On-Line City Weather Forecast System

Lin Zhang

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

April 2003

ABSTRACT

On-Line City Weather Forecast System

Lin Zhang

This document presents a case study of implementing a server side Model View Controller (MVC) design pattern using JSP and Servlet technologies. From the example, we can understand how and why this pattern and these technologies work so well when developing server-side Java web applications.

On-Line City Weather Forecast System is a web application that runs on Tomcat web server. Its raw data comes from an existing weather website, *http://weatheroffice.ec.gc.ca*. The system consists of two parts: one for processing weather data, the other for visualizing weather conditions. The primary audience for the system is weather database administrator but is also developed to provide non-administrator ability to view current weather as well as weather forecast for next 5 days. A weather processor is designed to parse and select data from the real-time weather web site and save it in users designated database created by MS Access on the local server. A weather server delivers a subset of this data to the clients in response to a query formulated in a SQL statement.

ACKNOWLEDGEMENTS

I would like to express my appreciation to my supervisor, Professor Peter Grogono. Without his patience and guidance, I could not finish my work. Furthermore, I'd like to thank Professor Shiri. Nematollaah for his constructive remarks.

I am also grateful to my family for their support and encouragement.

Table of Contents

1. Introduction 1

1.1 Overview of the System 1

1.2 Tools Used 2

2. System Requirements and Analysis 3

2.1 Problem Statement 3

2.2 Use Case Analysis 4

2.2.1 User Analysis 4

2.2.2 Use Case Descriptions 5

2.2.3 Use Case Diagram 8

2.3 Non Functional Requirements 9

3. System Design 11

3.1 Design Rationale 11

3.2 System Design 15

3.2.1 Weather Data Processing Subsystem 15

3.2.1.1 Implementation of Associations 16

3.2.1.2 Class Diagram 19

3.2.1.3 Sequence Diagram 21

3.2.1.4 Object Models 22

3.2.2 Weather Viewing Subsystem 24
3.2.2.1 Implementation of Associations 25
3.2.2.2 Sequence Diagram 26

4. Implementation 27

4.1 System Environment 27
4.2 Using JDBC Connection Pool Technology 28
4.3 Inter-Servlet Communication 30
4.4 Implementation of MVC 32

4.4.1 Startup 32
4.4.2 Views 34
4.4.2.1 Index View 34
4.4.2.2 Weather Record List View 35
4.4.2.3 Current City Weather Forecast View 36
4.4.2.4 Weather Detail View 37
4.4.2.5 Next 5-Day Weather Forecast View 37
4.4.2.6 Last 7-Day Weather Forecast View 38

4.4.3 Controller 39
4.4.4 Service 41
4.4.4.1 Add Weather Service 43
4.4.4.1.2 Saving Weather Record 47
4.4.4.2 Query Current Weather Service 47
4.4.4.5 Query Next 5-Day Weather Service 50

5. Conclusion 52

5.1 Summary 52
5.2 Discussion and Future Work 52

References 54

Appendix A: Installation Guide 56
Appendix B: User Manual 58

List of Figures

Figure 2.1: Use Case Diagram 9
Figure 3.1: MVC Architecture for a Web Application 12
Figure 3.2: Modules with Event and Information Flow 16
Figure 3.3: Class Diagram 20
Figure 3.4: Sequence Diagram for Creating a Weather Record 21
Figure 3.5: Modules with Event and Information Flow 25
Figure 3.6: Sequence Diagram for Querying Weather Records 26
Figure 4.1: Server-Side Implementation of MVC 28
Figure 4.2: Interface of Class ConnectionPool 29
Figure 4.3: Index View for DBA 35
Figure 4.4: Weather Record List View 35
Figure 4.5: Current City Weather Forecast View 36
Figure 4.6: Weather Forecast View 37
Figure 4.7: Next 5-Day Weather Forecast View 38
Figure 4.8: Last 7-Day Weather Forecast View 39
Figure A1: Login Page 58
Figure A2: Sign Up Page 59
Figure A3: Index Page for Non-DBA 60
Figure A4: Add Record Page 61
Figure A5: Delete Record Menu 62
Figure A6: Delete Record Page 63
Figure A7: Record List Page 63
Figure A8: Query Record Menu 64
Figure A9: Query Record Page 65
Figure A10: Current Weather Conditions of Selected Cities 66
Figure A11: Any Date Weather Page 67

1. Introduction

1.1 Overview of the System

The On-Line City Weather Forecast System is a request-reply and a subscription system for disseminating and broadcasting real-time weather information [1]. The system consists of client and server that communicate using a HTTP protocol. A weather database that resides at the server side maintains weather observation data such as current weather and next 5-day weather forecast information, which is published by Environment Canada's Green Lane. When a weather client submits a request form to retrieve a subset of these data, the weather server parses the request, queries the weather database, and sends the result back to the client.

In this project, the source of original data comes from web pages of 14 cities in the official weather web site of Environment Canada and the documents of these web pages, written in HTML (Hypertext Markup Language), are plain text files with special tags and some texts describing weather conditions. In order to extract weather data from these web pages, an HTML parser program is designed to take one document as an input file, search for specific text products e.g. temperature, cloud description, wind, pressure, visibility, etc, and store the results including current weather and next 5-day weather forecast information into a weather database on the local server. A weather server is responsible for distributing the weather data in JSP format to the clients.

1.2 Tools Used

The On-Line City Weather Forecast System is developed in the WindowNT/2000 environment. The following tools are used for the system development:

- JSP (Java Server page), Servlet, Java language are used for implementing the web application.
- MS Access 2000 is used for database management including data storage and data retrieval.
- ODBC Data Source Administrator is used for setting up ODBC data source, which is connected to the database on the local server.
- Internet Explorer 5.5 is used as a web browser to support user interface for the web application.

This report is organized as follows: Chapter 1 states overview of the system; Chapter 2 is focusing on system requirements and analysis; Chapter 3 describes system design; Chapter 4 states the implementation details, and conclusions are given in Chapter 5.

2. System Requirements and Analysis

2.1 Problem Statement

There are two main tasks in this system: one is to extract weather information of 14 cities from a Canadian official weather web site www.weatheroffice.ec.gc.ca, and store it into a user-defined weather database; the other task is to publish daily weather conditions of each city on any specified date. The weather database can be maintained manually by Database Administrator or automatically by the system. In order to keep current weather data, the system can update itself 9 times maximum to keep accurate weather information verses other weather systems. The weather database used in the system only keeps newly parsed weather records, formerly stored today's weather records can be overwritten automatically. There are two categories of users in the system. For different users, it implements different functionalities.

Database Administrator

A DBA is a particular user of a database who can login to the On-Line City Weather Forecast System, Weather Data Processing Subsystem, and is allowed to access to the weather database.

- Get current weather data of any selected cites, and save it in the weather database.
- Delete some obsolete weather records from the weather database.
- Query weather records from the weather database on a specified condition.

Common User

A common user is a user who can login to the Weather Viewing Subsystem and view daily weather conditions.

- View today's weather conditions of any selected city.
- View next 5 days weather forecast of any selected city.
- View previous week weather conditions of any selected city.
- Search weather conditions of any selected city on a specified date.

2.2 Use Case Analysis

In order to capture the functionalities of this system, we make full use of use cases [4] and use case diagrams to explain the system. Use case is a sequence of transactions that a user performs with the system. Now we consider the system from two different users' perspectives: the DB Administrator and the common user. According to the circumstances of each user, the system provides some functionalities.

2.2.1 User Analysis

In this system, two categories of users are defined:

1) Database Administrator

   Skills and knowledge:

- Traditional database management (managing and maintenance of database).
- Tomcat web server management. Tomcat from version 3.0 up to current version.
- Understanding the web technology (like HTML, XML etc) and terminology (like Web Portals, ASP, etc).

2) Common client (those who have no right to access to database)

Skills and knowledge:

- User should have some basic knowledge of how to use internet browser (Internet Explorer5.0 up or Netscape6.0)

2.2.2 Use Case Descriptions

2.2.2.1 DB administrator (DBA) logs into the system

DBA opens a browser, and goes to login web page of the On-Line City Weather Forecast System; DBA enters username and password; A checking program is performed to check if the user is a valid DBA; If it is approved, DBA can go to the home page of Weather Data Processing Subsystem.

2.2.2.2 DB administrator creates weather records

DBA logs into the On-Line City Weather Forecast System; DBA chooses "Extract weather" from the menu, an add form is shown on the screen; DBA chooses one or more cities from city list on the screen and presses the button "Go"; an HTML parser program is performed and weather records including temperature, pressure, visibility, wind etc information are created; Weather records are submitted to weather database.

2.2.2.3 DB administrator deletes weather records for a period of time

DBA logs into the On-Line City Weather Forecast System; DBA chooses "Delete Weather" from the menu and three options is shown on the screen; DBA chooses button "Delete Period Weather" and a city list is shown on the screen; DBA chooses one city

and enters starting date and ending date; delete program is performed to list all the records needs to be deleted; DBA confirms to delete chosen record.

2.2.2.4 DB administrator deletes current day's weather records of any cities

DBA logs into the On-Line City Weather Forecast System; DBA chooses "Delete Weather" from the menu and three options is shown on the screen; DBA chooses button "Delete Current City Weather" and a city list is shown on the screen; DBA chooses one or more cities from the city list; a delete program is performed to delete all the current day's weather records of the selected cities.

2.2.2.5 DB administrator queries weather records for a period of time

DBA logs into the system; DBA chooses "Query Weather" from the menu and two options is shown on the screen; DBA chooses button "Query Period Weather" and a city list is shown on the screen; DBA chooses one city and enters starting date and ending date; a query program is performed to list all the relevant weather records on the screen; DBA can views weather detail including wind, pressure, visibility, etc for any specified date.

2.2.2.6 DB administrator queries current day's weather records of any cities

DBA logs into the On-Line City Weather Forecast System; DBA chooses "Query Weather" from the menu and two options is shown on the screen; DBA chooses button "Query Current City Weather" and a city list is shown on the screen; DBA chooses one or more cities from the city list; a query program is performed to show all the current day's weather records of the selected cities.

2.2.2.7 DB administrator logs out the system

DBA logs into the system, DBA chooses Log Out from the menu and a login form is shown on the screen.

2.2.2.8 Common user logs into the system

User opens a browser, and goes to login web page of the system; user enters username and password. A checking program is performed to check if user's information is complete; If it is approved, common user can go to home page of Weather viewing Subsystem, else common user goes to sign up page.

2.2.2.9 new user signs up the system

User opens a browser, clicks "Sign up now" and a sign up page is shown on the screen; user inputs username, password, and re-entry password; a sign up checking program is performed to check if information is valid; if it is correct, common user goes to home page of Weather Viewing Subsystem.

2.2.2.10 Common user queries today' weather record

Common user logs into the Weather Viewing Subsystem; user chooses "Current Conditions" from the menu and a query form is shown on the screen; user chooses one city listed on the screen; a query program is performed to list today's weather conditions including temperature information, wind, description etc.

2.2.2.11 Common user queries next 5-day weather records

Common user logs into the weather viewing subsystem; user chooses "Next 5-Day Weather" from the menu and a query form is shown on the screen; user chooses one city

from city list on the screen; a query program is performed to list next 5-day weather conditions including weekday, weather description.

2.2.2.12 Common user queries last 7-day weather records

Common user logs into the Weather Viewing Subsystem; user chooses "Past One-Week Weather" from the menu and a query form is shown on the screen; user chooses one city from city list on the screen; a query program is performed to list last 7-day weather conditions including temperature, wind, weather description etc.

2.2.3 Use Case Diagram

Figure 2.1 provides a high-level use case diagram of the On-Line City Weather Forecast System.

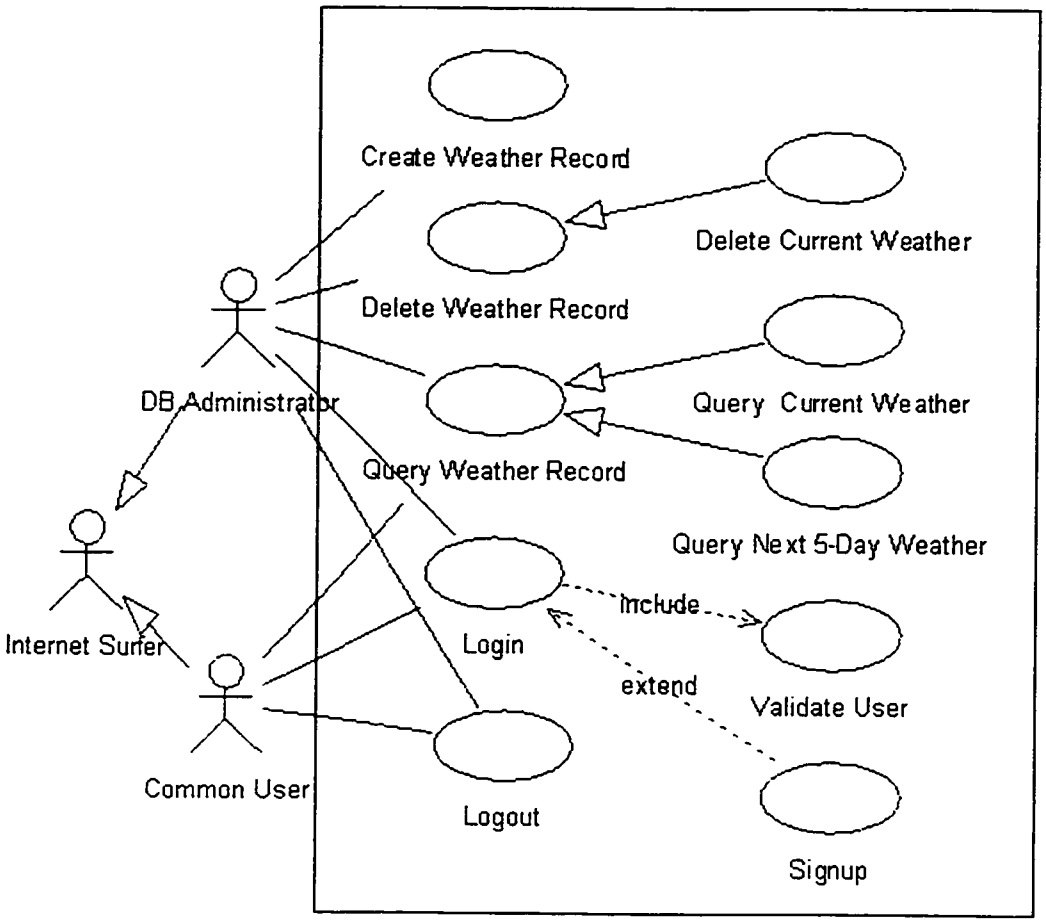

Figure 2.1: Use Case Diagram

2.3 Non Functional Requirements

2.3.1 Computer Hardware and Software Requirements

The On-Line City Weather Forecast System should perform all its functionalities under the following hardware and software environments:

- Sever side:
  - Operating system: Microsoft windows 98/2000/NT 4.0 or higher.

- Database Management System: Microsoft Access2000.
- Minimum hard disk space 1GB

❖ Client side:

- Operating system: Microsoft windows 98/2000/NT 4.0 or higher.
- Minimum modem connection rate: 56kps
- Internet Brower: Microsoft Internet Explorer 5.0 or higher.

2.3.3 Interface Requirement

The interfaces of the system should be clear, understandable and easy to use, so user can use the system without any difficulty.

2.3.4 Safety Requirement

In order to distinguish DB administrator and common user, identification checking should be performed before he or she enters into the system. Thus, we can avoid unauthorized access to the system and also prevent non-DBA from accessing the weather database.

3. System Design

3.1 Design Rationale

Now we enter into design phase. In this phase, system architecture should be carefully chosen because the wrong beginning of the design will endanger the development of the whole project. So our system architecture should focus on ease of maintenance, application performance, and compatibility with existing systems. According to the specifications provided by requirement analyst above, the system should reach the following goals: DB administrator on client side opens a browser to parse weather information, and send back to the database on server side; common client on client side opens a browser to retrieve weather conditions from the database that resides on server side. In addition, DB administrator can do some maintenance to the weather database. In this section, we discuss some main issues concerned in the design phase.

3.1.1 Choice of Architecture

In the system design phase, we take advantage of Model-View-Controller (MVC) [6] architecture to develop our system. The goal of MVC design pattern is to separate the application object from the way it is represented to the user (view) from the way in which the user controls it (controller). Now we consider the roles of MVC components played in a web application. All web applications are event driven. We consider an HTTP Request as an Event. When a web browser from client side sends an event to web application server, a controller from the server side receives the event and performs relevant functionality. It processes the event and according to the current state of the

client, determines the next state of the client. Then it invokes a target view to display the required output. Views and controllers on the server side are considered as presentation layer at web application. Models are separate from the presentation layer views and controllers. It provides Problem Domain services to the Presentation Layer including access to Persistent Storage (database). Both View and Controller may send messages down to the Model and in turn it will send back appropriate responses. Figure 3.1 will shows the complete MVC Architecture design for a Web Application.

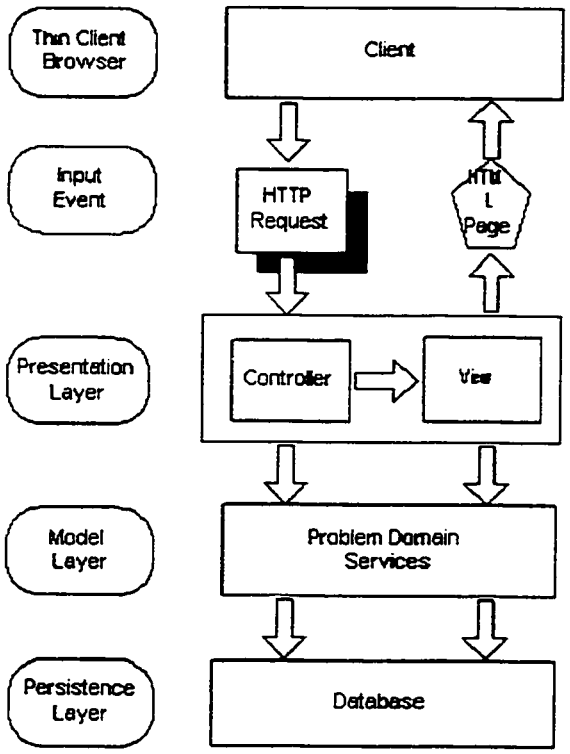

Figure 3.1: MVC Architecture for a Web Application.

Advantages of choosing model-view-controller architecture:

- Multiple views using the same model:

The MVC feature of separating model from view decides that multiple views can have the same model.

- Easier support for new types of clients:

If the system adds a type of client, it simply provides a new view and controller for it and connects them with the existing models.

- Ease of growth: controllers and views can grow as the model grows

3.1.2 Use JSP Technology as MVC Views

In the On-Line City Weather Forecast System, we choose JavaServer pages (JSP) [8] technology to create interactive pages as part of a web-based application. Compared with other scripting languages like HTML, JavaScript, Active Server Pages (ASP), JSP combines static HTML with XML and Java code and hence can produce dynamic, portable, and scalable and easily maintained web pages [9].

Standard JavaScript can generate HTML dynamically on the client. But it can only deal with situations where the dynamic information is on the client's environment. HTTP and form submission data is not available to JavaScript. Because it runs on the client, JavaScript cannot access server-side resources such as databases.

Active Server Pages (ASP) is a similar technology from Microsoft. However, JSP has more advantages. First, the dynamic part is written in Java, which it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.

3.1.3 Use Servlet Technology as a MVC Controllers

Java Servlets [10] are server side Java code run in a server application to answer client requests. A servlet is a dynamically loaded module that services requests from a web server. It is secure, portable, and easy to use.

Compared with traditional server-side solution Common Gateway Interface (CGI) [11], Java Servlet solution provides its own way to keep database connections persistent by using a connection pool, and processing multiple client requests concurrently by making use of Java's multithreading feature. These two features are important advantages of Java Servlets over a CGI solution. On the other hand, the main drawback of the CGI technique is inefficiency in handling concurrent client requests.

3.1.4 Use Microsoft Access to create database as data storage

Microsoft Access is a relational database used on desktop computers to manage information on different levels for different purposes. It is easy to user and also comes equipped with Wizards that help the novice to create tables, forms, queries, and reports.

3.1.5 Scalability

One of the key issues for any online system is its ability to be scaled to large applications. In the On-Line City Weather Forecast System, the feature of MVC design pattern determines the application is still manageable and understandable even it is scaled to a large one. When a common interface receives incoming Events from HTTP Requests, it instantiates an appropriate service class based on the current state of the client. After executing the condition evaluation, it moves the client to the appropriate next state and

the correct view for the next state would be invoked and sends back to the client. The client browser then changes to the new state.

3.1.6 Portability

In order to ensure that the system can run on other platforms, we developed this project using Java technology. One of the major advantages of Java language is portability. Java code can be run on a Java Virtual Machine instead of a physical computer. Since Java programs are portable, a complete system can be developed, tested and integrated before the hardware is available for deployment.

3.2 System Design

We identified two subsystems for the On-Line City Weather Forecast System: Weather Data Processing Subsystem and Weather Viewing Subsystem. We distinguished two subsystems by username and password. Only database administrator can go to Weather Data Processing Subsystem and common user can only go to Weather Viewing Subsystem.

3.2.1 Weather Data Processing Subsystem

In the Weather Data Processing Subsystem, we define three modules: User Interface, Event Handler and Database. Figure 3.2 shows these three modules and their relationship. In User Interface module, we define all the interfaces that users will interact with. In Event Handler module, we defined a controller interface to handle user's input and instantiate services to implement the functionalities of the web application. In Database module, we define tables to store weather data and users' personal information. In this

section, we mainly discuss the associations between components in each module as well as associations between two modules. In the final section, we analyze the static and dynamic structure of the subsystem by drawing class diagram and sequence diagram.

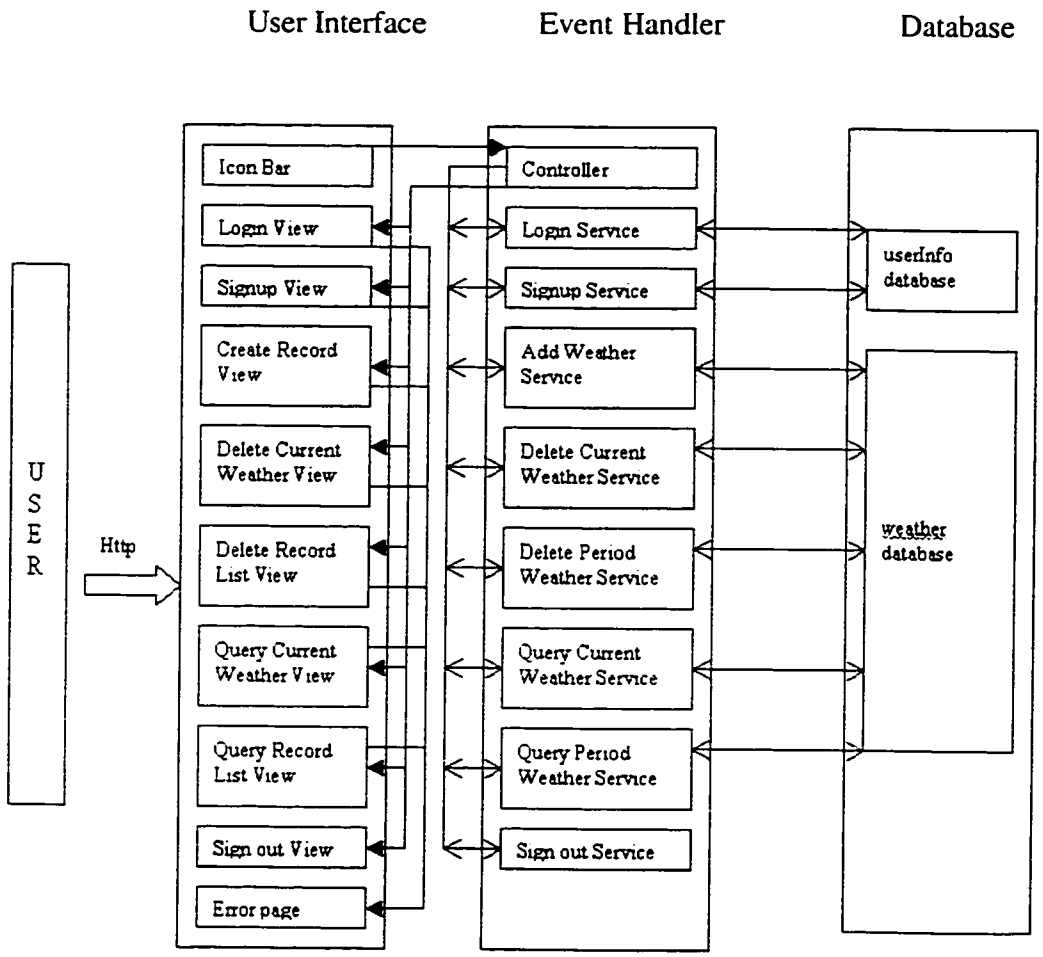

Figure 3.2: Modules with Event and Information Flow

3.2.1.1 Implementation of Associations

3.2.1.1.1 Associations between User Interface and Event Handler

The On-line City Weather Forecast System is a web-based system, so the association between User Interface and Event Handler modules is the association between web client and web server. When a web client clicks a button or triggers an event, a request is made to web server. The web server passes the request to a controller and then the controller instantiates necessary service to weather object or user info object. Later on, the controller forwards the results to the web client. Icon bar in User Interface module has a pointer to the controller in Event Handler module and the controller in Event Handler module has pointer references to all the views in User Interface module.

3.2.1.1.2 Associations between Event Handler and Database

- Two-way association between Login Service and userInfo database

  Login Service sends user's personal information to userInfo database and gets user's validation message from userInfo database. This is implemented by providing a pointer from Login Service to userInfo database and a pointer from userInfo database to Login Service.

- Two-way association between Sign Up Service and User Info Database

  Sign Up Service sends user's personal information to User Info Database and User Info Database stores the information and returns submission result to Sign Up Service. This is implemented by providing a pointer from Sign Up Service to User Info Database and a pointer from User Info Database to Sign Up Service.

- Two-way association between Create Record Service and weather database

Create Record Service sends current weather object and next 5-day weather object to weather database and the weather database stores the objects and returns submission result to Create Record Service. This is implemented by providing a pointer from Create Record Service to weather database and a pointer from weather database to Create Record Service.

- Two-way association between Delete Record Service and weather database

  Delete Record Service sends chosen city and date message to weather database and the weather database deletes relevant records and returns existing records to Delete Record Service. This is implemented by providing a pointer from Delete Record Service to weather database and a pointer from weather database to Delete Record Service.

- Two-way association between Query Record Service and weather database

  Query Record Service sends chosen city and date message to weather database and weather database retrieves some relevant records to the Query Record Service. This is implemented by providing a pointer from Create Record Service to weather database and a pointer from weather database to Create Record Service.

3.2.1.1.3 Associations within Event Handler module

Controller in the Event Handler module reacts to user's input and determines how to handle the incoming requests. It instantiates a Service class and after executing the

created Service, results are returned to the controller. So there is two-way association between Controller and Services. This is implemented by providing a pointer reference from Controller to Services and a pointer from each Service to Controller.

3.2.1.2 Class Diagram

The relationships between classes and the structure of each class in the Weather Data Processing Subsystem are shown in Figure 3.3. From the diagram, we can see that Controller class and StartUp class extend from HttpServlet class; LoginChecking, Add_Weather, Delete_City_Weather, Del_Period_Weather, Query_Cur_Weather, Query_Period_Weather class extend from Service class; StartUp class is used for updating weather database automatically several times a day according to the number entered by the user when the web server starts up; Controller class can instantiate a Service class and handle user's incoming request. HTMLParser class can parser a HTML file and save results into two string arrays.

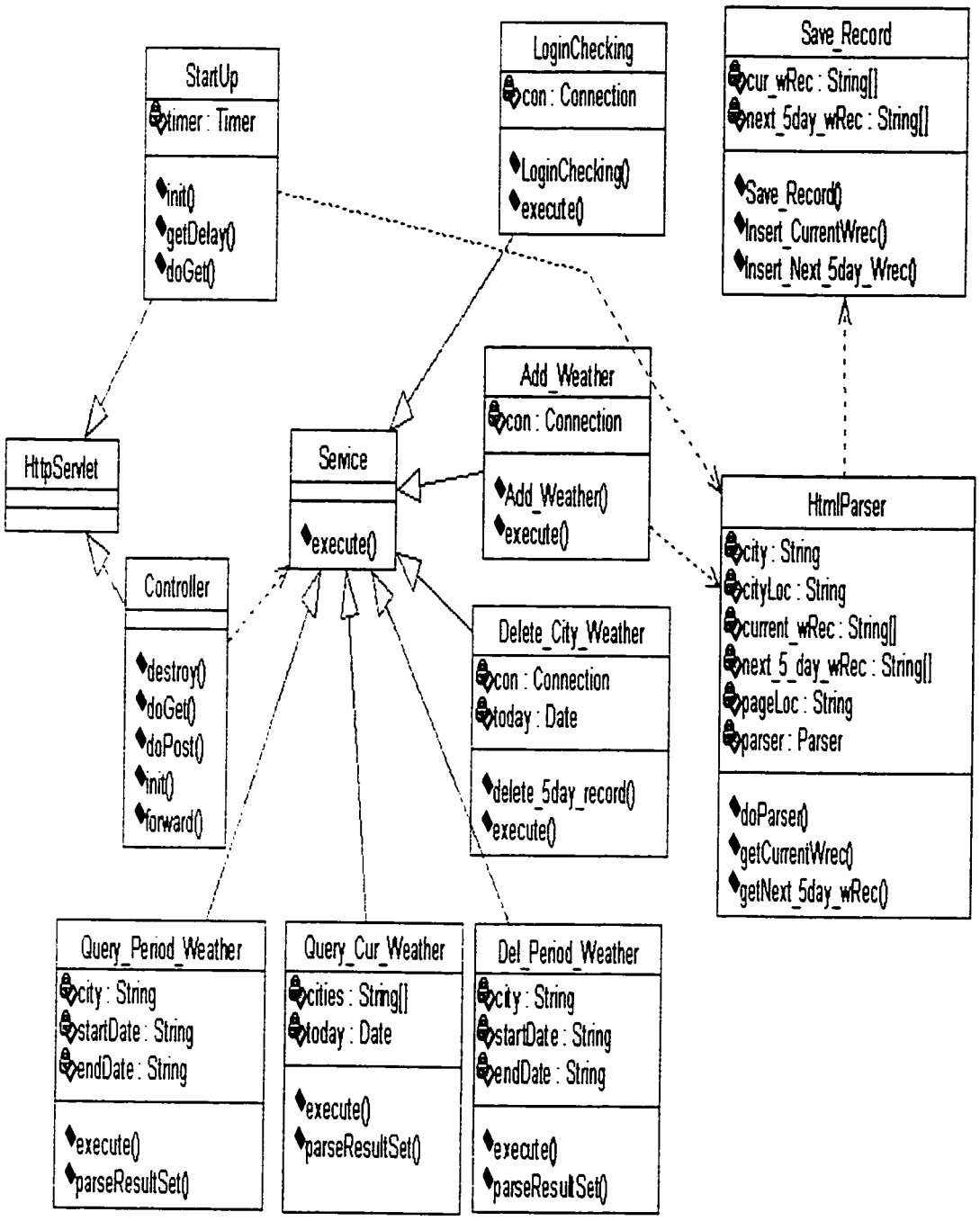

Figure 3.3: Class Diagram

3.2.1.3 Sequence Diagram

Figure 3.4 is a sequence diagram for the scenario: DB Administrator logins to the On-Line City Weather Forecast System, creates a weather record and saves the record into weather database.

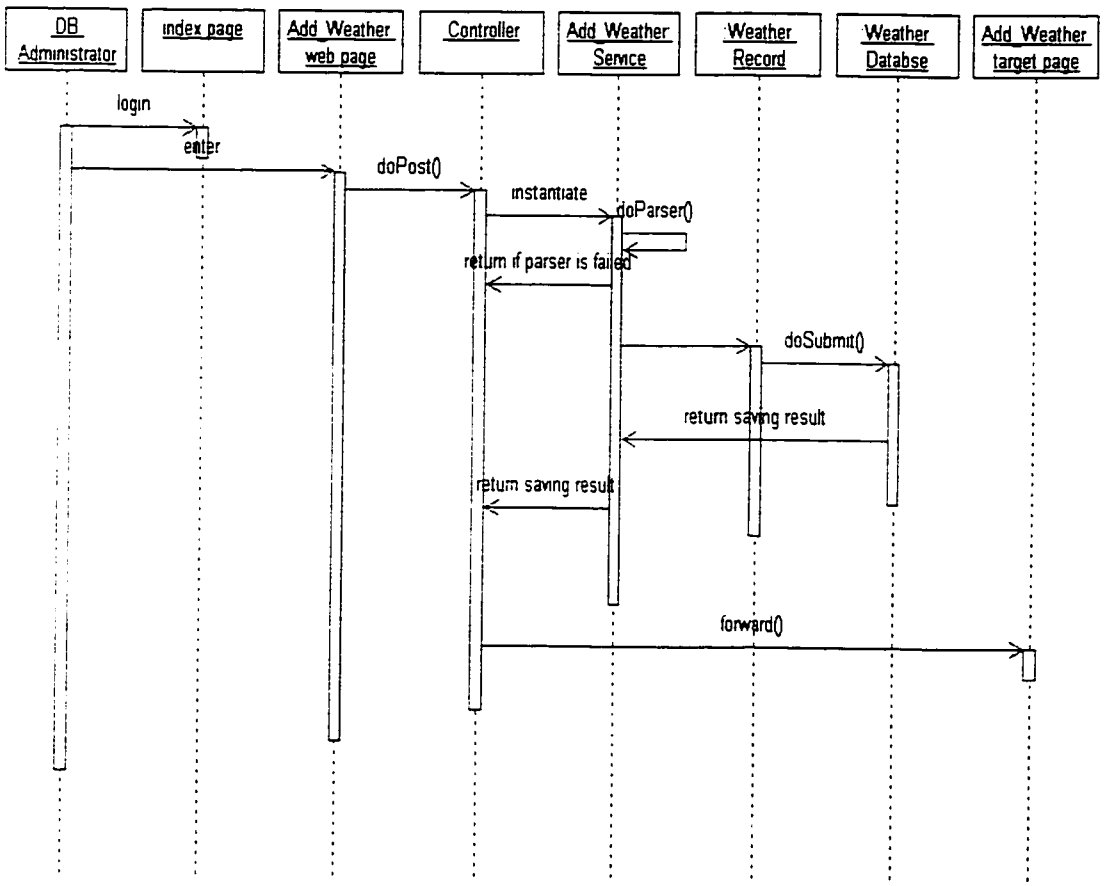

Figure 3.4: Sequence Diagram for Creating a Weather Record

3.2.1.4 Object Models

According to the requirement analysis and concerned with the architecture of the system, we defined three object models: user personal info model, current weather model and next 5-day weather model. User personal information model will be modeled in the userInfo database-login table. Current weather model will be represented as weather database-current weather table and next 5-day weather model will be modeled as weather database-5-day-weather table. In order to make the tables available to the application, ODBC data source should be set up. Next we will introduce the attributes in each object model and each of the object's attributes will also be included in each table representation.

- user personal information model

User personal info model is used for user registration. It includes username, password and another attribute level that is for distinguishing common user from DB Administrator.

Required attribute to model user personal information

Attribute	Type
username password level	String String char

- current weather model

Current weather model is used for modeling current weather information parsed from existing weather forecast web site. The content of the model will be stored into current weather table.

Required attribute to model current weather information

Attribute	Type
date	String
city	String
description	String
temperature	String
pressure	String
visibility	String
humidity	String
wind	String
detail	String
am	char

- next 5-day weather model

Next 5-day weather model is used for modeling next 5-day weather forecast information parsed from existing weather forecast web site. The content of the model will be stored into next 5-day weather table.

Required attribute to model next 5-day weather information

Attribute	Type
City	String
Date	String
Weekday1	String
Desc1	String
Weekday2	String
Desc2	String
Weekday3	String
Desc3	String
Weekday4	String
Desc4	String
Weekday5	String
Desc5	String

3.2.2 Weather Viewing Subsystem

In Weather Viewing Subsystem, we defined three modules in the same way as in Weather Data Processing Subsystem. In User Interface module, we added Current Weather View, Next 5-Day Weather View, Last 7-Day Weather View and Date Weather View so that user can view weather information from difference perspective. In Event Handler module, we added Query Current Weather Service, Query Next 5-Day Weather Service, Query Last 7-Day Weather Service and Query Date Weather Service to implement search functionalities. Figure 3.5 shows three modules and their relationship.

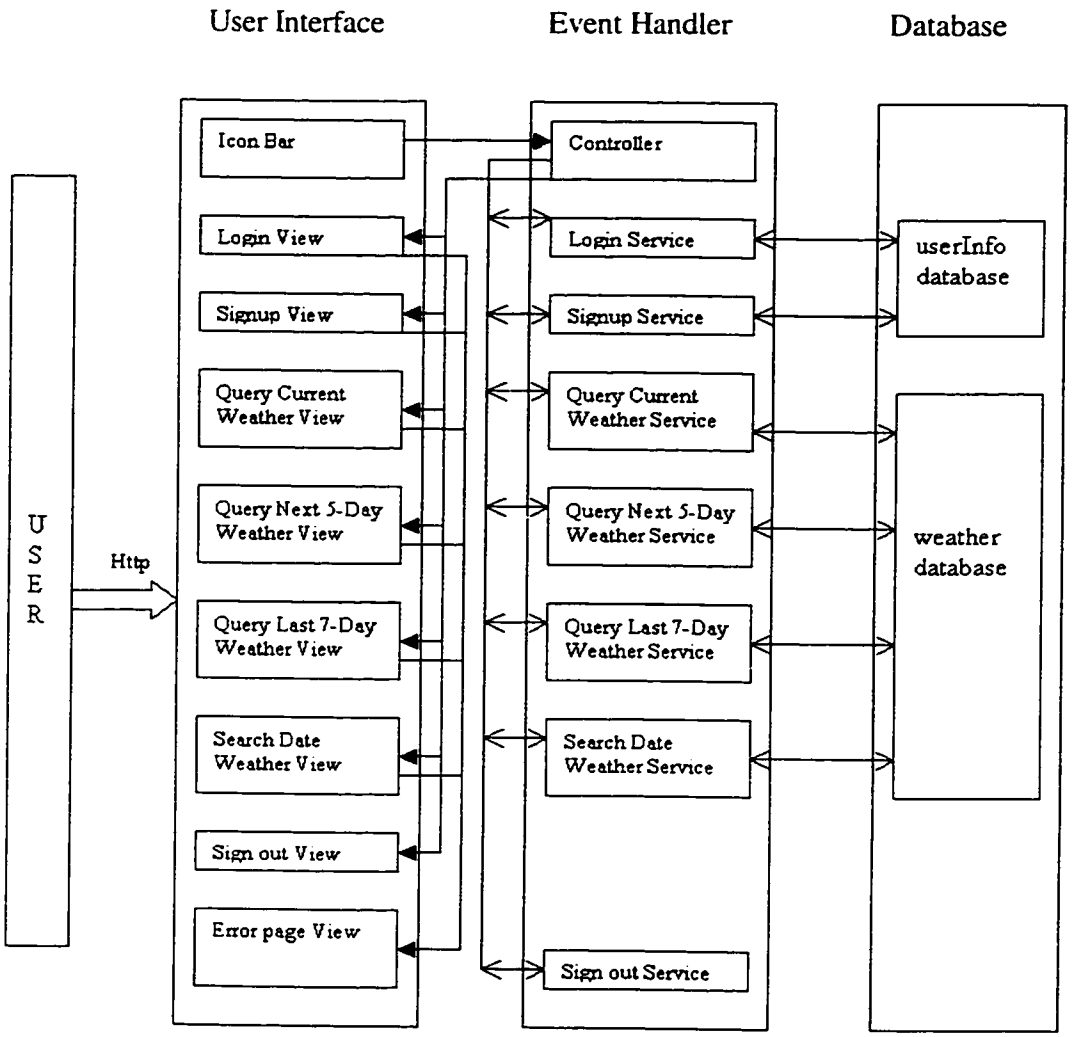

Figure 3.5: Modules with Event and Information Flow

3.2.2.1 Implementation of Associations

The association between Interface module and Event Handler module, the association between Event Handler module and Database module and the association within Event Handler module in Weather Viewing Subsystem is almost the same as in the Weather Data Processing Subsystem introduced above.

3.2.2.2 Sequence Diagram

Figure 3.6 is a sequence diagram for the scenario: common user logins to the On-Line City Weather Forecast System, enters querying conditions, relevant records are retrieved from weather database and sent back to the client.

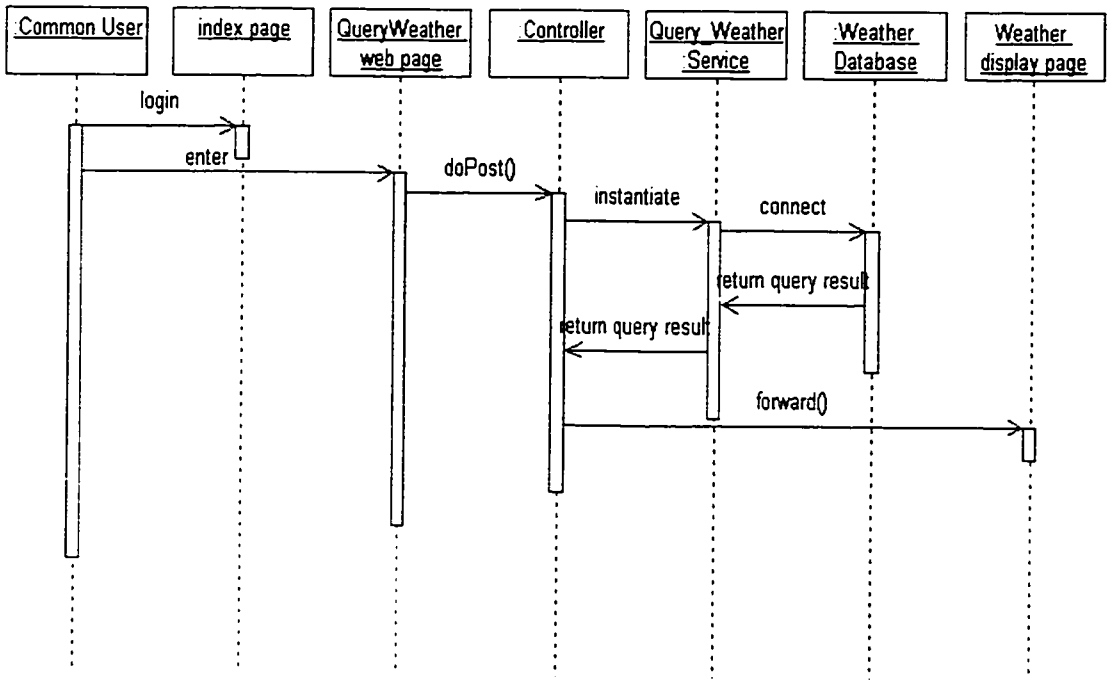

Figure 3.6: Sequence Diagram for Querying Weather Records

4. Implementation

4.1 System Environment

We use tomcat (version 3.3)[13] as our web server that is developed in an open environment and released under the Apache Software License. Tomcat is a container of servlets and JavaServer Pages. It can be used stand-alone or integrated with a web server such as Apache or Internet Information Server. In our project, Tomcat acts as a stand-alone web server and stores MVC components. It takes several steps to implement the server-side MVC design pattern. When the web server receives a request from a web client, it passes the request to Controller. After performing some services to the object models, the Controller forwards the results to a JSP view. The JSP view then sends results to the web server and the web server sends it back to the client. Figure 4.1 shows how to implement the server-side MVC.

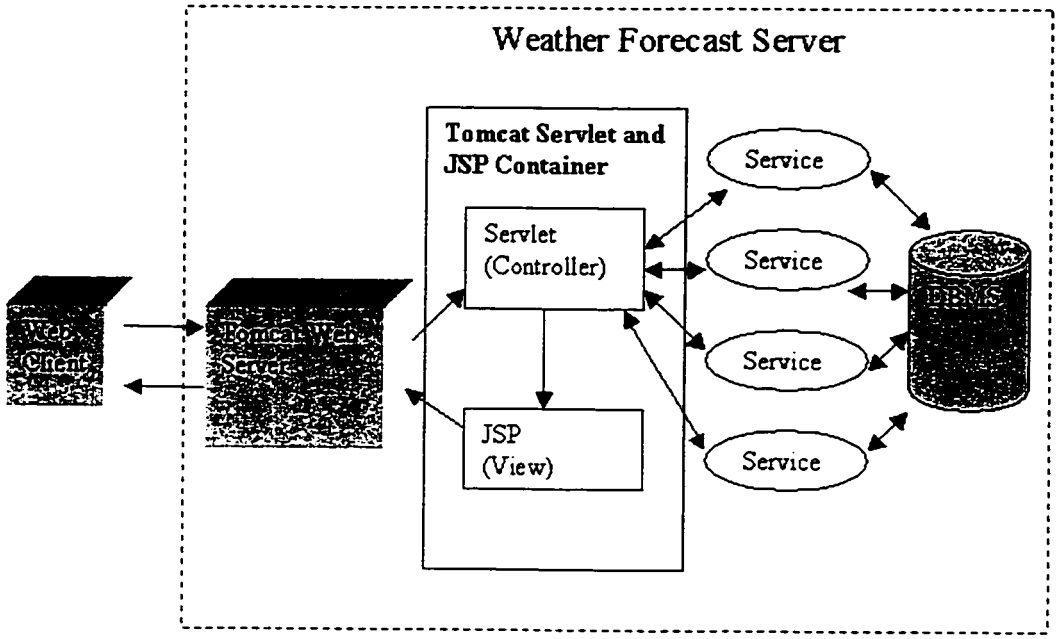

Figure 4.1: Server-Side Implementation of MVC

4.2 Using JDBC Connection Pool Technology

We use JDBC-ODBC Bridge that is provided by Sun with JDK1.1 or later to access a database. When a coming request communicates with a database, a single connection to the database should be established and it should be destroyed after finishing. However connecting to a database is time consuming since the database must allocate communication and memory resources as well as set up security checking, thus results in slow servicing of requests. JDBC Connection Pool gives a solution to solve the problem. We can create a pool of *n* connections to the database. Thus instead of servicing one request at a time we can service *n* number of requests at once. Establishing the connection once and then use the same connection for subsequent requests can therefore

dramatically improve the performance of a database driven web application [15]. The interface of class *ConnectionPool* is defined in Figure 4.2.

```
public class ConnectionPool {

 // Fields
 private String driver;
 private String url;
 private int size;
 private String username;
 private String password;
 private Vector pool;

 // Constructors
 public ConnectionPool() { }

 // Methods
 public void setDriver(String p0) { }
 public String getDriver() { }
 public void setURL(String p0) { }
 public String getURL() { }
 public void setSize(int p0) { }
 public int getSize() { }
 public void setUsername(String p0) { }
 public String getUsername() { }
 public void setPassword(String p0) { }
 public String getPassword() { }
 private Connection createConnection() throws Exception { }
 public synchronized void initializePool() throws Exception { }
 private void addConnection(PooledConnection p0) { }
 public synchronized void releaseConnection(Connection p0) { }
 public synchronized Connection getConnection() throws Exception { }
 public synchronized void emptyPool() { }
}
```

Figure 4.2: Interface of Class ConnectionPool

From above definition, we know that the *ConnectionPool* object should meet the following requirements:

- It holds *n* number of open connections.

- When the pool is close, all connections to the database are released
- If *n+1* connections are requested, it can create a new connection and add it to the pool.

4.3 Inter-Servlet Communication

In order to access the userInfo database, weather database and save servicing time of requests, we create *LoginConnectionPool* servlet and *WeatherConnectionPool* servlet. ODBC data sources login from userInfo database and weather from weather database should be set up before creating these servlets. Instead of making a ConnectionPool object available to only one servlet, we add the ConnectionPool object into a reference to *javax.servlet.SerlvetContext*, which is shared by all servlets. This makes the ConnectionPool object available to all the servlets. Now we can give an example code in *LoginConnectionPool* servlet.

```
ConnectionPool pool=new ConnectionPool();
pool.setDriver("sun.jdbc.odbc.JdbcOdbcDriver");
pool.setURL("jdbc:odbc:login");
pool.setSize(4);
pool.setUsernamе("");
pool.setPassword("");
pool.initializePool();
ServletContext context=getServletContext();
context.setAttribute("LOGIN_CONNECTION_POOL", pool);
```

The preceding code first creates an instance of *ConnectionPool* with 4 open connections to userInfo database. Then it gets a reference to the *ServletContext* that is shared by all the serlvets and calls its *setAttribute*() method to adds the *ConnectionPool* object to the shared *ServeltContext*. This makes the *ConnectionPool* avalilabe to other servlets.

To make the connections available before the first request is made, we set up the *LoginConnectionPool* and *WeatherConnectionPool* as preloaded servlets when the Tomcat engine starts up, which is implemented in *web.xml* file:

```
<servlet>
        <servlet-name>ConnectionPool.LoginConnectionPool</servlet-name>
        <servlet-class>ConnectionPool.LoginConnectionPool</servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
<servlet>
         <servlet-name>ConnectionPool.WeatherConnectionPool</servlet-name>
         <servlet-class>ConnectionPool.WeatherConnectionPool</servlet-class>
         <load-on-startup>2</load-on-startup>
</servlet>
```

The above code ensures that the *ConnectionPool* objects one of those is defined in *LoginConnectionPool* servlet and the other is in *WeatherConnectionPool* servlet are available to all the other servlets when the web server starts.

4.4 Implementation of MVC

In this part, we concentrate on implementing a MVC design pattern of the web-based system. We discuss the implementation details of JSP views, controller as well as some services that are connected to the object models.

4.4.1 Startup

Before implementing the MVC design pattern, we first introduce *Startup* class. *Startup* class is an Httpservlet that is loaded when Tomcat engine starts and keeps running until the server is close. This is done by *web.xml* file:

```
<servlet>
    <servlet-name>StartUp</servlet-name>
      <servlet-class>StartUp</servlet-class>
    <load-on-startup>3</load-on-startup>
</servlet>
```

When *Startup* class executes, user is asked to input a number n for recording the times of updating weather data in weather database. After typing the number n, the systems will update weather database automatically n times a day. This is done in the *init ()* method of *Startup* class. The source code in the *init ()* method of *Startup* class is in listing:

```
public void init(ServletConfig config) throws ServletException {
  super.init(config);
```

```
try{
System.out.println("Enter recording time(1-9):");
int t=System.in.read()-'0';
while (t<0 || t>9){
  System.out.println("Enter recording time(1-9):");
  t=System.in.read()-'0';
}
TimerEvent e=new TimerEvent();
int delay=(24/t)*60*60;
if (timer==null){
 timer=new Timer(1000*delay, e);
  timer.start();}
else if (!timer.isRunning())
  timer.restart();
System.out.println("Data is waiting for updating");
}
catch(IOException e){}
}
```

First it asks user to enter a number 0-9 from the keyboard and take the number as the times of updating weather database per day. An instance of *Timer* executes a *TimerEvent* object at a fixed interval that is responsible for extracting weather data from 14 existing web pages and saving it into weather database.

4.4.2 Views

Now we start to define the interfaces that users interact with. From the requirements described in previous section, we need to define Index view, Weather Record List view, Current City Weather Forecast View, Weather Detail View, Next 5-Day Weather Forecast View, Last 7-Day Weather Forecast View and an Error Page. In this section we introduce each of them.

4.4.2.1 Index View

Index View is homepage for the On-Line City Weather Forecast System. It is represented in two files *Weather_DB.jsp* and *Weather_Viewer.jsp*. One file is for Weather Data Processing Subsystem and the other is for Weather Viewing Subsystem. An image of the Index View in Weather Data Processing Subsystem is in Figure 4.3.

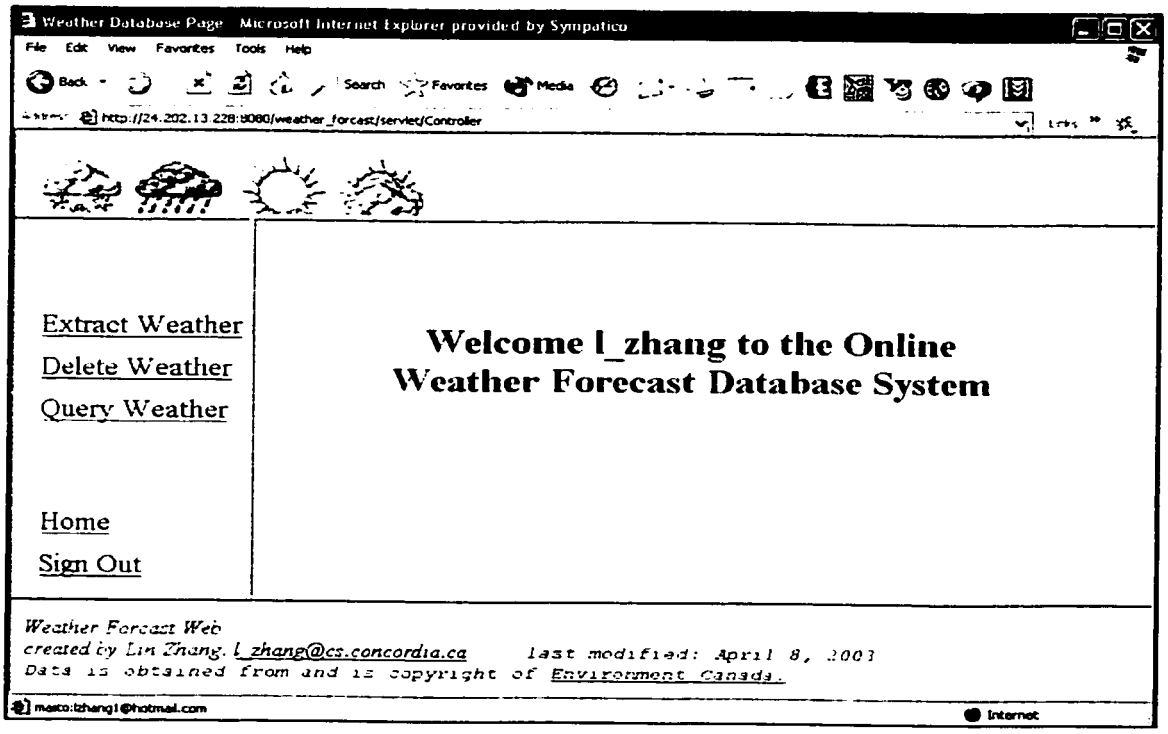

Figure 4.3: Index View for DBA

4.4.2.2 Weather Record List View

The Weather Record List View is represented in the file *Query_Weather_Viewing.jsp*. An image of the view is in Figure 4.4.

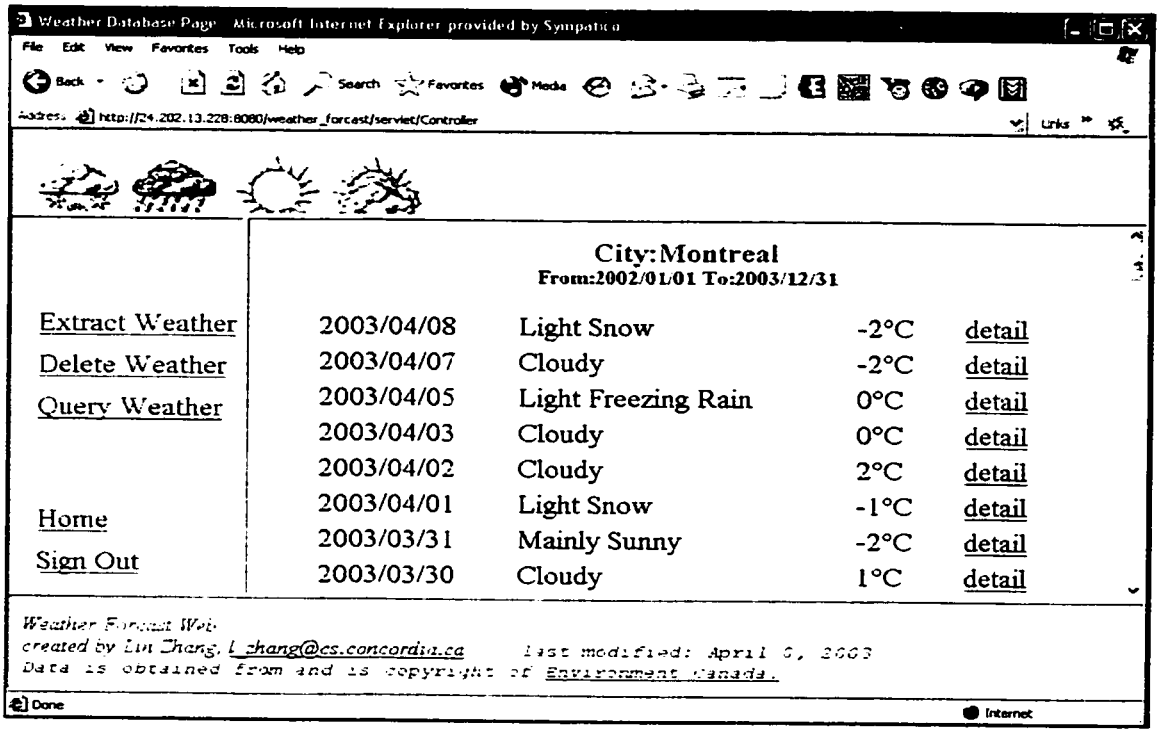

Figure 4.4: Weather Record List View

Weather records are received from RECORDS attribute of a request object, which is inherited from *javax.servlet.ServletRequest*. Their values including date, weather description and temperature are stored in HashMap objects. The Weather Record List View displays the values of those records on the screen. This is done by the following code snippet:

```
Object[] records=(Object[])request.getAttribute("RECORDS");
for (int x=0; x<records.length; x++){
```

```
HashMap record=(HashMap)records[x];
out.println("<TR><TD>"+record.get("DATE") + </TD>" +
    "<TD>+ record.get("DESCRIPTION") + "</TD>" +
    "<TD>" + record.get("TEMPARATURE") + "</TD>" +
    "<TD><a target=_blank href=/weather_forcast/servlet/Controller?"+
    "service=RecordDetail&date="+record.get("DATE")+
    "&target=/WeatherInfo.jsp>detail</a></TD></TR>");
}
```

4.4.2.3 Current City Weather Forecast View

Current Weather View is represented in the file *CurrentWeatherList.jsp*. It lists all the current weather forecast of selected cities including weather description, temperature. An image of Current Weather Forecast View is in Figure 4.5.

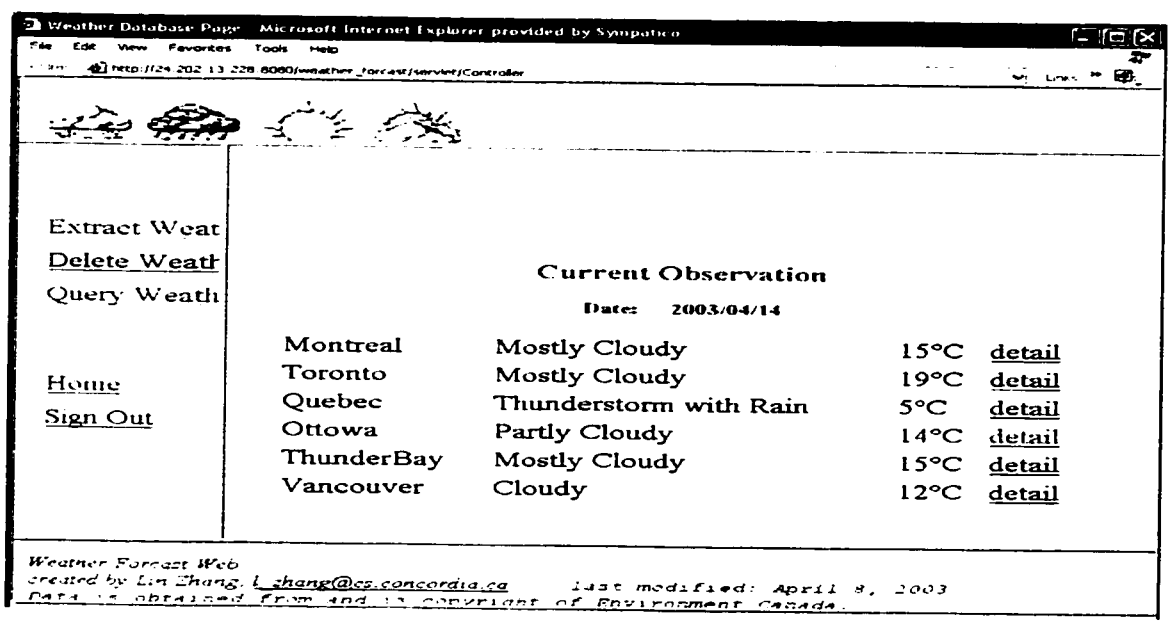

Figure 4.5: Current City Weather Forecast View

4.4.2.4 Weather Detail View

Weather View is represented in the file *WeatherInfo.jsp*. It lists detail weather information: temperature, weather condition, pressure, visibility, humidity, wind and description. An image of Current Weather Forecast View is in Figure 4.6.

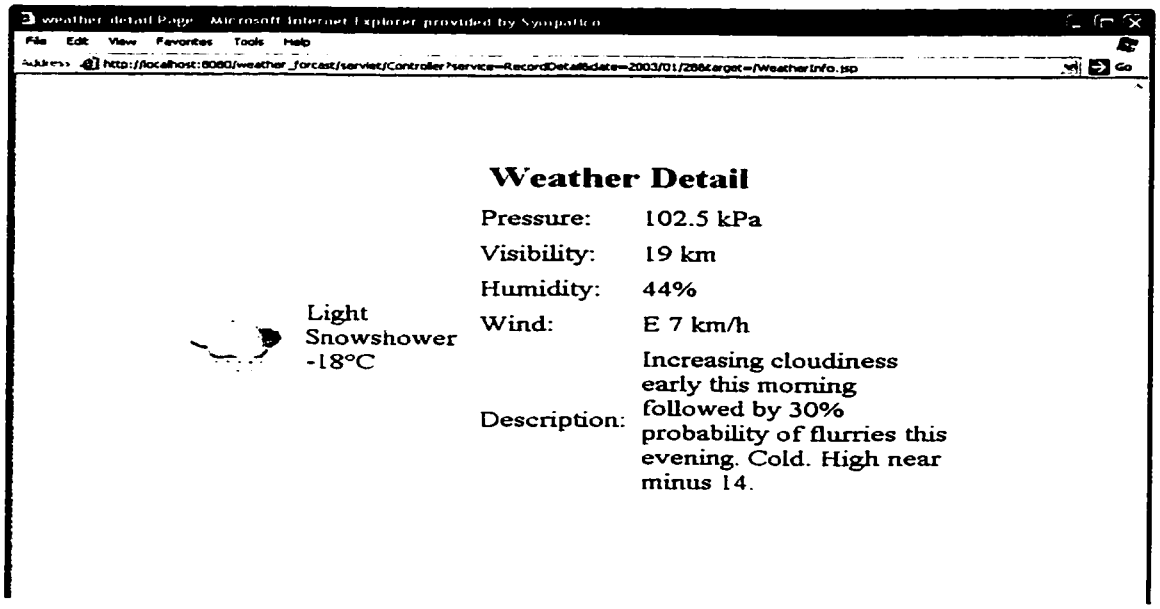

Figure 4.6: Weather Forecast View

4.4.2.5 Next 5-Day Weather Forecast View

The Next 5-Day Weather Forecast View is represented in the file *List_5day_Weather.jsp*. An image of the Next 5-Day Weather Forecast View is in Figure 4.7.

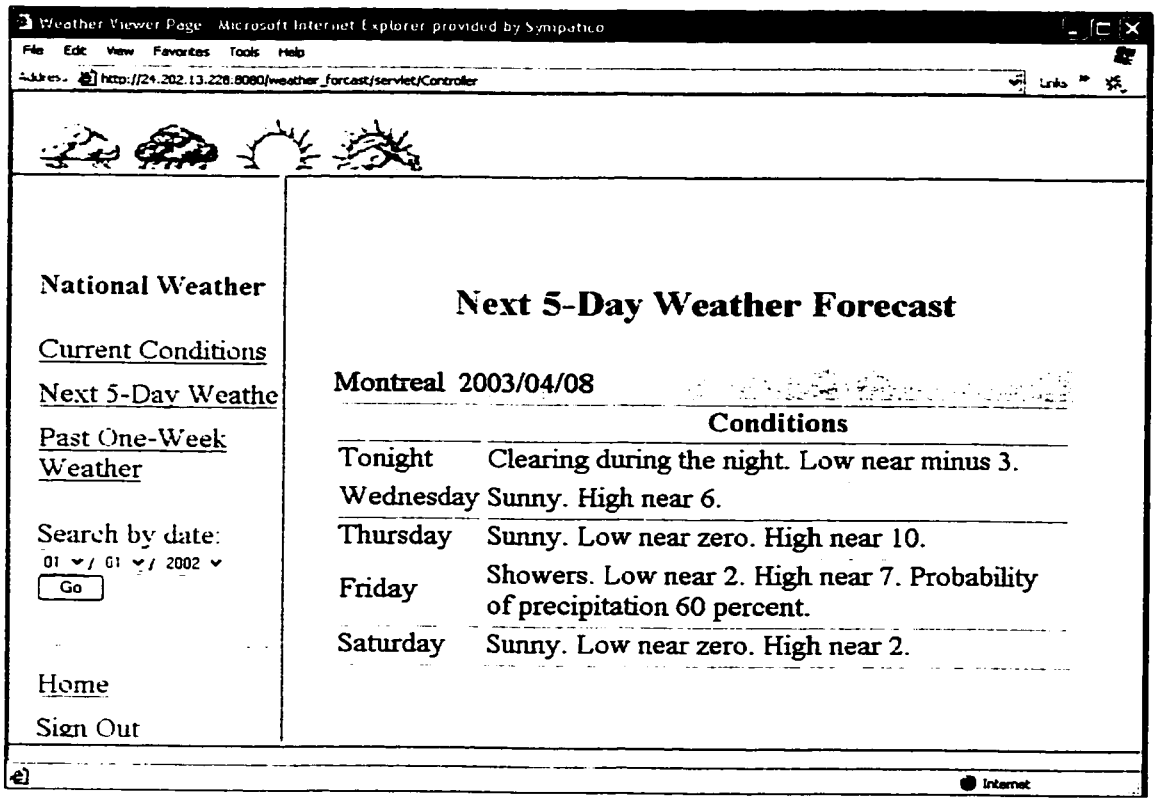

Figure 4.7: Next 5-Day Weather Forecast View

In the file *List_5day_Weather.jsp*, a weather record consisting of next 5 days weather description is received from RECORD attribute of a request object. The Next 5-Day Weather View displays the contents of the weather record on the screen.

4.4.2.6 Last 7-Day Weather Forecast View

The Last 7-Day Weather View is represented in the file *Past_7day_Weather.jsp*. Figure 4.8 shows an image of the Last 7-Day Weather Forecast View.

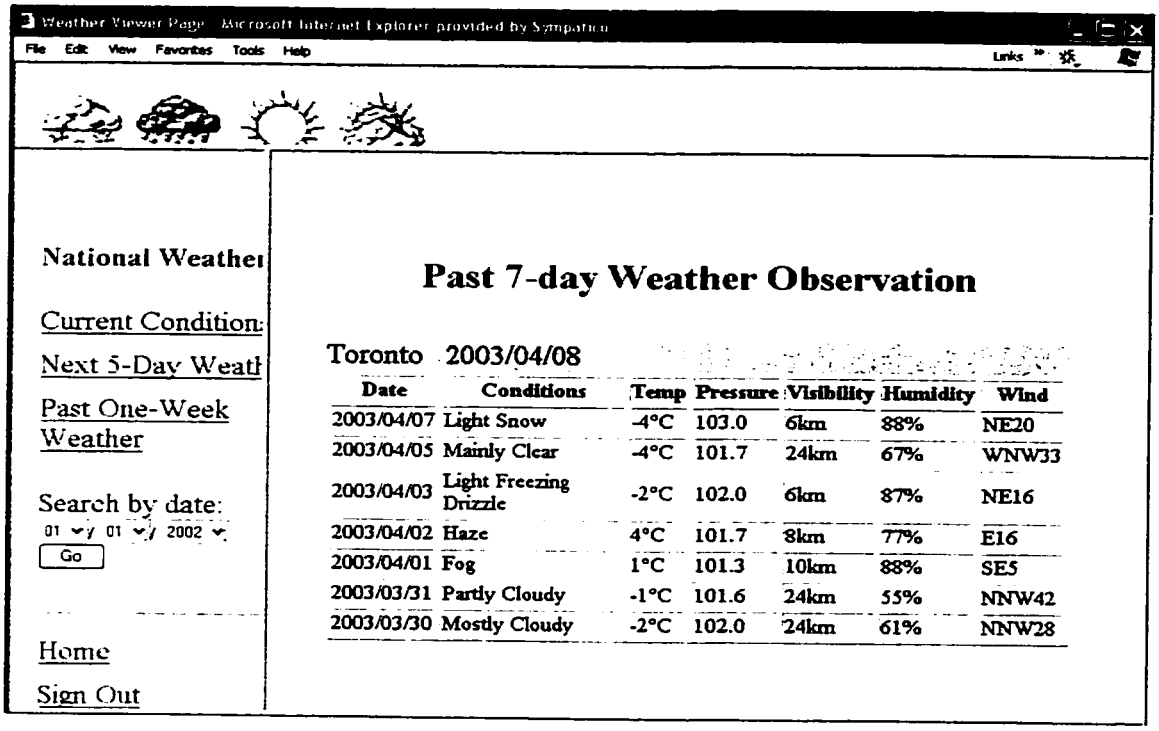

Figure 4.8: Last 7-Day Weather Forecast View

In the file *Last_7day_Weather.jsp*, weather records consisting of last 7 days weather observation data are received from RECORDS attribute of a request object. The Last 7-Day Weather View displays the contents of the weather records including last seven days weather conditions, temperature, pressure, etc, weather information on the screen.

4.4.3 Controller

Controller is a central part in the MVC design pattern. It reacts to the user's request and decides how to deal with the coming request. In our project, the controller is implemented as a servlet. It does the following work:

- Receive user's request, parser service name and target JSP file name from the request.

- Instantiate a Service class by the service name and call *execute ()* method to handle user's request.
- Forward the result of the executed service to the target JSP file for viewing.

The following code implements the functionality of the Controller:

```
String serviceName=request.getParameter("service");
   if (serviceName==null){
    throw new ServletException("No service named");
   }
String target=request.getParameter("target");
if (target==null){
    throw new ServletException("No target named");
}
 ServletContext context=getServletContext();
Class cls=Class.forName(serviceName);
Service service=(Service)cls.newInstance();
service.execute(request, response, context);
forward(request, response, target);
```

Now we give an example of a request form containing service parameter and target parameter.

```
<Form name="query_weather"
```

```
    action="/weather_forcast/servlet/Controller" method="post">
    <INPUT type="submit" name="submit" value="  Query  ">
    <INPUT TYPE="hidden" value="Query Weather" name="service">
    <INPUT TYPE="hidden" value="/Query_Weather_Viewing.jsp" name="target">
</Form>
```

When the client submits the above request, the *Controller* will execute service *Query Weather* and forward the executed result to target JSP file name *Query_Weather_Viewing.jsp*.

4.4.4 Service

When the *Controller* class receives client's request, an instance of *Service* class will be created and its *execute ()* method will be called to perform its functionality. Here we define a public interface *Service* class that acts as prototype of all the services. In the Service interface, only one method execute () is defined. The following code does this:

```
public interface Service {
 public void execute(HttpServletRequest request,
  HttpServletResponse response,
  ServletContext context) throws Exception;
}
```

In this project, we define several services. Each service performs its own functionality by implementing the execute () method of Service interface. Now we list all the services performed in the system.

- *Add Weather* Service is to add creating records into weather database.
- *Delete Current Weather* Service is to delete current day's records from weather database of selected cities.
- *Delete Period Weather* Service is to delete a period time of weather records of a specific city.
- *Delete Database* Service *is to delete all the weather records from weather database.*
- *Query Current Weather* Service is to retrieve current weather records of selected cities from weather database.
- *Query Period Weather* Service is to retrieve a period time of weather records of a specific city.
- *Sign Up* Service is to register new user into the system.
- *Login Checking* Service is to check user's authorization.
- *Query Next 5-Day Weather* Service is to retrieve next 5 days' weather condition from weather database.
- *Past 7-Day Weather* Service is to retrieve last 7 days' weather condition.
- *Search Date Weather* Service is to search any data's weather condition and
- *SignOut* Service is to perform logging out of the system.

In the next part of this section, we focus on four services: *Add Weather* Service, *Query Current Weather* Service, *Delete Current Weather* Service, *Search Current Weather* Service and *Query Next5-Day Weather* Service.

4.4.4.1 Add Weather Service

The *Add Weather* Service is an implementation of interface *Service.execute ()* method. When a user selects one or more cities from the city list on the screen, and presses Save Selected Cities button, *execute ()* method will start its execution.

First, it receives a list of cities from the request. According to the web page location of each city, an HTML parser is then performed to open an HTML file, select weather data including current weather and next 5 days weather forecast, and save it into weather database. In the final, the result of submission is assigned to an attribute of a request object called SAVING_FLAG. If the value of SAVING_FLAG is 1, *Add Weather* service will send message "records have successfully inserted into weather database" to its target JSP file. Otherwise, it will send an error message "submission is failed". The sample code in *execute ()* method is in listing:

```
String[] city=request.getParameterValues("cities");
String[] current_wRec=new String[11];
for (int i=0;i<city.length;i++)
{
 current_wRec[0]=city[i];
```

```
    HtmlParser parser=new HtmlParser(city[i]);
    parser.action_Performed();
    current_wRec=parser.getCurrentWrec();
    Object[] next_5_day_wRec=parser.getNext_5Day_Wrec();
    Save_Record s=new Save_Record(current_wRec, next_5_day_wRec);
    int result=s.action_Performed(request,response,context);
    if (result==0)
    {request.setAttribute("SAVING_FLAG", "0");//failed
     return;}
  }
  else request.setAttribute("SAVING_FLAG", "1");//successful
```

4.4.4.1.1 Weather Parser

Now we introduce the weather parser occurred in *Add Weather* Service. It uses *HTMLEditorKit.Parser* in package *javax.swing.text.html* and its implementing class *ParserDelegator* in package *javax.swing.text.html.parser* in Java Swing system to parser an HTML file.

In this project, weather parser should exact weather information of 14 cities from website http://weatheroffice.ec.gc.ca. First, according to the name of each city received from *Add Weather* Service, the parser finds weather web page location of the city, opens an HTML file and prepares to read the input stream. Then it calls parse *()* method from an instance

of class *PaserDelegator* to parse the given stream and drive the given callback with the results of the parse. A subclass of *HTMLEditorKit.ParserCallback TempParserListener* is designed to receive the parsing results, select desired weather data and save those data into records current_wRec and next_5_day_wRec. This is done by the following code:

```
if (pageLoc.indexOf("://") > 0)
{
  URL u = new URL(pageLoc);
  Object content = u.getContent();
  if (content instanceof InputStream)
    r = new InputStreamReader((InputStream)content);
  else if (content instanceof Reader)
        r = (Reader)content;
      else throw new Exception("Bad URL content type.");
}
else r = new FileReader(pageLoc);
parser = new ParserDelegator();
parser.parse
  (r,new TempParserListener(current_wRec, next_5_day_wRec), true);
r.close();
```

When class *HTMLEditorKit.ParserCallback* receives arbitrary sized chunks from a HTML document, it recognizes markup elements and separates them from the plain text. There are several methods In class *HTMLEditorKit.ParserCallback* to handle incoming

streams: *handleText(), handleComment(), handleStartTag(), handleEndTag()* etc. Our class *TempParserListener* inherits class HTMLEditorKit.ParserCallback and only one method *handleText ()* are implemented, other methods are ignored.

Next we discuss the general ideas of handling texts from a HTML file in method *handleText()*. In this system, the weather parser should extract 12 items of weather information, which are general weather condition, temperature, pressure, visibility, humidity, wind, current weather description and next 5 days' weather forecast. In its implementing class *TempParserListener*, variable *count* is used for item counter and variable *countDay* is used for day counter. There are total 9 situations considered in handling text entries. Situations from 0 to 8 mainly deal with relevant weather information and situation 9 deals with irrelevant text input. The source code of method *handleText()* is in listing:

```
public void handleText(char[] data, int pos) {
    ///////// enough weather information, no more parser///////
    if ((count==12) && countDay==5) return;
    String t=new String(data);
    if (next!=9) {
      if (t.equals(" ")) return;
      if (t.equals("UTC")){ next=0;return;}
      count++;
      addRecord(t);
```

```
        next=9;
      }
      setNext(t);
    }
```

In above code snippet, method *setNext()* is used to decide which situation the current input text is in. If the incoming string matches one of titles in a definition list, variable *next* is assigned to a new value. Otherwise, it still remains 9. If *next* is not equal to 9, next input string from the HTML file is the data need to be selected.

When method *handleText()* receives a new string , it checks the value of variable *next*. If the value is equal to 9, go to execute method *setNext ()* to receive next input string, otherwise, plus 1 to item counter, and saves the string into current weather object or next 5-day weather object.

4.4.4.1.2 Saving Weather Record

Save_Record class is for saving extracted weather information into weather database. It stores current weather object into table weather_table and stores next 5-day weather object into 5_day_weather table.

4.4.4.2 Query Current Weather Service

The *Query Current Weather* Service is an implementation of the interface *Service.execute()* method in Weather Data Processing Subsystem. When a user selects

one or more cities from the city list on the screen and presses *Display Current Conditions* button, *execute ()* method will start its execution.

First, it receives names of the selected cities from the request object, and opens an unused connection from a ConnectionPool object *WEATHER_CONNECTION_POOL* to connect weather database. Second, it performs a select statement on the weather_table under the conditions of current date and name of each city and returns a ResultSet object. If current record exists, the ResultSet is converted into a HashMap object. In the final, a list of HashMap objects is assigned to RECORDS attribute of the request object and sent back to the target view of the service. This is done by the following code in method execute ():

```
String[] city=request.getParameterValues("cities");
Vector results=new Vector();
java.util.Date today=new java.util.Date();
SimpleDateFormat formatter;
formatter=new SimpleDateFormat("yyyy/MM/dd");
String to_day=formatter.format(today);
pool=(ConnectionPool)
context.getAttribute("WEATHER_CONNECTION_POOL");
con=pool.getConnection();
Statement statement=con.createStatement();
String queryStr;
if (con!=null){
```

```
for (int i=0;i<city.length;i++){
  queryStr="SELECT * "+
   "FROM weather_table "+
   "WHERE (date= '" + to_day + "')"+
    " AND (city='"+ city[i] +"')";
  ResultSet rs=statement.executeQuery(queryStr);
  boolean moreRecords=rs.next();
  if (moreRecords)
   results.add(parseResultSet(rs));}
request.setAttribute("RECORDS", results.toArray());
}
```

4.4.4.3 Delete Current Weather Service

The *Delete Current Weather* Service is an implementation of the interface *Service.execute()* method. When a user selects one or more cities from the city list on the screen and presses *Delete* button, *execute ()* method will start its execution.

It receives names of the cities from the request object, and opens a connection from a WEATHER_CONNECTION_POOL object to connect weather database. Second, it performs a delete statement on weather_table under the conditions of current date and each name of the city and returns a ResultSet Object. If the ResultSet is empty, "current weather data for the city is not available" will be shown on the target view of the service. Otherwise, "current weather has successfully been deleted" will be shown.

4.4.4.4 Search Current Weather Service

The *Search Current Weather* Service is an implementation of the interface *Service.execute()* method. When a user selects one city from the city list on the screen and presses Search button, *execute ()* method will start its execution.

First, it receives name of the city from the request object, and opens a connection from a WEATHER_CONNECTION_POOL object to connect weather database. Second, it performs a select statement on weather_table, and returns a ResultSet containing a weather record with the chosen city and current date. Last, When the ResultSet is returned, it is converted into a Hashmap object and put back to the target file called *CurrentWeather.jsp*.

4.4.4.5 Query Next 5-Day Weather Service

The *Query 5-Day Weather* Service is an implementation of the interface *Service.execute()* method. When a user selects one city from the city list on the screen, and presses Display Next 5-Day Weather Condition button, *execute ()* method will start its execution.

First, it receives name of the city from the request object, and opens an unused connection from a WEATHER_CONNECTION_POOL object to connect weather database. Second, it performs a select statement on 5_day_weather table and returns a ResultSet containing a weather record with the chosen city and current date. Last, when

the ResultSet is returned, it is converted into a Hashmap object and assigned to an attribute of request object called RECORD. This is done by the following code.

```
if (con!=null){
    Statement statement=con.createStatement();
    String query="SELECT * "+
     "FROM 5_day_weather "+
     "WHERE (date= '" + current_day + "')"+
     " AND (city='"+ city +"')";
    ResultSet rs=statement.executeQuery(query);
    boolean moreRecords=rs.next();
    HashMap record=new HashMap();
    if (moreRecords)
        record=parseResultSet(rs);
    record.put("CITY", city);
    request.setAttribute("RECORD", record);
}
```

5. Conclusion

5.1 Summary

The On-Line City Weather Forecast System is a web-based application for extracting weather data and observing weather conditions. It provides a friendly interface that users can interact with. In addition, in order to keep the weather data up to date, weather database updates its information automatically up to 9 times.

In this document, we give an example of how to design a web application using MVC pattern and how to leverage a server-side implementation of the MVC. It is demonstrated that it is a good choice to choose MVC architecture. We can take advantage of its feature separating presentation layer from model layer to scale up the application easily. And also it is proved that a web application implemented by JSP, Java, and Servlet technology is efficient, portable and independent of platform.

5.2 Discussion and Future Work

In the on-line system, weather parser is a central part of processing weather data. Now it works well to select weather data from existing web site and save it into user-defined weather database for later viewing. However, the current parser has some potential drawbacks that might affect the performance of the weather system.

The working procedures of the parser have been introduced in design phase. The general principle of the HTML parser is: if an input string matches one of the titles in a definition list, next input string is the weather data need to be extracted. Now we suppose the page format of 14 cities is same, so the weather parser can work well. However, the web pages are dynamic. They update contents everyday. If the titles that are used to identify weather data change e.g. from upper case to lower case, the parser might lose data or even result in system crash. This is a potential problem in our weather parser. It is easy to overcome the drawback by observing the contents of those web pages when the weather system runs abnormally. If a title is found different, we come back to the weather parser program and add the new title into title definition list so that the HTML parser can handle all the possible cases to ensure the system runs well.

Current weather parser ignores all the tags in the HTML files and can only extract text input. To make the system illustrated, we can add a new method into the parser, which can handle image tags and extract weather images from the existing HTML files. In addition, some advanced search functionalities are expected to add into the system so that the weather system can present different views to the clients.

References

[1] Oleg Kiselyow, "Implementing Metcast in Scheme*", Software Engineering, Naval Postgraduate School, Monterey, CA 93943 http://okmij.org/ftp/papers/Scheme-Metcast-paper.ps.gz

[2] Environment Canada, http://weatheroffice.ec.gc.ca

[3] "Weather Processor (WXP)", http://weather.unisys.com/wxp/, August 2002

[4] Paul Harmon, Mark Watson, "Understanding UML: The Developer's Guide: with a Web-Based Application in Java", Academic Press/Morgan Kaufmann, November 1997

[5] James Goodwill, "Developing Java Servlets(2nd Edition)", Sams,2001

[6] uidesign.net, "Server-side MVC Architecture", October 1999 http://www.uidesign.net/1999/papers/webmvc_part1.html

[7] Aimin Han, "BIB TEX Server", Computer Science Department, Concordia University, Canada, 2001.

[8] Sun Corporation, "JavaServer Pages(TM) Technology", http://java.sun.com/products/jsp/

[9] "JSP Web Hosting", http://www.low-cost-web-hosting-guide.com/jsp-web-hosting.shtml

[10] Sun Corporation, "Java(TM) Servlet Technology", http://java.sun.com/products/servlet/

[12] Amanda W. Wu, Haibo Wang and Dawn Wilkins, "Performance Comparison Of Alternative Solutions For Web-To-Database Applications", October 2002,

http://rain.vislab.olemiss.edu/~ww1/homepage/project/mypaper.htm

[13] The Jakarta Site - Apache Tomcat, http://jakarta.apache.org/tomcat/

[14] Hans Bergsten, "Improved Performance with a Connection Pool", September 1999

http://www.webdevelopersjournal.com/columns/connection_pool.html,

Appendix A: Installation Guide

Windows NT/2000 is needed for installing the system.

Install and Configure Tomcat 3.3

1. Download Jakarta-tomcat-3.3 and unzip the archive into a directory.
2. Set up Environment variable
    - Pick up a control panel icon and click system. In System Properties, select
    - Advanced tag and click Environment Variables button
    - Set up a new system variable JAVA_HOME={jdk1.3.1 directory}
    - Set up a new system variable TOMCAT_HOME={tomcat installation directory}

Create ODBC System Data Source

1. Copy database login.mdb and weather.mdb to the location {tomcat installation directory}\webapps\weather_forcast\database.
2. Start the application ODBC in control panel.
3. Select the DSN tab and click the Add button to add a new data source.
4. Select the Microsoft Access Driver and click Finish button. ODBC Microsoft Access Setup screen will be seen.

5. Enter the string" login" as the data source name and enter the location of login.mdb and click OK.
6. Repeat the above steps, enter string "weather" as the data source name and enter the location of weather.mdb.

Copy all the files under the directory of Tomcat

In order to make the system run appropriately, we should copy the files of the On-Line Weather Forecast System under correct directory of the web server:

- Move all JSPs for weather data processing subsystem to the <SERVER_ROOT>\webapps\weather_forcast\ directory. Move all JSPs for weather viewing subsystem to the <SERVER_ROOT> \webapps\ weather_forcast\ client\ directory.
- Make sure Service class, Controller class and ConnectionPool package is in the classpath of Tomcat <SERVER_ROOT>\webapps\weather_forcast\WEB-INF\classes\ directory.
- Copy all the compiled the service implementations and class files to the <SERVER_ROOT>\webapps\weather_forcast\WEB-INF\classes\ directory.

Startup Tomcat Web Server

Go to Tomcat installation directory and double click "startup.bat" to start the Tomcat server. Open your browser to the following URL: http://localhost:8080/weather_forcast/Login.jsp.

Appendix B: User Manual

Login Page

For security, only authorized user can go into the system. The login page (see Figure A1) prompts user to input its username, password. If user is not registered, Sign Up Page will be shown on the screen. Otherwise, user goes to index page of the system.

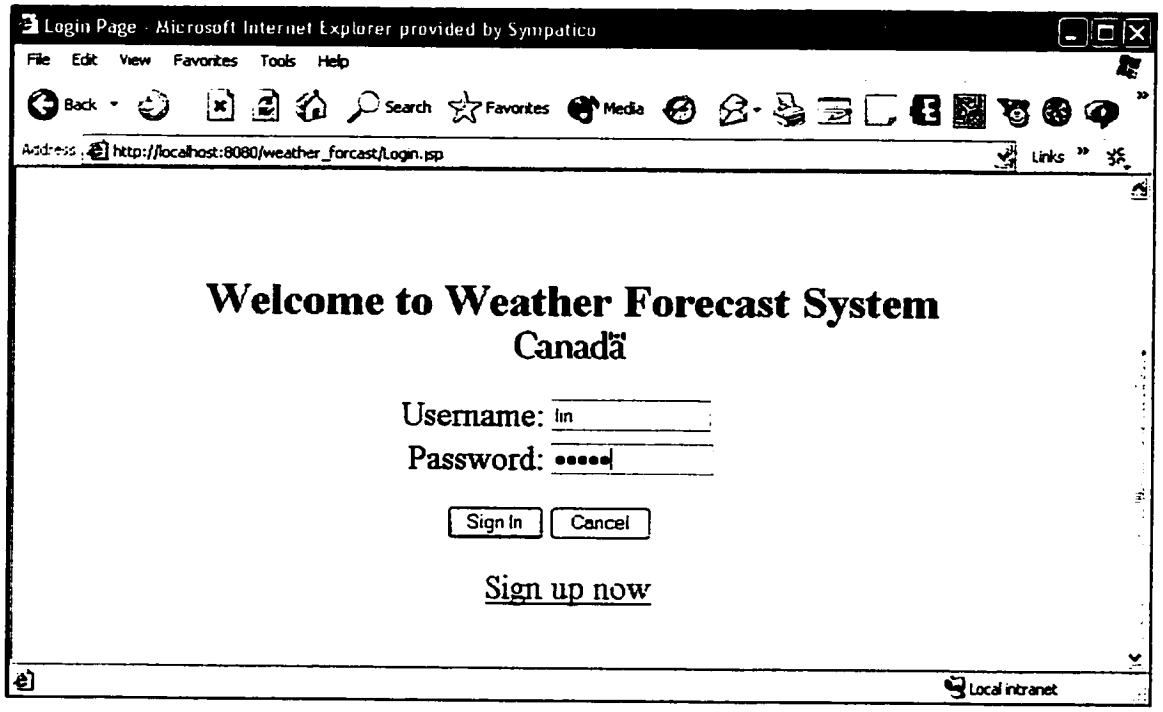

Figure A1: Login Page

Sign Up Page

This page (see Figure A2) provides a simple registration form for unauthorized user. It prompts user input username, password and reentry password. If user has existed or password is incorrect, a message" User has Existed" or "Please re-enter your Password" will be shown on the top of the page. Otherwise, the new user has successfully registered and goes to index page of the system.

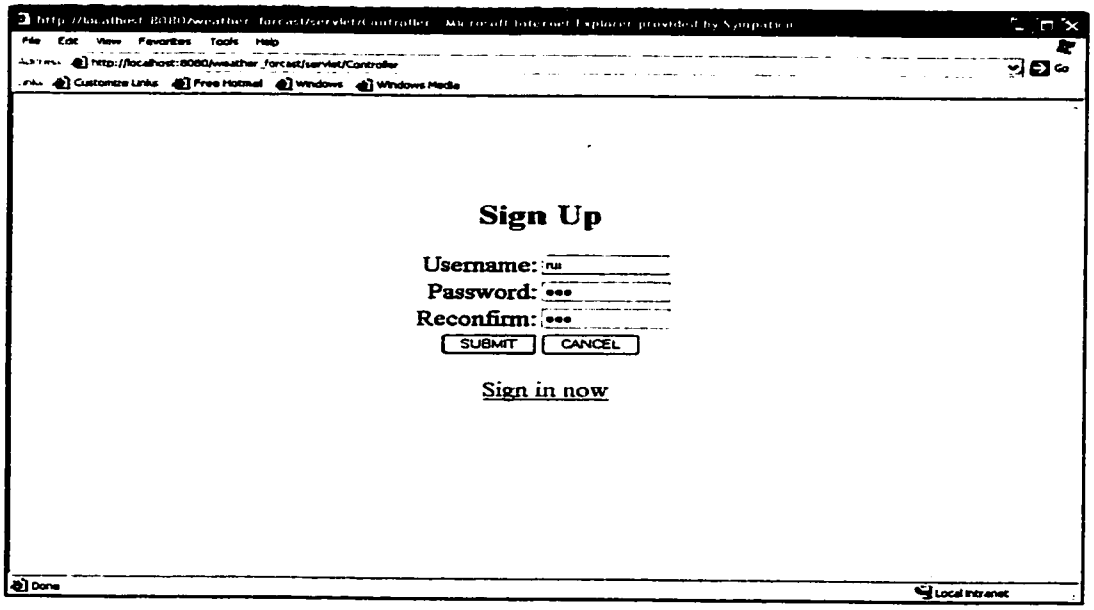

Figure A2: Sign Up Page

Index Page

1. DB-Administrator Index Page

If login is successful, DBA goes to index page of Weather Data Processing Subsystem.

2. Non-Administrator Index Page

If login or sign up is successful, non-DBA goes to index page of Weather Viewing Subsystem (see Figure A3).

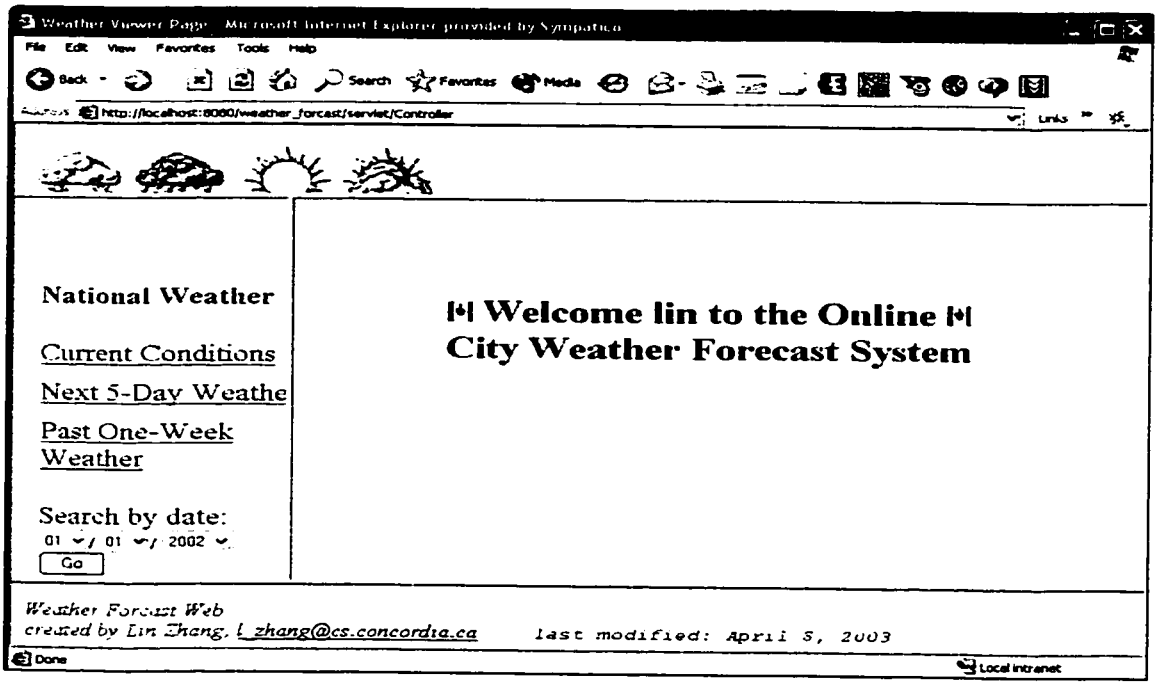

Figure A3: Index Page for Non-DBA

Add Record

To add a weather record, click *Extract Weather* in the navigation bar of index page, a city list is shown on the screen. Select one or more cities and click *Save Weather Data* button (see Figure A4). If submission is successful, message "*Weather data has successfully inserted into Weather Database*" is shown on the screen. Otherwise, an error message "*submission is failed*" is shown.

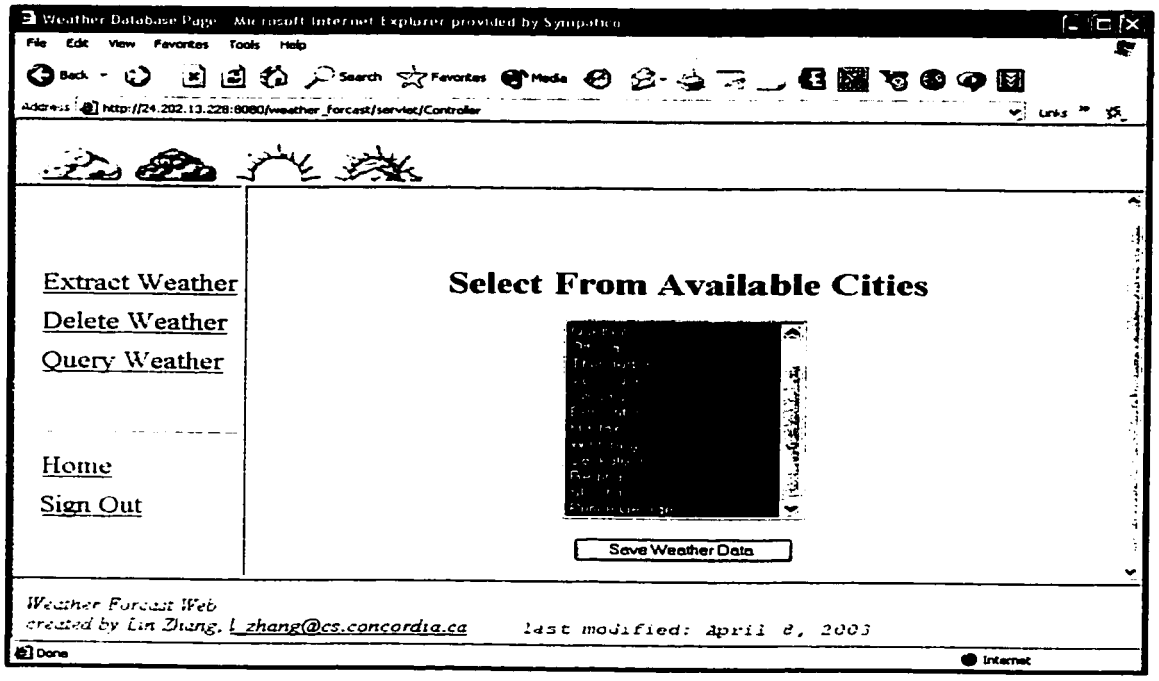

Figure A4: Add Record Page

Delete Record

To delete a weather record, click *Delete Weather* in the navigation bar of index page, a menu with three options (Figure A5) is shown on the screen.

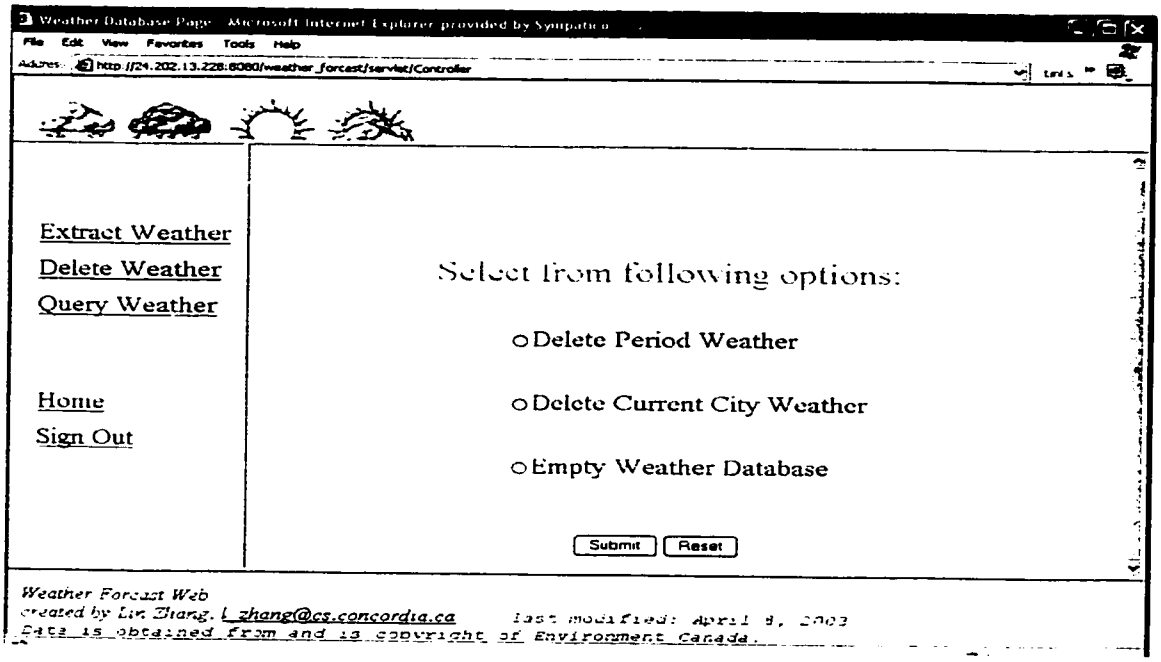

Figure A5: Delete Record Menu

1) Delete a Period of Time of Weather Records

Choose button *Delete Period Weather* from the delete record menu, click *Submit* button, an interface (see Figure A6) is shown on the screen. Select one city, enter starting date, ending date and click *GO* button. A record list (see Figure A7) with the chosen city from starting date to ending date is shown on the screen. Click *delete* to delete any record.

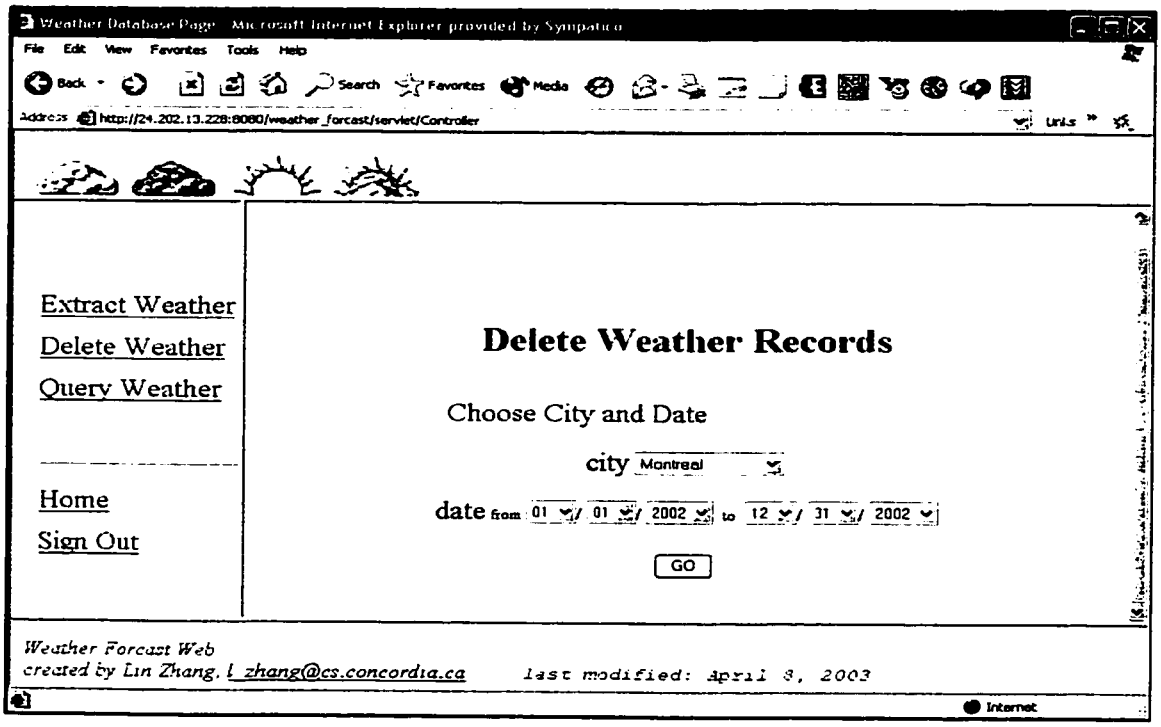

Figure A6: Delete Record Page

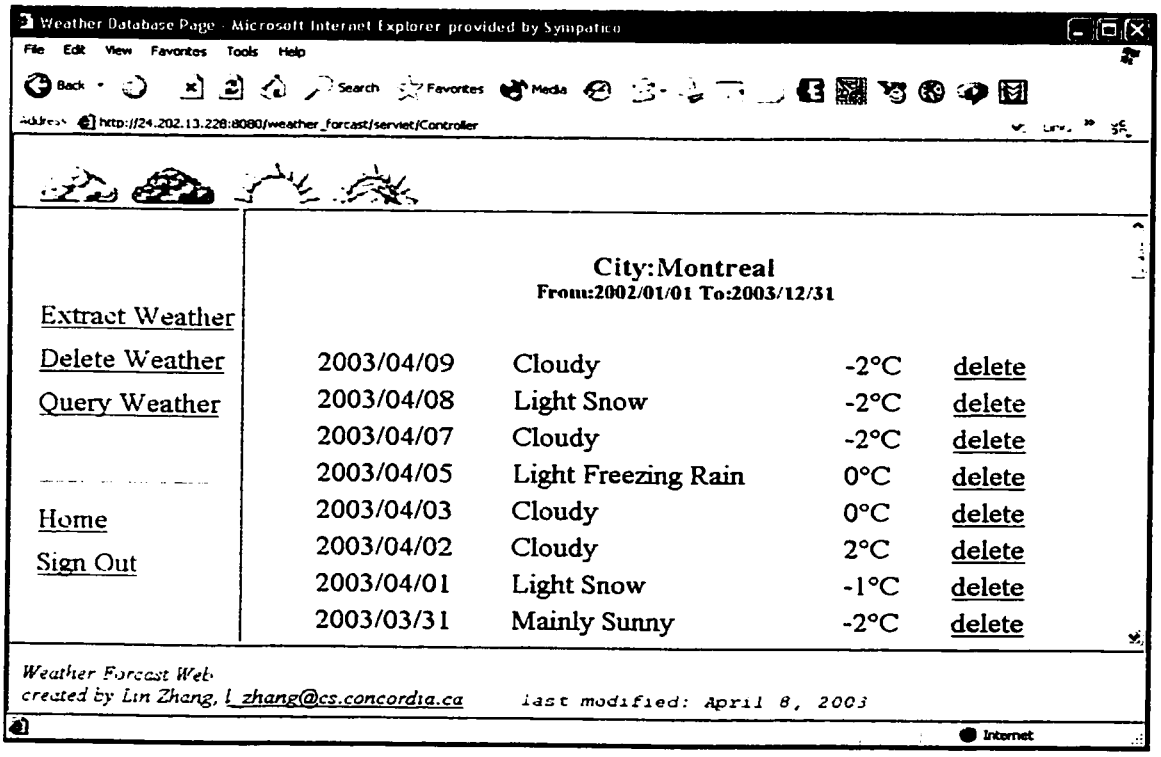

Figure A7: Record List Page

2) Delete Current Weather Records

Choose button *Delete Current Weather* from the delete record menu, click *Submit* button, a city list is shown on the screen. Select one or more cities from the city list and click *Delete* button, current weather records of selected cities are deleted from the weather database.

Query Record

To query a record, click *Query Weather* in the navigation bar of index page, a menu with two options is shown on the screen (Figure 8).

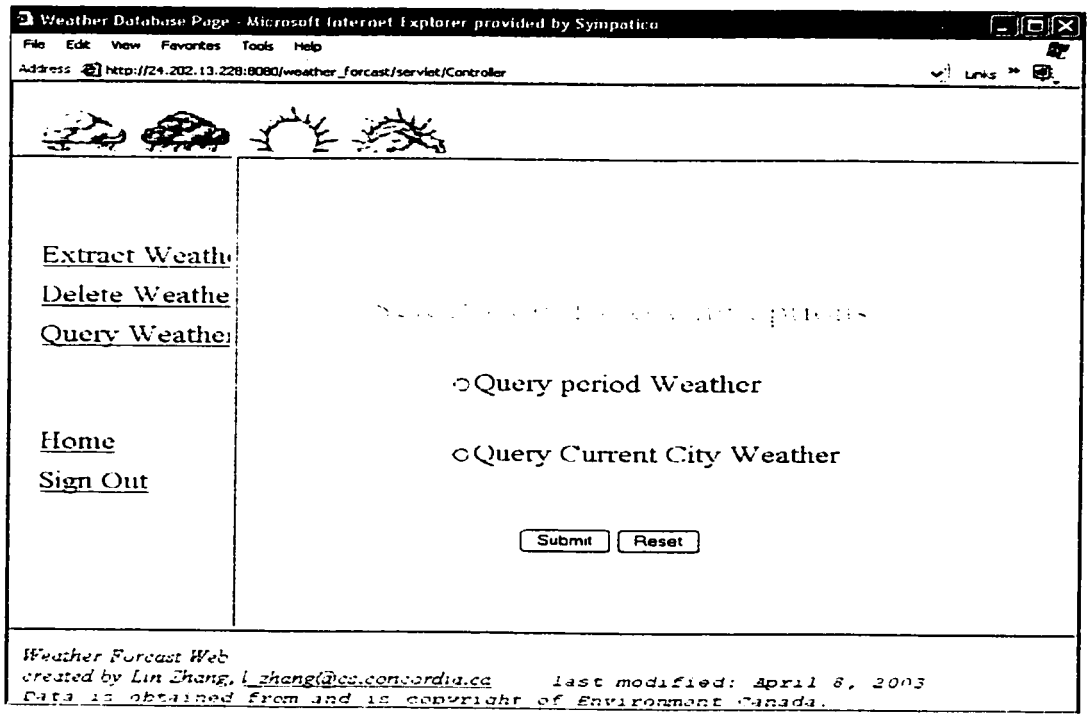

Figure A8: Query Record Menu

1) Query a Period Time of Weather Records

Choose *Query Period Weather* Button and click *Submit*, an interface (Figure A9) is shown on the screen. Select one city, enter starting date and ending date, click *GO* button.

A record list with the chosen city from starting date to ending date is shown on the screen. Click *Detail* to see the detail description of that day's weather.

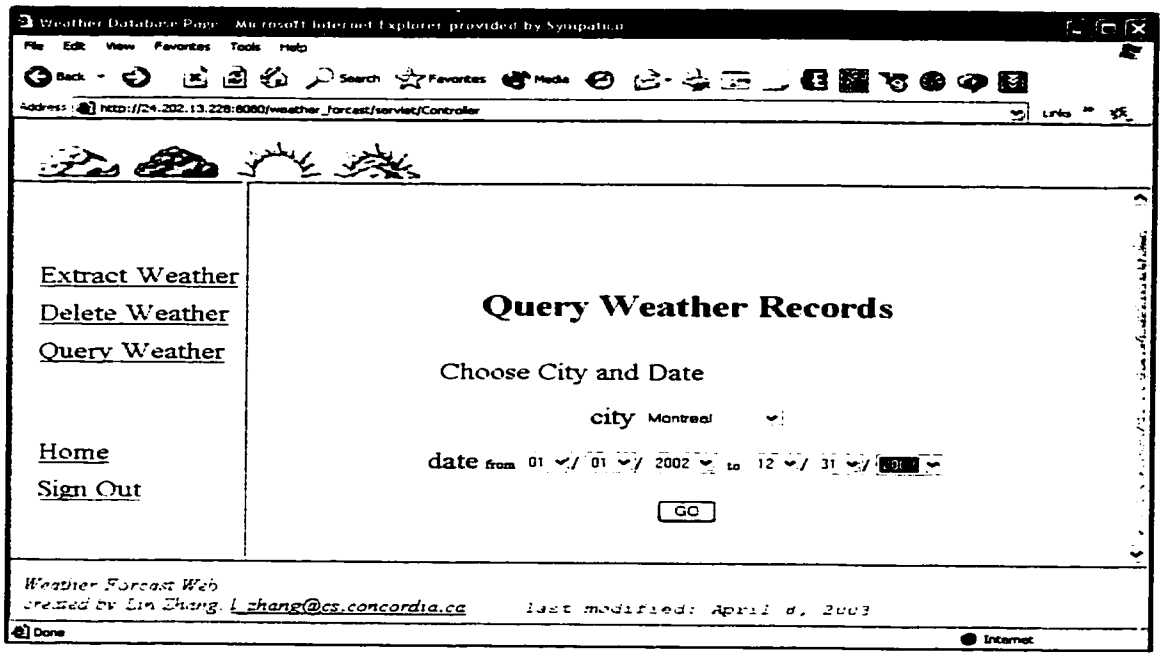

Figure A9: Query Record Page

2) Query Current Weather Records

Choose button *Query City Current Weather* from the query record menu, click *Submit* button, a city list is shown on the screen. Select one or cities from the city list and click *Display Current Conditions* button, current weather records of selected cities are shown on the screen (Figure A10).

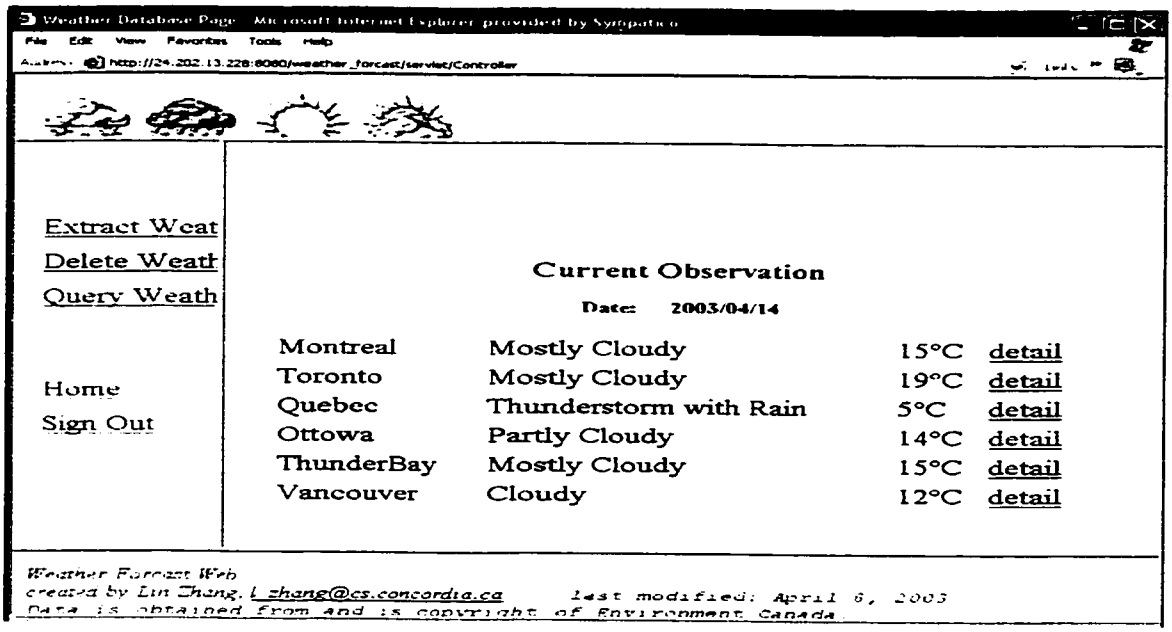

Figure A10: Current Weather Conditions of Selected Cities

View Today Weather

To observe today weather, click *Current Conditions* in the navigation bar of index page, a city list is shown on the screen. Select one city and click *Display Current Conditions* button. The detail observation of current weather is shown on the screen.

View Next 5-Day Weather

To observe next 5-day weather forecast, click Next 5-Day Weather in the navigation bar of index page, a city list is shown on the screen. Select one city and click *search* button. Next 5-day weather forecast is shown on the screen.

View Any Date Weather

To observe any date weather information, go to the navigation bar of index page, input date and click *Go* button, a city list is shown on the screen. Select one city and click *Search* button, that day's weather detail is shown (see Figure A11) on the screen.

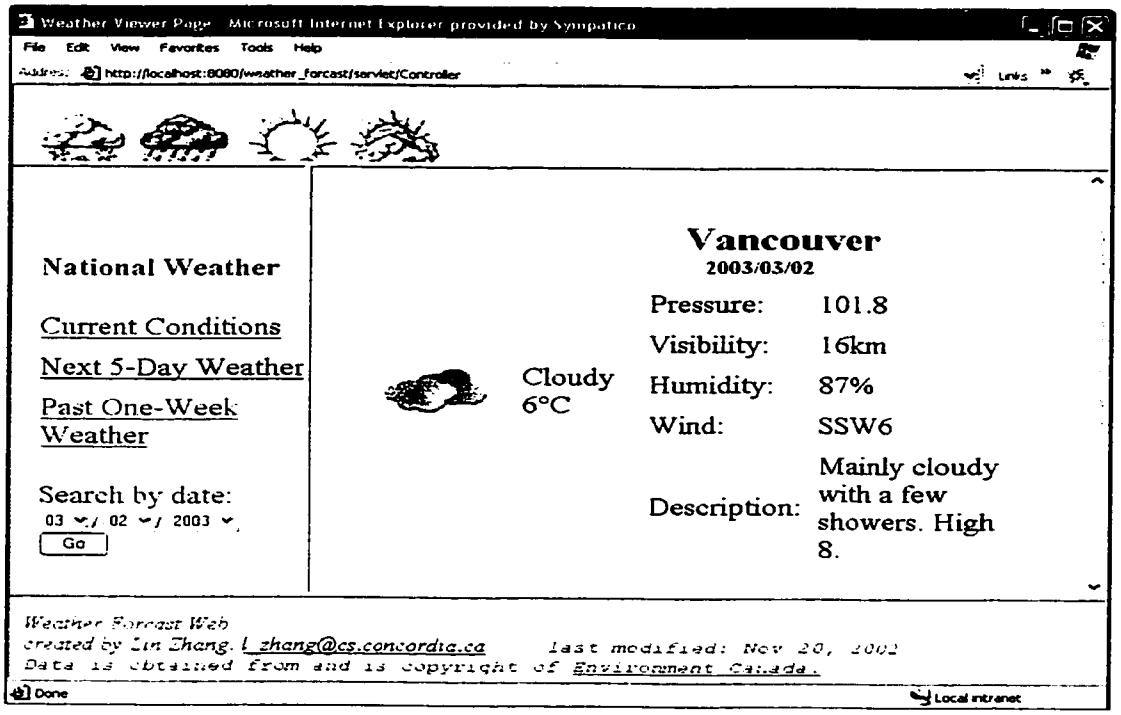

Figure A11: Any Date Weather Page