CONNECTED CAMPUS-ORIENTATION PROJECT

A Thesis

Presented in Partial Fulfillment of the Requirements for
the Degree Master of Science in the
Graduate School of The Ohio State University

By

Ravishankar Subramaniyan, B.E.
Graduate Program in Computer Science and Engineering
The Ohio State University

2010

Master’s Examination Committee:
Dr. Dong Xuan, Adviser
Dr. Feng Qin
ABSTRACT

Presently, The Ohio State University (OSU) routinely uses email and websites to communicate with students. These approaches do not provide real-time communication and they also contribute to a student’s current information overload. Under the present system students cannot receive immediate and cost-effective notification of emergencies or upcoming campus events.

One instance of the above problem occurs while new and prospective students are presented with a great deal of information during orientation programs on campus. This is burdensome for tour guides, who must remember all this information, and is also prone to error. For instance, they might lead a group of students past the recently renovated library but might forget to mention the renovations.

Connected Campus (CC) provides a new platform that enables free, instantaneous communication. CC sends push notifications via OSU Wireless to students’ mobile devices, each of which is uniquely tied to an OSU username (name.n), and is free for use by any OSU affiliate. CC comprises of a mobile device application, which receives the notifications, and a web application, which transmits them.

CC thus overcomes the limitations of existing university communication systems and also satisfies the need for a cost-effective, real-time communication platform. From a student perspective, it is desirable to simply receive pushed information on a mobile device
than having to reach out and pull the information from email or the web. The pushed information will be transmitted as short notifications to quickly capture student’s attention and deliver content.

Orientation leaders can distribute CC-equipped mobile devices to new and prospective students. As they pass a particular location on campus with OSU Wireless access, the CC server can send push notifications to student’s devices informing them about that building’s purposes, policies, and upcoming events, based on the location details sent from the device. This reduces the burden on guides and possible errors. For example, when students walk by the main library, it can transmit notifications about the recent renovations, which guides might have forgotten to mention.

This thesis deals with designing an iPod-based application prototype for the orientation problem mentioned above. The reason for selecting the iPod touch device is because of the popularity of these devices among students.
ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my advisor Dr. Dong Xuan for his continuous support and guidance throughout this project. This research could have not been completed without his advice, patience and encouragement.

I would also like to thank my committee member Dr. Feng Qin for his time and support. I would also like to thank Mr. Adam C. Champion and Mr. James Walton for their inputs and guidance.

Last and most of all, I am thankful to my parents and my family and friends for their encouragement, help, love and support to help me make my dreams come true.
VITA

Jun 2002-May 2006  B.E. Computer Science & Engineering
  CEG, Anna University, India

Jul 2006-Aug 2008  Senior Software Engineer
  Wipro Technologies, India

Apr 2008-present  Graduate Research Associate
  The Ohio State University

FIELDS OF STUDY

Major Field: Computer Science and Engineering
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>iv</td>
</tr>
<tr>
<td>Vita</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td>viii</td>
</tr>
</tbody>
</table>

Chapters:

1. **Introduction** .................................. 1
2. **Orientation Application** ..................... 3
3. **System Architecture** .......................... 9
   3.1 Architecture and Design Considerations .... 9
   3.2 Message-Driven Model ......................... 13
   3.3 Data Model .................................... 15
4. **System Implementation** ........................ 17
   4.1 Front-end/Client iPod Application ......... 17
      4.1.1 Core Location Framework ................. 18
      4.1.2 MapKit Framework ........................ 19
      4.1.3 BlockId Calculation ...................... 20
      4.1.4 Token generation ......................... 21
      4.1.5 SOAP Communication ...................... 23
      4.1.6 Application Shutdown .................... 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2 Back-end Web/Server Application</td>
<td>23</td>
</tr>
<tr>
<td>4.2.1 Event List Retrieval</td>
<td>24</td>
</tr>
<tr>
<td>4.2.2 Web Services</td>
<td>24</td>
</tr>
<tr>
<td>4.3 Push Notification Services</td>
<td>26</td>
</tr>
<tr>
<td>4.3.1 Urban Airship</td>
<td>26</td>
</tr>
<tr>
<td>5. System Workflow</td>
<td>30</td>
</tr>
<tr>
<td>6. Conclusions and Future Work</td>
<td>34</td>
</tr>
<tr>
<td>6.1 Conclusions</td>
<td>34</td>
</tr>
<tr>
<td>6.2 Future Work</td>
<td>35</td>
</tr>
<tr>
<td>Bibliography</td>
<td>37</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Orientation Application Workflow</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Push notification from CC Server to the client application via Urban Airship</td>
<td>7</td>
</tr>
<tr>
<td>3.1</td>
<td>System Architecture</td>
<td>10</td>
</tr>
<tr>
<td>3.2</td>
<td>Device - APNS Authentication</td>
<td>13</td>
</tr>
<tr>
<td>3.3</td>
<td>Sequential Flow of Data</td>
<td>14</td>
</tr>
<tr>
<td>3.4</td>
<td>ER Diagram of the Database Schema</td>
<td>15</td>
</tr>
<tr>
<td>4.1</td>
<td>Token Generation</td>
<td>22</td>
</tr>
<tr>
<td>4.2</td>
<td>Registration Message Structure</td>
<td>28</td>
</tr>
<tr>
<td>4.3</td>
<td>POST Message Structure</td>
<td>28</td>
</tr>
<tr>
<td>5.1</td>
<td>Device showing the Current location Information</td>
<td>31</td>
</tr>
<tr>
<td>5.2</td>
<td>Device sending updates to the CC server</td>
<td>32</td>
</tr>
<tr>
<td>5.3</td>
<td>Event Notification popping up on the device</td>
<td>33</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

Today students cannot cheaply receive immediate notifications of emergencies or campus events and are overloaded with lot of information in the form of email and websites. In emergencies, OSU Public Safety must send text messages to students’ mobile phones, which costs thousands of dollars and the messages do not always reach students in time. Also during orientation programs, new and prospective students are presented with excessive information that tour guides must manually provide, which is burdensome.

Connected Campus (CC) simply pushes information to students that they need to know (such as emergency notifications) and elect to receive (such as upcoming events). It is intended to be a free platform that any OSU affiliate can use. The platform has two components:

- A Mobile device application (mobile application for short) that receives notifications.
- A Web application that sends notifications and manages events.

Apart from the above two major components, the project also uses third party components like Apple Push Notification Service (APNS) [1] and Urban Airship (UA) [2]. Subsequent chapters provide a detailed explanation of these components.
CC uses mobile devices to receive notifications for the following reasons:

- Devices can access the Internet via Wi-Fi, which is free on campus.
- Devices can uniquely correspond to name.n, by mapping it to their device ids.
- The handheld form factor of these devices make them easy to carry in a pocket, unlike desktop and laptop computers.
- Devices can receive push notifications.

The Orientation application which is the major focus of this thesis is a part of the Connected Campus project. This application notifies the user with the events closer to their current geographical location. The CC server on obtaining the location details of the device, identifies the list of events in the neighboring locations and transmits them back to the device using APNS and UA. We implement the Orientation application on iPod touch devices due to their popularity among students and cost-effectiveness.

The rest of the thesis is organized as follows:

Chapter 2 provides a brief overview of the Orientation application.

Chapter 3 describes the application’s system architecture, including the design considerations, message flow, and the data model.

Chapter 4 provides the system implementation details and the various frameworks used in order to achieve the functionality.

Chapter 5 explains the workflow of the implementation with the results achieved.

Concluding remarks and directions for future work are given in Chapter 6.
CHAPTER 2

ORIENTATION APPLICATION

The Orientation application focuses on notifying the user about events close to his geographical location. Based on the current location of the user, events occurring in a 100 m radius with the user’s location as center is sent to the user’s device. The application also notifies the user with more important deadlines and events occurring on the campus. Figure 2.1 illustrates this.

The major components of the Orientation application are the Front-end/Client application and the Back-end Web/Server application. The application also uses third party components like the APNS and UA in order to send push notifications to the respective devices. Chapter 3 provides further explanation of the architecture and design considerations of the Orientation application.

An application on an iPhone or iPod touch is often only a part of a larger application based on the client-server model. The client side of the application is installed on the device; the server side of the application has the main function of providing data to its many client applications. A client application occasionally connects with its server and downloads the data that is waiting for it. Email and social-networking applications are examples of this client-server model.
But what if the application is not connected to its server or even running on the device when the server has new data for it to download? How does it learn about this waiting data? Push notifications are a solution to this dilemma. A push notification is a short message that a server has delivered to a device; the device, in turn, informs the user of a client application that there is data to be downloaded. If the user enables/accepts this feature and the application is properly registered, the notification is delivered to the device and possibly the application.

Apple push notifications—also known as remote notifications—lets users know when applications on their iPhone and iPod touch devices have data waiting for them on their servers. They use a persistent IP connection with the device. When a device receives a notification specific to an application and that application isn’t running, it notifies the user through an alert message, a distinctive sound, or a number badging the application (or some
combination of these). The user may then launch the application, which then downloads the data from its server, which is also known as its provider. Introduced in iPhone OS 3.0, push notifications serve the same purpose that a background application would on a desktop computer, excepting multi-tasking. Unlike desktop OSes, iPhone OS doesn’t support multi-tasking.

The Orientation application is mainly developed for smartphones which have the capability to use Wi-Fi to connect to the Internet. For reasons given in the Introduction, we have chosen iPhone/iPod as our target devices, specifically iPod touch devices. The iPhone/iPod devices use Google Maps APIs to retrieve their current location information, which basically consists of the latitude and longitude details of the current geographical location.

There are actually three ways available for the iPhone to determine our location: GPS, cell tower triangulation, and Wi-Fi Positioning Service (WPS). GPS is the most accurate, followed by cell towers and Wi-Fi. GPS reads microwave signals from multiple satellites to determine the current location. Cell tower triangulation determines the current location by doing a calculation based on the locations of the cell towers in the phone’s range. Cell tower triangulation can be fairly accurate in cities and other areas with a high cell tower density but becomes less accurate in areas where there is a greater distance between towers.

The last option, WPS, uses the IP address from an iPhone or iPod’s Wi-Fi connection to make a guess at the location by referencing a large database of known service providers and the areas they service. iPhone and iPod Touch uses Skyhook’s WPS as the primary location engine for Google Maps and other applications. Skyhook Wireless (formerly known as Quarterscope) is a Boston-based company that has developed a technology for determining geographical location using Wi-Fi as the underlying reference system. Using the MAC addresses of nearby wireless access points and proprietary algorithms, WPS can determine
the position of a mobile device within 20-30 meters. It provides service similar to GPS without GPS hardware and can also integrate with GPS-enabled devices to provide hybrid positioning. WPS is imprecise and can be off by many miles. iPhones can use two or three of these localization technologies, while the iPod touch can only use Wi-Fi, but it beats nothing.

When within range of a few Wi-Fi hotspots, the iPod can triangulate its position to show their current location and hence because of this triangulation the location (latitude, longitude) values retrieved are not 100 percent accurate. The Front end application is thus programmed to retrieve the location to the nearest 100 m range. The devices on retrieving the latitude and longitude communicate them to the CC server over HTTP. Simple Object Access Protocol (SOAP) is used to achieve this communication. In order to lessen battery utilization load on the devices and to avoid frequent updates to the server, updates are done only once a minute and can also be updated only when there is movement of the user by more than 10 m. This also avoids frequent reverse geocoding, thus efficiently utilizing the limited amount of reverse geocoding capacity. These parameters can be changed based on real-time measurements.

SOAP is a simple XML-based protocol to let applications exchange information over HTTP. More simply, SOAP is a protocol for accessing a Web service. SOAP provides a way to communicate between applications running on different operating systems with different technologies and programming languages. The Front end application thus sends latitude, longitude and timestamp values to the CC server using SOAP.

The CC server hosts the database and the Web services. The database contains records about the events and their location details and it requires user authentication in order to create/modify the records.
On receiving the latitude and longitude values from the iPod device, the CC server identifies the events that are in close proximity to the user. This is achieved by dividing the complete OSU campus area into 100 m × 100 m square blocks and each of these blocks are labeled with unique numbers. The reason for selecting this block size is to reduce cases of improper events selection. The retrieved latitude and longitude values are mapped to their corresponding Block Number by the client application and then sent to the CC server. The server uses this Block Number to identify the events that are in close proximity to the device current location. This list of events along with the some of the high priority events are selected randomly and sent back to the device using Urban Airship as push notifications. Urban Airship provides an interface that facilitates to implement the push notifications.

The CC server connects with APNS through a persistent and secure channel while monitoring incoming data intended for their client applications and this is achieved by using the Urban Airship APIs. When new data for an application arrives, the CC server prepares and sends a notification through Urban Airship to APNS, which pushes the notification to the target device. The device finally receives the list of events that occur in its close proximity. Figure 2.2 illustrates this process.

![Figure 2.2: Push notification from CC Server to the client application via Urban Airship](image)
In the next chapter we shall understand the overall system design of the Orientation application.
3.1 Architecture and Design Considerations

The architecture shown in Figure 3.1 gives a brief overview of the high level system structure of the Orientation project. As explained in the model, the iPod touch devices have the Orientation Client application installed. The Orientation application uses the Google Maps APIs and the MapKit framework to obtain the current location parameters of the device. The Google Maps API in turn uses the Wi-Fi connection to determine the latitude and longitude values of the current location as explained in Chapter 2. The client application refreshes the location once in a minute and the updated values are sent to the Connected Campus server using the SOAP framework.

The CC server implements the Web services and exposes the UpdateLocation Web method, to the client application. It uses the JBoss Application Server (AS) to implement the Web services and uses MySQL to maintain the database. Upon receiving the updated location values of the device, the server updates the location values into the device table only if the particular device record already exists in the database. Only on successful update is the event information in the neighboring location of the device generated. In order to identify the neighboring locations of the device, the device also sends a value called
Figure 3.1: System Architecture
the 'blockId’ along with the updated location values. The server uses this information in
determining the events in the neighboring locations.

The CC Server uses the registration and push interface methods to interact with Urban
Airship. Urban Airship provides these interfaces for the CC server to send the notifications
to the device via the APNS. Urban Airship provides a cloud-based computing service to
send push notifications and helps to avoid APNS perception of denial of service, which
may arise upon sending repeated push notifications from the CC server. UA uses Amazon
Elastic Cloud Compute (EC2) to achieve scalability. Amazon EC2 is a web service that
provides resizable compute capacity in the cloud. It is designed to make web-scale com-
puting easier. Amazon EC2 works in conjunction with Amazon Simple Storage Service
(Amazon S3), Amazon SimpleDB and Amazon Simple Queue Service (Amazon SQS) to
provide a complete solution for computing, query processing and storage across a wide
range of applications.

REpresentational State Transfer (REST) is a key design idiom that embraces a state-
less client-server architecture in which the web services are viewed as resources and can
be identified by their URLs. Web service clients that want to use these resources access a
particular representation by transferring application content using a small globally defined
set of remote methods that describe the action to be performed on the resource. REST is an
analytical description of the existing web architecture, and thus the interplay between the
style and the underlying HTTP protocol appears seamless. The CC server uses the REST-
ful APIs HTTP PUT and HTTP POST to communicate to the Urban Airship interface. The
Urban Airship Register interface is used to identify whether the target application is still
installed on the requested device. This is achieved by using the APNS feedback service de-
scribed as follows. APNS maintains a list of inactive devices - devices that do not have the
target application installed. Only upon a successful Registration call from the UA interface is the Push method invoked. Urban Airship establishes a secure socket connection to the APNS to communicate the information from the CC server.

After authenticating the information from the Urban Airship using its public key, APNS sends the push notifications to the device using a previously established TLS peer-to-peer authentication created for generating a token for the device. A device initiates a TLS connection with APNS, which returns its server certificate. The device validates this certificate and then sends its device certificate to APNS, which validates that certificate. An iPhone application must register to receive push notifications, typically right after it is installed on a device. Upon receiving a registration request from an application, iPhone OS connects with APNS and forwards the request. APNS generates a device token using information contained in the unique device certificate. The device token contains an identifier for the device’s session with APNS. It then encrypts the device token with a token key and returns it to the device. The device returns the device token to the requesting application as an NSData object.

Every notification the CC server sends to APNS via Urban Airship for delivery to a device must be accompanied by the device token it obtained from the client application on that device. APNS decrypts the token using the token key, thereby ensuring that the notification is valid. It then uses the device ID contained in the device token to determine the destination device for the notification. An authentication process similar to the Device-APNS happens at the APNS-Urban Airship side.
3.2 Message-Driven Model

The sequence diagram in Figure 3.3 shows the sequence of actions that take place during the complete cycle of the Orientation application.

The four main entities of the Orientation application are the iPod touch device, Connected Campus server, the Urban Airship and APNS. The diagram explains the flow of messages across the different entities and the consequences thereof.

The diagram also explains how the Orientation application connects to the server and the different possible scenarios that might arise based on the data flow across the different entities - iPod touch, CC server, Urban Airship and APNS.
Figure 3.3: Sequential Flow of Data
3.3 Data Model

The Web application uses a database schema which is built from the following ER diagram shown in Figure 3.4.

![ER Diagram of the Database Schema](image)

Figure 3.4: ER Diagram of the Database Schema

The data model is designed in such a way that all the necessary design constraints are handled efficiently. Any Event has associated long and short descriptions and it may occur at multiple locations. The reason for maintaining two different descriptions is due to
restrictions on push notification message length. The Event Occurrence has a relationship with the Place table. It contains information about the date and time of the event and also the Block ID information from the Place-Event Occurrence mapping. The Place Table maintains a location’s latitude, longitude co-ordinates, Block ID and also information about the location. The User and Group tables maintains login information in order to access the web application.

The Place table is preloaded with all the building information available in the OSU campus, containing their respective latitude, longitude and Block ID values.

JBoss AS provides an interface for the administrator to Create, Read, Update and Delete the records of the database. In order to create an Event, the administrator will first be authenticated and later will be allowed to access/modify the records. To create an Event, the administrator has to enter the place where the event occurs by choosing from a list. This exhaustive list contains all the buildings in the OSU Columbus campus, including the Oval and South Oval. The administrator also fills out the Event description, date and time and sets the priority of the Event. The events can be priority events or non-priority events. An Event is a priority event when it is applicable for the complete student community. Events such as ‘Career Fair’ or ‘Homecoming Parade’ can be chosen as priority events.

The web interface would allow anyone to view the list of Events, but only an administrator will be allowed to create, modify or delete the Events. An event can even occur at many places on the same day and hence the data model was designed to have the Event and Event Occurrences as separate tables.

The next chapter describes the system implementation of the Orientation application.
CHAPTER 4

SYSTEM IMPLEMENTATION

The CC Orientation application has two parts: a Front-end Client iPod application and a Back-end Web application that is accessible online at http://champion.cse.ohio-state.edu. The iPod/iPhone application registers with Apple Push Notification Service (APNS) to receive push notifications and APNS creates a persistent network connection to the mobile device. The application immediately connects to the CC server updating its location details.

The web application, which may be accessed from Apple mobile devices or other computer systems, authenticates OSU users and manages the event details. To avoid overburdening the APNS servers, each CC Orientation user may only run the iPod/iPhone application and receive push notifications on one Apple mobile device that is uniquely tied to his name.

Upon receiving location details, the Web application notifies the devices about the events closer to it via APNS.

4.1 Front-end/Client iPod Application

The Client iPod application uses Google Maps APIs to render the current location of the user. It obtains the latitude and longitude of the user’s current location within a range of 100 m in most cases. The Client iPod application uses a location manager object to retrieve the most recent location data.
4.1.1 Core Location Framework

iPhones and iPod touch devices have the ability to determine their location using a framework called Core Location. As discussed in Chapter 2, there are three ways to determine location in iPhone devices. Core Location actually decides which method to use based on what is available to the device and the desired accuracy. Here, the Core Location framework uses WPS to determine the location of iPod devices.

To use Core Location, we simply need to create a location manager instance and ask it to start sending updates. It can provide position, altitude, and orientation, depending upon the device’s capabilities. A CLLocationManager object is created to deliver location events. First, we create an instance of the CLLocationManager object, assign a delegate object to it, configure the desired accuracy and distance filter values, and call the startUpdatingLocation method. The desired accuracy is set by appropriately initializing the property `desiredAccuracy` with values like `kCLLocationAccuracyBest`, `kCLLocationAccuracyNearestTenMeters`, etc. The application initializes this property to `kCLLocationAccuracyNearestTenMeters`. The location service returns an initial location as quickly as possible, returning cached information when available.

Whenever the device location is updated or when the startUpdatingLocation is called Location Manager notifies the Client iPod application by calling `didUpdateToLocation:fromLocation:` delegate method and updates `CLLocation` object that defines the current location of the iPhone/iPod devices. The `CLLocation` object has the properties, latitude and longitude that provides the geographic location of the devices to the specified degree of accuracy.
4.1.2 MapKit Framework

The MapKit framework embed a fully functional map interface into the application window. The map support provided by this framework includes many of the features normally found in the Maps application. It uses Google Maps services to provide map data. An MKMapView object provides an embeddable map interface similar to the one provided by the Maps application. We use this class as-is to display map information and to manipulate the map contents from our application. We can center the map on a given coordinate, specify the size of the area we want to display, and annotate the map with custom information.

In order to specify the region of the map to be displayed, the region and span properties are set accordingly. A region is defined by a center-point and the span defines how much of the map should be visible at the given point, based on the set zoom level. Specifying a large span results in the user seeing a wide geographical area and corresponds to a low zoom level and vice-versa. The MapKit framework includes built-in support for displaying the users current location on the map and this is achieved by setting the showsUserLocation property to Yes. We can also add an annotation object such as dropping a pin at the current user location on the MapView.

Annotations are pieces of map content that are defined and layered on top of the map itself. The MapKit framework implements annotations in two parts: an annotation object and a view to display that annotation. An annotation object is any object that conforms to the MKAnnotation protocol. When an annotation needs to be displayed on screen, the map view is responsible for making sure the annotation object has an associated annotation view. It does this by calling the the mapView:viewForAnnotation: method of its delegate object when the annotation’s coordinate is about to become visible on the screen. Before creating a new annotation view object, the delegate’s mapView:viewForAnnotation: method should
always check its reuse queue to see if an existing view is available by calling the map view’s `dequeueReusableAnnotationViewWithIdentifier:` method. If that method returns a valid view object, the view can be reinitialized and returned; otherwise, a new view object should be created and returned.

The application also looks up the placemark information such as the street address, city, state, and country associated with the retrieved coordinate values using `MKReverseGeocoder` class. To start a reverse geocoder query, the application creates an instance of the `MKReverseGeocoder` class, assigns an appropriate object to the delegate property, and then calls the start method. If the query completes successfully, the delegate method receives an `MKPlacemark` object with the results. Placemark objects are themselves annotation objects—i.e., they adopt the `MKAnnotation` protocol—and thus we can add them to our map view’s list of annotations.

4.1.3 BlockId Calculation

The entire OSU campus area is divided into 100 m × 100 m square blocks and each of these blocks is labeled with unique numbers. The reason for selecting this block size is to reduce cases of improper event selection. The blockId for any given (latitude, longitude) pair is determined as follows:

To a large extent, the complete OSU campus area is covered by defining a reference (latitude, longitude) pair whose top left-most point is at (40.02°N, −83.05°W) and another pair whose bottom right-most point is at (39.99°N, −83.005°W). This area comprises of 34 × 39 blocks, each of which is area 100 m × 100 m square. The blocks are given a unique number in row-major order starting with the top left-most block whose value is 0.
The blockIds of all other (latitude, longitude) pairs are calculated from the reference point \((40.02^\circ N, -83.05^\circ W)\), whose blockId is 0. The blockId is calculated by measuring the horizontal and vertical distances of the point from the reference point. These distances are then divided by 100 to get a unique \((x,y)\) pair. A unique blockId is derived from this \((x,y)\) pair as \(39\times x+(y+1)\). The horizontal and vertical distance between latitude, longitude pairs is calculated using the Core Location API method \(getDistanceFrom\). The method measures the distance between the two locations by tracing a line between them that follows the curvature of the Earth.

For example, the location co-ordinates of Dreese Labs is \((40.002354^\circ N, -83.016063^\circ W)\). The horizontal distance between this location and the reference point would be the distance between the latitude, longitude pairs- \((40.02^\circ N, -83.05^\circ W)\) and \((40.02^\circ N, -83.016063^\circ W)\) and the vertical distance would be the distance between \((40.02^\circ N, -83.05^\circ W)\) and \((40.002354^\circ N, -83.05^\circ W)\). The horizontal and vertical distance are then divided by 100 to obtain the \((x,y)\) pair \((19,28)\). The unique blockId of Dreese Labs would then be \(39\times 19+(28+1)\), which equals 770.

There are two main reasons for using the blockId. First, it eliminates improper event selection. Second, it helps in the retrieval of events in close proximity, by just selecting the events in the neighboring block numbers. This also ensures that no event closer to the user location is avoided.

### 4.1.4 Token generation

As explained in the previous chapters, an iPhone application must register to receive push notifications; it typically does this right after it is installed on a device. iPhone OS
receives the registration request from an application, which connects with APNS, and forwards the request. APNS generates a device token using information contained in the unique device certificate.

The Client application invokes the method `RegisterforRemoteNotificationTypes`, which on successful completion invokes the callback function `didRegisterForRemoteNotificationsWithDeviceToken`. This provides the device token after registering the device with APNS server. Note that the device token uniquely identifies the session with APNS, not the device. Token generation is illustrated below in Figure 4.1.

![Figure 4.1: Token Generation](image)

The device identifier is obtained from the `uniqueIdentifier` property of the `UIDevice` class. This information is then passed to the Web services hosted on the CC server using SOAP. The Client application binds these parameters into the SOAP message and transports them via HTTP.
4.1.5 SOAP Communication

Upon retrieving the coordinate values the devices communicate them to the CC server over HTTP. Simple Object Access Protocol (SOAP) is used to achieve this communication. SOAP is fundamentally a stateless, one-way message exchange paradigm for XML-based information. SOAP enables client applications to easily connect to remote services and invoke remote methods.

The SOAP message has the parameters Latitude, Longitude, blockId of the (latitude, longitude) pair, the device token, the device identifier and the timestamp generated for the device embedded in it. Thus whenever there is change in the latitude and longitude values change or the application starts, it communicates the updated values to the Web services hosted on the server at http://champion.cse.ohio-state.edu/ using SOAP. These values are embedded into the SOAP header and is communicated to the CC webserver using the HTTP-POST method with the encoding in place. The SOAP Request is sent using a NSURLConnection object with the target location as http://champion.cse.ohio-state.edu:8080/connectedcampus/cc.

4.1.6 Application Shutdown

When the application is closed (by hitting the home button), it informs the server to stop sending the event notifications, as there will be no location updates to trigger these notifications.

4.2 Back-end Web/Server Application

The CC Web application is implemented on a Ubuntu 9.10 Linux server using Java Enterprise Edition (EE) 5 with Apache 2.2 and MySQL 5. The application is deployed on
JBoss AS 5.1. It extensively uses the JBoss Seam 2.2 Web framework (Seam for short), which “bridges together” JavaServer Faces (JSF) 2.0, Facelets, RichFaces, and Java Persistence API (JPA). We chose Java EE with Seam as our implementation platform due to its ability to rapidly generate Web applications with create, retrieve, update, and delete operations and Seams enhancements to Java EE. Many Connected Campus use cases can be implemented as such operations.

4.2.1 Event List Retrieval

Based on the device’s current blockId and block’s position, its neighboring blocks are determined. If the block is located in one of the corners of the campus area, it has only minimal neighboring blocks. This can reach a maximum of 9 blocks including the device’s current block (this count will be less in case of the corner blocks). Based on the calculated block numbers, the database is queried for events occurring on any of these blocks. This is achieved using a random number generator. The server picks an event from any one of the neighboring blocks based on the randomly chosen blockId. The results of this query is then sent to the target device using APNS via Urban Airship. In the future, more blocks can be considered to generate the event list for the device. This can avoid an empty event list when the device is in an area with no events.

4.2.2 Web Services

We have implemented JBoss Web services in order to receive the updates from the Client iPod application. The Web services implement the updateLocation Web method:

The Web method UpdateLocation receives the update from the client application and updates the database record accordingly. The method receives the following parameters: latitude, longitude - obtained from the CoreLocation APIs, blockId - described above,
timestamp - the device timestamp when the update was sent, devicetoken - a unique identifier for the device’s session with APNS and deviceid - a unique string to each device based on various hardware details. Once the Web service is successfully built, it generates the WSDL file under the JBoss server directory.

On invoking the UpdateLocation Web method, it updates the Device table record which matches the given device ID, otherwise the update is ignored (thus no notifications are sent to that device). It calculates the adjacent blocks associated with this block id and collects the events at all these locations only if the database record is successfully updated. After compiling the events, the method pushes this information to the device via Urban Airship.

Before pushing the data the Web services registers the device token with Urban Airship using the Application Key and the Application secret to check whether the device token is active. As we explain in section 4.3.1, Urban Airship uses the Application Key and Application Secret to uniquely identify the Client application and send push notifications via APNS. If the token registration is successful, the method returns a HTTP status code of 200 and the server pushes the data to the device only upon successful registration. The Web services uses the Java HttpClient [6] API in order to achieve the registration and push functionality. The Web service uses the HttpClient API to connect to the URL http://go.urbanairship.com/ and uses the Credentials class to authenticate to the Urban Airship. The Registration and Push methods use the PutMethod and PostMethod classes of the HttpClient API to send the requests to Urban Airship.

In this way, only the registered devices shall receive the notifications.
4.3 Push Notification Services

Apple Push Notification Service (APNS for short) is the centerpiece of the push notifications feature. It is a robust and highly efficient service for propagating information to Apple mobile devices. Each such device establishes an accredited and encrypted IP connection with the service and receives notifications over this persistent connection. If a notification for an application arrives when that application is not running, the device alerts the user that the application has data waiting for it.

The notifications originate from the CC server which connects with APNS through a persistent and secure channel while monitoring incoming data intended for their client applications. This is enabled by using the Urban Airship APIs. The flow of remote-notification data is one-way. The server composes a notification package that includes the device token for a client application and the payload. The server sends the notification to APNs which in turn pushes the notification to the device using Urban Airship.

The Client Orientation application implements the function `didReceiveRemoteNotification` that receives the notifications from the Apple Push Notification Service and displays them.

4.3.1 Urban Airship

Urban Airship provides an infrastructure that allows a server to send bulk messages as well as individual messages to devices. Urban Airship eases establishing the socket connection between the CC server and APNS.

It requires the client application to be registered in the Urban Airship administrative interface using the Push SSL Certificate which it uses to communicate with APNS. The Push SSL certificate identifies the Client application to APNS.
Each application has an Application Key and both an Application Secret. As we discuss shortly, the Application Master Secret authenticates push notifications. Credentials are supplied in HTTP Basic Authentication, always over our HTTPS connection. The Application Secret is to be included in the application to perform registration.

Urban Airship provides two main functions in order to achieve push functionality:

- Registration
- Push

The Web service invokes these functions to push the events list information in the neighboring locations to the device. The Web service uses the HttpClient APIs and JSON to send the data to the Urban Airship.

JSON [7], short for JavaScript Object Notation, is a lightweight computer data interchange format. The JSON format is often used for serialization and transmitting structured data over a network connection. Its main application is in Ajax web application programming, where it serves as an alternative to the XML format.

The Registration method (Figure 4.2) uses the Application Key and Application Secret to authenticate the requests. HTTP PUT to /api/device_tokens/ <device_token> registers a device token at the Urban Airship. This is necessary for broadcasts, and recommended (but optional) for individual pushes. This returns HTTP 201 during the device’s first registration and HTTP 200 OK for any updates. We use the registration call to query Apple’s feedback service which identifies any inactive devices and avoids sending push notifications to those devices. There is an optional JSON payload to specify an alias or tags for this device token.
The Push method uses an **HTTP POST** with `/api/push/` as parameter to send push notification to one or more users. As shown below in Figure 4.3, the payload is in JSON with content-type `application/json`.

The response is an HTTP 200 OK with no content, or an HTTP 400 Bad Request if the structure was invalid. This POST is authenticated with the Application Key and the Application Master Secret. The JSON payload has the structure as shown below in Figure 4.3.

```
{
    "device_tokens": [
        "some device token",
        "another device token"
    ],
    "aliases": [
        "user1",
        "user2"
    ],
    "tags": [
        "tag1",
        "tag2"
    ],
    "schedule_for": ["2016-07-27 22:48:00", "2016-07-28 22:48:00"],
    "exclude_tokens": [
        "device token you want to skip",
        "another device token you want to skip"
    ],
    "aps": {
        "badge": 10,
        "alert": "Hello from Urban Airship!",
        "sound": "cat.caf"
    }
}
```

Figure 4.2: Registration Message Structure

Figure 4.3: POST Message Structure
The next chapter describes the Orientation application’s system workflow.
CHAPTER 5

SYSTEM WORKFLOW

The Orientation application developed achieves the required functionality of sending notifications to the user’s iPod touch device based on its location. The device uses the Core Location Framework and the Google Maps APIs to record the current location of the device, accurate within a range of 100m. This is shown in Figure 5.1. The device has to be connected to the OSU Wireless network in order to achieve this functionality. In the tests we conducted the location details thus obtained are accurate, within a range of few meters (10-15m) in most cases with places of dense Wi-Fi connectivity. The application uses the SOAP framework to update the results to the CC server. The Google maps take a couple of seconds to load and to display the current location of the device.

The CC server is built on Ubuntu 9.10 Linux server using Java Enterprise Edition (EE) and the JBoss application server is used for providing the Web services logic. The server exposes the UpdateLocation Web method for the client to interact with the server. The Web method receives the updated values and updates the device table, if that device record already exists in the database.

The Client iPod application refreshes the location once per minute, updates the location details to the server, and updates the location immediately whenever the refresh button is
clicked. This is shown in Figure 5.2. Upon receiving the location details, the server generates the events information based on the current blockId of the device. For example, if we are at Dreese Labs, the associated blockId is 770. The neighboring blocks for this location are 730, 731, 732, 769, 771, 808, 809, 810 and 770 (current blockId). A random generator with a seed of 9 is used to choose one of the 9 neighboring blocks. The event information for the chosen block is determined by accessing the database in order to generate the payload for the Urban Airship push method.

Once the push method return success, it takes a couple of seconds for the push notification to reach the device. The push notification comes up with a ping sound to inform the user of the notification. We also simulated location updates to the CC server using a soapUI tool instead of the device sending the updates. This caused the notifications to pop-up on the device even when the application is closed. The notification asks the user to click the
Figure 5.2: Device sending updates to the CC server

view button in order to view the event details. This is shown in Figure 5.3
Figure 5.3: Event Notification popping up on the device

The next chapter concludes the thesis and discusses directions for future work.
CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The Orientation application targets real-time and location specific notifications to new and prospective OSU students. This system efficiently uses the free Wi-Fi services available on campus and may help reduce students’ information overload. The real time notifications inform the students about the events in their corresponding locations, thereby facilitating their planning activities. The handheld form factor of the mobile devices makes them more attractive to student’s and may reach a wider audience in using the application. The application completely removes the cost incurred in the earlier alert system which is based on sending and receiving text messages and hence it proves to be financially viable.

The Orientation application is thus a cost-effective, real-time communication platform that can provide improved communication for new and prospective OSU students.
6.2 Future Work

The following features can be considered as enhancements to the current work.

- The Orientation project can be further extended to consider user preferences while sending event notifications. Presently, notifications are just collections of events occurring in neighboring blocks. The application can be modified to consider users’ personal choices while selecting these event notifications. Thus notifications that match users’ interests can be made available to them with high priority, when the number of events exceeds the prescribed maximum.

- CC can also be extended to provide Emergency notifications. When emergency situations arise, OSU Public Safety must send bulk text messages to students’ mobile phones via the Buckeye Alert system. Though Buckeye Alert rapidly notifies the student body of emergencies, it costs several thousand dollars to send each set of bulk text messages and cellular carriers charge students to receive them. In such scenarios, OSU Public Safety can send push notifications via CC to all students’ devices to inform them of an emergency. The notifications are transmitted for free over OSU Wireless, thereby saving the cost to send text messages. Unlike text messages, students are not charged for receiving the notifications. For localized emergencies, such as a building fire, evacuation notifications can be sent to everyone in the emergency locations. The application functions as a prototype to the Emergency notification system and this functionality can be achieved by generating emergency event notification based on the location information of the devices stored on the server.

- It can also be extended to include a faculty notification system, an extension that facilitates information exchange between faculty and their students.
• It can also be extended to provide the route to reach the event destination when selecting the event. The user’s current location can be obtained from the database and route to the destination can be obtained using Google maps API’s.

• The application can be extended to automatically trigger event notifications even if there are no location notifications from the device. This automated event notification system can be used in Emergency Notification system and broadcast messages are sent from Urban Airship. Broadcast messages are sent using the application key and it can use the same pushdevice method implemented on the server with just changing the PostMethod parameter to api/push/broadcast/.

• Another enhancement feature would be to allow authorized users to create new events in the system on the fly. This would be something similar to adding events to the database when the authorized user comes across the event location.

• The project can also be integrated with Carmen and BuckeyeLink to provide a more integrated communication system of OSU.

• The project can also be implemented on other smartphone platforms like Symbian and Android.
BIBLIOGRAPHY