

Copyright  
by  
Michael Stathopoulos  
2012

The Report Committee for Michael Stathopoulos  
certifies that this is the approved version of the following report:

**Revolver: Synchronized Visual Event Capture Using  
Mobile Devices and Cloud Services**

APPROVED BY

SUPERVISING COMMITTEE:

---

Sarfraz Khurshid, Supervisor

---

Adnan Aziz

**Revolver: Synchronized Visual Event Capture Using  
Mobile Devices and Cloud Services**

by

**Michael Stathopoulos, B.S.**

**REPORT**

Presented to the Faculty of the Graduate School of  
The University of Texas at Austin  
in Partial Fulfillment  
of the Requirements  
for the Degree of

**MASTER OF SCIENCE IN ENGINEERING**

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2012

To my wife Lisa, my love and my best friend.

## Acknowledgments

I would like to thank Dr. Sarfraz Khurshid and Dr. Adnan Aziz for their invaluable guidance and insight on this report. I would also like to thank my professors and classmates for the knowledge and experience they shared with me during my first journey through graduate school.

# **Revolver: Synchronized Visual Event Capture Using Mobile Devices and Cloud Services**

Michael Stathopoulos, M.S.E.  
The University of Texas at Austin, 2012

Supervisor: Sarfraz Khurshid

The proliferation of mobile computing devices with powerful sensing and communication capabilities has created an immense social landscape of awareness and connectedness. Social media applications have been largely designed for asynchronous expression and collaboration among individuals. Though these models have served as suitable surrogates for social interaction in a rapidly evolving digital age, they have been insufficient at connecting people spatially and temporally. This report describes Revolver: an application utilizing the state-of-the-art in mobile and distributed computing to provide users with a shared sense of time and space. Revolver allows users to synchronously capture image data of their surroundings with the ability to virtually reconstruct an event from the separate sources. We present the rationale for the project, design considerations, implementation details, results of the prototyping effort, and conclusions to carry this project to future phases of development for viable deployment.

# Table of Contents

<b>Acknowledgments</b>	<b>v</b>
<b>Abstract</b>	<b>vi</b>
<b>List of Tables</b>	<b>x</b>
<b>List of Figures</b>	<b>xi</b>
<b>Chapter 1. Introduction</b>	<b>1</b>
1.1 Related Work . . . . .	5
<b>Chapter 2. Specification</b>	<b>8</b>
2.1 System Description . . . . .	8
2.2 User Stories . . . . .	11
2.2.1 Corroborated Evidence . . . . .	12
2.2.2 Security and Surveillance . . . . .	13
2.2.3 Visual Tour . . . . .	15
2.3 Requirements . . . . .	16
<b>Chapter 3. Design &amp; Implementation</b>	<b>18</b>
3.1 Mobile Nodes . . . . .	19
3.1.1 Design . . . . .	19
3.1.2 Implementation . . . . .	24
3.2 Server Nodes . . . . .	27
3.2.1 Design . . . . .	27
3.2.2 Implementation . . . . .	30
3.3 Client Nodes . . . . .	32
3.3.1 Design . . . . .	32
3.3.2 Implementation . . . . .	33

<b>Chapter 4. Results</b>	<b>35</b>
4.1 Experiences . . . . .	35
4.1.1 Mobile Platforms . . . . .	36
4.1.2 Server Platforms . . . . .	38
4.2 Effort . . . . .	39
4.2.1 Mobile Platforms . . . . .	39
4.2.2 Server Platforms . . . . .	40
4.3 Performance . . . . .	42
<b>Chapter 5. Conclusions</b>	<b>45</b>
5.1 Future Areas of Work . . . . .	46
<b>Appendix A. User Interfaces</b>	<b>50</b>
A.1 Mobile UI . . . . .	50
A.2 Server UI . . . . .	54
A.3 Client UI . . . . .	56
<b>Appendix B. Measurements</b>	<b>58</b>
B.1 Communication Measurements . . . . .	58
B.2 Software Measurements . . . . .	59
B.2.1 Mobile Application Prototypes . . . . .	60
B.2.1.1 2011/Android/Java . . . . .	60
B.2.1.2 2012/iOS/Objective-C . . . . .	60
B.2.2 Server Application Prototypes . . . . .	61
B.2.2.1 2011/GAE/Java . . . . .	61
B.2.2.2 2012/GAE/Python . . . . .	61
B.2.2.3 2012/MAMP/PHP . . . . .	62
B.2.3 Client Application Prototypes . . . . .	63
B.2.3.1 2012/JRE/Java . . . . .	63
<b>Appendix C. Future Work</b>	<b>64</b>
C.1 Mobile Nodes . . . . .	64
C.1.1 Design . . . . .	64



<b>Bibliography</b>	<b>67</b>
<b>Vita</b>	<b>71</b>

## List of Tables

1.1	Common smart-phone sensors and measurement . . . . .	2
2.1	System requirements . . . . .	17
3.1	Foundation Framework classes for moment capture . . . . .	25
B.1	Statistical results for Wi-Fi and 3G . . . . .	59

## List of Figures

2.1	Use case diagram . . . . .	9
2.2	Event schedule . . . . .	10
2.3	Event capture and view . . . . .	11
3.1	Manual capture mode behavior . . . . .	20
3.2	Auto capture mode behavior . . . . .	20
3.3	Attitude and location for mobile devices . . . . .	21
3.4	Event entity model . . . . .	22
3.5	Moment upload sequence . . . . .	23
3.6	Moment-manager class diagram . . . . .	24
3.7	Event scheduling sequence . . . . .	29
3.8	Event data-model ERD . . . . .	30
3.9	Data presentation pipeline . . . . .	33
4.1	Server code size in LOC . . . . .	41
4.2	Server implementation effort in hours . . . . .	41
A.1	Authentication . . . . .	50
A.2	Permissions . . . . .	51
A.3	Scheduling . . . . .	51
A.4	Date and time selection . . . . .	52
A.5	Selecting mode . . . . .	52
A.6	Inviting contributors . . . . .	53
A.7	Event capture . . . . .	53
A.8	Web scheduling . . . . .	54
A.9	Web viewing . . . . .	55
A.10	Transformed image presentation . . . . .	56
A.11	Image at map location . . . . .	57

B.1	Normal distributions for Wi-Fi & 3G . . . . .	59
C.1	Movie capture mode behavior . . . . .	66
C.2	Event entity model (movie capture mode) . . . . .	66

# Chapter 1

## Introduction

The proliferation of mobile computing devices with powerful sensing and communication capabilities has created an immense social landscape of awareness and connectedness. “Total mobile-cellular subscriptions reached almost 6 billion by end 2011, corresponding to a global penetration of 86%” [36]. The latest generations of “smart-phones” have a multitude of sensors which can be used to measure, analyze, and digitally record one’s local environment; Table 1.1 lists some common sensors for such devices and what they measure. Such sensing, computing, and communicating ubiquity is creating a virtual global nervous system. This explosion in technological innovation is rapidly fostering the emergence of specializations within the Information Technology domain such as “cloud computing” [27] and “big data” [3]. The means by which people can connect and share information is expanding; the challenge is in finding novel ways of configuring and using these technological building-blocks to solve problems and enhance our lives.

Sometimes software applications push the boundaries of the technology underlying them. These so-called “killer apps” act as fulcrums between the experiential value users perceive of them and the limitations imposed on their

Table 1.1: Common smart-phone sensors and measurement

Sensor	Measurement
Gyroscope	Orientation
Magnetometer	Direction
GPS	Location
Accelerometer	Acceleration
Pressure	Altitude
Camera	Images, Video
Microphone	Audio

potential due to the systems they run on. If an application garners enough user demand, producers of the supporting technology are more willing to extend the capabilities of their products. In turn, as the capability of software infrastructure expands, the next generation of applications come in to further challenge the boundary; this synergistic process accelerates technological advancement. Revolver aspires to be such an application.

Revolver is a multi-user distributed application which leverages state-of-the-art technologies in mobile sensing and computing, network communications, cloud services, data persistence, presentation, and social-network integration. From a functionally fundamental perspective, Revolver is a mobile application which captures visual images and associated meta-data such as device orientation (relative to a frame of reference), geographic location, and time-stamp. Similar to existing social-media applications, Revolver enables users to share and access data utilizing a commonly accessible infrastructure such as the Internet. Furthermore, Revolver enables collaboration on a par-

ticular activity or event between spatially separate users. The key ingredient to Revolver’s novelty is that it utilizes physical clocks to coordinate and synchronize the activity of each user with one another, and then combine the collective data to form a spatial view of each moment (in the cloud).

By synchronizing the visual-capture functionality of the mobile application for all users of a particular event—utilizing each mobile device’s local clock—Revolver is capable of aggregating the data uploaded from each device and constructing a multi-perspective visualization for each instance of time. Revolver entertains the idea of having a multitude of “eyes” connected to one shared “brain”, capturing scenes vastly apart in space but precisely at each moment. Revolver could gain momentum in a number of personal and organizational application-domains. Individuals would value the ability to share their present space with distant family and friends and see what others are seeing at the moment. Groups could actively collaborate on projects to create multi-perspective motion pictures. Existing business could use Revolver to augment their existing information technologies and new businesses could be formed with Revolver as a key ingredient. If popularized, Revolver could then challenge the state-of-the-art for faster communications, expanded bandwidth, higher-precision timing & synchronization, and better power efficiency. Furthermore Revolver has the potential to challenge the existing paradigms of data visualization, presentation, and analysis.

A user can schedule an event using a mobile device or other client interface, and optionally invite other users to participate. Using the mobile

application and device, each user can initiate visual capture of their local environment independent of other user; the application determines a synchronized interval for the device to capture data. At each moment, the visual data—along with data on the device’s position and orientation—are uploaded to a server and persisted. The data can then be retrieved, combined, and presented in various ways using various clients including machine-native and web applications. The most compelling presentation format is to spatially arrange the visual data for each moment of time to create a kind of 3-dimensional motion picture.

The rest of this chapter presents the results of a literature survey on related work as well as research on similar commercially available products. Chapter 2 describes Revolver in further detail, enumerating scenarios of use, and eliciting a set of essential requirements for implementation. Chapter 3 presents design, architecture, and implementation details for the latest prototype efforts for the mobile, server, and client aspects of the system. Chapter 4 provides some commentary on the results of the prototyping effort including analysis on system performance and experiences with the chosen technologies and approaches. Finally Chapter 5 concludes the report with elaboration on the challenges future iterations of Revolver must overcome to be a commercially viable product; proposals are presented for the evolution of Revolver and its infrastructure. Supporting materials to this report are in the appendices.



## 1.1 Related Work

There are a wide range of photography and video related applications for mobile devices commercially available, and many more are under development. Given some of the characteristics of Revolver described so far, the functionality may seem similar to panorama applications like Occipital’s *360 Panorama* [30] or Microsoft’s *Photosynth* [28] for mobile devices. Panoramic images—commonly referred to as *panoramas*—involve computational techniques for image stitching [35] and some of these techniques can be applied to Revolver’s data presentation, however the key differences are:

- Panoramas are produced with a single camera which is moved about to capture the scene within a frame. The camera can only capture linear scenes if the camera is moved in a straight line or concave scenes if the camera is panned from a fixed point. Revolver uses several cameras to capture a scene in any geometry, and is capable of capturing convex scenes.
- Panoramas traditionally have relied on image processing techniques to determine matching patterns in various images and place the images spatially based on those patterns. Revolver provides meta-data with each image which indicates location and orientation of camera when image was captured, therefore speeding the processing of scenes.
- Panoramas are static scenes. Revolver can capture scenes consecutively, creating a motion-picture.

Besides having panoramic functionality, *Photosynth*-like Revolver-has the ability to create scenes of various geometries based on images from different perspectives. *Photosynth* is the result of work by Noah Snavely, Steven Seitz, and Richard Szeliski [33, 34]. Their work models 3-dimensional scenes from publicly available images on the Internet and uses image processing techniques for scene construction. Their work utilizes images taken at different times from various sources to construct scenes, whereas Revolver creates scenes from images taken at the same time and from a relatively consistent set of cameras which can lead to a less mosaic scene. Furthermore their work relies on computationally intensive techniques to place images based on image-data alone where Revolver provides meta-data for image placement.

The most closely related work is that of Grosvenor and Cheattle described in US Patent 7139018, *Synchronized cameras with auto-exchange* [21]. The patent describes a system very much like Revolver where the following characteristics are highlighted from the patent:

- Creating images using digital recording devices.
- Subjecting the users devices to a synchronization event.
- Establishing communication connection between recording device and database.
- Transmitting image from device to database without input from user.
- Enabling user to access recording from database.

The patent describes synchronization capabilities directly between devices as well as client-server models, whereas Revolver relies solely on the client-server model. The patent suggests that digital images are synchronized per event by a synchronization code and uploaded via the Internet to a dedicated database for the particular code; this implies the synchronization creates a pool of shared images where Revolver actually synchronizes images to a moment in time. The patent mentions the use of digital recording devices but does not suggest the use of smart-phones and one aspect of Revolver's design is that of mobile applications that can be executed on various smart-phones; in fact, the patent suggest the incorporation of specialized controls into digital cameras for the purpose of device synchronization. Furthermore the patent makes no mention of how the data is to be processed or presented, only that the data can be shared between devices or later retrieved from a database; no mention is made of utilizing device sensors for location and orientation of images. The patent was filed in 2002, before the mass proliferation of high-capability smart-phones, and granted in 2006. Though Revolver's inception was not until late 2011, it occurred before the discovery of this patent. Nevertheless for Revolver to become a commercially viable product we feel it would have to be evaluated against this patent prior to any commercial development.

## Chapter 2

# Specification

This chapter starts with the system description as a basis for formulating specifications for the application. The system is further elaborated on by use of user stories to narrate several scenarios of application use. We then attempt to elicit some requirements from these scenarios which guide implementation in chapter 3.

### 2.1 System Description

Revolver is a software-system application which has different functionality and user-interface characteristics based on the use-cases the user is executing or the role the user is playing at a given time; Figure 2.1 presents the actors and use-cases for the application.

Users can be one or several individuals that interact with every aspect of the system, or groups with specific roles. The actors and their roles are:

**User** A group collaborating through the application, using the entire system.

**Coordinator** A group scheduling events for other users and analyzing the results using cloud services and client interfaces.

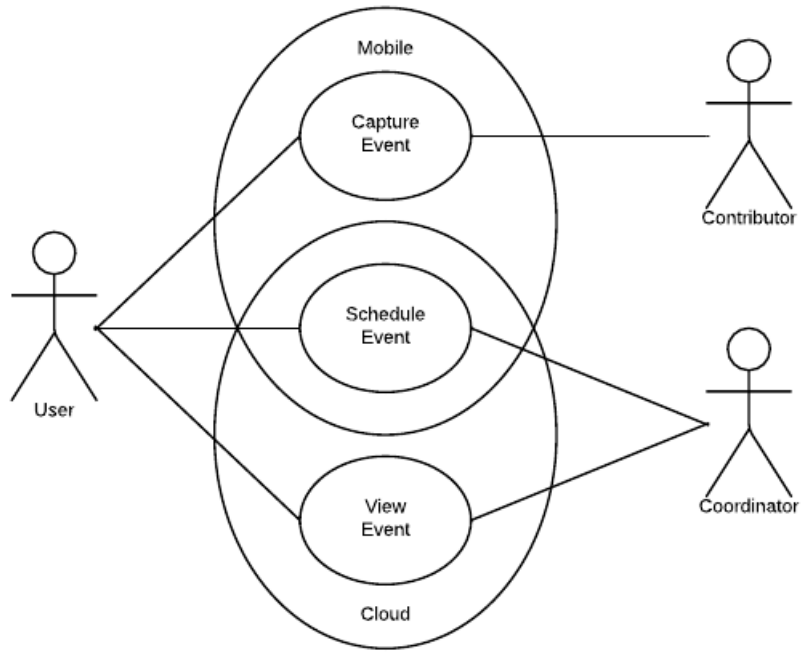


Figure 2.1: Use case diagram

**Contributor** A group capturing event data using mobile devices.

The use-cases are separated between mobile device application and cloud application (which could be interfaced via native-platform client or web browser). The use-cases together form a process for utilizing the application and their order is:

**Schedule Event** Configure identification, date, and time of event and invite collaborators.

**Capture Event** Record and transfer visual and meta-data.

**View Event** Aggregate, process, and present data.



Figure 2.2: Event schedule

Figure 2.2 depicts the process of scheduling an event. A single user configures the event parameters and registers with the cloud server. The event parameters are pushed to contributing users via notification services, or the user's device pulls from the server on demand. One of the parameters is the starting date and time of the event. All devices then have a reference-time and can use their local clocks to synchronize capture. Users can then initiate event capture at will; the mobile application calculates when to capture

the data and triggers the function at the configured interval. The captured data is immediately uploaded to the cloud server where it can be presented by a client application; the data is available as soon as it is uploaded for each moment. Figure 2.3 depicts the capture and presentation of event data. With the data being persisted and accessible through cloud services, it can be accessed and utilized at multiple locations concurrently and independent of client or platform.

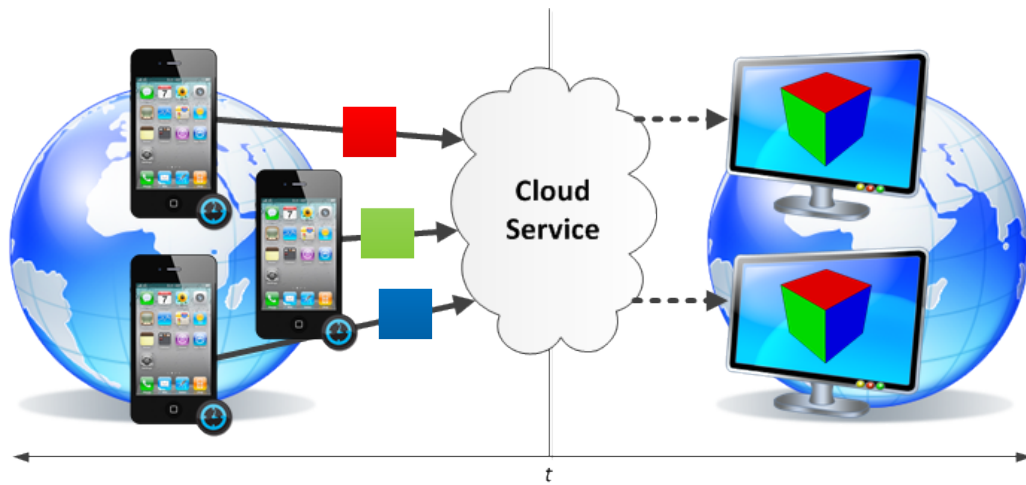


Figure 2.3: Event capture and view

## 2.2 User Stories

The specifications are first enumerated in a series of user stories. The user stories are grouped into three scenarios (Sections 2.2.1 to 2.2.3) and are written in prose form. Although initially describing user interaction with the system using this informal method makes it more challenging to elicit discrete

requirements, it leaves room for further extension and enhancement which fits with the vision of the product for future iterations.

Each hypothetical scenario takes several users through collaborative interactions with the system, attempting to highlight key features along the way. Each scenario also represents a different domain Revolver is applicable to; where applicable, the scenarios also reference related works which provide additional context to the problem domain. In common, the stories start out with a user coordinating an event for which other users will eventually collaborate on or utilize in some fashion. The key features are *emphasized* in the stories for later inclusion into the requirements.

The scenarios described represent only a subset of intended uses for the application and are representational of the scope of this report. Further extensions of Revolver are discussed in the conclusions in Chapter 5.

### **2.2.1 Corroborated Evidence**

Amanda is coordinating an event of “civil disobedience” with many like-minded individuals she is *connected to through social media* to protest some recently enacted public policies her and her friends found contentious. Amanda, tending to be quite organized and mindful of civil laws and liberties, wanted her and friends to be prepared to deal with any unfavorable consequences of their intended actions.

Amanda had recently read an article about a mobile application developed by the American Civil Liberties Union of New Jersey. The application



enabled users to record video of law enforcement officials potentially infringing on their civil liberties and immediately upload the video to a server for backup storage and analysis in case their device was confiscated [25]. Amanda contemplated the benefits of such an application; the ability to *capture visual evidence* which is *automatically and immediately uploaded* from the device and *persisted at an off-site location* would be beneficial in a legal defense case.

Amanda done some research and came across Revolver, which she realized would fulfill the some of the specific needs she was anticipating. She realized she could use Revolver to *coordinate* with several of her friends for this particular event. Besides capturing and uploading visual events, she saw that Revolver was capable of also recording the *location and orientation* of each event from each user's perspective as well as a *time-stamp* of the event. It dawned on her that such an application could provide corroborated evidence in a legal defense: multiple witnesses supporting each others case by providing visual evidence of a single event from different perspectives. Amanda was certain the use of Revolver would provide some insurance for her and her friend's civil activities.

### **2.2.2 Security and Surveillance**

Sam's Security is a fledgling security service trying to differentiate itself from other firms to get a grasp on the security-services market. Although a relatively small start-up, Sam would like to use technology to give his company an edge but since his business is independently financed he has a *restrictive*

*budget* to work with initially.

Sam was intrigued by an article he read in a trade journal about Visual Sensor Networks (VSNs), particularly of the author’s description of the system (emphasis ours):

A VSN consists of a group of nodes, each equipped with a *low power, embedded processor, energy source, image sensor, and some type of transceiver for communication*. These nodes must also be capable of communication with the network base station or sink where the *data is collected and further processed for end-user consumption*. The ability of these nodes to communicate with each other creates the possibility for event and anomaly detection over a group of nodes known as a cluster. [26]

After finding Revolver it dawned on Sam that it could be used to implement an affordable VSN for his security business. Since his security guards were already equipped with smart-phones as part of their standard equipment, he realized he already had a mobile VSN. Using Revolver, Sam could *centrally coordinate and monitor* the local environment in *near-real-time* with *all perspectives synchronized to each other*. He could schedule guards on duty to use their devices to capture images of the grounds being secured and get an overall visual perspective of the environment by *aggregation and combination of visual data from separate and distant sources*. Sam could now utilize affordable *ubiquitous* technology to give his business a competitive edge.

### 2.2.3 Visual Tour

John had to go on a business trip to Frankfurt, Germany; this was his first trip to the country and he thought it would be good to take a vacation in Germany with his wife Nancy right after his business was done. The plan was for John to fly in the previous week and Nancy to fly in the following weekend and meet John at his hotel.

When John got to Frankfurt, he took the local train to the vicinity of the hotel. He thought it would be easy enough to find from foot, but that turned out not to be the case. The hotel was not clearly marked and was located in the center of a town plaza which further obscured it from view of the nearby streets. John did not want Nancy to have the same experience as he did, and he hoped to avoid the experience again in other cities and towns in the country.

When John found Revolver, he quickly realized how it could benefit him and Nancy in their travels. John went back to the train stop and coordinated an event which he invited Nancy to. He then proceeded to walk from the train stop to the hotel *manually capturing photos* of inconspicuous landmarks along the way; as the photos were captured, information such as location, orientation, and time were also saved with the photos.

With Revolver, *participants to events have access to the visual data of other users*, so Nancy could see all of John's photos. Furthermore, the application could *associate each picture with a time and location on a map*;

Nancy could get a “visual tour” of the path John walked and what he saw. Now Nancy felt confident she could follow in John’s footsteps and meet with him so they could start their journey through German together.

## **2.3 Requirements**

Based on the user stories of Section 2.2, we have enumerated various requirements the application must fulfill in Table 2.1. The requirements are numbered based on a prefix corresponding with the sub-section representing the scenario they came from. The requirements are specified in the order they are encountered from the emphasized statements of the scenarios.

Table 2.1: System requirements

<b>Req. #</b>	<b>Requirement</b>
2.2.1.1	Social media integration
2.2.1.2	Visual data capture
2.2.1.3	Automatic & immediate upload
2.2.1.4	Off-site persistence
2.2.1.5	Coordinate between users
2.2.1.6	Capture orientation & location
2.2.1.7	Capture time-stamp
2.2.2.1	Low-powered embedded devices with sensors & communications
2.2.2.2	Processed data presentation
2.2.2.3	Central coordination & monitoring
2.2.2.4	Near-real-time operation
2.2.2.5	Synchronized operation between users
2.2.2.6	Data aggregation/combination for presentation
2.2.2.7	Various mobile/embed platform support
2.2.3.1	Manual capture mode
2.2.3.2	Shared data among users
2.2.3.3	Present data through map application

## Chapter 3

# Design & Implementation

As depicted in Figures 2.2 and 2.3, Revolver relies on mobile devices, cloud services, and client interfaces for its utility; it is by definition a *distributed system* [31]. It is a heterogeneous system of diverse components or *nodes*, connected by common protocols, where the nodes are grouped by specific functions. The mobile-device nodes perform a very specific function which is to capture and transmit data to servers; outside of event synchronization, mobile devices do not serve as clients to the captured data (in the scope of current design). The server nodes perform data transference and persistence. The recent availability of reliable, configurable cloud services has blurred traditional concepts of server nodes; now the nodes are a “cloud” where computing hardware, network infrastructure, and operating system appear as a single node to users and administrators. Client applications which process and present the captured data are also nodes in the system.

The following sections present some key design artifacts and implementation details for each node respectively. The artifacts and details were derived from the requirements in Chapter 2 and some aspects of the design and implementation are traced back to Table 2.1. The implementation details

expose some aspects of platform-specific technologies used to deploy and test the application during the prototype effort.

## 3.1 Mobile Nodes

### 3.1.1 Design

Mobile devices in Revolver play a limited but crucial role in the application as data collectors and transmitters; though simple in user function and interaction, the complexities have been pushed down the application stack closer to the hardware. There are two modes of capture operation: *manual* (Figure 3.1) and *automatic* (Figure 3.2). In manual mode, no synchronization between devices occurs and a user can trigger capture free of constraint; this mode is suitable for Requirements 2.2.3.1 to 2.2.3.3. The intended mode for collaboration and advanced visualization is automatic, where the device’s capture function is constrained to a clock regardless of when the user triggers it. Automatic mode covers requirements 2.2.1.5, 2.2.2.4, and 2.2.2.5 among others.

Besides image data, the device’s *attitude* and *location* are captured and associated with the image in the form of meta-data. The attitude represents the device’s orientation relative to a frame of reference and can be measured by angles of rotation about the device’s **x**, **y**, and **z** axis. The location represents the devices geographical coordinates relative to a frame of reference and can be measured by latitude and longitude. Common frames of reference are true north—as opposed to magnetic north—and the direction of gravity. Figure 3.3

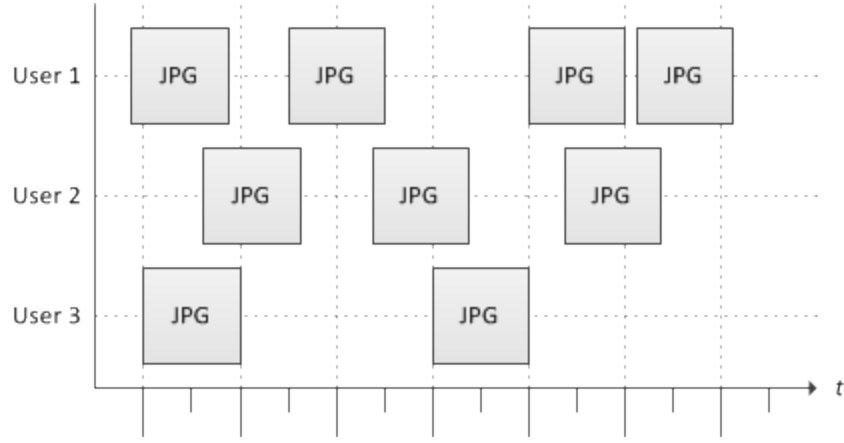


Figure 3.1: Manual capture mode behavior

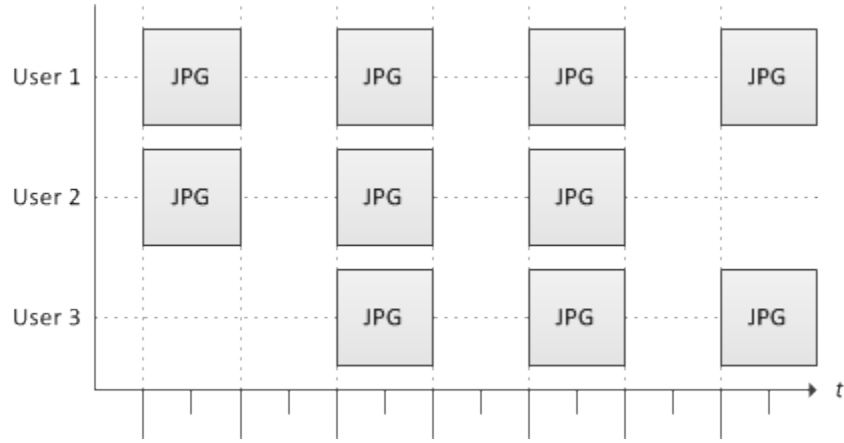


Figure 3.2: Auto capture mode behavior

depicts common measurement and reference parameters and their relation to the device<sup>1</sup>.

The attitude, location, and image comprises an entity known as a *mo-*

---

<sup>1</sup>Relationship of parameters to device may vary with hardware and operating system.



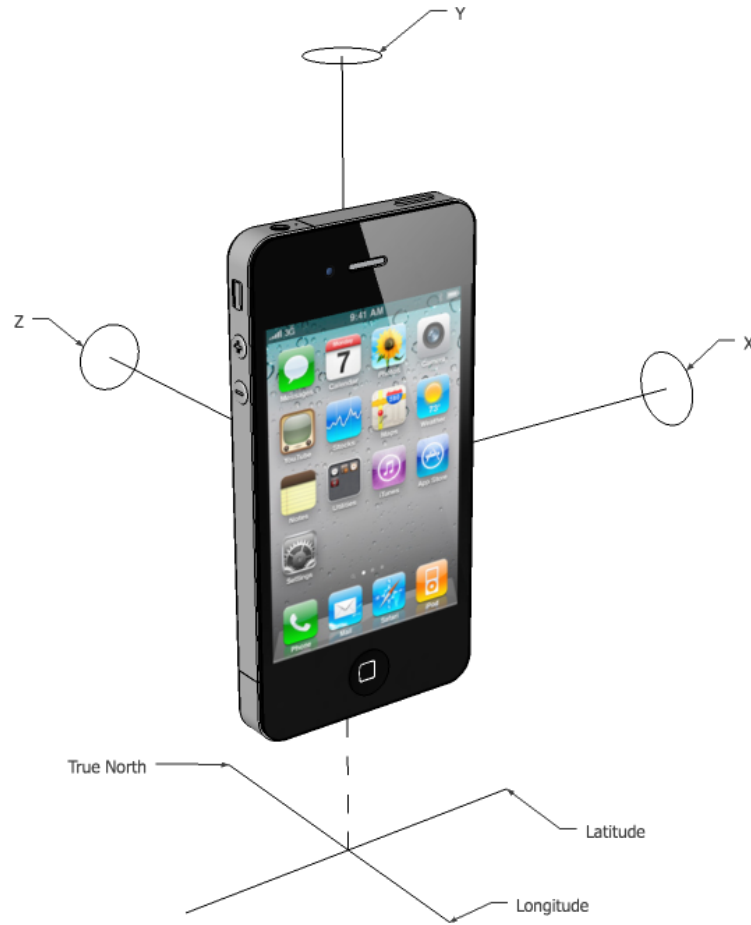


Figure 3.3: Attitude and location for mobile devices

*ment*. A moment is created every time a users device automatically or manually captures data for upload to the server; parameters of each moment are examined in Section 3.2.1. Moments from all users are aggregated and associated with an *event* which is scheduled prior to capture. Figure 3.4 presents

an abstract model of the relationship between events, moments, and the data-units comprising moments.

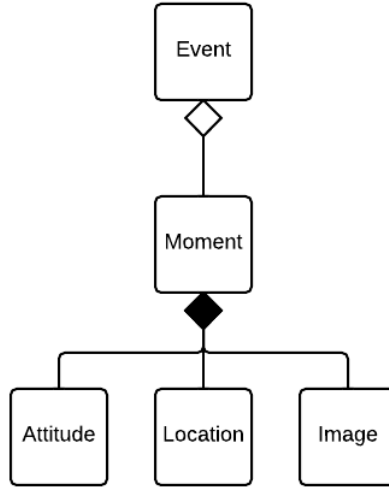


Figure 3.4: Event entity model

When in automatic mode and a capture session is initiated, a static or singleton object<sup>2</sup> manages the synchronization and capture operations and schedules upload of the data; this object can be referred to as the *Moment-Manager* (MM). The MM compares the local time capture is initiated with the time the event was created; it then determines a delay before it schedules a timer at interval. When the timer triggers, the MM collects all data and dispatches it to an object that will upload the data to the server asynchronously and concurrently. Figure 3.5 depicts this sequence of operations.

---

<sup>2</sup>A static or singleton object is required for mutually exclusive access to device resources and coordination between device sub-systems.

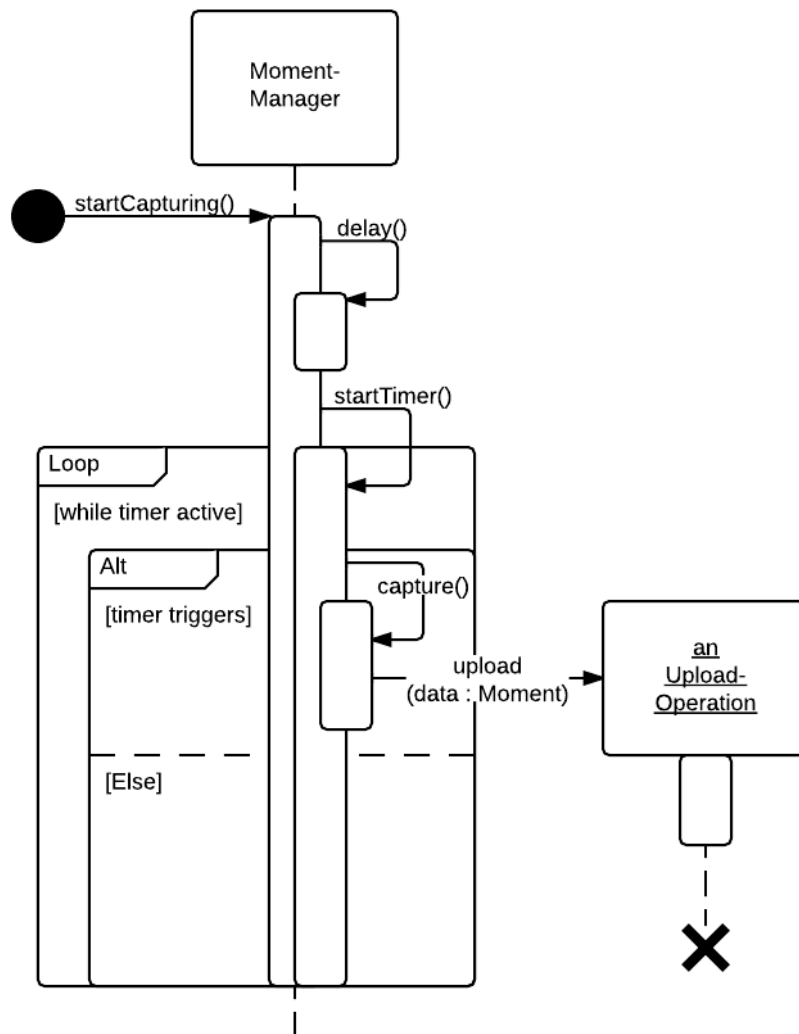


Figure 3.5: Moment upload sequence

### 3.1.2 Implementation

The prototype described in this report was implemented on the Apple iOS platform<sup>3</sup>. The iOS platform was chosen primarily for the provision of data consistency between device generations due to Apple’s control over both hardware and operating system.

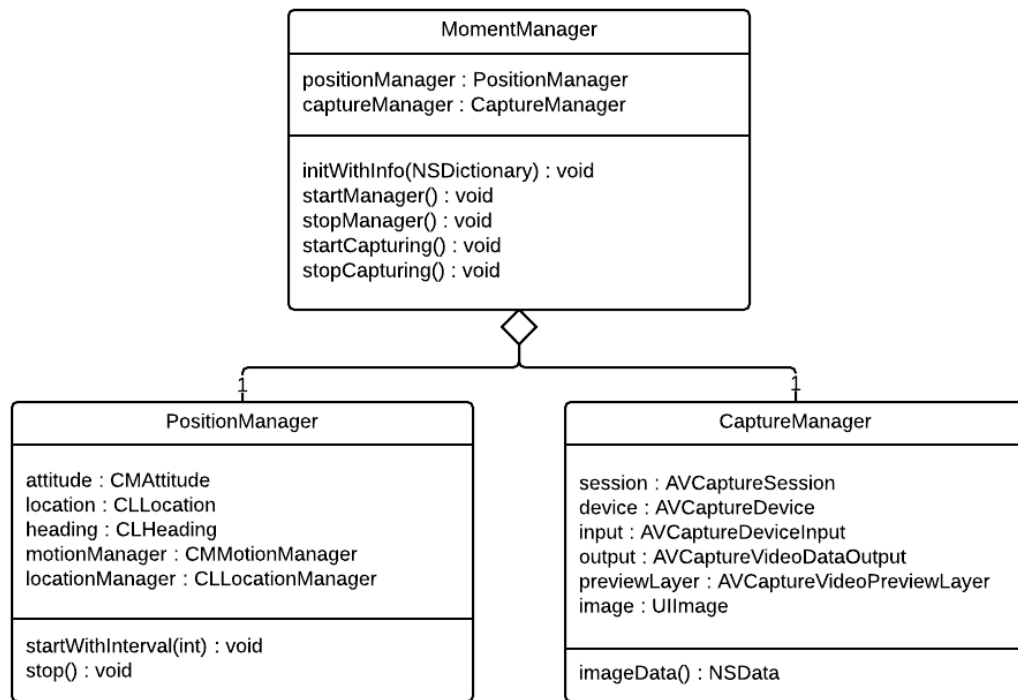


Figure 3.6: Moment-manager class diagram

The MM previously mentioned has a *PositionManager* (PM) which asynchronously manages resources for collecting device attitude and location

---

<sup>3</sup>An earlier prototype was also developed on the Google Android platform for previous coursework.

and a *CaptureManager* (CM) which manages resources for capturing images. Figure 3.6 shows a class diagram of the MM and its associated PM & CM objects. Table 3.1 describes the significant *Foundation Framework* classes the PM & CM incorporate to perform their respective function.

Table 3.1: Foundation Framework classes for moment capture [6]

Class	Description
CMMotionManager	Provides accelerometer data, rotation-rate data, magnetometer data, and other device-motion data such as attitude.
CLLocationManager	Defines interface for configuring delivery of location- and heading-related events.
AVCaptureSession	Coordinates flow of data from AV input devices to outputs.
AVCaptureDevice	Represents a physical capture device and the properties associated with that device.
AVCaptureDeviceInput	Captures data from an AVCaptureDevice object.
AVCaptureVideoDataOutput	Processes uncompressed frames from the video being captured.
AVCaptureVideoPreviewLayer	Used to display video as it is being captured by an input device.

The user interface for configuring and capturing events was designed with speed and efficiency in mind since the mobile device is just a sensor and may have to be set up and used quickly. Section A.1 in the appendices provides screen-captures of the prototype user interfaces from the application storyboard. The procedures for authenticating the application, scheduling and event, and initiating capture as well as key implementation details of each

procedure are enumerated:

1. If not already logged in, the user authenticates the application with their credentials (Fig. A.1). In this prototype, the Facebook API is used for authenticating the application using the OAuth protocol [2] and the user's Facebook credentials. If this is the user's first time logging into the application, Facebook redirects the user to a browser site where they can set permissions and allow the application access to their information (Fig. A.2).
2. The user may select an event to join from existing events in their list or they may create a new event and fill in the details such as name, starting & ending date, and mode (Fig. A.3). Selector controls constrain the user from selecting invalid inputs for date & time (Fig. A.4) and mode parameters (Fig. A.5).
3. When scheduling an event, the user may invite contributors to collaborate on the event (Fig. A.6). In this prototype, a custom user-control by Facebook called *FBFriendPickerViewController* to display a list from the user's social graph and provide a collection of selected users to the application [10]. For other application domains, an independent authentication and user-selection service could be implemented.
4. The user can then select the event they want to initiate from the list of created or invited events. The camera-preview interface is presented where the user can start and stop capturing the event (Fig. A.7).

## 3.2 Server Nodes

### 3.2.1 Design

Compared to the mobile node, the server node is considerably less complex in design; it merely serves as the data backbone to the other nodes. All interactions of the system are client-server where the client (including mobile nodes) make requests of the server to transmit data, receive data, or both; though in this current design the server does not “push” data to other nodes, it may be considered for future enhancement. Other design considerations of the system are “fat” clients, where the application logic is distributed to the other nodes, and stateless services where client-server interactions are completely self-contained [31].

Another attractive set of design principles incorporated into Revolver is that of REST or RESTful Web services.

REST defines a set of architectural principles by which you can design Web services that focus on a system’s resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages. [32]

The cited article by Rodriquez suggests a REST implementation follows four basic principles:

- Explicit use of HTTP methods: POST (create), GET (retrieve), PUT (change), and DELETE (remove)

- Use of stateless services
- Directory-like structure for URIs
- Transfer in XML or JavaScript Object Notation (JSON) format

The attractiveness of this design pattern—besides its logical formulation and use of existing standards—is that it is becoming a de facto standard for Web services. This makes Revolver’s client-server interfaces intuitive, maintainable, and scalable.

Figure 3.7 shows the sequence for a mobile device scheduling an event with the server; a similar sequence would occur for other interactions such as getting a list of invited events or retrieving a moment from the server. An HTTP POST method transfers the event object to an event-handler object. The event-handler verifies the data is within type- and structure-constraints before passing to a database-handler. The database handler transforms the data-object from its native format to an entity-schema compatible with the underlying database. An ID auto-generated by the database system is then transferred back up the stack to the initiator; this ID signifies successful persistence of the data and is also the key within the client to be able to retrieve that particular record of event.

Figure 3.8 depicts a prototypical entity-relationship model of the data that is handled by the server. This model assumes a schema but may be altered and represented differently if the underlying database technology were



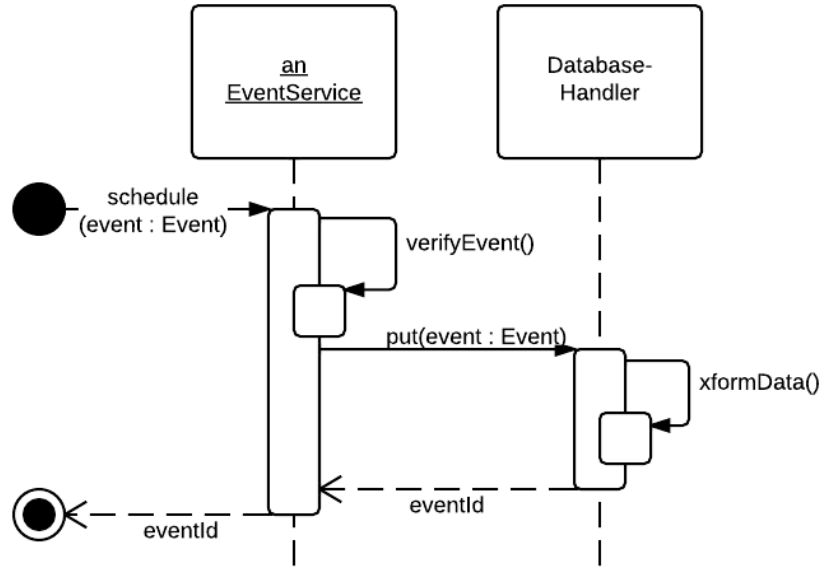


Figure 3.7: Event scheduling sequence

schemaless; fundamentally the data would be the same and only its relative organization would differ. Notable in the diagram are the attributes  $qx$ ,  $qy$ ,  $qz$ , and  $qw$  in the moment entity: these are parameters of a quaternion [18] which represents the device's orientation; other parameters such Euler angles or that of a rotation matrix may be substituted or added. Another notable attribute is that of *image* in moment: this design assumes the binary data is persisted directly in the database but depending on size of image<sup>4</sup>, it may be persisted as a file in the file-system<sup>5</sup> and this attribute would serve as a pointer to the file.

<sup>4</sup>May refer to video data in future iteration; Appendix C.

<sup>5</sup>May be persisted as file-like object in specialized datastore; see *Blobstore* [20].

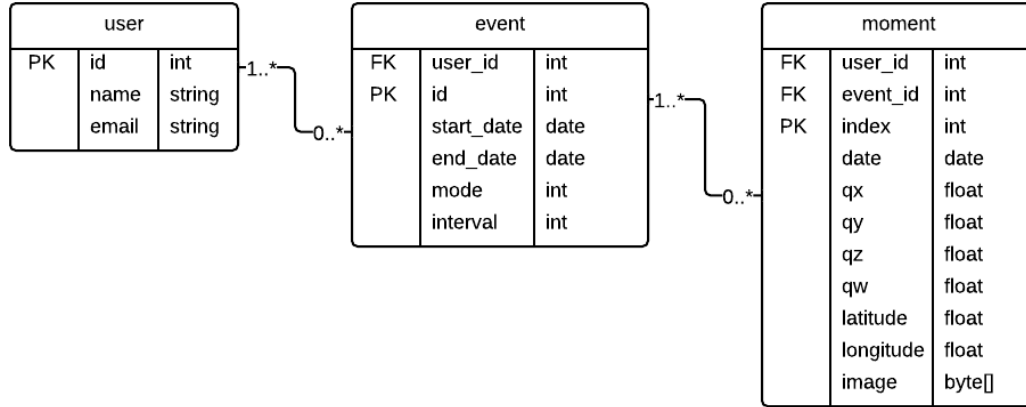


Figure 3.8: Event data-model ERD

### 3.2.2 Implementation

Traditional server implementations may have utilized a Linux/Apache/MySQL/PHP (LAMP) stack on servers and infrastructure that were rented or owned but the recent availability of inexpensive (and sometimes free) cloud services have expanded options for developers and administrators. LAMP (or other) stacks can be executed on Amazon Web Services (AWS, [1]); this leverages the on-demand scalability of cloud services with the maturity of these traditional solution-stacks. For this report we developed a prototype implementation for the Google App Engine (GAE, [20]).

GAE provides run-time environments and support for a limited set of programming languages. Prototypes for this report were developed with the Java and Python languages for cross-comparison<sup>6</sup>. The GAE run-time is a

<sup>6</sup>Refer to Section B.2.2 for measured comparison.

sand-boxed environment; constraints on library support are specified for each environment. GAE does not support a traditional file-system but rather a distributed, fault-tolerant data-base using the Paxos consensus algorithm [9]. Experiences with GAE are further detailed in Chapter 4.

Each node-type was implemented on different platforms using different languages; as REST specifies, a common protocol was established for data exchange. Though XML and JSON are supported on all platforms, the *property list* (plist) format was chosen because it could be generated and parsed up to three times faster on the iOS platform than the other formats for files of equal size<sup>7</sup>. Plist supports binary, XML, and ASCII format with the binary format being most efficient; the binary format when parsed provides an XML-like structure of nested key-value pairs. Plist is suitable as a data-exchange protocol because it is constrained by supported types and includes the data-type in the structure; parsers are able to verify and reconstruct the entire structure into local objects suitable for processing. Open-source plist libraries are available for many popular languages; some languages come bundled with them as a standard library. Suitability was demonstrated on the use of plist for data transfer between iOS and GAE [17]; libraries utilized for the other languages were also successful.

---

<sup>7</sup>Source: *Building a Server-Driven User Experience* [5].

### 3.3 Client Nodes

#### 3.3.1 Design

Image processing and data visualization are subjects of considerable breadth and depth; volumes have been written on the subjects a few of which are referenced in this report [12, 35, 38]. This section provides a design for the presentation layer adequate for basic demonstration of Requirements 2.2.2.2, 2.2.2.6, & 2.2.3.3. A more elaborate treatment of the topic would have expanded the scope of this report unfeasibly and is left as a future area of work.

Figure 3.9 presents a pipeline architecture for a simple visual-presentation client. Prior to the presentation of the visual data, the client would present to the user a navigation-and-selection interface enabling the user to select which grouping of data to visualize; examples of groupings include:

- Images for all users from a selected event.
- Images for selected users of a specific event.
- Images for a selected index (moment) from all users for a specific event.

Once the group is selected, the client would retrieve the data from the server; this could be accomplished by the client transferring to the server a query string or query parameters and the server returning a structured object of fetched moments. Ideally the data retrieval could happen concurrently with setup of the visual environment which would include background coloring and

data placement. The visual data would be transformed into place based on the meta-data from each moment of the retrieved set. Finally the scene would be presented to the user; interactive controls could enable them to navigate the visual scene.

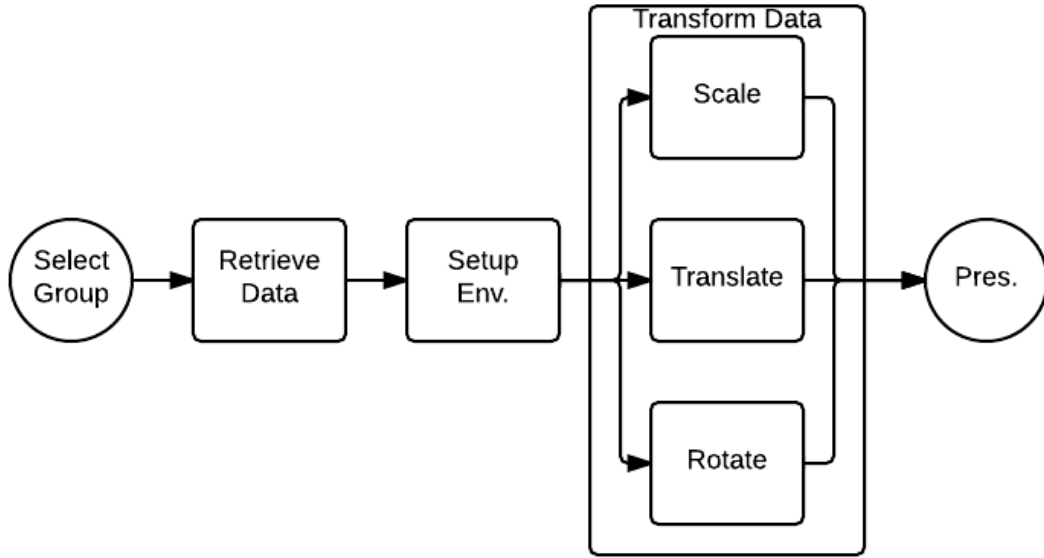


Figure 3.9: Data presentation pipeline

### 3.3.2 Implementation

A simple proof-of-concept prototype was implemented with *Processing*, “an open source programming language and environment for people who want to create images, animations, and interactions” [13]. Alternatively, the client could have been implemented with a variant of Processing called *Processing.js* [14] which interprets Processing code into JavaScript for execution in a browser. Figures in Appendix A present example output from moments

retrieved from the sever and transformed. Additionally, an open-source library called *Unfolding* [29] was used for placing the images in location over street maps.

At its core, Processing is a Java library which provides a graphics context for the creation of visual structures. It was initially created as a tool to help teach fundamentals of computer programming and is a very simple language and library to learn and use for visualizing data. It supports 2-dimensional as well as 3-dimensional rendering including support for OpenGL constructs. There is a considerable amount of support for Processing in the open-source community and many third-party libraries are available for simplifying tasks and work-flows.

Processing is ideal for rapid prototyping and for this reason was chosen to implement the pipeline architecture of Section 3.3.1. Processing is also a very powerful tool for creating rich, interactive data-visualization experiences.

## Chapter 4

### Results

This chapter attempts to summarize some of the more notable observations, experiences, and outcomes of the prototyping effort in order to help guide future work on Revolver. Section 4.1 discusses our experiences with the selected technologies used for the prototypes and the challenges encountered. Section 4.2 discusses the amount of effort that went into each aspect of the system prototype and provides reasoning for the effort. Section 4.3 makes some observations about the performance of particular aspects of the system and the challenges they pose for future iterations.

#### 4.1 Experiences

Overall the outcome of the prototyping effort was as expected with the selected technologies. Different technologies were selected to prototype different aspects of the system in the hopes to be able to compare the experiences and choose the best technologies to support Revolver. Some aspects of Revolver must run on multiple platforms—such as the mobile application—and the comparison between platform-technologies is essential for developing a consistent experience between the different platforms. With other aspects of

Revolver such as the server application, it is reasonable that a single platform be selected to serve the system; however, effort can be a contributing factor to selection and Section 4.2 expands further on the topic.

#### **4.1.1 Mobile Platforms**

Revolver prototypes were made for both the Google Android and Apple iOS platforms. Both platforms have advantages and disadvantages with regards to development and usability and it is not the intention of this report to favor one platform over the other; rather, an effort is made to distinguish the characteristics of each which require further examination and development efforts in order to provide a consistent user experience between the platforms.

One area of differentiation has more to do with hardware than software. The types and qualities of sensing hardware differ between platforms; with the Android platform, this is more significant as Android supports many providers of devices with various models whereas iOS has only one provider with a few generational models. Information about the sensing hardware used in the devices are difficult to ascertain because it is most often considered a trade secret by the manufacturer; therefore the only practical information available is the type and configuration of positional parameters available through the platform APIs. The coordinates for device orientation are the same for both platforms and both can provide orientation in the form of Euler angles, rotation matrices, or quaternions; location for both platforms are based on a standard geographic reference. One difference is the Android platform provides a means in the API



to set the parameter units (*e.g.*, degrees vs. radians) whereas iOS requires the developer to write additional code for unit conversion. Comparative experiments would be needed to calibrate precision between devices in order to provide more accurate orientation representation.

Another area of differentiation also has to do mostly with hardware differences and that is in camera performance; the performance aspects are discussed in Section 4.3 but here we discuss the different ways the camera can be implemented. Both platforms enable developers to integrate a platform-standardized camera control into their applications; however these camera controls had constraints which competed with Revolver’s requirements so lower-level APIs were used to develop custom camera-controls. Both the Android and iOS platforms expose interfaces for enabling the camera and triggering the capture of a picture; in both platforms this locks the camera while it takes the picture, which involves focus and light adjustment. The locking of the camera puts constraints on the minimum capture interval in auto mode. An alternative approach (the one taken for the most recent prototype of this report) is to enable the camera in video mode and capture frames from the video stream. With this approach, there is seemingly no constraint on minimum interval as an image is always readily available. The caveat is in the quality of image captured as video-quality images are considerably less than that of images taken in an image-mode.

#### 4.1.2 Server Platforms

Both AWS and GAE provided free usage tiers for their cloud-service offerings however AWS required a method of payment prior to accessing the platform where GAE did not. AWS offered many options for run-time environment including several Linux and Microsoft variants; GAE only offered a few and they were targeted towards an implementation language rather than an operating system. These were the primary reasons for selecting GAE as the cloud-service platform for the prototype effort; otherwise both offerings seem suitable as a back-bone to Revolver. A key difference between the two is that AWS offered services with traditional file-system capabilities whereas GAE only provided specialized APIs for data persistence and access. The cloud-services market is currently very dynamic; these particular observation regarding platform capabilities may soon be obsolete as more features become available rapidly.

In the GAE platform, prototypes were developed in both Java and Python. Both variants incorporate “servlet” objects which handle the HTTP requests and responses; the Java version uses classes from the standard library and the python version uses an open-source Web framework but the interfaces are modeled nearly identically. One key difference between the two variants is the interface to the data-persistence mechanism. The Java version has separate entity objects which require attributes be placed one at a time before the entity is persisted. The Python version allows for sub-classing of the entity objects so constructors could be used for instantly creating and persisting data-objects

in their native form.

## **4.2 Effort**

This section provides some measured and observed efforts in the implementation of the various prototypes. Effort of time in researching and implementing the prototypes is estimated and code-size is analyzed; the code measurements can be found in appendix B.

### **4.2.1 Mobile Platforms**

The effort involved in implementing prototypes for each mobile-device platform was roughly equivalent despite our different levels of experience on each platform<sup>1</sup>. We believe the reason for this is the quality of documentation available for both platforms. Mobile computing is a very competitive market currently focused around user experience. To have applications with intuitive interfaces and fluid functionality it is essential that developers have a thorough understanding of the building-blocks to the platform so both vendors had taken great strides to detail and support their respective platforms. Despite the equality of effort, implementing the mobile aspect of Revolver was no trivial task. Mobile-computing platforms are fairly new and despite the effort put into documentation, knowledgeable and experienced assistance is relatively sparse. Mobile computing lacks the maturity that desktop computing has, and the platform paradigms are slightly different. It is approximated that 40 hours

---

<sup>1</sup>Objective-C approximately 10 years; Java approximately two years.

of continuous research and development time was put into each prototype. Despite the equivalence in time effort, the amount of code put into the iOS prototype was about 27% more than that of the Android prototype; this can be attributed the difference in code for controlling the chosen user-interface elements.

#### **4.2.2 Server Platforms**

The difference in effort for implementing the server prototypes for Revolver was more noticeable. Two functionally equivalent prototypes were implemented in Java and Python (for the GAE platform) and one in PHP (for local or cloud servers) which had a subset of functionality. The Python variant was 50% smaller in code size than its Java counterpart. The PHP variant had nearly 63% less code than the Python version and 81% less code than the Java version. Although the PHP version lacked code for user-interface, the relative size can be deduced given that PHP, like the other versions, uses HTML and the other versions had roughly the same amount of UI code. If the PHP version had the maximum amount of UI code between the other two versions, it would be nearly 10% smaller than the Python version and nearly 50% smaller than the Java version. The approximate effort in continuous hours of research and programming was 60 for the Java implementation, 20 for the Python implementation, and 15 for the PHP implementation. Figures 4.1 and 4.2 depict the measurements for visual comparison and Section B.2.2 in the appendices the measured code-size for each implementation.

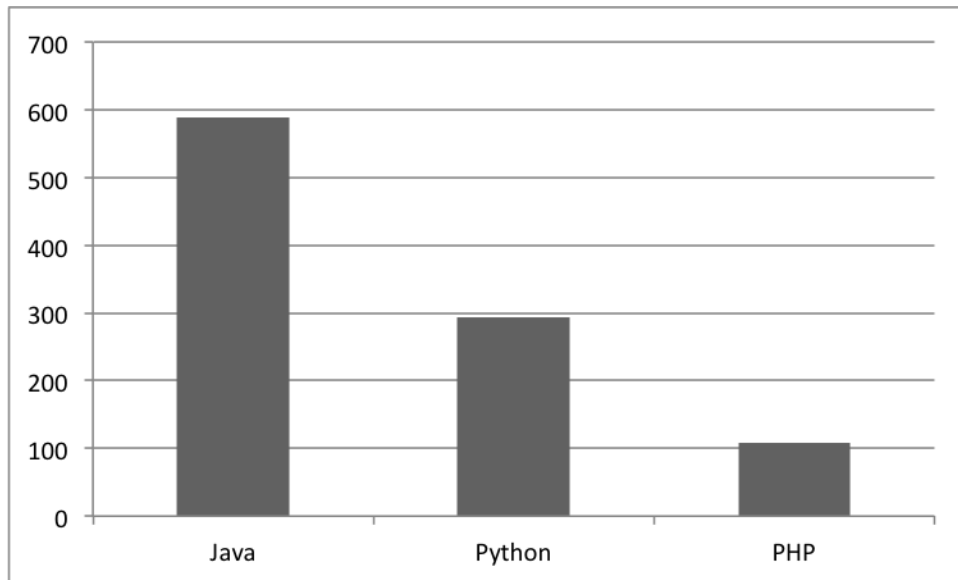


Figure 4.1: Server code size in LOC

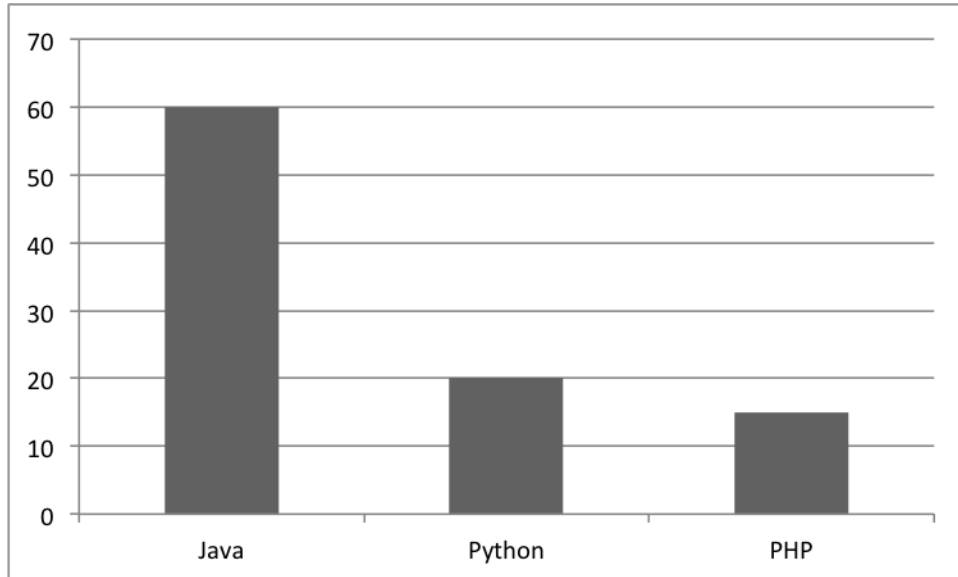


Figure 4.2: Server implementation effort in hours

Several factors went into the differences between implementations. With regards to the time effort, the Java implementation was our first experience with the GAE platform as well as Web programming and *JavaServer Pages* (JSP) so the learning-curve was steep. Given that, we feel the time-effort for the Python version would still have been less than that of the Java version (though not so accentuated) due to the object-orientated nature of the Python library for the data-store as was mentioned in Section 4.1.2. Also Python is a less verbose programming language than Java and this additionally accounts for the relative code-size. One noticeable challenge in implementing both GAE versions was in the quality of documentation available for both the Java and Python APIs; we feel this affected both time and code-size. Though the languages are mature, the environments and supporting libraries are relatively new. PHP is a mature technology with adequate documentation and a considerable experience-base and we feel this provided a slight advantage in the effort.

### 4.3 Performance

The performance of the system as a whole was sufficient for the requirements scoped within this report; only for future enhancements some performance characteristics may be re-evaluated. Most of the observed potential bottlenecks in performance would occur on the mobile platform. One such potential bottleneck was seen in the measured performance of uploading the data to the server via Wi-Fi and 3G networks; the comparison is provided in

appendix B. However though the 3G network took twice as long to upload the data than the Wi-Fi network, this did not have an impact on the application performance due to the upload process happening concurrently to the capture process. Only might this become an issue if several moments are queued for transfer and the application is shut down prematurely; the application would need to be modified to accommodate the upload process continuing in the background. This may be a non-issue as faster cellular networks are emerging and 3G will eventually become obsolete for such platforms.

Another performance consideration was that of image quality versus time to capture. As mentioned previously in the chapter, capture of high-quality images locks the camera and puts constraints on the minimum interval for automatic capture mode; experiments performed set the minimum whole-number interval to two seconds on both platforms without causing the application to freeze. To be able to capture moments faster, a different sub-architecture had to be implemented to extract images from a video stream. There was no observed lower-limit on capture interval as the camera resource asynchronously updates the video output; however the image quality was less than that taken when the camera performed in image-capture mode. Images captured regularly at highest available resolution ranged from 1.5 MB to 3.0 MB in size; images captured from the video stream ranged from 15 KB to 30 KB. Besides affecting the algorithms used to capture images, the choice in mode affects the time to upload images and the way the images are persisted from the server side. The divergence in performance characteristics would

cause a considerable bifurcation in code-bases for both the mobile and server applications and would need to be further evaluated for future iterations of Revolver.

A final notable performance aspect for this report was that of synchronization accuracy. Revolver relies on the device’s local clock to keep moments synchronized between devices. There are two challenges to this approach:

- The clocks of the devices tend to drift over time<sup>2</sup>. Often the drift is minute and the device system may periodically re-calibrate to the network time but observed drift was enough to cause moment indexes to differ between devices.
- Devices often have different master time-sources. Even if devices periodically re-calibrate their local clocks to their network time source, the networks themselves may have different time sources.

If the particular use-case involved stitching images together at a certain moment, the slip in indexes could be significant to cause anomalies in the scene. Image processing algorithms could accommodate for the potential anomalies but it is desirable to have more precision in the synchronization process to provide the servers more consistent data-sets, and mechanisms do exist for synchronizing network devices to common time sources.

---

<sup>2</sup>Experiments showed a particular device’s clock drifted by 0.6 seconds within an hour.



## Chapter 5

### Conclusions

This report presented Revolver, a distributed-systems application for capturing visual events synchronously between users. Some software applications are developed to solve concrete problems while others are designed to address more abstract problems. Revolver attempts to do both:

**The concrete problem:** Revolver is meant to enable users to have a shared perspective of the world at a common moment in time regardless of their geographical distance.

**The abstract problem:** Revolver aims to push the boundaries of existing technologies through value-creation.

The report started out by introducing the problems being addressed and comparing existing works both in research and commercially available products. It then described the system in more elaborate detail: defining the user and system roles, describing some user scenarios, and extracting some requirements for the system. The report then provided some design and implementations details for each aspect of the system; notable design artifacts and technology details were presented. Then an analysis of the prototyping effort was given

highlighting some significant experiences, quantifying the effort involved, and assessing performance aspects.

Revolver is an ambitious project. The report has detailed some technical challenges still in need of further evaluation and a particular aspect of the system (the client node) could have easily doubled the effort of this project and filled a report independently. We feel this report has presented a solid vision for Revolver in its inception, and now proposals are made in Section 5.1 to continue the work started, eventually drive it to market, and set the stage for its future generations.

## **5.1 Future Areas of Work**

This section presents proposal for the future course of Revolver. We first detail the more immediate steps that can be taken to see Revolver to market as a first-generation product. Secondly we initiate a future-vision for Revolver's functionality in the hopes that efforts for research and development along these lines are some day inspired.

Considering marketability of Revolver as the most immediate goal, we feel having a system that adequately meets the functionality outlined in the requirements of this report is essential. Though the prototype effort produced some good implementations as a starting point for the mobile and server applications, the visualization client would require the most immediate and focused attention. The ability to orient and seamlessly combine images in a 3-dimensional scene would seem to be first priority as it is the most com-

selling use-case for Revolver, followed by the ability to present the oriented images in a map application or virtual environments for visual tours. Though Processing was chosen as an initial technology to present the concept, other technologies for visualization exist such as *OpenCV* [22] or platform-native OpenGL. Processing was chosen not only for its ease of use but also for its wider platform support including modern Web browsers with Processing.js.

Once a visual client is more thoroughly developed, one challenge in testing it would be to have enough users of the application capturing a common scene simultaneously to produce a sufficient data-set in order to stitch a complete scene. If not enough data is available, sophisticated algorithms may have to be employed to fill the visual gaps; otherwise the gaps are tolerated by the user, but the most desirable compromise is to determine an acceptable threshold within which visual processing is feasible to fill in for lack of data. From a testing perspective; a utility application could be written to simulate the moment-capture functionality of Revolver and allow the user to configure which user is capturing the data and at what index. This way a single user and device could be used to simulate a multitude of users and devices at the same location and at the same time.

If users are capturing moments from significantly separate locations (i.e. different cities) it may be desirable for the visual client to automatically detect a configurable threshold and change the type of scene being rendered, perhaps automatically switching to a map visualization-mode. As was stated in the introduction “Revolver has the potential to challenge the existing paradigms

of data visualization, presentation, and analysis”; this becomes more apparent when one considers the nature of the data and the fact that the data sources could be considerably spaced from each other.

As a secondary priority for enhancement, it would be desirable to address some of the performance issues brought up in Section 4.3; specifically the issues of image quality and time synchronization. For the issue of image quality versus automatic interval, this could be addressed a number of ways. One option is to have a selectable mode for the mobile application where the user can decide the quality of image they want to capture for a given event and be restricted on the minimum interval of capture; if this mode were captured in the event object, then all users would be constrained to this mode and therefore the data produced by all users would be consistent. Alternatively this differentiation in functionality could be licensed separately in the same application or different-grade applications; one could be provided free in the marketplace to attract users and the other provided at a cost. The application would have to be modified to be able to upload the larger pictures in background processes if it is minimized, and the server could be modified to have a different set of services—and hence a different way to persist the data—based on the mobile application’s mode of operation. The chosen mode would also affect the visualization client’s performance regarding the retrieval of image data from the server, as well as image processing. We envision an initially available product supporting the rapid capture mode and lower-quality images.

The issue with time synchronization is considered of lesser priority and

may be seen as a future enhancement to an initial deployment. As stated previously in the report there are other means for accommodating the potential anomalies of images inaccurately synchronized, namely using image processing techniques. Another option is to add a layer to the mobile application stack which is responsible for synchronizing to a common time source via well-established *Network Time Protocols* (NTPs); this layer could then serve as the local clock for synchronized event capture. The approach taken should consider whether the computation should be pushed to the mobile application or the client application. Considering that mobile devices run on batteries, an additional process would only drain the charge more rapidly. On the other hand, more accurately synchronized data would enable more seamlessly integrated scenes and require less in complex processing algorithms. A more thorough treatment by users with an initial release may provide the necessary feedback to determine the course to take.

Finally we present a concept for functionality in a future generation of Revolver. Up to this point, the report has described the capture of discrete data-units in the form of images for the construction of scenes. It may some day be desirable for Revolver to capture video between users synchronously. We feel the challenges for such a use-case are deep and wide and we have provided some guiding questions in Appendix C as a starting point for exploration.

# Appendix A

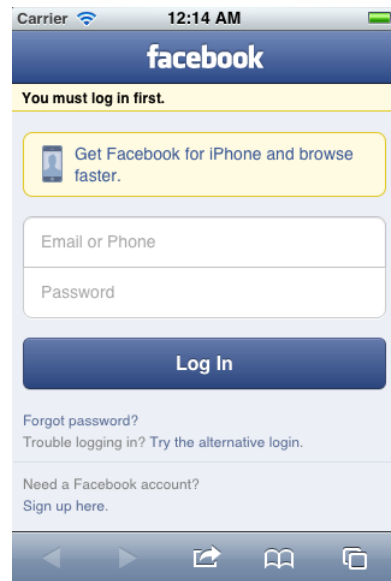
## User Interfaces

This appendix provides screen-captures of User Interface (UI) implementations from the prototype effort.

### A.1 Mobile UI

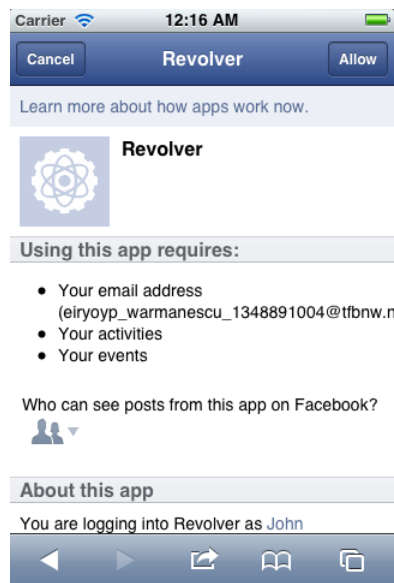


(a) Login redirect

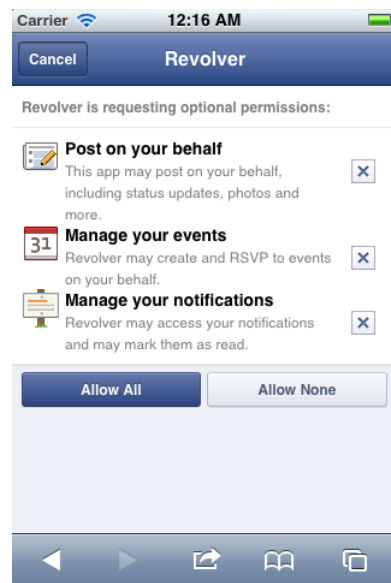


(b) Authenticaiton from browser

Figure A.1: Authentication

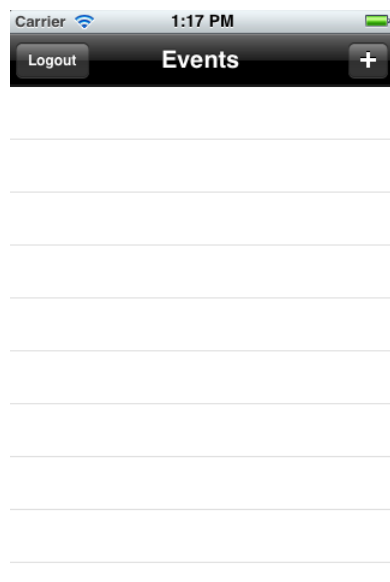


(a) Required permissions



(b) Optional permissions

Figure A.2: Permissions



(a) Event list

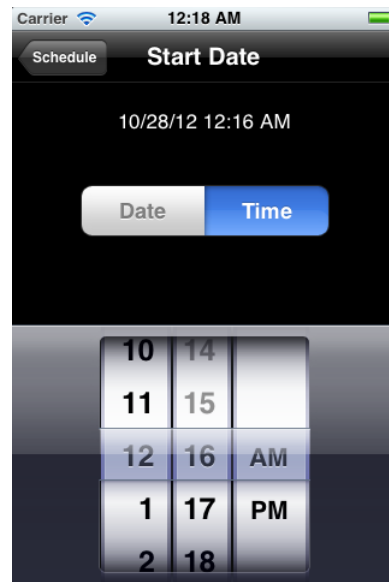


(b) Schedule details

Figure A.3: Scheduling



(a) Date picker



(b) Time picker

Figure A.4: Date and time selection

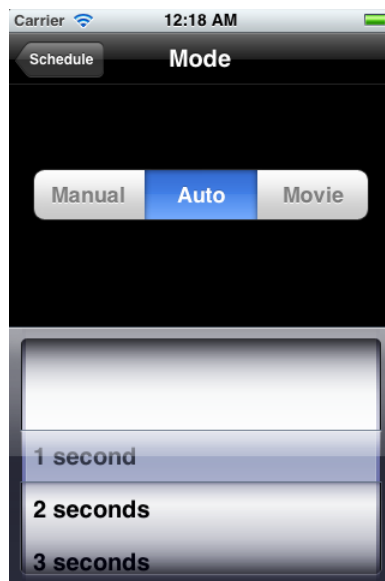


Figure A.5: Selecting mode



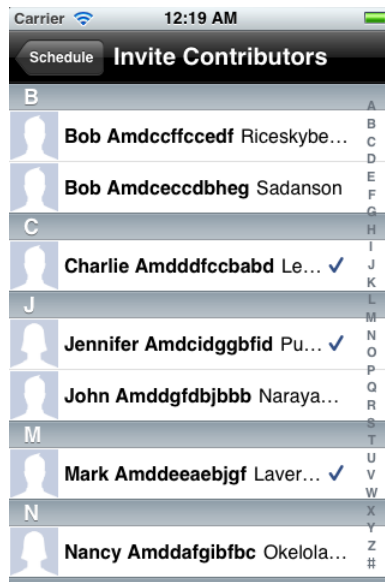
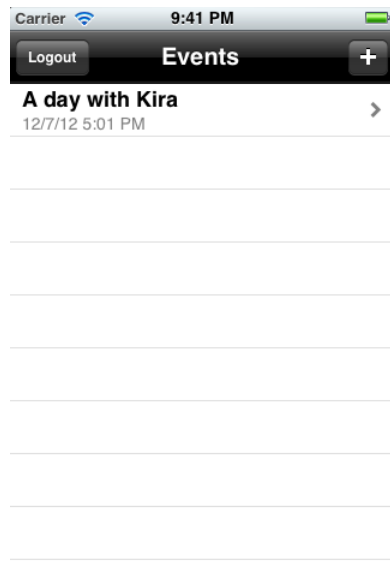
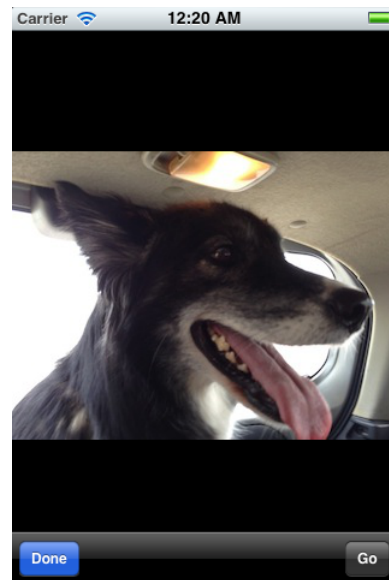


Figure A.6: Inviting contributors



(a) Scheduled event



(b) Preview and capture controls

Figure A.7: Event capture

## A.2 Server UI

### Schedule an event

**Event Name**

**Start Date**

**End Date**

**Mode** ☒ Auto ☐ Manual

**Interval**

**Contributors**

---

### My events

Id	Name	Start	End	Mode	Interval
<a href="#">35005</a>	Event_2012-11-16	2012-11-16	2012-11-17	auto	2


Figure A.8: Web scheduling

## Event Details


**Event Name** Event\_2012-11-16  
**Start Date** 2012-11-16  
**End Date** 2012-11-17  
**Mode** auto  
**Interval** 2  
**Coordinator** Mike  
**E-mail** mike.stathopoulos@utexas.edu  
**Contributors**

---


**Mike**



index: 0  
qx: 0.145043048254  
qy: -0.113917822695  
qz: 0.802206792847  
qw: 0.567846380857  
lat: 30.511425566  
long: -97.6464552061



index: 1  
qx: 0.146037909867  
qy: -0.194819565649  
qz: 0.601789741325  
qw: 0.760636199353  
lat: 30.5114644161  
long: -97.6464803518



index: 2  
qx: -0.038366518317  
qy: 0.516264287932  
qz: 0.00626868774487  
qw: -0.855546506264  
lat: 30.5115114805  
long: -97.6464864706

Figure A.9: Web viewing

### A.3 Client UI



Figure A.10: Transformed image presentation

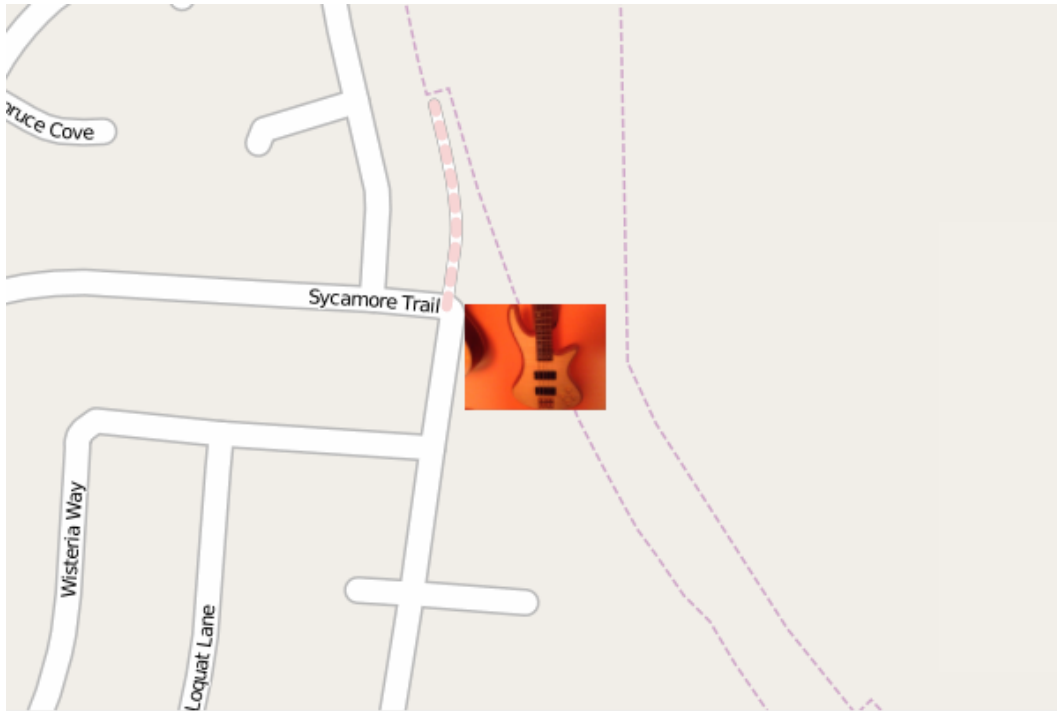


Figure A.11: Image at map location

# **Appendix B**

## **Measurements**

This appendix provides measurement results in support of the analysis of prototype implementation.

### **B.1 Communication Measurements**

This section provides a comparison of the measured round trip for communication from the time the mobile device initiates upload of a moment to the time the server responds; the comparison is between Wi-Fi and 3G networks. Table B.1 shows the mean and standard deviation for each network from a sample-size of 10 measured in seconds and Figure B.1 shows the normal distributions of each. The interval between upload was one second; upload occurred concurrently to capture operation is asynchronous thread.

Table B.1: Statistical results for Wi-Fi and 3G

	Wi-Fi	3G
<b>Mean</b>	0.786	1.352
<b>St. Dev.</b>	0.160	0.420

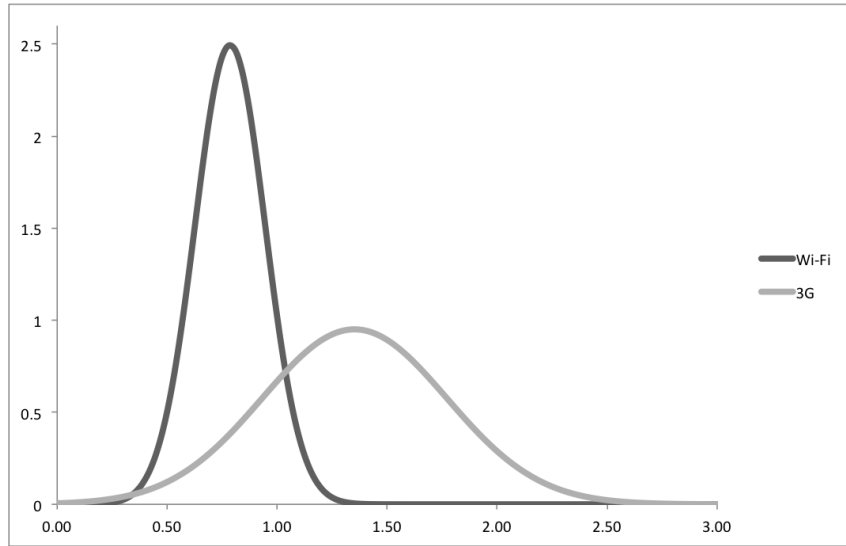


Figure B.1: Normal distributions for Wi-Fi & 3G

## B.2 Software Measurements

This section provides Lines of Code (LOC) measurements<sup>1</sup> for the various prototype implementations supporting this report. It is divided between mobile and server applications and the sub-subsections are titled in the form *year/platform/language* indicating year prototype was implemented, execution environment or platform, and primary language of implementation.

---

<sup>1</sup>Measurements performed using CLOC (<http://cloc.sourceforge.net>)

## B.2.1 Mobile Application Prototypes

### B.2.1.1 2011/Android/Java

18 text files.

18 unique files.

5 files ignored.

---

Language	files	blank	comment	code
Java	7	88	4	652
XML	6	21	0	115
SUM:	13	109	4	767

---

### B.2.1.2 2012/iOS/Objective-C<sup>2</sup>

32 text files.

32 unique files.

7 files ignored.

---

Language	files	blank	comment	code
Objective C	13	234	156	923

---

---

<sup>2</sup>Additional code for UI controllers.



C/C++ Header	12	84	85	129
--------------	----	----	----	-----

---

SUM:	25	318	241	1052
------	----	-----	-----	------

---

## B.2.2 Server Application Prototypes

### B.2.2.1 2011/GAE/Java

17 text files.

17 unique files.

7 files ignored.

---

Language	files	blank	comment	code
----------	-------	-------	---------	------

---

Java	5	51	1	375
------	---	----	---	-----

JSP	2	5	0	157
-----	---	---	---	-----

XML	3	8	6	57
-----	---	---	---	----

---

SUM:	10	64	7	589
------	----	----	---	-----

---

### B.2.2.2 2012/GAE/Python

11 text files.

11 unique files.

126 files ignored.

---

Language	files	blank	comment	code
Python	3	14	25	142
HTML	2	0	0	129
YAML	2	4	0	23
SUM:	7	18	25	294

---

### B.2.2.3 2012/MAMP/PHP<sup>3</sup>

14 text files.

14 unique files.

65 files ignored.

---

Language	files	blank	comment	code
PHP	4	12	0	75
SQL	1	5	3	27
XML	1	0	0	7

---

---

<sup>3</sup>Data transfer/persistence prototype; no UI.

SUM:	6	17	3	109
------	---	----	---	-----

-----

### B.2.3 Client Application Prototypes

#### B.2.3.1 2012/JRE/Java

6 text files.

6 unique files.

1959 files ignored.

-----

Language	files	blank	comment	code
----------	-------	-------	---------	------

-----

Java	2	23	0	142
------	---	----	---	-----

-----

SUM:	2	23	0	142
------	---	----	---	-----

-----

## Appendix C

### Future Work

This appendix provides supplementary specifications, design artifacts, or implementation details to serve as a basis for continued research and development on Revolver.

#### C.1 Mobile Nodes

##### C.1.1 Design

It was desirable to prototype a mode of the mobile-device application that could capture continuous (video) moment-data but this would have expanded the envisioned scope of the report. Figure C.1 depicts the envisioned behavior for this mode and Figure C.2 suggests an entity-relationship between the video data and meta-data. Many technical and theoretical challenges emerge when questioning this mode of behavior:

- Should there be any limits on how much video data should be captured per session?
- How much would this mode increase the drain on power-supply for devices?

- How frequently should moment meta-data be captured? What are the challenges of synchronizing the meta-data with frames of the video?
- How can the data be uploaded to the server while it is still being captured (incrementally and concurrently)? Would the upload have to wait until the session was done? If so, could the device upload in the background while the application is inactive?
- How could the visual data be presented?
- Could the audio data also be captured and presented? What kind of processing would be involved in synchronizing and combining the multiple audio sources?

We feel even more questions could be asked about this proposition, and a considerable amount of time could be spent trying to answer them all, so this concept has been shelved for potential future effort.

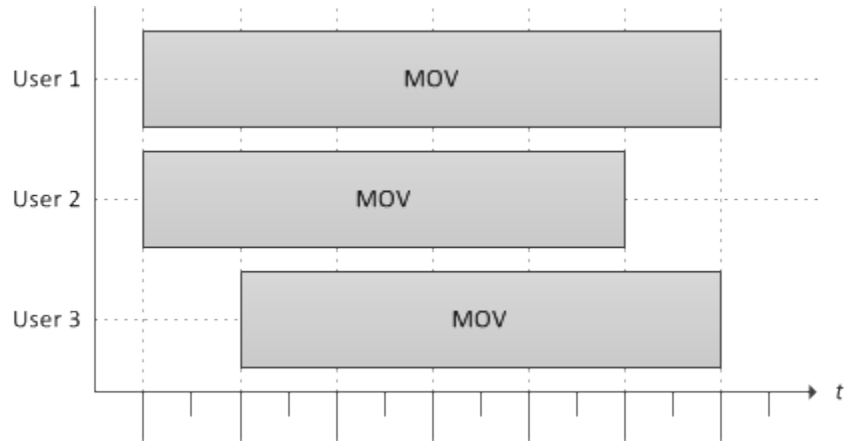


Figure C.1: Movie capture mode behavior

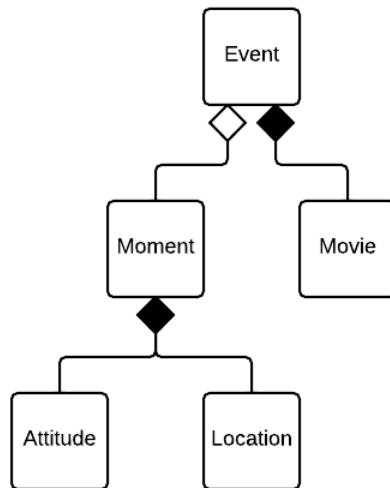


Figure C.2: Event entity model (movie capture mode)

## Bibliography

- [1] Amazon Web Services. <http://aws.amazon.com>, 2012.
- [2] OAuth. <http://oauth.net>, 2012.
- [3] E. Ackerman and E. Guizzo. 5 technologies that will shape the web. *IEEE Spectrum*, June 2011.
- [4] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, 2008.
- [5] Apple Inc. WWDC 2010 session videos. <https://developer.apple.com/videos/wwdc/2010>, 2010.
- [6] Apple Inc. iOS developer library. <https://developer.apple.com/library/ios>, 2012.
- [7] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice, 2nd Edition*. Addison-Wesley Professional, 2003.
- [8] Joshua Bloch. *Effective Java*. Addison-Wesley, 2008.
- [9] Tushar D. Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: an engineering perspective. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. ACM, 2007.

- [10] Facebook Inc. Facebook SDK reference. <https://developers.facebook.com/docs/sdks>, 2012.
- [11] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 2004.
- [12] Ben Fry. *Visualizing Data*. O’Reilly Media, Inc, 2008.
- [13] Ben Fry and Casey Reas. Processing. <http://processing.org>, 2012.
- [14] Ben Fry and Casey Reas. Processing.js. <http://processingjs.org>, 2012.
- [15] Hector Garcia-Molina, Jeffrey D Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Pearson Prentice Hall, 2009.
- [16] Vijay K Garg. *Concurrent and Distributed Computing in Java*. IEEE Press ; Wiley-Interscience, 2004.
- [17] Noah Gift and Jonathan Saggau. Connecting Apple’s iPhone to Google’s cloud computing offerings. <http://www.ibm.com/developerworks/web/library/wa-aj-iphone>, 2009.
- [18] Ron Goldman. Understanding quaternions. *Graphical Models*, March 2011.
- [19] Google Inc. Android developers. <http://developer.android.com>, 2012.



- [20] Google Inc. Google App Engine. <https://developers.google.com/appengine>, 2012.
- [21] David Grosvenor and Stephen Cheadle. Synchronized cameras with auto-exchange. Patent, 2006. US 7139018.
- [22] Intel Corporation, Willow Garage, and Itseez. OpenCV. <http://opencv.org>, 2012.
- [23] James F Kurose and Keith W Ross. *Computer Networking: A Top-down Approach*. Addison-Wesley, 2010.
- [24] Dean Leffingwell and Don Widrig. *Managing Software Requirements: A Use Case Approach*. Addison-Wesley, 2003.
- [25] Michelle Maltais. New Jersey ACLU app gives new meaning to 'police tape'. *Los Angeles Times*, July 2012.
- [26] A. Marcus and O. Marques. An eye on visual sensor networks. *Potentials, IEEE*, April 2012.
- [27] Peter Mell and Timothy Grance. SP 800-145: The NIST definition of cloud computing, 2011.
- [28] Microsoft Corporation. Photosynth. <http://photosynth.net>, 2012.
- [29] Till Nagel. Unfolding Maps. <http://unfoldingmaps.org>, 2012.
- [30] Occipital Inc. <http://occipital.com>, 2012.

- [31] Anthony Ralston, Edwin D. Reilly, and David Hemmendinger, editors. *Encyclopedia of Computer Science*. John Wiley and Sons Ltd., 2003.
- [32] Alex Rodriquez. RESTful web services: the basics. <https://www.ibm.com/developerworks/webservices/library/ws-restful>, 2008.
- [33] N. Snavely, S. M. Seitz, and R. Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, November 2008.
- [34] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3D. In *SIGGRAPH Conference Proceedings*. ACM Press, 2006.
- [35] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.
- [36] International Telecommunication Union. ITU world telecommunication/ICT indicators database. <http://www.itu.int>, 2012.
- [37] Aaron Vegh. *Web Development with the Mac*. Wiley, 2010.
- [38] Richard S. Wright, Benjamin Lipchak, and Nicholas Haemel. *OpenGL SuperBible*. Addison-Wesley Professional, 2007.

## Vita

Michael Stathopoulos received a Bachelor of Science degree in Manufacturing Engineering Technology with a specialization in Computer Integrated Manufacturing from Ball State University in 1998. For six years upon graduation, he worked as an engineer on manufacturing automation applications primarily in automotive manufacturing and material handling domains. In 2005 he went to work for a not-for-profit trade consortium, leading efforts in the development of international standards for digital communications and integrated system architectures in the industrial process automation domain. In 2010, he entered the Graduate School at the University of Texas at Austin and enrolled in the Software Engineering Program. He has long been fascinated with the art and science of software and its impact on human history.

Permanent address: `mike.stathopoulos@utexas.edu`

This report was typeset with  $\text{\LaTeX}^\dagger$  by the author.

---

<sup>†</sup> $\text{\LaTeX}$  is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's  $\text{\TeX}$  Program.