OmniView

Senior Design Final Report

Team May 14-22

Authors: Charles Patterson

Nick Schulze, Derek Petersen, Chris Tedford

Table of Contents:

- Introduction
- Project Design
  - Overview
  - Block Diagram
  - Google Glass Application Module
  - Web Server Module
  - MIRAGE Module
- Implementation of Project
  - Google Glass Application Module
  - Web Server Module
  - MIRAGE Module
- Testing Process
  - Google Glass Application Module
  - Web Server Module
  - MIRAGE Module
- Conclusion
- Appendix I: Operation Manual
  - Overview
  - Setup Google Glass Application Module
  - Using Google Glass Application
  - Setup Web Server Module
  - Setup MIRAGE Module
  - Notes
- Appendix II: Prototypes
  - Audio Transfer App
  - Google Play Services Map
- Appendix III: Other Considerations
  - World View Map

I. Introduction

For our senior design project, we were given a Google Glass and access to the MIRAGE tracking system. We were tasked with building an application for the Google Glass that acts as a heads up display and works in conjunction with the MIRAGE tracking system to track and display other users.

Throughout the course of development, secondary goals began to take shape. These included: developing an API so that other devices could use our tracking and display system; extending the tracking outside the doors of the mirage; allowing the user to upload custom maps to the application; fostering communication from one device to another; getting the system working from multiple devices; and the implementation of an overarching 'command center' website.

Our main focus throughout the course of the year was the Google Glass. We ended up creating an application that would interface with the MIRAGE tracking system to generate a map of the environment that displayed other users also in the MIRAGE. Additionally, we were able to use the Google Glass's voice-to-text capabilities to send audio to the main server and other users on the system. When the MIRAGE tracking system wasn't available, we were able to configure the Glass application so that it fell back to using its GPS capabilities.

To store the data being generated by the Glass, we set up a web application online that was backed by a database that could store the data. We were then able to write both POST and GET endpoints for the Glass application to use so that it could both retrieve other users' locations and send its current location. Having all the data available was incredibly conducive to a sort of 'command center' application, so we ended up extending our REST API into a full fledged website that could act as the brains of the operation.

Realizing we could extend these services and because of the necessity to test our mapping application, we decided to create applications for both Android and iPhone. The Android application ended up being a very close mirror of the Glass application as they are written in the same language. The iPhone application ended up having some different functionality, but the core Glass functionality of tracking, displaying, and communicating remained the same.

In order to accomplish everything we made use of many different resources. We used the MIRAGE system and its various software libraries to process the data packets. For the MIRAGe web application we used a php web framework to set up a backend that could be more granular than GPS. For the World View Web Application we used the Play web framework. Throughout the development of both web applications we made heavy use of Google's Maps API's as well as general javascript libraries. Both the Glass and Android applications were written in Java using Android Development Tools. The iPhone application was written in Objective-C.

II. Project Design

Overview

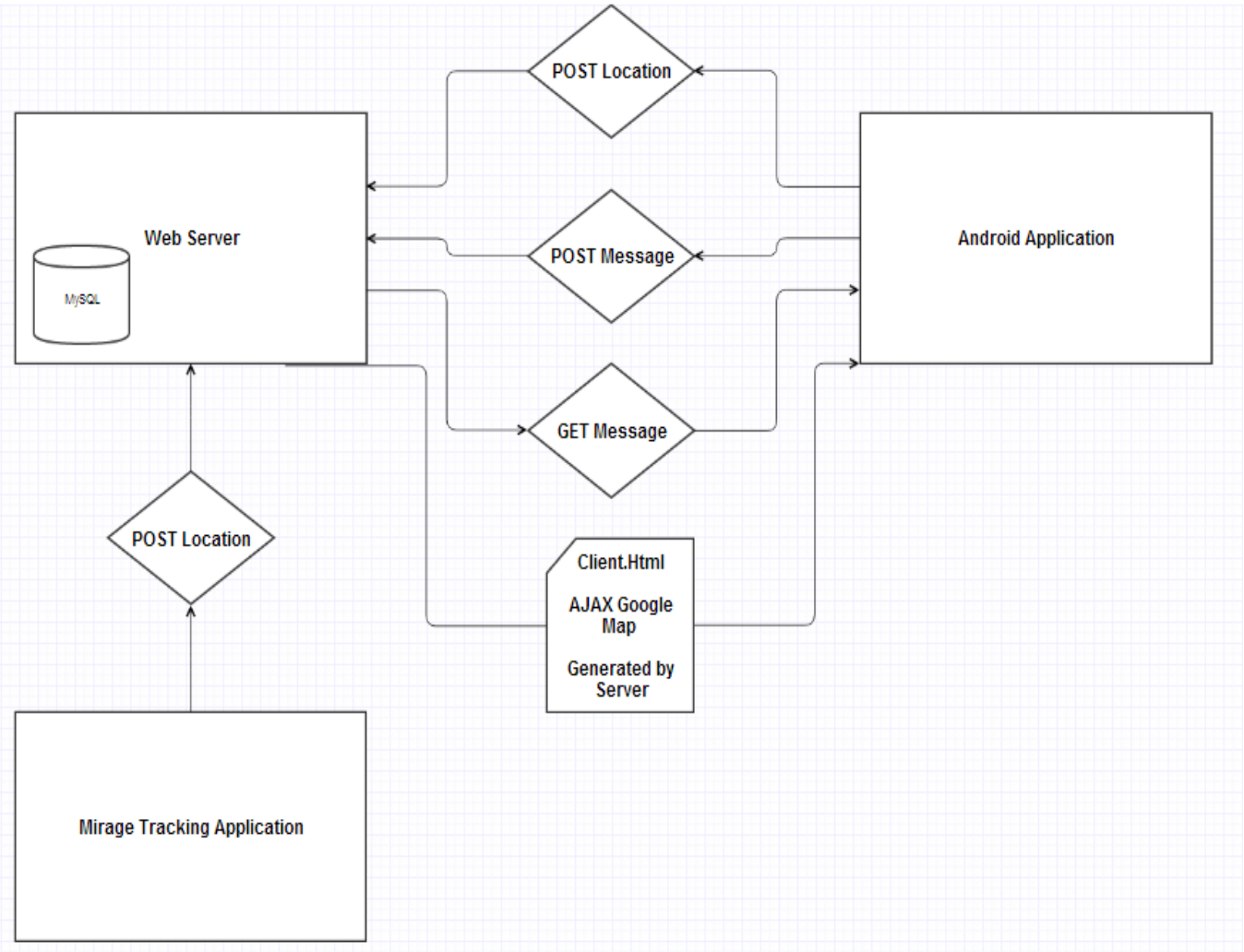
Omniview was designed with modularization and scalability in mind. To make our product work across many platforms and with a variety of programming languages we built the backbone of the project on a web framework. The web framework allows us to scale our project to be used with multiple Google Glass devices, Android devices, and iPhone devices. It also gives us a variety of access methods to our data, we are able to connect to the program through a Wi-Fi connection or through a mobile data connection. This means not every device has to be on the same network to work with our program.

One web server does the majority of the computing for the project. GPS and message data is collected in a variety of ways from the different sources, but it is all sent to our web server to be processed in the same way. This gives us uniform data that is easily accessible to each device after being processed. It also takes the computing load off of our devices.

There are currently two ways in which GPS data can be provided to the web server. The GPS data can either come from the connected devices built in GPS sensor or it can come from the MIRAGE tracking system. The MIRAGE tracking system gives us precise indoor location data while the built in sensors give us accurate outdoor location data. Combined, OmniView can be used in a multitude of places.

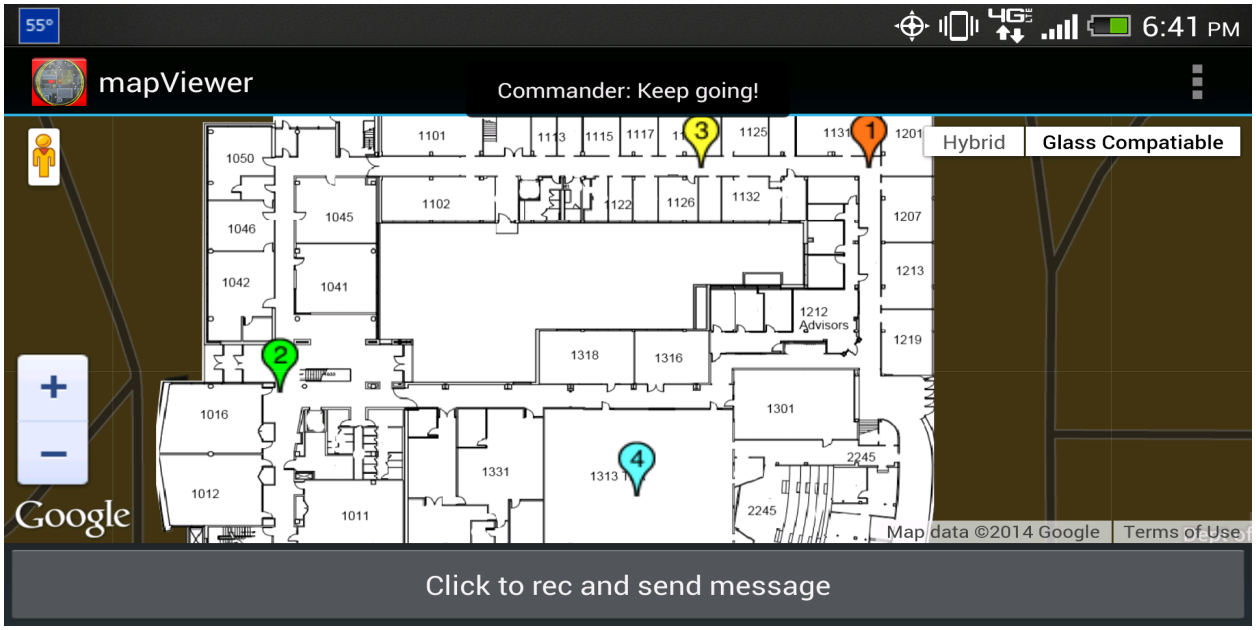
Block Diagram

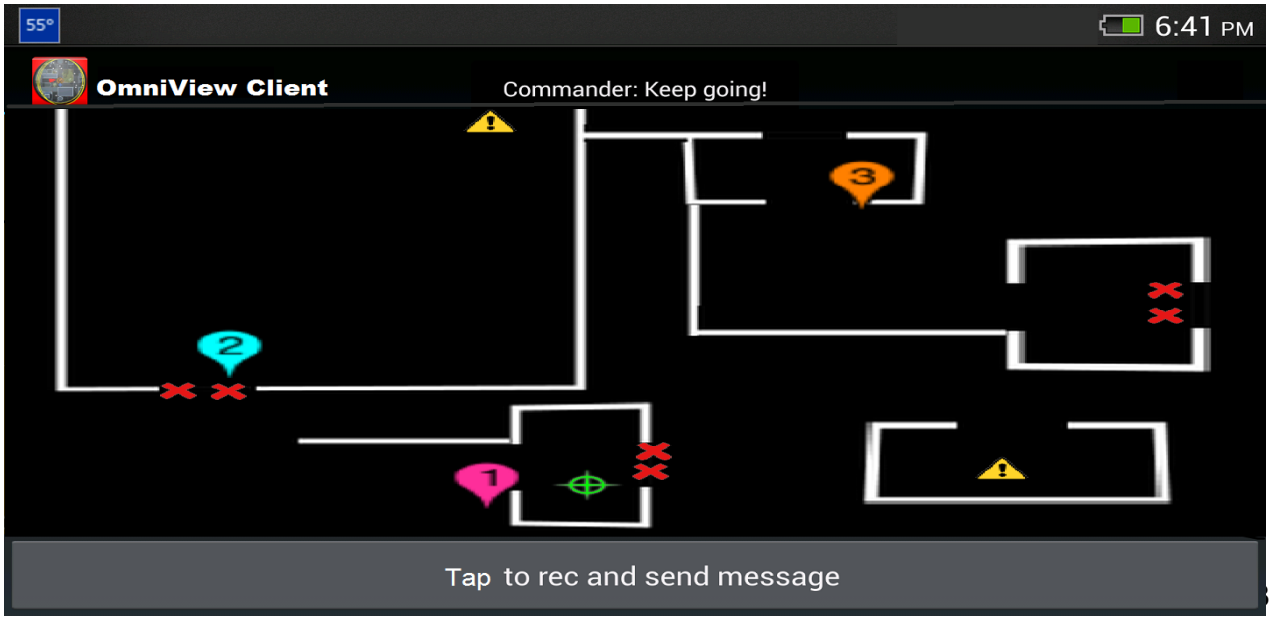
The modularization of OmniView gives us some distinct advantages when it comes to troubleshooting and testing as well as expansion and change. OmniView consists of three majors modules with small scripts connecting each module together. The three modules are a web server, a mobile application (Google Glass, Android, iPhone), and a MIRAGE tracking module. Each module sends data to the other modules through HTML request methods. These HTML requests methods let the mobile applications post their GPS data to the server and let the MIRAGE tracking module post it's data to the server. The mobile applications also retrieve messages from the server and send messages to the server through HTML requests.



Google Glass Application Module

The Google Glass application handles standard user interactions with the system on a non-administrative level. The application shows an Android WebView that renders the client web module. The web module shows a simple Google Map with all active users' locations represented by different map pins. This allows the user to view his/her location along with the other users currently using the system. The app also handles displaying new text messages from the server. The client also allows the user to create and upload new text messages utilizing google glass's voice-to-text conversion. The Glass module gathers its GPS location providers and polls all of them to get the most accurate coordinates possible before updating the server with its location data.

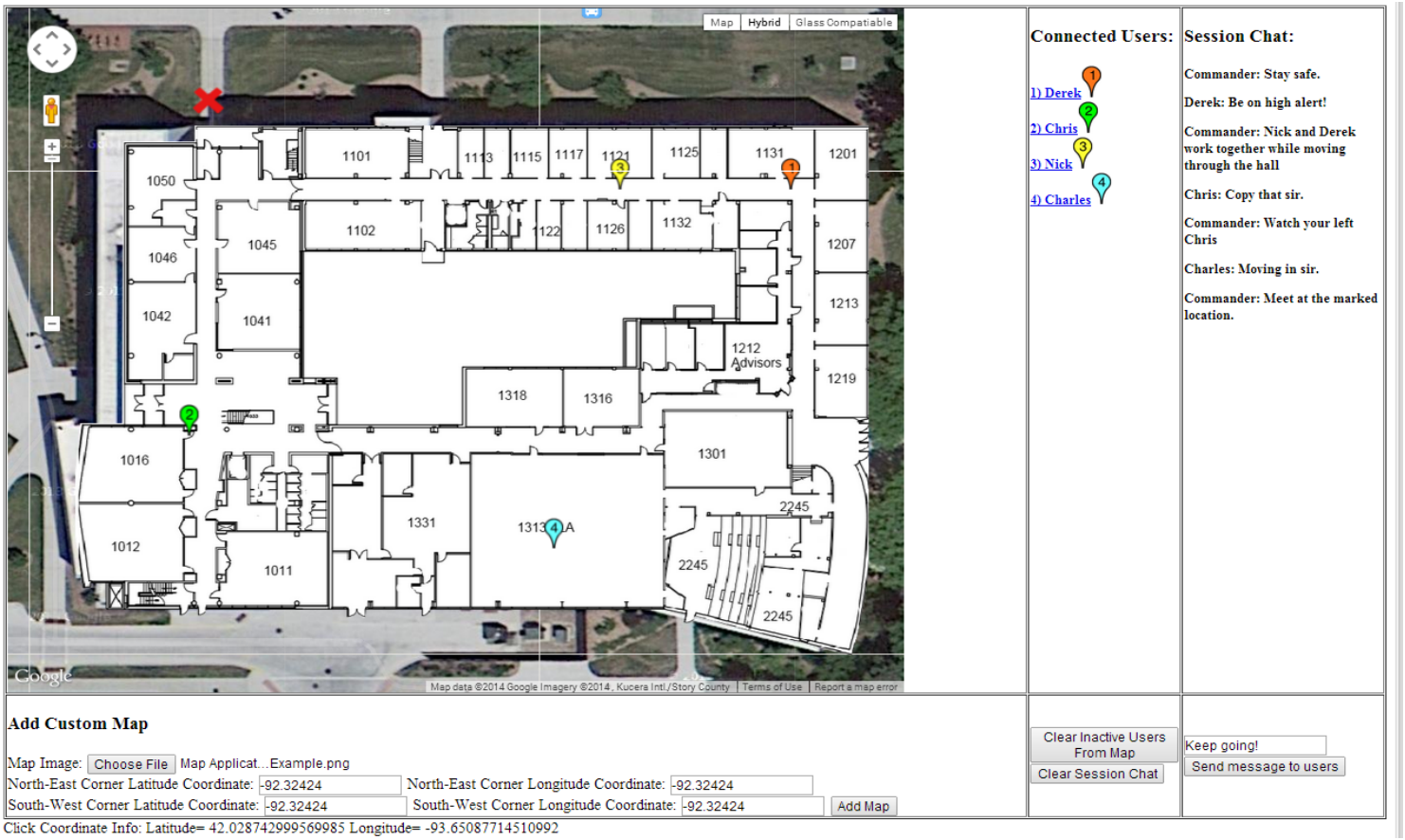




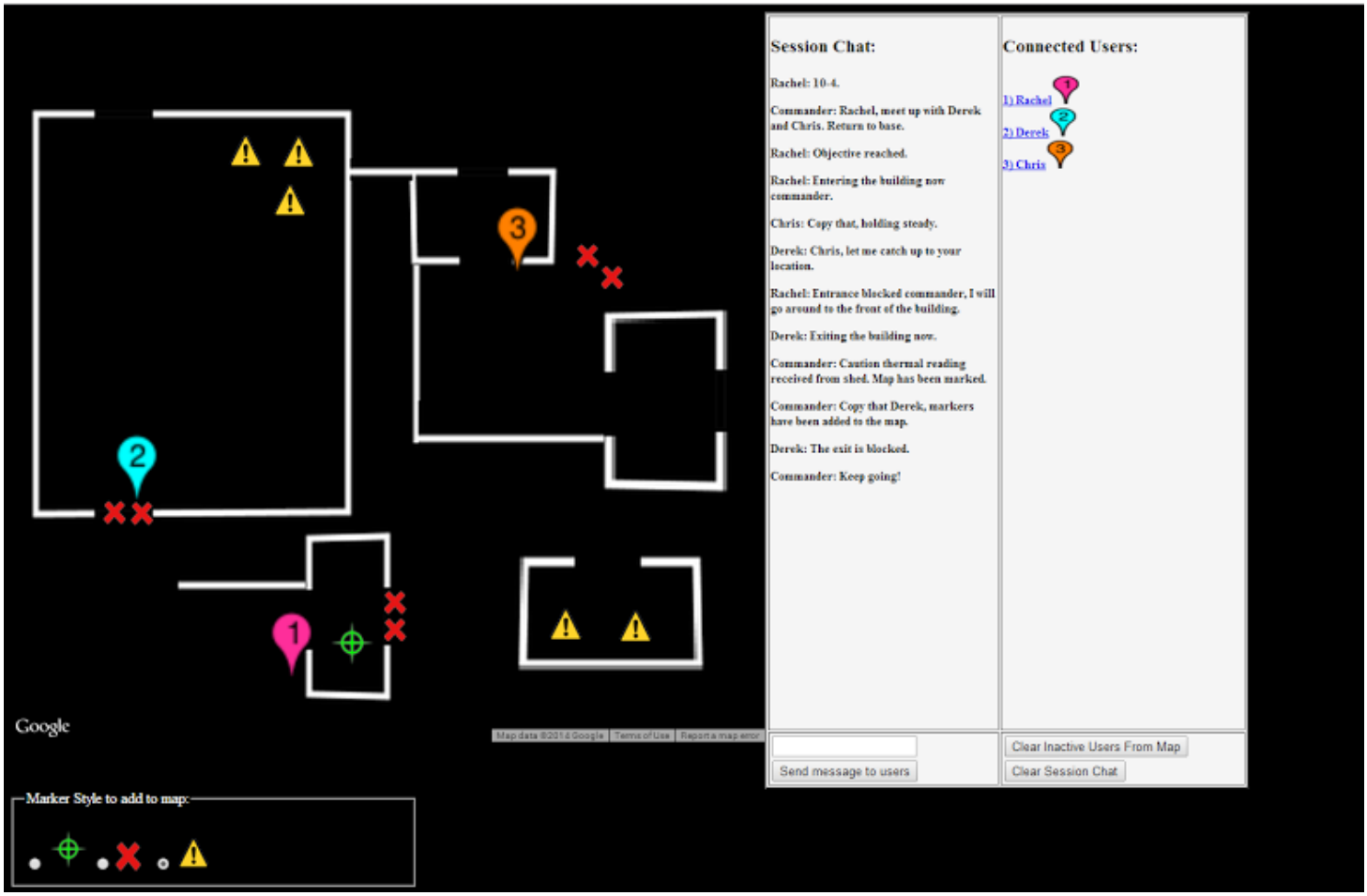
Web Server Module

The web server serves three main functionalities. It runs a Javascript module that builds a map using the Google Maps service. The map includes custom ground overlays, user markers, and user information. It renders two maps, a server view and a client view. The second major functionality is to provide communication from the server to the clients and vice versa. It also receives data communication from the MIRAGE and mobile applications. Finally, the web server stores all of the data and information for OmniView in a MySQL database.

The web server acts as the command center for the OmniView application. A “commander” has the control to add new custom drawn maps to the OmniView for all clients to see. The commander can also send messages to users.



The “commander” or user of the web server can add custom markers to the map to notify users of an objective location, caution areas, or blocked off locations. The server has two modes it can run in, a full world map view and a MIRAGE map view. The MIRAGE map view takes advantage of the precise indoor MIRAGE GPS locations, but is limited to a single map area of 40’ by 40’. This module is used for Army training missions in a closed environment.



MIRAGE Module

The MIRAGE module is a program that takes data from the tracking system, converts it to our data format, and then sends it to the server application.

It works by integrating with MetaTracker. MetaTracker receives coordinates of objects within the room in a cartesian x,y,z format in units of feet. In order to integrate with our system, we convert those units to GPS coordinates based on a simple mathematical formula. Once those units are in GPS coordinates, we use libcurl to assemble and send a POST request to the server.

III. Implementation of Project

Google Glass Application Module

The Glass Application Module is implemented as an Android Java project. The Android layout is simply a WebView component filling the entire screen. The WebView points to the URL of the web client module. Calls to the web client module are simply made as Javascript calls directly to the WebView.

Text messages located on the server, meant for the Glass Application module are read from the server and presented to the user using the Google Android Toast library. This displays the message in a pleasant message box at the top of the screen for five seconds. The messages are pulled from the server by making an HTTP POST request to the Web Server Module.

The Glass Application Module is required to provide location updates from its GPS device and submit them into the server. When the app submits its coordinate data, it's also required to include a user ID based on the user's name (entered in via voice recognition at the initialization of the app) and the device's model number. Google Glass, along with other Android devices, allow the GPS device to be extended by other attached GPS devices. The implementation is designed to take advantage of this by polling all attached devices for location data and determining the best possible update before sending it to the server.

Web Server Module

The web server uses the Google Maps JavaScript API v3 to render the map. PHP scripts read user's location data and messages from the MySQL database, send them to XML files, and then an AJAX function on the web server adds them to the Google Map. This AJAX function polls the XML files every 500ms to check for updates on user locations and messages and then updates the object on the map without needing to refresh the page.

```
▼<markers>
   <marker lat="42.02862" lng="-93.65137" html="Glass Id: 3awe5td" label="Chris"/>
   <marker lat="42.02836" lng="-93.65138" html="Glass Id: 2vad4fv" label="Derek"/>
   <marker lat="42.02862" lng="-93.65057" html="Glass Id: 3awe5td" label="Charles"/>
   <marker lat="42.02824" lng="-93.65098" html="Glass Id: 2vad4fv" label="Sasha"/>
 </markers>
```

The web server communicates with both the MIRAGE and the Google Glass application through the HTTP request methods POST and GET. The HTTP request methods are handled by PHP scripts running on the server. The Google Glass application and MIRAGE both send updated GPS data to the PHP script hosted on the web server. If the data is from a new source the PHP

script updates a the MySQL database table, location, with a new entry, if it has already received data from that source it will instead update the MySQL database entry. Messages from the Google Glass application also work in a similar fashion, a new message will be received by a PHP script on the server which will then send it to the Messages table in the database.

MIRAGE Module

The MIRAGE module is a c++ program using code from the existing metatracker software in the MIRAGE. The program isn't limited to any platform and just needs to be run on a computer that is connected to Iowa State's network.

It also communicates with the server application by using POST requests. This is done using libcurl to assemble and send the requests. It creates and sends a request every time it is notified of an object existing in the tracked area.

IV. Testing Process

Google Glass Application Module

Goal / Requirement	Test	Results
Messages are displayed to the App user within 1000ms after being posted to the server.	Hand-time from the amount of time between sending the message to the server and receiving the message on the glass client.	Average time: ~700ms
Location data is updated on the glass view within 600ms of appearing on the server.	Hand-time the amount of time between sending the message to the server and receiving the message on the glass client.	Average time: ~300ms

Web Server Module

Goal / Requirement	Test	Results
Each user's location updates within 500ms after they have moved.	Record the average time it takes for a users location to refresh.	Average time: 520 ms.

Project works with up to 10 users simultaneously	How many users can be on the system at one time without causing delays	Project can handle at least 12 if not more simultaneous users.
New messages enter the system within 500ms after they have been sent to the system.	Record the average time a it takes a message to be displayed on the system after being sent.	Average time: 649 ms.
Messages are sent to users within 500ms after they are entered into the server.	Record the average time a it takes a message to be displayed on the system after being sent.	Average time: 531 ms.
Web server is protected from cross site scripting attacks.	Test each input section against an array of popular XSS attacks and record the results.	Passed 15 tests. 4 tests caused errors in webpage, but were fixed by clearing session chat. Not a threat.
Web server is protected from SQL injection attacks.	Test each input section for SQL injection attacks and verify integrity of the data in the MySQL database.	Passed 10 tests at input locations. No SQL injection attacks made it to the database or through the system.

MIRAGE Module

Goal / Requirement	**Test**	**Results**
Data is retrieved from the MIRAGE	Verify we were able to track helmets in the room and manipulate data in our team's code	Location data is accurate, but sometimes difficult outside of the center of the room
Data is transmitted to the Server Application	Verify that we can make post requests in a reasonable amount of time	We can make about 10 post requests per second

-

V. Conclusion

OmniView is a great example of the potential that Google Glass and wearable technology have. It is a unique application that showcases a practical example of how it may be used in training simulations. Users new to the application will be able to understand it quickly because of the concept's prevailing appearance in modern video games.

Making a portable web-based infrastructure immediately made our project extendable to any client that was web enabled. We were able to view maps on phones, tablets, and computers in addition to Glass. The end product is therefore useful even to users that do not have the expensive Glass hardware.

We were able to witness limitations of the device in its current form. It's battery life is inadequate for this application of the screen is to be on for a long period of time or if video is being recorded constantly. The device also lacks libraries available to other android devices; this limitation in particular slowed our development and planning process. With these limitations in mind it was still exciting to see the project come together and work in the end.

Appendix I: Operation Manual

Overview

OmniView is easy to operate. Each module needs to be turned on or active and then they are set up to automatically sync together. Follow the instructions below to set up the three modules and then watch as they connect and data begins to flow through the system.

Setup Google Glass Application Module

First of all, if you're using Glass and not another Android device, you'll need to make sure that the Google Glass Launcher is installed on the device. Then all you need to do is install the android app to the device. The .apk will be located somewhere on the VRAC's server. Once the app starts up, any setup will be handled by the app itself.

Google Glass Launcher: https://github.com/justindriggers/Glass-Launcher

Using Google Glass Application

The application features a set of gesture commands through the touchpad located on the right side (when wearing) of the glass. The implemented commands and their functionality are listed below.

- Single Tap: Center map on yourself
- Long Tap: Record and send message to server
- Swipe up/down: Scroll to other users and center on them
- Swipe forward/backward: Zoom in / out

Setup Web Server Module

The web server is hosted on Dr. Gilbert's public Iowa State web server. It can be accessed at the following URL: http://public.vrac.iastate.edu/~gilbert/mirageglass/

There are two pages that run the control center and display the server map for the program. One page is for use inside the MIRAGE system and the other is for use with GPS data from the devices outside of the MIRAGE system. The two server pages look distinctly different and clearly serve different purposes. Pick the page to use for your setup and follow the specific instructions for that page as seen below.

MIRAGE server
Link: http://public.vrac.iastate.edu/~gilbert/mirageglass/mirageserver.html

To change the Mirage Room image… The page uses a single ground overlay rendered from the image named mirageRoom1.png located on the server. To change the ground overlay delete the mirageRoom1.png file from the server and upload a new image named mirageRoom1.png. The program will automatically resize the image to fit within the calibrated MIRAGE environment.

To add markers to the map… Select the marker style you want to add and double click anywhere on the map. It will take around 500ms for the marker to propagate on the map as it needs time to be added to the database. Note: Only one green goal marker can be set up in the environment at a time to avoid confusion.

To clear the current session chat or any markers / users… Click the clear markers/ session chat button. This will clear the objects from the database. If the MIRAGE system is running users will reappear on the map in less than 500ms but the markers will be cleared.

Real world server
Link: http://public.vrac.iastate.edu/~gilbert/mirageglass/server.html

To add new ground overlay…Use the add custom map area at the bottom of the page. Select a map image from your file system and enter the GPS coordinates for the NE and SW corners.

These corners will be the edges of your map. If you double click anywhere on the map the coordinates of that location will be displayed below this interface. If there is an issue with the way the map looks you can go into the database and edit the map coordinates. Please refer to the notes below for accessing the system database.

To change the map type… There are three map type options; satellite, street, and glass compatible. You can select any of these by clicking on the words located at the upper right hand corner of the map.

To clear the current session chat or any markers / users… Click the clear markers/ session chat button. This will clear the objects from the database. If the MIRAGE system is running users will reappear on the map in less than 500ms but the markers will be cleared.

To add a goal marker to map… Double click anywhere on the map.

MIRAGE Module

Setting up:

In order to work, the Motion Analysis software must be running, as well as the cortex repeater. The match maker software must also be active. Once the environment is set up, our program can be run.

Our module:

Our module is called “GlassClient” in the copy of metatracker that we have. It can be compiled with the command:

make -f Makefile.glass

The syntax for running the client program is:

./GlassClient [delayTime]

where delayTime is an optional sleep added in seconds. The sleep value can be used to slow the update frequency. The program will run in a terminal window until it is stopped with ctrl + C.

Notes

How to access web server files

1) Open an FTP connector system (FileZilla, WinSCP)

2) Ftp to : ftp.vrac.iastate.edu, port 22

3) Sign in with VRAC credentials.

4) Browse to the following directory: home/users/gilbert/public_html

How to access the database

1) Go the the following PHPmyAdmin URL:
https://p3nlmysqladm002.secureserver.net/grid50/6689/index.php

2) Login with the following credentials

Username: location
Password:

Appendix II: Prototypes

Audio Transfer App

The Audio Transfer Application was our earliest prototype. It was a proof-of-concept prototype that proved out the ability to transfer large amounts of data between a server and a client in the form of audio files. The program was very simplistic and successfully proved that large amounts of data could be transferred between the server and client relatively quickly.

Google Play Services Map

The Android system has it's own Google Maps API currently in the second version. This API uses the Google Play Services to render the map locally on the device. Our initial idea was to use the Android Google Maps API on each device to build concurrent maps with every users data. The Google Maps API allowed us to render beautiful fully functional maps on our Android devices, but when we tried to run our application on the Google Glass device we found out that due to the early stages of the Glass development Google Play Services, the service that rendered the map, was not yet available on the Google Glass system.

We opened a service ticket with Google to get the issue fixed. Eventually Google responded to our ticket and informed us they were working on the issue, but by ⅔ of the way through the year the issue was still not addressed and we were running out of time.

At that point we decided we needed a new way to render the map that would still give us live updates and be fully functional. Our alternative was to render the map on a web server and then display the map on the Google Glass application. Thanks to Google's wonderful WebView API we were able to display the map on the Google Glass application but also run the JavaScript to make full use of the features of Google Maps.

Appendix III: Other Considerations

World View Map

Introduction

The World View Map was born when two of the developers built web applications over the weekend. The team liked both of them and decided to leverage them in different ways. The World View web application is built to have both messages and photos posted to it as well as locations. It is different and cool, but we didn't think it was necessary for Glass application to have these capabilities.

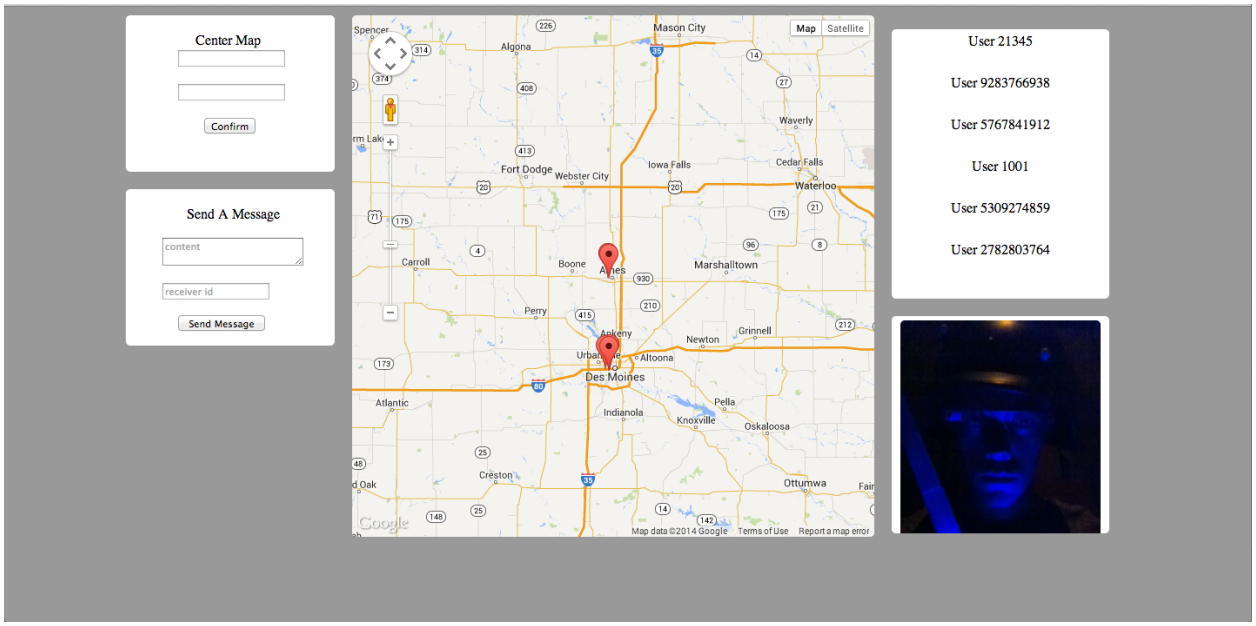
System Design

The World View web app makes heavy use of three API calls. These are:

/trackUser {"owner": "", "latitude": "", "longitude": ""}
/newMessage {"receiver": "", "content": ""}
/postPhoto {"owner":"", "latitude": "", "longitude": "", "image": ""}

After an API call is made, the web app adds the info to a database and returns the relevant information in the response. To generate the web page visuals, the web app makes heavy use of javascript and jquery and does this so that it can update the web page dynamically.

Visuals



iPhone Application

Introduction

Born out of the need for more devices using our APIs, we created an iPhone application that could receive messages, post photos, as well as track locations.

System Design

The iPhone application makes heavy use of the API calls mentioned above. It uses the native map system to plot the coordinates and a little bit of image processing in order to scale the images correctly. It makes continuous calls to continually track it's location as well as other users' locations.

Visuals