

CAMPUS DRIVER ASSISTANT ON AN ANDROID PLATFORM

by

Iana Zankina

A Thesis Submitted to the Faculty of
The College of Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

Florida Atlantic University

Boca Raton, Florida

December 2012

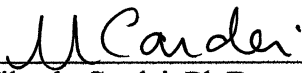
CAMPUS DRIVER ASSISTANT ON AN ANDROID PLATFORM

by


Iana Zankina

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Mihaela Cardei, Department of Computer and Electrical Engineering and Computer Science, and has been approved by the members of her supervisory committee. It was submitted to the faculty of the College of Engineering and Computer Science and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

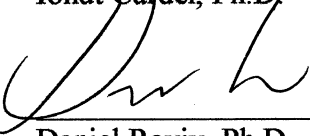
SUPERVISORY COMMITTEE:




Mihaela Cardei, Ph.D.
Thesis Advisor



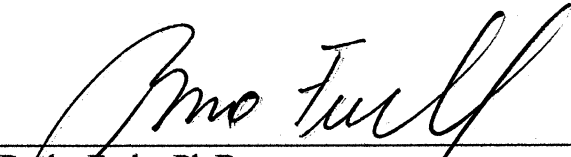
Ionut Cardei, Ph.D.



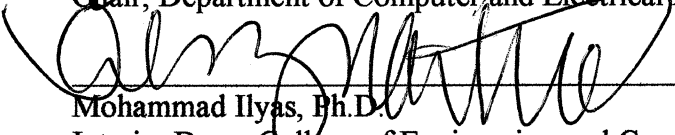
Daniel Raviv, Ph.D.



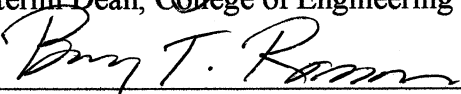
Imad Mahgoub, Ph.D.




Borko Furht, Ph.D.
Chair, Department of Computer and Electrical Engineering and Computer Science



Mohammad Ilyas, Ph.D.
Interim Dean, College of Engineering and Computer Science



Barry T. Rosson, Ph.D.
Dean, Graduate College



Date

ACKNOWLEDGEMENTS

I would like to thank my family for giving me the opportunity to continue my education and achieve my goals, without them this would not have been possible. They not only gave me the reason, drive, and encouragement to strive for my dreams, but they also gave me all the support and love I could have possibly needed. I hope I make them proud.

I would also like to sincerely thank my advisor, Dr. Mihaela Cardei, as well as all of my committee members, for their encouragement and advice. Their input has been priceless over the course of the writing of this manuscript. Their belief in my abilities helped built confidence and enthusiasm in my work. I appreciate the chance they gave me to be involved in a very cutting edge project and giving me valuable exposure to techniques I can utilize in my future endeavors.

ABSTRACT

Author: Iana Zankina
Title: Campus Driver Assistant on an Android Platform
Institution: Florida Atlantic University
Thesis Advisor: Dr. Mihaela Cardei
Degree: Master of Science
Year: 2012

College campuses can be large, confusing, and intimidating for new students and visitors. Finding the campus may be easy using a GPS unit or Google Maps directions, but this is not the case when you are actually on the campus. There is no service that provides directional assistance for the campus itself. This thesis proposes a driver assistant application running on an Android platform that can direct drivers to different buildings and parking lots in the campus.

The application's user interface lets the user select a user type, a campus, and a destination through use of drop down menus and buttons. Once the user submits the needed information, then next portion of the application runs in the background. The app retrieves the Campus Map XML created by the mapping tool that was constructed for this project. The XML data containing all the map elements is then parsed and stored in a hierarchal data structure. The resulting objects are then used to construct a campus graph, on which an altered version of Dijkstra's Shortest Path algorithm is executed. When the

path to the destination has been discovered, the campus map with the computed path overlaid is displayed on the user's device, showing the route to the desired destination.

CAMPUS DRIVER ASSISTANT ON AN ANDROID PLATFORM

FIGURES	viii
CHAPTER 1: INTRODUCTION	1
1.1 Motivation and Problem Statement	3
1.2 Related Works	3
1.3 Contribution	5
1.4 Project Tasks and Thesis Organization	7
CHAPTER 2: UML CLASS DIAGRAM	10
2.1 UML	10
2.2 UML Design Software	11
2.3 Campus Map Class Diagram	12
CHAPTER 3: THE ANDROID PLATFORM	16
3.1 Android Platform Background	16
3.2 Android Platform Architecture	18
3.3 Open Source Community	21
3.4 Java Android and the Campus Driver Assistant on an Android Platform	22
CHAPTER 4: THE XML PARSER	25
4.1 Background on XML Parsing	25
4.2 Parsers and Libraries Description	26
4.3 XML Parsing and the Campus Driver Assistant on an Android Platform	28
CHAPTER 5: THE CAMPUS MAP	31
5.1 Google Map of Campus	32
5.2 Map Editor Tool	33
CHAPTER 6: FINDING THE SOURCE-DESTINATION PATH	36
6.1 Designing the Graph Data Structure	37
6.2 An Algorithm for Finding the Shortest Path	39
6.3 Dijkstra's Shortest Path Algorithm	41

CHAPTER 7: COMPONENT INTEGRATION	45
7.1 Component Creation and Integration	45
7.2 Difficulties with Integration	50
CHAPTER 8: CONCLUSIONS	51
8.1 Conclusions on the Project	51
8.2 Future Works	52
REFERENCES	53

FIGURES

Figure 1: Project Task Diagram	7
Figure 2: UML Class Diagram	13
Figure 3: Android Version over Time	17
Figure 4: The Software Layers of the Android Platform [Spe12].	20
Figure 5: Our User Interface	24
Figure 6: Comparison of DOM and SAX Parsers [Dom12].....	27
Figure 7: Parser code snippet.....	30
Figure 8: Google Campus Map (Unaltered).	33
Figure 9: Altered Campus Map.....	35
Figure 10: Anatomy of a diagraph and XML code.....	38
Figure 11: Pseudocode of original Dijkstra's Algorithm [Col01].....	41
Figure 12: Dijkstra's Shortest Path Algorithm example running on a graph [Dij12].	42
Figure 13: Dijkstra's Alg. example running up to predefined destination node[Dij12]. .	43
Figure 14: Our pseudocode version of Dijkstra's.....	44
Figure 15: User interface with resulting map and path.....	49

CHAPTER 1: INTRODUCTION

It was not long ago that having a cellular phone was a luxury few people could afford, but in the last ten years there have been great technological advancements that allowed cell phones to become ubiquitous. Mobile phones have become easier to afford, smaller, faster, more powerful, and have managed to become interwoven in user's daily routines. Most importantly the king of cell phones, the smart phone, has much of the functionality of a personal computer, and so has made paper planners, calendars, address books, and even alarm clocks obsolete [Nel12]. Smart phones have been especially helpful in boosting social networking websites like Facebook and Twitter. It is now possible to access these sites and add text and pictures from any location, doing pretty much any activity that can be done on a computer. This has made it easier for people, who are geographically separated, to feel involved in each other's lives, and be able to keep in touch without much effort.

Smart phones have also made it easier for people who work on the move to stay in tune with constant email alerts and web browsing access. If this is not enough, there are endless applications to keep you entertained if you need it. From housewives needing to keep their kids occupied in doctor's offices, to businessmen waiting for another delayed flight, smart phone applications have become irreplaceable. Many of these applications- including games, book libraries, video streaming, GPS navigation and many more- are free of charge. They are also simple to install and use.

There are probably not many people who have not come across the Angry Birds game, where users with touch screen smart phones aim and shoot birds with a slingshot to destroy different structures. The Netflix app allows users to instantly stream movies and shows to their phones, entertaining them regardless of the absence of a television in the vicinity. Another very popular app is Google earth, which allows users to find different locations and landmarks via satellite imagery. There are also many apps created for specific businesses, such as banks, retail stores, and many more.

One of the most popular smartphone technologies is Android [And12]. It is an open source platform, meaning that its code is released to the public as soon as the new version of the platform is completed. In addition, any user with the desire and knowledge to modify or create Android applications is not only able to, but encouraged to do so. This is a unique way Android developers are interacting with users and their input makes the Android platform stay on the budding edge of the market. Users are able to customize their devices as they see fit, and they can share any apps they create easily with other Android users, spreading new ideas and software.

Other technological advancements have recently gained popularity. There are many devices and different applications that focus on directing the user to desired locations. Today's drivers are well equipped for travel thanks to the GPS units many have in their cars, which help them not only find their way, but also avoid congested routes and drive safely [Gps12]. These are not the drivers of yesteryear that have to stop for directions or get lost. GPS applications allow users to enter a destination and using their current coordinates, display the fastest way to get to their destination [Woo12]. Additional features have evolved over time, such as displaying congested routes, which

allow users to make the smart driving decisions and improve driving safety as well. This saves time and stress when going to unfamiliar places or taking long trips. Since this technology is readily available to scientists and engineers, it is therefore important for us to make use of it and improve it as much as possible for the sake of the users.

1.1 Motivation and Problem Statement

Google Maps and GPS have become very popular in recent years, with vast amounts of users relying on them for directions [Kin11], but their capabilities have not yet been applied to university campuses. Directions within campuses are not available using the Google maps application. The campuses can be quite large and confusing. New students and staff, as well as visitors can have a very difficult time getting around. Even when they are asking for directions, they often time cannot find the destination because the directions involve knowing the surrounding buildings and landmarks of the campus. This can get quite stressful, especially considering students are often on a schedule and need to get to classes on time. Eliminating this stress and confusion would improve the overall atmosphere of the campus. Since smart phones are a ubiquitous technology nowadays, it makes sense to use them to resolve such issues.

The problem that we address in this thesis is how to use the current advances in technology to provide a mechanism to facilitate drivers' navigation in campuses. This navigation system should provide driving directions to buildings and parking lots.

1.2 Related Works

There has not been much related work in the specific area of our project. Providing a route to a destination has been available for general directions for a while

using GPS and Google Maps, as was mentioned previously. However, there has not been much made available for the specific task of directing users around a college campus, using a map with an overlaid route. There are plenty of college websites that provide directions and maps, including interactive maps, but not ones that allow the user to select a destination and give them the shortest path to that destination. Performing some research we came across a few applications which were aimed at a similar idea to our application.

Google has been involved with a few projects that have similarities to ours. The one that most resembles our project is the 2007 Google Street View Project. This project involves taking images using car, trike or bike mounted cameras and mapping unique areas such as parks, university campuses and malls [Str12]. This allows users to interactively view the areas, seeing the location in 360°. While this technology is interesting, and it does allow very in debt mapping of the campuses, including pedestrian walkways as well as streets, it does not allow the user to enter their desired destination and have the shortest path to their destination be displayed on the map.

The University of California, Santa Barbara students are working on an Interactive Map Project [Wel12]. Their map allows users to select buildings to zoom into and locate, as well as finding a room within a building. This project does not allow users to find directions from one area of the campus to another.

One campus map that allows similar functionality to our intended application is the University of California, San Diego campus map. This map is accessed through a web browser and allows the user to select a source location and a destination. When this information is submitted, the shortest path is outlined on the map, and the distance and

expected walking time is shown [Kel12]. This page does not however, allow for driving directions within campus.

The campus map oriented projects discussed above, and our application, have one major difference, our application is intended for an Android smartphone device, whereas the other campus maps are web pages accessible from a browser but are not available as applications. Through our research, we found one available Android application that was focused on showing a user their current location, determined by the device GPS, overlaid on top of a college campus [Cam12]. The Campus Maps application available through the Google Store for Android devices, allows a user that downloads the app, to see their current location and movements shown on a map of a college campus, so long as the map is included in the list of available maps for the app. The reviews of this app are not very flattering [Cam12]. Users state that their campuses are not available, or that the map images are not good and so it is not helpful. Although the idea behind this application is similar to our own, it does not show the best, shortest route to a desired destination, and it currently does not support the FAU campuses.

There are not many applications or projects that are aimed at the same goals as our project, however there are some similar ideas being developed that may provide some useful input to our concept.

1.3 Contribution

In this thesis, we design and develop a campus driver assistant application that runs on an Android platform and is intended to assist campus drivers and users that want to find driving directions to a certain building in the campus. The application provides driving directions to the destination building or to a parking lot, and this parking selection

is based on the type of user (e.g. visitor, staff, and students). The computed path (e. g. the shortest-path) is displayed on the smartphone to assist the driver in reaching the destination.

The driving direction project falls under the umbrella of Campus 2020 initiative, managed by Dr. Daniel Raviv, which seeks to provide avenues to enhance and revolutionize campus life experience of students, staff, and visitors. A number of exciting project are underway, including campus driving directions application.

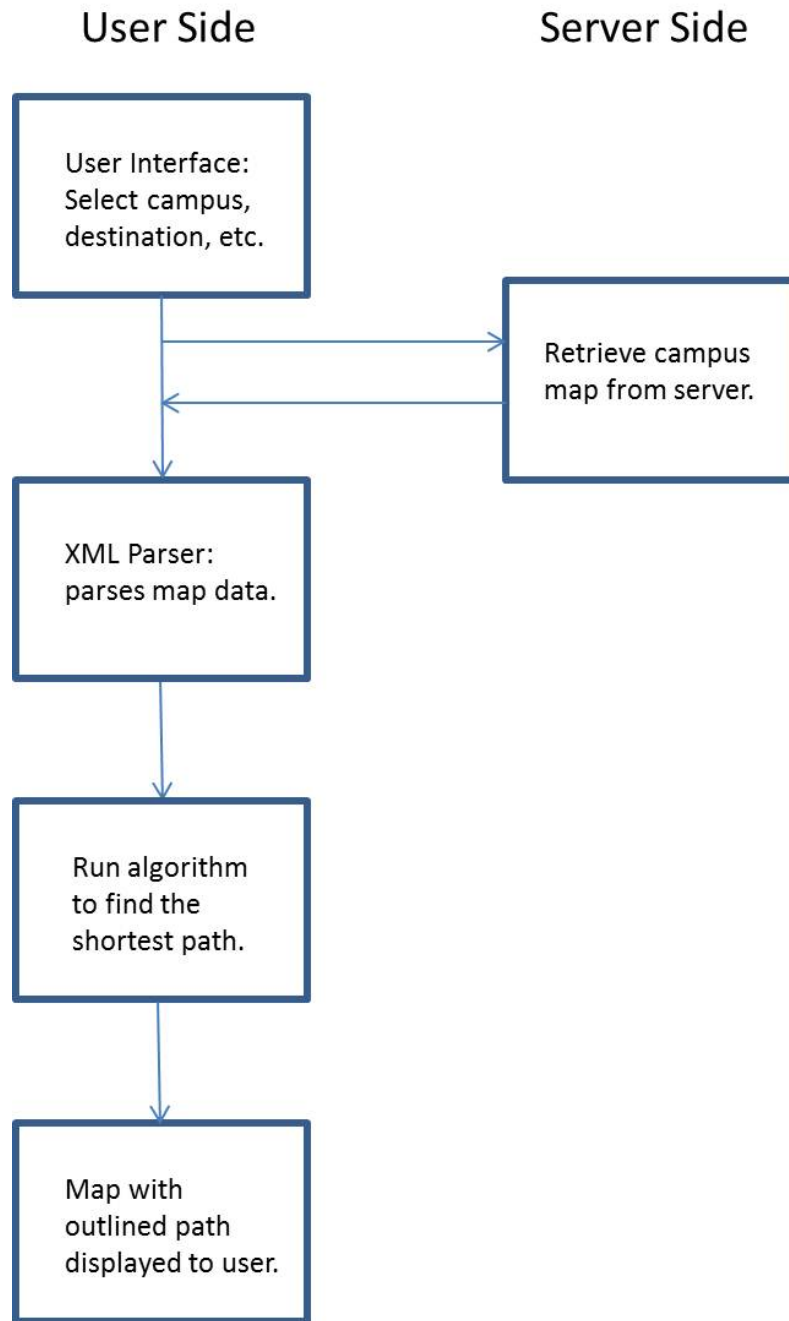


Figure 1: Project Task Diagram

1.4 Project Tasks and Thesis Organization

The main tasks of our application are presented in Figure 1 above:

- **UML class diagram:** the UML class diagram is basically the blue print of our application. It is a visual representation of how we will structure our classes. This is an important design step, because it allows us to have a starting point when coding. A UML diagram has a great organizational role and it prevents a lot of confusion and backtracking.
- **User Interface:** the user interface is the part of the app that interacts with the user. It allows the user to enter the needed information into the app, such as the destination and whether the user is a student, visitor, or staff. This information is then used by the app in computing a path to the destination.
- **Campus map tool:** we designed a campus map tool which can be used to create a map of the campus in XML format. This XML campus map is later used by the app to identify the campus map objects, such as buildings, streets, parking lots, walkways, etc.
- **Server side:** the server is used to store campus maps in XML format. The Android application connects to the server using HTTP protocol to retrieve the campus map as an XML file.
- **Campus map parsing:** the application parses the XML campus map in order to retrieve useful information, such as intersections, buildings, parking lots, road segments, traffic signs, etc. Based on this information, a data structure needed in building the shortest path is formed.
- **Finding a shortest path:** design of the algorithm that computes the shortest-path between the current user location (e.g. the source) and the intended destination.

The rest of the thesis is organized as follows. In chapter 2 we present the UML class diagram where we explain the classes used in our project and their attributes. We continue in chapter 3 with an introduction to the Android platform and explain the way in which it has been used in our project. Chapter 4 presents the XML parser, which is critical in retrieving information about the campus structure. In chapter 5 we discuss the mapping of the campus and the tools used. Chapter 6 focuses on the algorithm needed to find the shortest path to the destination. We continue in Chapter 7 with a discussion on the integration of all these components and difficulties we ran into during this process. We conclude our thesis in Chapter 8.

CHAPTER 2: UML CLASS DIAGRAM

When starting a project, an important step is to make a clear plan or outline to follow, such that each task is specified. This gives a clear jumping off point and keeps track of the project's progress. Creating such a model allows developers to lay out their thoughts and ideas about how to approach the project before the work begins, and prevents them from having to backtrack or start over due to confusion about what needs to be done. A model diagram ensures everyone working on the project knows how it will be done and allows for tasks to be divided clearly preventing overlapping of duties. The most accepted way of creating such a structured model is by using UML [UmlR12]. UML is used not only for engineering application projects but it also can be applied to many different fields, such as business and architecture.

2.1 UML

UML stands for Unified Modeling Language and refers to the idea of modeling or creating a visual representation of a project being developed, using a set of defined and accepted standards for such diagrams [Int12]. Before UML officially existed, there were three methodologies for modeling software systems: The Booch methodology, Object Modeling technique (OMT), and Objectory methodology [Int(2)12]. Each of these had strengths in different areas of modeling, and each had its own notations [Int(2)12].

Later on, the brilliant minds behind each of these methodologies decided to

collaborate and make one common set of accepted rules for modeling. This way all the aspects of modeling, such as analysis, design and use cases, would become compatible. First Grady Booch, and Jim Rumbaugh who fathered OMT, decided to work together and in 1995 came up with the Unified Method [UmlH12]. Later, Ivar Jacobson who came up with the Objectory Methodology, joined forces with the others and after some years of narrowing it down, in 1997 the UML was finally submitted for standardization [Int(2)12].

The UML encompasses many different aspects. There are activity diagrams, use case diagrams, sequence diagrams, collaboration diagrams, and class diagrams [Int(2)12]. Activity diagrams represent the control flow in the system. Use case diagrams look at the external entities that interact with the system, those entities are actors and the interactions with the system are events. Sequence diagrams show timed sequence of object interaction. Collaboration diagrams show the links between objects. The diagram we are most interested in for our project is the class diagram.

A class diagram is basically a collection of objects in an object oriented application, which share common structure, behavior, relationships, and semantics [Int(2)12]. A class diagram shows these aspects visually. A class is represented by a rectangular icon which has a location for the name, the attributes, and the operations. The relationships, dependency, inheritance, composition, and association/aggregation are depicted by special symbols at the end of arrows connecting the classes [UmlT12].

2.2 UML Design Software

There are quite a few different software programs available for designing UML diagrams. UMLet, Altova UModel, Visual Paradigm for UML, UML Designer, and ArgoUML are just some of the software available for designing UML diagrams. Most of

these are free, some are open source, and they are often available as a plug in to an IDE such as Eclipse, or as a stand-alone program. For our UML diagram we decided to work with ArgoUML. It is a free open source, stand-alone, platform independent tool with a user-friendly interface and is very easy to install and use [Tig12]. It is a user-friendly tool that supports the nine UML diagrams, and allows for easy export of the diagram images into MS Word and other programs. ArgoUML is a powerful software with many options and different tabs and screens. Its user interface allows users to build easy to read, well-organized diagrams. For our project, designing the class diagram is the main feature we are interested in and the software we chose to use was ArgoUML.

2.3 Campus Map Class Diagram

For our project the UML diagram is comprised of many classes in an inheritance hierarchy. Each class outlines objects, which are comprised in the map XML file.

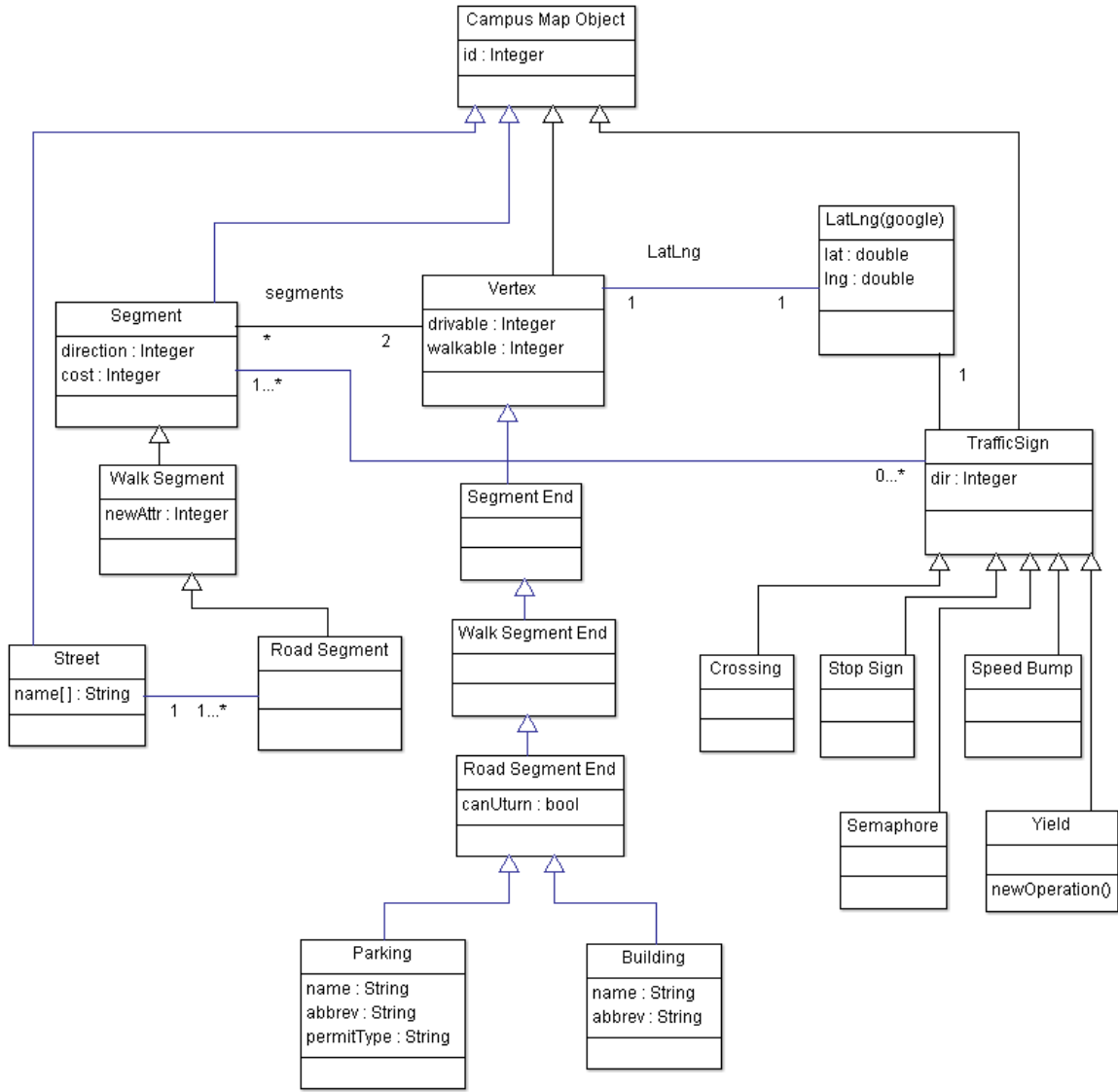


Figure 2: UML Class Diagram

Figure 2 depicts the class structure of our Campus Driver Assistant used by our Android Platform project. The class structure is basically built to hold and manipulate the data provided by the campus map. The root of the structure is the Campus Map Object class who is the parent class for all other objects in the project. This class provides every

subsequent decendent with an individual Id and is the basic, simplest unit of map data. This Id is provided by the map of the campus later on in our code.

Inheriting directly from the Campus Map Object class are two different classes which create two branches in the class structure. These are the Vertex class and the Segment class. A vertex is basically an ending point on the map, such as a building, an intersection, a parking lot and so on, and has two boolean attributes that store information on whether that vertex is walkable and/or drivable.

A segment is connecting two vertices. The relationship between the Vertex class and the Segment class is such that every vertex can have many segments, but every segment can have only two vertices (two endpoints). These classes in turn have their own subclasses. The Vertex class is a direct superclass to the Segment End class, which in turn has a subclass itself, the Walk Segment End class. The Walk Segment End class sets the walkable variable to true, and also has a subclass, the Road Segment End class. The Road Segment End class sets the drivable variable to true and contains a variable to determine if the segment end allows for U-turns. It also has it has two subclasses the Building subclass and the Parking subclass.

The Building class holds all the data for a campus building from the campus map, such as building name and abbreviations. The Parking class holds the information for a parking lot on campus, such as the name, the abbreviation, and permit type. The types of permits are faculty and staff, students, and visitors. On the othe branch of the Campus Map Object is the Segment class which contains the distance cost and direction attributes, as well as the vertices that are at its end points. The child classes to the Segment class are

the Walk Segment and Road Segment classes, which are pretty self explanatory, and set the walkable and driveable variables.

There is one class which stands on its own and is not a subclass of the Campus Map Object, this is the LatLng class. This class, as its name states, holds the latitude and longitude coordinates of all of the vertices in the campus map. There is an instance of the LatLng class within each Vertex object.

Another class that inherits from Campus Map Object is the TrafficSign class, which has an attribute dir for direction. A number of classes are inheriting from the TrafficSign class, such as Crossing, Stop Sign, Semaphore, Speed Bump, and Yield. These classes are used to augment the campus map with the traffic signs.

CHAPTER 3: THE ANDROID PLATFORM

The Android Platform is currently the most popular platform for mobile devices [And12]. As described in the introduction, it is unique in the way it is rooted in the open source ideology of sharing code and documentation with users and allowing them to customize and contribute to the software and generate new ideas for developers to work on. Here we shall examine the birth of the Android Platform and what led up to it.

3.1 Android Platform Background

The evolution of the Android platform we have come to know may not be what users expect. Everyone associates Android with Google Inc., but it is a little known fact that they did not actually create Android [His12]. In 2003, Android Inc. was started by Andy Rubin, Rich Miner, Nick Sears and Chris White, and it was not until 2005 that Google bought Android realizing its potential and fulfilling their goal of breaking into the mobile phone industry [His12] [HisO12]. Google made the first release of their Android Platform in September 2008, though at that time it was not used in any commercial devices [AndT12]. Although Android was bought by Google, its development is actually a group effort. On November 5th, 2007 the Open Handset Alliance was established [Ind12]. This alliance of technology and mobile industry leaders, led by Google, agreed to collaborate and develop the Android Platform in order to provide users with cutting

edge products at a faster pace and lower cost. The alliance is currently comprised of 84 companies who are “committed to commercially deploy handsets and services using the Android Platform” [Wha12].

The main idea behind the Android Platform was creating an open source software for mobile devices that prevents one provider from monopolizing the market with undisclosed technological advancement, stunting the growth of other providers and keeping costs high for consumers [AndO12]. This approach has been successful; the Android market is anything but stunted. Since their original release, they have made numerous releases, often making 2-3 per year. With every version of Android released, the platform gets more powerful and distinctive, becoming one of the leaders in Mobile Platforms [And12].



Figure 3: Android Version over Time

Figure 3 above depicts the evolution of the Android Platform [AndT12]. Android version 1.0 was the original release in September 2008, and was not yet used in commercial devices. Android version 1.1, released in February 2009, was an update to 1.0 and was the first to be used in a commercial device. Android version 1.5, codenamed Cupcake, was released in April 2009, and was the first to be utilized by manufacturers.

Android version 1.6, codenamed Donut was released in September 2009, and had improvements in the areas of the camera, screen, navigation and more.

In October 2009, version 2.0 was released, and shortly there after was updated as version 2.1 in January 2010, these two releases go by the name Éclair and was improved in the area of the user interface, keyboard, speed, calender, and had a phone based design, as did version 2.2 Froyo (Frozen Yogurt) that was released in May 2010. Gingerbread, version 2.3 was released in December 2010, and later had an update released in September 2011. Honeycomb is the codename for 3 different versions, 3.0, 3.1 and 3.2, which where released in February, June and July of 2011, and had a centric design. Ice-Cream Sandwich, version 4.0, was released in October 2011, and was focused on merging the designs of versions 2 and 3 , and improved many aspects of the platform such as a refined User Interface, improved email, social networking, and many others.

Recently in July of 2012, the newest version of Android, version 4.1 codenamed Jelly Bean was released. It focused on further improving some aspects of the User Interface as well as other parts of the platform such as photo checking, USB audio, and more [AndT12].

3.2 Android Platform Architecture

The Android Platform was developed using the Linux kernel, which is the open source core of the operating system, and is basically in charge of resource management [Ana12]. Linux was originally developed by Linus Torvalds in the early 1990's, and was influenced by the creation of Minix(a Unix operating system for personal computers).

The Kernel is subdivided into 3 levels, the system call interface (deals with read and write commands, etc.), the architecture-independent kernel code (it is the same in all

architectures), and the architecture-dependent kernel code (this is the platform specific code) [Ana12]. The kernel has a monolithic design, containing the basic services within the kernel. The biggest advantage of the Linux Kernel is its portability; it can be utilized in many different platforms. Linux is referred to as the most popular open source operating system and is both efficient and stable, making it a reliable base to build on, and so it seems a good choice for the makers of the Android Platform[Ana12].

The Android platform's software contains good User Interface capabilities, a good browser, large selection of connectivity options, and a good handle on graphics and media, and data storage methods [IntA12]. The platform also provides support for location-based services and accelerometer [IntA12]. Figure 4 depicts a simple view of the Android Platform software layers, and lists some of the services each performs.

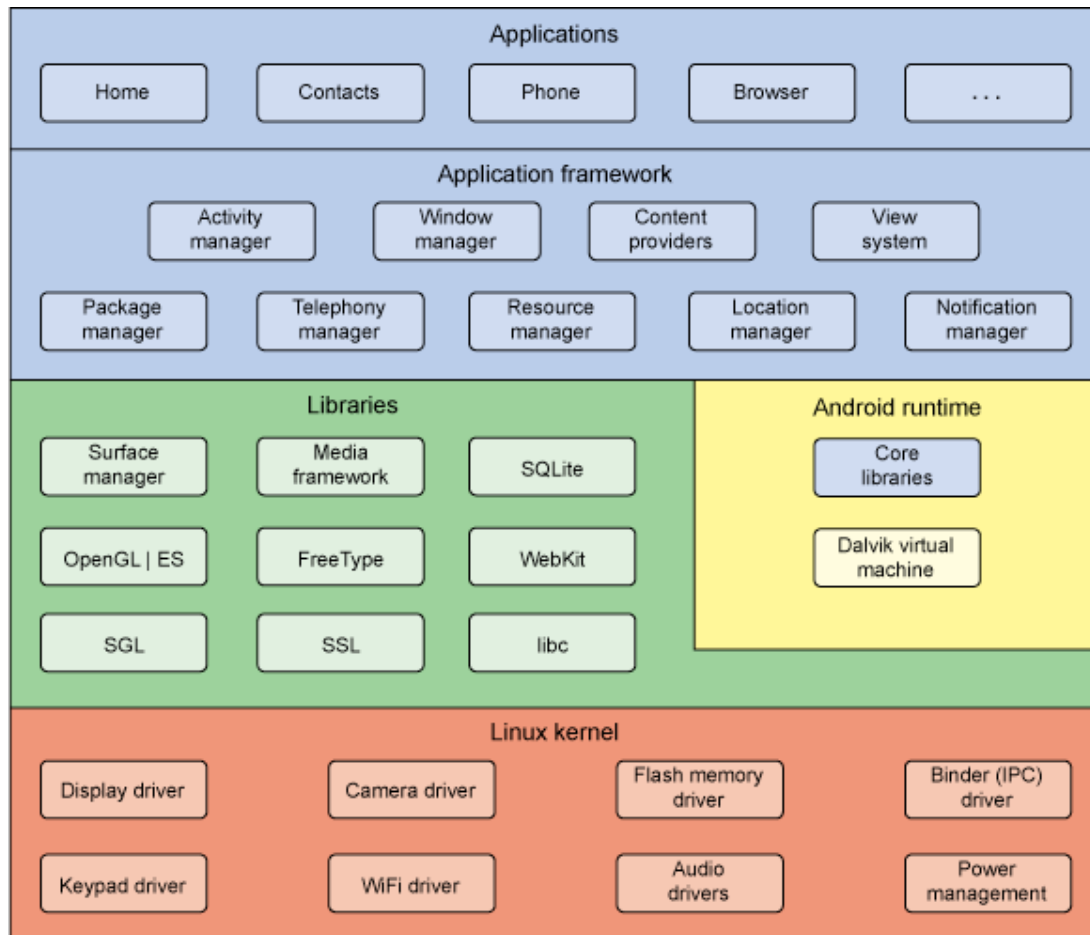


Figure 4: The Software Layers of the Android Platform [Spe12].

Android applications are written in Java and run on a Virtual Machine (Dalvik Virtual Machine), and are comprised of activities (to implement the UI for the app), services (for long running apps), content provider (to manage data access), and/or broadcast receivers (to respond to events or process data elements) [IntA12]. When an application is deployed, an XML file, the AndroidManifest file, containing the configurations for the app is also deployed; without this file the app would not be able to run [IntA2].

3.3 Open Source Community

The Android Platform was developed on the principle of open source software, meaning that the source code and documentation are available to the public free of charge. The main premise of this approach is communication [AndO12]. Google and their partners have established many forums where users and developers can interact and share information [AndO12]. There are also countless websites that provide blogs, tutorials, and examples of Android related information. As a result, it is not difficult for new users to break into the Android domain and begin developing apps and customizing their system. Android development, unlike other popular mobile platforms, is not only for the people who oversee the Android Open Source Project and develop the actual source code, it is a universal goal for all Android enthusiasts.

The way users can contribute in the improvement of the Android Platform is by writing new apps, reporting bugs, and contributing source code to the Android Open Source Project [AndO12]. As long as the code is suitable, meaning it is done in the same language and style (Java and OO), it should not be rejected. The way code is contributed is quite simple, the code is first submitted by the community member, then it is analyzed and verified by an approver (usually Google employee), and approved if it meets Android Project criteria [AndO12].

In order to develop Android applications, a user must download the Android SDK (Software Development Kit). Then it is possible to use an IDE to develop the app, such as Eclipse, which has an Android Developer Tools (ADT) Plugin that makes developing apps from your own computer, regardless of OS type, straight forward [Dev12]. Once the environment is set up, there are many libraries and classes available for use in the Java

language. Importing these into a project allows for many powerful operations of your app.

A convenient aspect of the SDK is the Mobile Device Emulator, which is basically a simulated android handheld device. This allows a person who does not have an Android device to create apps and test their behavior as it would be when running on the device. This illustrates another way in which Android displays its openness, by not requiring users that want to develop apps to have a device to work on, providing instead a way to freely create code, test it and improve the platform.

All these aspects of the Android Platform are an example of sharing and openness, and are contributing factors to the speed at which the Android movement is growing.

3.4 Java Android and the Campus Driver Assistant on an Android Platform

For this project we used the Eclipse IDE, and utilized the ADT plugin to create the application. The Android API's provide many useful packages that allow users to access the vast functionality of the Android device, such as the classes that allow the user to create a user interface for their application. Eclipse also allows for good integration of Java classes that are called from the Android Java Project. The app can incorporate user defined classes by importing them into the project and calling methods within the project.

Our app has the following tasks which were implemented using the Android capabilities:

- **User Interface:** where the user inputs data which is retained and utilized in the application. In our application, the user interface allows the user to enter information such as the destination building and user type (e.g. student, visitor,

and staff). It is also used to allow the user to choose an appropriate parking lot based on the selected user type.

- Google maps: the map of the campus is displayed for the user with the path overlaid.
- Connecting to the server: the application uses the HTTP protocol to connect to the server and retrieve the campus map.

The readily available Android packages allow easy access to many classes which are already implemented and tested. Some of the important packages we used for this project are:

- android.content – this allow the creation of the UI [Ref12].
- android.Activity – this deals with creating windows to place the UI [Ref12].
- android.os – provides basic operating system services [Ref12].
- android.widget – contains UI elements to use in the app [Ref12].
- javax.xml.parsers – contains the classes needed to parse XML documents [Ref12].
- org.w3c.dom.* - provides the official java bindings for the DOM [Ref12].

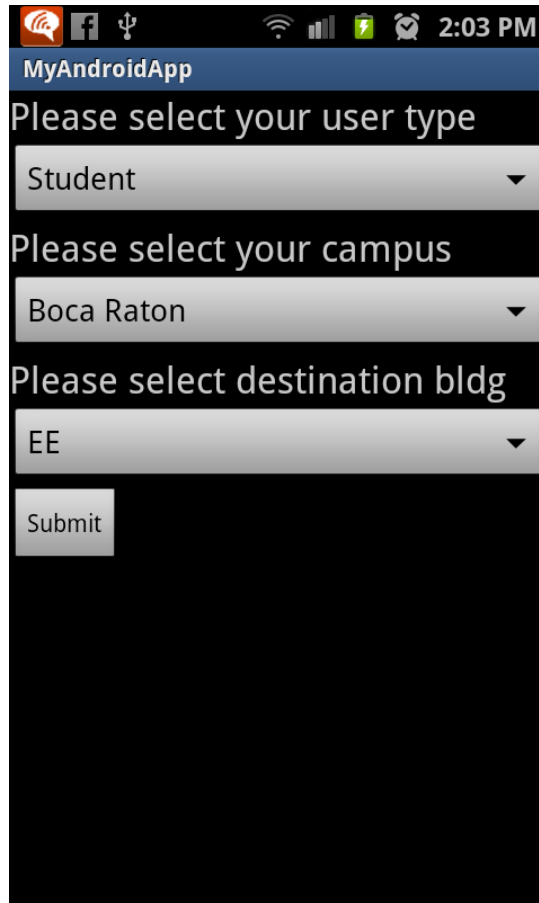


Figure 5: Our User Interface

Figure 5 above shows our user interface, with the drop down menus allowing the user to select the type, campus, and destination. When the submit button is pressed the information is processed by the application in the background.

CHAPTER 4: THE XML PARSER

In many software projects, data is sometimes received in bulk or in a different format than expected. In order to be used, it needs to be parsed, or broken up into component pieces that can be utilized by the program. For this parsing to occur, there must be some mechanism in place that can read and break apart the data and store the desired data in a format acceptable for use. One of the most common ways data is retrieved and needs to be parsed is when it is in XML format. XML stands for Extensible Markup Language, and it was designed originally to help with large-scale electronic publishing, but has also made an impact in exchange of wide variety of data via the web [Ext12]. It is not surprising, therefore, that XML parsers are implemented for many languages and platforms.

4.1 Background on XML Parsing

XML was developed in 1998 by the W3C (the World Wide Web Consortium) [Thi04]. It was developed as a specification for transferring and storing data. One of its major benefits is its simplicity. It is similar to HTML in the way it houses elements within tags and is easily understood by human readers as well as computers. It is also very useful since it can be used with different languages and platforms, so the data it holds is not available only to a limited group [Thi04]. Perhaps the most impressive

quality, and the reason why it is so valuable, is the fact that the author of the XML document can encode data using any tags they define [Thi04]. There is not a predefined list of element tags and attributes that users need to adhere to.

Since XML is very useful and popular, XML parsers are necessary to take advantage of all XML has to offer. XML parsers are implemented in quite a few languages, such as Java, C, C++, C#, and Python [Thi04]. There is also the option of creating an XML parser for specific needs. Parsing XML is so common now, that all modern browsers have a built in XML parser [XmlP12]. A parser basically reads the XML document, identifies the tags, and extracts the data between the tags. This allows a computer program to access and use the data from the XML file.

4.2 Parsers and Libraries Description

There are many implementations of parsers. They are available in different languages and platforms. For instance, the Expat XML parser library is written in C language but there is a module available for Python and Perl users [TheE12], while the Xerces parser has an implementation in C++ as well as Java, and Perl [TheA12]. Another parser, the XmlReader, is used to read XML data in the .Net Framework Silverlight [Pro12]. Although there are many different ways to approach the parsing of an XML file, the two most popular parser models available are the Document Object Model (DOM) and the Simple API for XML (SAX).

Figure 6, outlines some of the differences between the characteristics of the DOM parser and the SAX parser. The DOM parser is an object-based model, is good for parsing complex structures, and has optional validation [Dom12]. The SAX parser is an event-based model, works better for parsing simple structures, and also has optional

validation [Dom12]. The DOM parser is slower than the SAX, but unlike the SAX it can update the XML document in Memory. Another difference between the two parsers is their element sequencing. DOM is capable of traversing the entire tree structure in memory, whereas the SAX parser pinpoints a specific element within the overall structure and ignores the rest [Dom12]. There are pros and cons to both parsers, so deciding which to use depend on the needs of the application.

If the app is dealing with a large number of simple XML structures, the SAX parser would probably be of more use since it would cut down timing costs. If the app is dealing with complex XML structures and needs to be able to access the data in an orderly fashion, then the DOM parser would be a better choice. This decision does not have to be antagonized over too much however, because the two parsers share some of their implementation [Cla12]. Ultimately, for our project we needed to look at what characteristics were more important for our application, and choose accordingly.

	DOM	SAX
Type of interface	Object-based	Event-based
Object model	Created automatically	Must be created by application
Element sequencing ¹	Preserved	Ignored in favor of single events
Use of z/TPF memory	Higher	Lower
Speed of initial data retrieval	Slower	Faster
Stored information	Better for complex structures	Better for simple structures
Validation	Optional	Optional
Ability to update XML document	Yes (in memory)	No
Note: Element sequencing refers to the ability of the API to remember the order of the elements. DOM can traverse the tree structure in memory; SAX locates a specific element and ignores the surrounding elements.		

Figure 6: Comparison of DOM and SAX Parsers [Dom12].

4.3 XML Parsing and the Campus Driver Assistant on an Android Platform

Parsing was necessary for our Campus Driver Assistant since the map we retrieve from the server is formatted in XML. We had to select a parser in order to be able to break up the map XML data and store it in the hierarchal structure as was outlined in the class diagram. After considering the two most popular parsers, we selected to use the DOM parser. The XML file that we retrieve from the server is large and complex, thus we decided to use DOM to perform parsing.

The DOM parser comprises methods that traverse, access, insert, and delete from the XML tree [XmID12]. In order to manipulate the XML, it first needs to be loaded into a Document object. This is done by the DocumentBuilder Class and the DocumentBuilderFactory Class, which create an instance of the DocumentBuilder and allow it to parse XML from a variety of sources, including URL's as in our case [Cla12].

Once the Document is created, elements from the XML can be extracted using the tags as a way to identify and distinguish between the elements types. In our interpretation of the use of the parser, each element is stored with others of its kind, defined by the tag. Once all the elements are separated according to their tag, each individual element is parsed and an object of the appropriate class is created and its attributes are set. This establishes the class hierarchy structure we were aiming for, and we can use the stored data to establish a path to the user's destination.

Figure 7 displays a short snippet of code from the parser class. The first portion shows how we create the Document object and use it to parse the URL containing our map. The Document object now holds the information we need so we extract it by referring to the tag. We then create first a Node then an Element object which now holds

all the XML data we need in one bunch and needs to be extracted piece by piece to be usable. The next step is creating a `CampusMapObject` which will hold the hash maps of the segments and vertices that are the basic units we need in order to find the destination path for the user.

We continue with creation of a list structure that will hold all the building elements we extract from the `Element` object that contains all the XML data. Then for each of the buildings within the list (this is directed by the for loop), the function `parseBld` is called, which parses the specific building information, and then the building is added as a vertex in the `CampusMapObject` vertices hash table (or segments hash table for other elements, like road segments).

The next portion of the code shows this `parseBld` function, and the way it extracts all the attributes of the building element, such as the id, the directional coordinates, and the building name and abbreviation. The function then returns the `Cmo.Building` structure, which is a child class of the `Cmo.Vertex` class and is actually the object required for `CampusMapObject` to be able to add the building to the vertices hash table. This technique is used for all the elements in the XML structure until all of them are broken down and added as either vertices or segments to their appropriate hash table.

```

DocumentBuilder dbuilder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
org.w3c.dom.Document doc = dbuilder.parse(url);
Node nodeCmoColl = doc.getElementsByTagName("cmoCollection").item(0);
Element elCmoColl = (Element)nodeCmoColl;
String mapTitle = this.getNodeValue(elCmoColl, "title");
this.display(mapTitle);
cmoColl = new Cmo.CampusMapObjectColl(mapTitle);

// parse buildings:
NodeList bldList = elCmoColl.getElementsByTagName("bld");
for (int i=0; i<bldList.getLength(); i++) {
    Node node = bldList.item(i);
    Cmo.Building b = this.parseBld((Element)node);
    cmoColl.addVertex(b);
}

    .
    .
    .

private Cmo.Building parseBld(Element el) {
    int id = this.parseIdAttribute(el);
    Cmo.LatLng latLng = this.parseLatLng((Element)el.getElementsByTagName
("ll").item(0));

    String names = this.getNodeValue(el, "names");
    String abbrev = this.getNodeValue(el, "abbrev");
    return new Cmo.Building(id, latLng, names, abbrev);
}

```

Figure 7: Parser code snippet.

CHAPTER 5: THE CAMPUS MAP

The idea of mapping surroundings has been around for hundreds of years. It became a priority for sailors and merchant travelers, in the 15th and 16th centuries, to map coasts and keep a record of places where specific goods, markets, and resources were located [TheH12]. Since the time these early navigators, the mapping of the world has improved enormously, employing strategies and technologies they could not even have imagined. Mapping and navigation were revolutionized by the use of satellite and aircraft imagery [Goo12]. Satellites orbiting earth and aircraft flying over different areas supply in depth pictures of their surroundings, even ones previously inaccessible. The amount of precision and detail available from these sources is amazing, but the fact that internet users have access to this technology free of charge is remarkable. Applications such as Google Earth and Google Maps provide directions and images to users for free. After obtaining the digitally compiled maps and photographs, they incorporate them into their applications, which are easily accessible to users through the internet [Goo12]. Using these applications, users can find and view locations anywhere in the world, from getting a glimpse of distant far off places such as the Egyptian pyramids, to getting the address of the closest coffee shop to their current location. Mapping has surely been transformed in the last few decades.

5.1 Google Map of Campus

In order for us to create a well detailed, all-encompassing map of the Florida Atlantic University campuses, we needed to obtain a satellite image of the each campus. Focusing first on the Boca Raton campus, we used the Google Maps API which allows users to embed a map obtained from Google Maps into their web-sites and applications using JavaScript, and even overlay their own data over the map image, as we do with the path in this application [GooM12].

Figure 8 depicts the original campus map obtained from Google Maps with no path overlaid and little detail available apart from the imagery. Google Maps currently provides directions for the campus itself, but not inside the campus. That is, it does not provide directions to individual buildings and locations inside the campus. When a user gives the destination as a building within the university campus, the directions given will take the user into the campus and state that the destination has been reached, but they will not take the user directly to the building they expected. This can create confusion and stress for users. The goal of our application is to develop an application that provides driving directions inside the campus. For this we need to design a mapping tool that allows us to add buildings, parking lots, street information, etc.



Figure 8: Google Campus Map (Unaltered).

5.2 Map Editor Tool

Since the Google Map of the Boca Raton campus is not detailed enough for our purposes, a special tool was designed to add data to the map and easily access the map's XML. This map editor tool was created using HTML, JavaScript, and the Google Maps API v3. The tool is basically an HTML page stored on the server, which has JavaScript code embedded within to provide dynamic functionality, and utilizes the Google Maps API to access and manipulate the campus map.

The Map Editor tool allows easy manipulation of the map. Since a lot of the buildings, streets and other aspects of the campus are not marked in the original, this has to be done manually, and this tool made it fast and easy. The Map Editor allows

operations to be performed on the map that result in the XML file being created and then permits this version to be saved onto the server. It also allows for existing XML versions to be opened and manipulated. Some of the operations that can be performed on the map are listed below.

- Left click on the map to place a node. Click vertex to toggle selection.
A new vertex is in selected state.
- Right-click on a node to edit properties (for Building and Parking nodes).
- Create a segment from the selected vertex by left-clicking another vertex.
- Left-click on a RoadSegment to toggle direction.
- Right-click on a RoadSegment to add street names or to delete it.

When the map is initially loaded into the map editor tool, the above operations are used to add the data to the map. There is a window on the page that shows the XML of the map currently being manipulated. The changes to the map are visually overlaid on the map in different colors and buttons as stipulated within the JavaScript file; this is displayed by Figure 9. Note that this map is not displayed for the user.

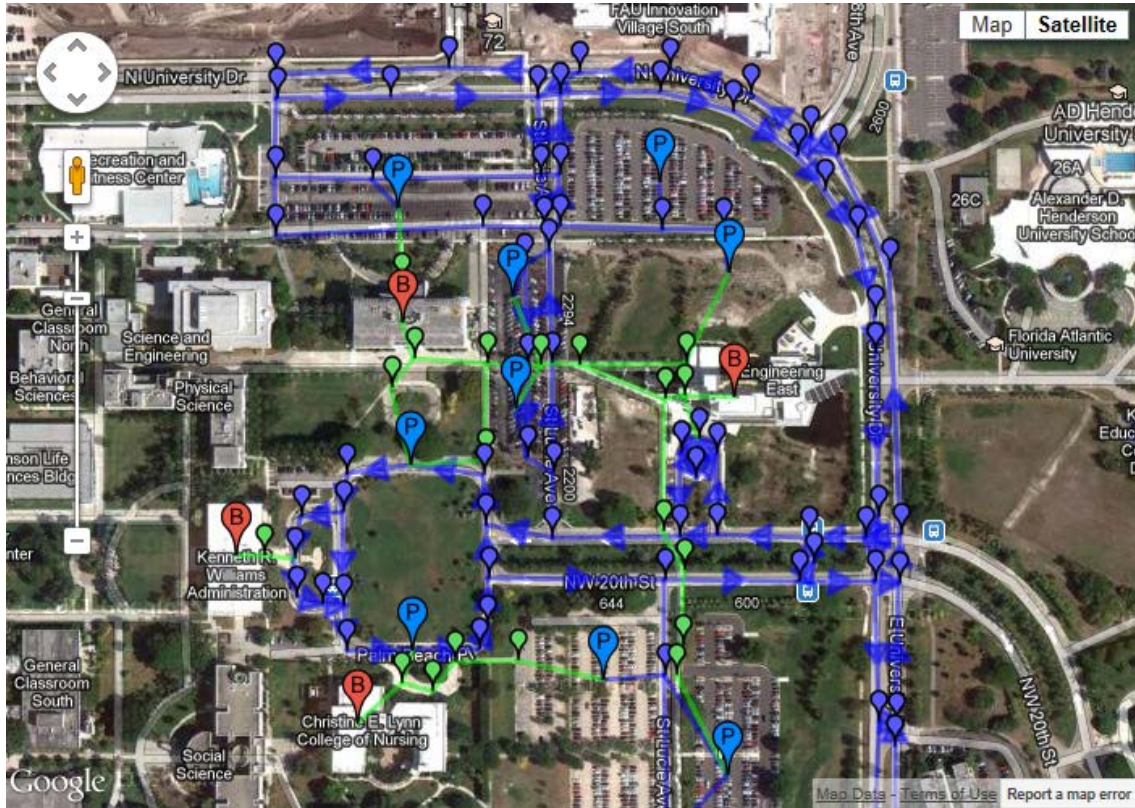


Figure 9: Altered Campus Map

Once the data for all the vertices and segments, including id's, geographical coordinates, etc., has been entered the XML file containing the information is complete. The complete, detailed XML file is the one that is retrieved from the server when our application runs. This XML code is what the parser breaks down and stores. The data is then used to find the shortest path to the user's destination and a clean map overlaid with only the resulting shortest path is displayed to the user in their Android device.

CHAPTER 6: FINDING THE SOURCE-DESTINATION PATH

Once the data parsing and appropriate classification of the objects has been done, the next step is to create a data structure or “a way to store and organize data in order to facilitate access and modifications” [Cor01]. In Computer Science, a graph refers to a set of vertices and a set of edges that connect two vertices [Ski12] and this is the data structure selected for this application. The graph will hold the objects and allow an algorithm to compute the shortest path along a group of edges.

Graphs can be directed or undirected, cyclic or acyclic, connected or strongly connected, etc. Directed graphs use directed edges to determine the vertex that can be reached from the edge. If a graph is undirected, it means both vertices that the edge connects, can be reached using that edge [Ski12]. A graph is cyclic if it contains a cycle of motion which repetitively accesses the same nodes, or vertices, with every pass. Connected graphs are those in which there is a path between every two vertices, but that path need not be direct [Ski12]. Strongly directed refers to directed graphs which contain a path from each node to the others, so if there is a unidirectional path from one node, node A, to another, node B, there must be another path available from node B to node A. There are other characteristics a graph can exhibit, but for our purposes, the graph structure is simple.

6.1 Designing the Graph Data Structure

Since our graph represents streets, building, and walkways, it is logical that it needs to be directed, cyclic, and connected. It would not make sense for our edges to be undirected since roads have a direction. Although most of our edges are bidirectional, such as the roads in our campus map, there are some areas where roads are one-ways, or unidirectional. This may create a cycle in our graph. A must-have characteristic of our graph is connectivity. Our graph must be strongly connected, meaning every node in our graph must have a path available to every other node otherwise there would be a vertex, or building, which has no available path leading to it, meaning that it would be an unreachable destination. Having a destination that is unable to be reached would make little sense in an application which is supposed to direct drivers to desired destinations.

One other characteristic in our graph is weighted edges. In order to represent the distance between vertices, our edges have a field that stores the “cost” of the edge. The cost refers to the length of the edge, or the specific distance between the two vertices the edge connects. The cost is actually determined by the geographical coordinates in the Google Map in our Map Editing tool. The distance between the endpoints of the segment is computed and then set as a field within the map XML. When the map XML is retrieved and parsed, the field containing the cost is set for each segment in the graph.

Figure 10 shows an example of a graph similar to the one utilized for this project (left), and also shows a snippet of our XML code (right). The code shows how the XML file is constructed; the tag organization, the attributes of each element, etc. The elements are defined by opening and closing tags. In the piece of code shown, the first element described is a parking lot, with the tag name `<par id=?>` and end tag `</par>`, with unique

ids for every element. It has coordinate attributes with tags <ll>, and internal tags <lat> and <lng> which hold the latitude and longitude coordinates. The parking element also contains other attributes, such as the <parking> field, which describes the type of user that can park in the lot: student, staff, or visitor.

The other element shown in the figure is a road segment with tag <rseg id=?>, and contains elements such as the id of the two endpoints of the segment, and an integer that describes the direction of the segment, which indicates if the segment is bidirectional or unidirectional, and if so what the direction is. The element also contains a cost attribute to indicate the length of the segment.

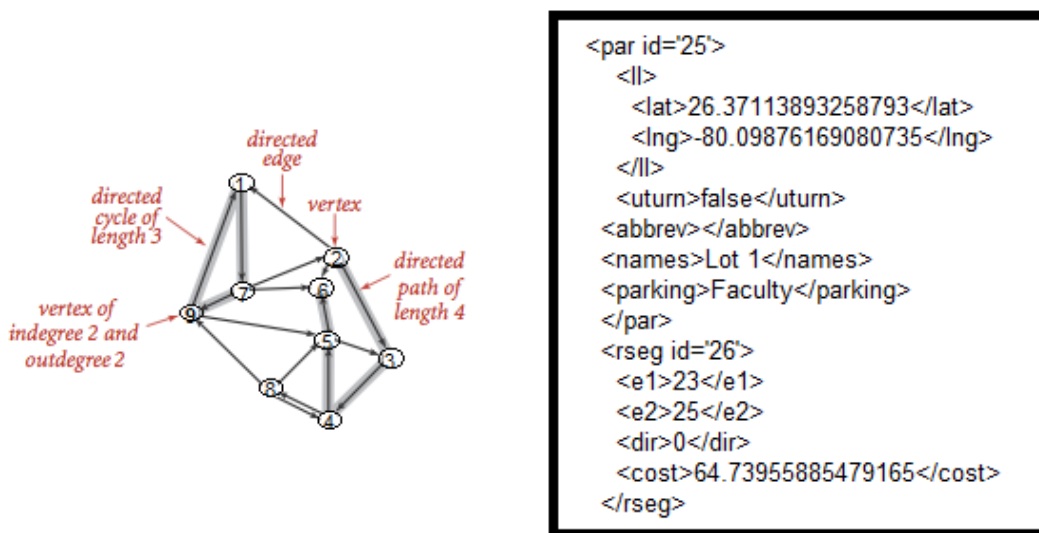


Figure 10: Anatomy of a diagraph and XML code.

The way we constructed our graph was by using a minimum priority queue structure where the nodes are arranged depending on their proximity to the source. A queue operates in a first-in first-out principle, but a minimum priority queue is different. It arranges the nodes not depending on which first arrived, but on predefined selection criteria, such as based on a comparison of a field within the nodes. In our case, each node

has a variable which holds the total distance from the source, and at each run of the algorithm that determines the shortest path, the field is adjusted and the queue is updated. This is done repeatedly until the destination is reached. The vertices, or nodes, whose shortest path has been discovered are removed from the queue and are placed in a HashMap structure which will later be used to recreate the shortest path between the source and destination.

6.2 An Algorithm for Finding the Shortest Path

An algorithm, as it pertains to computers, is described as a computational procedure that takes input and transforms it into output [Cor01]. Basically, an algorithm is an operation performed on some values to produce another set of values. There are many different algorithms available to solve a variety of problems by different approaches. For this application, an algorithm is necessary in order to produce the shortest path possible between one point on the graph and another. There are a number of possible algorithms that target finding a shortest path, and they share some of their techniques, such as the method named Relax, which actually checks how each edge would affect the path of the vertices it connects, if it shortens the path it is included in the branch holding the shortest path, if it lengthens it, it is not included [Cor01]. They also have their own way of approaching the problem, and are aimed at different graph types.

One of the popular algorithms that compute the shortest path between nodes in a graph is the Bellman-Ford algorithm. This algorithm deals with graphs that contain negative-weight edges [Cor01]. While searching for the shortest path, it makes sure that the graph does not contain a negative-weight cycle that is reachable from the source [Cor01]. If no such cycle exists it returns a boolean value of true, and produces the

shortest paths and their weights. If a negative-weight cycle does exist, it returns the boolean value of false, to signify no solution is possible. The main disadvantage of Bellman-Ford is that it has a large complexity $O(VE)$.

Another algorithm which is concerned with finding the shortest path in a graph is the DAG-Shortest Path algorithm. It deals with DAGs (Directed acyclic graph), and handles negative edges, but by the definition of the graph, no negative-weight cycles can occur so it does not need to deal with them [Cor01]. This algorithm sorts the graph topologically into a linear ordering of the vertices, and then it does one pass over the sorted vertices and processes each vertex by relaxing each edge leaving from that vertex [Cor01]. DAG-Shortest Path algorithm is the most efficient in terms of complexity $O(V+E)$ but we cannot use it since our graph is not acyclic.

Another important algorithm for computing a shortest path is Dijkstra's Shortest Path Algorithm. It deals with weighted, directed graphs with non-negative edges [Cor01]. The basic idea behind this algorithm is to implement a minimum priority queue, choosing from a few possible structures, like a min-heap or an array. Each of the vertices in the queue has an adjacency list that keeps track of the nodes that are directly accessible from that particular vertex. The algorithm utilizes the adjacency list as it loops over each vertex, starting from the source, extracts it from the queue and relaxes each edge coming out of it, checking if the edge should be included in the shortest path branch of the adjacent node, or if another node has provided an edge with a lower cost [Col01]. Figure 11 below is an example of the pseudocode of Dijkstra's Algorithm, and the basis for our interpretation of the algorithm. The complexity of Dijkstra is $O((V+E)\lg V)$.


```

DIJKSTRA( $G, w, s$ )
1 INITIALIZE-SINGLE-SOURCE( $G, s$ )
2  $S \leftarrow \emptyset$ 
3  $Q \leftarrow V[G]$ 
4 while  $Q \neq \emptyset$ 
5   do  $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6      $S \leftarrow S \cup \{u\}$ 
7     for each vertex  $v \in \text{Adj}[u]$ 
8       do RELAX( $u, v, w$ )

```

Figure 11: Pseudocode of original Dijkstra's Algorithm [Col01].

6.3 Dijkstra's Shortest Path Algorithm

The algorithm selected to find the shortest path from the user's current position to the desired destination is Dijkstra's Shortest Path Algorithm. From the description above, it is the algorithm that works well with our campus map graph. Our graph has no negative-weight edges and we use a min-priority queue data structure to store it. The min-priority queue is implemented using a predefined `java.util.PriorityQueue` object, in which we order the nodes by comparing each node's current distance from the source provided by the edges currently being relaxed. The node at the head of the queue, whose edges will be relaxed when the node is extracted, has the smallest distance from the source.

As is shown in Figure 12, Dijkstra's shortest path algorithm creates a tree that expands from the root (or source) out until all the vertices are contained within the branches of the tree [Dij12]. No vertex is part of more than one branch, since each is explored by the branch that creates the shortest path from the source to that vertex, and

only one branch can contain that path. In the figure, the yellow colored node is the source and the branches stem out from it until all the nodes are included.

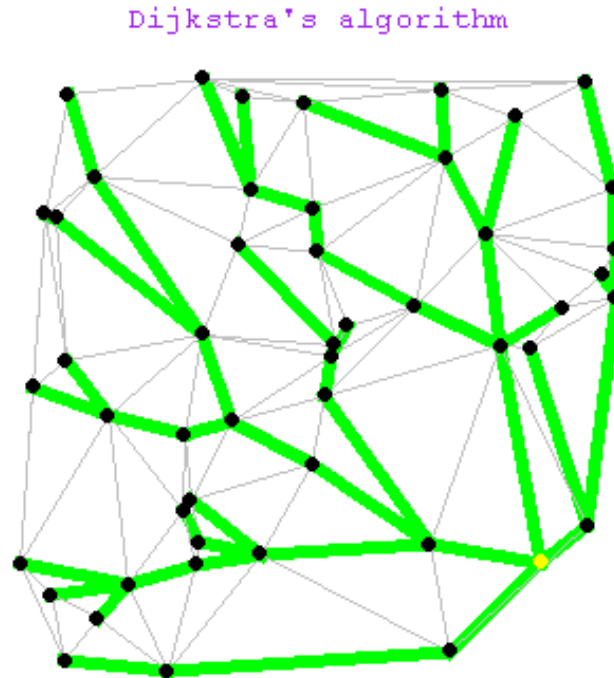


Figure 12: Dijkstra's Shortest Path Algorithm example running on a graph [Dij12].

For our application, the algorithm was optimized according to our objective. It is not necessary for every node in our structure to be included in the shortest path tree. Our interest lies with the path between the source and the user defined destination. Once the destination is reached, or is part of a branch of the shortest path tree, we stop the algorithm; that means we do not continue to discover the remainder nodes. Figure 13 shows an example of Dijkstra's Shortest Path Algorithm running until it reaches the destination.

Dijkstra's algorithm

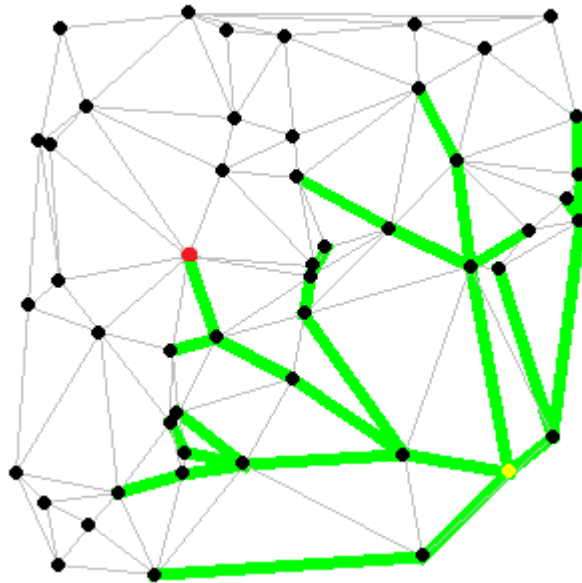


Figure 13: Dijkstra's Alg. example running up to predefined destination node[Dij12].

Another way the algorithm was altered from the typical interpretation is that since we are not exploring shortest paths to all the vertices, but just to the destination, there is no need to add all the vertices to the queue since that would just take up time, space, and add unnecessary bulk. Instead, we have chosen to begin with just the source in the queue, and then as it is removed in order to be placed into the hash table of completed vertices, the vertices it shares edges with are added to the queue.

This is the technique used for the rest of the vertices as well, until the destination is reached and the algorithm stops. Figure 14 shows the pseudocode version of our implementation of the algorithm. It extracts the vertex with the minimum distance out of the queue, and then checks to see if the vertex is the destination. If that is the case the algorithm stops, if not, the OurRELAX operation is done. This operation refers to the part

of the code where each vertex at the other ends of the segments associated with the current vertex are also extracted and the segment lengths are compared and updated, and then added back to the queue. Once this is done the algorithm continues to do this until it encounters the destination.

```
DIJKSTRA( $G, w, s$ )
1 INITIALIZE-SINGLE-SOURCE( $G, s$ )
2  $S \leftarrow \emptyset$ 
3  $Q \leftarrow S$ 
4 while  $Q \neq \emptyset$ 
5     do  $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6      $doneV[] \leftarrow u$ 
7     if  $u = \text{destination}$ 
8         do break
9     for each vertex  $v \in Adj[u]$ 
10    do OurRELAX( $u, v, w$ )
```

Figure 14: Our pseudocode version of Dijkstra's.

CHAPTER 7: COMPONENT INTEGRATION

In any project that entails multiple components, the biggest challenge is integrating the parts into one operational, cohesive whole. This is usually difficult due to the fact that problems and error arise with the introduction of every new part since it was previously only required to operate on its own. Issues with compatibility are not evident until an attempt to combine components takes place.

The best way to begin integration is to add components one at a time. Once this combined portion is working as expected another part can be added. This way the errors and compatibility problems can all be worked out on a smaller scale to ensure nothing is missed or overlooked. If the pieces are instead just thrown together all at once and the application is tested, the chances of it running are minimal and the amount of problems will un-doubtedly be so large as to cause much confusion when trying to resolve them.

7.1 Component Creation and Integration

The order in which the components of the project were created, impacted the way in which the components were integrated since our strategy was to add each component once it was functional on its own, and then begin work on a new component. The first part of the project to be tackled was the UML diagram. Since it lays out the structure of the data, it is difficult to begin any other portion of the project before this one. Once the class hierarchy was agreed upon, and the diagram was complete, the coding components

could be initiated.

The first coding component to be realized was the User Interface (UI). This component is composed of three classes, one for the activity that displays the menu options to the user, one that implements the listener that responds to the user selections, and one that implements the activity that calls all the other features of the program and displays the results to the user. This is all done with the Android packages, which access different features of the Android device, such as activities, menus, and windows for the UI. This is a logical starting point, since without this component the program cannot be tested as it would run within an Android device. To test this code, the Android testing device (Samsung Galaxy S Blaze) is connected to the computer and the run option is selected in the Eclipse IDE. As long as the device drivers are correctly installed, a menu appears and asks you to select the device. Once this is done, the app runs on the device and the UI should be visible. As soon as the app was working as expected, work on the next component could begin.

The Campus Map Tool, which allows the construction of the detailed map that supplies the XML data to be extracted, was being developed simultaneously. It was necessary in order to create the detailed campus map and be able to provide the XML to the application component of the project. As soon as the tool was completed, the creation of the in depth campus map began, and many testing versions were made.

In order to distinguish between different kinds of data obtained from the UML, the project required a class that outlined the kinds of data types of which objects could be created and stored. This class hierarchy is vital to the program and works in tandem with the parser, creating the object types of each piece of data according to the tag the parser

finds. Once the hierarchy was developed, the parser was built to create objects of the specified types as it extracted the data.

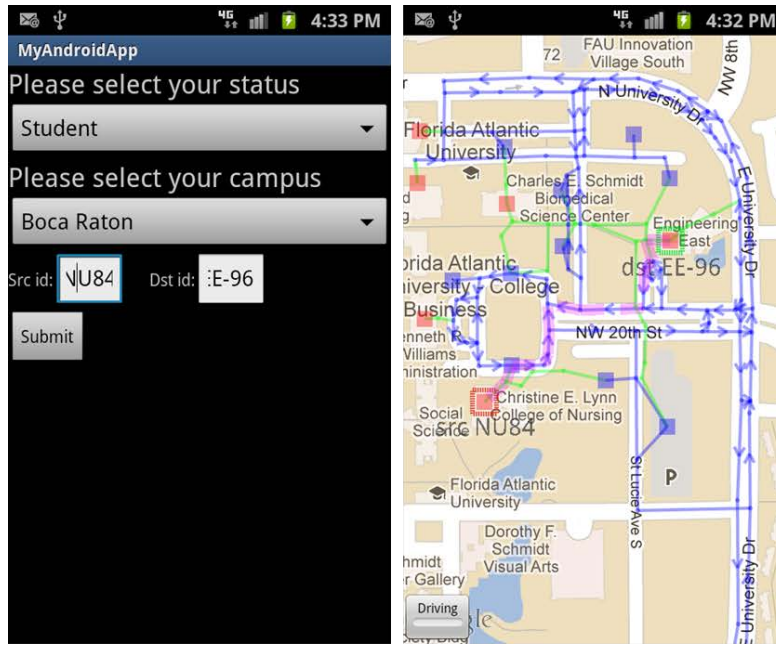
The next component tackled was the XML parser. In order for any of the other components to have the data they will need to function, the campus map XML needs to be retrieved from the server and the data needs to be broken up and stored. These tasks are done by the class implementing the DOM Parser. When assured that the parser is working correctly, the class is added to the Android UI project, and the project is tested and errors are corrected.

The next step was to create the algorithm to find the shortest path from the user's current location to the destination. For the development of this component, the first thing that had to be done was to choose an algorithm from the available shortest path algorithms. Once Dijkstra's Shortest Path Algorithm was selected, and the code was created based on the pseudocode, the component was added to the project and tested. When we were satisfied that the algorithm was working correctly, work began on the next component. The last task regards displaying the shortest route on the Google map, providing driving directions to the driver.

Figure 15 shows a scenario of running our application on an Android smartphone platform. As illustrated in Figure 15a and 15c, for testing purposes, we created two fields where the user enters the source and destination id's, and when the information is submitted, the application's different components work together as expressed above in the thesis. The final results are shown in Figure 15b and 15d, where in the user's device, a map is displayed with the route overlaid. In the left hand bottom corner of the screen of

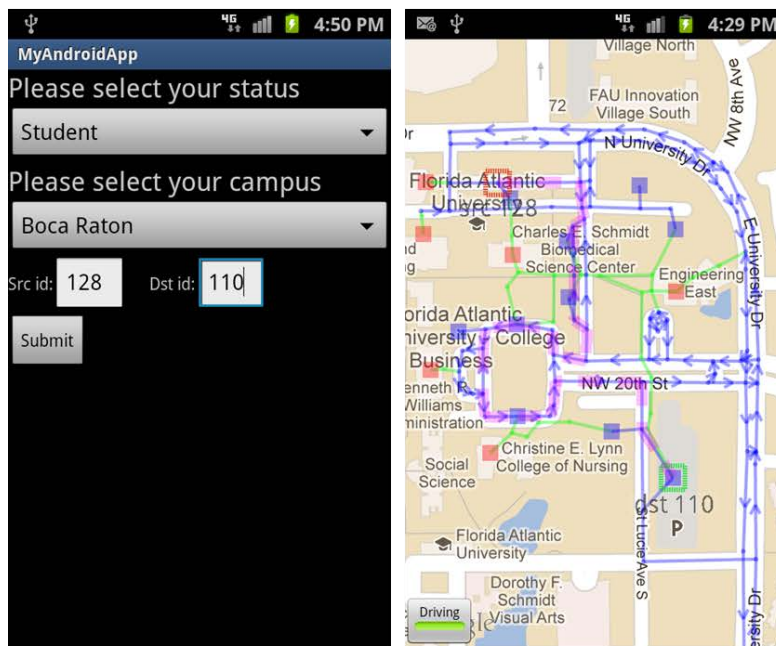
Figure 15b and 15d, there is a button marked “Drivable”, which allows the user to toggle between the driving direction and walking direction paths.

Figure 15a and 15 b display the directions required for a walkable path from the user to the destination. This is visible in the map since the walkable paths are displayed in green with the pink overlay on top of the shortest path. An example of a drivable path is shown in Figure 15c and 15d. The drivable paths on the map are shown in blue with the pink overlay signifying the shortest path to the destination. It is important to state that the drivable paths may be used in walkable routes; however the opposite is not true. Currently for testing purposes the map allows the user to click on different squares along the outlined portion in order to set the source and destination, and the route overlay, as well as the length, are calculated between the two selected vertices.



a

b



c

d

Figure 15: User interface with resulting map and path.

7.2 Difficulties with Integration

One of the difficulties we ran into when integrating was when combining the parser to the project, which at the time was basically the Android UI. Initially, the plan was to use the SAX parser. This implementation was tedious and it was hard to maintain the state of the parser with multiple levels in the hierarchy of objects. We decided to use a DOM parser and the implementation was more efficient.

Another issue when it came to integration was Dijkstra's Algorithm. Designing the priority queue structure to store the graph was challenging, because we had to provide an efficient way for nodes to access their neighbors and update their distance in the queue. Our objective was to be able to do this without using a linear traversal of the queue, which would increase complexity. Another issue was that in the priority queue data structure, changing the total cost does not rearrange/sort the nodes in the queue. To deal with this, we had to dequeue the element, relax the cost, and then insert the element back.

As expected in all implementation projects, besides design, implementation and component integration, we perform several iterations of testing and fixing the errors.

CHAPTER 8: CONCLUSIONS

In today's fast pace world, it is convenient to be able to depend on directional software to help you get to destinations promptly. Applications such as Google Maps and GPS units have become very popular with consumers for that reason [Kin11]. Just like any relatively new technology however, there is still plenty to improve on before these applications encompass all the desired options. This project focuses on providing directions inside a university campus, since applications like the ones stated above cannot direct users within the campus to specific buildings or parking lots. Since campuses are usually large and confusing, it is often hard for people to find their way around, so using a tool like GPS is very helpful. It is unfortunate that the directions provided by GPS would only direct the user to the campus, but would not be able to lead them to the specific destination inside the campus.

8.1 Conclusions on the Project

The Campus Driver Assistant on an Android Platform is an Android application written in Java programming language, and is intended for Android smart phones and devices. The app implements a user interface to establish the user type and desired destination, and then retrieves a map from the server in order to find the shortest path to that destination. It does this by parsing the XML data the map contains, which has been defined using the Map Editing tool constructed for the project. Then an optimized form

of Dijkstra's Shortest Path Algorithm is run on the graph containing the element objects from the XML. Once the path is found, it is displayed for the user in their device, overlaid on the campus map.

8.2 Future Works

The campus driving directions application can be enhanced with additional features and in the end can be integrated with other projects falling under the Campus 2020 umbrella. Some future works pertaining to our project are:

- GPS supplied source- Accessing the GPS capabilities of the phone to ascertain the current user location.
- Display detailed directions - have a text box next to the map displaying the path, with list of step by step directions.
- Dynamic directions – turn-by-turn directions that are updated depending on user movements.
- Use voice directions - implement voice instructions of directions so users are verbally alerted when to turn, or when they have reached the destination, etc.

REFERENCES

- [Ana12] Anatomy of the Linux Kernel, Jones, M.T.,
<http://www.ibm.com/developerworks/linux/library/l-linux-kernel/>, last accessed
September 2012.
- [And12] Android-Discover Android, <http://www.android.com/about/>, last accessed
September 2012.
- [AndO12] Android Open Source Project,
<http://source.android.com/about/philosophy.html>,
<http://source.android.com/index.html>,
<http://source.android.com/community/index.html>,
<http://source.android.com/faqs.html#why-did-we-open-the-android-source-code>,
last accessed September 2012.
- [AndT12] Android Timeline and Versions, [http://faqoid.com/advisor/android-
versions.php](http://faqoid.com/advisor/android-versions.php), last accessed September 2012.
- [AniX12] An Introduction to XML, <http://www.xml.com/pub/a/98/10/guide0.html>, last
accessed October 2012.
- [Cam12] Campus Maps Simex,
[https://play.google.com/store/apps/details?id=com.simexusa.campusmaps_full&h
l=en](https://play.google.com/store/apps/details?id=com.simexusa.campusmaps_full&hl=en), last accessed November 2012.

- [Cla12] Class DocumentBuilder,
<http://docs.oracle.com/javase/1.4.2/docs/api/javax/xml/parsers/DocumentBuilder.html>, last accessed October 2012.
- [Cor01] Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C., “Introduction to Algorithms”, Second Edition, 2001.
- [Dev12] Developer Tools, <http://developer.android.com/tools/index.html>, last accessed September 2012.
- [Dij12] Dijkstra’s Algorithm,
<http://www.cs.sunysb.edu/~skiena/combinatorica/animations/dijkstra.html>, last accessed October 2012.
- [Dom12] DOM versus SAX,
http://publib.boulder.ibm.com/infocenter/tpfhelp/current/index.jsp?topic=%2Fcom.ibm.ztpf-ztpfdf.doc_put.cur%2Fgtpx1%2Fdomsax.html, last accessed October 2012.
- [Ext12] Extensible Markup Language (XML), W3C, <http://www.w3.org/XML/> last accessed October 2012.
- [Goo12] Google Earth Map Data,
<http://computer.howstuffworks.com/internet/basics/google-earth6.htm>, last accessed October 2012.
- [GooM12] Google Maps Developer Documentation,
<https://developers.google.com/maps/documentation/>, last accessed October 2012.
- [Gps12] GPS.gov Roads and Highways, <http://www.gps.gov/applications/roads/>, last accessed September 2012.

- [His12] The History of Android Operating System,
<http://www.android30tablet.com/android-news/history-of-android-operating-system/>, last accessed September 2012.
- [HisO12] History of Android: First Applications, Prototypes, and Other Events, Simon Hill, 2011, <http://www.brighthub.com/mobile/google-android/articles/18260.aspx>, last accessed September 2012.
- [Ind12] Industry Leaders Announce Open Platform for Mobile Devices
http://www.openhandsetalliance.com/press_110507.html, last accessed September 2012.
- [Int12] Introduction to OMG's Unified Modeling Language (UML),
http://www.omg.org/gettingstarted/what_is_uml.htm, last accessed September 2012.
- [IntA12] Introduction to Android Development, Ableson, F.,
<http://www.ibm.com/developerworks/opensource/library/os-android-devel/>, last accessed September 2012.
- [Int(2)12] Introduction to the Unified Modeling Language,
ftp://ftp.software.ibm.com/software/rational/web/whitepapers/2003/intro_rdn.pdf, last accessed September 2012.
- [Kel12] Kelly, M., Lindquist, D., Campus Map, <http://campusmap.michaelkelly.org/map>, last accessed November 2012.
- [Kin11] Kincaid, J., Marissa Mayer: 40% Of Google Maps Usage Is Mobile (And There Are 150 Million Mobile Users), 2011,

<http://techcrunch.com/2011/03/11/marissa-mayer-40-of-google-maps-usage-is-mobile-and-there-are-150-million-mobile-users/>, last accessed October 2012.

[Nel12] Nelson, R., A Brief History of Cell Phone Design, Jun 2012,

<http://news.yahoo.com/a-brief-history-of-cell-phone-design.html>, last accessed September 2012.

[Pro12] Processing XML Data with XmlReader and XmlWriter (Silverlight),

[http://msdn.microsoft.com/en-us/library/cc189001\(v=vs.95\).aspx](http://msdn.microsoft.com/en-us/library/cc189001(v=vs.95).aspx), last accessed October 2012.

[Ref12] Reference of Android API's,

<http://developer.android.com/reference/android/package-summary.html>, last accessed October 2012.

[Sed12] Sedgewick, R., and Wayne, K., "Algorithms", Fourth Edition,

<http://algs4.cs.princeton.edu/42directed/>, last accessed October 2012.

[Ski12] Skiena, S., Lecture 10: Graph Data Structures,

www.cs.sunysb.edu/~algorithm/video-lectures/2007/lecture10.pdf, last accessed October 2012.

[Spe12] Speed delivery of Android devices and applications with model-driven

development, Holstein, B.,

<http://www.ibm.com/developerworks/rational/library/model-driven-development-speed-delivery/>, last accessed October, 2012.

[Str12] Street View Partner Program,

<http://maps.google.com/help/maps/streetview/partners/>, last accessed November 2012.

- [TheA12] The Apache Xerces Project-xerces.apache.org <http://xerces.apache.org/>, last accessed October 2012.
- [TheE12] The Expat XML Parser, <http://www.libexpat.org/>, last accessed October 2012.
- [TheH12] The History of Maps,
<http://www.1worldglobes.com/History/historyofmaps.htm>, last accessed October 2012.
- [Thi04] Thiruvathukal, G.K., XML in Computational Science, ©2004
http://ecommons.luc.edu/cgi/viewcontent.cgi?article=1008&context=cs_facpubs&sei-redir=1&referer=http%3A%2F%2Fwww.google.com%2Furl%3Fsa%3Dt%26rct%3Dj%26q%3Dparsing%2520xml%2520in%2520computer%2520science%26source%3Dweb%26cd%3D3%26ved%3D0CD4QFjAC%26url%3Dhttp%253A%252F%252Fecommons.luc.edu%252Fcgi%252Fviewcontent.cgi%253Farticle%253D1008%2526context%253Dcs_facpubs%26ei%3DepN0UM7jI4TVrQG20IG4CQ%26usg%3DAFQjCNFTakFvwJB_gqCq6DAUiEpesT7Afw#search=%22parsing%20xml%20computer%20science%22 , last accessed October 2012.
- [Tig12] Tigris.org Open Source Software Engineering Tools, <http://argouml.tigris.org/>, last accessed September 2012.
- [UmlH12] UML History, http://www.sa-depot.com/?page_id=217, last accessed in September 2012.
- [UmlR12] UML Resource Page, <http://www.uml.org/#Articles>, last accessed September 2012.

- [UmlT12] UML tutorial: Part One—Class Diagrams. Robert C. Martin,
<http://www.objectmentor.com/resources/articles/umlClassDiagrams.pdf>, last
accessed September 2012.
- [Vie12] Viescas, A., What Does it Mean to Parse Data?
http://www.ehow.com/info_10021819_mean-parse-data.html, last accessed
October 2012.
- [Wel12] Welcome to the project site for the UCSB Interactive Campus Map,
<http://code.google.com/p/ucsb-icm/>, last accessed November 2012.
- [Wha12] What would it take to build a better mobile phone?
<http://www.openhandsetalliance.com/index.html>, last accessed September 2012.
- [Woo12] Wood, T.D., How to Use a Hand Held GPS Receiver, Aug 2012,
<http://www.rei.com/learn/expert-advice/gps-receiver-howto.html> last accessed
September 2012.
- [XmlD12] http://www.w3schools.com/dom/dom_parser.asp, last accessed October 2012.
- [XmlP12] XML Parser, http://www.w3schools.com/xml/xml_parser.asp, last accessed
October 2012.