WEATHERINFO: A WEB-BASED WEATHER DATA CAPTURE SYSTEM

JEFFREY J. STEINER,* TOSHIMI MINOURA, AND WEN XIONG

Abstract
Past and current weather data are needed for many agriculture and natural resource management applications. With the rapid development and popularity of web technology solutions, the Internet has become a major means for collecting and presenting information. The WeatherInfo system prototype was developed as a framework platform to provide end users a way to automatically retrieve weather information over the Internet to support simulation models or other applications needing these kinds of data. To make data retrieval and manipulation easy by users, WeatherInfo was implemented with a relational database to store weather data extracted directly from the U.S. National Weather Service (NWS) website. The database is automatically updated daily. Based on daily updated data, weekly and monthly data are automatically calculated using the daily updated data. The system also includes a Web service implemented in ASP.NET to support data retrieval by clients who desire weather data in either graphic or table form. The system framework was set up to demonstrate data retrieval over the web from the NWS website (http://www.nws.noaa.gov) for 19 locations in California for periods from 1 February 2003 to the present date. The WeatherInfo system demonstration can be accessed at: http://yolo.een.orst.edu/WeatherInfo/index.htm (verified 22 Oct. 2004).

WITH THE RAPID DEVELOPMENT of computer science technology and the popularity of web solutions, the Internet has become a major means for collecting and displaying information. Many computer applications that help users make decisions concerning the management of natural resource quality or the scheduling of agricultural practices such as irrigation and pesticide applications have weather data requirements. The NWS collects weather data from a wide variety of sources and provides the content in textual form on the Web. However, these data cannot be directly accessed live from the NWS website, so users must request copies or manually copy values. The WeatherInfo system was developed to directly capture past and current weather over the Internet from websites that can then be used by application programs requiring these kinds of data. The NWS website was used to demonstrate the WeatherInfo system prototype.

WeatherInfo gathers desired weather data from a website, stores them in a relational database, retrieves the requested data from the database, and sends them to a client site. Based on a request submitted from a web browser, the system allows a user to obtain the weather data for use in simulation models or to view either as a graph or table on a web page. In a request, the city, time period, and time interval can be specified for the data to be retrieved. The data demonstrated are (i) date (year, month, and day), (ii) temperatures (high, low, and average), and precipitation amount.

The system web service provides a procedural interface for remote client programs. The WeatherInfo Web Service is a .NET web service that supports Simple Object Access Protocol (SOAP). The Web service interface is described 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, thus providing a common platform for operations.

WeatherInfo System Overview

The WeatherInfo system allows end users to easily obtain weather data from a website (e.g., NWS website) by using a web service that supports data exchange. Client programs such as simulation models can retrieve data by a procedural interface provided by the web service. The system consists of five modules: (i) WeatherDataCollector, (ii) WeatherData, (iii) WeatherInfo Web Application, (iv) WeatherInfo Web Service, and (v) WeatherClient (Fig. 1) that are described below.

WeatherDataCollector
In this demonstration, WeatherDataCollector gathers weather information from the website run by the NWS (http://www.nws.noaa.gov). The weather data for major cities of every state are available from this site. Our framework demonstration uses 19 sites in California. WeatherDataCollector uses .NET class WebRequest to get weather records with HTTP requests. The retrieved data are extracted with regular expressions and converted into an Extensible Markup Language (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 Windows 2000 job scheduler.

WeatherData
A relational database named WeatherData was developed with Microsoft SQL Server and stores the weather data extracted by WeatherDataCollector. The database contains five tables: City, HourlyWeather, DailyWeather, MonthlyWeather, and YearlyWeather. The schema diagram for this database is given in Fig. 2. An explanation of the notation used by schema diagrams is found in many references (e.g., Ramakrishnan, 1997).

WeatherInfo Web Application

To allow web users direct viewing of the weather data managed by WeatherInfo, a web application was implemented to generate graphic or table views of weather data for each request submitted from a web browser.

WeatherInfo Web Service

A web service is a remote-accessible application whose interface is defined in XML (Smith, 2001). A client’s software can interact with a web service through method calls supported by a proxy object. Messages based in XML are passed by SOAP. The WeatherInfo Web Service component of the WeatherInfo system is implemented with ASP.NET (Smith, 2001; Mitchell et al., 2002; Stiefel and Oberg, 2002) to return weather information as graphs or tables. A web service client can obtain data by calling the following methods: ReadTempGraphByDay(), ReadTempGraphByMonth(), ReadPcpGraphByYear(), ReadPcpGraphByDay(), ReadPcpGraphByMonth(), and ReadXMLGraphByYear().

WeatherClient

To test the WeatherInfo Web Service, a web application named WeatherClient was created to invoke methods provided by WeatherInfo Web Service and display data returned by those methods.

Design and Implementation

The five WeatherInfo system components are implemented with ASP.NET, including ADO.NET. The programming language used was C#.

WeatherData Database

The WeatherData database has five tables where the data values are stored. The table City stores the location attributes and URL address where the weather data can be found for the geographic location (Table 1) the tables HourlyWeather, DailyWeather, MonthlyWeather, and YearlyWeather contain the weather data for different time periods (Table 2). The City and weather data tables are related by cityID.

WeatherData Collector

The WeatherDataCollector periodically extracts weather information from the website of the NWS and stores those data in the WeatherData database. The WeatherDataCollector is implemented as a class DataService. Data collection is performed in 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, and
Table 1. Attributes of table City.

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
<th>Data type (length)</th>
<th>Key‡</th>
</tr>
</thead>
<tbody>
<tr>
<td>cityID</td>
<td>city identification number</td>
<td>integer (4)</td>
<td>PK</td>
</tr>
<tr>
<td>cityName</td>
<td>city name</td>
<td>variable length</td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>state name</td>
<td>character string (50)</td>
<td></td>
</tr>
<tr>
<td>weatherURL</td>
<td>URL address for city weather station</td>
<td>character (200)</td>
<td></td>
</tr>
</tbody>
</table>

† PK indicates the primary key, which is a standard database definition (Date, 1995).

3. A Transact SQL function OPENXML and SQL-stored procedures insert the data in XML representation into the WeatherData database.

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

Retrieving Weather Data

The WeatherDataCollector uses .NET class Web Request to retrieve weather data. The source code performing this operation is given in Fig. 3a. Web Request is a class in the namespace of System.NET that encapsulates an HTTP request. The request can be sent from an application, such as WeatherData Collector, to a particular URL, such as the NWS website. 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.GetResponse, Stream object objStream is created to read the target page. Finally, the weather data are loaded from objStream to objReader.

Regular Expressions

A regular expression allows us to search for a string and replace it with another string (Mansour, 1999). 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 NWS website. For instance, &lt;tr&gt; is sometimes incorrectly displayed as &lt;tr even though a browser displays the correct information. An example of regular expressions that extract the desired weather data is shown in Fig. 3b.

In addition, other regular expressions are used to detect missing weather data, so detected missing data are replaced by empty strings using a procedure that loops through the record whenever WeatherData Collector imports data from the website.

Inserting Weather Data

Data are inserted using a stored procedure read_xml (Fig. 3c). 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.

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 the 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 (Fig. 3d).

Updating the Database

To periodically add current data in the WeatherData database, the SQL job scheduler was set up for the WeatherInfo system. Scheduled jobs are written as SQL statements that are executed daily, monthly, or yearly. An example of an SQL statement that is executed daily is shown in Fig. 3e. Monthly and yearly weather data are also calculated by stored SQL statements that are executed by the job scheduler.

Table 2. Attributes of weather data HourlyWeather, DailyWeather, MonthlyWeather, and YearlyWeather.

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
<th>Data type (length)‡</th>
<th>Key‡</th>
</tr>
</thead>
<tbody>
<tr>
<td>weatherTime</td>
<td>time value of the weather</td>
<td>datetime (8)</td>
<td>PK</td>
</tr>
<tr>
<td>avgPcp</td>
<td>average precipitation value</td>
<td>float (8)</td>
<td></td>
</tr>
<tr>
<td>highTemp</td>
<td>highest temperature value</td>
<td>float (8)</td>
<td></td>
</tr>
<tr>
<td>lowTemp</td>
<td>lowest temperature value</td>
<td>float (8)</td>
<td></td>
</tr>
<tr>
<td>avgTemp</td>
<td>average temperature value</td>
<td>float (8)</td>
<td></td>
</tr>
<tr>
<td>cityID</td>
<td>identification number of a city</td>
<td>int (4)</td>
<td>PK, FK</td>
</tr>
</tbody>
</table>

‡ Data types datetime, float, and int represent special number, floating point number, and integer, respectively.

§ PK and FK indicate primary and foreign keys, respectively, which are standard database definitions (Date, 1995).
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. Web pages generated by WeatherInfo Web Application are shown in Fig. 4. The 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 the graph and the data table back to the web browser.

WeatherInfo Web Service

A web service provides a convenient way to exchange data among different kinds of applications and across various platforms. WeatherInfo Web Service allows clients to acquire weather data through this web service. When a set of data is requested, the web service returns...
(d) Executing the stored procedure.

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

(e) SQL statement that will update the WeatherData database daily.

```sql
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.cityID
    group by HourlyWeather.cityID, Year(HourlyWeather.weatherTime),
            Month(HourlyWeather.weatherTime),
            Day(HourlyWeather.weatherTime.weatherTime)
order by HourlyWeather.cityID, Year(HourlyWeather.weatherTime),
        Month(HourlyWeather.weatherTime), Day(HourlyWeather.weatherTime.weatherTime)
```

(f) Producing a table of weather data in an XML.

```csharp
1  DataAdapter adpt = new DataAdapter;
2  StringBuilder sbXML = new StringBuilder();
3  adpt.Fill(ds, "WeatherData");
4  ds.WriteXml(new XmlTextWriter(sbXML), XmlWriteMode.WriteSchema);
```

(g) Converting a graph into binary format so an image can be passed to a client.

```csharp
1  string sURL;
2  sURL = "http://localhost/users/xiongwen/WeatherInfo/WeatherInfo/
    StreamGraph.asp?cityID=" +cityID +"&getGraph="
    +(int)type + "&fromTime=" +start_time + "&toTime=" +end_time;
3  WebRequest wrGETURL;
4  wrGETURL = WebRequest.Create(sURL);
5  Stream urlStream = wrGETURL.GetResponse().
    GetResponseStream();
6  int curInt = urlStream.ReadByte();
7  ArrayList imgArray = new ArrayList();
8  while (curInt != -1){
7  imgArray.Add(Convert.ToByte(curInt));
10  curInt = urlStream.ReadByte();
11  }
12  Byte tmpByte = 0;
13  Byte[] imageBytes = (Byte[])imgArray.ToArray
    (tmpByte.GetType());
14  return imageBytes
```

(h) Obtaining a graph from WeatherInfo Web Service.

```csharp
1  WeatherInfoWS ws = new WeatherInfoWS();
2  Byte[] imageBytes = ws.ReadTempGraphByDay(city, state,
    startDate, endDate);
3  MemoryStream ms = new MemoryStream(imageBytes);
4  Response.ContentType = "image/gif";
5  ms.WriteTo(Response.OutputStream);
```

Fig. 3. Examples of C# programming code used to implement the features described in the text.
Fig. 4. Examples of the graphic (top) and table (bottom) web pages used for retrieving and displaying data for the specified geographic location and time period. Top, middle, and bottom broken lines in the graphic output represent high, average, and low daily temperatures.

it in XML format. If a graph of the data is requested, the web service returns the image in binary format. The methods supported by the WeatherInfo Web Service interface can be retrieved as a web page (Fig. 5).

The outputs of methods ReadXMLByHour(), ReadXMLByDay(), ReadXMLByMonth(), and ReadXMLByYear() are written in XML. The code writing the output in XML is shown in Fig. 3f. When a web service client invokes one of the above methods, the results are returned in XML that binds with the DataSet created at the client side.

To pass an image to a client, WeatherInfo Web Service converts the graph into binary format by three steps (Fig. 3g):

1. the graph is retrieved from its URL (Lines 2–5),
2. ArrayList imgArray is declared to hold the image (Lines 611), and
3. the image is converted into bytes and passed to array imageBytes (Lines 12–14).

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.

WeatherClient

WeatherClient is a web service client that verifies a weather data table or 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 (Fig. 3h). 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. The image is not saved as a file in this approach.

Remarks

We created the WeatherInfo system to collect weather data from web pages and display them as graphs and tables. The WeatherInfo Web Service is appealing for web-based applications on different platforms be-
cause a web service provides a common procedural interface that can be shared by multiple applications. Web-based applications can use this interface to access the WeatherInfo Web Service. The WeatherInfo system prototype demonstrated here provides a web service for accessing temperature and precipitation data from 19 different locations in California using the NWS website. This approach can be custom-modified and extended to more locations in other states and used for accessing additional weather parameters such as humidity. Only line graph formats are currently supported.

A tactical utility of WeatherInfo could be demonstrated if used for irrigation or pesticide application timing by downloading the prior day’s data, then using it to run a simulation model, and notifying the user automatically if any changes to the irrigation or pesticide application schedule are recommended.

A limitation of this kind of technology is dependence on the web page being accessed remaining the same. A change in display format will render the WeatherInfo system ineffective for accessing the desired data from a specific website. Another limitation is if NWS were to begin providing its information in machine-readable format (XML); then the features demonstrated by WeatherInfo would not be needed. However, WeatherInfo could still be used to prepare, aggregate, and add value to the raw data supplied by primary providers such as NWS.

Documentation for the WeatherInfo system is available free on request from the corresponding author. The authors ask that when any component parts of the WeatherInfo system are utilized, that citation credit be given. A working demonstration of WeatherInfo can be accessed at http://yolo.een.orst.edu/WeatherInfo/index.htm. Access to this web link depends on the NWS maintaining their site in the configuration used by WeatherInfo.

Acknowledgments

This research, in cooperation with the Yolo County Resource Conservation District, was supported in part by a grant from the CALFED Bay-Delta Restoration Project, Ecosystems Restoration Program (Contract 01-N25). Thanks also to T. Wuttiwat for technical assistance and general computer science management for the CREEDA project.

References

Date, C.J. 1995. An introduction to database systems. 6th ed. Addison-Wesley, Reading, MA.
Smith, S.A. 2001. ASP.NET. Que Publ., Indianapolis, IN.