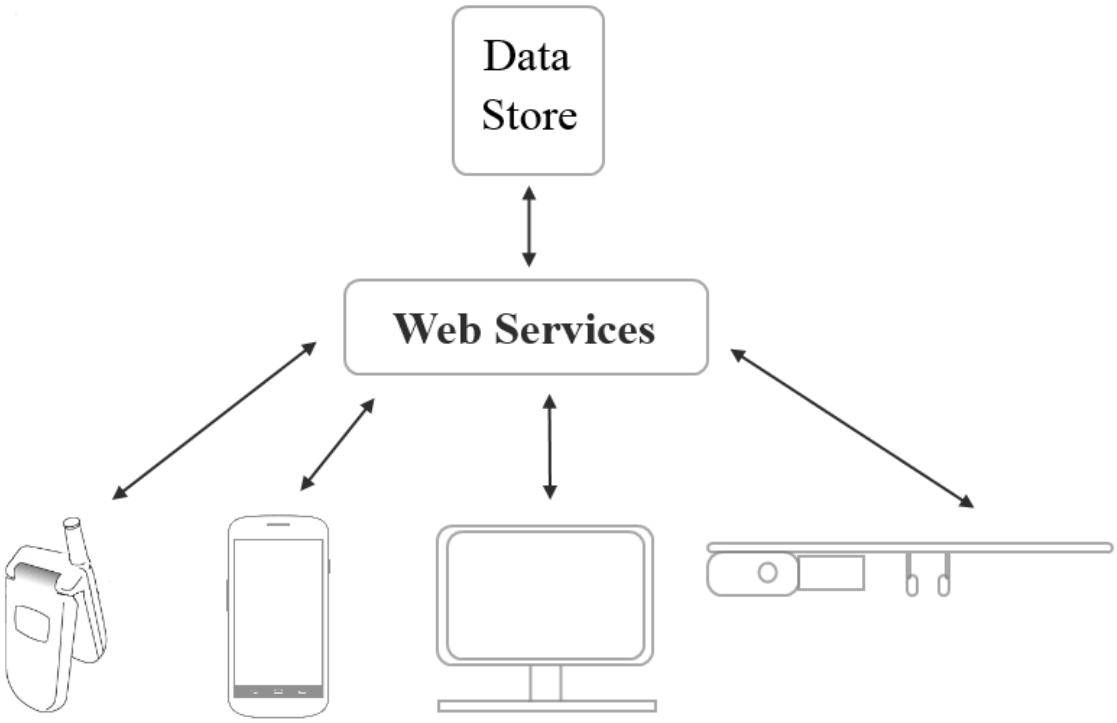

Figure 4.1: System communication and architecture overview

The web client application is accessible from any device with an internet connection. A landing page briefly describes the system and supplies a link to sign in to

the web application. If the user is not authorized, a generic unauthorized error page displays. If the user is authorized, the system brings the user to the web application. Within the web application, links provide access to the Dispatch, Incidents, Responders, Responder Devices, Departments, and Apparatus administration pages.

The first page visible to a user after authenticating is the Dispatch page, which contains a form to collect incident information and a button to dispatch the incident to all affected responders. Upon clicking this button, the system sends an instant notification to all affected Google Glass and native Android application users, and sends an SMS message to non-Android phone users.

The Incidents page displays all recent incidents and includes a button to set each incident as completed, as well as a link to navigate to the Response Details page which displays all the responders active with that incident. This Response Details page also includes an interactive map zoomed in to the location of the incident. A table displays the current status of all responders involved with the incident and refreshes automatically at a small interval so the display is always up to date. The web client application also includes a Responders page which displays all registered responders and includes the ability to add, edit, or delete as necessary. Similarly, the Responder Devices, Apparatus, and Departments pages list their respective data with the same add, edit, and delete capability.

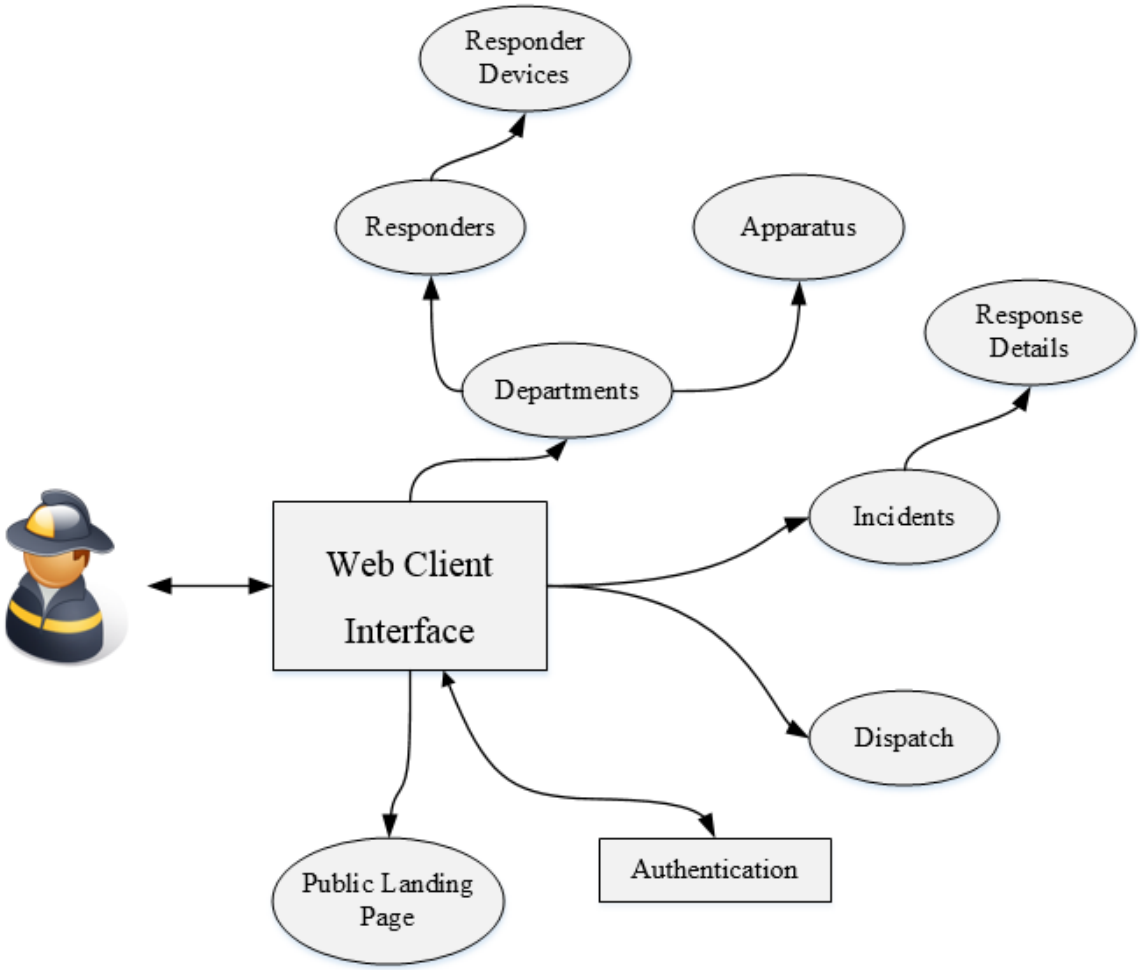

Figure 4.2: Web client interface flow

The backend of the web application is responsible for handling interaction with the Mirror API for communication with Google Glass. Glass users navigate to a URL within the system using any device to complete a one-time authentication of their account and 'install' the Glassware to their Glass. Outbound notifications are sent using the credentials created during that process, while responses sent back to the system from a user's Glass are captured and handled by a part of the backend web application.

When the system notifies Glass of an incident, a timeline card is inserted into each affected user's Glass timeline. The card contains a static map of the incident, the

address of the incident, and a description. Options are also included in the card, consisting of Acknowledge and Navigate. Choosing Navigate opens Glass's native turn-by-turn navigation system and routes the user directly to the scene. Navigate is available on every card until the user is on scene.

Upon choosing Acknowledge, the initial card is updated with new options. The Acknowledge option is replaced with an En-route option. A third option, Cancel, is also added and visible on every subsequent card, allowing the user to cancel the last action taken. After clicking the En-route option, the system updates the option to On Scene, followed by Returning to Station, and lastly In Service.

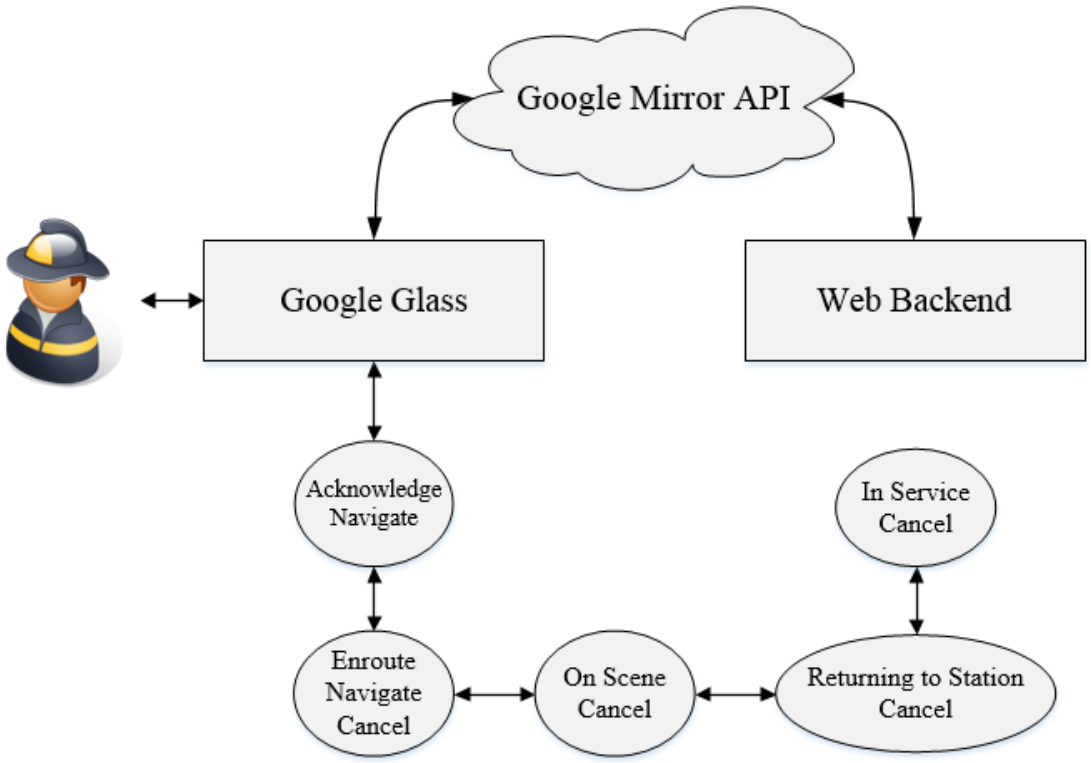

Figure 4.3: Google Glass flow

An Android application handles the native device requirements. The application includes a Main Menu view that allows the user to register and unregister with the system. Once registered, the Main Menu also includes a list of currently active incidents. The system is capable of receiving push notifications, which instantly alerts the device upon dispatch. By either clicking on the notification or clicking an incident within the incidents list on the Main Menu, the user is brought to the Incident Details view.

The Incident Details view always displays the address and description of the incident, an interactive map zoomed in to the location of the incident, and a Response Details button to change the view to show a list of all responders involved with the incident. There are two buttons that change or disappear - Acknowledge and Navigate. Upon pressing Navigate, the system opens the device's native turn-by-turn navigation application and routes the responder directly to the scene. Navigate remains visible on the user interface until the user is on scene. Clicking Acknowledge notifies the web application that the responder has acknowledged the call. It also accesses the user's current GPS coordinates to calculate an estimated time of arrival at the responder's station, and notifies the web application of that data.

After clicking Acknowledge, that button disappears and a grid of buttons takes its place, with each button depicting a different apparatus name, as well as a Cancel button to cancel the previous action. Once the user arrives at the station, dresses in the proper personal protective equipment, and begins to leave the station on a vehicle, the user can click on the button with the value of the apparatus name for which they are en-route with. Doing so updates the web application, signaling that the responder is en-route to the scene on the specified apparatus. The grid of buttons then disappears and a single action

button is displayed, titled On Scene. After clicking On Scene, it changes to Returning to Station, and then finally to In Service.

Clicking the Response Details button brings a new view into focus. This view is a list of all responders involved with this incident. The list includes each responder's name, followed by a description of the responder's current status, such as acknowledged call, estimated time of arrival, or en-route to the scene. A Back to Incident Details button is visible, and when clicked, returns focus back to the Incident Details view.

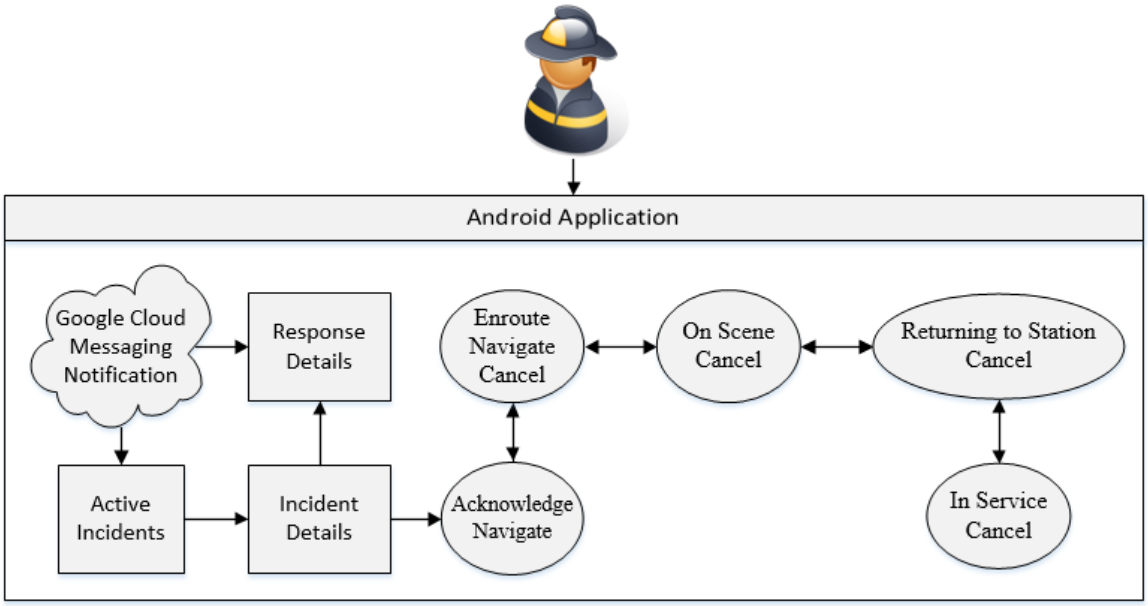

Figure 4.4: Android application flow

To fulfill the ubiquitous requirements of the system, the native application uses geo-fences to automatically detect when a user enters or leaves a geographical location. A geo-fence is a virtual boundary, generally defined by a single center point, along with a radius, which creates a circular boundary. Upon dispatching an incident, the system defines a small radius geo-fence around the location of the user's department and a larger

radius geo-fence around the scene. The system can then automatically detect when the user arrives at the department, leaves en-route to the scene, arrives on scene, and returns to the station. The system is then completely invisible to the user while still providing the same information as if the user was manually interacting with it.

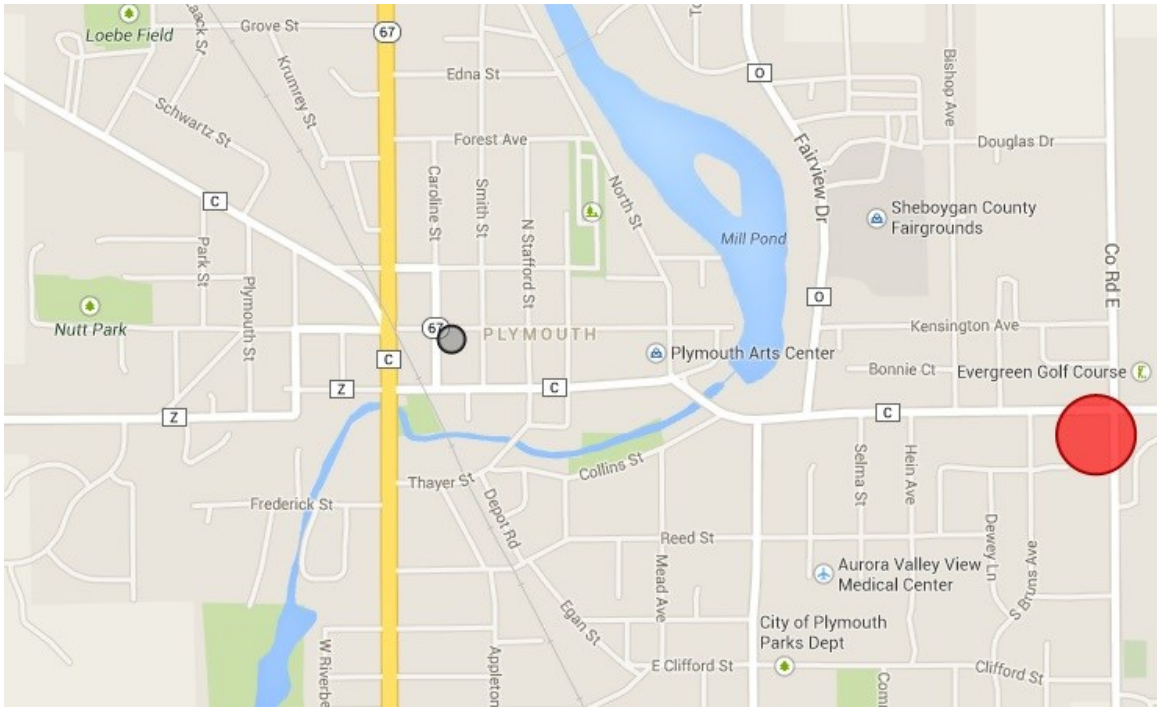

Figure 4.5: Example of two geo-fences of differing sizes

The final targeted device type is non-Android devices and standard cellular phones without a data plan, integrated GPS, or other sensors. To allow interaction with such devices, the web application sends SMS messages to the phone from the web application. Users can simply reply to this SMS, with or without any text in the reply message. This allows the user to interact with the system with as little as one click. The reply is routed to an email inbox hosted by the web application and then parsed. Similar to the previous user interfaces, the first interaction is acknowledging the call, while

subsequent replies via SMS updates the time that the user is en-route to the scene, on scene, returning to station, and back in service.

Interoperability is achieved through using the backend of the web application as a centralized service with the intelligence to know when and to whom to send updated information, and in what format. Whenever a device updates a status, the web application in turn sends notifications to the devices instructing them to update their user interfaces. For the Android application, any change in any responder's status updates the Response Details view by adding or updating the responder entry in the displayed list. This is done in real-time and requires no user interaction such as refreshing the view. Similarly, if a responder sends a status update using Google Glass or via an SMS message, the Incident Details view on that user's Android application automatically updates the user interface to show the appropriate buttons. Interoperability with Google Glass works just the same. If a responder were to send an update from a different device, the web application knows to update the card in that responder's timeline with the appropriate options. All devices stay synchronized without any further user interaction, regardless of the device type.

CHAPTER 5: IMPLEMENTATION

5.1 Development Environment

The development environment was shaped by the technologies chosen to implement the system. Eclipse, along with Android Development Tools and the Google Plugin for Eclipse, were used as the integrated development environment (IDE). Android was chosen for native mobile development mainly due to its global and national market share, as well as its popularity within the department where this system was tested. In 2013, there were nearly 1 billion smartphones shipped, with 80 percent of those phones running Android [17]. Although Android was chosen for the development of this project, there are no technical barriers stopping the development of an iOS, Windows Mobile, or other native application that would seamlessly integrate with the system.

5.2 Web Client Application and API

Performance, reliability, and scalability make the cloud an obvious choice to host the web application. Google App Engine was chosen due to the use of Android and the Mirror API, as well as its proven track record in configurable performance, reliability, and scalability [18]. App Engine also provides an easy to use email platform that can both send and receive emails, allowing simple parsing of incoming responses from responder's SMS replies. Likewise, Google Cloud SQL, a cloud based relational database, was chosen as the main data store for the same performance, reliability, and scalability reasons [19].

Java and JavaServer Pages (JSP) with servlets were chosen as the main server side programming languages [20]. Java was chosen for consistency with the Android

development, as well as its popularity when paired with Google App Engine. JSP, combined with servlets, provides more than enough functionality to handle requests and create the client side markup. Standard JavaScript along with the jQuery library provided the necessary tools to handle DOM manipulation and client-server interaction with the application programming interfaces (API) provided by the web application [21]. The Bootstrap CSS framework was used to provide fluid and responsive web design, along with system specific CSS [22].

Interaction between the devices and the backend web application is handled through secure RESTful (Representational State Transfer) services [23]. In general, the web services provide an interface for reading and writing to the data store. To create the services, data objects were written to model the entities of emergency response. At this point, the Google Plugin for Eclipse was used to generate basic endpoint classes, defining a generic shell of an API to list, retrieve, insert, and delete the specified model. Annotations are added to the methods to specify the endpoint location and other necessary attributes. This plugin greatly accelerates the initial development process by providing boilerplate classes that are easily modified and extended to provide the required business logic.

The Response Details page, which reloads the status of responders involved with the incident every 10 seconds using an AJAX (Asynchronous JavaScript and XML) [24] call to the web backend, along with an interactive map of the incident, illustrates the combination and implementation of the above technologies. This is the view intended to be available on a large screen hung in the department or on laptops in command vehicles.

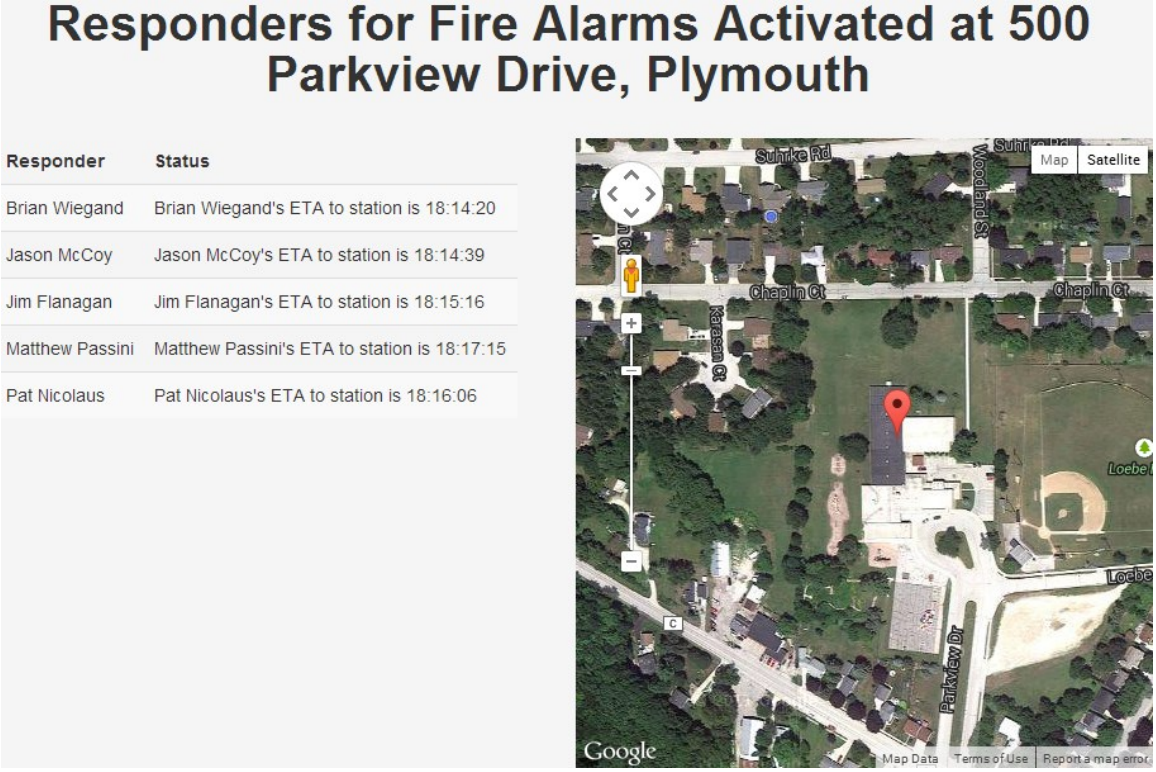

Figure 5.1: Web client view of active responders, their statuses, and an interactive map

5.3 Google Glass

In order to communicate with Google Glass, the web application handles interaction with the Mirror API. This type of service oriented architecture, coupled with the Mirror API, creates what Google calls Glassware [25]. There is no code from this system installed or running natively on Google Glass. The web application sends data to the cloud based Mirror API, which in turn communicates with the user's Glass. When the user communicates with this web application's services, the user's Glass first communicates with the Mirror API, which then forwards the request to this system's web application.

The web application not only sends timeline cards, contacts, and subscriptions to the Glass devices registered with the system, but also receives the actions taken by the

user, such as when a user chooses a menu item from a timeline card, like acknowledge. The web application then looks up the responder based on the credentials sent from the Mirror API and is able to update the data store with the data sent. It then sends a notification to the Mirror API to update the user's card with new options after successfully writing to the data store. When an incident is dispatched, all affected responders who are registered in the system with Glass instantly receive a card in their timeline.

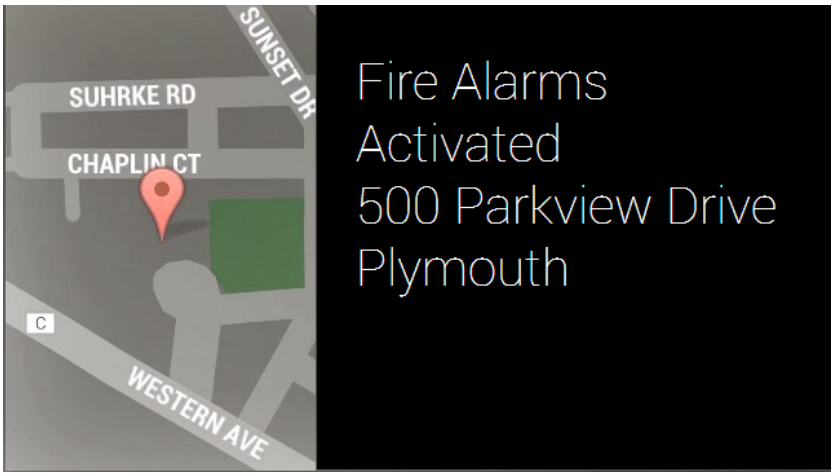

Figure 5.2: Example card sent to Glass after being dispatched

5.4 Non-Android Cellular Phones

In order to allow non-Android cellular phone users the ability to interact with the system, the web application sends an email to affected responders' cellular phones. The email is sent to the phone's email address as determined by the service provider. This email is received as a text message on the phone, which includes the incident address and description, as well as a link that opens a native mapping application if the phone is capable. Users can simply reply to this message with or without any actual text in the

body to indicate they acknowledge, are en-route to the scene, on scene, returning to the station, and finally, back in service. This allows for extremely fast and simple one click updates from any cellular phone.

The system uses Google's Mail API to both send and receive messages [26]. The same email address that is used to send the messages is used to receive them. Upon receiving an email, the system is configured to invoke a servlet, which in turn parses the incoming message. Based on the user's phone number, which is part of the email address, the system updates the responder's status in the data store accordingly. The system then sends updates to all other registered devices to keep them synchronized without further user interaction.

5.5 Security

Google's Users API is used within the web application to implement authentication and authorization. Through the Users API, the system can authenticate users via Google Accounts. Although this creates an otherwise unnecessary technical requirement for web application users to create a Google account, the account is free, easy to setup, and allows for simple and secure user authentication. If the user is already signed in to their Google account during the browsing session, the system automatically authenticates based on those credentials. If the user is not currently signed in, the system redirects the user to a sign in page hosted by Google, which ensures secure authentication.

Once authenticated via the Users API, the system can determine if the user is an administrator within the web application. Google App Engine provides a simple

interface in its API console for managing administrators. By adding Google email addresses to the administrator list, the system can utilize the Users API to determine if the authenticated user is authorized as an administrator within the system. If the user is not authorized, the system displays a generic 403 forbidden page. If the user is indeed authorized as an administrator, the Users API forwards the user on to the page that was requested, or defaults to the dispatch page if no specific URL was indicated. Subsequent calls to the web application APIs are made by passing along the Google Accounts User, so the web application can verify the user is authorized to make the call.

Another level of security is added through Google's API console. Google provides an interface to specify authorized origins for clients accessing the web application's APIs. The origin is the unique combination of protocol, hostname, and port. Multiple origins are accepted in order to allow the client application to run on different protocols, domains, and subdomains. In this case, we only want one specific JavaScript web client application to have access to the APIs, so that origin is the only one specified.

5.6 Android Application

The cornerstone of the native Android application is the implementation of Google Cloud Messaging (GCM). GCM allows applications to message and notify Android devices instantly without having to implement custom polling or other antiquated techniques [27]. When an incident is dispatched, the system sends a GCM message to all affected responders who have registered their device with the system through the native application's Main Menu view. The user does not need to have the

application running in order to receive the notification. The GCM service handles all aspects of queuing and ensuring proper delivery to the targeted Android device. When the device receives a notification it vibrates and/or plays an alert sound, based on the device's settings chosen by the user, and displays an icon in the notification area. To view the details, the user simply slides down on the notification area to open Android's notification drawer to see the address and description of the incident. Clicking on this notification opens the native application and displays the Incident Details view.

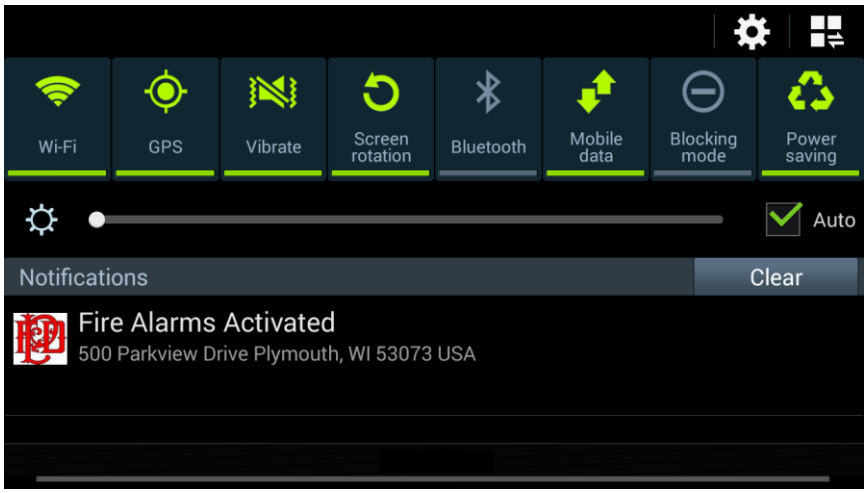

Figure 5.3: GCM notification in the Android notification drawer

Figure 5.4: Incident view displays incident details, interactive map, and action buttons

A second type of GCM message is sent to affected responders any time a fellow responder's status is updated from any device. When each device receives the GCM message, it automatically updates the Response Details view and either adds or updates the entry in the list of responders. The message includes the necessary data payload, so the application does not need to first connect to the web application to obtain the status information. All of this is done in the background without further user interaction. If the user happens to be viewing the Response Details at the time, the user interface automatically updates without a manual refresh. Similarly, if the responder updates his

or her own status via a different device, yet another GCM message is sent to all of the devices registered to that user to automatically update the Incident Details view and display the proper buttons.

Figure 5.5: Response Details view displays a list of responders and their current status

The user can also open the application manually. The first time the application is opened, the user is prompted to choose which Google Account they would like to authenticate with. If there is not one available, the system provides a sign up process to create a free account. Once an account has been chosen, the user must click the Registration button. Upon clicking this button, the application calls the GCM service to create a unique device registration id, which is then persisted to the web application data store as well as locally within the Android device. After registering the Google Account and device, a system administrator must assign the account to a department through the web client application. While this step could be handled by the user within the native

application, requiring an administrator to first assign new users through the web application adds yet another layer of security, stopping users from randomly installing the application and receiving incidents and updating responder statuses fraudulently.

Once the user and device are setup, the Main Menu loads any currently active incidents. Clicking on an incident within the list opens the Incident Details view. The user interface of the view hides or shows buttons depending upon the current status of the responder. For instance, if the responder had previously acknowledged from this device or any other, the En-route button would be visible. Lastly, an interactive Google Map fragment is displayed, zoomed in to location of the call.

Upon acknowledging an incident, the application reads the user's current GPS coordinates, if available, and calls the Google Maps API to estimate the time of arrival at the responder's station and updates the data store accordingly. The application takes advantage of Android's geo-fencing capabilities by creating a virtual boundary around the department and around the scene. The application automatically detects when a responder arrives at the station, leaves the station to go en-route to the scene, arrives on scene, leaves the scene, and then arrives back at the station. Each time a geo-fence boundary is hit, the application automatically updates the data store via the web application in the background. All other devices are then sent notifications and their user interfaces are updated and synchronized.

Much like making calls from the JavaScript client to the web application APIs, calls from the Android client are authenticated by the application itself and by passing the current user of the application. In the Google API console, a client ID is generated after

entering the SHA1 fingerprint of the key used to secure the Android application. This stops other Android applications from accessing the API endpoints.

CHAPTER 6: EVALUATION

6.1 General Evaluation

The system was tested with the Plymouth Fire Department, a volunteer department located in Sheboygan County, Wisconsin. The test group included Android phones running the native application, iPhones receiving the text message alert, and standard cellular phones also receiving the standard text message alert. I was able to share my Google Glass with various members so they could experience the wearable technology and mimic real life scenarios while interacting with this system.

At the time of this writing, an interface was being created with a national leader in computer aided dispatching (CAD), as well as a leader in emergency reporting, which will allow for real-time integration with Sheboygan County's dispatch center as well as with the fire department's reporting system. While this interface was not implemented in time for evaluation, the rest of the system was fully functional, allowing for the fire department to run multiple mock trials as if they were real emergency incidents.

The first goal of the overall evaluation of the system was to gauge the members' responsiveness to using such a system. After explaining the basics of the system including the native Android application, usage of text messaging, and the administrative pages, the members unanimously found the system beneficial and easy enough to use in emergency situations.

Responders of all technical levels found the user interface straightforward and easy to interact with, specifically citing easy to click large and obvious buttons. Although iPhone users were disappointed a native application was not available for them,

the inclusion of the link to open a native mapping application from the text message proved sufficient. While the mapping feature is unavailable on standard cellular devices, the ability to simply hit the reply button to acknowledge the call and update subsequent statuses was impressive enough. The only notable functionality missing for non-Android users is the ability to view a list of active responders and what their statuses are.

The idea of hanging a large screen in the apparatus bay or gear room, where the personal protective equipment is housed, was also unanimously supported. This would allow anyone at the station to quickly view the status of all active responders and make the necessary decisions based on that information. It was also agreed that the web application's Response Details page should be used with the laptops that are kept in a few of the apparatus so commanders could keep track of the overall response while en-route and then throughout the incident once on scene.

When members tried on Google Glass, they immediately understood how this wearable device not only decreases the amount of time it takes to view and acknowledge a call, making it extremely convenient, but also how much safer it is. For instance, many responders keep their mobile devices stowed in their pockets. When an incident would be dispatched and the device alerted, users would not know what the alert was for until they dug it out of their pocket and unlocked it. With Google Glass, an audible alert is played, and a single tap to the trackpad, or a slight nod of the head, displays the card containing the dispatch information. It takes just two more taps of the trackpad to acknowledge the call and update subsequent statuses. Although this only saves approximately 10 seconds versus using the native Android device, every second counts during the response to an emergency. Many members also mentioned they thought it was

safer, as they felt it would be acceptable to quickly glance up at the card while doing other activities where they would not feel safe digging out their phone and unlocking it to view the incident information. The limitation on Google Glass is the same as non-Android devices, where the user can not view a list of active responders.

6.2 Benefits of an Automated Redundant Alerting System

The first opportunity to measure the effectiveness of the application is during the dispatch process. While this cannot be empirically tested as the interface with the computer aided dispatch (CAD) was not completed, information from the company provided the necessary information to make reliable estimations. Dispatchers generally ask for the location of the emergency first, followed by a description. Once these two components are entered into the CAD system, it is able to identify which department should respond. At this point, the interface between this system and the CAD system would be initiated and this system would immediately send the notifications to the impacted responders. This would all be done automatically while the dispatcher is still on the phone gathering further information or beginning to activate the primary radio alerting system.

In order to dispatch over the radio alerting system, the dispatcher must first send out the specific tones for a department, which generally lasts about 10 seconds, then relay the address and description of the emergency, which usually lasts at least another 10 seconds. By having the CAD system automatically notify this system, it would save at least 25-30 seconds on every call. If the dispatcher has to first alert police before dispatching fire or emergency medical services, using this system could save minutes.

With the exponential growth of fire over time, 30 seconds can be the difference between a couch being on fire and the entire room being fully engulfed.

This system provides an invaluable secondary feature, being that it would create a backup alerting system, providing redundancy that so few departments have today. Primary alerting systems are typically built on proprietary and localized radio systems. There are times when the alerting system may be unreliable, such as when a signal is weak due to the location of the responder's pager, which is often times the case deep inside an industrial building, or because the responder is simply out of geographical range. Furthermore, the radio system's infrastructure may fail or be damaged by a storm or malicious attack. In such an event, the primary alerting system could be down for a subset of an area or an entire region. Departments must then rely on someone receiving a phone call from the dispatch center, and then contacting other members to notify them of the incident. Implementing this system would provide a parallel alerting method that would act as the primary alerting system if the main system was unavailable or responders were out of its range.

6.3 Effectiveness in Reducing Response Times

Mock incidents were created using real life scenarios as models in order to test the application as if it were a true emergency incident. While there is no substitution for testing in true, real life emergencies, there is a slight upside to mock incident evaluation. In this controlled environment, we are able to more calmly talk through the decisions that were made, and even consider changing procedures based on the new information that this system provides.

The keys to a safe and efficient response are: not leaving the station with room in the apparatus just as more members are arriving, not waiting at the station for members who are not responding, and recognizing as soon as possible when it is necessary to request additional resources and departments to respond for mutual aid. It is important to first understand the standards of a department in order to correctly calculate the benefits of the system. For the test department, standard operating guidelines generally require the first out apparatus to wait until it is filled to capacity before leaving, which is generally 6 responders, depending on the apparatus and incident type. Unfortunately, as a volunteer department, no one knows who is available or how far away they are. Thankfully, this department rarely has trouble filling the first trucks shortly after the dispatch.

Control Variables		
Test Setup	Department is dispatched to a kitchen fire. Volunteers respond to the fire station. Record when each volunteer is ready to leave on a truck. Record when the first two trucks leave the station. Record when any mutual aid is called.	
Environment	Plymouth Fire Department, Sheboygan County, WI	
Equipment	<u>Without System Implemented</u> Pagers Radios	<u>With System Implemented</u> Google Glass Android smartphones iPhones Basic cellular phones Pagers Radios

Table 6.1: Control variables of the testing scenarios

Each scenario starts with the department being dispatched to a report of a kitchen fire within the city limits. In scenario 1, the first three responders arrived at the station and were dressed and ready within 2 minutes and 30 seconds from the time of call. The fourth responder first arrived at the station near the 2 minutes and 30 seconds mark, a fifth near the 3 minute mark, followed by a sixth responder at 3 minutes and 30 seconds from the time of call. Many members agreed that it would be a tough judgment call whether to leave with only three responders, or wait until the fourth, fifth, or sixth member was dressed and ready to go. After asking that question, the members were informed that in this scenario, 5 more responders arrived at the station between 3 minutes and 30 seconds and 5 minutes from the initial time of call. Once the members were aware that a total of 7 more responders arrived at that station between 3 and 5 minutes from the time of call, they stated they would have left once they had a crew of 4, knowing that at least one more full truck would be en-route within minutes to provide a backup crew. Had the system been in place, informing the initial responders of whom else was responding and when, it could have decreased the response time by more than 2 minutes.

Manipulated Variables **Scenario 1** 11 responders ready to leave within 6:30 from time of dispatch			
Time Since Dispatch	**Firefighters Ready**	**Results Without System Implemented**	**Results With System Implemented**
<0:30			All available firefighters have acknowledged
3:30	4		First truck en-route
4:30	5		
5:30	6	First truck en-route	Second truck en-route
6:30	11	Second truck en-route	

Table 6.2: Scenario 1 - First truck en-route 2 minutes faster with system implemented

A second scenario was executed with the same setup. The department was dispatched to a kitchen fire and 3 responders arrived and were in the apparatus, ready to leave within 2 minutes and 30 seconds. But in this scenario, no other responders were on their way to the station. Special events such as weddings, large training conferences, or even hunting seasons can significantly drain volunteer departments of their available responders. Members agreed that since they saw no other responders coming into the station, they would leave and call for mutual aid from surrounding departments. It was pointed out that the call for mutual aid took 3 to 4 minutes to occur. Had the system been in place, members agreed that the call for mutual aid could have been initiated within 1 minute from time of call, decreasing the overall response time of all departments by 3 minutes.

Manipulated Variables

Scenario 2
3 responders ready to leave within 2:30 from time of dispatch
No other responders are available

Time Since Dispatch	Firefighters Ready	Results Without System Implemented	Results With System Implemented
<0:30			All available firefighters have acknowledged
0:30-0:45			Call for mutual aid
2:30	3		First truck en-route
3:00-4:00		First truck en-route Call for mutual aid	

Table 6.3: Scenario 2 - First truck en-route faster and mutual aid called 3 minutes faster

A last scenario was given using the same incident setup as scenario 2. However, this time, a dozen more responders arrived together 6 minutes after the call. Members

agreed that it would yet again be a tough judgment call as to when to leave with only 3 members on the apparatus. Should you wait 15 seconds to see if anyone else is arriving? 45 seconds? 3 minutes? Many stated they would again have called for mutual aid at this point due to a lack of man power. However, what they didn't know was that enough responders were already on their way, but were further away than usual. Again, the members agreed that if the system were in place, they would have left with 3 members, reducing the response time by minutes, and also would not have called for the unnecessary mutual aid.

Manipulated Variables

Scenario 3
3 responders ready to leave within 2:30 from time of dispatch
12 more responders ready to leave 6:00 after time of call

Time Since Dispatch	Firefighters Ready	Results Without System Implemented	Results With System Implemented
<0:30			All available firefighters have acknowledged
2:30	3		First truck en-route
3:00-5:00		First truck en-route Call for mutual aid	
6:00	15	Second truck en-route	Second truck en-route

Table 6.4: Scenario 3 - First truck en-route faster and no unnecessary request for mutual aid

While the above scenarios depict real-life instances of likely emergency incidents, not every incident will see these significant reductions in response times. Moreover, the success of such a system is wholly dependent on the members using the system. Every active member of the test department had a device capable of working with the system, but that is certainly not the case with every department. Also, the responders must want

to use the system. Again, the test department was very responsive to the idea of this additional requirement, especially after discussing the benefits such information can provide. Lastly, the usage of the system must become a habit, as even if the responders want to use the system, it can take quite some time to alter a routine. Luckily, emergency responders are trained to continually improve and adopt new procedures that when practiced, become second nature.

CHAPTER 7: FUTURE WORK AND CONCLUSION

7.1 Enhancements

This system barely scratches the surface of what a full featured emergency response system is capable of. While this system focuses on the response aspect of emergency management, there are many enhancements and additions that could be implemented for this focus alone. Aside from response management, an all-encompassing emergency response system would include the ability to create, update, and view pre-planned procedures and imagery, such as building floor plans, hydrant locations, apparatus placement, natural gas line locations, and the location and safe handling instructions of on-site hazardous materials. It would also provide real-time tracking of responders and apparatus on an interactive map, display biological vital signs of responders such as breathing rate, pulse, and oxygenation levels, as well as environmental variables such as the amount of air remaining in a self-contained breathing apparatus (SCBA), atmospheric oxygen levels, and amount of noxious or explosive gases present. When wearable hardware becomes capable of withstanding high temperatures and other extreme conditions, systems could incorporate live video feeds of responders within a building and thermal imaging cameras so remote viewers could help locate hot spots and achieve a real-time understanding of the environment responders are in.

Aside from the ideas enumerated above, suggestions for improving this system were relatively minor. A native iOS app that mimicked the capabilities of the Android app topped the list. Ideas to incorporate more display information into the wearable platform, Google Glass in this case, were discussed. The intention of the wearable device was to provide a fast and always-present way to update a responder's current status, not

display lists of information that would require copious amounts of scrolling in order to review. However, adding the ability to scroll through a list of active responders and their statuses would be a relatively easy addition, regardless of its debatable merit.

Another common suggestion was to add more options and customization for responding to requests for emergency medical services (EMS). This facet of emergency management was intentionally left out as it was outside the scope of this project. The vast number of system requirements for managing patient care necessitates a separate and dedicated application, requiring extensive research and development. However, slight customization is possible to allow this system to include EMS specific response management.

7.2 Revisions

A few aspects of the technical implementation warrant further review. One serious issue is the lack of reliability when communicating via SMS. There is a high likelihood of a non-Android phone not receiving a message or the system not receiving the reply from the user in a timely matter. This system hinges on the timely transmission of accurate data. As long as the device has signal, a call is much more reliable in time sensitive situations. A potential solution is to allow responders to acknowledge incidents by dialing a phone number, which would automatically detect the phone number of the caller and update that responder as acknowledging the latest incident. The caller simply hangs up just after connecting, or allows the phone service to end the call.

The use of a relational database is another implementation choice that deserves further consideration. While this system currently utilizes a cloud based relational

database allowing for complex relational queries, the trade-off is the inherit performance limitations, especially when scaling, compared to a NoSQL data store. If the system were to be released to a large number of regions and ran from the same instance, serious consideration should be given to switching the data structures to utilize a NoSQL data store allowing for extreme scalability.

7.3 Conclusion

The field of emergency response and management is ripe for innovation and technological advancement. As we continue to pursue ubiquitous computing, we must remember to accept the stepping stones necessary to guide us there. Wearable computing, sensor integration, and device interoperability are strong advancements in technology that will continue to shape our universe. Through advanced communication and information sharing, this system illustrates a powerful and novel way to better protect and save the lives and property of our fellow citizens by reducing response times to emergency incidents. With every piece of hardware and software that begins to harmoniously communicate, such as the applications and devices within this system, we move closer to weaving technology into everyday life and creating a world where technology lives within a human environment rather than humans living within a technological environment.

BIBLIOGRAPHY

[1] Mark Weiser. “The computer for the 21st century”. Scientific American, Vol. 265, No. 3. Sept 1991.

[2] Karter, MJ, Jr.; Stein, GP. "US Fire Department Profile 2012". National Fire Protection Association Fire Analysis and Research Division. Oct 2013.

[3] NFPA. "Fire Department Calls." National Fire Protection Association. http://www.nfpa.org/research/reports-and-statistics/the-fire-service/fire-department-calls/fire-department-calls.

[4] United States Fire Administration. “Voice Radio Communications Guide for the Fire Service”. FEMA. Oct 2008.

[5] W. D. Davis, M. K. Donnelly, M. J. Selepak. (2006). *Testing of Portable Radios in a Fire Fighting Environment*. National Institute of Standards and Technology, Building and Fire Research Laboratory.

[6] Poslad, Stefan. (2009). *Ubiquitous Computing Smart Devices, Smart Environments and Smart Interaction*. Wiley. ISBN 978-0-470-03560-3.

[7] Alberth, JR., William P. “Coupling an electronic skin tattoo to a mobile communication device”. US Patent Application Number 20130297301. 7 Nov 2013.

[8] International Telecommunication Union. “The World in 2013 ICT Facts and Figures”. Information and Communication Technologies Data and Statistics Division. Feb 2013.

[9] CTIA. “Annual Year-End 2012 Top-Line Survey Results”. The Wireless Association. June 2013.

[10] Grier, Robin. "Interoperability Solutions". Interoperability. Catalyst Communications. 28 May 2011.

[11] Allen, D. K., Karanasios, S., Norman, A. “Information sharing and interoperability: the case of major incident management.” European Journal of Information Systems. Aug 2008

[12] IamResponding. “Emergency Responder Reply System”. www.iamresponding.com.

[13] CSN Philly. “New App to Improve Firefighting”. http://www.csnphilly.com/article/new-app-improve-firefighting. 19 Nov 2013.

[14] CNN. “Fighting fires with the help of Google Glass”. http://www.cnn.com/2014/01/21/tech/innovation/google-glass-firefighter/. 21 Jan 2014.

[15] Globe. “WASP: Wearable Advanced Sensor Platform”. http://www.globeturnoutgear.com/innovations/wasp.

[16] Android Wear. “Information that moves with you”. http://developer.android.com/wear/index.html.

[17] Canalys. “Android on 80% of smart phones shipped in 2013”. http://www.canalys.com/newsroom/android-80-smart-phones-shipped-2013. 30 Jan 2014

[18] Google App Engine. “Platform as a Service”. https://developers.google.com/appengine/. 7 Feb 2014.

[19] Google Cloud SQL. “Relational Databases in Google’s Cloud”. https://developers.google.com/cloud-sql/.

[20] JSP. “JavaServer Pages Technology”. http://www.oracle.com/technetwork/java/jsp-138432.html.

[21] jQuery. “Write less, do more.” http://www.jquery.com/.

[22] Bootstrap. “The most popular front-end framework for developing responsive, mobile first projects on the web”. http://www.getbootstrap.com/.

[23] Fielding, Roy Thomas. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD dissertation. University of California, Irvine.

[24] Garrett, Jesse James. “Ajax: A New Approach to Web Applications” https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php. 8 Feb 2005.

[25] Google Mirror API. “Google Mirror API Overview”. https://developers.google.com/glass/develop/mirror/index. 19 Nov 2013.

[26] Google Email API. “Mail Java API Overview”. https://developers.google.com/appengine/docs/java/mail/. 27 Feb 2014.

[27] Google Cloud Messaging. “Google Cloud Messaging Overview”. http://developer.android.com/google/gcm/gcm.html.