Fall 12-15-2012

Application of Web Mashup Technology to Oyster Information Services

Christian Chuindja Ngniah
University of New Orleans, cchuindj@uno.edu

Application of Web Mashup Technology to Oyster Information Services

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science
Database Systems and Distributed Applications

by

Christian Chuindja Ngniah

B.S. University of New Orleans, 2010

December, 2012

Acknowledgment

Foremost, I would like to express my sincere gratitude to my advisor Professor Shengru Tu for the continuous support of my Masters Study and research, for his patience, motivation, and enthusiasm.

I would also like to thank my thesis committee: Dr. Mahdi Abdelguerfi and Dr. Thomas Soniat for their encouragement and guidance, and Ms. Susan Colley Theodosiou.

I thank my fellow lab mates working with me on the Oyster Sentinel project: Nathan Cooper and Janak Dahal for the stimulating discussions and for all the fun we have had in the last eighteen months. Also I thank my friends: Siegfried Kamga, Herbert and Arlene Twase, and Claire McDaniels for their support.

I would like to specially thank Mr. Stephen Sontiag at Applied Sciences Associates for his assistance.

Last but not the least, I would like to thank my family: my parents Michel and Ruth Ngniah, for giving birth to me at the first place and supporting me spiritually throughout my life. Much consideration also goes to my siblings: Jean-Claude, Bertrand, Guy, Melanie, and Valery for their advices and my dear niece Myriam.

Table of Contents

List of Figures

Abstract

Web mashup is a lightweight technology used to integrate data from remote sources without direct access to their databases. As a data consumer, a Web mashup application creates new contents by retrieving data through the Web application programming interface (API) provided by the external sources. As a data provider, the service program publishes its Web API and implements the specified functions.

In the project reported by this thesis, we have implemented two Web mashup applications to enhance the Web site *oystersentinel.org*: the *Perkinsus marinus* model and the Oil Spill model. Each model overlay geospatial data from a local database on top of a coastal map from Google Maps. In addition, we have designed a Web-based data publishing service. In this experimental system, we illustrated a successful Web mashup interface that allows outside developers to access the data about the local oyster stock assessment.

KEYWORDS: Web mashup, Oysters, Web application, Web service, Google Maps, OceansMap, *Perkinsus Marinus*, Oil Spill, Fisheries, Databases, Application Programming Interface

Chapter 1: **Introduction**

In 2009, Florida, Alabama, Mississippi, Louisiana, and Texas produced 61% of oysters for our entire nation's annual consumption [LWF, 2009]. Oyster Sentinel (*oystersentinel.org*) is a Web-based community monitoring the environmental health of estuaries along the Gulf of Mexico. The main purpose of this Web site is to evaluate the freshwater resources needed to sustain oysters, control parasites and predators, and support other estuarine-dependent organisms. Modeling tools are provided to assess the condition of oysters in stations along the Gulf coast. The following features are included among the collection of modeling tools: the Habitat and Restoration Suitability Indices that assess the impact of salinity alterations on oyster habitat and on reef restoration and the Oyster Sustainability Model that simulates effects of fishing on oyster habitat.

In the efforts to enhance *oystersentinel.org*, we have designed and implemented two models: the *Perkinsus marinus* model that assesses the extent of oyster infection by the parasite *Perkinsus marinus* and the Oil Spill model that assesses the impact of oil spills on oyster habitat. In building these two capabilities, we have applied Web Mashup techniques and used large portions of data that are provided by external resources such as Google Maps, the Louisiana Oil Spill Coordinator's Office (LOSCO), the Environmental Response Management Application (ERMA), and the National Oceanic and Atmospheric Administration (NOAA)**.** The Web Mashup technology gives developers tools to consume external resources, as well as tools to create interfaces to internal data. To investigate the latter capability, we have constructed a Web interface that will allow the programs developed by third party developers to use *oystersentinel.org* as a data source.

Chapter 2: **Background**

2.1 Web application

A Web application is an application that is accessed over a network and that is essentially a user-to-machine communication. It is stored and used from a server and users can access it via a Web site. Web applications on *oystersentinel.org* are implemented using PHP and run on an Apache server.

2.2 Web services

Contrary to a Web application, in which humans interact with a remote program across the Internet, a Web service is a machine-to-machine communication. The World Wide Web Consortium (W3C) describes a Web service as a software system designed to support interoperable machine-to-machine interaction over a network. It allows two remote machines to exchange XML formatted messages via HTTP.

2.3 Web mashup

A Web mashup is a hybrid Web application that fetches data from two or more sources and displays it with an entire new intent. IBM describes Web mashups as a genre of interactive Web applications that draw upon content retrieved from external data sources to create new and innovative services [Merrill, 2006].

Examples of Web mashups include:

- **WeatherBonk.com**, which combines *Google Maps*, *WeatherBug*, and *National Weather Service* to provide a weather forecaster and traffic reporter.
- **HousingMaps.com**, which combines *Google Maps* and housing posts on *Craigslist* to provide listing of houses for sale or for rent on a detailed map.

2.4 Drupal

Drupal is open source content management system (CMS). It is Web-based and requires an Apache or Microsoft IIS server, PHP (version 5.2 or higher), and a MySQL or PostgreSQL database engine to deploy a Web site [Drupal, 2012]. It is mainly used to allow non-Web developers to manage Web sites. In its core, Drupal is backed by a database where content, metadata, and configuration details are stored. When a request for a Web page is made, Drupal fetches the content from the database and builds the page using a template. Figure 1 illustrates the main components of a Drupal installation and the interactions between a Web page request and a response.

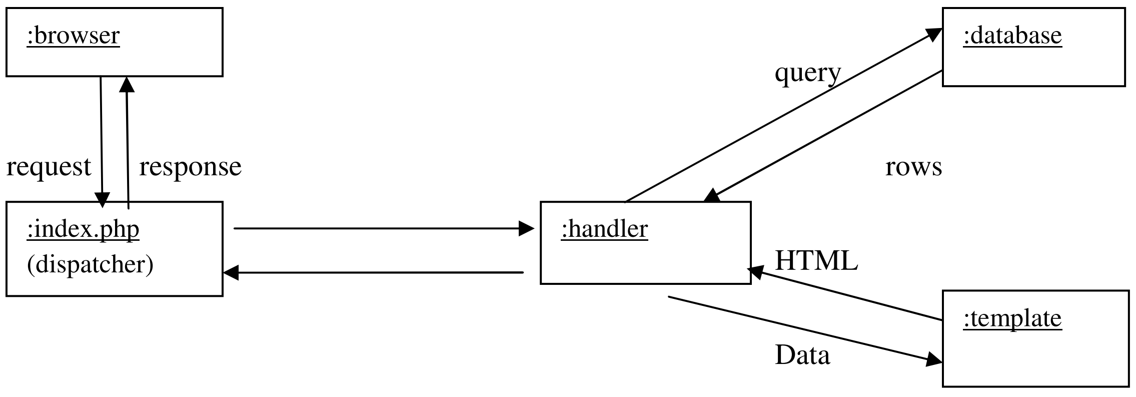


Figure 1 - Drupal Communication Diagram

In a CMS, users' management, user interface enhancement, and security are elements that improve the quality of a Web site. In Drupal, these capabilities are encapsulated in modules. A module is an extension or plug-in that adds functionality to a Web site. A collection of modules provides additional features to a Drupal installation.

From an end-user's perspective, Drupal provides an intuitive interface to search for modules and install them in one click. From a developer's perspective, Drupal allows programmers to create modules in four main steps: create the module folder (path: sites/all/modules), create the module file (.module extension), create a <modulename>.info file to tell Drupal about the module, and implement hooks, which are PHP functions to facilitate the integration of the module into the actions of Drupal core. A module is automatically bound to Drupal's database. Many built-in functions are available to configure the module, validate data, or query the database.

2.5 Google Maps

Google Maps is a Web service from Google that allows Web developers to easily add maps functionality to a Web site [Google, 2012]. The service can be consumed through its Maps JavaScript interface version 3.9, which is the latest version released. With its rich collection of classes to manipulate geographic data, Google Maps is a common data source for Web mashups. The models on *oystersentinel.org* primarily use classes from the main library:

- google.maps.Map to display a map on the Web site.

- google.maps.Marker to place stations on the map (*Perkinsus marinus* & Oil Spill models).
- google.maps.InfoWindow to display additional information about a station.
- google.maps.Polygon to display oil spills on the map (Oil Spill model).

Google Maps also provides libraries to compute distances and draw shapes on a map:

- Geometry library: google.maps.geometry.spherical to compute geodesic distances (Oil Spill model), google.maps.geometry.poly to check if a polygon contains a marker (Oil Spill model).
- Drawing library: google.maps.drawing.DrawingManager, google.maps.Circle & google.maps.Rectangle to allow users to draw shapes on a map (Fisheries Web-based service).
- Visualization library: google.maps.visualization.HeatmapLayer to show weighted markers on a map (Fisheries Web-based service).

2.6 OceansMap

Developed by RPS - Applied Sciences Associates, OceansMap is an online interactive Web-based oil spill simulation and spill response system, which consists of a GIS-based graphical user interface, a spatial database and Web services from MapSERVER and the Environmental Data Server (EDS) [ASA, 2012]. This service is primarily used to predict the path of an oil spill. It can also be used to improve the response to a disaster and minimize the damages. Only accessible through the Web site *oilmapweb.com*, OceansMap has an intuitive interface for users to enter data and run the prediction model. The Web mapping service presents the output as a series of projected snapshots of the oil spill. It provides tools to predict and analyze an oil spill and is a key component of the Oil Spill model.

2.7 JQuery

JQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid Web development [The jQuery Foundation, 2012]. It is cross-browser, meaning that it abstracts the JavaScript interpreters found in the main browsers. Mozilla, Google, Opera, Internet Explorer and Safari have a different JavaScript interpreter on their respective browser and using JQuery in our project helps create reliable scripts that will run smoothly on any browser. The Web site *oystersentinel.org* mainly uses handler functions because it allows programmers to implement specific behaviors when an action is triggered on a Web page. The main handlers used are:

- jQuery(<button>).click(function( ){ }) when a user clicks on a button
- jQuery(<dropdown>).change(function( ){ }) when a user selects an option in a dropdown list
- jQuery(<button>).submit(function( ){ }) when a user submits a form
- jQuery.ajax(function( ) { }) when making Ajax calls. An Ajax call is an asynchronous HTTP request made to a server.

Chapter 3: **Design**

Designing a model involves understanding the components interacting in the system and the data passed and returned by each of them. The design also involves the structure of the platform, the database schema, and external resources.

3.1 System Structure

3.1.1 Web applications in Content Management System

The Web site *oystersentinel.org* is a platform to inform marine biologists, fishermen, or any concerned parties in the states of the Gulf of Mexico about the oysters in the area. In addition to periodically updating the content of Web pages, the Web site manager also needs to manage news and alerts. However, scientific data and information are often managed by a scientist rather than a software engineer. In these cases, using a content management system (CMS) provides a solution. A CMS is a system used to manage the content of the Web site *oystersentinel.org*. It allows the content manager to create, edit, and remove content from a Web site without needing a technician such as a Web master. The chosen CMS is Drupal. It provides management tools and an intuitive user interface for content managers.

The Web applications implemented in *oystersentinel.org* need internal data to be stored in a more secure and performing database, and require external data sources to be integrated. There are at least two methods to meet this need. One method is to create a Drupal module that connects to a database other than Drupal's content database. It would involve setting up an extra server to provide that connection. Figure 2 is the communication diagram of a Drupal installation using an extra server to connect to a second database.

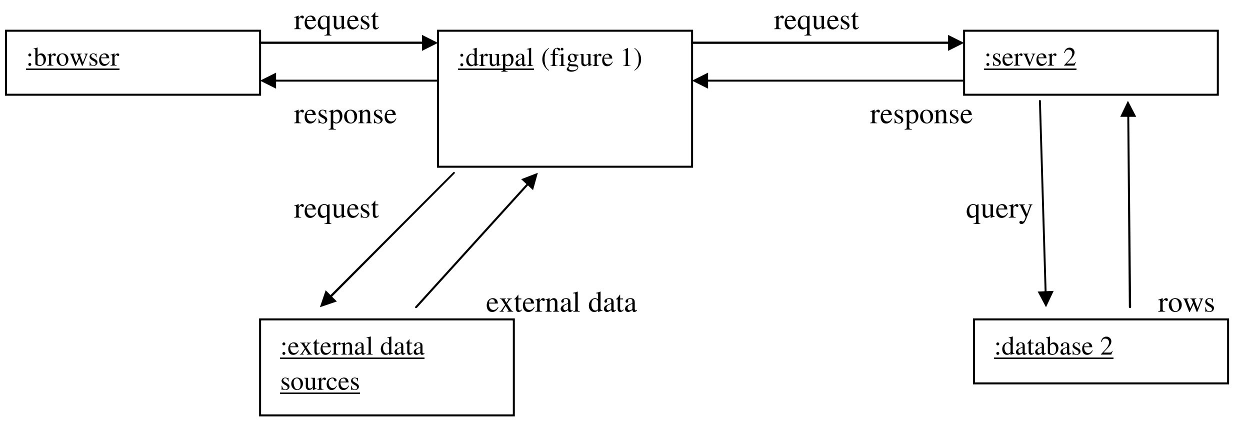


Figure 2 - Web Applications in Drupal using an extra server and database

Another method is to use a controller following the MVC design pattern. When a user sends a request, Drupal processes it (Figure 1) and creates the HTML page with a template. We created a bridge between Drupal and our MVC-designed application by including the controller when our modeling tool is requested (see code snippet below). When included, the controller builds the view and appends it to Drupal's template. This is a more flexible design approach that can facilitate the interaction between the second database and the applications on *oystersentinel.org*.

```
<?php
   if(strpos($_SERVER['REQUEST_URI'], '/?q=areas') !== false){
      include 'mvc/controllers/areas_controller.php';
   }elseif(strpos($_SERVER['REQUEST_URI'], '/?q=map_stations') !==
false){
      include 'mvc/controllers/stations_controller.php';
   } elseif(strpos($_SERVER['REQUEST_URI'], '/?q=stations_data') !==
false){
      include 'mvc/controllers/data_controller.php';
   }
?>
```

3.1.2 A Design following MVC

MVC stands for Model – View – Controller and it is a design pattern that aims to separate the data, its manipulation, and its presentation. The model is in charge of saving and fetching data. The controller manipulates the data fetched by the model and the view presents them to the user. Regardless of the type of data storage and the programming language, MVC patterns promote code reusability and isolation of business logic from user interface.

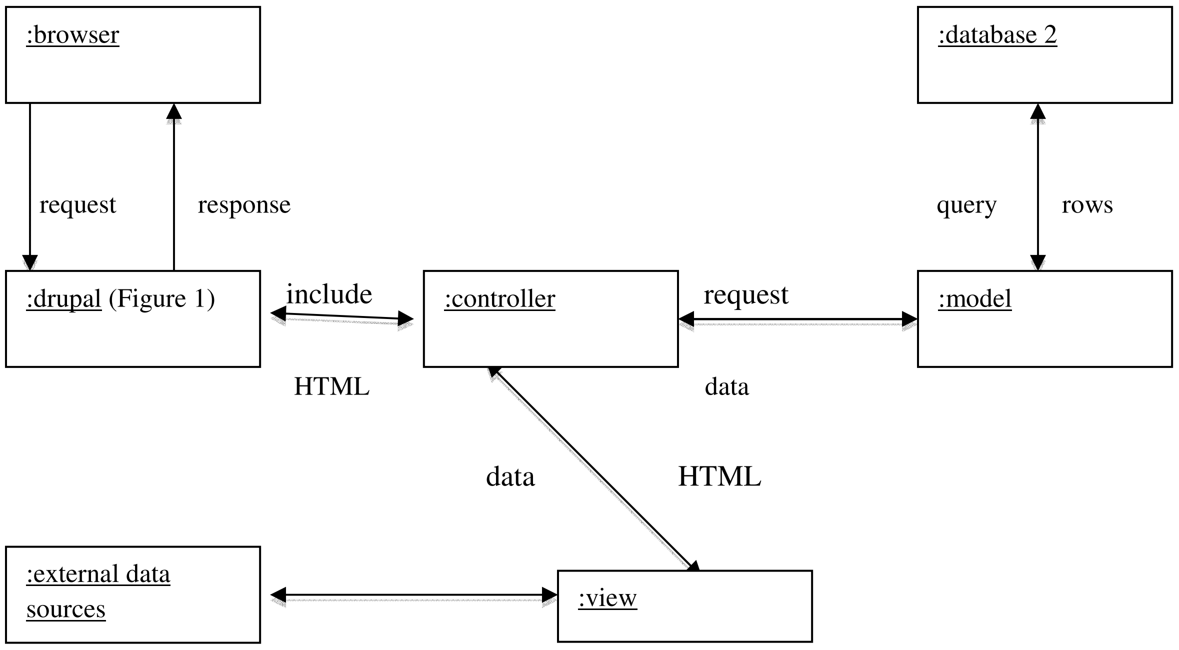


Figure 3 - Communication Diagram: Drupal & MVC

The controller is the main component of a MVC structure. It retrieves data via the model and uses views to create the Web pages. It fits well with the capabilities we are developing on *oystersentinel.org*. The controller contains the code necessary to interact with the model and the view, and run the model. To apply this design to the Drupal CMS system, it needs to be invoked by client's request through Drupal's template (Figure 1).

3.2 *Perkinsus marinus* Model

Perkinsus marinus is a bio-indicator of mesohaline salinity regimes and is the principal parasite of *Crassostrea virginica*, also called eastern oyster, a species of oyster native to the Gulf of Mexico. The primary goal of this model is to help assess the level of infection of oysters along the Gulf coast. For data collection, the Gulf of Mexico is divided into eighteen areas in which there are more than seventy stations. Each area has at least one station and each station has an owner who is a biologist in charge of collecting data. The periodically collected data includes the quantity of sampled oysters, their size, and the quantity of sampled oysters infected by *Perkinsus marinus*. Monitoring the health of oysters with this model is vital in order to preserve and maintain a sustainable quantity of oysters because a high level of infection could be an indicator of oyster mortality.

The *Perkinsus marinus* model can be accessed by different type of users: Website administrators, station owners, and general users. In order to maintain a reliable dataset and provide proper privileges to each user, it is important to set up an assets manager. This manager grants features to users according to their privileges. For example, an owner can add a station, edit or delete owned stations, and manage the data. A general user can only view areas, stations, and data. The following table shows the action privileges of each type of user.

	Administrator	Owner	General User
Area	A, E, D, V	V	V
Station	A, E, D, V	A, E*, D*, V	V
Data	A, E, D, V	A*, E*, D*, V	V

Legend: A - Add, E - Edit, D - Delete, V - View; * only owned station

The assets manager follows the MVC design pattern and each component (Areas, Stations, and Data) has a controller that manages a part of the application. Owners are the primary source of information in this model. They collect data at their stations and enter them into the system. They can also run the model and get a result that is descriptive of the health of oysters at their stations. It is imperative to build a user interface that allows them to easily select any owned stations and enter the data with a minimum risk of errors. For cases in which areas and stations have location coordinates, using a map will enrich the interface and provide a visual representation of the managed assets.

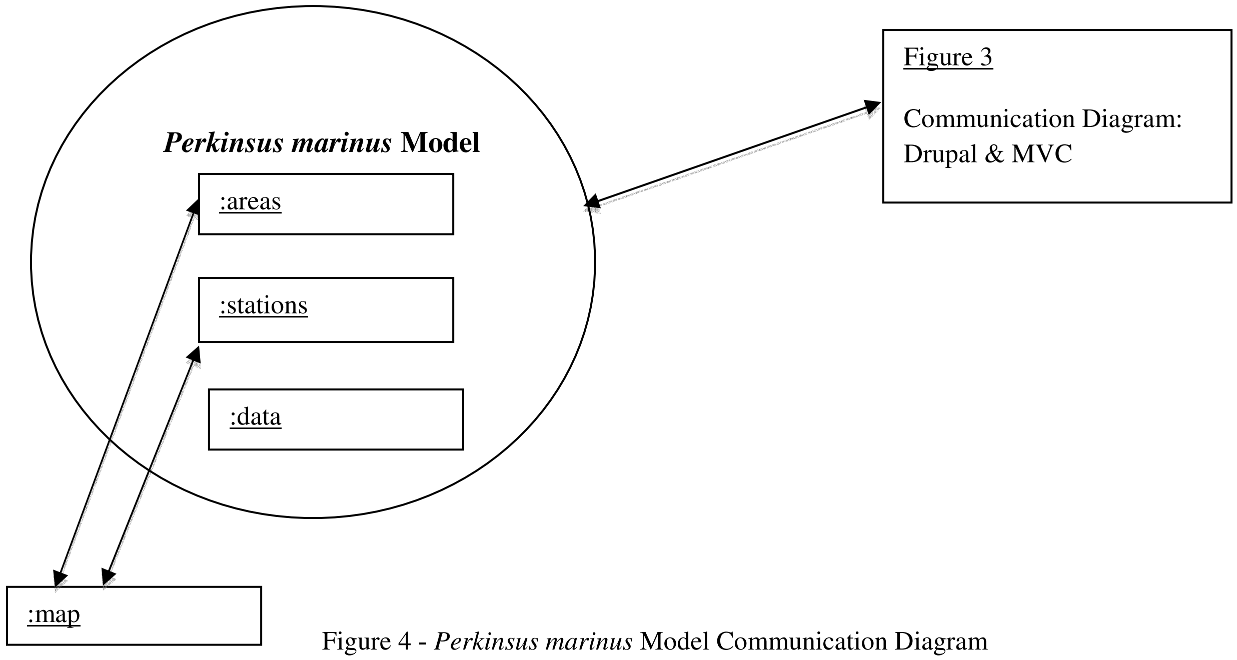


Figure 4 - *Perkinsus marinus* Model Communication Diagram

Figure 4 illustrates the components that are active in the *Perkinsus marinus* model. The system follows the MVC pattern and allows us to implement the Web application without using Drupal modules. It also gives us a quicker way to integrate a map from an external data source.

3.3 Oil Spill Model

The Oil Spill model is a geospatial model that provides a comprehensive set of capabilities in data retrieval and is implemented with a focus set on information and decision-making about oyster habitat environments. After the explosion of Deepwater Horizon in April 2010, authorities and experts have been trying to implement solutions that will effectively reduce the impact of similar disasters in the future. The Deepwater Horizon oil spill released about 5 million barrels of crude oil in the Gulf of Mexico between April and July of 2010 [Soniat, 2011]. It is considered the largest environmental disaster in U.S. history, with extensive damage to marine and wildlife habitats in the Gulf of Mexico. The Oil Spill model is to help decision makers and domain experts to evaluate impact of oil spills to oyster resources, predict time and place of impact, decrease response time, prioritize efforts, and even assist in the determination of monetary damages.

The first step in designing this model is to select the resources. In the event of an oil spill, oyster reefs are the assets being monitored. In the design of the *Perkinsus marinus* model, we introduced a second database that is used to store data on oysters and their stations. It is a natural choice to get station data from the same database.

The second step consists of finding reliable data sources about oil spills. When an oil spill occurs, many governmental organizations are involved in the process of protecting wildlife and informing the general public. The Louisiana Oil Spill Coordinator's Office (LOSCO), the Environmental Response Management Application (ERMA), and the National Oceanic and Atmospheric Administration (NOAA) are well known for their efforts in 2010 in the Deepwater Horizon oil spill. The Oil Spill model can use them to gather information on an oil spill and its progression. With these external data in the model, experts would have access to a more comprehensive analysis. The particularity about the Oil Spill model is its prediction feature. Domain experts can use this feature to analyze the course of an oil spill and detect the resources (oyster reefs) that need to be prioritized. This will reduce response time and help in protecting oysters in danger. The prediction tool requires access to weather data such as currents, winds, and temperature. It also requires the use of a reliable mathematical equation that will produce the path of the oil spill at a specific time. However, developing it in-house would demand extensive hours of programming and the outcome might not be efficient. Instead, the Oil Spill model would consume a proven and reliable Web service to complement the prediction tool.

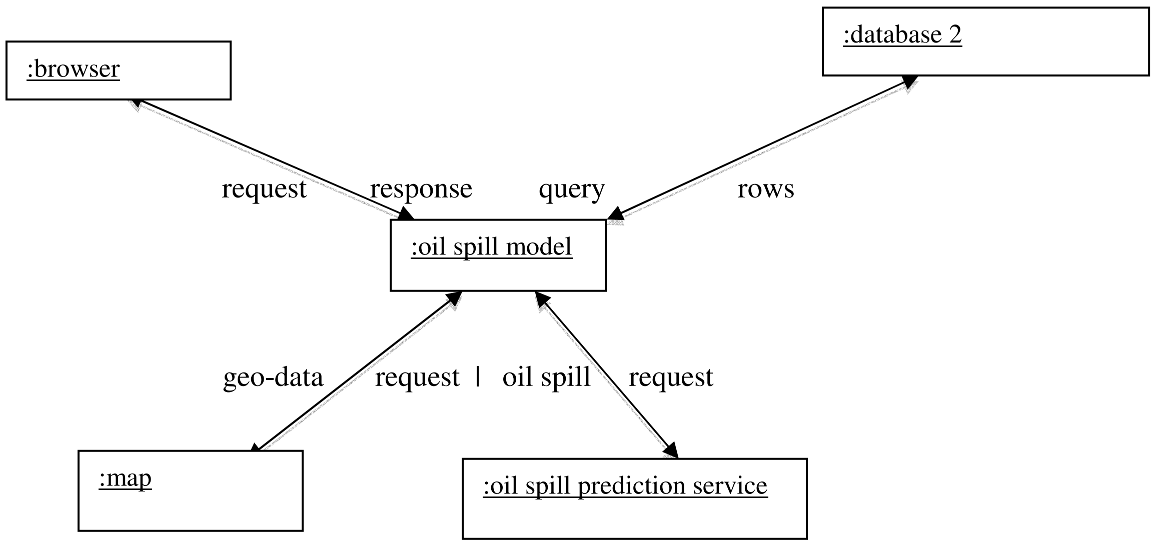


Figure 5 – Oil Spill model Communication Diagram

3.4 Fisheries Web-based Service

Web mashup techniques promote data reusability by allowing developers to consume external data into their Web application. However, a large percentage of Web sites provide data on their platform but do not implement interfaces for developers to access those data. This limitation negatively affects the development of rich and reliable Web applications. The Fisheries Web-based service is designed solely for the purpose of understanding programming interfaces and the methods to access them externally. It is not a live interface.

The interface will allow outside programmers to use internal data in external applications without compromising the reliability and the validity of those data. For security reasons, it is imperative to hide the structure of the database. Outside developers do not need to know the type of database, the tables, nor the query language used internally. The primary objective of the interface is to abstract this information. Its second objective is to grant read-only access to a specific set of data. Figure 6 describes how the Fisheries API interacts with the other components of the system.

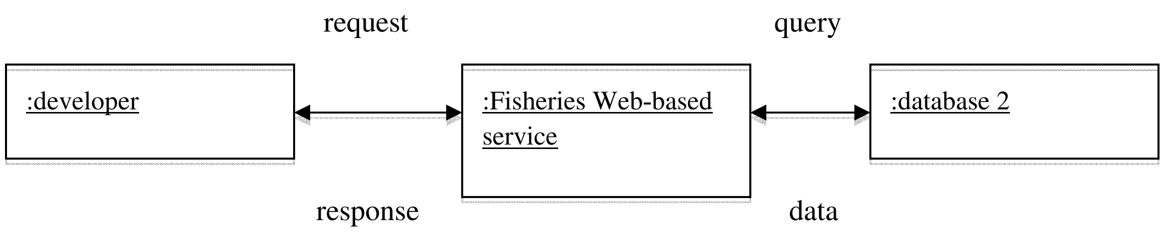


Figure 6 - Fisheries API Abstract Communication diagram

Chapter 4: **Implementation**

Using the design described in chapter 3, we now are to implement each application. It involves primarily configuring the system structure and the database, and then writing code for each component in the design.

4.1 System Structure

In section 3.1, we described the system structure of *oystersentinel.org*, the framework used, and the solution to deploy our Web applications. For security and performance reasons, the Web applications implemented on the Web site are backed by a Microsoft SQL Server (MSSQL) database instead of the Drupal's MySQL content database. MSSQL is commonly used for Web applications on a Windows platform. It also provides buffer management, transparent data encryption, and enterprise-level management tools for developers. With the Web site hosted on a Windows server machine, it is a natural choice to use a MSSQL database to store sensitive data on oysters.

As described in section 3.1.3, the MVC model component handles access to the database. It encapsulates the most commonly needed functions such as connecting to the database, running a query, getting a specific row, or getting all the rows. Thus, we have implemented a "generic" component, model.php, located in the /mvc/models folder. Any PHP function can use this component by including this file in it. It contains the following functions:

- db_model_connect(): creates a connection to the MSSQL database

- runSQL($sql, $params, $id) : takes a SQL query and its parameters and runs it on the database. It will return the corresponding rows or a boolean.

- get($table, $id): fetches a row in a table with the specified id.

```
<?php
   function get($table, $id){
     if($id == 0){
       $query = "SELECT TOP 1 * FROM ".$table;
       $params = array();
     } else {
       $query = "SELECT * FROM ".$table." WHERE id = ?";
       $params = array(0 => array('value' => $id, 'type' => PDO::PARAM_INT));
     }
     return runSQL($query, $params);
   }
?>
```

- getAll($table, $order_by): returns all the rows of a table, and order it if specified.

```
<?php
   function getAll($table, $order_by = ''){
     $query = "SELECT * FROM ".$table;
     if(!empty($order_by)){
       $query .= " ORDER BY ".$order_by;
     }
     return runSQL($query);
   }
?>
```

4.2 *Perkinsus marinus* Model

The *Perkinsus marinus* model is built around two major pieces: the assets manager and the mapping service. The assets manager provides the forms to manage oysters and handles user privileges. Using Drupal's global variable $user, the assets manager can determine the role of a user and present the forms granted by the user's privileges. The role of the mapping service lies in the visual aspect of the application. Google provides maps and directions on their Web site *maps.google.com* and has developed an API that allows developers to use their maps on any Web sites. Built with JavaScript, Google Maps API is a client-side service running on the user's browser. Any browser with a JavaScript interpreter can run Google Maps. Google Maps has also a wide array of classes and functions that developers can use to overlay their data on top of a map.

The *Perkinsus marinus* model's main functionalities are for an owner to: add data about sampled oysters in a station, run the model, and analyze the level of infection. Figures 7 & 8 show the complete database structure of this model.

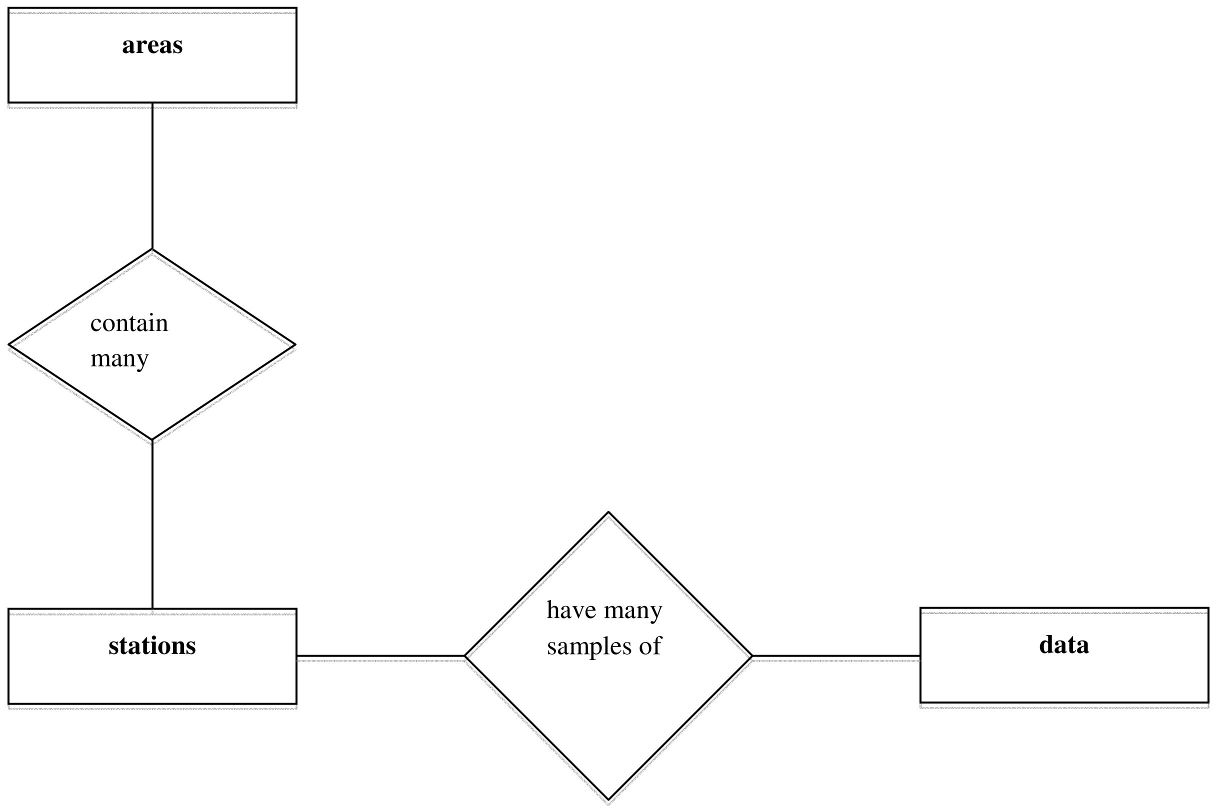


Figure 7 - Entity-Relationship Diagram- *Perkinsus marinus* model

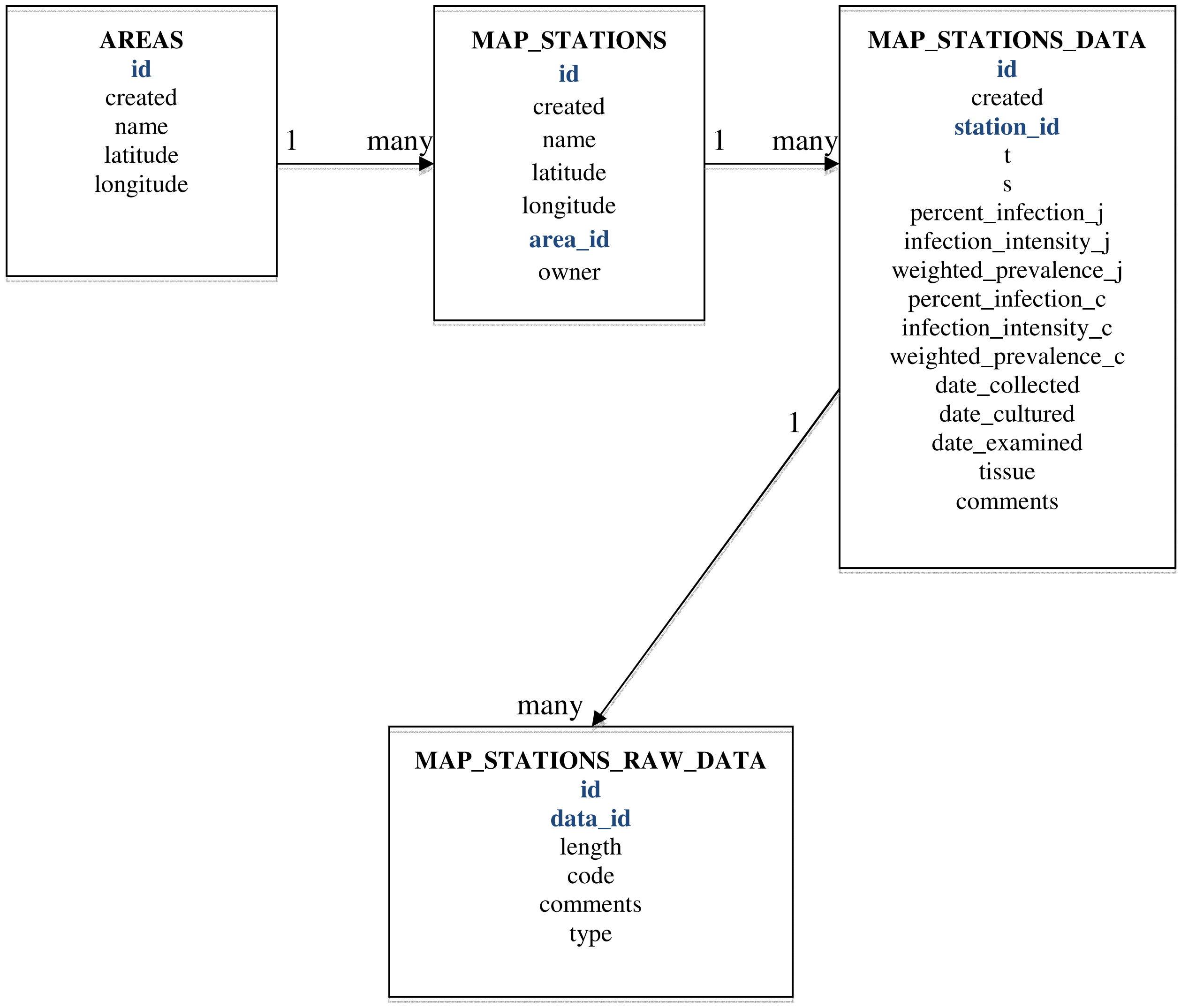
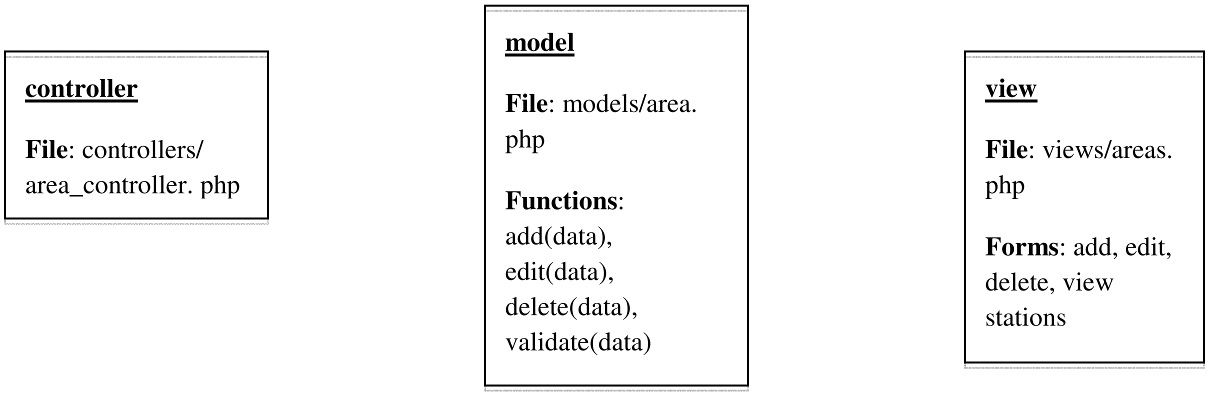


Figure 8 - Schema Diagram - *Perkinsus marinus* model

The user interface for this model is made up of three pages that will guide the user and minimize errors.

4.2.1 Areas Selection

Before selecting a station, a user first selects its area. To implement the areas page, we created a controller, a model, and a view. The interface of the areas page is composed of a map from Google Maps showing all the areas currently in the system and all the forms accessible by the current user. The following class diagram shows the different components and their actions.



controller
File: controllers/ area_controller. php
model
File: models/area. php
Functions: add(data), edit(data), delete(data), validate(data)
view
File: views/areas. php
Forms: add, edit, delete, view stations

Figure 9 - Areas Class Diagram

The controller in this application is invoked from Drupal's template page (Figure 1) and its role is to fetch all the areas from the database and handle any form submission (see code snippet below).

```
<?php

// Include areas model which contains functions to interact with the 'areas'
table
      include dirname(__FILE__).'/../models/area.php';
      $areas = getAll('AREAS', 'name'); // Retrieving all the areas

      // Add an area
      if(isset($_POST['add_area'])){
            $data['name'] = $_POST['add_area_name'];
            $data['latitude'] = $_POST['add_area_lat'];
            $data['longitude'] = $_POST['add_area_long'];

            $valid = validate($data); // Validate Data submitted
            $err = "";
            if($valid) add($data); // Save data in database
            else $err = "&error=true";
            header("Location: /?q=areas".$err);
      }
```

By including the model, the controller can use the functions getAll(), validate(), and add() to fetch and save data in the database. It also uses the edit function when an administrator edits an area, or the delete function when an administrator deletes an area.

The area model includes the "generic" model as described in section 4.1 and implements the functions add() (see code snippet below), edit(), delete(), and validate().

```
<?php
   include 'model.php';

   function add($data){
     $query = "INSERT INTO AREAS (created, name, latitude, longitude) VALUES
(?, ?, ?, ?)";
     $now = date('Y-m-d G:h:s');
     $params = array(
       0 => array('value' => $now, 'type' => PDO::PARAM_STR),
       1 => array('value' => $data['name'], 'type' => PDO::PARAM_STR),
       2 => array('value' => $data['latitude'], 'type' => PDO::PARAM_STR),
       3 => array('value' => $data['longitude'], 'type' => PDO::PARAM_STR)
     );
     $result = runSQL($query, $params);
     if(isset($result['error'])) $return = 0;
     else $return = 1;
     return $return;
   }
?>
```

The areas controller only uses one view and all the forms (add, edit, delete, view) are appended according to the user's privileges by using Drupal's global variable $user (see code snippet below). When a user is registered in the Web site, he/she is assigned a role (poweruser, owner, or authenticated user). Drupal stores this information as an array 'roles' in the global variable $user that we can access from the MVC controller.

```
<div class="titles">
       <div class="titles_item"><p class="expand" id="view"></p><h3>View
Area</h3></div>
       <?php if(in_array('poweruser', $user->roles)){ ?>
       <div class="titles_item"><p class="expand" id="add"></p><h3>Add
Area</h3></div>
       <div class="titles_item"><p class="expand" id="edit"></p><h3>Edit
Area</h3></div>
       <div class="titles_item last"><p class="expand"
id="del"></p><h3>Delete Area</h3></div>
       <?php } ?>
 </div>
```

It is in the view that the external service and data from Google Maps is consumed and integrated with the data from the database (see code snippet below).

```
<script type="text/javascript" src="http://maps.googleapis.com/maps/api/js?
sensor=false"></script>

<script type="text/javascript">
   jQuery(document).ready(function(){
     var neworleansLat = 29.951066;
     var neworleansLong = -90.071532;
     var myLatlng = new google.maps.LatLng(neworleansLat, neworleansLong);
     var myOptions = {
            zoom: 6,
            center: myLatlng,
            mapTypeId: google.maps.MapTypeId.HYBRID
     };
     var map = new google.maps.Map(
            document.getElementById("areas_map"),
            myOptions
     );
```

By using a JavaScript API, we can easily mix data from our MSSQL database and Google Maps, even though they belong to two different contexts. Data from the database is accessible via a PHP array and before the areas page is sent to the client as a HTML file, the server interprets the data and makes it accessible to Google Maps (see code snippet below).

```
<?php foreach($areas as $area){ ?>


     var marker = new google.maps.Marker({
        position: new google.maps.LatLng(
            <?php echo $area['latitude']; ?>,
            <?php echo $area['longitude']; ?>
        ),
        map: map,
        title:"<?php echo $area['name']; ?>",
        icon: "<?php echo variable_get('file_public_path', conf_path() .
'/files'); ?>/images/circles_30x30.png"
     });
     marker.setMap(map);
     google.maps.event.addListener(
        marker,
        'click',
        function() {
          location.href = "/?q=map_stations&area=<?php echo $area['id']; ?>";
        });
<?php } ?>
```

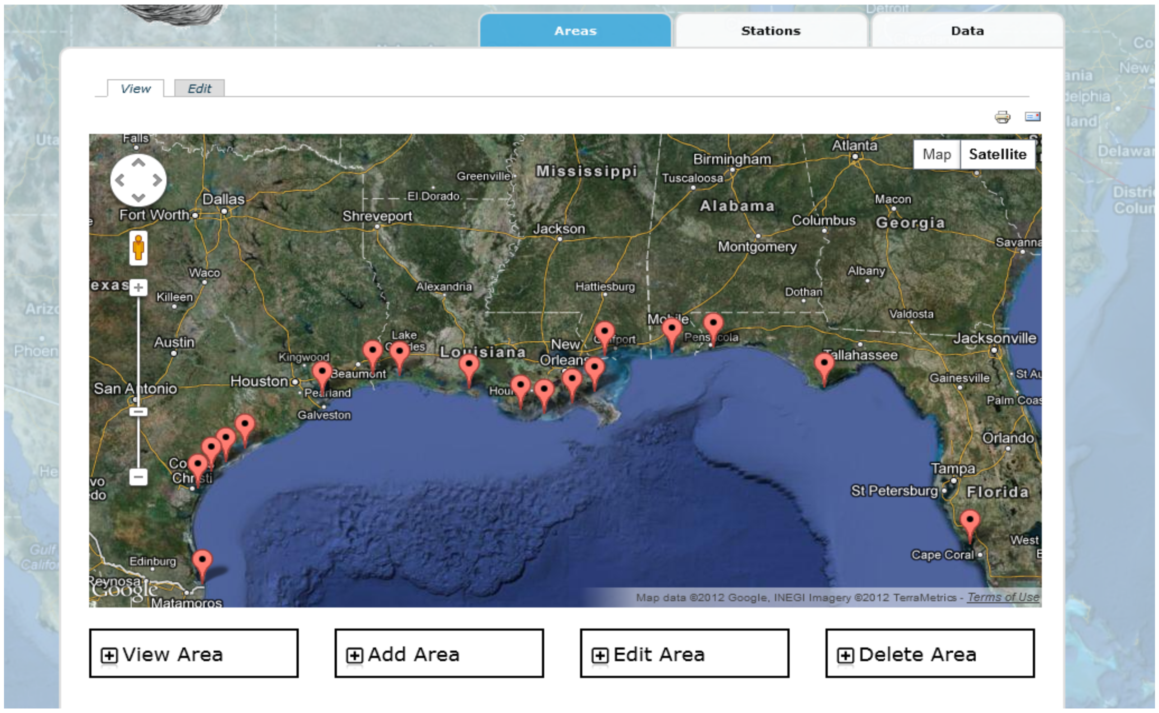


Figure 10 - Areas page

Figure 10 is a snapshot of the Areas page when visited by an administrator. The map shows markers representing the center of each station and the administrator can use the different tabs to view, add, edit, or delete any area.

4.2.2 Stations Selection

After selecting an area in the areas page, the user is taken to the stations page. The implementation of the stations page follows the same pattern as the areas page. We created a controller, a model, and a view. The class diagram in Figure 11 shows the components and their actions.

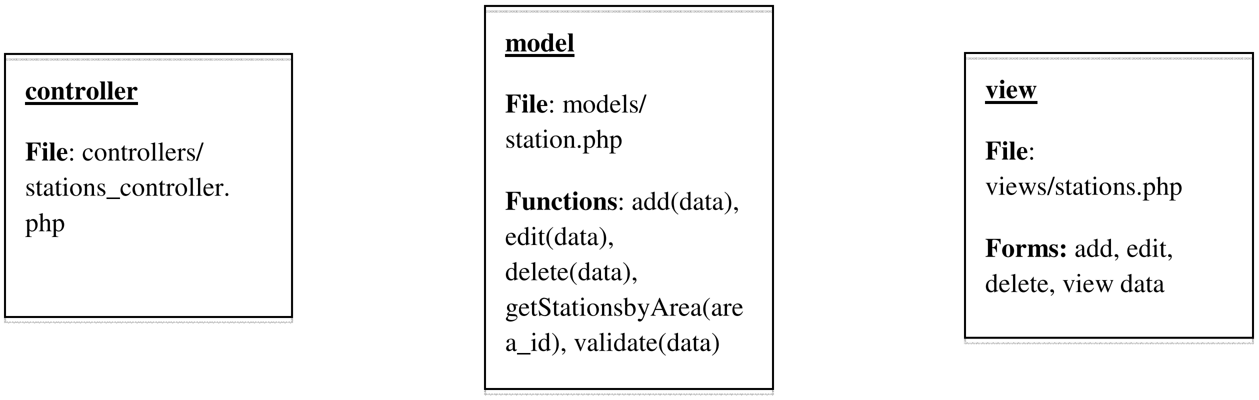
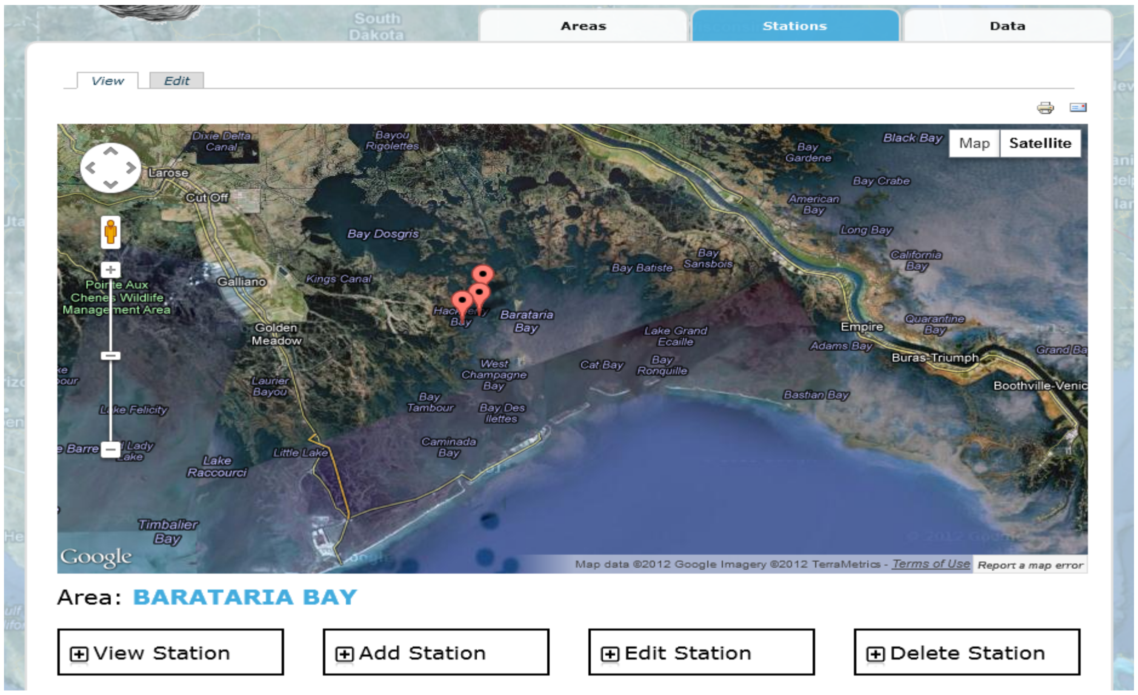

controller
File: controllers/ stations_controller. php
model
File: models/ station.php
Functions: add(data), edit(data), delete(data), getStationsbyArea(area_id), validate(data)
view
File: views/stations.php
Forms: add, edit, delete, view data

Figure 11 - Stations Class Diagram

The stations controller is similar to the areas controller. It also uses one view to display the forms (add, edit, delete, view) and checks the global variable $user (see code snippet below). Only an administrator or an owner is allowed to add, edit, or delete a station.

```
<div class="titles">
   <div class="titles_item">
      <p class="expand" id="view"></p><h3>View Station</h3>
   </div>
   <?php if(in_array('poweruser', $user->roles) || in_array('owner', $user-
>roles)){ ?>
   <div class="titles_item">
      <p class="expand" id="add"></p><h3>Add Station</h3>
   </div>
   <div class="titles_item">
      <p class="expand" id="edit"></p><h3>Edit Station</h3>
   </div>
   <div class="titles_item last">
      <p class="expand" id="del"></p><h3>Delete Station</h3>
   </div>
   <?php } ?>
</div>
```



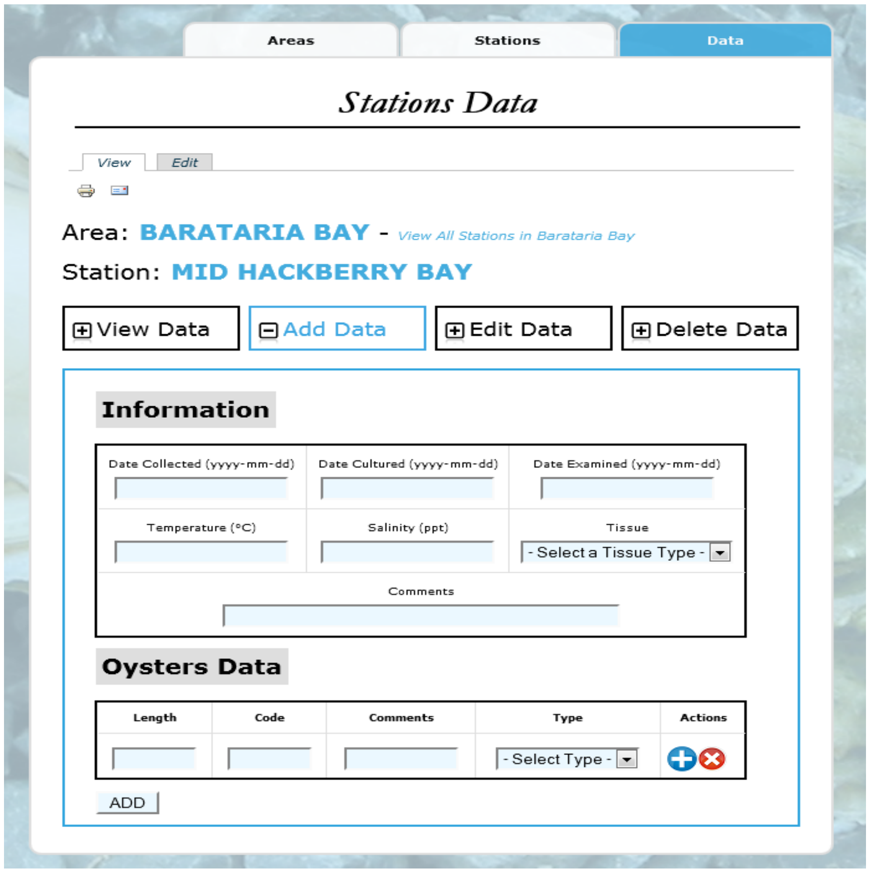
Area: BARATARIA BAY

⊞ View Station ⊞ Add Station ⊞ Edit Station ⊞ Delete Station

Figure 12 – Stations page (Barataria Bay stations)

4.2.3 Data Page

The data page follows the same design pattern as the areas and stations pages but some database tables and functions were added. After selecting the area and the station, the user is taken to the data page, which is where an owner can enter the number of oysters, their size, and disease code. After submitting the form (Figure 13), the controller formats the data and computes three critical values: the percent infection, the infection intensity, and the weighted prevalence (Figure 14). These values are stored in the database and represent the level of infection in that station at the time of sampling (Figure 15).



Area: BARATARIA BAY

Station: MID HACKBERRY BAY

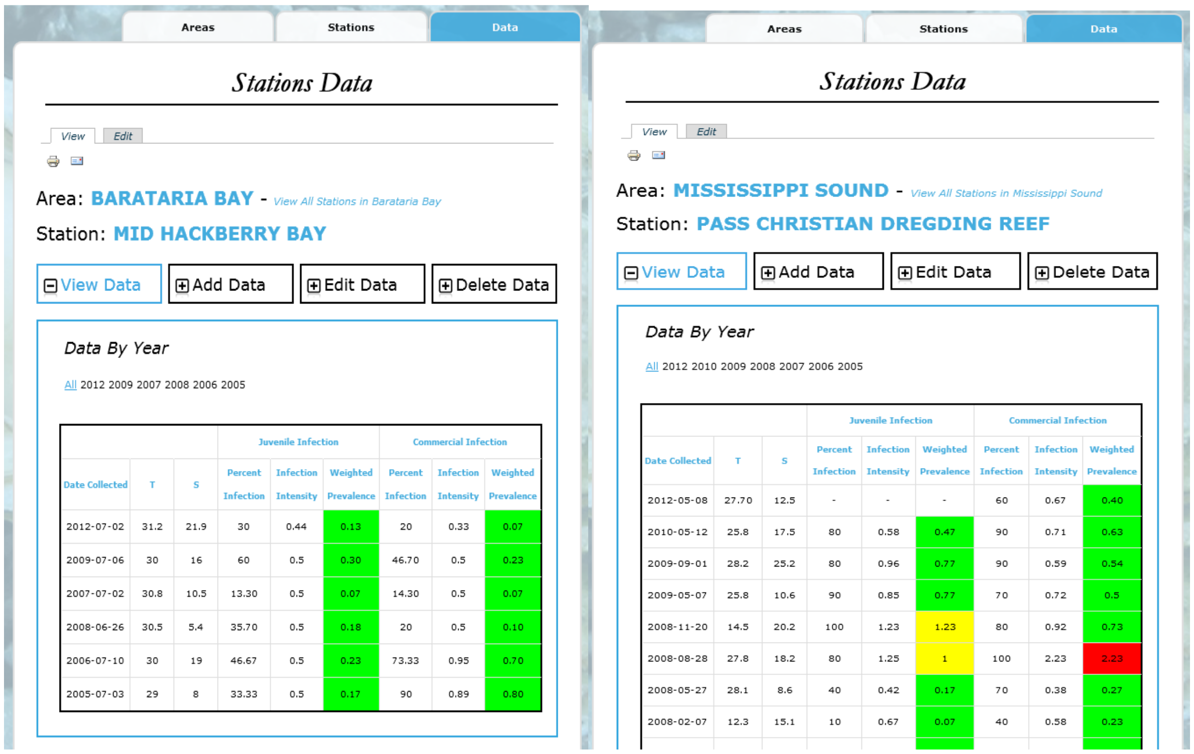
Date Collected (yyyy-mm-dd)	Date Cultured (yyyy-mm-dd)	Date Examined (yyyy-mm-dd)
Temperature (°C)	Salinity (ppt)	Tissue: - Select a Tissue Type -
Comments		

Length	Code	Comments	Type	Actions
			- Select Type -	

Figure 13 – Add Data form

```
<?php
   //Validate Oysters Data
   $valid_oysters = validate_oysters($data['oysters']);

   if($valid_oysters){
      $data['percent_infection_c'] = ($oysters[1]['count'] == 0) ? '' :
($oysters[1]['infected'] * 100) / $oysters[1]['count'];

      $data['infection_intensity_c'] = ($oysters[1]['infected'] == 0) ? ''
: $oysters[1]['disease'] / $oysters[1]['infected'];

      $data['weighted_prevalence_c'] = ($oysters[1]['count'] == 0) ? '' :
$oysters[1]['disease'] / $oysters[1]['count'];

      //If Station's data and Oysters data are valid, save all
      $last_id = add($data);
      add_oysters($data['oysters'], $last_id);
   }
?>
```

Figure 14 - Calculating the percent infection, infection intensity, and weighted prevalence



Stations Data

Area: BARATARIA BAY - View All Stations in Barataria Bay

Station: MID HACKBERRY BAY

Data By Year

All 2012 2009 2007 2008 2006 2005

			Juvenile Infection			Commercial Infection		
Date Collected	T	S	Percent Infection	Infection Intensity	Weighted Prevalence	Percent Infection	Infection Intensity	Weighted Prevalence
2012-07-02	31.2	21.9	30	0.44	0.13	20	0.33	0.07
2009-07-06	30	16	60	0.5	0.30	46.70	0.5	0.23
2007-07-02	30.8	10.5	13.30	0.5	0.07	14.30	0.5	0.07
2008-06-26	30.5	5.4	35.70	0.5	0.18	20	0.5	0.10
2006-07-10	30	19	46.67	0.5	0.23	73.33	0.95	0.70
2005-07-03	29	8	33.33	0.5	0.17	90	0.89	0.80

Stations Data

Area: MISSISSIPPI SOUND - View All Stations in Mississippi Sound

Station: PASS CHRISTIAN DREGDING REEF

Data By Year

All 2012 2010 2009 2008 2007 2006 2005

			Juvenile Infection			Commercial Infection		
Date Collected	T	S	Percent Infection	Infection Intensity	Weighted Prevalence	Percent Infection	Infection Intensity	Weighted Prevalence
2012-05-08	27.70	12.5	-	-	-	60	0.67	0.40
2010-05-12	25.8	17.5	80	0.58	0.47	90	0.71	0.63
2009-09-01	28.2	25.2	80	0.96	0.77	90	0.59	0.54
2009-05-07	25.8	10.6	90	0.85	0.77	70	0.72	0.5
2008-11-20	14.5	20.2	100	1.23	1.23	80	0.92	0.73
2008-08-28	27.8	18.2	80	1.25	1	100	2.23	2.23
2008-05-27	28.1	8.6	40	0.42	0.17	70	0.38	0.27
2008-02-07	12.3	15.1	10	0.67	0.07	40	0.58	0.23

Figure 15 – Data for Stations: Mid Hackberry Bay: minimal level of infection since 2005 (*weighted prevalence < 1*); Mississippi Sound: Overall healthy area besides increase of infection in summer of 2008 (*weighted prevalence >= 1 & < 2* – **yellow** and *weighted prevalence >= 2* – **red**).

Figure 16 shows the tables and views involve in this piece of the application. Data unique to each sample such <station_id> are stored in the table "map_stations_data" in one row. However, the number of sampled oysters varies and to normalize the database, oyster data are stored in a separate table "map_stations_raw_data". The table "map_stations_data" has a primary key <id> that is used as a foreign key <data_id> in the table "map_stations_raw_data". Setting up these relational tables helps maintain a consistent dataset and obtain an efficient retrieval of data when viewing or editing a row.

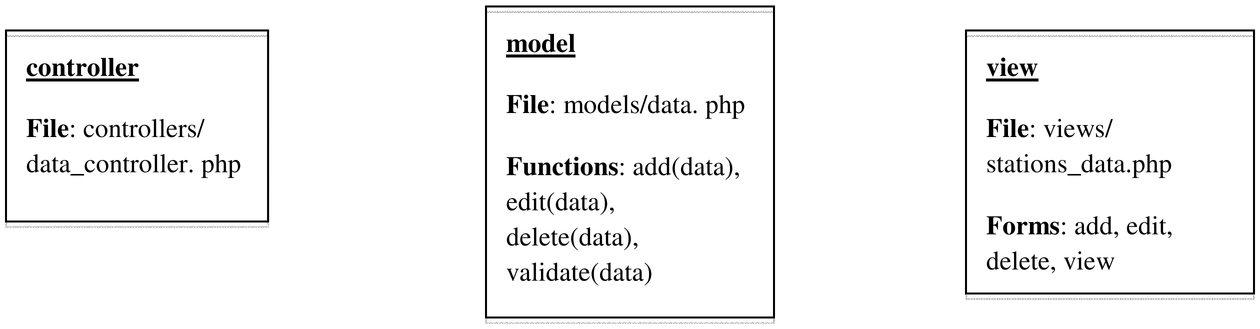

controller
File: controllers/ data_controller. php
model
File: models/data. php
Functions: add(data), edit(data), delete(data), validate(data)
view
File: views/ stations_data.php
Forms: add, edit, delete, view

Figure 16 - Data Class Diagram

The data controller uses one view to display the forms (add, edit, delete, view) and checks the global variable $user (see code snippet below) to grant access to each form.

```
<div class="titles data">
   <div class="titles_item">
      <p class="expand" id="view"></p><h3>View Data</h3>
   </div>
   <?php if(in_array('poweruser', $user->roles) || (in_array('owner', $user->roles) && $user->name == $station[0]['owner'])){ ?>
   <div class="titles_item">
      <p class="expand" id="add"></p><h3>Add Data</h3>
   </div>
   <div class="titles_item">
      <p class="expand" id="edit"></p><h3>Edit Data</h3>
   </div>
   <div class="titles_item last">
      <p class="expand" id="del"></p><h3>Delete Data</h3>
   </div>
   <?php } ?>
</div>
```

The *Perkinsus marinus* model is a 3-step Web application that uses Web mashup technology to enrich the user interface. It helps administrators, owners, and general users, in understanding the spatio-temporal distribution of the parasite across the Gulf of Mexico.

4.3 Oil Spill Model

The Oil Spill model is an application that has been designed but not implemented. For the purpose of demonstrating and experimenting Web Mashup technology, we implemented a simpler version of the originally designed Oil Spill model. This version of the Oil Spill model is built around five main components: the mapping service, the oyster stations, the oil spills manager, the prediction tool, and the query tool.

4.3.1 Mapping service

As used throughout the Web site, Google Maps is the mapping service used in the user interface of this model. The map helps position stations and place markers for oil spills. It also facilitates the representation of the oil spill as it progresses and the different classes provided by Google Maps API helps implement the geospatial queries. The following graph shows the role of the map relative to the other components of the model.

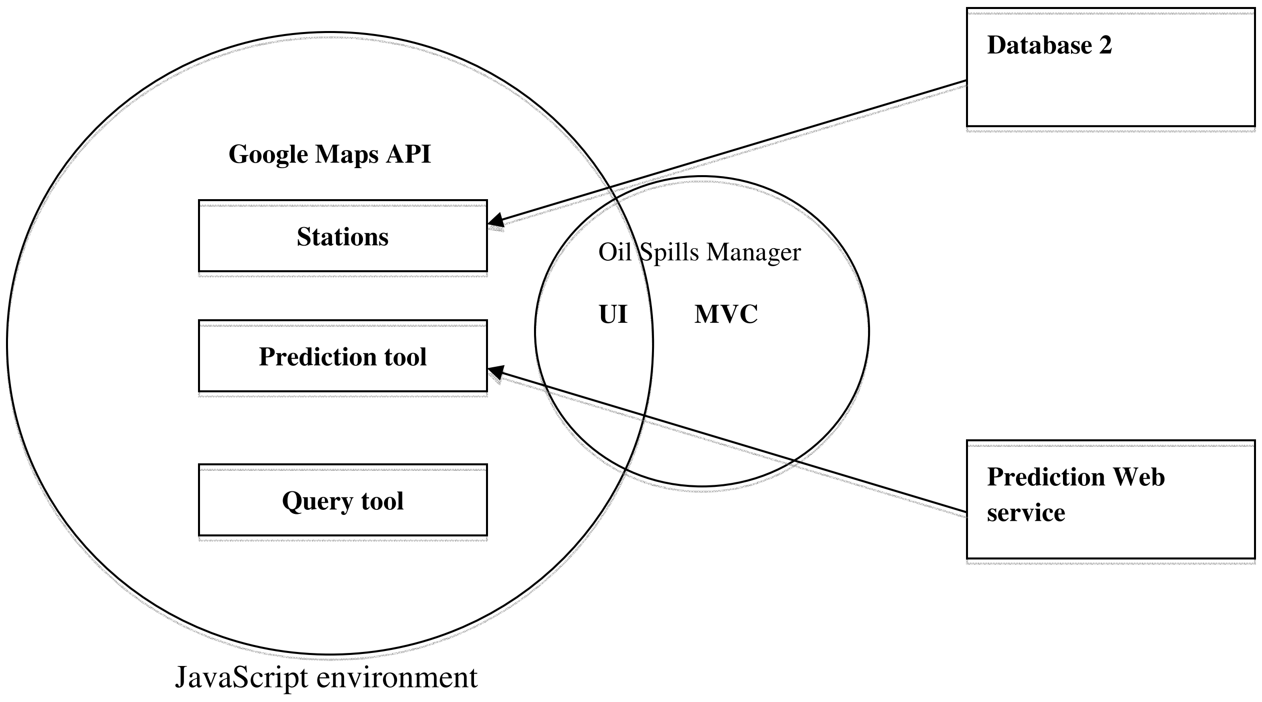


Figure 17 - Components of Oil Spill Model

4.3.2 Stations

As identified in the design section 3.3, oyster stations are the resources that are being monitored by the model. They are the same resources that are being managed in the *Perkinsus marinus* model. In consequence, it is a natural choice to use the MVC model 'station.php' described in section 4.2.2. It provides a function that fetches the list of all stations and their coordinates. The stations will be placed on the map.

4.3.3 Oil Spills Manager

When an oil spill occurs, it is monitored by various official agencies. Administrators of *oystersentinel.org* can use data from these agencies to add basic information about the oil spill.

The basic Information includes name, coordinates, starting date and time, and water temperature. Entering this information in the Oil Spills manager helps keep a local record of the oil spill and enrich our model. Figures 18 & 19 describe the database structure used.

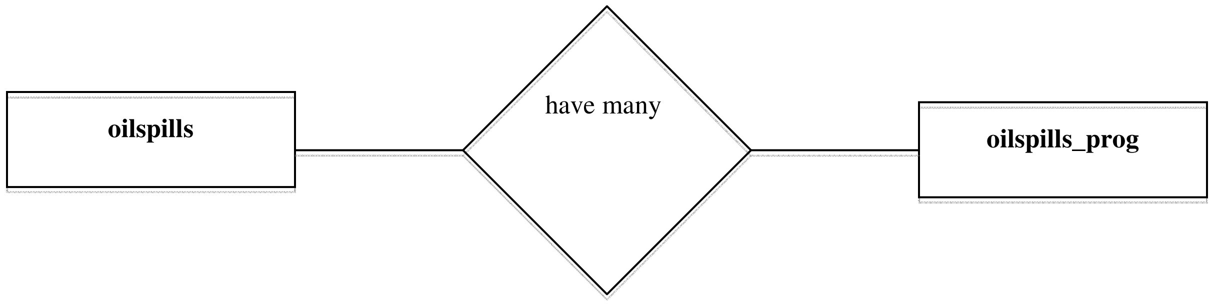


Figure 18 - Entity-Relationship Diagram- Oil Spill model

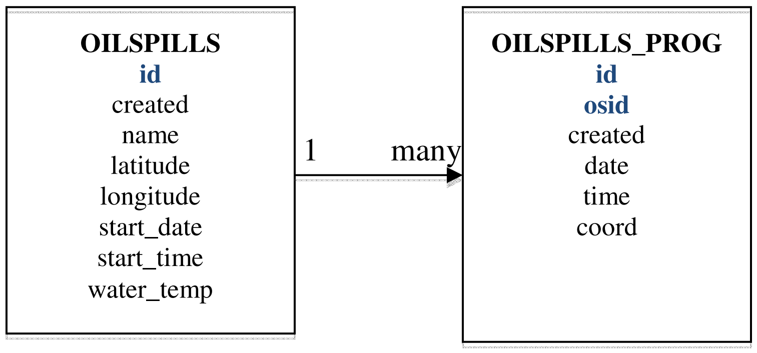


Figure 19 - Schema Diagram - Oil Spill model

The oil spills manager (Figure 20) is a simple Web application that follows the same MVC design pattern used in the *Perkinsus marinus* model.

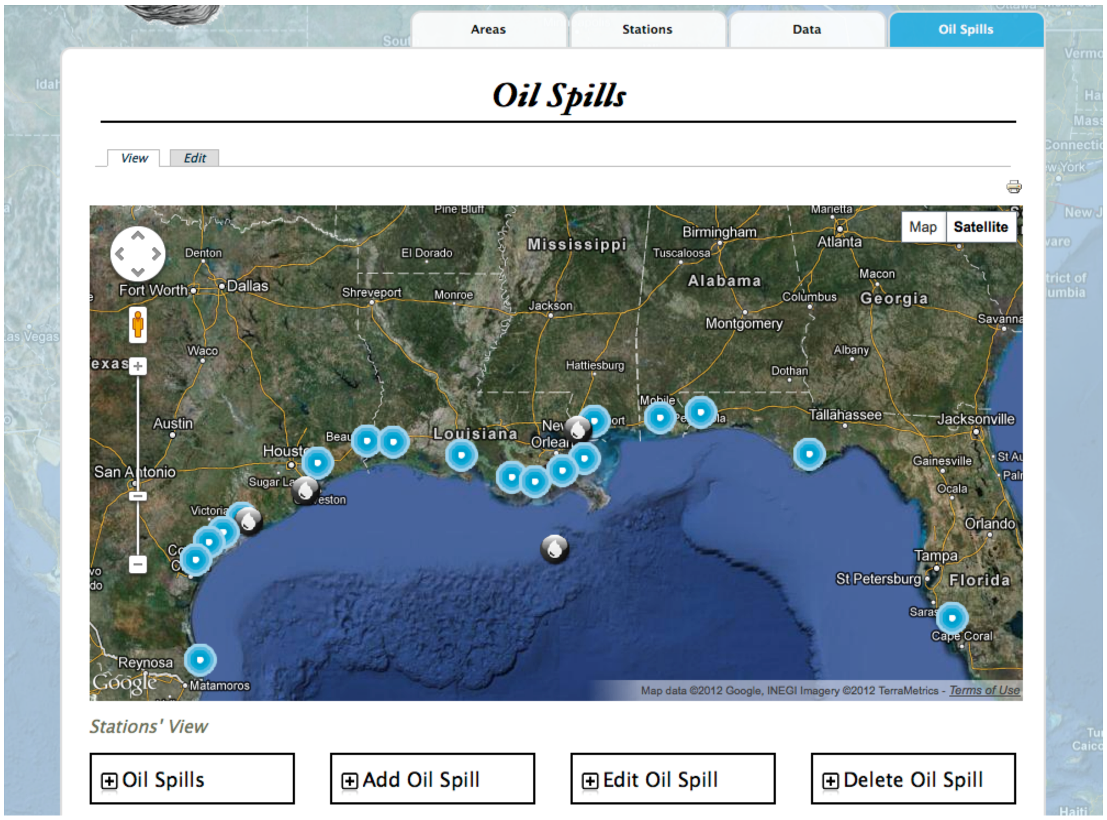


Figure 20 - Oil Spills manager

controller

File: controllers/ oilspill_controller. php

model

File: models/ oilspill.php

Functions: add(data), addProg(data), edit(data), delete(data), validate(data), getProgressions(id)

view

File: views/oil_spills.php

Forms: add, edit, delete, view data

Figure 21 - Oil Spills manager Class Diagram

Web site administrators can add, edit, or delete any entry in the system. The Oil Spills manager also provides an interface to update oil spills. After the start of an oil spill, it is imperative to track it, record its progression, and know its latest status. After selecting an oil spill on the map, the administrator is taken to the profile page of that oil spill (Figure 22). The profile page gives users tools to update the status of an oil spill and visualize the spill. Each update is represented in the system as a snapshot. After manually getting data from an official agency, the administrator can enter the date and time the snapshot was taken, as well as the polygon coordinates forming the updated shape of the oil spill.

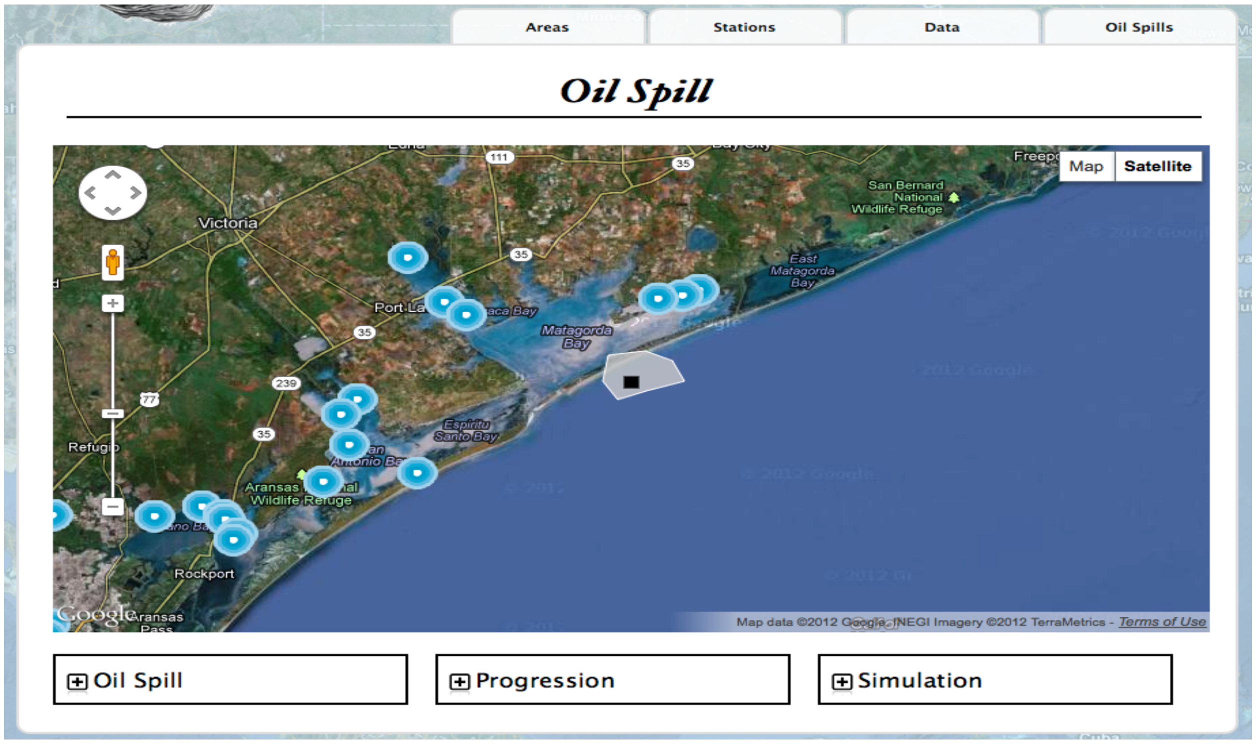


Figure 22 - Oil Spill profile page

Each snapshot is stored in the table <oilspills_prog> (Figure 19) by using the MVC model function addProg(data) (see code snippet below).

```
function addProg($data){
    $query = "INSERT INTO OILSPILLS_PROG (created, date, time, coord, osid)
              VALUES (?, ?, ?, ?, ?)";
    $now = date('Y-m-d G:h:s');
    $params = array(
                0 => array('value' => $now, 'type' => PDO::PARAM_STR),
                1 => array('value' => $data['date'], 'type' =>
PDO::PARAM_STR),
                2 => array('value' => $data['time'], 'type' =>
PDO::PARAM_STR),
                3 => array('value' => $data['coord'], 'type' =>
PDO::PARAM_STR),
                4 => array('value' => $data['osid'], 'type' =>
PDO::PARAM_INT)
             );
    $result = runSQL($query, $params);
    return (isset($result['error'])) ? 0 : 1;
}
```

After one or more snapshots are recorded, any user can use the Oil Spill viewer. The viewer is a piece of the oil spill manager and its role is to provide a visual representation of the oil spill from the start to the latest update. The viewer consists of two operations:

- View the snapshots chronologically: user can view the evolution of the oil spill.

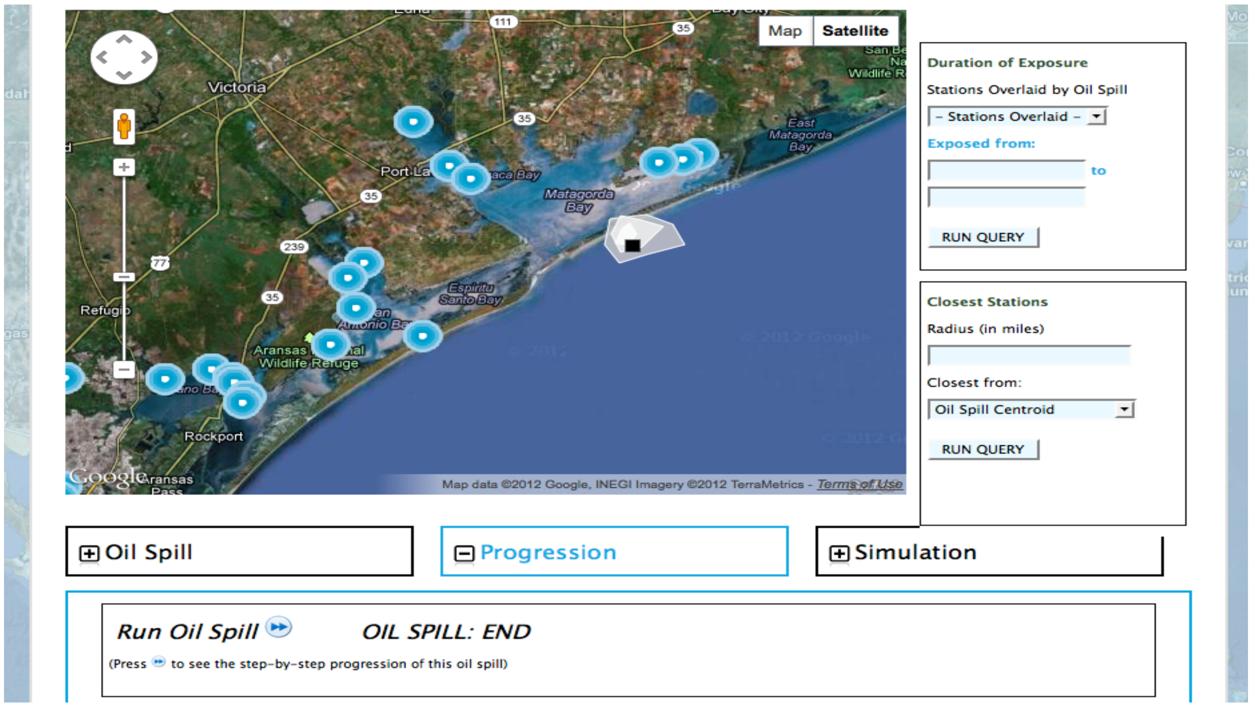
- View the snapshots one at the time: user can view the oil spill at a specific time.

The list of stations and the oil spill location are shown on the map via JavaScript Google Maps API. Google Maps also provides the class "google.maps.Polygon" that handles the drawing and manipulation of polygon objects on a map. With oil spills' polygons saved as strings, we use a JavaScript function (see code snippet below) to parse the coordinates and create a polygon object that represents the oil spill.

```
function setUpPolygon(count, colors, z, paths, stcol, fOpac){
    var pathArray = new Array();

    // Parsing coordinates
    var p = paths.toString().split(';');
    for(var i=0; i<p.length; i++){
       var c = p[i].toString().split(',');
       if(c[1] != '' && c[0] != ''){
          // Building an array of coordinates
          pathArray[i] = new google.maps.LatLng(c[1], c[0]);
       }
    }
    return new google.maps.Polygon({
                clickable:false,
                geodesic: true,
                strokeColor: stcol,
                strokeOpacity:1.0,
                strokeWeight:1,
                fillColor: "#"+colors[count],
                fillOpacity: fOpac,
                visible:false,
                zIndex: z,
                paths: pathArray });
 }
```

In its two operations, the Oil Spill viewer uses JQuery and the polygon method "setVisible(boolean)" to change the value of the attribute 'visible' and show or hide any polygons (see code snippet below).

```
/* Handles viewing Oil spill progression */
jQuery('#os_progressions').change(function(){
   for(i=0; i< stations_markers.length; i++){
      stations_markers[i].setVisible(true);
   }
   index = jQuery(this).val();
   if(index == -1)
      for(i=0; i< polygons.length; i++){
         polygons[i].setVisible(true);
      }
   else {
      for(i=0; i< polygons.length; i++){
         polygons[i].setVisible(false);
      }
      polygons[index].setVisible(true);
   }
});
```



Duration of Exposure

Stations Overlaid by Oil Spill

– Stations Overlaid –

Exposed from: to

RUN QUERY

Closest Stations

Radius (in miles)

Closest from:

Oil Spill Centroid

RUN QUERY

⊞ Oil Spill ⊟ Progression ⊞ Simulation

Run Oil Spill *OIL SPILL: END*

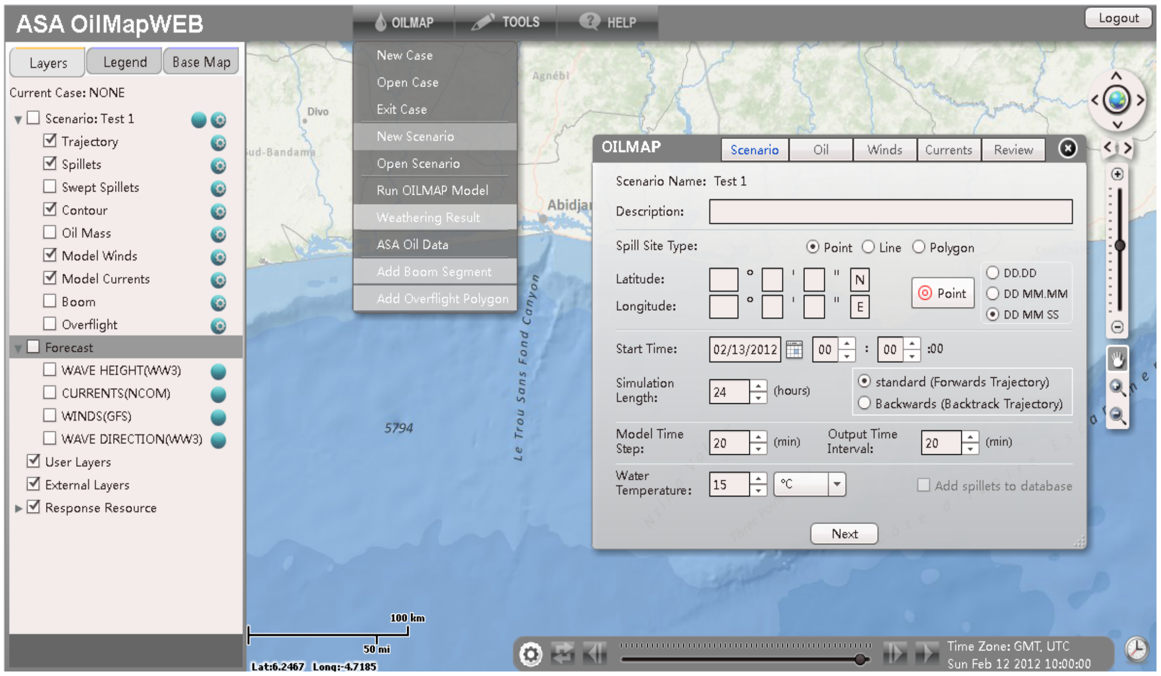
(Press to see the step-by-step progression of this oil spill)

Figure 23 - Oil Spill Viewer

4.3.4 Prediction Tool

The prediction tool is the centerpiece of the Oil Spill model. However, developing it in-house requires the use of external services to provide weather data (currents, winds, and temperature), and a reliable mathematical function to compute the area covered by the oil spill at any future time. After extended weeks of research, one existing tool satisfied the criteria set for our model: OceansMap. Developed by RPS - Applied Science Associates (ASA), a science and technology solutions company with headquarters in Rhode Island, OceansMap is an online service that allows registered users to analyze an oil spill and predict its path over the course of a scenario.

OceansMap is described as an online interactive Web-based oil spill simulation and spill response system, which consists of a GIS-based graphical user interface, a spatial database, and Web services from MapSERVER and the Environmental Data Server (EDS). OceansMap provides a rich interface that allows users to set up oil spill scenarios, run a trajectory and weathering model, and visualize the results of the model. The system also includes interactive GIS functions and tools to manage spill planning activities. Figures 24, 25, and 26 are screenshots of OceansMap and they are used here to show the different features provided by the service.



Current Case: NONE

- ☐ Scenario: Test 1
  - ☑ Trajectory
  - ☑ Spillets
  - ☐ Swept Spillets
  - ☑ Contour
  - ☐ Oil Mass
  - ☑ Model Winds
  - ☑ Model Currents
  - ☐ Boom
  - ☐ Overflight
- ☐ Forecast
  - ☐ WAVE HEIGHT(WW3)
  - ☐ CURRENTS(NCOM)
  - ☐ WINDS(GFS)
  - ☐ WAVE DIRECTION(WW3)
- ☑ User Layers
- ☑ External Layers
- ☑ Response Resource

OILMAP

Scenario Name: Test 1

Description:

Spill Site Type: [✓ Point] Line Polygon

Latitude: ° ' " N

Longitude: ° ' " E

DD.DD DD MM.MM [✓ DD MM SS]

Start Time: 02/13/2012 00 : 00 :00

Simulation Length: 24 (hours) [✓ standard (Forwards Trajectory)] Backwards (Backtrack Trajectory)

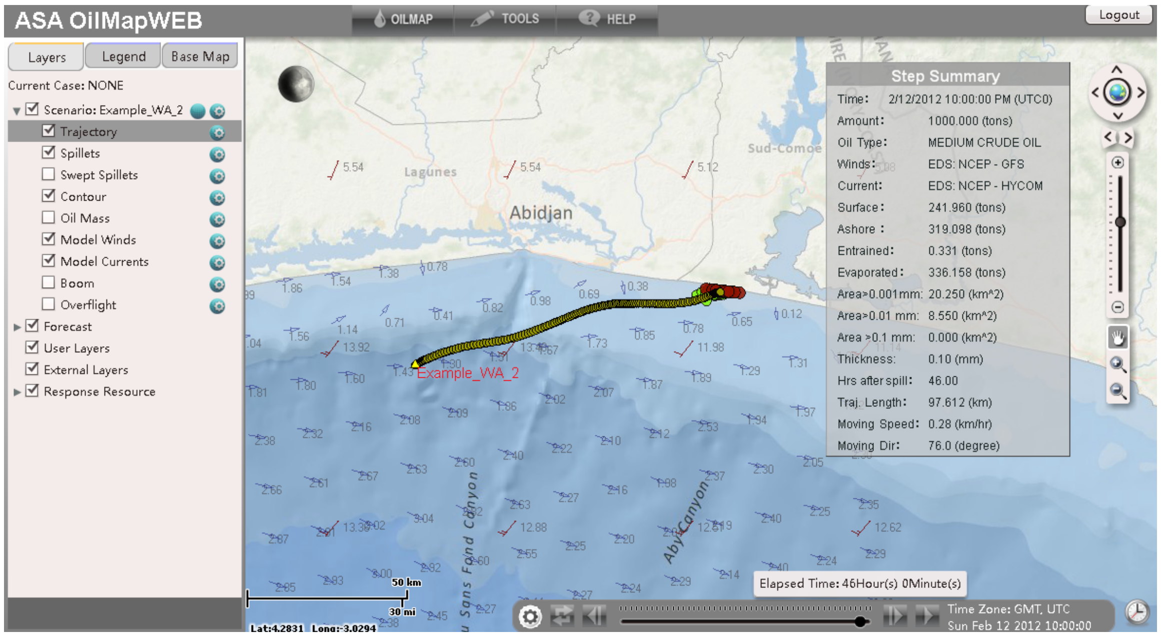
Model Time Step: 20 (min) Output Time Interval: 20 (min)

Water Temperature: 15 °C ☐ Add spillets to database

Lat:6.2467 Long:-4.7185

Time Zone: GMT, UTC
Sun Feb 12 2012 10:00:00

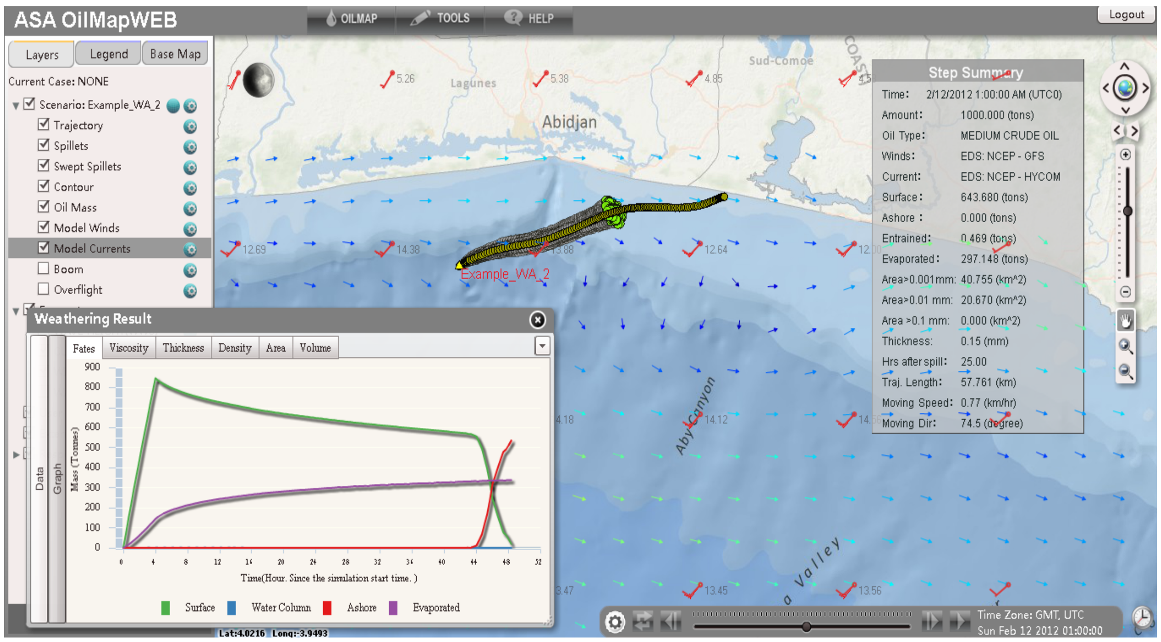
Figure 24 – OceansMap: Creating an Oil Spill scenario (1) (2)



Step Summary

Time:	2/12/2012 10:00:00 PM (UTC0)
Amount:	1000.000 (tons)
Oil Type:	MEDIUM CRUDE OIL
Winds:	EDS: NCEP - GFS
Current:	EDS: NCEP - HYCOM
Surface:	241.960 (tons)
Ashore :	319.098 (tons)
Entrained:	0.331 (tons)
Evaporated:	336.158 (tons)
Area>0.001mm:	20.250 (km^2)
Area>0.01 mm:	8.550 (km^2)
Area >0.1 mm:	0.000 (km^2)
Thickness:	0.10 (mm)
Hrs after spill:	46.00
Traj. Length:	97.612 (km)
Moving Speed:	0.28 (km/hr)
Moving Dir:	76.0 (degree)

Figure 25 - OceansMap: Viewing an Oil Spill scenario (1) (2)



Step Summary

Time:	2/12/2012 1:00:00 AM (UTC0)
Amount:	1000.000 (tons)
Oil Type:	MEDIUM CRUDE OIL
Winds:	EDS: NCEP - GFS
Current:	EDS: NCEP - HYCOM
Surface:	643.680 (tons)
Ashore :	0.000 (tons)
Entrained:	0.469 (tons)
Evaporated:	297.148 (tons)
Area>0.001mm:	40.755 (km^2)
Area>0.01 mm:	20.670 (km^2)
Area >0.1 mm:	0.000 (km^2)
Thickness:	0.15 (mm)
Hrs after spill:	25.00
Traj. Length:	57.761 (km)
Moving Speed:	0.77 (km/hr)
Moving Dir:	74.5 (degree)

Figure 26 – OceansMap: Other GIS functions available (1) (2)

(1) Courtesy of Applied Sciences Associates (ASA); (2) OceansMap is previously known as OilMapWeb

OceansMap is a standalone application stored on a Web server. Registered users can access it through the Web site *oilmapweb.com* and as of today, there is no interface available for developers to remotely consume it. A direct and full collaboration between RPS - ASA and *oystersentinel.org* is necessary in order to design an interface and implement the services available. Even though a full integration of OceansMap with *oystersentinel.org* is not possible at this time, it is important to describe and understand the ideal interface that the Oil Spill model would require.

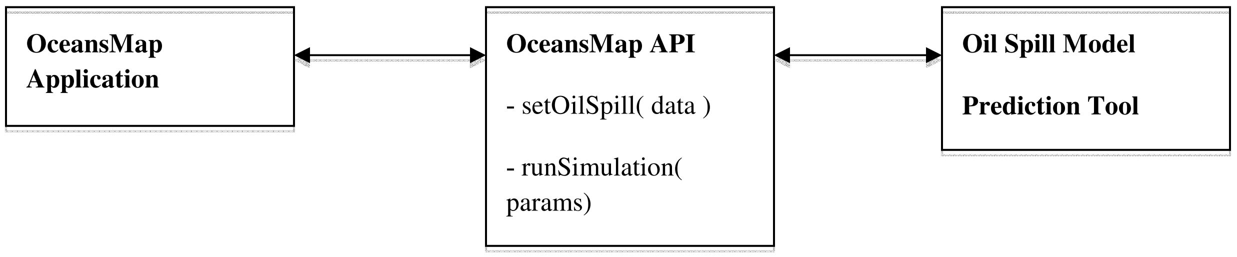


Figure 27 - OceansMap - Oil Spill Model Communication Diagram

The interface for OceansMap needs to run as a Web service and the authentication protocol is omitted for the purpose of this thesis. Using a Web service has many advantages:

- Technical specifications such as the platform, the programming languages, and the database design are all abstracted.
- Web service standards are well established. This guarantees the reliability and compatibility of the service with remote platforms.
- Web services are accessed via HTTP requests. This aspect allows developers to consume the services in any context whether it is using PHP, Java, or JavaScript. It provides flexibility when integrating the service with other applications.

From the Oil Spill model's perspective, the interface for OceansMap needs two main functions:

- setOilSpill(name, longitude, latitude, start) : When using OceansMap, the user first enters basic information about an oil spill (Figure 24) or retrieves an oil spill already saved in the system. The method "setOilSpill( )" serves that purpose. As parameters, it takes the oil spill's name, coordinates, and start date. It then returns an ID.
- runSimulation(ID, volume, length): On OceansMap, after entering information on the oil spill, the user can run a simulation by providing the volume of oil to be released and the desired length of the simulation. OceansMap will run its model and show the oil spill and its shape in interval of one and six hours (Figure 25). The method "runSimulation( )" has a similar behavior. By supplying the oil spill's id, the volume, and the length of the simulation as parameters, the function returns the polygons representing the shapes of the oil spill over time.

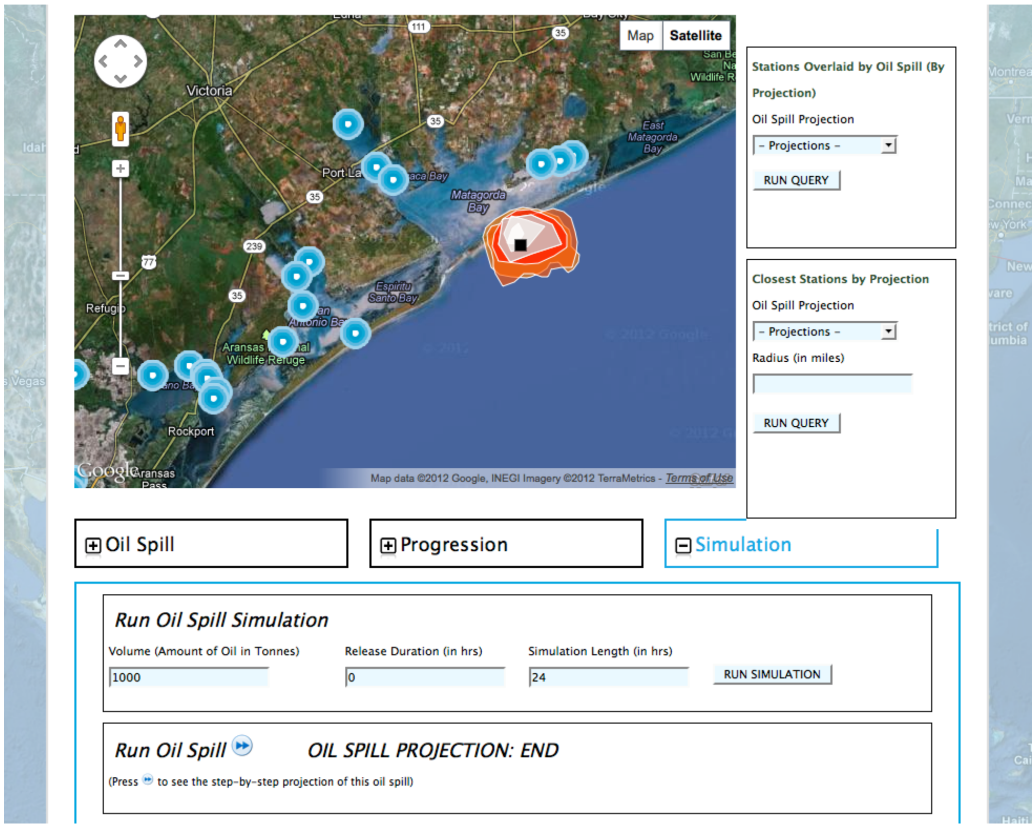
The return object of the function "runSimulation( )" should be formatted as a GML. GML is a Geography Markup Language that is similar to XML and tailored specifically for geographic data such as polygons. The following sample code illustrates a simple example of a GML Polygon element.

```
<gml:Polygon>
    <gml:exterior>
        <gml:LinearRing>
            <gml:posList>
                -89.692383 30.015836 -88.703613 29.539053 -89.450684 28.269553
-90.74707 29.271066 -90.461426 30.03486 -90.109863 30.129925 -89.692383
30.015836
            </gml:posList>
        </gml:LinearRing>
    </gml:exterior>
</gml:Polygon>
```

The prediction tool will use URLs with GET parameters to access each of these methods. Here are examples of URLs to consume OceansMap Web service:

- http://*www.oilmapweb.com*:8080/func=setOilSpill&name=oil_spill&long=-90.0&lat=30.0&start=2012-11-2
- http://*www.oilmapweb.com*:8080/func=runSimulation&vol=1000&length=24

With the model's components used in the JavaScript environment (Figure 17), it is a natural choice to consume OceansMap in JavaScript. To query the service using URLs, the Prediction tool can make Ajax calls to *oilmapweb.com* and retrieve the data in the specified format. Ajax calls are asynchronous requests sent to a server using JavaScript. The GML data received from running the simulation will be parsed and added to the map via Google Maps Polygon class.



Stations Overlaid by Oil Spill (By Projection)
Oil Spill Projection
- Projections -
RUN QUERY

Closest Stations by Projection
Oil Spill Projection
- Projections -
Radius (in miles)
RUN QUERY

Run Oil Spill Simulation
Volume (Amount of Oil in Tonnes): 1000
Release Duration (in hrs): 0
Simulation Length (in hrs): 24
RUN SIMULATION

Run Oil Spill *OIL SPILL PROJECTION: END*
(Press to see the step-by-step projection of this oil spill)

Figure 28 - Oil Spill Prediction tool

Figure 28 illustrates a demonstration of the Prediction tool. After running the simulation, the Oil Spill model shows projected paths with orange and red colors, in contrast to the white and grey colors used for actual paths. Another feature available in the Prediction tool is the projection viewer, similar to the Oil Spill viewer described in section 4.3.3. The viewer helps users visualize oil spill projections in chronological order or visualize each projection at the time.

4.3.5 Query Tool

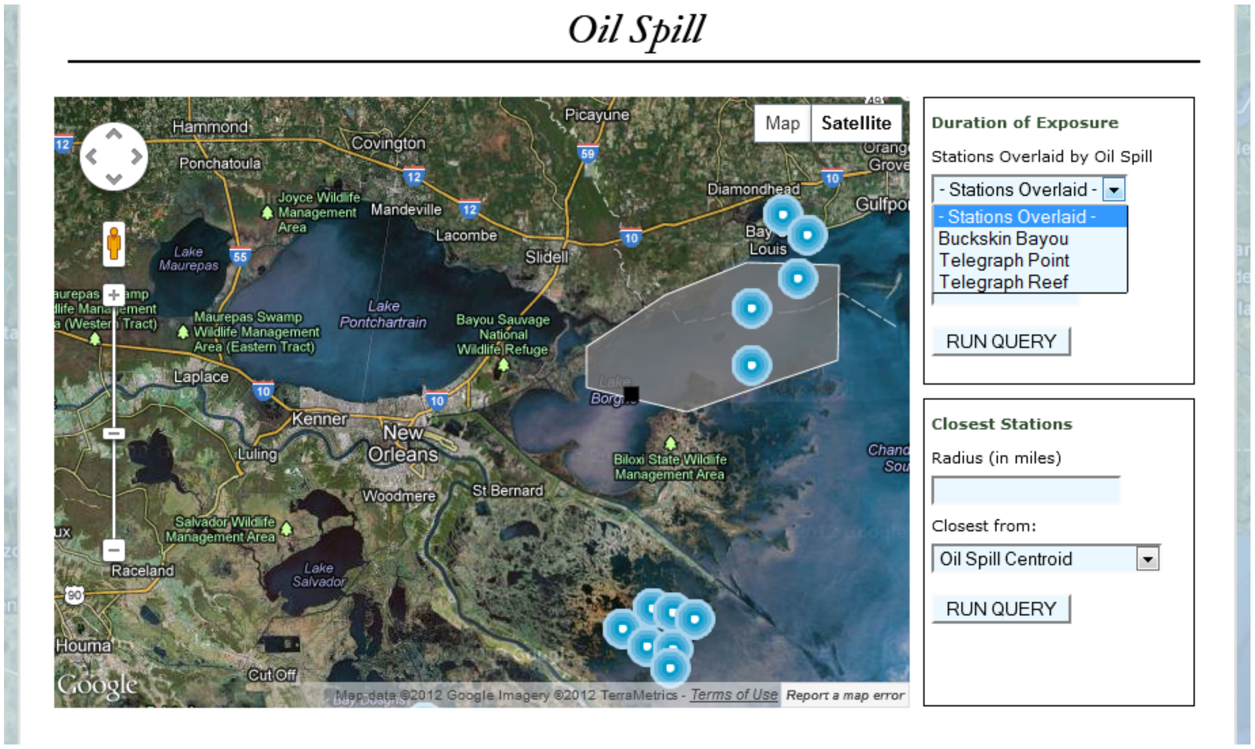
The Query tool is another important component of the Oil Spill model. After implementing the Oil Spill manager and the Prediction tool, administrators can manage oil spills while users can view them. The Query tool is what makes this model valuable. It is a collection of queries that users can use to check the stations that are covered by the oil spill or the ones that are within a certain radius from the oil spill. Domain experts and marine biologists can use this Query tool to prioritize their efforts in the event of an oil spill. With the Prediction tool in place, the Query tool can also estimate, for example, the stations that might be covered by an oil spill within ten hours or the stations that will be within a one-mile radius from an oil spill in ten hours. These queries add more weight in reducing response time and allowing marine wildlife and resources to be better protected.

The Query tool is implemented in JavaScript. Using data from any component in the model is done with fewer overheads and it reduces the server's processing power by running all the queries on the users's computer.

With the stations and oil spills loaded from our internal database, we can find the stations that are overlaid by the oil spill. Google Maps provides the method 'google.maps.geometry.poly.containsLocation(coord, poly)'. It takes two parameters: the coordinates of a marker/point and a polygon. It returns true if the marker/point is within the surface of the polygon. Using an iterative loop, we can enumerate all the stations and retrieve the ones that are located within the surface of an oil spill (see code snippet below).

```
<?php foreach($stations as $j => $station){ ?>

    coord = new google.maps.LatLng(<?php echo $station['latitude']; ?>, <?php
echo $station['longitude']; ?>);
    var overlaid = new Array();
    var d = 0;
    for(var i=0; i<polygones.length; i++){
        if(google.maps.geometry.poly.containsLocation(coord, polygones[i])){
           overlaid[i][d] = <?php echo $station['id']; ?>;
           d++;
        }
    }
<?php } ?>
```

After running the simulation, the same Google Maps method is used to find the stations that might be overlaid by a projection. Figure 29 is a screenshot demonstrating the query and showing the stations that are currently overlaid by an oil spill.



Duration of Exposure

Stations Overlaid by Oil Spill

- Stations Overlaid -
- Stations Overlaid -
Buckskin Bayou
Telegraph Point
Telegraph Reef

RUN QUERY

Closest Stations

Radius (in miles)

Closest from:

Oil Spill Centroid

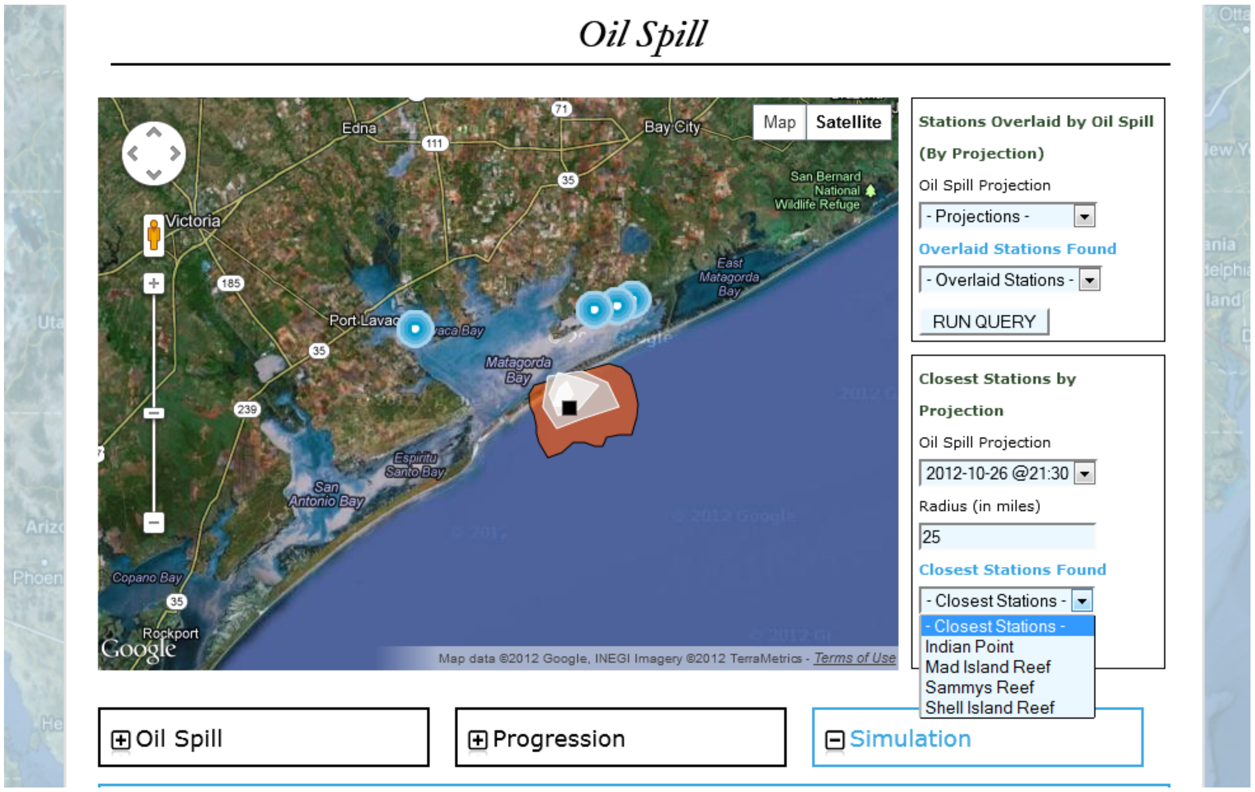
RUN QUERY

Figure 29 – Stations currently overlaid by the oil spill

To filter the stations that are within a certain distance from the oil spill, we used the Google Maps method 'google.maps.geometry.spherical.computeDistanceBetween (coord1, coord2)'. Taking in count the rounded shape of Earth, this method takes the coordinates of two shapes as parameters and returns the distance between them. Using a loop, we can enumerate all the stations and calculate the distance between them and the oil spill. If the difference is within the radius entered in the query, then the station is added to the list (see code snippet below).

```
jQuery('#q1_run').click(function(){
   // Latest shape of oil spill
   var last = polygones.length - 1;
   // Distance specified in the query
   var distance = jQuery('#q1_distance').val() * 1609.344;

   // Stations within distance are added to dropdown list
   var closest = '<option value="-1">- Closest Stations -</option>';
   polygones[last].setVisible(true);

   // Get center of latest oil spill coordinates
   var polyCenter = getBounds(polygones[last]).getCenter();

   // Loop through each station
   for(var i = 0; i < stations.length; i++){
        if(google.maps.geometry.poly.containsLocation(stations[i][3],
polygones[last]) ||
google.maps.geometry.spherical.computeDistanceBetween(polyCenter,
stations[i][3]) <= distance){

          // Show station that is within distance
          stations_markers[i].setVisible(true);
           // Append station to dropdown list
          closest += '<option value="'+ i +'">'+ stations[i][0] +'</option>';
        }
   }
   map.setCenter(polyCenter);
   jQuery('#q1_closest').empty().append(closest);
   jQuery('#q1_response').show();
   map.setZoom(10);
});
```

Figure 30 is a screenshot demonstrating the query and showing the stations that will be within twenty-five miles from the oil spill.



Stations Overlaid by Oil Spill (By Projection)

Oil Spill Projection: - Projections -

Overlaid Stations Found: - Overlaid Stations -

RUN QUERY

Closest Stations by Projection

Oil Spill Projection: 2012-10-26 @21:30

Radius (in miles): 25

Closest Stations Found: - Closest Stations -

- Closest Stations -
Indian Point
Mad Island Reef
Sammys Reef
Shell Island Reef

Figure 30 – Stations within twenty-five miles of oil spill projection

The Oil Spill model is a more complex Web mashup than the *Perkinsus marinus* model. In addition to using a rich map from Google Maps, it fetches data from a local database and uses a prediction service (i.e. OceansMap) to create a comprehensive application that can have a very beneficial role in protecting the oyster resources of the Gulf of Mexico.

<u>Disclaimer</u>: Figures 20, 22, 23, 28, 29, and 30 and the information associated do not represent actual oil spills. Data used in the Oil Spill model were made up to test and demonstrate its capabilities.

4.4 Fisheries Web-based service

Data reusability is the primary goal of Web mashups. It gives developers the ability to reuse existing data from external resources. However, a large percentage of Web sites provide data on their platform but do not implement interfaces for developers to access those data. One example of this limitation is the Oil Spill model currently unable to use the service OceansMap as described in section 4.3.4. The Fisheries Web-based service is designed solely for the purpose of demonstrating programmatic interfaces and the methods to access them externally. It is not a live interface.

In addition to monitoring the health of oysters in the Gulf coast, *oystersentinel.org* also monitors the quantity of oysters by type at stock assessment stations in Louisiana. These data are collected in stock assessments by Coastal Study Area (CSA) managers and stored in a database. Managing these data helps visualize the increase or decrease of a type of oysters in any given station. Outside developers will use the Fisheries API to query our database and extract information on oysters and stations.

From its design, the Fisheries Web-based service will serve as a bridge between outside developers and our internal database. Developers will have two functions available to retrieve the list of stations in the Gulf of Mexico and the quantity of sack oysters in each station. Figure 31 describes how the Fisheries interface interacts with the other components of the system.

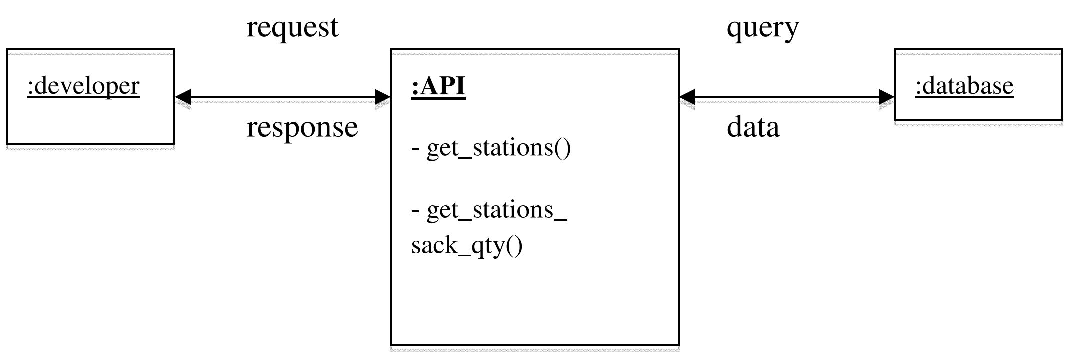


Figure 31 - Fisheries API Abstract Communication diagram

The database used to store oyster data is the MSSQL database used in the previous models. CSA managers collect samples from stations and separate seed, sack, and spat oysters. From the samples, they store the average quantity of each type of oyster in the given sample area.

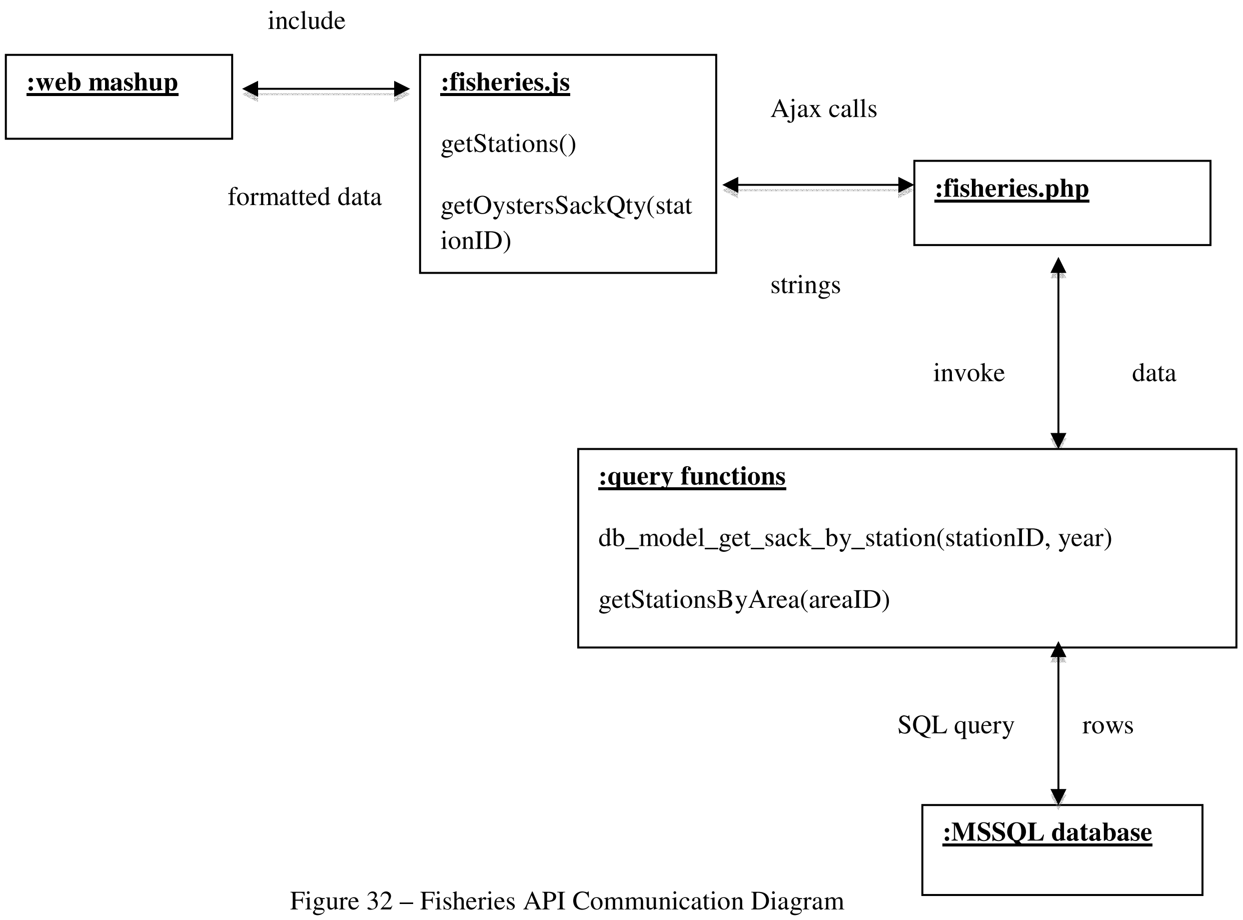


Figure 32 – Fisheries API Communication Diagram

4.4.1 Query functions

Data are stored in relational tables and to query the database. We implemented two query functions:

- getStationsByArea(areaID): It is implemented in the *Perkinsus marinus* model (Figure 9) for the MVC model for stations (section 4.2, figure 11). The function takes an area ID as the parameter and returns the stations located in that area. When passed 0 as the area ID, the function returns all the stations in the table (see code snippet below).

```
  function getStationsByArea($area_id = 0){
     if($area_id == 0){
       $query = "SELECT * FROM MAP_STATIONS ORDER BY name";
       $params = array();
     } else {
       $query = "SELECT * FROM MAP_STATIONS WHERE area_id = ? ORDER BY name";
       $params = array(0 => array('value' => $area_id, 'type' =>
PDO::PARAM_INT));
     }
     $result = runSQL($query, $params);
     if(isset($result['error'])) $return = array();
     else   $return = $result;

     return $return;
  }
```

- db_model_get_sack_by_station(): Originally implemented for the Oyster Sustainability model, this function fetches the average of sack oysters in an area during a certain year. It takes as parameters a station ID and a year, and returns an associative array of station ID and average quantity.

The query functions form the first layer of abstraction needed to hide our database internal structure to the outside world. The second layer is the file 'fisheries.php'.

4.4.2 fisheries.php

This file, located in the /webservices folder, handles API requests on the server side. It checks the GET variables passed with the request, invokes the two query functions implemented in section 4.4.1, and returns data as a string (see code snippet below).

```
<?php
      date_default_timezone_set('America/Chicago');

      // Query 1 - Getting list of stations
      if(isset($_GET['query']) && $_GET['query'] == 'stations'){
            //Get lists of stations
            include '../models/station.php';
            $stations = getStationsByArea(0);

            foreach($stations as $i => $s){
                  $response[] =
$s['id'].','.$s['name'].','.$s['latitude'].','.$s['longitude'];
            }
            print_r(implode(';', $response));

      // Query 2 - Getting sack oysters average quantity by station
      } elseif(isset($_GET['query']) && $_GET['query'] == 'qty') {
            include '../models/station.php';

            //Get sack oysters qty of a station
            $tmp = db_model_get_sack_by_station(2, 2012);
            foreach($tmp as $t){
                  $oysters[] = $t;
            }
            $stations = getStationsByArea(0);
            foreach($stations as $i => $s){
                  if($s['id'] == $_GET['stationID']){
                   $response = (isset($oysters[$i])) ? $oysters[$i] + 15 : 25;
                   break;
                  }
            }
            print_r($response);
      }
?>
```

This PHP function is not directly available to outside developers. We implemented a third layer of abstraction to represent the public interface.

4.4.3 fisheries.js

Following Google Maps API design, the Fisheries API will be accessible via the file 'fisheries.js' located in the /webservices folder. When included, two operations are made available for use (see code snippet below): get the list of stations and fetch the average quantity of sack oysters given a station.

```
function getStations(){
    var stations = new Array();
    jQuery.ajax({
          type: 'GET',
          url: 'http://www.oystersentinel.org/webservices/fisheries.php',
          data: { query: 'stations'},
          dataType: "text",
          async: false,
          success: function(data){
                var temp = data.toString().split(";");
                for(var i=0; i<temp.length; i++){
                      stations[i] = temp[i].toString().split(",");
                }
          }
    });
    return stations;
 }

 function getOystersQty(stationID){
    var density = -1;
    jQuery.ajax({
          type: 'GET',
          url: 'http://www.oystersentinel.org/webservices/fisheries.php',
          data: { query: 'qty', stationID: stationID},
          dataType: "text",
          async: false,
          success: function(data){
                density = data;
          }
    });
    return density;
 }
```

Each JavaScript function makes Ajax calls to the file 'fisheries.php'. However, the call is asynchronous, meaning that the function will not wait for a response before continuing. To work around that limitation, jQuery provides the parameter <async> which, when set to false, waits for the server's response. We also use the GET variables: <query> to specify the type of request, and <stationID> to specify the station. The function getStations() does not take any parameter. It makes an Ajax call to 'fisheries.php' and receive a string back from the server. It formats it and returns an array of stations. The function getOystersSackQty (stationID) takes a station ID as its only parameter. After receiving the reply from the Ajax call, it simply returns the density as a string. By including the script file 'fisheries.js' in any application, an outside developer is able to consume the Fisheries service and create a Web mashup. To demonstrate this Web-based service and the JavaScript API, we built two simple Web mashups.

Disclaimer: Due to the sensitive nature of oyster data, these Web mashups use dummy data. The stations used in these examples do not represent actual stock assessments stations. The average quantities of sack oysters are from the year 2009 and are disassociated from their original stations. However, these examples support the primary intent of experimenting Web mashup interfaces.

4.4.4 Examples of Web mashups using the Fisheries API

The first Web mashup is a Web page that displays the average quantity of sack oysters by using a heat map (Figure 33). To create this page, we use the Google Maps and the Fisheries APIs (see code snippet below). Google Maps provides the map and methods to create the visualization layer for heat maps. The Fisheries API provides the stations and the densities. This Web page retrieves data from two external data sources and displays them in a new content, which makes it a Web mashup.

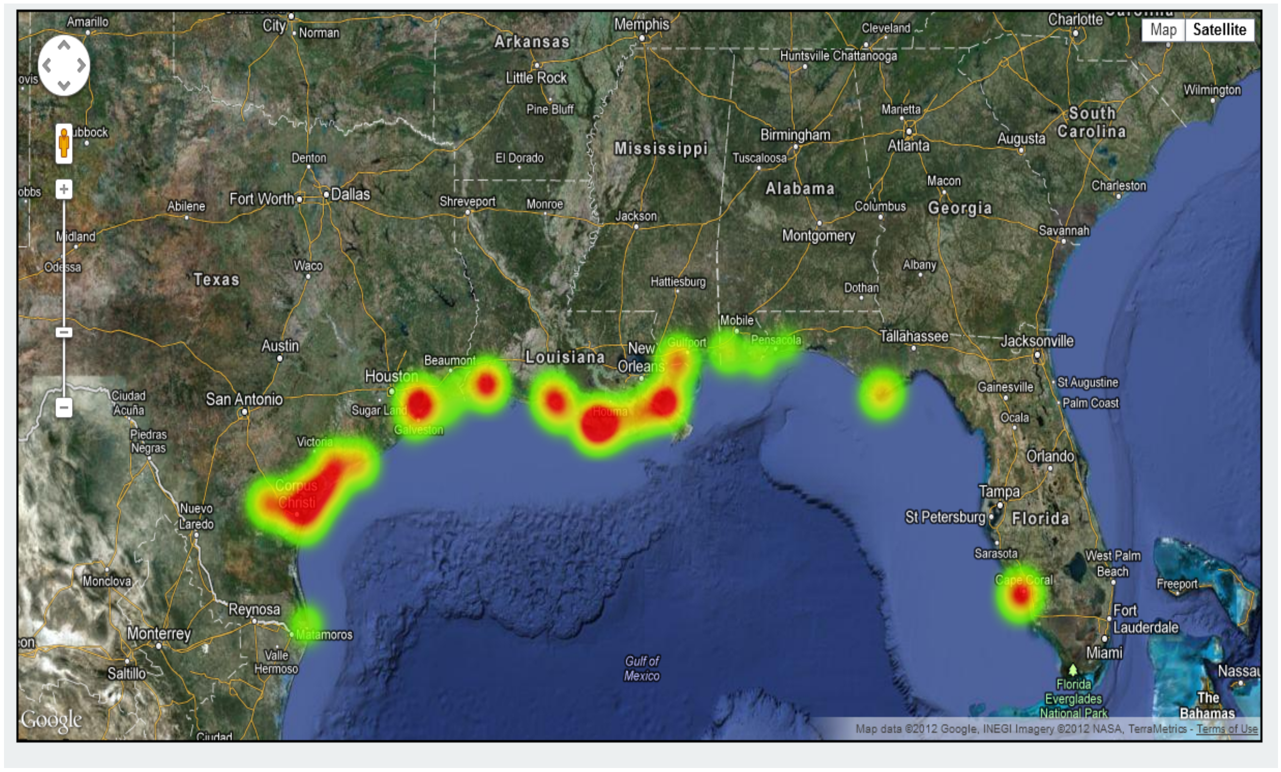


Figure 33 – Web mashup Example #1 [(3)]

The code below is the HTML/JavaScript code used to implement this Web Mashup. An outside developer first needs to include the Google Maps API and the <fisheries.js> file. Google Maps provides the classes and methods to add a map to the Web page and produce a heat map using Google Maps' visualization library. By using the average quantity of sack oysters as the weight for each station, it is possible to assign a heat index (color) to each location and create a heat map calling the method 'google.maps.visualization.HeatmapLayer( )'. The fisheries interface provides the functions implemented in section 4.4.3, allowing the developer to get the list of stations and the average quantity of sack oysters per station.

(3) Not actual average quantity of sack oysters

```
<html xmlns="http://www.w3.org/1999/xhtml">
   <head><title>Web Mashup Example #1</title>
       <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
       <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery
.min.js"></script>
       <script type="text/javascript"
src="http://137.30.149.37/webservices/fisheries.js"></script>
       <script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=false&v=3&
libraries=visualization,drawing"></script>
   </head>

   <body>
       <div id="map"
style="position:relative;height:600px;width:1200px;margin:10px auto 25px
auto;border:solid 2px; #000000;">
       </div>
      <script type="text/javascript">
        jQuery(document).ready(function(){
         var myLatlng = new google.maps.LatLng(29.951066, -90.071532);
         var myOptions = {
            zoom: 6,
            center: myLatlng,
            mapTypeId: google.maps.MapTypeId.HYBRID };
         var map = new google.maps.Map(document.getElementById("map"),
myOptions);
         var stations = getStations();
         var heatData = new Array();
         for(var i=0; i<stations.length; i++){
          heatData[i] = {
            location: new google.maps.LatLng(stations[i][2], stations[i][3]),
            weight: parseInt(getOystersQty(stations[i][0]), 10)};
          }
         var heatmap = new google.maps.visualization.HeatmapLayer({
            data: heatData,
            radius: 30,
            map: map,
            opacity: 0.7});

      });
        </script>
   </body>
</html>
```

The second Web mashup is a more complicated application. The Web application shows a map of oyster stations along the Gulf coast. It also allows users to draw rectangles and circles on that map and it displays the average quantity of sack oysters in the area selected, as well as the stations included in that area. This is another way to utilize our Fisheries API with Google Maps API. The code snippet below is the JavaScript code used to consume the Fisheries API.

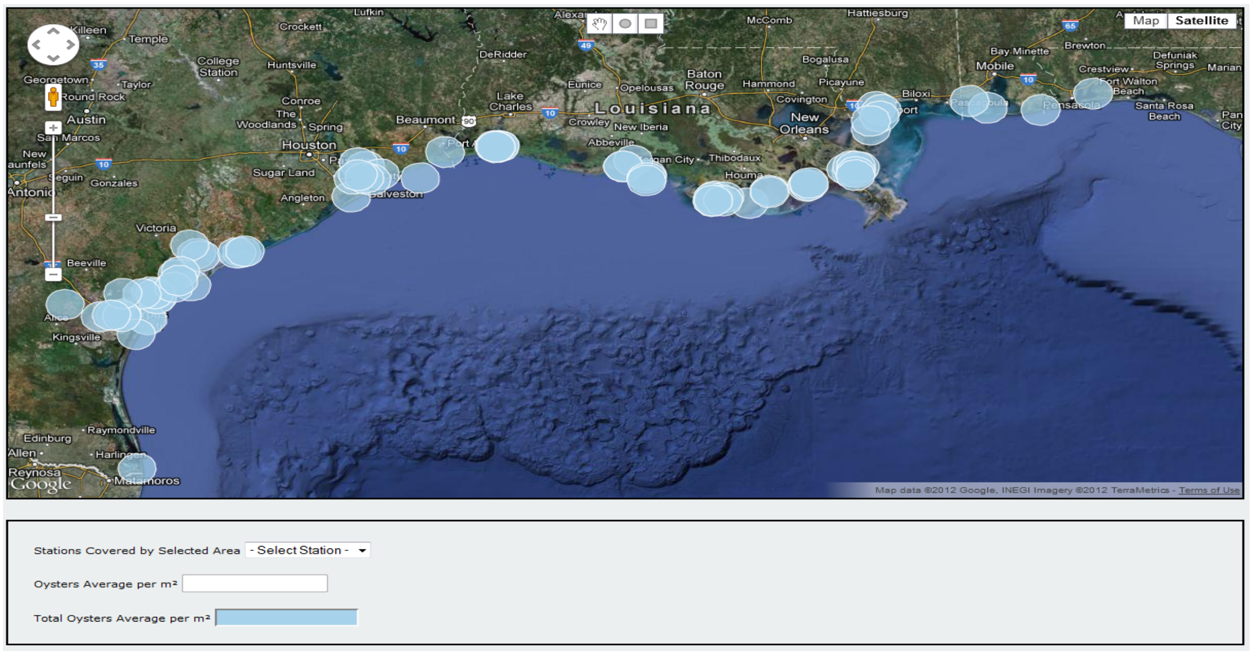
```
 var stations = getStations();
 var circles = new Array();
 for(var i=0; i<stations.length; i++){
      circles[i] = new google.maps.Circle({
         center: new google.maps.LatLng(stations[i][2], stations[i][3]),
         radius:20000,
         fillColor:'#A6D2EB',
         fillOpacity:0.7,
         strokeColor:'#FFFFFF',
         strokeWeight:1,
         strokeOpacity:1.0,
         visible:true,
         zIndex: 100});
      circles[i].setMap(map);
   }

  function fetchQty(circles, stations, bounds){
      var qty;    var total = 0; var count = 0;
      for(var i=0; i<circles.length; i++){
          if(bounds.intersects(circles[i].getBounds())){
            qty = getOystersQty(stations[i][0]);
            total += parseInt(qty,10);
            count++;
            jQuery('#stations').append('<option value="'+ qty +'">'+
stations[i][1] +'</option>');
          }
      }
      jQuery('#qtyt').val(total/count);
  }
```

Google Maps API provides a drawing library to allow users to draw shapes such as rectangles and circles on the map (see code snippet on next page).

```
var drawingManager = new google.maps.drawing.DrawingManager({
                          drawingMode:google.maps.drawing.OverlayType.MARKER,
                          drawingControl: true,
                          drawingControlOptions: {
                            position: google.maps.ControlPosition.TOP_CENTER,
                            drawingModes: [
                                google.maps.drawing.OverlayType.CIRCLE,
                                google.maps.drawing.OverlayType.RECTANGLE
                            ] },
                          circleOptions: {
                            fillColor: '#ffff00',
                            fillOpacity: 0.4,
                            strokeWeight: 1,
                            clickable: false,
                            zIndex: 1
                          }});
drawingManager.setMap(map);
```

Once the shape is completed, we implemented an event handler that will retrieve the stations contained in the shape and fetch their density (see code snippet).

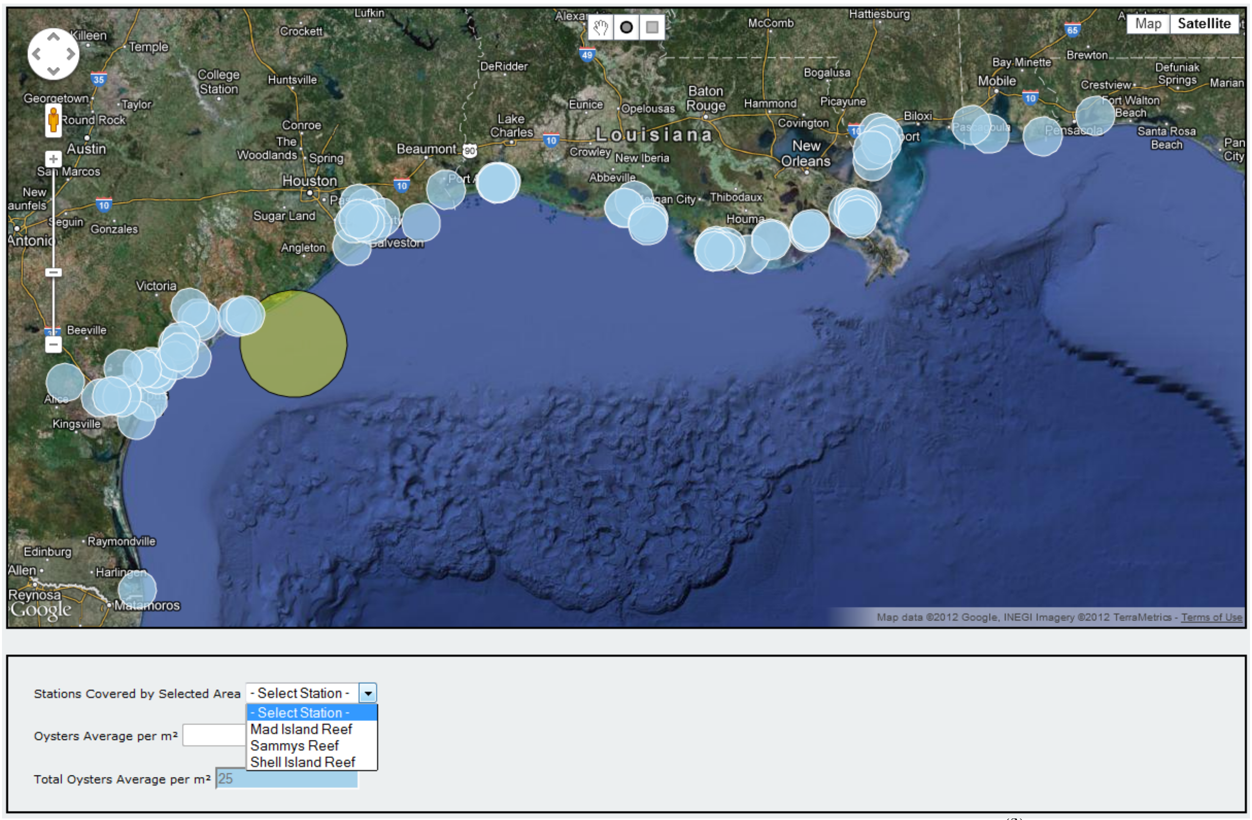
```
var currentShape = null;
google.maps.event.addListener(drawingManager, 'circlecomplete',
      function(circle) {
         if(currentShape != null)
            currentShape.setVisible(false);
         currentShape = circle;
         var bounds = circle.getBounds();
         fetchQty(station_markers, stations, bounds);
      }
);

google.maps.event.addListener(drawingManager, 'rectanglecomplete',
      function(rectangle) {
         if(currentShape != null)
            currentShape.setVisible(false);
         currentShape = rectangle;
         var bounds = rectangle.getBounds();
         fetchQty(station_markers, stations, bounds);
      }
);
```


Stations Covered by Selected Area - Select Station -

Oysters Average per m²

Total Oysters Average per m²

Figure 34 – Web Mashup Example #2


Stations Covered by Selected Area - Select Station -

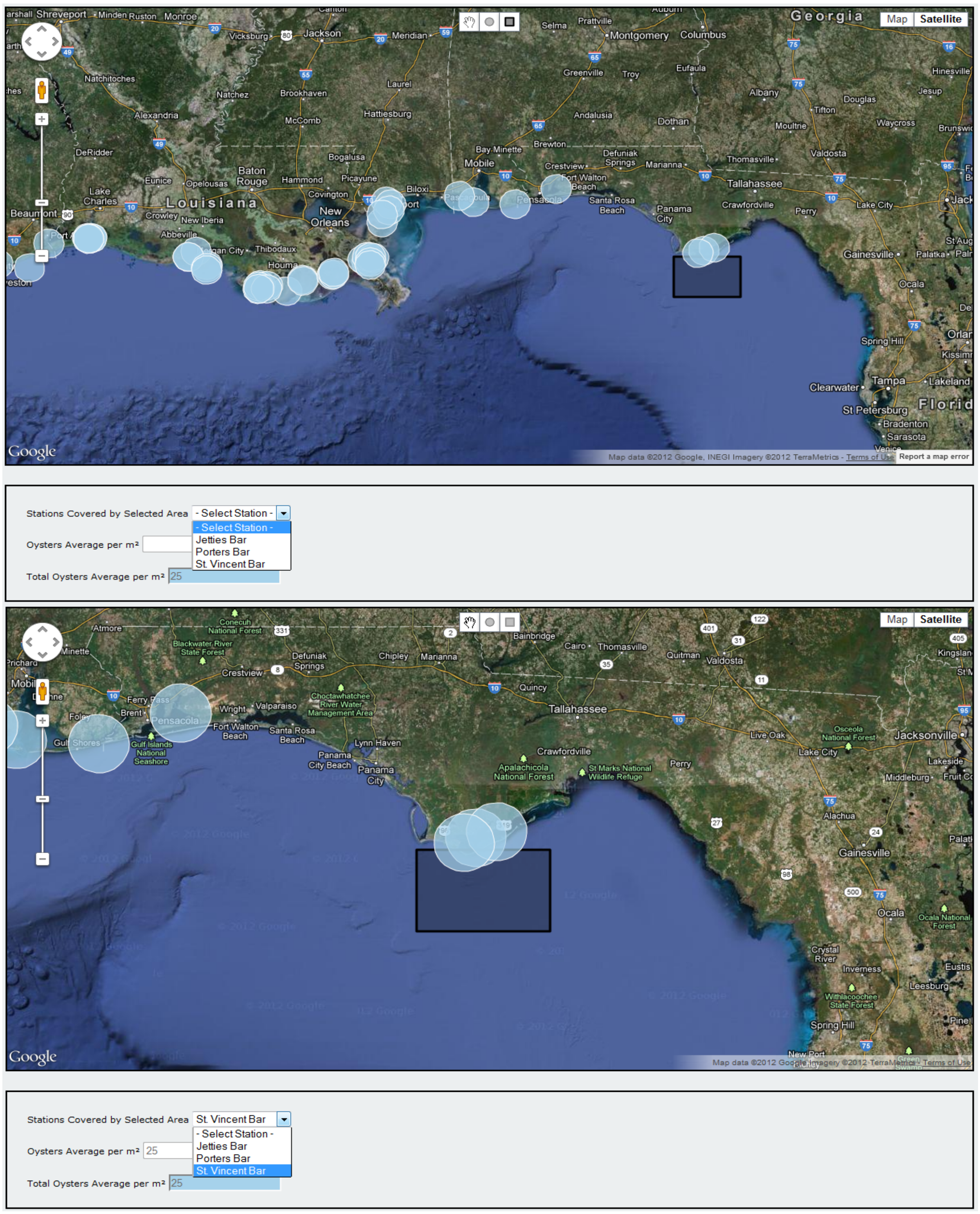
- Select Station -
Mad Island Reef
Sammys Reef
Shell Island Reef

Oysters Average per m²

Total Oysters Average per m² 25

Figure 35 – User selects an area (circle) and gets data on closest stations [(3)]

[(3)] Not actual average quantity of sack oysters



Stations Covered by Selected Area: - Select Station -
- Select Station -
Jetties Bar
Porters Bar
St. Vincent Bar
Oysters Average per m²:
Total Oysters Average per m²: 25

Stations Covered by Selected Area: St. Vincent Bar
- Select Station -
Jetties Bar
Porters Bar
St. Vincent Bar
Oysters Average per m²: 25
Total Oysters Average per m²: 25

Figure 36 - User selects an area (rectangle) and gets data on closest stations (3)

(3) Not actual average quantity of sack oysters

Chapter 5: **Conclusion**

In this thesis, *oystersentinel.org* uses Web Mashup technology to present raw data on oysters to interested users with a great level of usability and intuition. Integrating the OceansMap service into the Web site would enhance the capabilities of domain experts to handle oil spill disasters. In the event of another oil spill occurring in the Gulf of Mexico or anywhere else, scientists would share a strong interest in using a comprehensive application such as the Oil Spill model. We have also implemented a Web Mashup API that supports third-party developers to use the data available at *oystersentinel.org*.

The advantages of the Web Mashup technology to our development are:

1. We can provide map-rich user interfaces without a map server.
2. We can reuse existing and reliable applications.
3. We can provide outside developers an API to access internal data without directly interacting with the database.

Web Mashup technology also comes with some disadvantages such as security, authentication, and scalability. The standards have not yet been set on how to authenticate a user, and developers must rely on the external resources to remain available and scalable. Even though it is not a complete domain, Web Mashup technology allows rapid application development at a reduced cost.

References/Bibliography

Applied Sciences Associates (ASA). 2012. Welcome to OceansMap. [Online] Available at: http://www.oilmapweb.com [Accessed 21 October 2012]

Booth David, Haas Hugo, McCabe Francis, Newcomer Eric, Champion Michael, Ferris Chris, Orchard David. 2004. Web Services Architecture. [Online] (Updated 11 February 2004) Available at: http://www.w3.org/TR/ws-arch/ [Accessed 15 August 2012]

Drupal. 2012. About Drupal. [Online] Available at: http://drupal.org/about [Accessed 25 October 2012]

Google. 2012. Google Maps Developer Documentation. [Online] (Updated 12 October 2012) Available at: https://developers.google.com/maps/documentation/ [Accessed 22 October 2012]

Louisiana Department of Wildlife & Fisheries (LDWF). 2009. Oyster stock assessment report of the public oyster areas in Louisiana seed grounds and seed reservations. Louisiana Department of Wildlife and Fisheries. Baton Rouge, LA. Pages i - x

Merrill, Duane. 2006. Mashups: The new breed of Web app. [Online] (Updated 24 July 2009) Available at: http://www.ibm.com/developerworks/xml/library/x-mashups/index.html [Accessed 21 August 2012]

Ogrinz, Michael. 2009. Mashup Patterns: Designs and Examples for the Modern Enterprise. Understanding Mashup Patterns. Addison-Wesley Professional. Chapter 1, 1 - 51

The jQuery Foundation. 2012. jQuery – Documentation. [Online] Available at: http://docs.jquery.com/ [Accessed: 25 October 2012]

Soniat, Thomas M., Sarah M. King, Matthew A. Tarr, Megan A. Thorne. 2011. Chemical and physiological measures on oysters (*Crassostrea virginica*) from oil-exposed sites in Louisiana. Journal of Shellfish Research, Vol. 30, No 3, 713-717

Rouse Margaret. 2011. Content Management System. [Online] (Updated January 2011) Available at: http://searchsoa.techtarget.com/definition/content-management-system [Accessed 30 October 2012]

Vita

The author was born in Yaounde, Cameroon. He obtained his Bachelor’s Degree in computer science from the University of New Orleans in 2010. After a year of professional work, he came back to the University of New Orleans to join the Computer Science graduate program to pursue a Masters in database systems and distributed applications. He served as a graduate assistant and worked on the Oyster Sentinel project under the supervision of Dr Abdelguerfi and Dr Tu from the computer science department, and Dr Soniat from the biology department.