2009

Online Circular Calendar

Praveen Athmanathan Panneerselvam
San Jose State University

Online Circular Calendar

A Writing Project
Presented to
The Faculty of the Department of Computer Science
San José State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
by

Athmanathan Panneerselvam, Praveen

Fall 2009

Abstract

A calendar is a system to organize days for social, commercial or administrative purpose. Many calendar systems are available today. The calendar system helps the user in scheduling his/her events or tasks over a time period. This period may be an hour, a day, or even months. Due to increase in user's activities, events that need to be scheduled in the calendar grow tremendously. Moreover, there are events that occur every year which require a good visualization for mental manipulation. As a result there is a difficulty in organizing these events in the current calendar system. The main idea of this project is to provide a calendar system in which users can organize the events easily, and to close the gap between the actual software and the mental model of the users. None of the current calendar systems have the ability to manipulate and plot graphs throughout the year. The data is user dependent and can be of any sort like temperature, rainfall, stock analysis etc., Apart from this; good visualization techniques can be used for the calendar system to make the events apparent to the users. By this way user can view the overall picture of the events and will have clear idea about their events. This paper describes the implementation of such a calendar system with good visualization.

Contents

List of figures

1 Introduction

There are many classifications in calendar system. Solar calendar, lunar calendars, arithmetic and astronomical calendars, Chinese, Hebrew, Hindu, Julian, Ethiopian, Thai solar, Buddhist, Gregorian and fiscal are the few calendars that are commonly known. Out of these calendar systems Gregorian is widely used. When these calendars are programmed in computer it is known as electronic calendars. These electronic calendars are also called Calendaring Software. Such software enables users to control and maintain their schedules and activities electronically. Calendar software is further classified in to online and offline. Offline calendar application needs to be installed in a computer like any other software and the calendar cannot be accessed outside that computer. This drawback led to the online calendars. Online calendars are prominent nowadays. Main advantage of the online calendar is its accessibility. It can be accessed from any place through internet. While there are hundreds of online calendar systems available, the calendar from Google, Yahoo and MSN dominates other online calendar systems.

This document reflects the work done for the project "Online Calendar System" during the course work "Writing Project – CS 298." In this system, calendar is projected in circular manner to make good visualization of the calendar system as opposed to the traditional linear calendar visualization. Calendars are mainly used for managing events and tasks. When such events occur every year, circular calendar is the best way to represent the repeated events. Keeping the calendar rotatable lets the users to visualize the event patterns when they start with different months than January. This system integrates the calendar events with the weather information. Weather information like temperature and rainfall data can be plotted over the calendar. Thus providing the interface as a single point to manipulate all kinds of calendar data the user might have. Not only events, data which are similar to events or temperature information can be fed to this interface to visualize that information.

All of the current online systems are similar in their functionality and features. The drawbacks of these calendars are: does not target all kinds of users, and no visualization for events that occur every year.

2 Project Architecture

2.1 3-Tier Architecture

3-Tier model is incorporated for this application to support scalability and flexibility. Data Tier handles database, and manages the persistent data. Application encapsulates the business logic. Client tier acts as the interface between user and application tier. Application tier is split in to two layers: layer 1 and layer2. Layer 1 interacts with the database and provides services to layer2. Layer2 utilizes layer1 and in turn provides its services to client tier. So the communications are occurred between client tier and layer1, layer1 and layer2, layer2 and database.

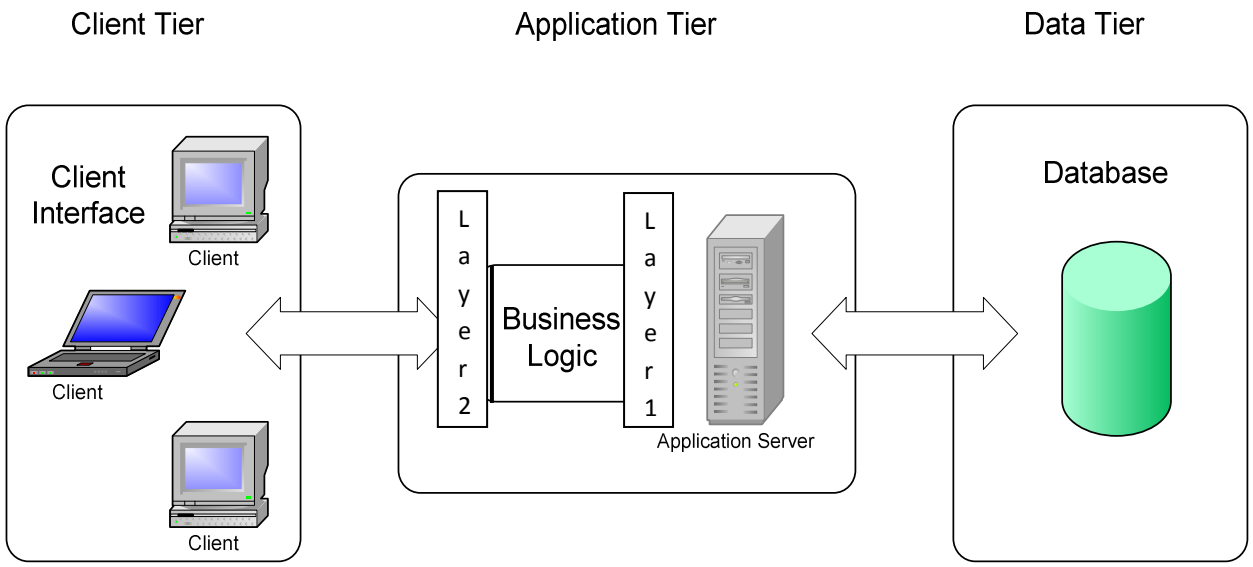


Figure 1: Architecture

The application layer is split in to two layers to provide good scalability. The segregation of the layers allows the total client interface to be replaced without doing any modifications in layer 2. All the abstract functions needed to interact with the database are written in Layer 1 and the components related with the calendar interface are implemented in layer 2. This separation of layers not only allows scalability but also supports security, durability and data abstraction.

This project contains both server side and client side components that lie in their respective tiers. The load is spread eventually on both sides. The client side components visualize the user

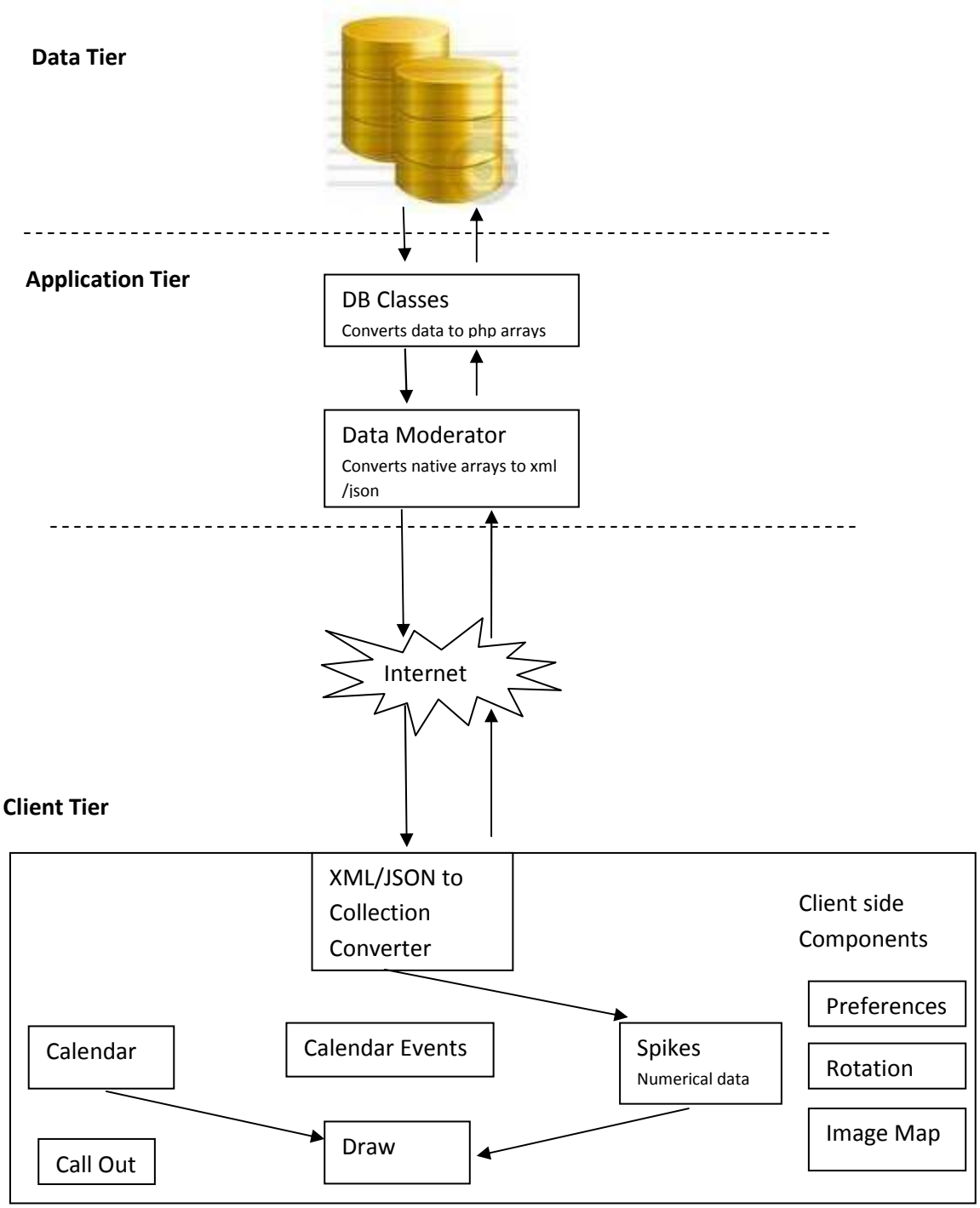


Figure 2 Project Architecture

data over the calendar and also let the user to manage their data. These components run in flash inside the browsers and are responsible for communicating with server side components. Server side components answer client queries by analyzing the data from the database. The detailed description of these components follows.

2.2 Server Side Components

Server side components lie in application tier and is further divided in to two layers. DB Classes are responsible for fetching the data from the database and acts as a driver for Data moderator to pull or push the data from or to the database. This layer is highly coupled with database and all the transactions related with the database must pass through this class. DB classes take care of creating and destroying connections to the database. It is designed in such a way that each instance of the DB class has its own connection. An instance of the DB class is created for each request and is kept alive until the response for that request is dispatched.

Data moderator acts as an interface for the calls made by client side components. All the requests to get data for the client side components must seek Data moderator. Data moderator analyzes the HTTPService call from client side components to determine the data it needs to take from database using DB classes. It then fetches the actual data through DB class, process the data and converts it to the PHP arrays. Finally, the arrays are encoded to JSON or XML based on the client component's request.

2.3 Client Side Components

All of these components are packed in a SWF file and sent to client browsers initially. Each component in the package independently interacts with the Data moderator at the server. However on the way back to its response, every component needs XML/JSON to collection converter. Since all the responses from the server are either XML or JSON, a pass through the converter is mandatory. The converter converts the XML or JSON to the flash native array collections. Once the data is converted to the array of arrays, it is handed over to their individual component to process the result.

2.3.1 Calendar

This is the first component called after the successful user authentication. It creates the basic calendar visualization. Twelve months in the year are plotted as sectors of two concentric circles. Each month is an instance of a polygon. At first the co-ordinates for each month are computed and stored in their appropriate polygon objects. The co-ordinates are computed by drawing two concentric circles, one is inner and the other is outer.

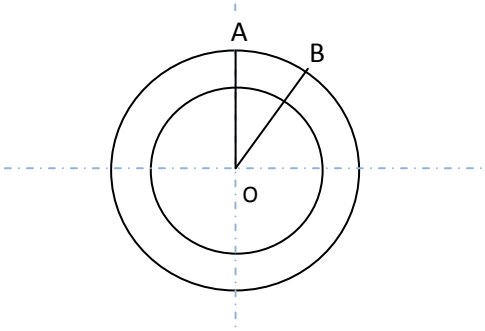
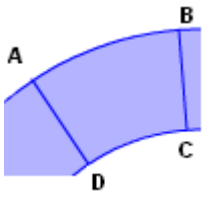
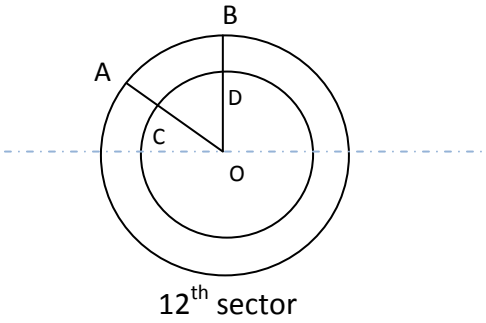


Figure 3: Sector

The imaginary radius is drawn from the center point to the outer circle at an angle of 0 degree. Similarly another imaginary radius is drawn at an angle of 30 degrees. The four points where these two radius lines intersect the two circles are stored as border co-ordinates of a polygon. Shift up the angle by 30 degrees and do the same process again to find the border co-ordinates for the next polygon. Similarly repeat the process until the angle reaches 360 degree. Border coordinates for twelve polygons are computed at the end of this iteration.

Computing Co-ordinates for 12th sector(Month: December)





Quadrants in Flash

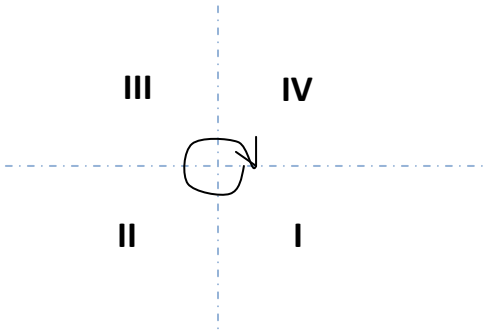


Figure 4: Sectors and Quadrants

There are twelve sectors in the circle (for twelve months). The sector shown above is the last sector

Since calendar has to start from OB, OB has to be treated as 0 degree as against 270 degree.

→Eqn 1

Hence an offset of 270 degree is added to all angle calculations

Theta' = (Theta + 270) % 360

Let's see the Coordinate Computations for the 12^{th} sector

Given from the picture:

points A,B,C,D

Center O(x,y)

innerRadius r1 = OC = OD

outerRadius r2 = OA = OB

sector = 12 (coordinates for the sector to be computed)

Angle COD = 30 degree (all the sectors have the angle of 30 degree)

startAngle = (sector – 1) * 30 = 330

Angle OC is 330 degree (as per the assumption from eqn 1)

Theta' = (330 + 270) % 360 = 240 degree

Dx = r1 * cos (theta') + x; Dy = r1 * sin (theta') + y

Ax = r2 * cos (theta') + x; Ay = r2 * sin (theta') + y

Theta' = Theta' + 30 degree

Theta'= 270 degree

Cx = r1 * cos (theta') + x; Cy = r1 * sin (theta') + y

Bx = r2 * cos (theta') + x; By = r2 * sin (theta') + y

Thus the coordinates for points A,B,C and D are computed. The above computation is done for all the twelve sectors to find its border coordinates. The iteration starts at angle 0 degree and progresses every 30 degree for each sector and finally ends at 330(12^{th} sector) degree. Since the next sector is the first sector, the iteration stops at 330 degree and for this sector the above computations were exhibited.

Next step is to find the points on the circle from point A to point B and find the coordinates of points that lie over the arc.

Given:

Co-ordinates: A, B, C, D

Center O(x,y)

Angle COD = Angle AOB = 30 degree

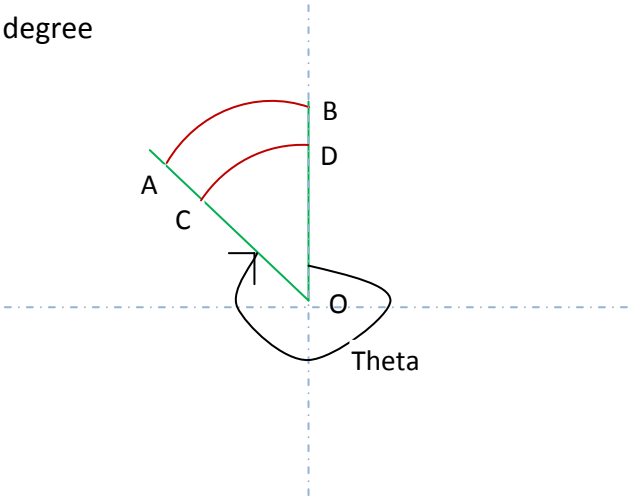


Figure 5: Points on the circle

Let's assume the no of points to be computed is same as the angle of the sector i.e a point is computed for each angle.

noOfPoints = Angle COD = 30

Theta = 330

Theta’ = 240

```
arrayCoordinates = array();
STEPS = 1;
DO STEPS < 30
        angle = Theta’ + STEPS
        pointOnTheCircleX = x + r * cos(angle)
        pointOnTheCircleY = y + r * sin(angle)
        arrayCoordinates.push(pointOnTheCircleX);
        arrayCoordinates.push(pointOnTheCircleY);
END
```

arrayCoordinates contains the coordinates for all the points on the circle between the points A and B. Similarly find the co-ordinates of all points between point C and point D. Once all the co-ordinates are computed from their border coordinates store it in their polygon’s object.

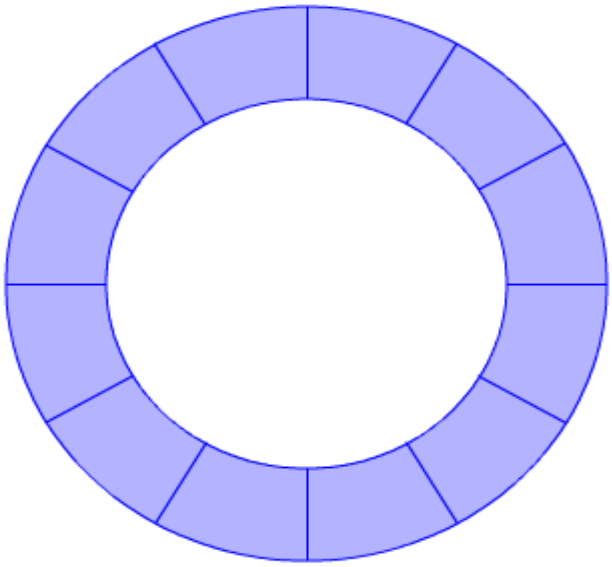


Figure 6: Polygon Faces of the Calendar

This step is repeated for all of the twelve polygons. The 'draw' is a package in this application which takes care of drawing. It has Lines, Arcs and other utils classes to support basic drawing. Line class draws a line from one co-ordinate to the other coordinate if two points are passed. If more than two points are passed it just connects all the points by iterating the array of points. Lines are drawn for the computed co-ordinates of all the polygons using the Line class. Thus, the blocks of twelve polygons arrayed in circular manner are drawn as in figure 6, which provides the basic visualization of a calendar. These polygon blocks forms a stage for the calendar events.

2.3.2 Calendar Events

This component is responsible for plotting events over the calendar. HTTPService call is made to the Data Moderator to pull up the events for the authenticated user. Once this data passes through all the stages and finally available as native arrays, the array is iterated to plot the events. The event object in the array holds information such as start date, end date, name, and description. To be plotted in the calendar the start and end date must be in the range 0 to 360, this will let the event to be plotted in angle. The day in which the event start in the year is computed from the event start date. The result is then calibrated towards the range. The same is applied to the end date. Next step is to check for the position in the calendar the event has to be plotted. If there is any event already added in the calendar, pad values to the height to check an empty slot in the next level. Values are padded until an empty slot is found in that polygon. Padding cannot be applied more than four times as there are only room for five rows of events in a polygon. If the event under subject could not be placed under any of the five rows, then the event is discarded. Otherwise, the coordinates are computed knowing the start and end angle of the event and plotted in the calendar using the draw class.

2.3.3 Spikes

This component is responsible for drawing numeric data around the calendar that forms Spikes. Things that happen throughout the year like temperature, rainfall, shares etc., are calibrated on percent basis and drawn from the same center point as the concentric circles. The outer circle stands as the base line for the spikes.

2.3.4 Call out

Call out creates name tags for the events. The number of call outs for each quarter is calculated forehand and the distance between the call outs within the quarter is computed. This makes the call outs to be equidistant from each other within each quarter. Call outs are drawn from right to left for the quarters 1 & 4 and left to right for the quarters 2 & 3.The flat line of fixed length is drawn from the other side and extends until it meets the imaginary line that is kept close to the circles. All the call out lines stay parallel until it meet the imaginary line, after that it start to bend up or down to meet their corresponding event start angle.

2.3.5 Image map

Image map is the basis of all these drawings. When an image is made clickable it is called image map. The Line class is called after finding the coordinates of the polygons that makes the polygon objects clickable by adding appropriate mouse events. Image map events look for the actions to happen over the images. When a particular action takes place, these actions will be captured and processed. For example, on mouse over tool tip has to be shown and mouse out tool tip has to be disabled. Each month acts as a separate area. Any single month can be clicked to perform action on that month.

2.3.6 Rotation

Rotation is one of the noticeable features of this calendar. When a calendar is rotated, the start angle of the calendar is altered. The new start position of the calendar is computed from the rotation made and the offset is added to each component in the calendar. All the coordinates hence forth computed are recalculated using the offset and the all the components are redrawn.

2.3.7 Preferences

Preferences appear next to the calendar and display the list of tags associated with the events, and the files that the spike spun around the calendar. Users can control the items to be displayed in the calendar by choosing their preferences. When a tag is chosen all the events in the calendar are removed followed by a HTTPService call to the data moderator in the server. Data moderator then queries the db class to find the events that belong to the selected tag or

tags. Finally, the data moderator formats the result in XML and sends it to the Calendar Events component that again repeats the whole procedure of adding the events to the calendar.

3 Design

There are four packages in the client side namely Draw, Controls, Events, and Model. All the components are kept under the controls package. The custom event classes that are required for the components are placed under Events package. The necessary functions to draw the coordinates are placed under draw. The dependency between those packages are represented below

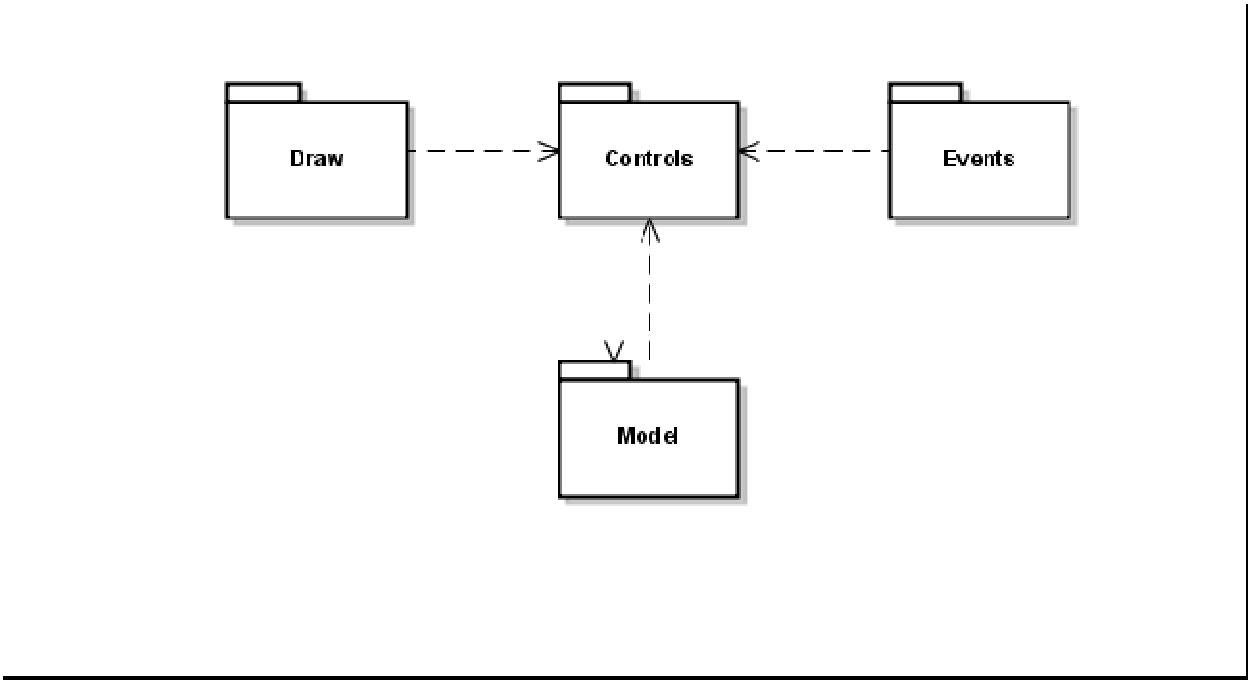


Figure 7: Design

The database design involves three tables one for the user, one for events and the another to track details about uploaded files. User table is indexed by the primary key userId. UserId is referenced as a foreign key in events and files tables. Events and files tables have their own primary key columns naming eventid and fileId. One user may have zero or more events or files and this mapping is shown in the below figure. The functions that can be applied to the Events model are addEvents(), editEvents() and deleteEvents(). These functions do their respective

operations as its name suggests. When a user uploads a file, the file is copied in to the files repository under the name of the fileId from the files table.

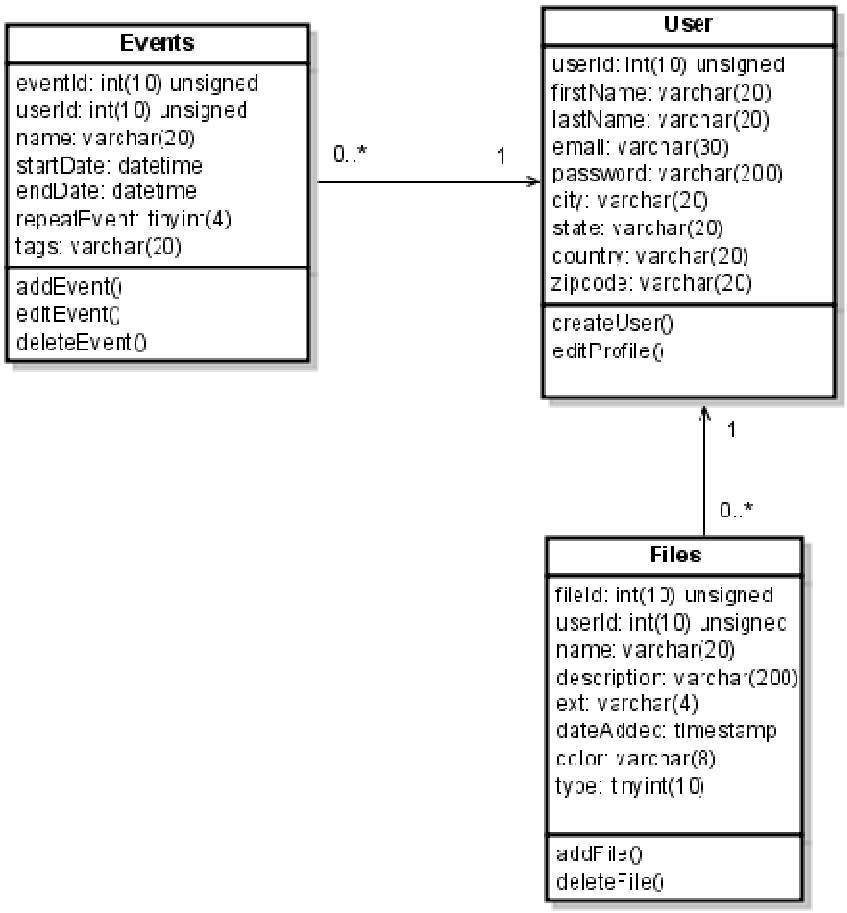


Figure 8: Database design

4 Patterns

There are two patterns incorporated for this application. One is MVC Model View Controller and another is Singleton pattern. MVC is incorporated at the full application level including client side and server side. Singleton is only approached only in client side.

4.1 Singleton Pattern

There are few singleton classes in the client components. Only one instance of each of those classes is alive throughout the user session. Instances for those classes cannot be created by other classes. This behavior is essential to share some data across the application using the same instance.

The structure of the singleton class looks like

```
Package {
        Class Singleton {
                Private static var _instance:Singleton = new Singleton(SingletonLock);
                Public function Singleton(lock: Class) {
                        If(lock != SingletonLock) {
                                Throw new Error("Invalid Singleton Access. Use Modal.instance");
                        }
                }
                Public function get instance() : Singleton {
                        Return _instance;
                }
        }
}
Class SingletonLock {
}
```

The object for Singleton class is created during the creation of the class using the SingletonLock class. The created instance is returned when any other class tries to access Singleton's object. Without SingletonLock class Singleton cannot be instantiated. Since SingletonLock is available only to Singleton, no other class could instantiate Singleton. The classes that are singleton in this application are

4.1.1 MediatorModel

MediatorModel is used as a mediator to fire events. When two independent classes i.e., classes with no relation needs to communicate with each other to inform the other class about some completion of an action, either of these classes cannot create instance on each other to float events between them. This restriction can be overcome by creating a singleton mediator class. This class is kept public and made accessible across the application. The independent classes register with the mediator class and one class pass the information to the other class by informing the mediator class about the action. The mediator class in turn informs the target class about the information it got from the source class.

4.1.2 CommonFunctions

This class holds common functions that are used throughout the drawing. Functions include getCoordinateForAPoint(): find coordinates for a point when the angle and radius are known. getRadians(): Get the radians from degree converstionToActualAngle(): Offsets are used while drawing the calendar. This function returns the actual angle when a modulated angle is sent

4.2 MVC

Model View Controller is also adopted in Client side components. Most of the controllers have its logic separated from the view and model. Controllers are written in action script and Views are written in mxml (Macromedia XML). Users, Polygon faces are the Models that are even written in action script.

4.2.1 User

User class is served as a Model. Since only one user is active in the client side, this class is created as a singleton. This class does not hold any events and just stores information about the user.

4.2.2 Calendar

Calendar is the view as it just shows the presentation of the calendar. The logic that is needed to draw the calendar is written in action script and acts as a controller. The view of the calendar has the ability only to present the component and add some styles to the component if needed.

5 Event Management

All the components in the client side are managed using events. The components are executed when some action in the other component happens. As the execution of the components depends on the completion of other components, all components need to register with appropriate events to start its execution. Few of the events in the components are created specifically for the calendar and others are existing events in flex. The events are dispatched using

dispatchEvent(evt);

If the event being passed is a custom event, then the object of the custom event needs to be instantiated before passing it. The events are captured using

object.addEventListener(EventName,CallBackFunction);

object is the Object of a class which dispatches the event. The above line hooks the function to be called with the dispatcher.

The events that happen in the calendar components are

5.1 Polygon Faces Created:

This event is represented by the constant POLYGON_FACES_CREATED and a custom class is created for this event called PolygonFacesCreated. This event fires when the polygon faces for the months are computed. Information about the completion of the previous task should be passed to next task so that the next task starts. The next task is to plot events on the created polygons using Calendar Events. The definition of this class is given in the appendix. Both the class that triggers this event and the class that reacts to this event must register with the class PolygonFacesCreated.

Event Dispatching:

```
var eventObj:PolygonFacesCreated = new
PolygonFacesCreated(PolygonFacesCreated.POLYGON_FACES_CREATED);
        eventObj.polygonFaces = polygonFaces;
        dispatchEvent(eventObj);
```

Event Capturing:

```
CreateCoordinates.instance.addEventListener(PolygonFacesCreated.POLYGON_FACES
_CREATED,plotEventDetails);
```

Create Coordinates is the class that actually dispatches the event. CreateCoordinates.instance gives the existing instance of the class and an event listener is added to the object.

5.2 Mediator Model Events

The description of this singleton class is discussed in section 4.1 This class supports the following events.

5.2.1 CALENDAR_EVENTS_MODEL_COMPUTED

When the Calendar Events component finishes its job by plotting all the events in the calendar, it requests the MediatorModel to fire this event. After getting the request from the Calendar Events, MediatorModel dispatches this event. Finally, this event is captured in CallOut component to create call outs for the created events.

5.2.2 CALENDAR_EVENT_REFRESH

When there is a change in the events that is plotted over the calendar, this event is triggered. For example, user may delete an event through Manage Events section which is completely independent of the Calendar Events. In order to let Calendar Events to re plot the events, the ManageEvents have to inform CalendarEvents about the action taken. The bridge is made through the MediatorModel's CALENDAR_EVENT_REFRESH. When this event is triggered the CalendarEvents remove all the existing events from the calendar and plots the events again on the calendar.

5.2.3 CALENDAR_SPIKE_REFRESH

When a file is added or removed, this event is triggered to re draw the spike that is spinning around the calendar.

5.3 ApplicationEvents

User authentication component is kept separate from the highly coupled drawing components. ApplicationEvents is a singleton class that provides the bridge between the authentication components and drawing components. Upon successful user authentication, the Calendar component is notified of the result using this ApplicationEvents. Since Calendar is the basic component and without the completion of it no other components initialize, it is enough to notify Calendar component alone about the result of user authentication. The ApplicaitionEvents has the following events

5.3.1 USER_LOGIN_SUCCESS

When the user logs in successfully, Calendar component is informed about the result and it will activate the drawing sequence discussed above.

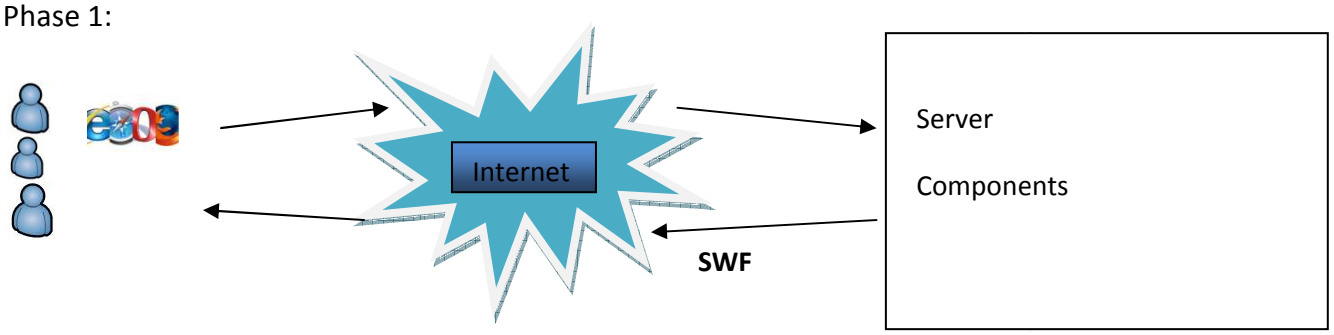
5.3.2 USER_LOGOUT_SUCCESS

This event is triggered when the user logs out of the application successfully and the control gets transferred to the authentication component to let the user login if necessary.

6 Workflow

Workflow involves two phases of communication. In the first phase, user initiates the request and the server responds by providing the packaged application embedded in the response as a flash object. The application is then fully downloaded in user browser. After the download is complete, the application initializes itself and asks the user for authentication which kicks off phase 2.

Phase 1:



Phase 2:

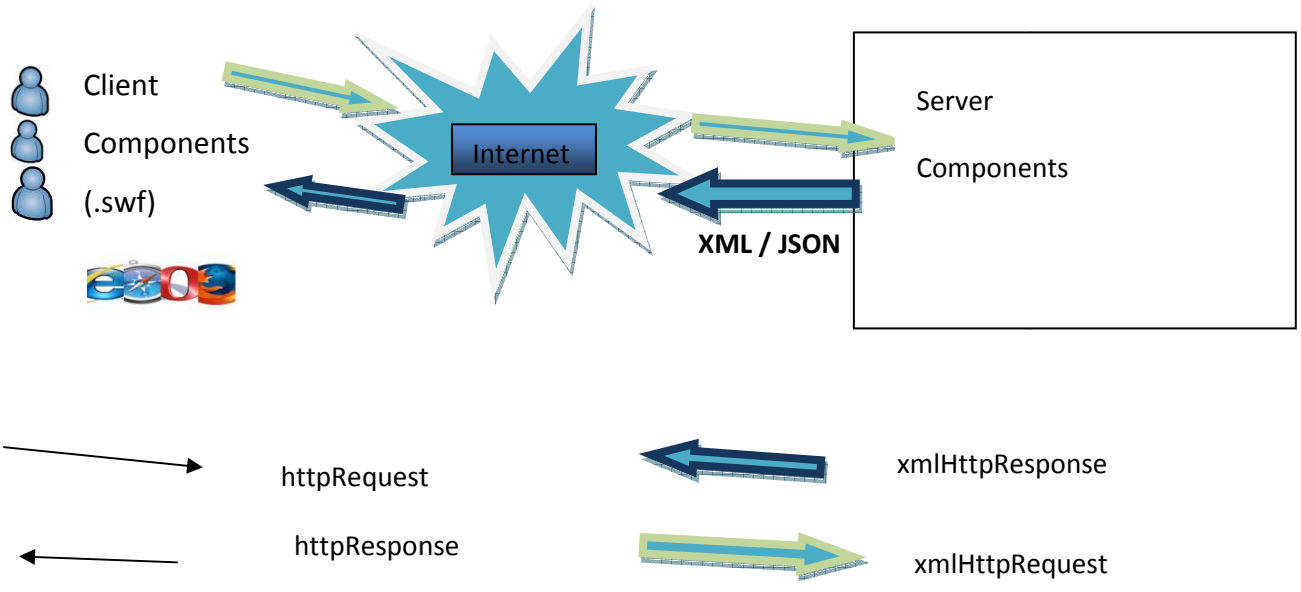


Figure 9: Project Workflow

All connections to the server in phase 2 are taken as Flex HTTP Service which is similar to xmlHttpRequest

6.1 Calendar Interface

The integration of client side components produces the calendar which is shown in the below figure. The first component is the representation of twelve months. The second component reflects events. The third component produces the call outs for the events and the last component draws the spikes for the rainfall data. All of these components can be enabled or disabled.

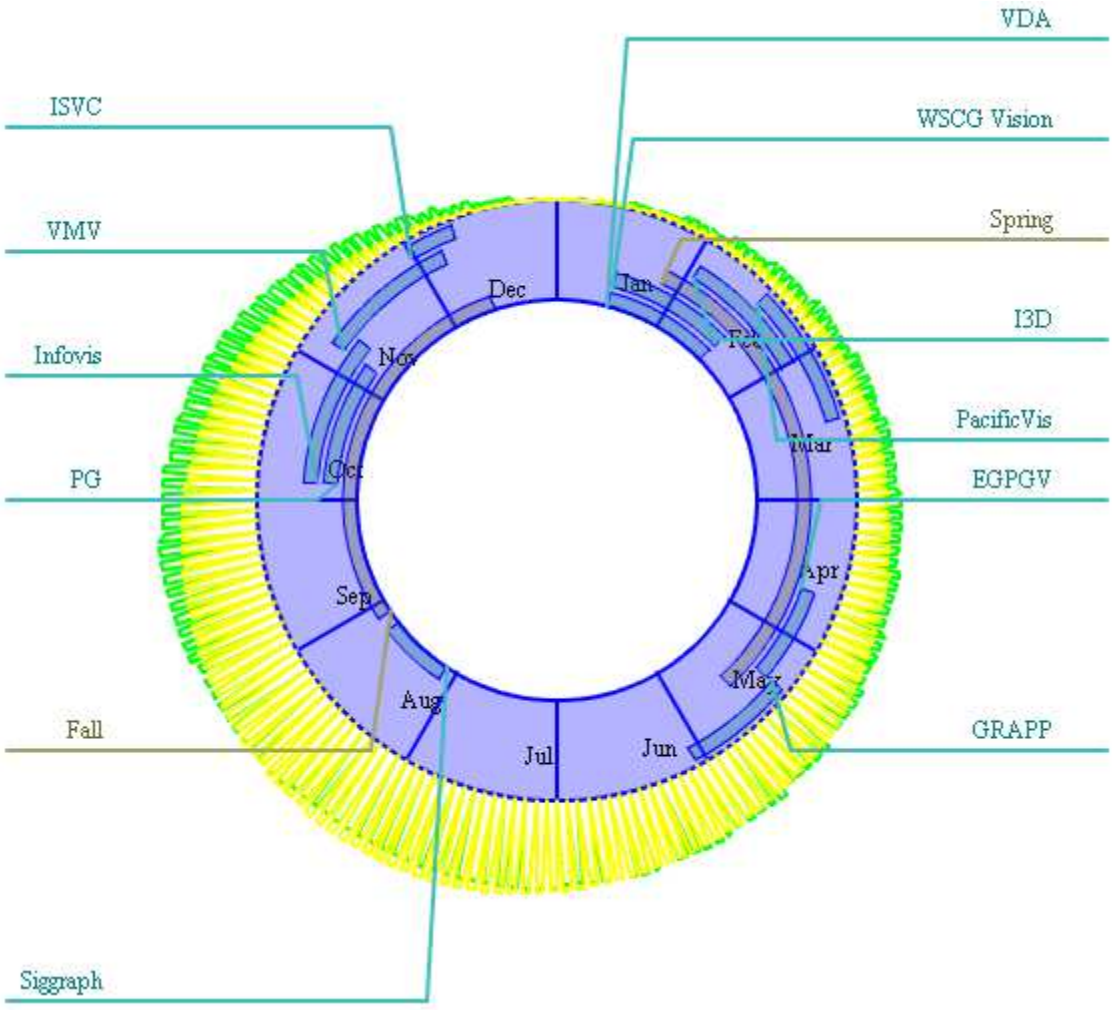


Figure 10: Calendar

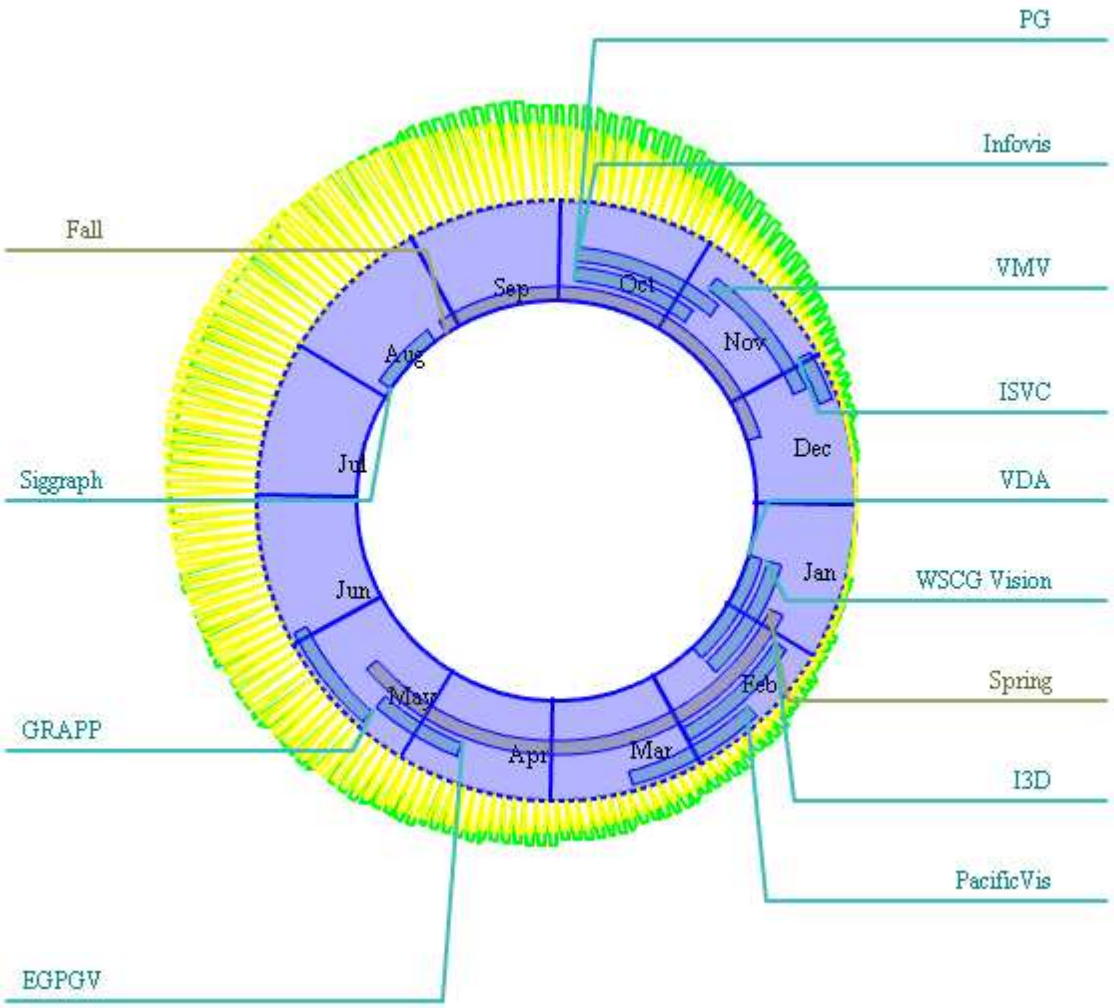


Figure 11: Calendar with a rotation of 90 degree

Figure 3 shows the representation of the calendar controller. The months in the year are plotted in a circular fashion as polygons. All the months together form an image and each month acts as an individual entity. Any month can be clicked to view more information about the activities and events on that month. The small scales over the polygons which spans across months are events. Each event is identified through the call out lines. The length of the scale denotes the duration of the event. Similarly the start position and end position is directly mapped to start date and end date. If the user has more events and if he could not clearly see the events he can hide the low priority events. User has the option to show or hide events based on the tags. Events may or may not be associated with tags. The list of available tags is

shown on the preferences area from where user can select the tags whose events need to be displayed. User can dynamically add, edit or delete events. The complete view of the calendar interface along with its preferences is shown in the below figure.

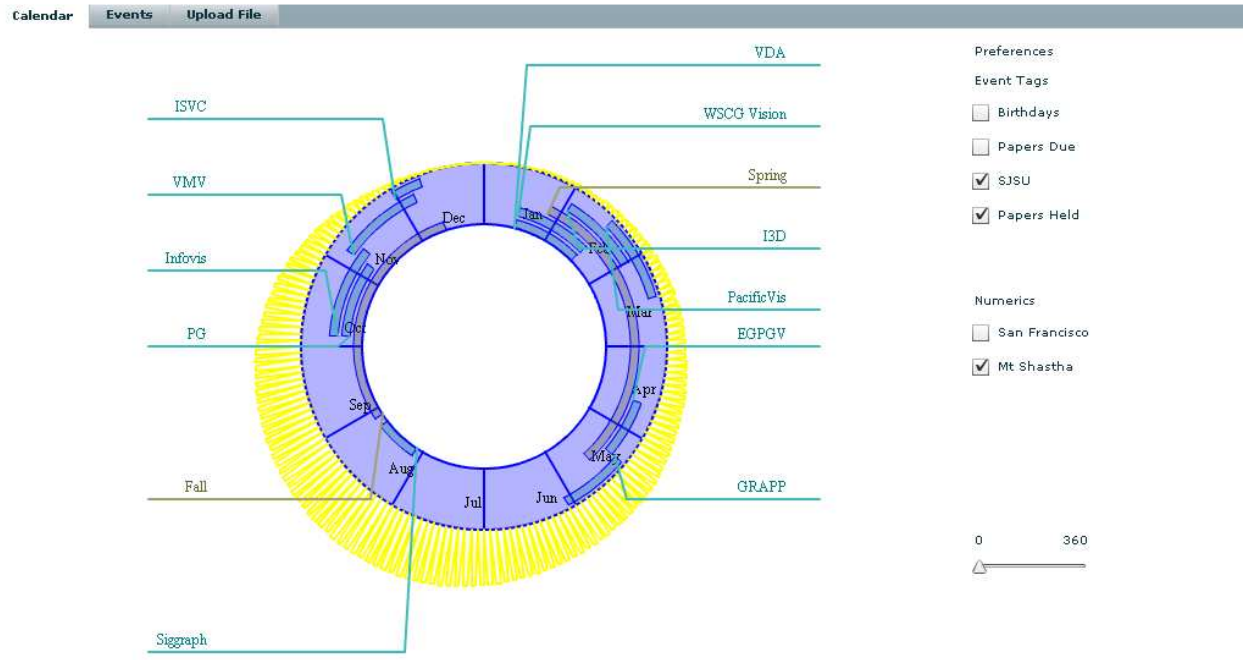


Figure 12: Calendar and its Preferences

6.2 Plotting Rainfall

Normally, the angle is measured in anti clockwise from 0 to 360. But the drawing tools consider the angle in clockwise direction. So 30 degrees in paper is equivalent to 330 degrees in the drawing tool. To avoid confusions, this project measures the angle in clockwise direction and the zero degree is equivalent to 90 degree in paper and 270 degree in the drawing tool. Basically, this interface assumes the month January starts at an angle 0 degree, February starts at an angle 30 degree eventually December starts at an angle 330. It takes time clock as the metaphor to measure angle. Angle calculations are made to offset these differences.

Plotting the rainfall information for a particular day needs the angle the current day is away from the day one. Once the angle is determined the pointer will go to the outer region of the calendar at that angle from the center point. From that point it will find the co ordinate for the rainfall data which will lie in the same line at some distance. Once the coordinate is found, a line will be drawn from the previous day's rainfall coordinate to the current pixel position.

The outer region of the calendar is considered as the zero line for plotting rainfall information. For example if the rainfall value for a day is 3 inches, the zero point for this day will be the pixel at the outer region of the calendar at an angle of theta degrees from the day one with respect to center. The value for each date is taken from the XML file and the corresponding coordinate for each date will be computed.

User can add as many cities as he/she wants for the rainfall data to be displayed over the calendar. All the cities will be displayed in the preference list and will let the user to hide/show any of the cities rainfall.

6.3 Manage activities

The events and the spikes that appear in the calendar are managed separately. Event uploader is a component that manages the events. It provides a way to take input from the user to add an event and displays the list of existing events in a table. User can delete the events by selecting the event and click delete to make that operation. Spikes are handled in a different way. It asks the user to input the xml file that contains data throughout for a year. The files are directly uploaded in the server and tagged with userId. Similar to Event uploader, it shows the list of files already uploaded. User can select the files and delete those if they do not want those files.

In the back end the user data is organized in three tables

7 Environments

7.1 MySQL

MySQL is an open source relational database system (RDBMS) and is used widely because of its fast performance, high reliability and ease of use. MySQL is used in this project as a database for data persistence. SQL Structured Query Language needs to be used to manipulate the data.

7.2 PHP

PHPs are used to perform business logic. Layer 1 and Layer 2 of the server side components are fully written using PHP. Since PHP supports Object Oriented Programming System, a good level of abstraction is provided. The class that deals with database provides such abstraction. As

there are multitudes of library functions available in PHP, many tasks are achieved in an easiest way.

7.3 Flex 3

It is the technology from Adobe Systems for the development and deployment of Rich Internet Applications. The applications built over flex 3 uses adobe flash player to run in the browser. Flex files uses MXML, a XML based user interface markup language to layout controls and graphics, and uses action script to perform scripting. Flex 3 and action script is used to perform the graphical part of this project. Basically flex 3 lies in the presentation layer of the project.

7.4 XML

XML is an Extensible Markup Language. Data is transferred as XML between layers in the application tier, and also between the application tier and presentation tier. This data structure facilitates sharing of data between other applications too.

8 Appendix

8.1 Table Structure

8.1.1 User table

```
DROP TABLE IF EXISTS calendar.user;

CREATE TABLE user (

 userId int(5) unsigned NOT NULL AUTO_INCREMENT,

 firstName varchar(20) NOT NULL,

 lastName varchar(20) NOT NULL,

 email varchar(30) DEFAULT NULL,

 password varchar(200) DEFAULT NOT NULL,

 city varchar(20) DEFAULT NULL,

 state varchar(20) DEFAULT NULL,

 country varchar(20) DEFAULT NULL,

 zipcode varchar(20) DEFAULT NULL,
```

```
  PRIMARY KEY (userId)
) ENGINE ENGINE=InnoDB DEFAULT CHARSET=utf8
```

8.1.2 Events table

```
DROP TABLE IF EXISTS calendar.events;
CREATE TABLE events (
  eventId int(5) NOT NULL AUTO_INCREMENT,
  userId int(11) unsigned NOT NULL,
  name varchar(20) DEFAULT NULL COMMENT 'Event Name',
  startDate datetime DEFAULT NULL COMMENT 'Event start date in mm/dd/yy hh:mm:ss',
  endDate datetime DEFAULT NULL,
  repeatEvent tinyint(4) DEFAULT NULL COMMENT '1 yearly   2 monthly  3 weekly  4 daily ',
  tags varchar(20) DEFAULT NULL,
  PRIMARY KEY (eventId)
) ENGINE ENGINE=InnoDB DEFAULT CHARSET=utf8
```

8.1.3 Files table

```
DROP TABLE IF EXISTS calendar.files;
CREATE TABLE files (
  fileId int(5) unsigned NOT NULL AUTO_INCREMENT,
  name varchar(20) NOT NULL COMMENT 'Name of the file',
  description varchar(200) DEFAULT NULL,
  userId int(11) unsigned DEFAULT NULL COMMENT 'uploaded by the user',
  ext varchar(4) NOT NULL COMMENT 'File Extension',
  dateAdded timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  color varchar(8) DEFAULT '0x000000',
  type tinyint(10) DEFAULT NULL,
  PRIMARY KEY (fileId)
```

) ENGINE ENGINE=InnoDB DEFAULT CHARSET=utf8

8.2 Events

8.2.1 Calendar Component

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="init()" xmlns:controls="sjsu.ccalendar.controls.*"
      xmlns:LibControls="flexlib.controls.*">
<mx:Script>
            <![CDATA[
                  import sjsu.ccalendar.events.PolygonFacesCreated;
                  import flexlib.controls.*;
                  import sjsu.ccalendar.CommonProperties;
                  import sjsu.ccalendar.controls.Polygon;
                  import mx.core.UIComponent;
                  import sjsu.ccalendar.controls.CreateCoordinates;
                  import mx.controls.Alert;
                  import sjsu.ccalendar.controls.LoadXMLData;

                public var objCreateCoordinates:CreateCoordinates;
                public var objPolygon:Polygon;
                public var circle1:UIComponent;
                private var polygonCoordinates:Array = new Array();

                  private function init():void {
                  objCreateCoordinates = CreateCoordinates.instance;

objCreateCoordinates.addEventListener(PolygonFacesCreated.POLYGON_FACES_CREAT
ED,plotSectors);
                  objCreateCoordinates.createFaces();
                  }

                  public function rotateBy(angle:int) : void {

                        imgMap.map = null;
                        objPlotEvents.rotateBy(angle);
                        objFaceTexts.canvasTextFields.removeAllChildren();
                        objCreateCoordinates.rotateBy(angle);

                  }

                  private function
plotSectors(event:PolygonFacesCreated):void {
                        var polygonFaces:Array = event.polygonFaces;
                        var mapArray:Array = new Array();
                  for(var k:int;k<polygonFaces.length;k++) {
                        objPolygon = polygonFaces[k];
                        polygonCoordinates[k] =
objPolygon.getPointsInAllSides();
                        var singleArea:area = new area();
                        singleArea.href = "";
                        singleArea.alt = objPolygon.getName();
```

```
                        singleArea.coords =
objPolygon.getPointsInAllSides().toString();
                        singleArea.target = "_blank";
                        singleArea.shape = "POLY";
                        singleArea.fillColor = objPolygon.color;
                        mapArray.push(singleArea);
                  }
                  imgMap.map = mapArray;
                  //shapeClick="navigateToURL(new URLRequest(event.href),
event.linkTarget)" fillColor="0x009dff"
                  }

            ]]>
      </mx:Script>

      <LibControls:CalendarImageMap scaleContent="true" id="imgMap"
            source=""
            outlineAlpha="1" outlineColor="0x0000ff"
            fillAlpha=".3"
            showToolTips="true" width="100%" maxWidth="554"
            shapeClick="mx.controls.Alert.show('hi'+imgMap.toolTipField)"
x="0" y="0" >
      </LibControls:CalendarImageMap>
      <controls:FaceTexts id="objFaceTexts"/>
      <controls:PlotEvents id="objPlotEvents" />
      <controls:DrawNumerics />
      <controls:CallOut />
</mx:Canvas>
```

8.2.2 Event Mediator

```
package sjsu.ccalendar.model
{
      import flash.events.Event;
      import flash.events.TextEvent;

      import mx.collections.ArrayCollection;

      [Bindable]
      [Event (name = "calendarEventsModelComputed", type="flash.types.Event")
]
      [Event (name = "calendarEventRefresh", type="flash.types.TextEvent") ]
      [Event (name = "calendarSpikeRefresh", type="flash.types.TextEvent") ]

      public class MediatorModel
      {
            public static var CALENDAR_EVENTS_MODEL_COMPUTED:String =
"calendarEventsModelComputed";
            public static var CALENDAR_EVENT_REFRESH:String =
"calendarEventRefresh";
            public static var CALENDAR_SPIKE_REFRESH:String =
"calendarSpikeRefresh";

            private static var _instance: MediatorModel = new MediatorModel (
SingletonLock );
```

```
            public static function get instance () : MediatorModel { return
_instance; }
            public function MediatorModel( lock: Class) {
                  // Verify that the lock is the correct class reference.
                   if ( lock != SingletonLock )  {
                        throw new Error( "Invalid Singleton access.  Use
Model.instance." );
                   }
            }

            public var allEventsCollection: ArrayCollection = new
ArrayCollection();

            public function fireCalendarEventsModelCreated() : void {
                  var evt:Event = new Event(CALENDAR_EVENTS_MODEL_COMPUTED);

                  dispatchEvent(evt);
            }

            public function fireEventsRefresh(tags:String = "") : void {
                  var evt:TextEvent = new TextEvent(CALENDAR_EVENT_REFRESH);
                  evt.text = tags;
                  dispatchEvent(evt);
            }
            public function fireSpikeRefresh(tags:String = "") : void {
                  var evt:TextEvent = new TextEvent(CALENDAR_SPIKE_REFRESH);
                  evt.text = tags;
                  dispatchEvent(evt);
            }

      }
}
class SingletonLock
{
} // end class
```

8.2.3 Authentication Events

```
package sjsu.ccalendar.events
{
      import flash.events.Event;

      import mx.collections.ArrayCollection;

      [Bindable]
      [Event (name = "loginSuccess", type="flash.types.Event") ]
      [Event (name = "logoutSuccess", type="flash.types.Event") ]

      public class ApplicationEvents
      {
            public static var USER_LOGIN_SUCCESS:String = "loginSuccess";
            public static var USER_LOGOUT_SUCCESS:String = "logoutSuccess";
            private static var _instance: ApplicationEvents = new
ApplicationEvents ( SingletonLock );
```

```
            public static function get instance () : ApplicationEvents {
return _instance; }

            public function ApplicationEvents( lock: Class) {
                  // Verify that the lock is the correct class reference.
                   if ( lock != SingletonLock )  {
                        throw new Error( "Invalid Singleton access.  Use
Model.instance." );
                   }
            }

            public var allEventsCollection: ArrayCollection = new
ArrayCollection();

            public function fireLoginSuccess() : void {
                  var evt:Event = new Event(USER_LOGIN_SUCCESS);
                  dispatchEvent(evt);
            }

            public function fireLogoutSuccess() : void {
                  var evt:Event = new Event(USER_LOGOUT_SUCCESS);
                  dispatchEvent(evt);
            }

      }
}
class SingletonLock
{
} // end class
```

8.2.4 Lines Class

```
package sjsu.ccalendar.draw
{
      import flash.display.Graphics;

      import mx.core.UIComponent;
      import mx.styles.CSSStyleDeclaration;
      import mx.styles.StyleManager;

      [Style(name="outlineThickness", type="Number", format="Length",
inherit="no")]
      [Style(name="outlineColor", type="uint", format="Color", inherit="no")]
      [Style(name="outlineAlpha", type="Number", format="Length",
inherit="no")]
      [Style(name="fillColor", type="uint", format="Color", inherit="no")]
      [Style(name="fillAlpha", type="Number", format="Length", inherit="no")]

      public class Lines extends UIComponent
      {
            private var sprite:UIComponent = new UIComponent();
            private var g:Graphics = sprite.graphics;
            public var outlineThickness:Number;// =
getStyle("outlineThickness");
            public var outlineColor:uint ;// = getStyle("outlineColor");
            private var outlineAlpha:Number;// = getStyle("outlineAlpha");
```

```
		private var fillColor:uint;// = getStyle("fillColor");
		private var fillAlpha:Number;// = getStyle("fillAlpha");
		private var _coords:Array = new Array();
		private var _shape:String;
		//public var id:int;

		//private static var stylesInitialised:Boolean = initStyles();
				/**
		 * @private
		 *
		 * The default styes are defined here.
		 */
		private static function initStyles():Boolean {
			var sd:CSSStyleDeclaration =
			StyleManager.getStyleDeclaration("Lines");

			if (!sd)
			{
				sd = new CSSStyleDeclaration();
				StyleManager.setStyleDeclaration("Lines", sd, false);
			}

			sd.defaultFactory = function():void
			{
				this.outlineColor = 0xFF0000;
				this.outlineAlpha = 1;
				//this.outlineThickness = 1;
				this.fillColor = 0xff0000;
				this.fillAlpha = 0;
			}
			return true;
		}

		public function Lines(shapeType:String="open")
		{
			//this.outlineColor = 0xFF0000;
			this.outlineAlpha = 1;
			this.outlineThickness = 1;
			this.fillColor = 0xFF0000;
			this.fillAlpha = 0;
			_shape = shapeType;


		}

		public function set coords(value:Array):void {  _coords = value;
}
		public function set shape(value:String):void {  _shape = value; }

		//public function set

		public function drawLine() : void  {
		//    this.outlineColor = getStyle("outlineColor");
			g.lineStyle(outlineThickness, outlineColor, outlineAlpha);

			if(_shape == "close")
				drawPoly();
```

```
                else if (_shape == "open")
                        drawLinesFromCoords();

                this.addChild(sprite);
        }

        private function drawPoly():void {
                g.beginFill(fillColor, fillAlpha);
                g.moveTo(_coords[0], _coords[1]);

                //since we moved to the first point, we loop over all
points starting on the second point
                for(var i:int=2; i<_coords.length; i+=2) {
                        g.lineTo(_coords[i], _coords[i+1]);
                }
                //got to remember to reconnect from the last point to the
first point
                g.lineTo(_coords[0], _coords[1]);
                g.endFill();
        }
        public function drawLinesFromCoords():void {
                g.moveTo(_coords[0], _coords[1]);

                //since we moved to the first point, we loop over all
points starting on the second point
                for(var i:int=2; i<_coords.length; i+=2) {
                        if(_coords[i] != -1 && _coords[i+1] != -1)
                                g.lineTo(_coords[i], _coords[i+1]);
                        else
                                g.moveTo(_coords[i+2], _coords[i+3]);
                }
                //Just don't connect the last & first point
        }

    }
}
```

8.3 Rainfall Information

Rainfall information for the month January 2008 is shown below, this file actually continues for all the months.

```
<?xml version="1.0"?>

<result>
<data>
<no>50</no>
<no>50</no>
<no>70</no>
<no>80</no>
<no>90</no>
<no>80</no>
```

<no>70</no>
<no>50</no>
<no>55</no>
<no>55</no>
<no>50</no>
<no>55</no>
<no>0</no>
<no>0</no>
<no>5</no>
<no>5</no>
<no>5</no>
<no>5</no>
<no>5</no>
<no>5</no>
<no>5</no>
<no>0</no>
<no>5</no>
<no>0</no>
<no>5</no>
<no>5</no>
<no>5</no>
<no>5</no>
<no>5</no>
<no>35</no>
.
.
.
</data>
</result>

8.4 Screens

8.4.1 Preferences

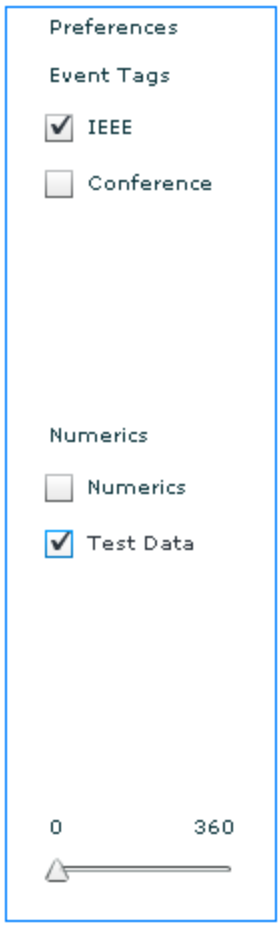


Figure 13: Preference Component

8.4.2 Events Uploader

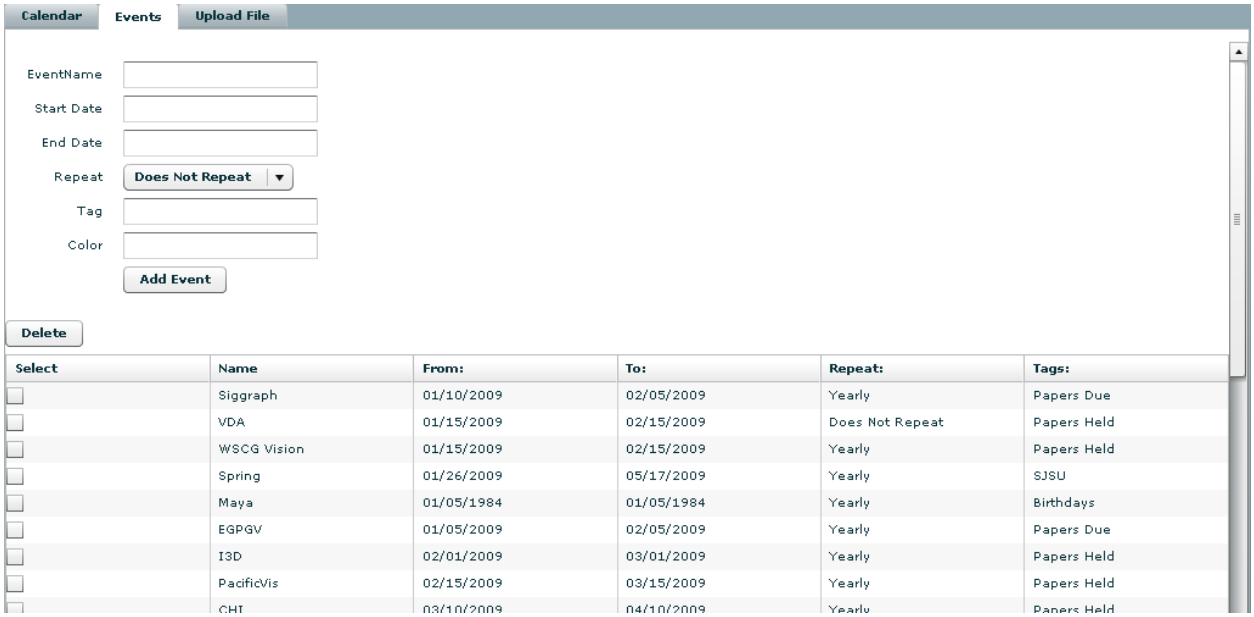


Figure 14: Event Uploader

8.4.3 File Uploader

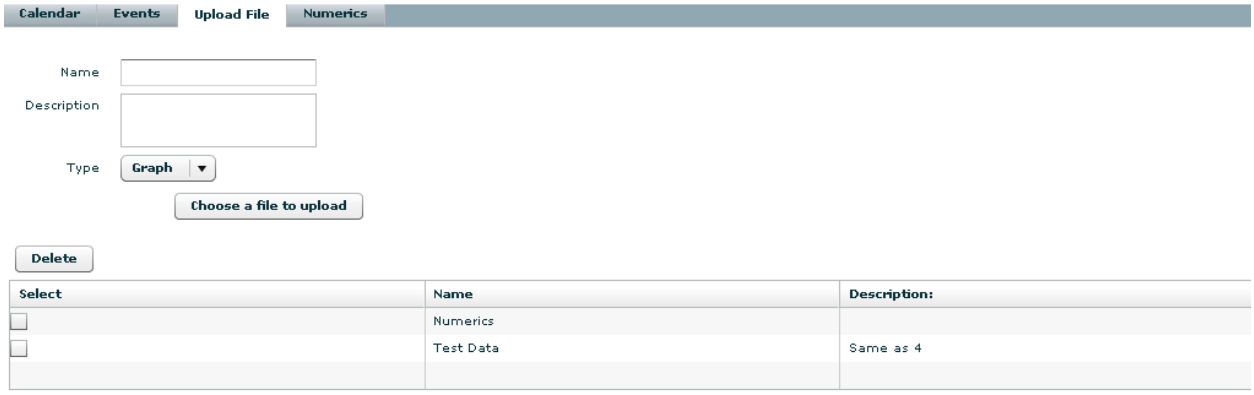


Figure 15: File Uploader

9 References

[1] Arrington, M. *Calendar comparisons.* http://www.techcrunch.com/2007/01/04/online-calendar-wiars/

[2] *DynaCal.* (2008). http://www.dynacal.com/

[3] *Event keeper.* (2008). http://www.eventkeeper.com/

[4] *Family planner.* (2008). http://www.familytimeplanner.com/

[5] *Google calendar.* (2008). http://www.calendar.google.com.libaccess.sjlibrary.org

[6] *LoCalendar.* (2008). http://www.localendar.com/

[7] Null, J. *Weather services.* http://ggweather.com/

[8] *Schedule organizer.* http://www.schedule-organizer.de/en/

[9] *Weather.* (2008). http://www.wrh.noaa.gov.libaccess.sjlibrary.org

[10] *Yahoo calendar.*, 2008, from http://www.calendar.yahoo.com/