

# **BUILDING A NOTIONAL APPLICATION ATOP AN OPEN SOURCE CLOUD PLATFORM TO MANAGE WEATHER INFORMATION**

by

MANIEL J SOTOMAYOR-RODRÍGUEZ

A project report submitted in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING  
in  
COMPUTER ENGINEERING

UNIVERSITY OF PUERTO RICO  
MAYAGÜEZ CAMPUS  
2010

Approved by:

---

Pedro Rivera-Vega, PhD  
Member, Graduate Committee

---

Date

---

Jaime Seguel, PhD  
Member, Graduate Committee

---

Date

---

Manuel Rodríguez-Martínez, PhD  
President, Graduate Committee

---

Date

---

Lev Steinberg, PhD  
Representative of Graduate Studies

---

Date

---

Erick E Aponte, PhD  
Chairperson of the Department

---

Date

## **ABSTRACT**

### **BUILDING A NOTIONAL APPLICATION ATOP AN OPEN SOURCE CLOUD PLATFORM TO MANAGE WEATHER INFORMATION**

By

*Maniel J. Sotomayor Rodríguez*

GeoCloudNap is a notional application designed and developed in this research project. It is a Web-based application to be deployed atop a Cloud Platform designed to manage weather information obtained from distributed sources throughout the Web. The Web application permits to geographically locate points of interest and store their weather conditions on a geospatial database. GeoCloudNap leverages well-known architectural paradigms and design patterns to provide a fully functional system. Key concepts and technologies to build the system are presented along with a description of their implementation.

## **RESUMEN**

### **DESARROLLO DE UNA APLICACIÓN NOCIONAL SOBRE UNA PLATAFORMA “CLOUD” ABIERTA PARA MANEJAR INFORMACIÓN DEL CLIMA**

Por

*Maniel J. Sotomayor Rodríguez*

GeoCloudNap es una aplicación nocional diseñada y desarrollada como parte de este proyecto de investigación. Este es una aplicación tipo Web a ser desplegada sobre una plataforma “Cloud” diseñada para manejar información obtenida de fuentes distribuidas a través del Web. La aplicación tipo Web permite localizar geográficamente puntos de interés y almacenar sus condiciones de clima en una base de datos geo-espacial. GeoCloudNap utiliza paradigmas arquitecturales y patrones de diseño bien conocidos para proveer un sistema completamente funcional. Conceptos y tecnologías claves para construir el sistema son presentados junto con una descripción de su implementación.

Copyright © by

Maniel J. Sotomayor-Rodríguez

2010

Non nobis, Dómine, non nobis:  
sed nómini tuo da glóriam. Psalmus 113

## **ACKNOWLEDGEMENTS**

To: my Lord, family and friends: for your love, prayers, support and understanding.

To: my spiritual director Fr Angel Santos: for your prayers, guidance, friendship and wisdom.

To: Fr Rafael Capó, SchP and Fr Friar Luis Padilla, OFMCap: for your prayers, support and friendship.

To: my graduate committee: Dr Pedro Rivera-Vega, Dr Jaime Seguel and especially, my advisor Dr Manuel Rodríguez-Martínez, for your time and knowledge, and for granting me this opportunity.

To: my professors: for your dedication.

To: my colleagues: for sharing your insights and all the laughs.

Thank you, God bless you all.

# TABLE OF CONTENTS

ABSTRACT.....	II
RESUMEN .....	III
ACKNOWLEDGEMENTS .....	VI
TABLE OF CONTENTS .....	VII
FIGURE LIST.....	VIII
<b>1 INTRODUCTION .....</b>	<b>1</b>
1.1 PROBLEM STATEMENT .....	1
1.2 PROPOSED SOLUTION: GEOCLOUDNAP .....	2
1.3 CONTRIBUTIONS .....	4
1.4 REPORT ORGANIZATION .....	4
<b>2 BACKGROUND .....</b>	<b>5</b>
2.1 GOOGLE MAPS.....	5
2.2 GOOGLE WAVE.....	7
2.3 APPLE IOS SDK.....	9
2.4 DJANGO WEB FRAMEWORK.....	10
2.5 EUCALYPTUS SOFTWARE INFRASTRUCTURE.....	11
<b>3 GEOCLOUDNAP SYSTEM OVERVIEW .....</b>	<b>13</b>
3.1 GOALS .....	13
3.2 COMPONENTS .....	14
3.3 GEOCLOUDNAP ARCHITECTURE .....	16
3.4 PRESENTATION TIER OVERVIEW.....	20
<b>3.4.1 User Interface on Google Wave.....</b>	<b>23</b>
3.5 APPLICATION LOGIC TIER OVERVIEW .....	25
3.6 DATA MANAGEMENT TIER OVERVIEW.....	26
<b>4 GEOCLOUDNAP IMPLEMENTATION .....</b>	<b>28</b>
4.1 PRESENTATION TIER IMPLEMENTATION .....	28
<b>4.1.1 User Interface on iPhone.....</b>	<b>35</b>
4.2 APPLICATION LOGIC TIER IMPLEMENTATION .....	37
4.3 DATA MANAGEMENT TIER IMPLEMENTATION.....	41
4.4 SERVICE PROVIDERS AS VIRTUALIZED TIERS .....	44
<b>5 SUMMARY AND CONCLUSIONS .....</b>	<b>47</b>
5.1 FUTURE WORK .....	49
REFERENCES .....	51

## FIGURE LIST

Figures	Page
Figure 1.1: Screenshot of the user interface of a GeoCloudNap prototype .....	2
Figure 2.1: Screenshot of the user interface of Google Maps.....	6
Figure 2.2: Screenshot of the user interface of Google Wave .....	8
Figure 2.3: Diagram of iOS technology layers .....	9
Figure 3.1: Diagram of client technology layers .....	14
Figure 3.2: Diagram of service providers technology stack .....	15
Figure 3.3: Diagram of three-tier architecture .....	17
Figure 3.4: Diagram of web system within cloud infrastructure .....	18
Figure 3.5: Screenshot of description of the GeoCloudNap user interface .....	20
Figure 3.6: Screenshot of user interface of GeoCloudNap prototype within Wave .....	24
Figure 4.1: Diagram of technology stack of presentation tier .....	28
Figure 4.2: Screenshot of user interface of GeoCloudNap .....	29
Figure 4.3: Code snippet to load Google Maps API .....	30
Figure 4.4: Code snippet to draw information window of markers.....	31
Figure 4.5: Code snippet to dynamically generate menu for selecting marker types .....	32
Figure 4.6: Screenshot of mobile user interface for iPhone.....	36
Figure 4.7: Diagram of technology stack of application logic tier .....	37
Figure 4.8: Code snippet of object data models for Place and Marker entities .....	39
Figure 4.9: Diagram of technology stack of data management tier.....	41
Figure 4.10: Diagram of ORM data model of the GeoCloudNap relational schema .....	43
Figure 4.11: Diagram of virtualization of service provider tiers by IaaS cloud .....	44
Figure 5.1: Diagram of multi-cloud application architecture .....	49



# 1 INTRODUCTION

Cloud computing [9] is a promising paradigm designed to harness the power of computer networks and communications in a more cost effective way [11]. Clouds provide elastic capacity to serve a wide and constantly expanding range of information processing needs, including government, military, business and education. The Cloud Computing paradigm is maturing rapidly and is being considered for adoption in government and business platforms [17]. A diversity of Cloud Computing services as well as deployment technologies have emerged as proprietary or open source systems [3]. Open source systems refer to software systems whose source code is publicly available, allowing for immediate incorporation of improvements and adaptations of the system by its users. This project reports on GeoCloudNap, a cloud based, multi-client, web-based, notional application built to manage weather information [20].

## 1.1 Problem Statement

Organizations with diverse duties need to effectively respond and handle particular eventualities under different weather conditions. When these eventualities require the intervention of multiple organizations, collaboration and coordination between them is indispensable. Thus, indistinctly of the subject location, any eventuality bounded to a geographical extension can be dealt with. However, no actual tool aids their collaboration by providing a scalable, intuitive and accessible application to access relevant weather

information [20]. GeoCloudNap is proposed as the fitting interface for managing and persisting vital information on weather conditions.

## 1.2 Proposed Solution: GeoCloudNap

GeoCloudNap is a notional application that provides users with information about weather conditions at various geographic locations. Refer to Figure 1.1 to see its user interface.

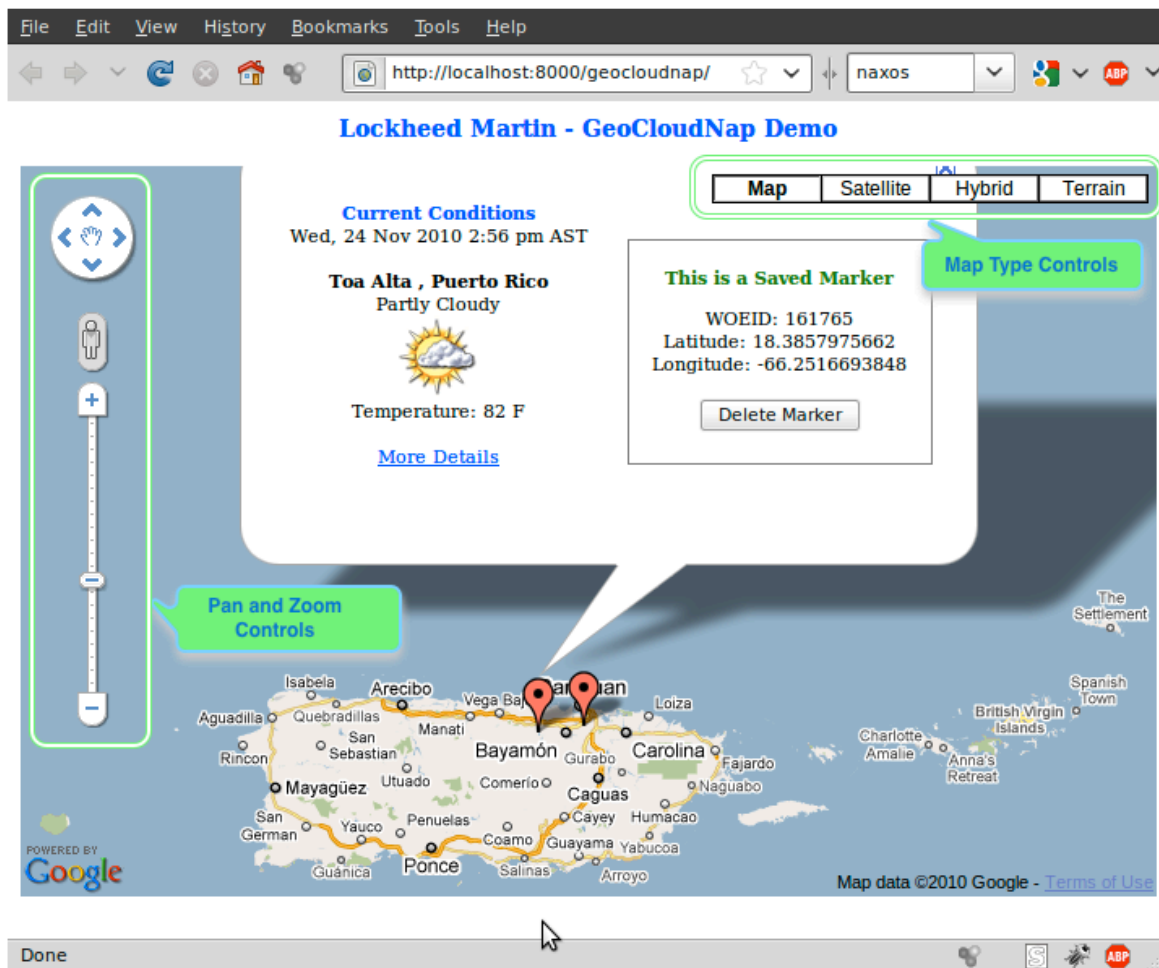


Figure 1.1: Screenshot of the user interface of a GeoCloudNap prototype

The application combines open source software packages for scalable distributed computing along with various web components. The web components provide the user interface whereas a database management system executes common operations to keep the weather data and the system metadata.

GeoCloudNap enables a user to click on a map location and mark it to save its geospositional World coordinates, its current weather conditions and its WOEID. The WOEID acronym stands for *Where On Earth ID* [22], a concept introduced as part of the web service *Yahoo! GeoPlanet*. The purpose of the WOEID is to assign to each place or entity a unique identifying number within the system. Each identifier can be used to query the web service *Yahoo Weather* [23] for current weather conditions and for the next day forecast. With the aid of the WOEID and having marker locations saved, relevant data values such as the place description can be retrieved from the database system. Also, historic data such as rainfall and humidity may be obtained from several distributed repositories.

Furthermore, the development of GeoCloudNap as a Google Wave extension would be convenient to enhance its use within a collaborative and multi-user environment. The Google Wave platform would provide the communication environment of multiple users while working on a task. This would allow users to coordinate tasks instantly while exchanging feedback through the Google Wave interface and the map display altogether. To benefit from it, the enhanced GeoCloudNap code would require to be incorporated into the Google Wave platform. From here, the following report documents the endeavors and whereabouts of the development of GeoCloudNap.

## **1.3 Contributions**

This project makes the following contributions:

1. Design of a web-based application atop a virtualized three-tier architecture.
2. Gathering and storing of weather information from third party web services.
3. Supporting of geospatial data types to accelerate the development.
4. Use of geospatial data types to support multidimensional data objects.
5. Sharing of a persisted marker set to promote collaboration among users.
6. Enable integration with other cloud platforms such as Amazon EC2.

## **1.4 Report Organization**

The rest of this report is organized as follow. Chapter 2 provides some background information and brief descriptions on the main technologies employed. Chapter 3 introduces key architectural concepts of GeoCloudNap along with a system overview. Chapter 4 provides details on the technical aspects of the notional application architecture and its components. Chapter 5 proposes additional features to be implemented as future work. Finally, chapter 6 concludes this project report with a summary.

## **2 BACKGROUND**

The following section introduces all major technologies employed for building and deploying a production release of the GeoCloudNap web application.

### **2.1 Google Maps**

The map service provided by the Google Maps web-based application has been available for more than five years now. It has been very popular ever since for locating places and businesses, getting route directions and even for planning trips. The Google Maps software components are freely supplied to developers as two API versions (i.e., version 2 and 3). These APIs are written in the JavaScript language, which lets developers embed the mapping services into web pages and make them freely available to consumers. Both APIs provide utilities for manipulating the maps and adding content to them as well.

Regarding its user interface, Google Maps projects a two-dimensional top view of a user-selected geographical portion of the earth surface. This top view of the map is then overlaid by various graphical controls that enable users update its image accordingly. For example, the Map Type Selection Controls, allows users select the map aspect from among multiple perspectives (e.g., roads map, terrain map, etc.) Also, there is the Pan and Zoom controls that permit users move through the map from a location to another or to bring nearer or farther the altitude view (i.e., zoom in or out.) Besides, content can be added as well as being associated to user-selected locations. For this, Google Maps uses markers to pinpoint

user-selected places, which display the associated information in a small window when clicked by users. Figure 2.1 displays a snapshot of the Google Maps user interface.

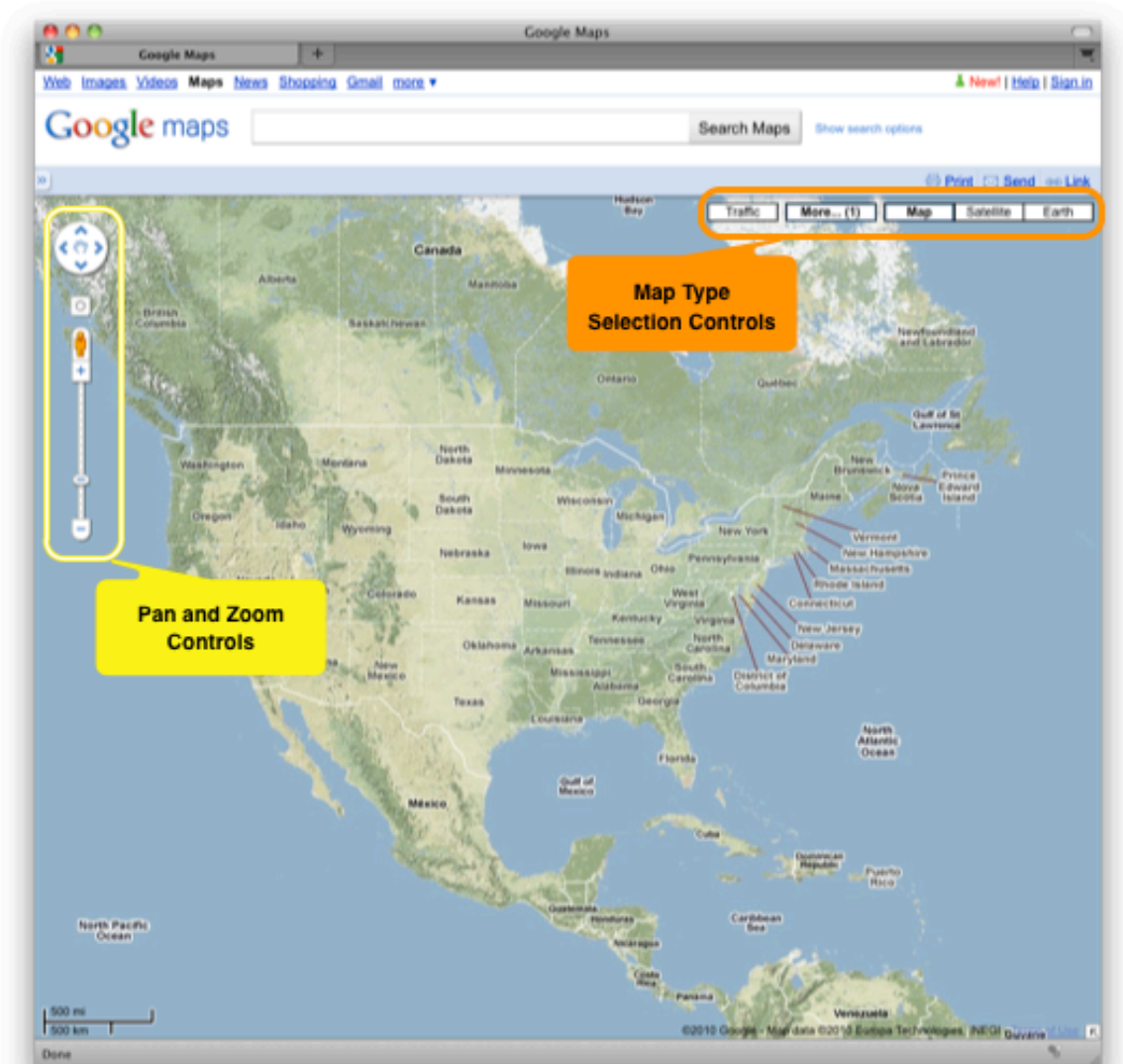


Figure 2.1: Screenshot of the user interface of Google Maps

## 2.2 Google Wave

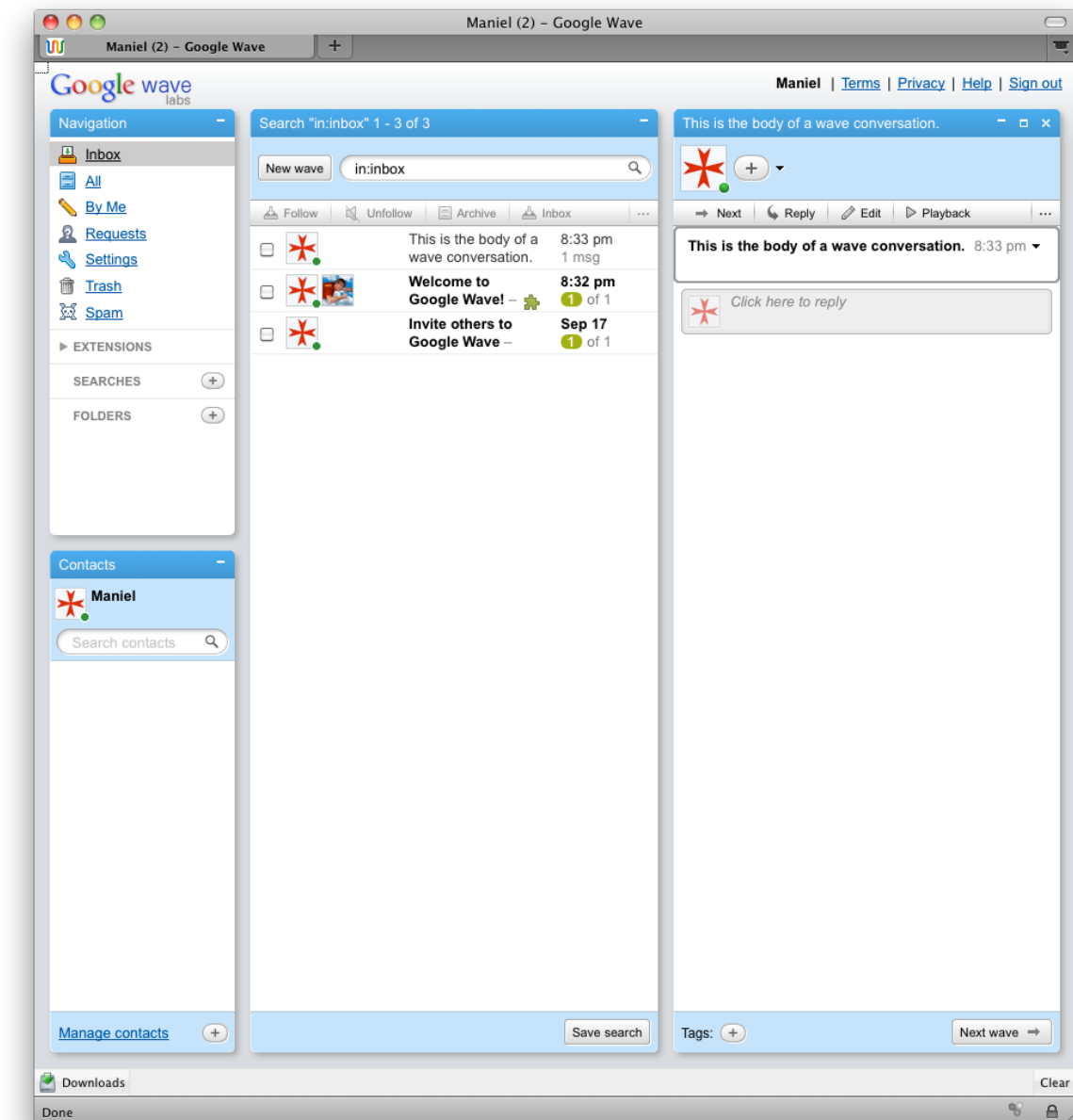
It has been more than a year since Google revealed Google Wave at their 2009 “Google I/O” web developer conference [7]. There, Google expressed their intention to open sourced Google Wave so that interested developers could help them mature and extend it. Meanwhile, Google would continue developing it and providing support to all subscribed developers.

Google Wave is a web-based application written in JavaScript that allows real-time communication and collaboration between several participants. The Wave concept is similar to a multi-party chat where its posted contents are editable by all participant users as a live and shared document. Furthermore, the content can include multimedia that can be rendered and received simultaneously by all parties in real-time.

The Google Wave platform provides developers with open APIs to build Google Wave extensions. Wave extensions aim at augmenting functionality to waves, which are to be integrated inside the platform itself. There are two types of Wave extensions, robots and gadgets; whose are combinable within a single wave. Robots are applications that can automate tasks as a wave participant as well as interact with the conversation. And gadgets are shared web applications that run within the wave and are accessible to all participants. The former runs on application servers (currently on Google AppEngine only) and the latter runs within wave clients.

To use a desired extension, users have to install them within the wave. Now, for developing these extensions, Google provides the Google Wave Sandbox. The Wave Sandbox is like a workspace equipped with development tools within Google Wave itself.

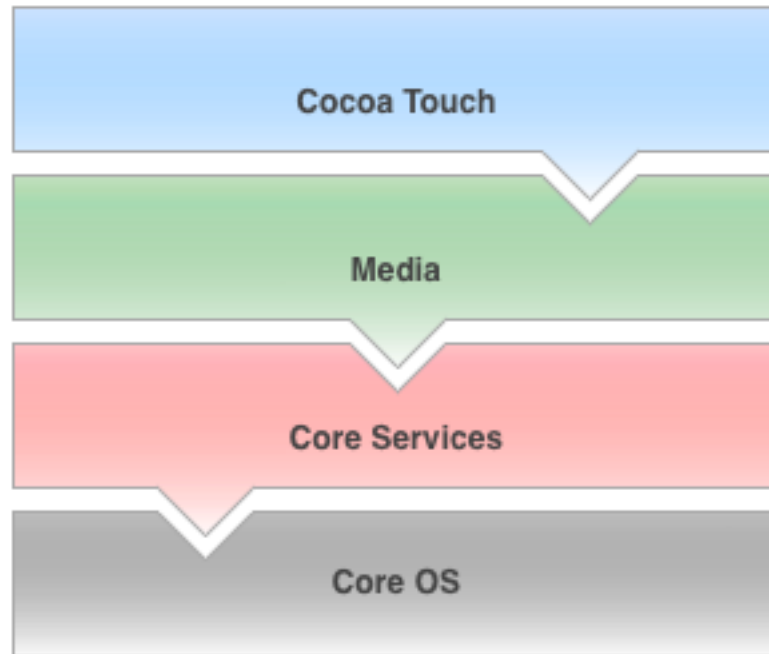
This way, developers access not only the entire functionality of Google Wave but can also reach a deployment platform for testing their extensions. Figure 2.2, illustrates the user interface of the Google Wave web application.



**Figure 2.2: Screenshot of the user interface of Google Wave**



## 2.3 Apple iOS SDK



**Figure 2.3: Diagram of iOS technology layers**

iOS comprises the underlying application infrastructure of Apple mobile devices such as the iPad, iPhone and iPod touch. It provides the operating system, the touch event handling and other technologies to support hardware features such as the accelerometers and the multi-touch interface. To develop on this platform, the iOS SDK provides the code libraries and tools needed to build and run applications for iOS [13]. The iOS is composed of four service layers that its organization can be visualized as a vertical stack of technologies (See Figure 2.3). Each technology layer exposes a set of well-defined interfaces to its respective upper next layer so that each can be accessible as in a top-to-bottom fashion. A brief description of each follows:

- **Core OS** – provides low-level data-types, fundamental input and output, and POSIX threads.
- **Core Services** – provides connectivity services (e.g., network sockets) among others.
- **Media** – provides fundamental technology to enable 2D and 3D drawing support as well as multi-media support.
- **Cocoa Touch** – provides the most fundamental infrastructure for applications (e.g., sophisticated data types, visual controls and access to hardware feature).

## 2.4 Django Web Framework

The Web can be distinguished by the following two essential trends; web content changes quickly, and new web services emerge steadily. These trends force the development of technologies to encourage software developers excel the Web capabilities. Django, as one of these technologies, is a framework built on Python, an interpretable scripting and high-level language. Within four years, the Django Web framework has surfaced as a promising technology for developing web applications rapidly. As a framework, it fosters a pragmatic and clean approach to build dynamic web applications with the help of several application programming interfaces (i.e., APIs.) The following comprises the Django Web framework, from which the first four are the most fundamental for developing and starting up swiftly with a fully functional web application:

- **Object-Relational Mapper (ORM) API** – provides dynamic access to SQL databases through the definition of Python object-based data models.

- **Automatic admin interface** – provides an intuitive user interface for adding and updating content within an administrative web page.
- **Elegant URL design** – provides a flexible method for designing the navigable URL addresses of web pages.
- **Template system** – provides a template language to separate the layout design and content from the Python code.
- **Cache system** – provides a caching mechanism for enhancing the web application performance.
- **Internationalization** – provides support for multi-language applications and language-specific functionality.

## 2.5 Eucalyptus Software Infrastructure

Eucalyptus [5] [16] is an open-source cloud software infrastructure deployable on Linux Operating Systems. This software platform implements a Cloud Infrastructure as a Service (IaaS), which is one of three service models that the Cloud Computing model is composed of [12] [14]. The Cloud Computing model empowers the swift provisioning of shared computing resources such as servers, storage and services through computer networks [14]. Equipped with these, IaaS allows consumers to deploy and control arbitrary operating systems and applications software. This in turn, alleviates consumers from worrying about managing the underlying IT infrastructure (e.g., networks, firewalls, other hardware, etc).

From among many of its benefits, our project advantaged on the following listed:

- **Allows building private clouds** – for testing the deployment of new features in applications without disturbing the production versions.
- **Deployable on legacy hardware and software** – Existing hardware can be used to build a private cloud on premise. Likewise, existing software can be migrated to virtual machines running on an existing cloud [21].
- **Compatible with several Linux kinds** – New servers instances for specific Linux distributions can coexist in the cloud.
- **Transition between private and public clouds** – Web applications that are hosted in servers located in private and public cloud can be easily managed.
- **Allows elastic server assignment** – New servers can be added on demand, as needed to meet the growth in user requirements [15].

## **3 GeoCloudNap SYSTEM OVERVIEW**

### **3.1 Goals**

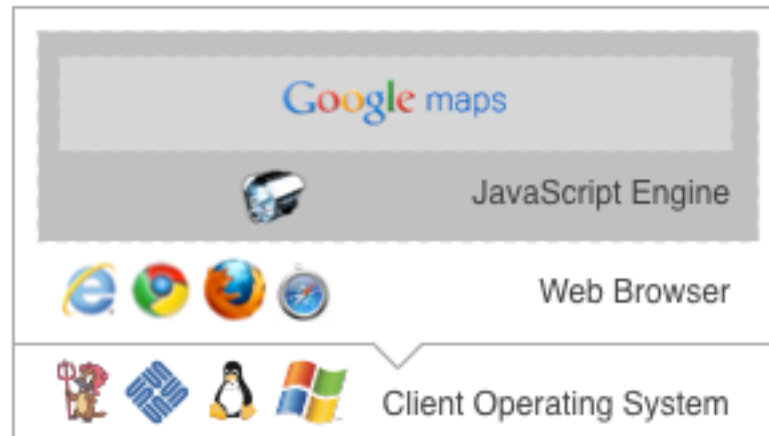
The project aims are:

- To enable users mark off and save places of interest on a map.
- To store weather information about marked off places.
- To help users store specific information about marked off places.
- To deploy the system as a group of virtual hosts within a cloud.

## 3.2 Components

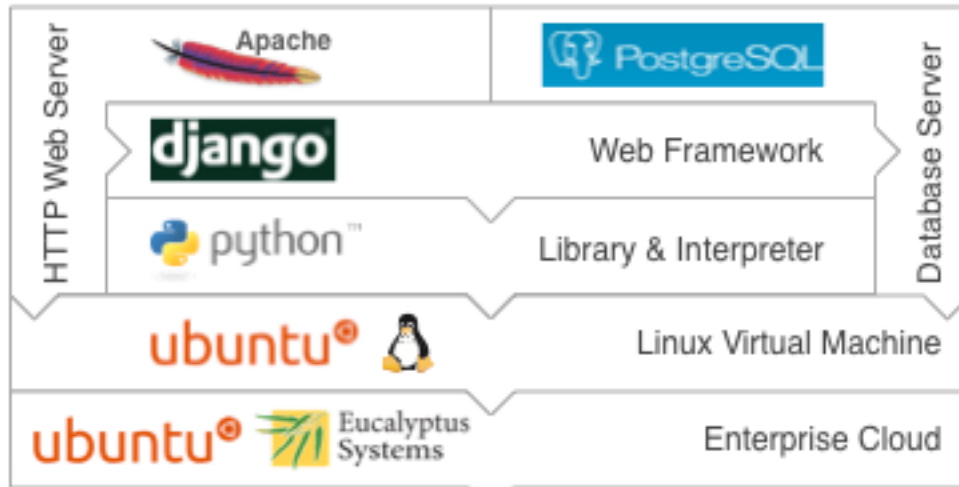
The components of the web-based application were built on the following technologies (refer to Figure 3.1 and Figure 3.2 to see their relative dependency stack):

- **Google Maps API v3** – a software package written in JavaScript to embed within web browsers mapping services along with graphical map displays.
- **Google Wave** – a JavaScript written web-based application platform for enabling communication and collaboration [7].



**Figure 3.1: Diagram of client technology layers**

- **Django Web Framework** – a software framework written in Python for building RESTful [6] web-based applications.
- **Apache Web Server** – an Apache Foundation open source web server.
- **PostgreSQL DBMS** – an open source SQL database management system.
- **Linux Operating System** – an UNIX variant open source operating system.
- **Eucalyptus Software Platform** – an open-source cloud computing system for implementing various kinds of infrastructures as a service.



**Figure 3.2: Diagram of service providers technology stack**

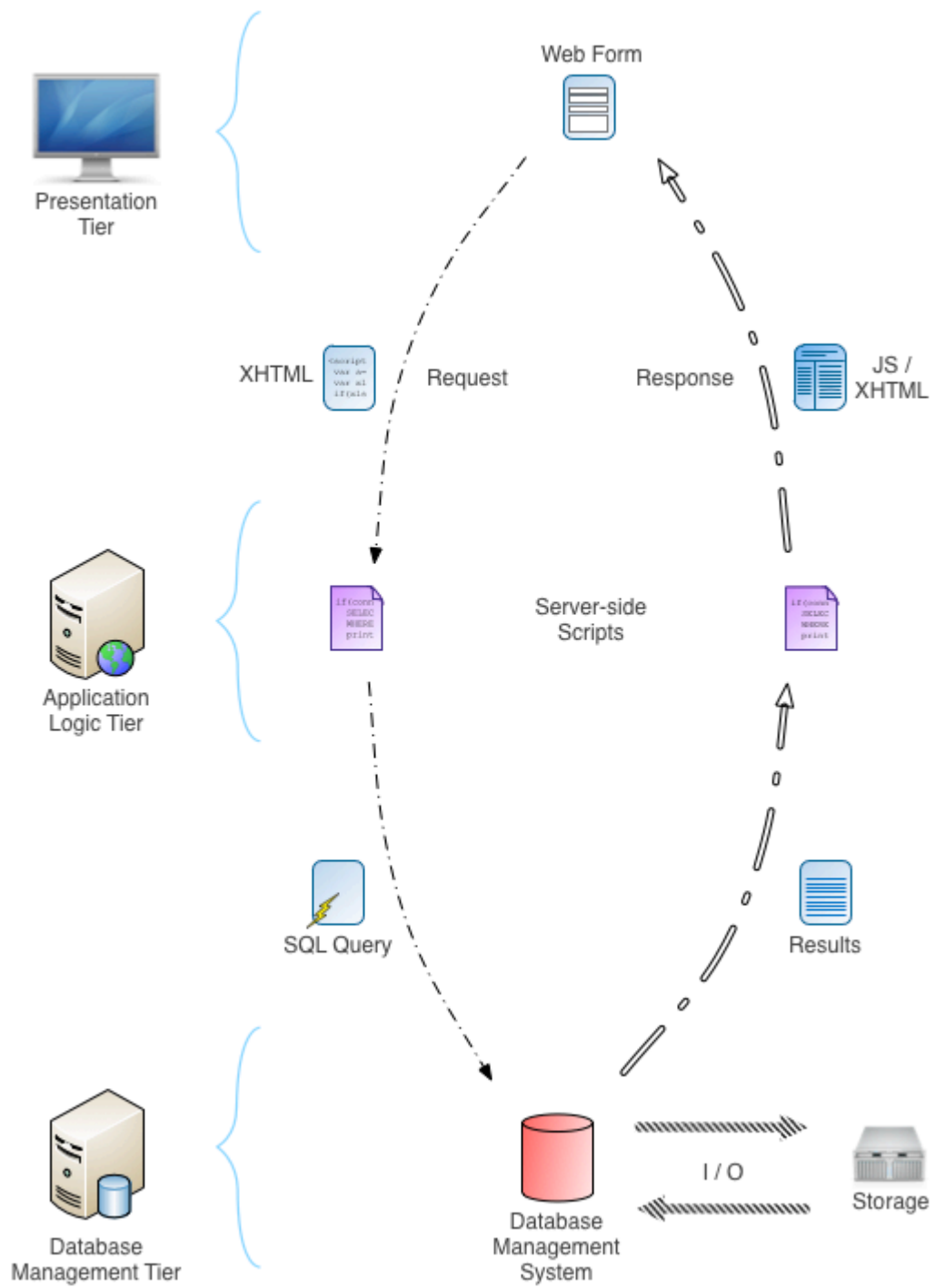
### 3.3 GeoCloudNap Architecture

GeoCloudNap stands for Geographic Cloud-based Notional Application, an acronym name that describes the application that serves the basis of this project. GeoCloudNap, is a web-application built on Google Maps, designed to manage different kinds of weather data and to run within a Cloud Computing infrastructure. Basically, the GeoCloudNap web application enables users to mark off places of interest and monitor their weather conditions. The data for weather conditions related to a particular place is gathered on-demand through third-party weather web services freely available from the Web (e.g., Yahoo Weather Services).

GeoCloudNap employs the three-tier computing model. This model establishes that the system functionality and associated workloads must be allocated and segregated as a multi-tier architecture between three separated processes. These processes, also named tiers, are identified as: the Presentation Tier (PT), the Application Logic Tier (ALT) and the Data Management Tier (DMT). Follow Figure 3.3 to see a depiction of the Three-tier architecture.

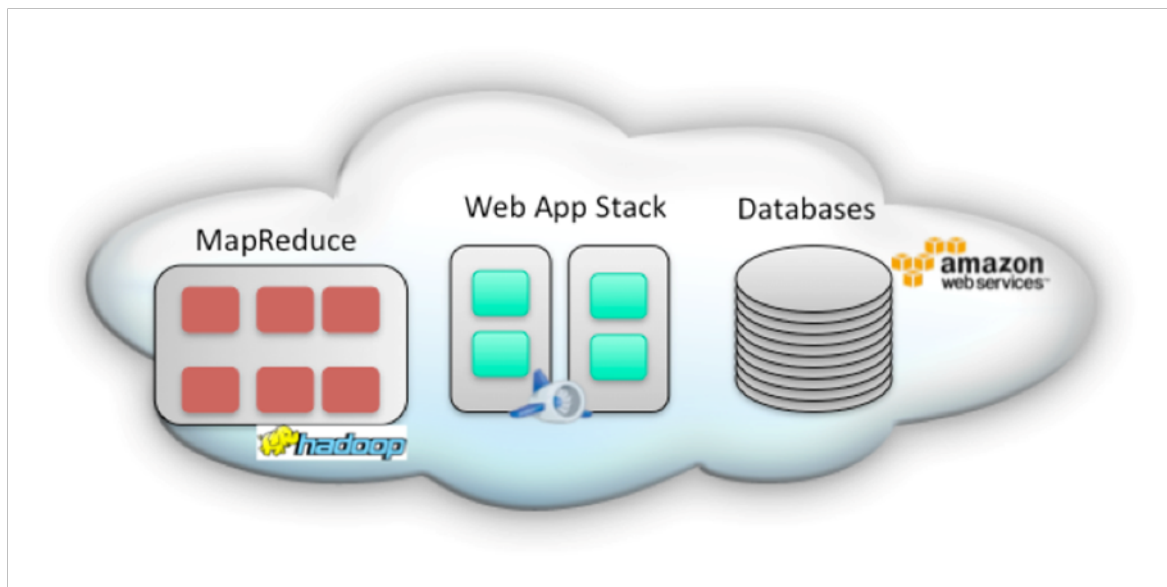
The PT main focus is to provide users with an interaction tool set to perform tasks intuitively, and to exhibit an understandable portrayal of those task results. That is, it provides the Graphical User Interface (GUI) and its associated tools to interact directly with. The second process (i.e., the ALT), concerns command processing requested by the PT and moving data along itself to make it accessible to the other two tiers. The DMT, being the third in the model, stores and retrieves data from its hosted database(s) to feed it back into the ALT to eventually provide it to users.





**Figure 3.3: Diagram of three-tier architecture**

Both, the application logic and data management tier are run on server platforms with plentiful resources for dispatching data to requestors. Thus, similarly to a simple client-server model, the Presentation Tiers, just as clients, act as scattered computers or devices that require consuming resources from service providers to be able to complete a task. These three separate processes operate physically through an interconnected network of computers. However, even though the traditional approach to implement the Three-tier architecture is by using three different physical hosts, GeoCloudNap virtualizes the ALT and DMT tiers. The virtualization [21] of these requires specific technologies to provide a virtual infrastructure so as to enable them to intercommunicate, that is, a Cloud Platform (Details on Section 4.4). Figure 3.4 shows a conceptual example of the systems virtualized within a Cloud Infrastructure.



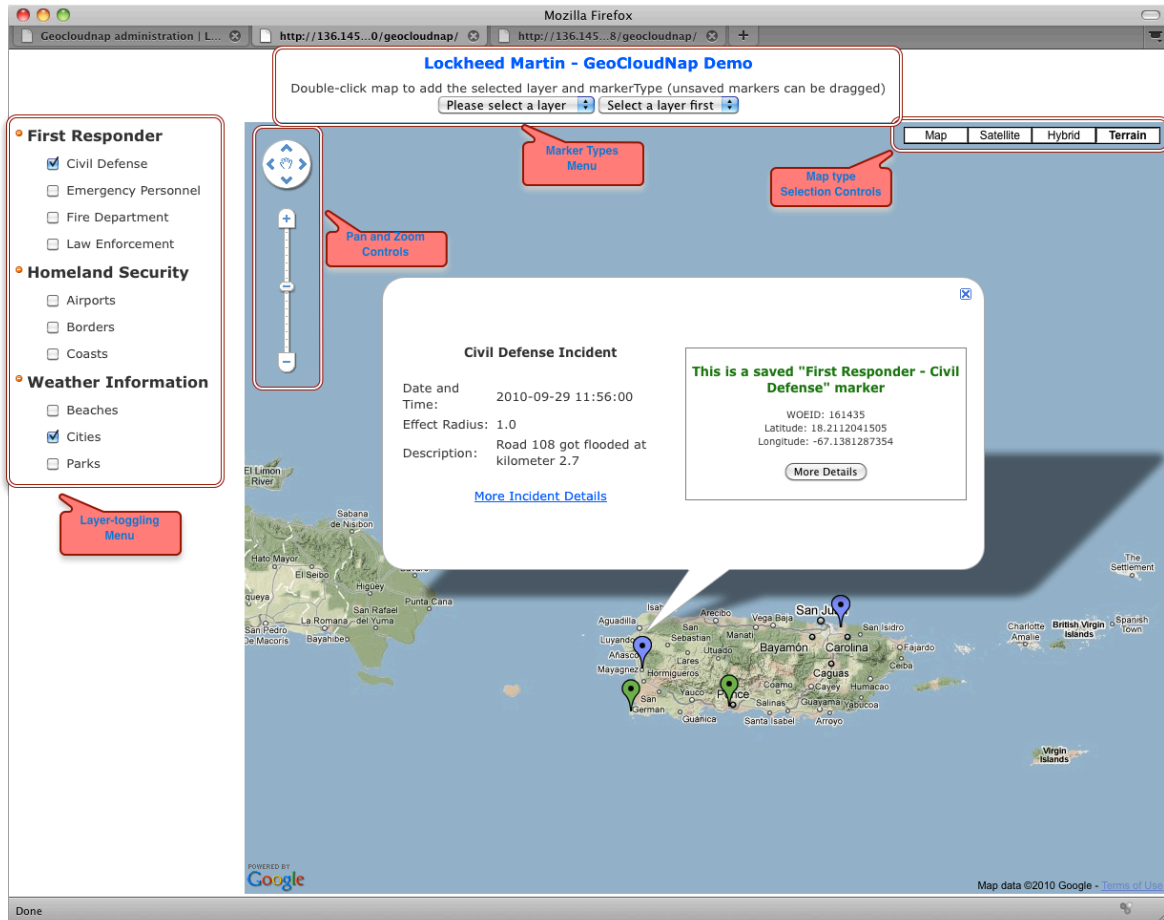
**Figure 3.4: Diagram of web system within cloud infrastructure**

Programmatically, GeoCloudNap implements these processes using the concepts of the **Model-View-Controller** (MVC) design pattern [2]. MVC is an architectural design

pattern that isolates the application logic (virtually hosted by the ALT) from the GUI (within PT). The MVC design pattern is ideal for GeoCloudNap due to its conceptual compatibility with the Three-tier architecture. It mainly benefits GeoCloudNap because it emphasizes on the separation of concerns. Among all benefits, the most relevant are that it permits to build and test each role (concern) independently. GeoCloudNap is organized as the components of the MVC design pattern in the following manner:

- **View** – Role implemented in the Presentation Tier. The user interface is built on Google Maps and JavaScript. It provides users with a map with a marker set where each marks a place for which a measurement of weather conditions is available.
- **Controller** – Role implemented in the ALT. The controller is built on Django as Python modules executed by the Python Interpreter, which collaborates with the Apache Web Server.
- **Model** – Assumes the role of the Data Management Tier. The model keeps the information about the weather measurements. For each available point, the latitude, longitude, temperature, conditions, and time of measurements are kept. Data is extracted from Yahoo! Weather service, and from other weather repositories in the Internet. Part of the Django-based model holds the metadata about system collections and users. This model supports three data stores:
  - a) SQLite – an embedded database library
  - b) PostgreSQL – a relational database engine, and
  - c) Google' Python Datastore for App engine.

### 3.4 Presentation Tier Overview



**Figure 3.5: Screenshot of description of the GeoCloudNap user interface**

The presentation tier hosts the user interface of GeoCloudNap. The user interface of the presentation tier is a web client that runs on any standard web browser. It consists of a webpage that embeds a customized map widget of Google Maps (See Figure 3.5) the main feature of the user interface. The map display widget enables a user to click on any place, mark it off with a marker icon and save it for future reference. Users, in turn, can associate information to markers while creating and saving them onto any map location.

By default, all markers associate weather information such as the current temperature and weather conditions. However, users can also associate other data fields to markers, depending on the particular type of each marker, for all markers have a specific type. For this, GeoCloudNap establishes several predefined marker types to enable users manage different kinds of information. The specific information that each marker type has, is pre-established and modeled as part of the application data model hosted by the data management tier (discussed later in Section 3.6).

Currently, GeoCloudNap defines ten marker types, which are organized and grouped into three different layers. This concept was introduced into GeoCloudNap to agglomerate a certain group of markers as a layer that could be of a special interest to users. The grouping of these markers would then, help users toggle their visibility on the map display so as to help them focus on specific marker characteristics. The toggling feature could then be of great convenience, especially when excessive cluttering eventually appears throughout a region of high interest. Marker characteristics would then be determined by the specific data that each marker type had defined. However, GeoCloudNap is designed in such a flexible way that the association of markers with layers can be done by application administrators through a separate and intuitive webpage (discussed later in Section 4.3).

When users click on a marker already saved, an information window with an appearance of a callout pops out of the map showing the specific properties of the marker given its marker type. For example, following Figure 3.5, a callout can be seen emerging from a purple marker displaying information relevant to a civil defense incident. Similarly to this

information window, users, submit specific data values into input fields corresponding to the selected marker given its type.

Concerning map appearance and navigability, the rectangular map display empowers users to easily explore maps of the Earth surface. Through it, and aided with intuitive controls within its left corner, users can move and navigate thoroughly (i.e., pan and zoom controls). Besides, by toggling the different options at the top right corner within the map, users are also able to switch displayable map types. For instance, in figure 8, the current map displayed is the Terrain map, one of the four map views available. The remaining three map views available are the following: the road map, the satellite map and the hybrid map. The satellite map exhibits the surface as an aerial view from a collection of pictures provided by the Google Maps service. Finally the hybrid map type displays the surface combining the road map and satellite views.

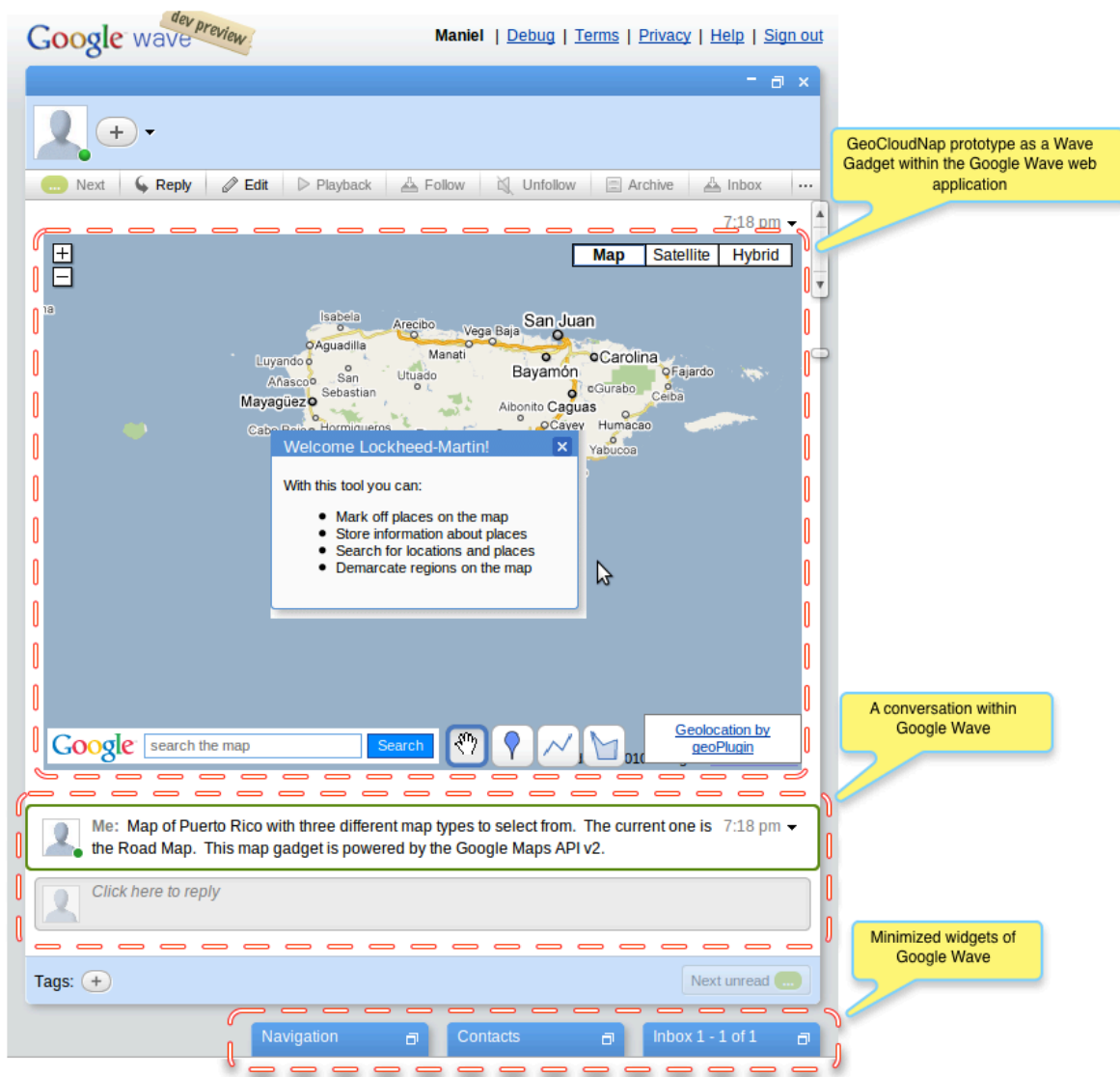
On the other hand, extra controls that interface with the customized Google Maps functionality are reachable outside of the map display. These include: the layer-toggling tree-view menu that is fixed next to the map display; and, the marker type drop-down menus that lie atop. The layer-toggling menu allows users hide and expose instantly a group of markers belonging to available layers. The marker type menus, helps users choose the right marker kind to spot the desired location and be able to associate it with its corresponding layer. The marker type menu allows users associate specific and pre-determined information (e.g., name of place, short description, etc.) to each place marked given a marker type. For, each marker type enables users to input proper data values to particular property fields having a diversity of meanings or semantics.

### 3.4.1 User Interface on Google Wave

Google Gadgets is a technology to embed third-party applications into the Google Wave platform so as to enable multiple users to consume multiple application features simultaneously [7]. With this, users can collaborate mutually by interacting with each other's work. In this case, the application to be deployed within Google Wave would be the GeoCloudNap notional application. For example, if a user marks-off a place on the map, another remote user could also manipulate that same mark-off through his local map view of the same place on his local machine. This way, users not only share and document their actions in progress, but can also see them live and uninterrupted. Figure 3.6 shows a screenshot of a browser displaying the GeoCloudNap prototype as a Google Wave Gadget.

To fulfill this, the JavaScript client code that enhances the Google Maps application needed to be incorporated into the Google Wave platform. The incorporation of the GeoCloudNap code requires to be packed into an installer so that users can embed them into waves. What follows is to adapt the Django based implementation of the server side code so as to be able to interface it with the GeoCloudNap Google Wave extension.

Even though a prototype was successfully put into the Google Wave platform, the initiative of its further development had to be abandoned. This was due to the public announcement of Google about the cancellation of the building of the Google Wave platform as a standalone product. Google affirmed that the internal protocols and some of the Google Wave features were to be integrated as part of other products in their portfolio [8].



**Figure 3.6: Screenshot of user interface of GeoCloudNap prototype within Wave**



### **3.5 Application Logic Tier Overview**

The Application Logic Tier (ALT) of GeoCloudNap serves the computing processing needs to the presentation tier besides different content types. Two are the content types served to the presentation tier: a part of the displayable content and all the data from task results performed through the user interface by users. From among all content displayable by the user interface, the application logic tier serves the client code (scripts) and web documents. The web browser hosted by the presentation tier depends on web documents to be able to show the embedded Google Maps display and all other web components of the GeoCloudNap user interface.

Parts of these web documents consist of client scripts written in JavaScript. These client scripts execute, through the web browser, specific functions that help obtain user input and show the customized map display of Google Maps. The remaining type of web documents consist of web forms that instruct the layouts the web browser needs to follow to display the map and menus within its view port. The content of these documents is limitedly static and most of it is dynamically generated in the application logic tier by the server-side scripts. These server-side scripts are called templates in the Django Web framework terminology and are written in the Python language as well.

All content served by the application logic tier is made available to the presentation tier, that is, published, by the Apache Web Server. Thus, to be able to publish all content after each user request, the Apache Web Server has to coordinate with the Python interpreter before emitting a response. This process occurs repeatedly, given that all Django templates embed code into the web documents to be able to dynamically generate all content sent to the

presentation tier. Thus, any application logic written in Python using the Django Web Framework is able to provide the dynamic content that the user interface requires at the Presentation Tier.

Furthermore, the application logic tier also hosts several Django-based modules that interface with the underlying database system at the Data Management Tier. These modules implement the Object-Relational Mapper libraries of the Django Web framework to ease the access to data hosted by the Data Management Tier. These modules comprise one of the most fundamental parts of the application logic tier, because these define the information that the Presentation Tier displays and that the Data Management Tier manages. Without these, interaction between the Presentation Tier and the Data Management Tier would be impossible, and therefore, no information management would happen.

### **3.6 Data Management Tier Overview**

The Data Management Tier hosts the service provided by the Database Management System (DBMS) [19]. The DBMS purpose is to control the creation and modification of data entities, with their respective attributes, of the GeoCloudNap Database. This way, the Data Management Tier enables the Application Logic Tier to consume all of the GeoCloudNap data through the underlying DBMS. The DBMS that stores all of the GeoCloudNap data is PostgreSQL. PostgreSQL is the most advanced open-source DBMS available in the public domain.

The data managed by GeoCloudNap includes information submitted by users that require of standard data types like real numbers, timestamps and character strings.

Nevertheless, support for spatial data types is also indispensable for locating and marking off places on the map. With spatial data types, GeoCloudNap is able to store world coordinate values into the Database in a standardized manner. However, given that PostgreSQL does not support spatial data types natively, the use of extension libraries is required as an enhancement.

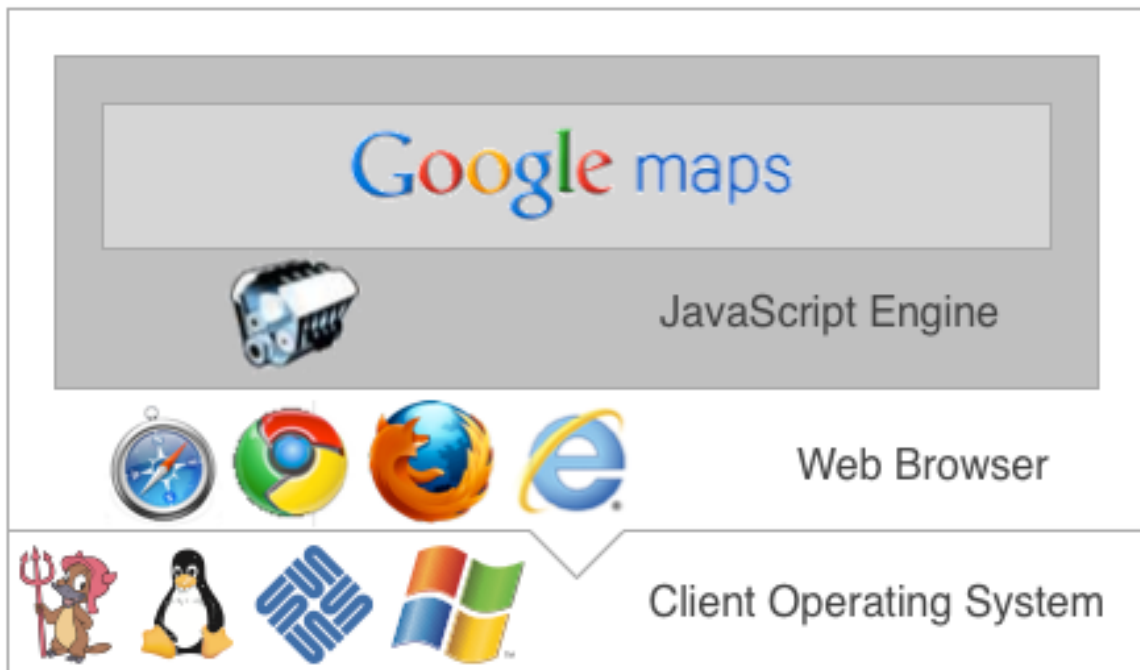
The support libraries that GeoCloudNap incorporates along PostgreSQL are provided by the PostGIS extensions [10]. PostGIS is a collection of open-source library extensions implemented exclusively for the PostgreSQL DBMS. PostGIS enables database designer and application developers to implement spatial data types as naturally as implementing PostgreSQL native data types. Besides providing spatial data types, PostGIS also features a set of enhancements to PostgreSQL that permit developers to query for spatial data values. These enhancements allow the Django-based application logic to directly query the DBMS for all of GeoCloudNap data in the Data Management Tier.

The application logic queries the DBMS using a set of libraries packaged by the Object-Relational Mapper (ORM) API provided as part of the Django Web framework distribution. Thus, the entire database schema is easily accessible from the Application Logic Tier. Furthermore, the database schema is built with the aid of the ORM API. Basically, the Django ORM provides the code libraries necessary to define all data models. Once the data models have been defined, all database tables are built by compiling the data models definitions using a script provided by the Django Web framework. The ORM API also provides the plumbing necessary to execute queries and transactions by using SQL directly from within the application logic itself.

## 4 GeoCloudNap IMPLEMENTATION

This chapter presents the implementation details for the different tiers used in our GeoCloudNap system.

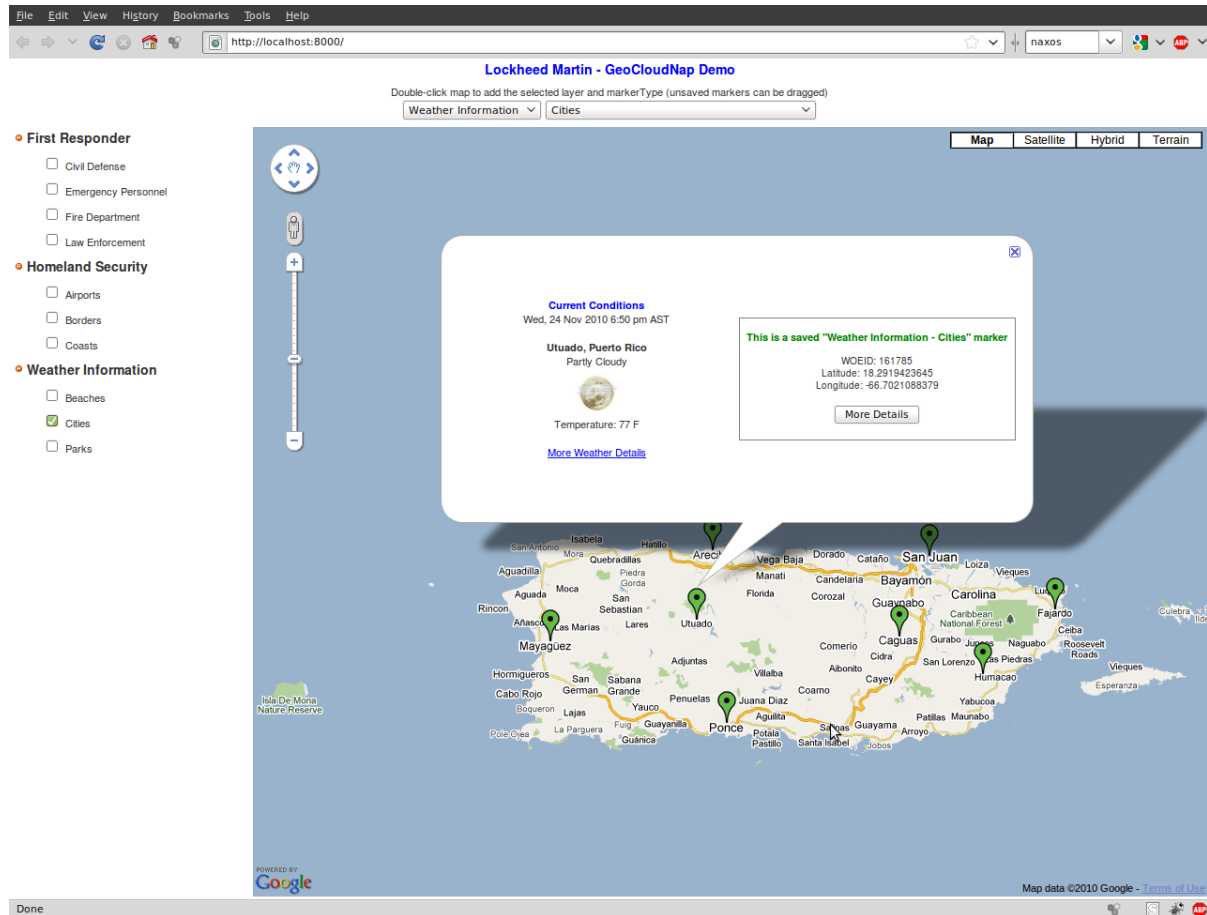
### 4.1 Presentation Tier Implementation



**Figure 4.1: Diagram of technology stack of presentation tier**

Within our context, clients can be any kind of computer or mobile device (e.g., Laptop, Smartphone, etc) providing basic user interface controls capable of displaying color graphics. However, clients are required to be equipped with the minimum hardware necessary to navigate the Internet through a thin client interface such as a web browser. Yet, these web browsers also require having the JavaScript language engine enabled. The JavaScript engine


would then provide the essential execution of the Google Maps and Google Wave APIs, service tools and user interfaces. Figure 4.1, illustrates the requirements of technologies that the presentation tier needs to have and their relationships.



**Figure 4.2: Screenshot of user interface of GeoCloudNap**

The user interface of the GeoCloudNap web application (Figure 4.2) consists of an embedded Google Maps display, with two menus altogether, one at its top and another to its left. The menu listing at the map top provides two drop down menus for selecting the marker category and sub-category to pinpoint with on the map display. This top menu list allows users to select the layer and category (i.e., marker type) a marker will belong to when

marking-off a place of interest. However, the purpose of the menu listing at the left of the map is for toggling the view of a selected marker group in a marker type basis. These marker groupings are distinguished as layers, which are configurable through the administration user interface. Layers avoid cluttering throughout the map display and ease users interaction by hiding irrelevant marker groups where excessive amounts are shown simultaneously.



```
1{% load dajaxice templatetags %}
2<html>
3<head>
4<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
5<link type="text/css" rel="stylesheet" href="/media/css/stylesheet.css" />
6
7<script type="text/javascript"
8  src="http://www.google.com/jsapi?key=ABQIAAAjNcB1fZNcpIZSSz0GFysSRR0e1C4EVk
9<script type="text/javascript" charset="utf-8">
10  google.load("jquery", "1.4.2");
11  google.load("maps", "3.x", {"other params" : "sensor=false"});
12</script>
13
14<!-- Added for treeview -->
15<!--<link rel="stylesheet" href="/geocloudnap/statics/jquery-treeview/demo/scre
16<!--<link rel="stylesheet" href="/geocloudnap/statics/jquery-treeview/jquery.tre
17<!--<script type="text/javascript" src="/geocloudnap/statics/jquery-treeview/jqu
18
19<link rel="stylesheet" type="text/css"
20  href="/media/imports/jquery-treeview/demo/screen.css" />
21<link rel="stylesheet" type="text/css"
22  href="/media/imports/jquery-treeview/jquery.treeview.css" />
23<script type="text/javascript"
24  src="/media/imports/jquery-treeview/jquery.treeview.js"></script>
25
```

**Figure 4.3: Code snippet to load Google Maps API**

The implementation of the user interface required the development of web documents and several client scripts is written in JavaScript. The main web document combined HTML content that included portions dynamically generated by templates written on the Django Web framework. The map display requires importing the Google Maps API library from the

JavaScript APIs repository of the Google website. Loading the maps API into the head section of the main web document imports the map display. Figure 4.3 highlights the code that loads the Google Maps API into the head section of the main web document.



```
155
156
157
158 function savedMarkerInfoWindow(aMarkerID, aLat, aLng,
159                               aWOEID, theLayerMarkerTypeID,
160                               theLayerName, theMarkerTypeName) {
161     tOIWWOEID = aWOEID;
162     Dajaxice.geocloudnap.getMarkerTypeInfoWindow('getMarkerTypeInfoWindowJS',
163     { 'aMarkerID': aMarkerID, 'aLat': aLat,
164       'aLng': aLng, 'aMarkerType': theMarkerTypeName });
165     return '' +
166       '<table class="infoWindow">' +
167       '<tr>' +
168       '<td width="50%">' +
169       '<p class="centered">' + $('#infoInInfoWindowInvi').html() + '</p>' +
170       '</td>' +
171       '<td width="50%">' +
172       '<div class="border">' +
173       '<p class="smallTitleGreen">This is a saved "' +
174       theLayerName + ' - ' + theMarkerTypeName + ' marker</p>' +
175       '<p class="xsmall">WOEID: ' + aWOEID + '<br />' +
176       'Latitude: ' + aLat + '<br />' +
177       'Longitude: ' + aLng + '</p>' +
178       '<p class="centered">' + $('#markerMonitoringButton').html() + '</p>' +
179       '</div>' +
180       '</td>' +
181       '</tr>' +
182       '</table>' +
183       '';
184 }
```

**Figure 4.4: Code snippet to draw information window of markers**

Figure 4.4 highlights the main code snippet that draws all information windows (i.e., callouts). The function that encloses the code snippet is executed after users click over any marker. It can be seen that the data fields of layer name and marker type name appear in the code, as well as, the coordinates and the WOEID of the selected location. Each time that markers are clicked by users, a request to the Application Logic Tier is made asking it to

return the data corresponding to the selected marker. Referring to Figure 4.2, all of the text and data fields requested for can be easily identified in the illustration. There, the selected marker has a WOEID number of 161785, were, at that moment, the weather conditions of its location appears to be partly cloudy with a temperature of 77 degrees Fahrenheit.

```

570
571 <tr>
572 <td colspan="2"><!-- <div id="addMarkerInfoWindowInvi" class="invi"> -->
573 <div id="addMarkerInfoWindowInvi">
574 <div id="addMarkerInfoWindow">
575 <table width="100%">
576 <tr>
577 <td colspan="2" class="small">
578 Double-click map to add the selected layer and markerType
579 (unsaved markers can be dragged) <br />
580
581 {% if layerMarkerType_list %}
582 <select id="layerSelect" class="smallNotCenter">
583 <option selected="selected" disabled="disabled" value="">
584 Please select a layer
585 </option>
586 {% for layer, markerType in layerMarkerType_list %}
587 <option value="{{ layer }}">{{ layer }}</option>
588 {% endfor %}
589 </select>
590
591 <select id="layerMarkerTypeSelect"
592 disabled="disabled" class="smallNotCenter">
593 <option selected="selected" disabled="disabled" value="">
594 Select a layer first
595 </option>
596 </select>
597 {% else %}
598 <p>No layer-makerType in system</p>
599 {% endif %}
600 </td>
601 </tr>
602 </table>
603 </div>
604 </div>
605

```

**Figure 4.5: Code snippet to dynamically generate menu for selecting marker types**

Referring to the external menus, there is the one at the top of the map display. This menu concerns the selection of the desired marker type that users need to choose before marking



off a location on the map. As said before, marker types enable users to associate specific information to a place pinpointed on the map. The particularity illustrated in Figure 4.5 is that not only static HTML content is needed to display the menu, but dynamically generated code too. Focusing in the reddish rectangle, various code statements can be identified having a black font and enclosed within brackets and percent signs. The Django Template system provides these code statements.

These code statements are written in the Python scripting language and are based on the Django Template system language. The use of these, allows the server-scripts in the Application Logic Tier to pre-process the generation of this web document before sending it to the Presentation Tier. This pre-processing happens at the Application Logic Tier, which permits for the dynamic generation of content within the web documents sent to the Presentation Tier. The Python interpreter through the libraries of Django performs the preprocessing, after the delegation of the Apache Web Server (discussed later on Section 4.2).

Many tasks can be performed helped by the Django Template system. On this case, a series of static HTML and template code statements are intercalated. Their purpose in here is to build the top menu, so that users can select the desired marker layer and types that the marker needs to belong to. The first code statement gets a list of the layers and marker types available to select from. Then, a layout written in static HTML code (writ by the “select” tag) builds the menu, and, enclosed within it the different options to select are add to the menu (writ by the “for” template statement).

Figure 16 depicts another section of the same web document that builds the main user interface. It can be seen that a similar approach was used to build the layer and marker types

toggling menu at the left of the map display. The section depicts the enclosing HTML “table” tag used to layout the table structure that contains the toggling menu. Again a series of multiple static and Django template-based statements intercalate throughout the reddish rectangle. First, a list of layers and marker types are retrieved from the data model. Then, for each layer and marker type available in the data model, an entry is added into the list check-boxed marker types (Refer to layer toggling menu shown in Figure 4.2).

```

609 <tr height="95%">
610   <td valign="top">
611     <table>
612       {% block menuBlock %}
613       <tr>
614         <ul id="example" class="treeview-famfamfam">
615           {% if layerMarkerType list %}
616             {% for layer, markerTypes in layerMarkerType list %}
617               <li>
618                 <span class="bold">{{ layer }}</span>
619                 <ul>
620                   {% for id, mT in markerTypes %}
621                     <li>
622                       <input type="checkbox" checked="checked"
623                         id="LayerMarker{{ id }}Checkbox">
624                       <label class="smallNotCenter"
625                         for="LayerMarker{{ id }}Checkbox">
626                         {{ mT }}
627                       </label>
628                     </li>
629                   {% endfor %}
630                 </ul>
631               </li>
632             {% endfor %}
633           {% else %}
634             <p>No layer-makerType in system</p>
635           {% endif %}
636         </ul>
637       </tr>
638     {% endblock %}
639   </table>
640 </td>
641 <td width="80%">
642   <div id="map_canvas" class="theMap" style="width: 100%; height: 100%"></div>
643 </td>

```

**Figure 16: Code snippet to dynamically generate menu to toggle display of markers**

### **4.1.1 User Interface on iPhone**

An implementation of a mobile user interface for GeoCloudNap was also made on the iOS platform, specifically for iPhone devices. Figure 4.6 shows a screenshot of the user interface. In this case, the map metaphor is used as in the case of the web application. The user is presented with a map showing different markers that indicate some information of interest. The user can click on a marker (“pin”) to see detailed information. In the current implementation, users can perform the following functions with the iPhone application:

- View or edit the markers available in the system.
- Add a new marker at the current location of the user. The internal GPS in the iPhone and the Core Location services are used to determine the user location.
- Add photos for events at a given marker. For this purpose, we use the internal camera contained in the iPhone.

The application uses the HTTPRequest libraries provided the technology layers from iOS to pose RESTful [6] requests to the Web application in the Application Logic Tier (ALT). All responses from the web application come encoded in JSON.

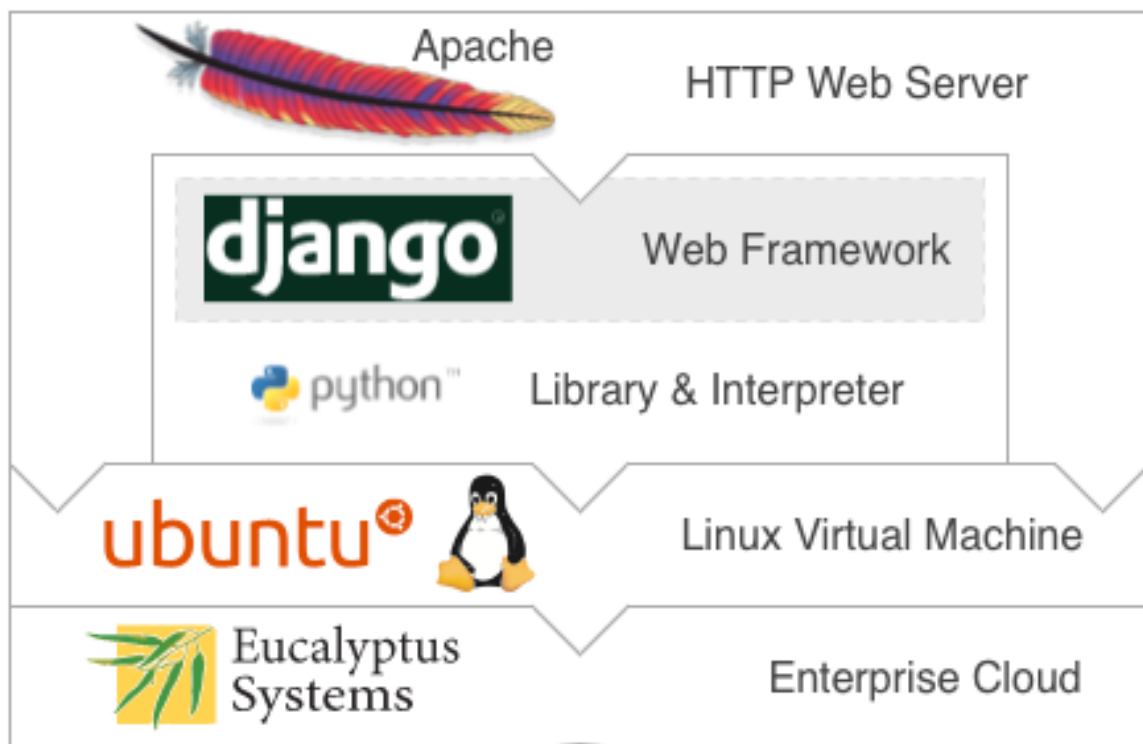
The iPhone application is very useful in the case of first responders (e.g., Policemen, Firemen, etc.) because it allows them to send updated information to the command center about a situation currently taking place. The command center can in turn send detailed instructions on how to proceed to the first responder.



**Figure 4.6:** Screenshot of mobile user interface for iPhone

## 4.2 Application Logic Tier Implementation

The Application Logic Tier (ALT) implements a diversity of technologies that mutually collaborate to empower the supply of computing processing power to exchange data and to host all content. Figure 4.7 conceptually illustrates the layered stack of the technologies employed by the ALT.



**Figure 4.7: Diagram of technology stack of application logic tier**

The ALT hosts and serves the web documents written in HTML, as well as the server-side scripts written with the Django Web framework libraries. Web documents are comprised of:

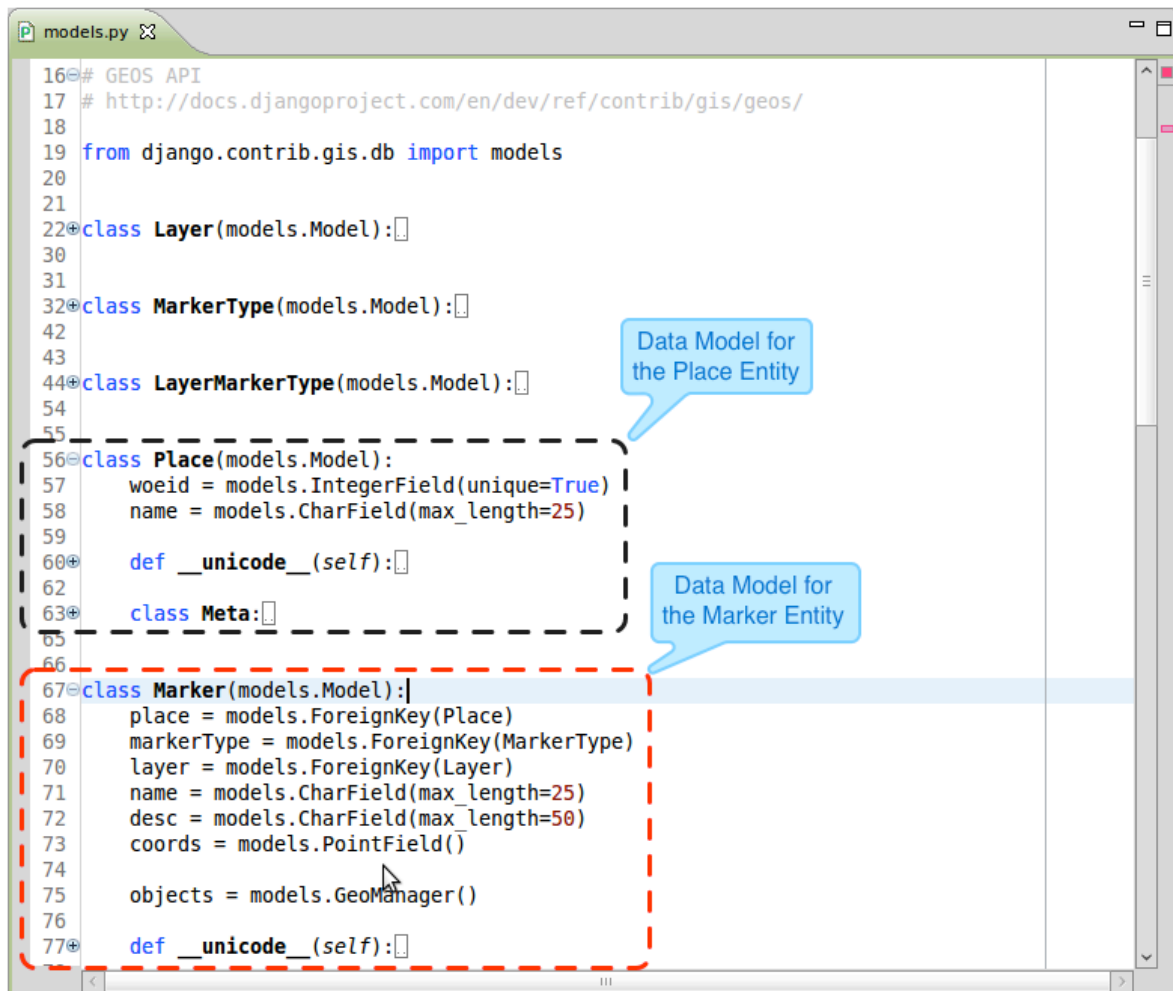
- All style sheet documents written in CSS that formats the appearance of the user interface in the PT.

- All images used to depict elements in the PT by its user interface.
- All documents written in static HTML code, which layouts the appearance of the user interface in the PT, along with the dynamic Django Templates-based code. (See section 4.1)
- All JavaScript client scripts to be delivered to the PT to handle all user input among other operations. (See section 4.1)

The Apache Web Server (AWS) is the system process that serves all static web documents to the PT by interfacing directly with its client web browser. But, from all the content that serves, the Django server-side scripts dynamically generate the most important of it. Thus, to be able to build the content as needed by the PT, the AWS requires interfacing also with the Python Interpreter. For this, the AWS depends on the installation of the “mod\_wsgi” module, which is used as an extension of the Web Server so as to interface with the Python Interpreter. Thus, permitting an anticipated execution by the Python Interpreter to generate the dynamic content before delivering the final web document to the client web browser in the PT.

Programmatically, server-side scripts can be seen as the Controller role of the **Model-View-Controller (MVC)** design pattern (Introduced at Section 3.3). Therefore, given its dependency with the Model, the Django server-side scripts need to interface with the data besides with the client. For this, the Django Object-Relational Mapper (ORM) API enables the interaction between the data model hosted by the Data Management Tier (DMT) and the application logic. This way, the ALT can access the data from the DMT so as to pre-process

the web documents along with their Django templates to be able to deliver to the PT all of the task results requested by users.



**Figure 4.8: Code snippet of object data models for Place and Marker entities**

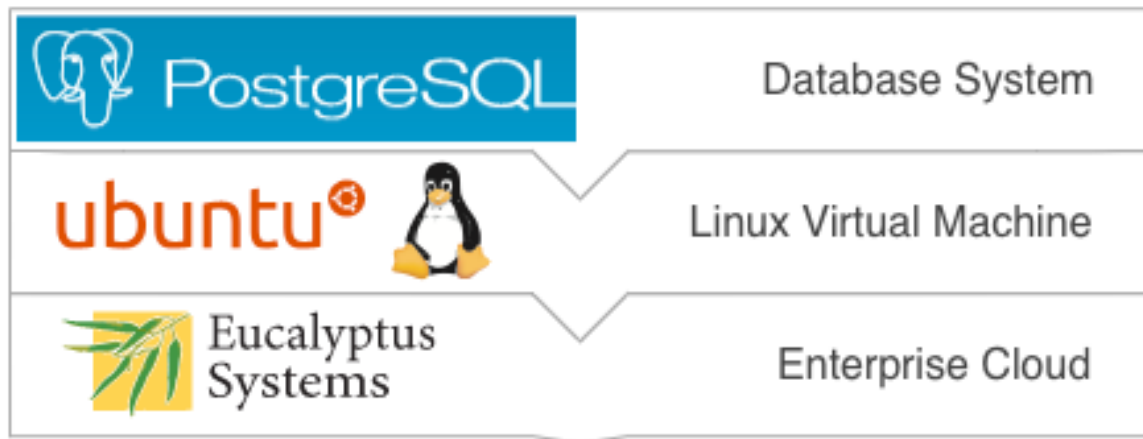
To use the ORM API, an object data model has to be defined for each entity that the application logic needs to manipulate. Each data model definition consists of designating a name to the entity and its fields, and to determine the appropriate data type of each. For example, Figure 4.8 shows how two model entities are defined and coded using the Django ORM API library. The example displays two model entities, the Place and the Marker

entities. It can be seen that the Place marker has two fields, the WOEID which is an integer type, and the name of the place, which is a character string type. As for the Marker data model, it can be seen that it has a name, a short description and the coordinates location of the marker. It can also be seen that three remaining fields that are of Foreign Key type, refer to other data model entities. These are references that are used to help define associations between the Marker data model and the other three data model entities. At this point these are considered objects that recur to the ORM API to represent the actual data entities managed by the DMT. It is the ORM API lower level libraries that abstract the particularities of communicating with the DMT to persist the data represented onto these data model objects.



### 4.3 Data Management Tier Implementation

The purpose of the Data Management Tier (DMT) is to store and provide all data handling functionalities. The storage and retrieval of data is performed by a software system called Database Management System (DBMS) [19]. It implements almost exclusively the DBMS as its main process. A DBMS provides the access mechanisms to data as efficiently as possible without requiring developers to involve with complex low-level operations. Database Management Systems carry on the creation, maintenance and aids the use of collections of data records. Figure 4.9 depicts the layered stack of the technologies implemented by the GeoCloudNap DMT.



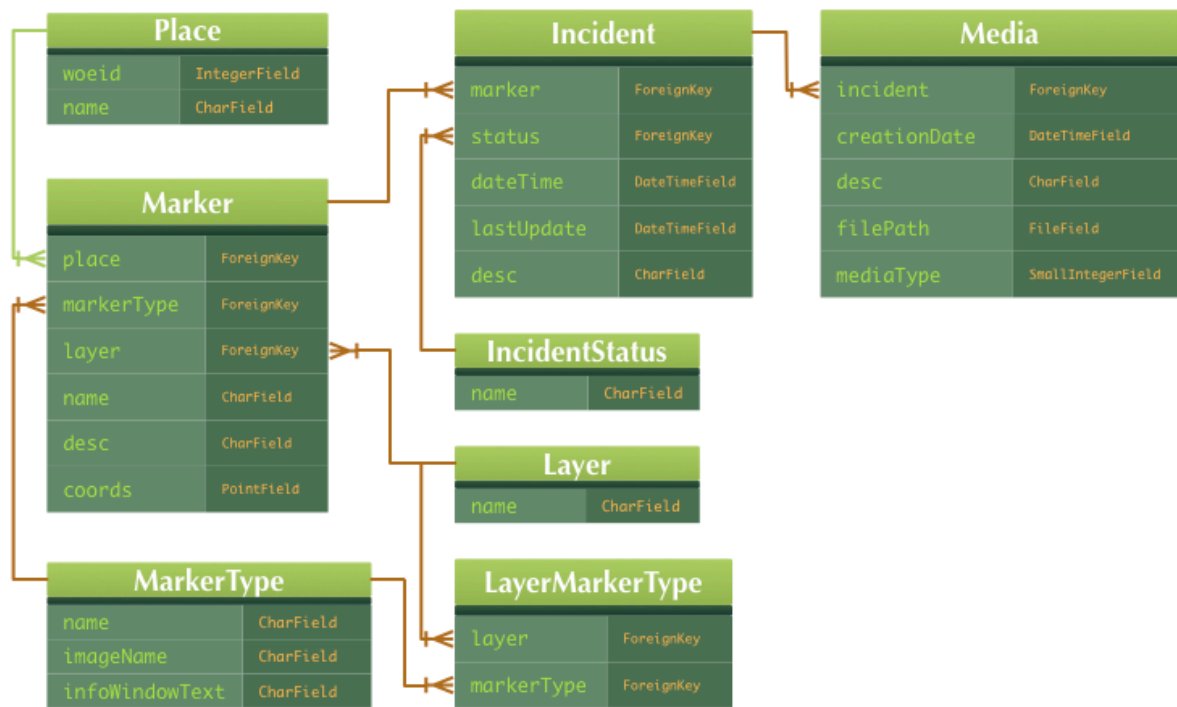
**Figure 4.9: Diagram of technology stack of data management tier**

The DBMS that GeoCloudNap uses is an open-source software package called PostgreSQL. PostgreSQL is one of the most advanced open-source relational Database Management Systems available to the public. A relational DBMS is one that implements the Relational Model for structuring and organizing the data in a database. Basically, the model establishes that, all entities can be structured as tables, where columns represent an entity

attributes. The way to interconnect the DBMS with the rest of the system is through the use of a driver. In this case, the Django Web framework provides “psycopg”, the appropriate driver to interconnect to PostgreSQL databases.

By using “psycopg”, the application logic implemented with Django is able to communicate with the DBMS and thus post and retrieve all data to be managed. The “psycopg” driver, in turn, enables the ORM API perform the required transactions and persist all incoming data from the external web services or the user interface. All data model objects representing entities then get translated into the GeoCloudNap relational schema hosted and managed by the DBMS. The relational schema of the GeoCloudNap database can be seen in Figure 4.10, which illustrates all the entities required to store places with their respective markers. It also shows how markers are associated to marker types and to incidents. Incident is the relation used to store specific information for some marker types.

The GeoCloudNap relational schema defines that for a place located in the map, there can only be a single marker related, which would make it a one-to-one relationship. As for the Marker relation, it is depicted that it can be associated to a single Marker Type, but a Marker Type can be related to many Marker, which would make it a one-to-many relationship. Likewise, the relationship between Marker and Incident is a one-to-many relationship, for a Marker can be related to many incidents, but an incident can be related to a single Marker. These are relationships that are defined by modeling the data as objects using the Django ORM API.



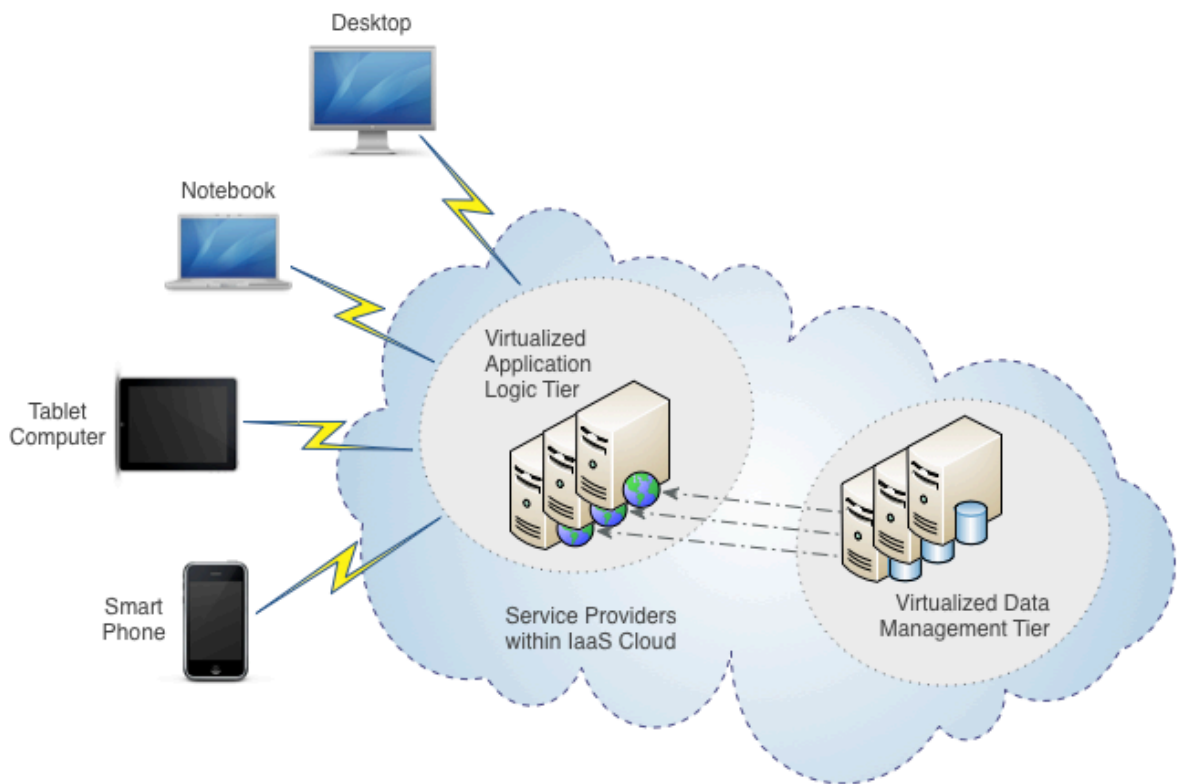
**Figure 4.10: Diagram of ORM data model of the GeoCloudNap relational schema**

To build the database, the Django Web framework provides a set of shell scripts that can be used to build the SQL statements from the ORM data model of entities. Basically, the shell scripts interpret the code defining the data model of entities and generate the set of SQL statements to be executed by the DBMS. The execution of the SQL statements is performed through the bridge established from the Application Logic Tier to the Data Management Tier by the “psycopg” driver. These statements of course, depend on all data types defined by PostgreSQL and PostGIS.

PostgreSQL supports many types of data, which are also already supported by the Django ORM API. However, the spatial data types that Google Maps require are not. PostGIS provide these spatial data types, an extension library for the PostgreSQL DBMS. For example, one of the spatial data types that GeoCloudNap depends on is the two-

dimensional Point. The Point field is used to implement and store the coordinates field of the Marker field to pinpoint each marker on the map.

## 4.4 Service Providers as Virtualized Tiers



**Figure 4.11: Diagram of virtualization of service provider tiers by IaaS cloud**

Within the system, the application logic tier and the data management tier are services hosted by an IaaS cloud. These two tiers are network-interconnected but deployed within a single cloud infrastructure (described later) throughout a set of two separate virtual machines. Thus, the application logic tier resides on one virtual machine and the data management tier on

another (i.e., on two loose virtual machines that act as apart servers). Figure 4.11 illustrates the deployment architecture of the virtualization of service provider tiers by an IaaS cloud.

A virtual machine (VM) [21] is a software-implemented machine that emulates the instructions execution (not programs) of a physical computer. The VM simulates an executing machine over a software infrastructure (i.e., layer) that abstracts the physical host machine architecture from all their particularities. This software layer is as fundamental to a cloud infrastructure as well as all the necessary hardware resources (e.g., networks, servers, storage, etc) for deploying a public cloud. This deployment model establishes that the cloud infrastructure is managed and owned by an organization selling cloud services, which makes them publicly accessible. A public cloud [14] refers to one of the four deployment models defined by the Cloud Computing paradigm (introduced in Background section). Eucalyptus [5] is an open source software infrastructure for implementing IaaS clouds on premise (i.e., private clouds).

The following two are the most relevant features of machine virtualization on public clouds:

- **Multiple instance deployment** - virtualization permits to run multiple virtual machines on a single physical machine so as to share its hardware resources concurrently.
- **Instance migration** – virtualization supports virtual machines migration to existing cloud infrastructures independently of underlying architectures of target physical host systems.

From these two features, the following benefits can be identified:

- **On-demand provisioning** - virtual machine instances and resources can be deployed quickly on demand and almost effortless.
- **Service diversification** – virtualization and deployment of multiple instances is ideal for providing a variety of isolated services for particular purposes.

In our case, virtual machines are ideal to provide services for specific purposes such as: web application hosting and processing, and, data management services. Particularly, the Apache Web Server provides the web application services from the application logic tier, while the PostgreSQL server the data management services from the data management tier. Thus, separate virtual machines host each particular tier within the same cloud infrastructure. This way, the data management and weather information services can be served from the application services deployed on the public cloud infrastructure as well as from other third-party web-services.

## 5 SUMMARY AND CONCLUSIONS

This project reported on a notional application built on top of an open source cloud infrastructure to manage weather information. The open source cloud infrastructure was implemented using the Eucalyptus Platform solution. The notional application provides users with information about current weather conditions at various geographic locations. GeoCloudNap has the functionality of storing the places of interest that have been marked off and their information about weather conditions as well as other related inputs from users.

The web view of the application was implemented with the Google Maps API. Users have the ability to click on a location on the map and access its World coordinates, and its weather conditions. The Where On Earth ID (WOEID) was used to obtain the weather conditions and forecast for a given location from Yahoo!'s Weather service. Users can save these locations into a local database and later retrieve them to compare data values. Stored locations are represented as customized markers in the map whose all related data is persisted by a Relational Database Management System (RDBMS).

PostgreSQL was the RDBMS used to store all data obtained from user input and the external weather services. The solution required the support of the PostGIS extension library to manage and store world coordinates location in the PostgreSQL. Storage of these and all data was achieved through the use of the Django Object-Relational (ORM) API. The processing logic leveraged heavily from the Django ORM so as to retrieve all data from the database to be able to deliver it to the user. The processing logic was developed with the Django Web framework, which was hosted by the open source Apache Web Server.

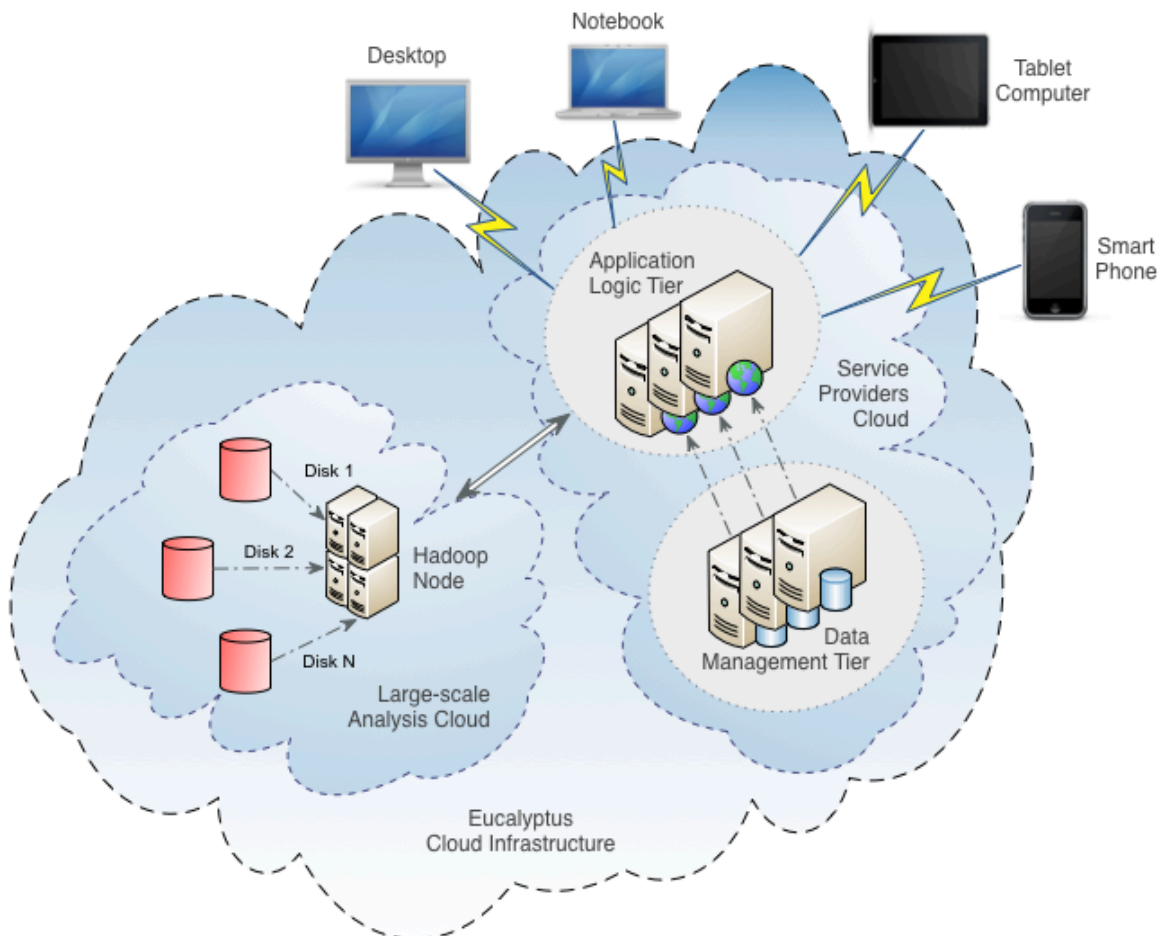
The GeoCloudNap notional web application followed the three-tier architecture and the Model-View-Controller through the use of the Django Web framework. All architectural roles were virtualized on top of the Eucalyptus Cloud Platform except the web browser client.

The project required for an arduous and continuous R&D effort so as to solve technical challenges that appeared during the process. The experience improved technical skills related to web development and cloud environment platforms. The success of the project development can be attributed to the direct exposition to the latest trends in technology. Leveraging from geospatial databases were a definitive advantage in terms of building effectively and in timely manner the web application. Moreover, cloud computing technologies worked well and provided the ability to add new instances on demand. This enabled use to add functionality to the system incrementally. We always had a working system, for demo and illustration purposes. We also had a replica of this system, which we used to try new features. Both of these versions were run on the same cloud platform. This validates the cloud as a platform for agile software development, and for rapid prototype development.



## 5.1 Future Work

Future plans contemplate adding the capability of a monitoring feature to the system. The feature will enable users to register points of interest whose weather data and forecasts are to be periodically retrieved from external data sources. Consequently, huge quantities of data can be quickly generated. Moreover, all data retrieved and stored, will be used to perform a diversity of large-scale analysis [18]. Thus, there will be imminent need for processing power and storage in a large-scale environment on-demand.



**Figure 5.1: Diagram of multi-cloud application architecture**

For this, it is planned to implement Hadoop to run data analysis jobs on demand and Hive as the data warehouse to provide storage for large data sets. Therefore, future plans also involve the deploying of an overall multi-cloud infrastructure (See Figure 5.1). The multi-cloud infrastructure would then consist of three interconnected clouds. Eucalyptus would provide the infrastructure with Linux-based virtual hosts to run all servers. Then, on top of this cloud platform, Hadoop would be used to build a cloud for large-scale analysis using the MapReduce programming model [1] [4]. Finally, Django-based web application servers could employ a separate cloud for hosting the web-based applications.

## REFERENCES

- [1] Abouzeid, A. (2009). HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. *Proceedings of VLDB*. Lyon, France.
- [2] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-Oriented Software Architecture: A System of Patterns* (Vol. 1). Wiley.
- [3] Cerbelaud, D., Garg, S., & Huylebroeck, J. (2009). Opening The Clouds: Qualitative Overview of the State-of-the-art Open Source VM-based Cloud Management Platforms. *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*. Urbana, Illinois.
- [4] Dean, J., & Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. *Proc. of 2004-OSDI*, (pp. 137-150). San Francisco, CA, USA.
- [5] Eucalyptus Systems. (2009 йил Aug). *Whitepapers, Eucalyptus*. Retrieved 2010 Feb from Eucalyptus, Your Environment: <http://www.eucalyptus.com/>
- [6] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.
- [7] Google. (2009). *Highlights from Google I/O 2009*. Retrieved 2010 Feb from Google I/O: <http://www.google.com/events/io/2009/>
- [8] Hölzle, U. (2010 04-Aug). *Official Google Blog*. Retrieved 2010 йил 08-Aug from Update on Google Wave: <http://googleblog.blogspot.com/2010/08/update-on-google-wave.html>
- [9] Hayes, B. (2008, July). Cloud Computing. *Communications of the ACM*, 51 (7).
- [10] Holl, S., & Plum, H. (2009). PostGIS. *GeoInformatics*, 3, 34-36.
- [11] Katz, R. H. (2009 Feb). Tech titans building boom. *IEEE Spectrum*, 4 (2), pp. 40-54.
- [12] Lenk, A. (2009). What's inside the Cloud? Ann Architectural Map of the Cloud Landscape. *Proc. ICSE Workshop on Software Engineering Challenges of Cloud Computing*, (pp. 23-31). Vancouver, Canada.
- [13] Mark, D., & LaMarche, J. (2009). *Beginning iPhone 3 Development: Exploring the iPhone SDK*. New York, USA: Apress.

- [14] Mell, P., & Grance, T. (2009 Oct). *NIST Working Definition of Cloud Computing*. Retrieved 2010 йил Feb from Cloud Computing Interoperability Forum: <http://groups.google.com/group/cloudforum/web/nist-working-definition-of-cloud-computing>
- [15] Moreno-Vozmediano, R., Montero, R. S., & Llorente, I. M. (2009). Elastic Management of Cluster-based Services in the Cloud. *Proceedings 1st Workshop on Automated Control for Datacenters and Clouds*, (pp. 19-24). Barcelona, Spain.
- [16] Nurmi, D. (2009). The Eucalyptus Open-source Cloud-computing System. *Proc 9th IEEE/ACM International Symposium on Cluster Computing and the GRID*, (pp. 124-131). Shanghai, China.
- [17] Patterson, D. A. (2008). Technical Perspective: The Data Center is the Computer. *Communications of the ACM*, 1 (51), p. 105.
- [18] Pavlo, A. (2009). A comparison of approaches to large-scale data analysis. *Proceedings 2009 SIGMOD*. Providence, RI, USA.
- [19] Ramakrishnan, R., & Gehrke, J. (2003). *Database Management Systems*. New York: McGraw-Hill.
- [20] Rodriguez-Martinez, M., Seguel, J., & Greer, M. (2010). Open Source Cloud Computing Tools: A Case Study with a Weather Application. *3rd International Conference on Cloud Computing (CLOUD)* (pp. 443-449). Miami, FL: IEEE.
- [21] Smith, J. E., & Nair, R. (2005). The Virtual Machine Architecture. *Computer*, pp. 32-38.
- [22] YDN GeoPlanet. (2010, Feb). *Key Concepts - YDN*. Retrieved from Yahoo GeoPlanet Guide: <http://developer.yahoo.com/geo/geoplanet/guide/concepts.html#woeids>
- [23] YDN Weather. (2010, Feb). *Yahoo! Weather RSS Feed*. Retrieved from Yahoo! Weather - YDN: <http://developer.yahoo.com/weather/>

AMDG ET BMV