WeatherInfo: A Weather Information System with a Web Service

Wen Xiong

Dept. of Computer Science
Oregon State University
Corvallis, OR 97331
xiongwe@cs.orst.edu

Abstract

With the rapid development of Internet technology and popularity of Web solutions, the Internet has become a major means for collecting information. The **WeatherInfo** system is conceived and developed to provide weather data for end users in tabular and graphic formats over the Internet. In order to make data retrieval and manipulation easier, **WeatherInfo** uses a relational database to store the weather data extracted from the Website run by the U.S. National Weather Service. The system also includes a Web service implemented in ASP.NET to support data exchange. This Web service offers several methods for client applications to retrieve the weather data.

Contents.. 2

Acknowledgements

The author would like to thank Toshi Minoura, Prasad Tadapalli, and Jeffrey Steiner for many helpful comments on this work.

1. Introduction

The U.S. National Weather Service is a government organization that collects weather data from a wide variety of sources. Those data are provided in textual form on the Web pages. To make them easy to comprehend, we developed the **WeatherInfo** system.

WeatherInfo gathers the desired weather data, stores them in a relational database, retrieves the requested weather data from the database, and produces Web pages to be sent to Web browsers. The system allows a user to obtain the weather data either as a graph or as a table based on a request submitted from a Web browser. In the request, the city, the time period, and the time interval for the data to be retrieved can be specified.

Furthermore, a Web service of this system provides a procedural interface for remote client programs. The **WeatherInfo Web Service** is a .NET Web service that supports Simple Object Access Protocol (SOAP). This Web service describes its interface in Web Service Description Language (WSDL). The system allows client applications written in different programming languages on different platforms to talk to the Web service.

The overview of the **WeatherInfo** is described in Section 2, and the implementation details of its five components are discussed in Section 3. Sections 4 concludes this report.

2. Overview of WeatherInfo

The **WeatherInfo** system allows end users to obtain weather data easily and promptly. This system uses a Web service to support data exchange. Client programs can easily process the retrieved data by a procedural interface provided by the Web service.

The system consists of five modules shown as Figure 2.1: **WeatherDataCollector, WeatherData, WeatherInfo Web Application, WeatherInfo Web Service,** and **WeatherClient.**

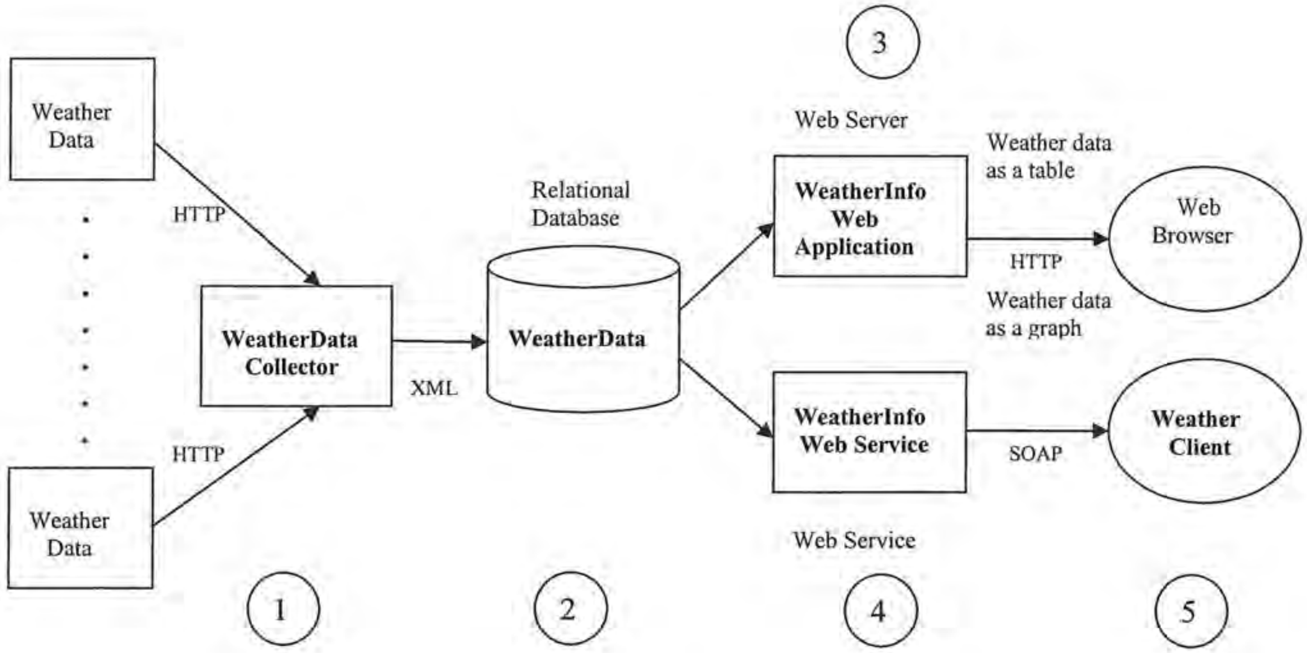


Figure2.1. Architecture of the **WeatherInfo** System.

2.1 WeatherDataCollector

WeatherDataCollector gathers weather information from the Web site run by U.S. National Weather Services. The weather data for major cities of all the states are available from this site.

WeatherDataCollector uses .NET class `WebRequest` to get weather records with HTTP requests. The retrieved weather data are extracted with regular expressions and converted into an XML file. **WeatherDataCollector** then executes Transact SQL function `OPENXML`, to read this XML file and uses a SQL stored procedure to insert the data into the database in the batch mode. **WeatherDataCollector** is activated once a day by the Window 2000 job scheduler.

2.2 WeatherData

A relational database named **WeatherData** was developed with Microsoft SQL Server. This database stores the weather data extracted by **WeatherDataCollector**. Figure 2.2 illustrates the schema of the database. The **WeatherData** database contains five tables: `City`, `HourlyWeather`, `DailyWeather`, `MonthlyWeather`, and `YearlyWeather`. The attributes in bold font are the primary keys and the attributes underscored are foreign keys.

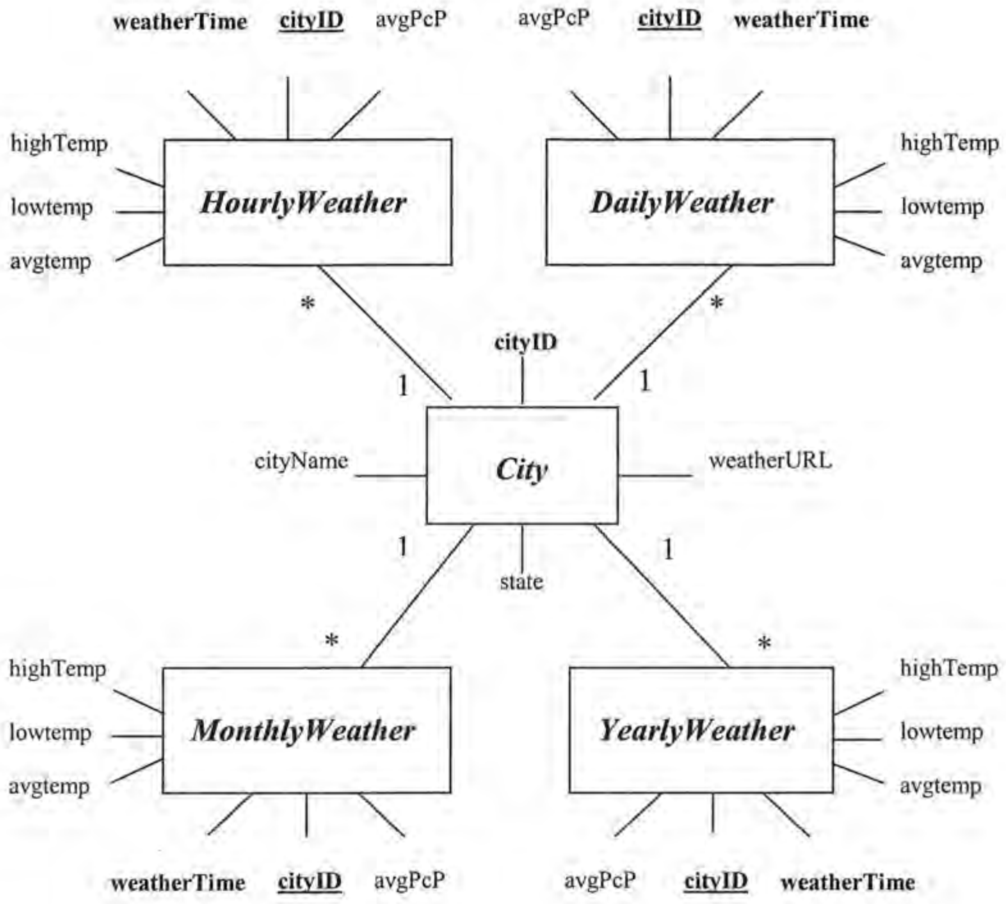
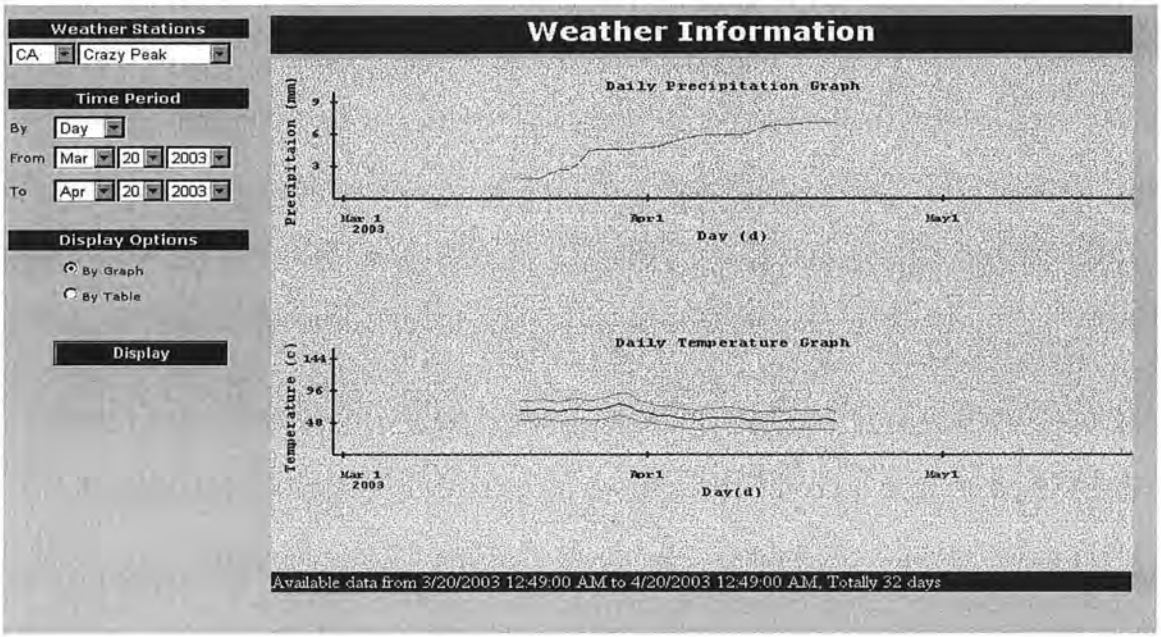


Figure 2.2. Schema of the **WeatherData** Database.

2.3 WeatherInfo Web Application

In order to enable end users to directly view the weather data managed by **WeatherInfo**, a Web application was implemented. The **WeatherInfo Web Application** generates a table and a graph of weather data for each request submitted from a Web browser. The sample of the **WeatherInfo Web Application** is shown in Figure 2.3.



Weather Stations

CA | Crazy Peak

Time Period

By Day

From Mar 20 2003

To Apr 20 2003

Display Options

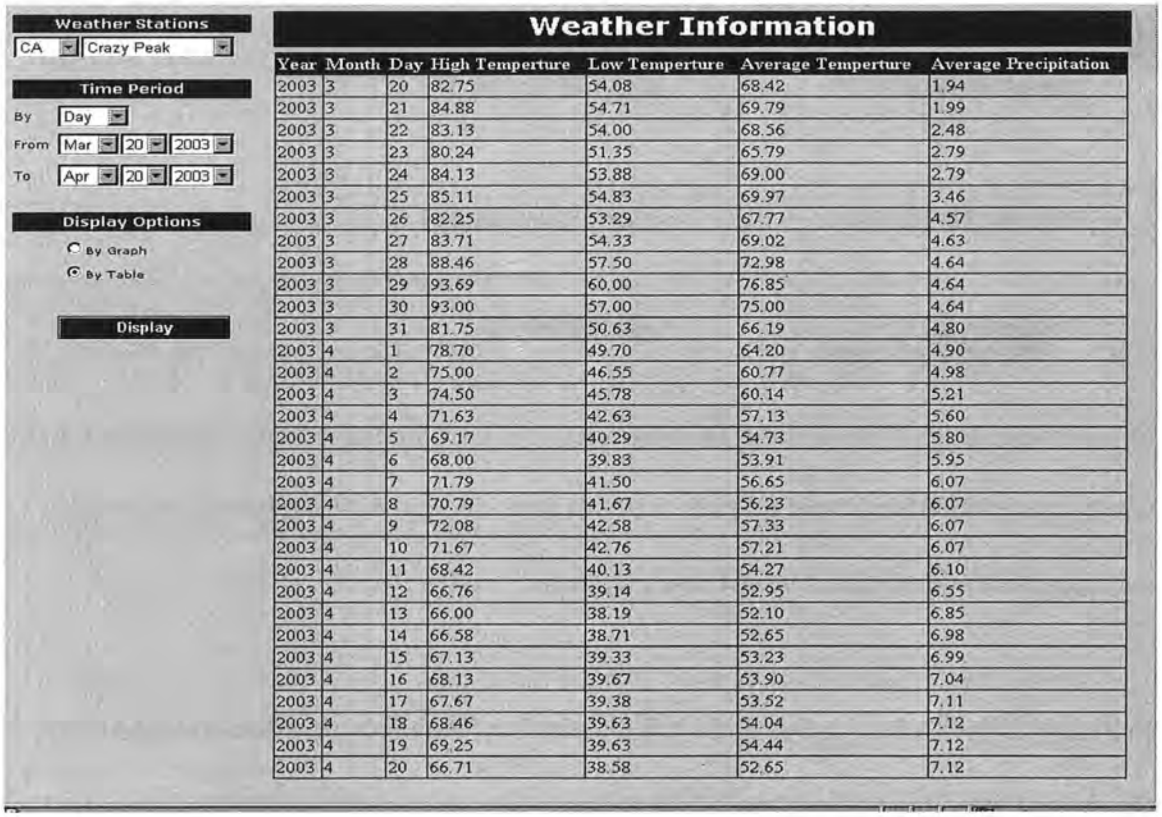
● By Graph

○ By Table

Display

Weather Information

Available data from 3/20/2003 12:49:00 AM to 4/20/2003 12:49:00 AM, Totally 32 days



Weather Stations

CA | Crazy Peak

Time Period

By Day

From Mar 20 2003

To Apr 20 2003

Display Options

○ By Graph

● By Table

Display

Weather Information

Year	Month	Day	High Temperture	Low Temperture	Average Temperture	Average Precipitation
2003	3	20	82.75	54.08	68.42	1.94
2003	3	21	84.88	54.71	69.79	1.99
2003	3	22	83.13	54.00	68.56	2.48
2003	3	23	80.24	51.35	65.79	2.79
2003	3	24	84.13	53.88	69.00	2.79
2003	3	25	85.11	54.83	69.97	3.46
2003	3	26	82.25	53.29	67.77	4.57
2003	3	27	83.71	54.33	69.02	4.63
2003	3	28	88.46	57.50	72.98	4.64
2003	3	29	93.69	60.00	76.85	4.64
2003	3	30	93.00	57.00	75.00	4.64
2003	3	31	81.75	50.63	66.19	4.80
2003	4	1	78.70	49.70	64.20	4.90
2003	4	2	75.00	46.55	60.77	4.98
2003	4	3	74.50	45.78	60.14	5.21
2003	4	4	71.63	42.63	57.13	5.60
2003	4	5	69.17	40.29	54.73	5.80
2003	4	6	68.00	39.83	53.91	5.95
2003	4	7	71.79	41.50	56.65	6.07
2003	4	8	70.79	41.67	56.23	6.07
2003	4	9	72.08	42.58	57.33	6.07
2003	4	10	71.67	42.76	57.21	6.07
2003	4	11	68.42	40.13	54.27	6.10
2003	4	12	66.76	39.14	52.95	6.55
2003	4	13	66.00	38.19	52.10	6.85
2003	4	14	66.58	38.71	52.65	6.98
2003	4	15	67.13	39.33	53.23	6.99
2003	4	16	68.13	39.67	53.90	7.04
2003	4	17	67.67	39.38	53.52	7.11
2003	4	18	68.46	39.63	54.04	7.12
2003	4	19	69.25	39.63	54.44	7.12
2003	4	20	66.71	38.58	52.65	7.12

Figure2.3. Outputs of the **WeatherInfo Web Application**.

2.4 WeatherInfo Web Service

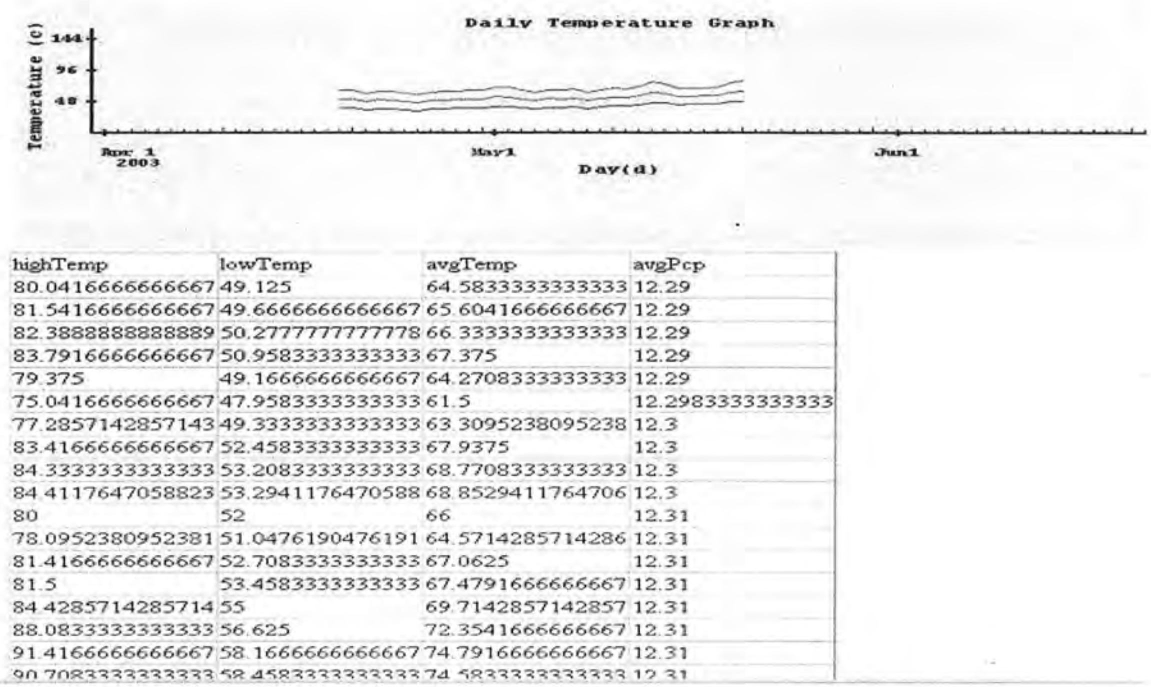
A Web service is a remotely-accessible application whose interface is defined in XML [3]. A client software can interact with a Web service through method calls supported by a proxy object. XML-based messages are passed by SOAP.

The **WeatherInfo Web Service** component of the **WeatherInfo** system is implemented with ASP.NET. This component returns weather information as graphs and tables. A Web service client can obtain those data by calling the following methods: `ReadTempGraphByDay()`, `ReadTempGraphByMonth()`, `ReadTempGraphByYear()`, `ReadPcpGraphByDay()`, `ReadPcpGraphByMonth()`, `ReadPcpGraphByYear()`, `ReadXMLGraphByHour()`, `ReadXMLGraphByDay()`, `ReadXMLGraphByMonth()`, and `ReadXMLGraphByYear()`.

2.5 WeatherClient

In order to test the **WeatherInfo Web Service**, a Web application named **WeatherClient** was created. **WeatherClient** invokes methods provided by **WeatherInfo Web Service** and displays the data returned by those methods.

In Figure 2.4, it shows a graph and a table created by **WeatherInfo Web Serivce** from the data returned by method `ReadTempGraphByDay()` and `ReadXMLByDay()`.



highTemp	lowTemp	avgTemp	avgPcp
80.0416666666667	49.125	64.5833333333333	12.29
81.5416666666667	49.6666666666667	65.6041666666667	12.29
82.3888888888889	50.2777777777778	66.3333333333333	12.29
83.7916666666667	50.9583333333333	67.375	12.29
79.375	49.1666666666667	64.2708333333333	12.29
75.0416666666667	47.9583333333333	61.5	12.2983333333333
77.2857142857143	49.3333333333333	63.3095238095238	12.3
83.4166666666667	52.4583333333333	67.9375	12.3
84.3333333333333	53.2083333333333	68.7708333333333	12.3
84.4117647058823	53.2941176470588	68.8529411764706	12.3
80	52	66	12.31
78.0952380952381	51.0476190476191	64.5714285714286	12.31
81.4166666666667	52.7083333333333	67.0625	12.31
81.5	53.4583333333333	67.4791666666667	12.31
84.4285714285714	55	69.7142857142857	12.31
88.0833333333333	56.625	72.3541666666667	12.31
91.4166666666667	58.1666666666667	74.7916666666667	12.31
90.7083333333333	58.4583333333333	74.5833333333333	12.31

Figure 2.4. A Graph and a table obtained from the **WeatherInfo Web Service**.

3. Design and Implementation of WeatherInfo

The **WeatherData** database, **WeatherDataCollector**, **WeatherInfo Web Application**, **WeatherInfo Web Service**, and **WeatherClient** are the major components of the **WeatherInfo** system. In this section, we describe the implementation details of these components. **WeatherInfo** is implemented with ASP.NET, including ADO.NET.

3.1 WeatherData Database

The **WeatherData** database has five tables. Figure 3.1 through 3.5 lists the attributes of these tables. In these tables, "PK" indicates a primary key, and "FK" is a foreign key.

Attribute Name	Description	Data Type	Key
`cityID`	Identification number of a city	int (4)	PK
`cityName`	Name of the city	varchar (50)	
`state`	Name of state	char (2)	
`weatherURL`	URL address of weather station of city	char (200)	

Figure 3.1. The Attributes of Table `City`.

Figure3.1 gives the attributes of table `City`. The weather data values are stored in table `HourlyWeather`, `DailyWeather`, `MonthlyWeather`, and `YearlyWeather`. The `City` table and others are related by `cityID`.

Attribute Name	Description	Data Type	Key
weatherTime	Time value of the weather	datetime (8)	PK
avgPcp	Average precipitation value	float (8)	
highTemp	Highest temperature value	float (8)	
lowTemp	Lowest temperature value	float (8)	
avgTemp	Average temperature value	float (8)	
cityID	Identification number of a city	int (4)	PK, FK

Figure 3.2. The Attributes of Table HourlyWeather.

Attribute Name	Description	Data Type	Key
weatherTime	Time value of the weather	datetime (8)	PK
avgPcp	Average precipitation value	float (8)	
highTemp	Highest temperature value	float (8)	
lowTemp	Lowest temperature value	float (8)	
avgTemp	Average temperature value	float (8)	
cityID	Identification number of a city	int (4)	PK, FK

Figure 3.3. The Attributes of Table DailyWeather.

Attribute Name	Description	Data Type	Key
`weatherTime`	Time value of the weather	datetime (8)	PK
`avgPcp`	Average precipitation value	float (8)	
`highTemp`	Highest temperature value	float (8)	
`lowTemp`	Lowest temperature value	float (8)	
`avgTemp`	Average temperature value	float (8)	
`cityID`	Identification number of a city	int (4)	PK, FK

Figure3.4. The Attributes of Table `MonthlyWeather.`

Attribute Name	Description	Data Type	Key
`weatherTime`	Time value of the weather	datetime (8)	PK
`avgPcp`	Average precipitation value	float (8)	
`highTemp`	Highest temperature value	float (8)	
`lowTemp`	Lowest temperature value	float (8)	
`aAvgTemp`	Average temperature value	float (8)	
`cityID`	Identification number of a city	int (4)	PK, FK

Figure3.5. The Attributes of Table `YearlyWeather.`

3.2 WeatherDataCollector

The **WeatherDataCollector** periodically extracts weather information from the Web site of the U.S. National Weather Service and stores those data in the **WeatherData** database. The **WeatherDataCollector** is implemented as a class `DataService.` The data collection is performed in the following three steps:

(1) The weather information is retrieved from the Website by an instance of .NET class `WebRequest.`

(2) The desired data in the retrieved pages are extracted with regular expressions and then converted into XML.

(3) A Transact SQL function `OPENXML` and SQL stored procedures insert the data in XML representation into the **WeatherData** database.

The scripts performing the above operations are stored in the MSSQL database server. They are executed by the MSSQL job scheduler.

3.2.1 Retrieving Weather Data

The **WeatherDataCollector** uses .NET class `WebRequest` to retrieve weather data. The source code performing this operation is given in Figure 3.6.

```
protected ArrayList web_site = new ArrayList();
private StreamReader objReader;
for (int i = 0; i < web_site.Count; i++){

     string sURL = “”;
     sURL = web_site[i].ToString();
     WebRequest wrGETURL;
     wrGETURL = WebRequest.Create(sURL);
     Stream objStream;
     objStream = wrGETURL.GetResponse().GetResponseStream();
     objReader = new StreamReader(objStream);
                        . . .
}
```

Figure 3.6. Retrieving weather data with class `WebRequest`.

`WebRequest` is a class in the namespace of `System.Net`. It encapsulates a HTTP request. The request can be sent from an application, such as **WeatherDataCollector,** to a particular URL, such as the Web site of the U.S. National Weather Service. The method `Create()` of `WebRequest` creates a new `WebRequest` object and initializes it with the URL of the target Website.

After the target page is obtained with `wrGETURL`, `Stream` object `objStream` is created for reading the target page. Finally, the weather data is loaded from `objStream` to `objReader`.

3.2.2 Regular Expressions

A regular expression allows us to search for a string and replace it with another string [6]. In our project, class Regex and its methods Replace() and Split(), are used to find strings representing weather data and convert them into XML. This XML representation is used by the SQL function OPENXML to insert the weather data into the database.

Several rules for regular expressions are employed to handle occasional incorrect formats that appear on the Web pages obtained from the Website of the U.S. National Weather Service. For instance, <tr> is sometimes be incorrectly displayed as <tr, even though a browser displays the correct information. An example of regular expressions that extract the desired weather data is shown in Figure 3.7.

```
string sLine = "";
while (sLine = objReader.ReadLine()) != null){

    if (sLine.Length >120 && sLine.Length < 160){
    sLine = Regex.Replace(sLine, @"<TD NOWRAP>. * Gust ", "");
    sLine = Regex.Replace(sLine, @"</TD><TD
                                 NOWRAP>.*<TDNOWRAP>", " 0");
    sLine = Regex.Replace(sLine, @"<TD NOWRAP>.*<TD NOWRAP>",
                                  " 0");
    sLine = Regex.Replace(sLine, @"<TD>", "");
    sLine = Regex.Replace(sLine, @"</TD>", "");
    sLine = Regex.Replace(sLine, @"<TR>", "");
    sLine = Regex.Replace(sLine, @"</TR>", "");
    sLine = Regex.Replace(sLine, @"<TD VALI. *>", "");
                   .    .    .
}
```

Figure 3.7. Regular expressions for extracting weather data.

In addition, other regular expressions are used to detect missing weather data. The detected missing data will be replaced with the most current data to produce a best estimate. If we couldn't find the correct value of the most current data, the detected missing data will be replaced by an empty string. The procedure to replace missing data is performed by looping through all the data record whenever **WeatherDataCollector** imports the data from the Website. The code that detects and replaces missing high temperature values is shown in Figure 3.8.

```
string[] results;
string prevHighTemp = "";
protected ArrayList input_line = new ArrayList();
for(int j = input_line.Count-1; j >= 0; j--){
     if(input_line[j].ToString() != null &&
     input_line[j].ToString()!= " "){
     results = Regex.Split(input_line[j].ToString(), @" ");
     string strHighTemp = results[colHighTemp];
     string[] nextResults;
     if (strHighTemp.StartsWirh("M")){
          string nextHighTemp = "M";
          int next = j - 1;
          while (nextHighTemp.StartsWith ("M") && next >= 0){
          nextResults = Regex.Split(input_line[next].ToString(),
          @" ");
          nextHighTemp = nextResults[colHighTemp];
          next--;
          }
          if (next < 0 && nextHighTemp.StartsWith("M"))
               strHighTemp = preHighTemp;
          else
               strHighTemp = nextHighTemp;
     }
                    .    .    .
   }
}
```

Figure 3.8. Using regular expression to detect missing high temperature value.

3.2.3 Inserting Weather Data

In order to perform data insertion, we created stored procedure read_xml as Figure 3.9. This procedure uses OPENXML, sp_xml_preparedocument, and sp_xml_removedocument. sp_xml_preparedocument and sp_xml_removedocument are system stored procedures of MSSQL Server 2000.

```
CREATE PROCEDURE read_xml
     @strXML ntext
AS
DECLARE @iDoc int
EXECUTE sp_xml_preparedocument @iDoc OUTPUT, @strXML
INSERT INTO HourlyWeather
     SELECT * FROM OPENXML(@iDoc, '/NewDataset/WeatherData', 2)
     WITH HourlyWeather as x
          WHERE (x.weatherTime NOT in (SELECT weatherTime FROM
          HourlyWeather and x.CityID in (SELECT CityID FROM
          HourlyWeather)
EXEC sp_xml_removedocument @iDOc
GO
```

Figure 3.9. Stored procedure read_xml.

In read_xml, sp_xml_preparedocument converts weather data in XML to strXML, and treats the strXML as a string input parameter. Then the procedure returns strXML as a handle iDoc to an internal representation. When the handle is passed to OPENXML, OPENXML retrieves weather data as a rowset. The SELECT statement retrieves all the columns in this rowset, so that INSERT statement can insert the weather data into the **WeatherData** database.

The stored procedure sp_xml_preparedocument loads and stores the XML document in the SQL Server cache. Once the internal representation of the weather data in XML is no longer needed, the memory is released by system stored procedure sp_xml_removedocument.

As read_xml inserts multiple rows of weather data into a database table in the batch mode, it can insert the data faster than when those data are inserted by SQL INSERT statement one at a time.

read_xml is executed in the **WeatherDataCollector** as shown in Figure 3.10.

```
sqlcmd.Connection = sqlconn;
sqlcmd.CommandType = CommandType.StoredProcedure;
sqlcmd.CommandText = "read_xml";
sqlcmd.ExecuteNonQuery();
```

Figure 3.10. Executing the read_xml stored procedure.

3.2.4 Updating the Database

In order to insert data into **WeatherData** periodically, the SQL job scheduler is set up. The scheduled jobs are written as SQL statements and are executed daily, monthly or yearly. The SQL statement executed daily is shown in Figure 3.11.

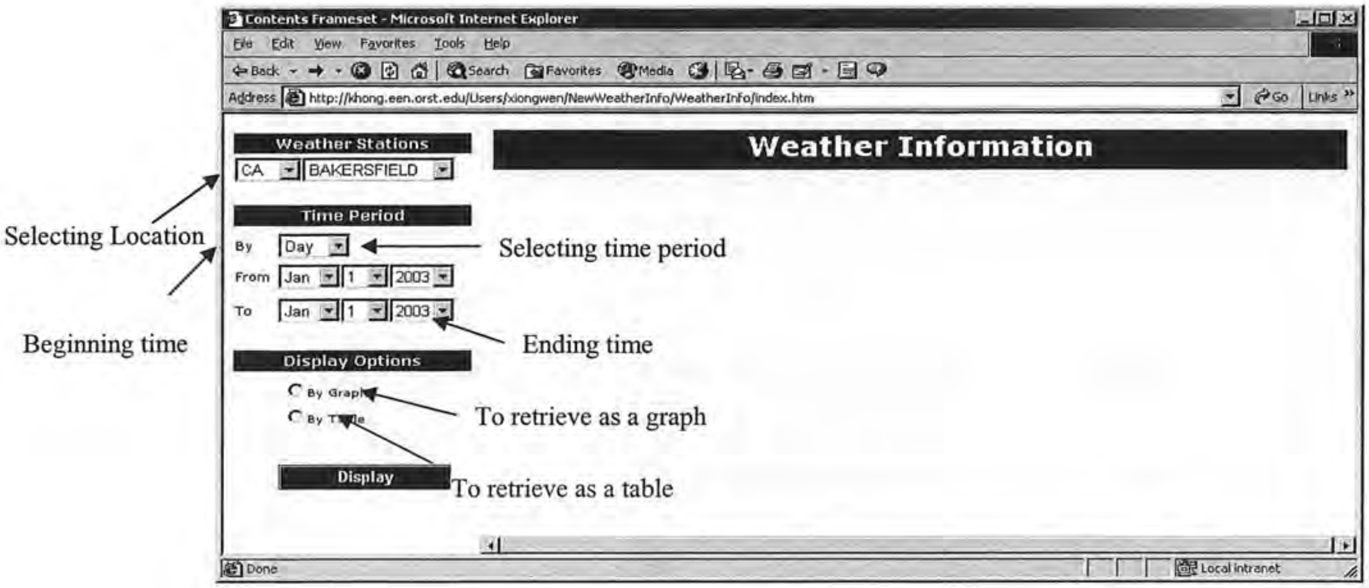
```
insert into DailyWeather
select Top 100 PERCENT HourlyWeather.cityID
       min(HourlyWeather.weatherTime) as weatherTime,
       max(HourlyWeather.HighTemp) as HighTemp,
       min(HourlyWeather.LowTemp) as LowTemp,
       avg(HourlyWeather.AvgTemp) as AvgTemp,
       avg(HourlyWeather.AvgPcp) as AvgPcp, from HourlyWeather,
       DailyWeather
where HourlyWeather.cityID = DailyWeather.cithID
group by HourlyWeather.cityID, Year(HourlyWeather.weatherTime),
       Month(HourlyWeather.weatherTime),
       Day(HourlyWeatherTime.weatherTime)
order by HourlyWeather.cityID, Year(HourlyWeather.weatherTime),
       Month(HourlyWeather.weatherTime),
       Day(HourlyWeatherTime.weatherTime)
```

Figure 3.11. The SQL statement for daily database update.

Monthly and yearly weather data are calculated similarly by stored SQL statements executed by the job scheduler.

3.3 WeatherInfo Web Application

WeatherInfo Web Application dynamically produces Web pages for weather data. These data are displayed either as a table or a graph, according to the requests submitted from Web browsers. A Web page generated by **WeatherInfo Web Application** is shown in Figure 3.12.



Address http://khong.een.orst.edu/Users/xiongwen/NewWeatherInfo/WeatherInfo/index.htm

Weather Stations: CA, BAKERSFIELD

By: Day

From: Jan 1 2003

To: Jan 1 2003

Figure 3.12. A Web page for retrieving weather data.

WeatherInfo Web Application consists of two components: **Graph Drawer** and **Weather Display**. The **Graph Drawer** component draws a graph for the weather data requested. **Weather Display** sends the GIF image of a graph and the data table back to the Web browser. In the following subsection we discuss the implementation details of these components.

3.3.1 The Graph Drawer Component

Graph Drawer uses classes in the .NET Framework to generate graphs for weather data. **Graph Drawer** consists of classes: `Recorder` and `LineRecorder`, where `LineRecorder` is inherited from `Recorder`. We will now describe the fields and the methods of these two classes.

Class `Recorder`

This class can be used to draw Line, text, rectangle graphs.

Fields

`ArrayList recorderData`

The values of the points needed to draw a graph are stored in this field. A point should be an instance of class `Point`.

`double xAxisLabelUnit`

This field specifies the value of tick marks interval along the X axis.

`double numDataPointsOnX`

This field gives the number of data points to draw on the X axis.

`double yAxisLabelUnit`

This field specifies the value of tick marks interval along the Y axis.

`string title`

The title of the graph is assigned to this field.

`string xAixsTitle`

The label for the X axis is assigned to this field.

`string yAixsTitle`

The label for the Y axis is assigned to this field.

`Color recorderColor`

This field defines the color of the graph.

`bool ifNegativeValues`

The value of `ifNegativeValues` indicates whether the data for the graph contains a negative value or not.

```
string fromTime
```

The start time submitted by a Web browser is stored in this field.

Constructor

```
public Recorder(ArrayList recorderData,
    double xAxisLabelUnit, double numDataPointsPerPixelOnX,
    double yAxisLabelUnit, string title, string xAxisTitle,
    string yAxisTitle, Color recorderColor,
    bool ifNegativeValues, int type, string fromTime)
```

Initializes the fields of a `Recorder` object.

```
public Recorder(ArrayList recorderDataMin,
    ArrayList recorderDataAvg, ArrayList recordDataMax,
    double numDataPointsPerPixelOnX, double yAxisLabelUnit,
    string title, string xAxisTitle, string yAxisTitle,
    Color recorderColor, bool ifNegativeValues, int type,
    string fromTime)
```

Initializes the fields of a `Recorder` object.

Methods

```
public void drawAxisSystem(Graphics g)
```

Draws the title and the axes together.

```
public void drawTitle(Graphics g)
```

Draws the title for the graph.

```
public void drawAxes(Graphics g)
```

Draws the X axis and the Y axis for the graph.

`public void drawTicksAndLabelsOnY(Graphics g)`

Draws the tick marks and the labels along the Y axis.

`public void drawTicksAndLabelsOnX(Graphics g)`

Draws the tick marks and the labels along the X axis.

`public static string MonthTransferChar(string integer)`

Converts the integer representing a month to a string.

`public static string MonthTransferInt(string month)`

Converts the character representing a month to an integer.

`public static int EndDayofMonth(int month, int year)`

Returns the number of days for a month specified.

`public void draw (Graphics g)`

Draws a graph by calling `drawAxisSystem(Graphics g)` and `drawRecorder(Graphics g)`.

`public void drawTogether (Graphics g)`

Draws a graph by calling `drawAxisSystem(Graphics g)` and `drawRecorderTogether(Graphics g)`.

`virtual public void drawRecorder(Graphics g)`

The child class of `Recorder` inherits this method, which draws a graph for `recorderData`.

`virtual public void drawRecorderTogether(Graphics g)`

The child class of `Recorder` inherits this method which draws a graph for `recorderDataMin`, `recorderDataAvg`, and `recorderDataMax`.

Class LineRecorder

Class LineRecorder is a child class of class Recorder. An instance of this class can be used to draw a line graph. There are no addition of fields other than those in class Recorder.

Constructor

```
public LineRecorder(ArrayList recorderData,
                    double xAxisLabelUnit, double
                    numDataPointsPerPixelOnX, double
                    yAxisLabelUnit, string title, string
                    xAxisTitle, string yAxisTitle, Color
                    recorderColor, bool ifNegativeValues,
                    int type, string fromTime)
```

Initializes the fields of a LineRecorder object.

```
public LineRecorder(ArrayList recorderDataMin,
                    ArrayList recorderDataAvg, ArrayList
                    recordDataMax, double
                    numDataPointsPerPixelOnX, double
                    yAxisLabelUnit, string title, string
                    xAxisTitle, string yAxisTitle, Color
                    recorderColor, bool ifNegativeValues,
                    int type, string fromTime)
```

Initializes the fields of a LineRecorder object.

Methods

```
public void drawRecorder(Graphics g)
```

Draws the record on the drawing object.

```
public void drawRecorderTogether(Graphics g)
```

Draws the record on the drawing object.

3.3.2 The Weather Display Component

The **Weather Display** component is responsible for generating a graph and a table requested by a Web browser.

3.3.2.1 Displaying Graph

Weather Display calls the **Graph Drawer** component to draw the requested graph with the data retrieved from **WeatherData**. After drawing the graph, the **Graph Drawer** passes control back to **Weather Display**. The graph is sent to the browser as part of the resulting page. Web server controls `Image`, `Panel`, and `DataGrid` are used by this component. The implementation of **Graph Drawer** is shown in Figure 3.13.

```
ArrayList dataArray = new ArrayList();
SqlDataReader reader;
While (reader.Read())
dataArray.Add(new Point
                  (reader.GetInt32(0),reader.GetDouble(1)));

dataArray.TrimToSize();
                  .    .    .

Recorder recorder=new LineRecorder(dataArray,
                      getXLabelUnit(dataArray),
                      getNumPointPerPixel(dataArray),
                      getYLabelUnit(dataArray),
                      "Daily Precipitation(mm)",
                      Color.BlueViolet,
                      ifNegativeValues(dataArray),
                      type,fromTime)
Bitmap bitmap = new Bitmap(700, 240);
Graphics g = Graphics.FromImage(bitmap);
recorder.draw(g);
Resonse.ContentType = "image/gif";
Bitmap.Save(Response.OutputStream, ImageFormat.Gif);
```

Figure 3.13. Generating an image by calling **Graph Drawer**.

The path for the image to be displayed by a Web browser is specified in the ImageUrl property. The implementation of ImageUrl is shown in Figure 3.14.

```
System.Web.UI.WebControls.Image graphImage;
graphImage = new System.Web.UI.WebControls.Image();
graphImage.ImageUrl =
          "StreamGraph.asp?getGraph=1&cityID="+cityID
          +"&fromTime="+fromTime+"&toTime="+toTime;
```

Figure 3.14. Displaying an image by ImageUrl.

3.3.2.2 Displaying Table

Class DataSet and class DataAdapter displays a table. The DataSet contains the weather data retrieved from the database which provides a relational model that is independent of the data source [1]. The DataAdapter provides the interface between the DataSet and the data source.

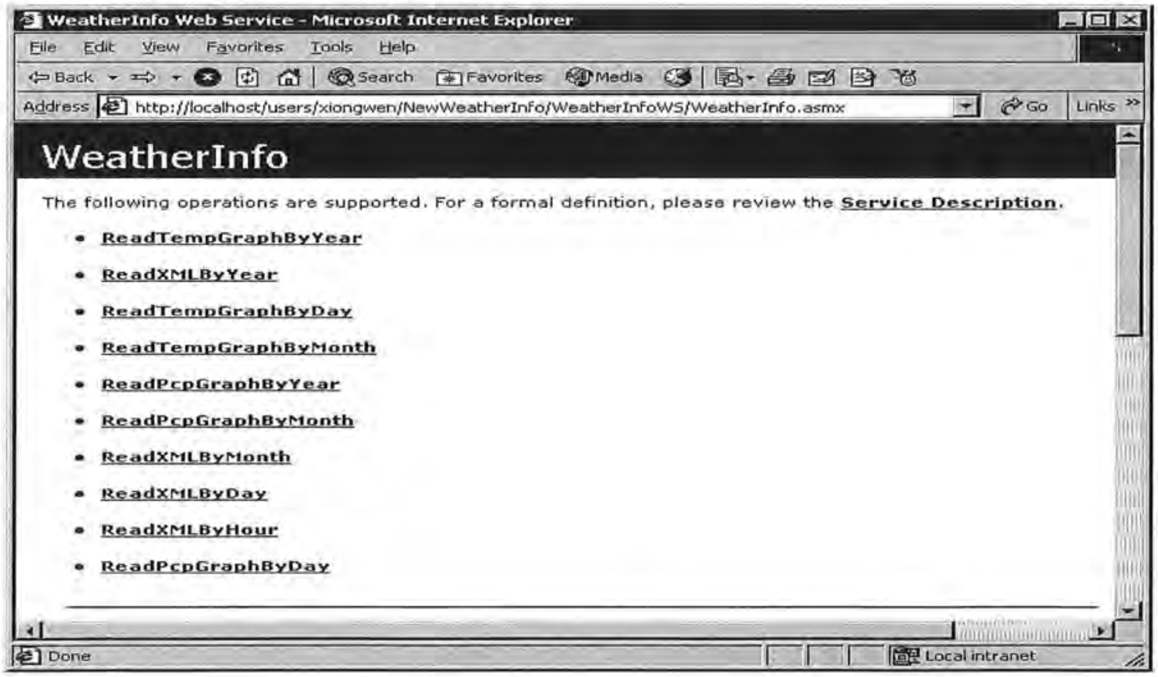
String selQry contains SQL select statement for collecting the data from the database, and weatherConn is the SQL connection object. The DataAdapter instance is initialized with these two arguments. The Fill() method of the DataAdapter adds the rows of weather data to the DataSet by using name weatherdata. The MyDataGrid, which is a WebControl DataGrid, binds the data source specified by the DataSource_property to it. The code for displaying a data table is shown in Figure 3.15.

```
System.Web.UI.WebControls.DataGrid MyDataGrid;
SqlDataAdapter adpt = new
                    SqlDataAdapter(selQry, weatherConn);
DataSet ds = new DataSet();
adpt.Fill(ds, "weatherdata");
DataTable dataTable = ds.Tables[0];
MyDataGrid.DataSource = dataTable;
MyDataGrid.DataBind();
```

Figure 3.15. Loading data into a DataSet.

3.4 WeatherInfo Web Service

A Web service provides a convenient means for exchanging data among different applications on various platforms. **WeatherInfo Web Service** allows its clients to acquire the weather data through this Web service. When a set of data is requested, the Web service returns it as XML format. If a graph of the data is requested, the Web service returns the image in binary format. The methods supported by the interface of **WeatherInfo Web Service** can be retrieved as a Web page as shown Figure 3.16.


WeatherInfo Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Go Links
Done
Local intranet

Address http://localhost/users/xiongwen/NewWeatherInfo/WeatherInfoWS/WeatherInfo.asmx

WeatherInfo

The following operations are supported. For a formal definition, please review the **Service Description**.

- **ReadTempGraphByYear**
- **ReadXMLByYear**
- **ReadTempGraphByDay**
- **ReadTempGraphByMonth**
- **ReadPcpGraphByYear**
- **ReadPcpGraphByMonth**
- **ReadXMLByMonth**
- **ReadXMLByDay**
- **ReadXMLByHour**
- **ReadPcpGraphByDay**

Figure 3.16. WeatherInfo Web Service.

3.4.1 Returning a Table of Weather Data

The outputs of methods `ReadXMLByHour()`, `ReadXMLByDay()`, `ReadXMLByMonth()`, and `ReadXMLByYear()` are written in XML. The code writing the output in XML is shown in Figure 3.17.

```
DataAdapter adpt = new DataAdapter;
StringBuilder sbXML = new StringBuilder();
adpt.Fill(ds, "WeatherData");
ds.WriteXml(new XmlTextWriter(new
StringWriter(sbXML)),XmlWriteMode.WriteSchema);
```

Figure 3.17. Producing table of weather data in an XML.

When a Web service client invokes one of above methods, the results is returned in XML. The XML data binds with the `DataSet` created at the client side.

3.4.2 Returning a Graph of Weather Data

In order to pass the image to a client, **WeatherInfo Web Service** converts the graph into binary format. The conversion is performed by the following steps: (1) the graph is retrieved from its URL (line 2 - line 5 in Figure 3.18), (2) `ArrayList imgArray` is declared to hold the image (line 6 - 11), and (3) the image is converted into bytes and passed to array `imageBytes` (line 12 - line 14). The conversion of an image into image bytes is shown in Figure 3.18.

```
string sURL;
sURL = "http://localhost/users/xiongwen/
        NewWeatherInfo/WeatherInfo/
        StreamGraph.asp?cityID=" +cityID +"&getGraph="
        +(int)type + "&fromTime="+startTime + "&toTime="
        +endTime;
WebRequest wrGETURL;
wrGETURL = WebRequest.Create(sURL);
Stream urlStream = wrGETURL.GetResponse().
                              GetResponseStream();
int curInt = urlStream.ReadByte();
ArrayList imgArray = new ArrayList();
while (curInt != -1){
     imgArray.Add(Convert.ToByte(curInt);
     curInt = urlStream.ReadByte();
}
Byte tmpByte = 0;
Byte[] imageBytes = (Byte[])imgArray.ToArray
                         (tmpByte.GetType());

return imageBytes
```

Figure 3.18. Convering a graph into bytes.

Six functions `ReadTempGraphByDay()`, `ReadTempGraphByMonth()`, `ReadTempGraphByYear()`, `ReadPcpGraphByDay()`, `ReadPcpGraphByMonth()`, and `ReadPcpGraphByYear()` have been defined to return a graph of a set of weather data requested. When a Web service client invokes each of these methods, the image in bytes is returned to the Web service client.

3.5 WeatherClient

WeatherClient is a Web service client that verifies a weather data table and a graph from **WeatherInfo Web Service**.

The MemoryStream instance in **WeatherClient** holds the binary image retrieved from the **WeatherInfo Web Service**. If a client wants to obtain the graph in one of the commonly used graphics formats such as a GIF, the binary image must be converted into that format. The implementation is shown in Figure 3.19.

```
WeatherInfoWS ws = new WeatherInfoWS();
Byte[] imageBytes = ws.ReadTempGraphByDay(city, state,
                                        startDate, endDate);
MemoryStream ms = new MemoryStream(imageBytes);
Response.ContentType = “image/gif”;
ms.WriteTo(Response.OutputStream);
```

Figure 3.19. Obtaining a graph from WeatherInfo Web Service.

The ContentType of Response specifies the image type as GIF. The OutputStream property of Response is used to send the image to a Web browser. In this approach the image is not saved as a file.

4. Conclusions and Future Works

As the Internet has become a major means for collecting information, we created the **WeatherInfo** system that collects weather data and displays them as graphs and tables. The system gathers the desired weather data, stores them in a relational database, retrieves the requested data from the database, and produces graphs and tables to be sent to Web browsers.

The implementation of Web services is appealing for Web-based applications on different platforms because Web services provide a common procedural interface to be shared by other applications. Web-based applications use this interface to access a Web service. The **WeatherInfo** system provides a Web service for such weather data as temperatures and precipitation at different locations.

As the formats of graphs, only line graphs are currently supported. In the future other formats such as bar charts and pie drafts may be supported. **WeatherInfo** may be further extended to support other data such as winds, tides, and humidity.

Reference:

[1] C.J.Date, An introduction to Database Systems, 6 Edition, Addison-Wesley, 1995

[2] Raghu Ramakrishnan, Database Management Systems, McGraw-Hill Companies, 1997

[3] Steven A. Smith, ASP.NET, Que, 2001

[4] Scott Mitchell, Steve Walther, ASP.NET: Tips, Tutorials, and Code

[5] Michael Stiefel & Robert J. Oberg, Application Development using C# and .Net, PHPTR, 2002

[6] Steve Mansour, A Tao of Regular Expressions, 1999