CMPE 275: Enterprise Software Components

Prof. John Gash

Report on

Project 1 : CLIMSPACE

Submission by team ABP,

Anupama,Patil 008083062

Babu,Thomas 007669116

Prasanna Raaj Kumar 008027747

Table of Contents

Project description:

We are assigned the role of an architect and lead investigator for WeatherSpot Company. WeatherSpot is participating in a joint project with the California Agriculture Planning Agency (CAPA), SJSU, and the WMO.

We have designed a system to provide weather data storage and management. We provide a public access tool for organizations to access current and historical weather data for use in agriculture and city/county planning. CAPA stores the data archives and request systems. Weather monitoring and collection is an ongoing task. WeatherSpot has setup a MOU with NWS to receive updates of their weather data. Also small mesonets or individual stations can upload data into the system. This crowd-sourcing inspired feature (upload data) is a prominent capability of the system. SJSU is advocating a dashboard view of current weather data in comparison to climatology data so our system produces files (kml) that can be visualized using Google Earth .Our system also supports Json output format for future usage.

Users of Climspace can be a general user or an analyst . General user is one who is requesting current weather details which is small in size. Analyst can request current or history of weather data which can be very large in size.

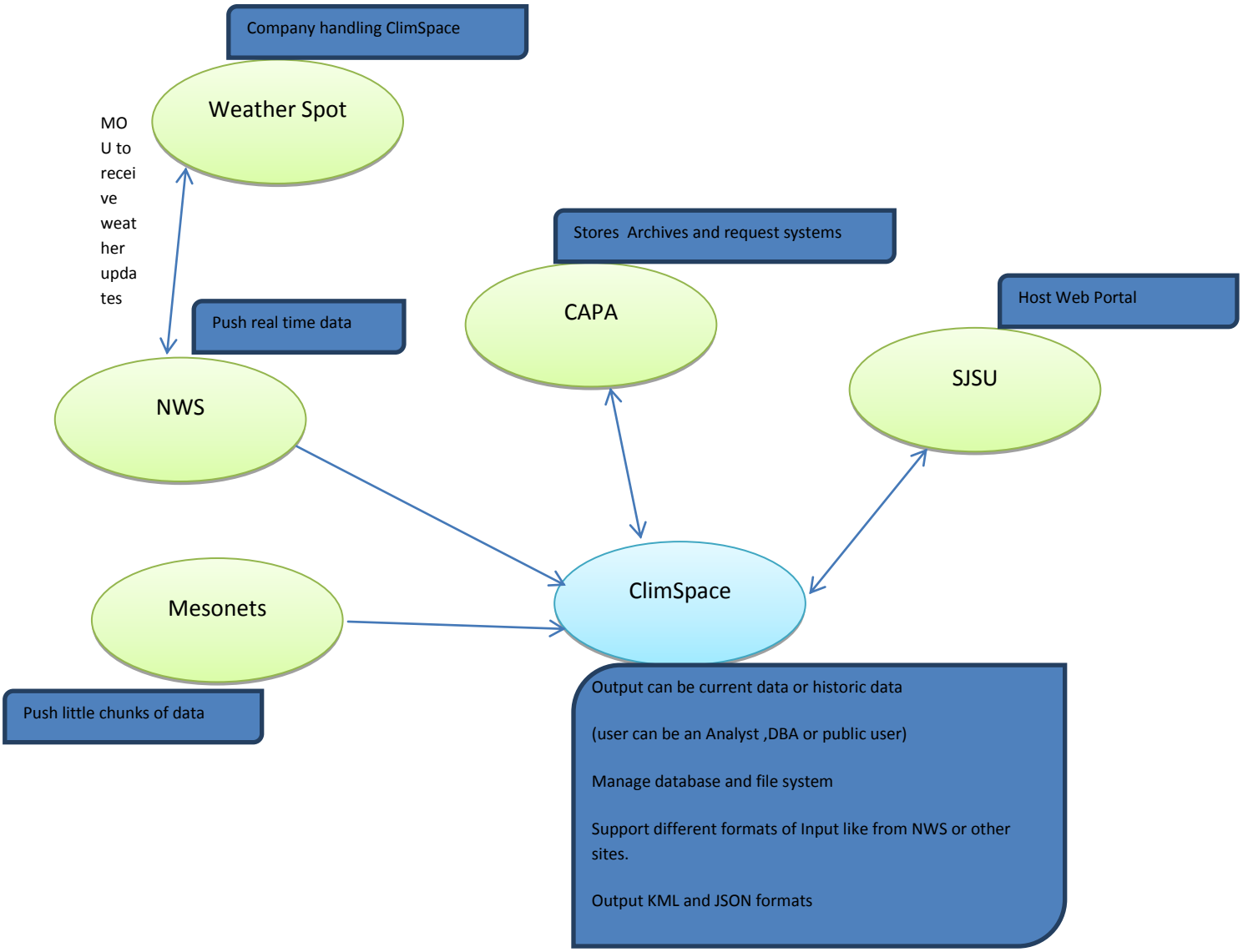


Figure 1 : Above diagram shows our initial requirement analysis.

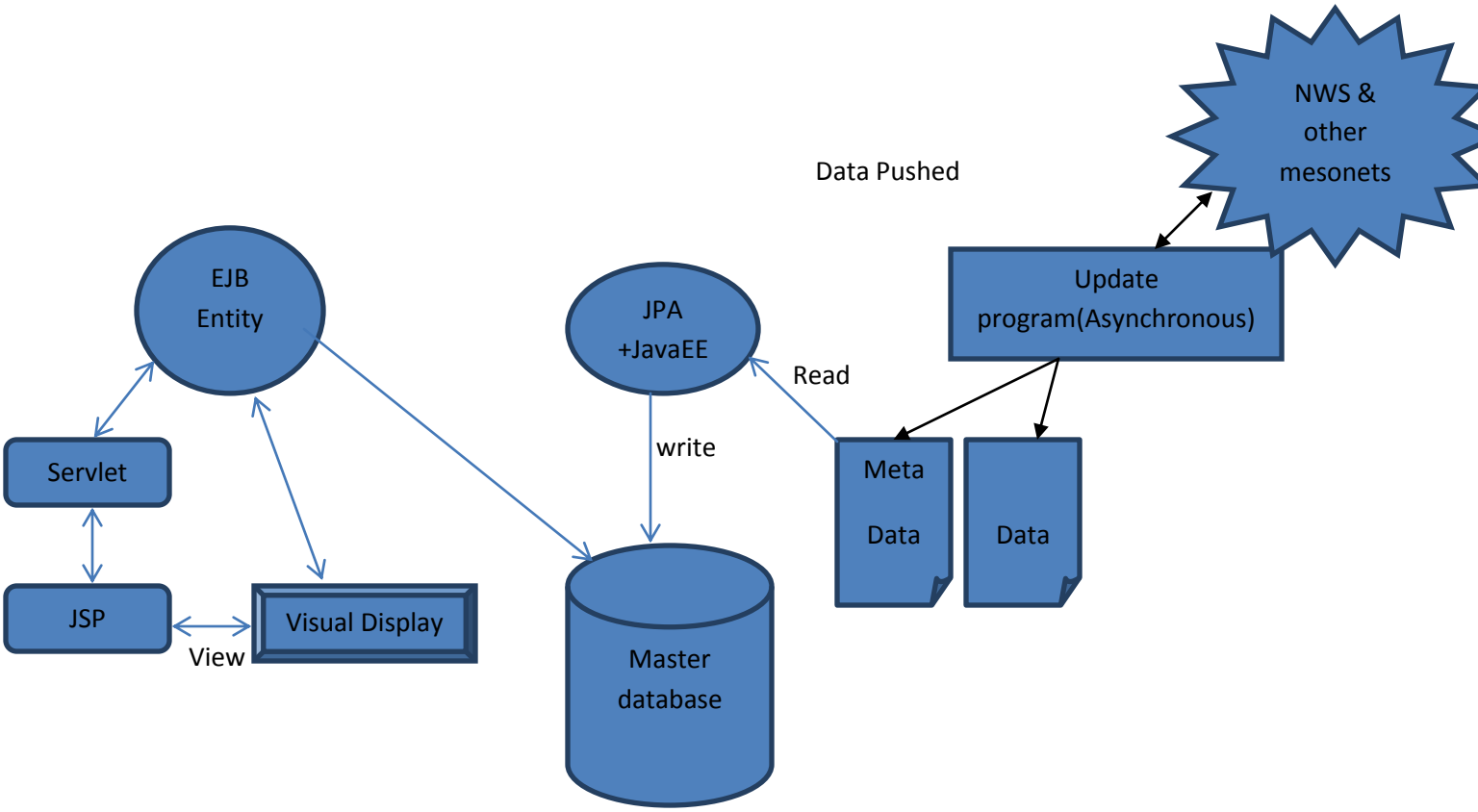


Figure 2 :This figure shows our initial understanding of the ClimSpace system.

System Architecture

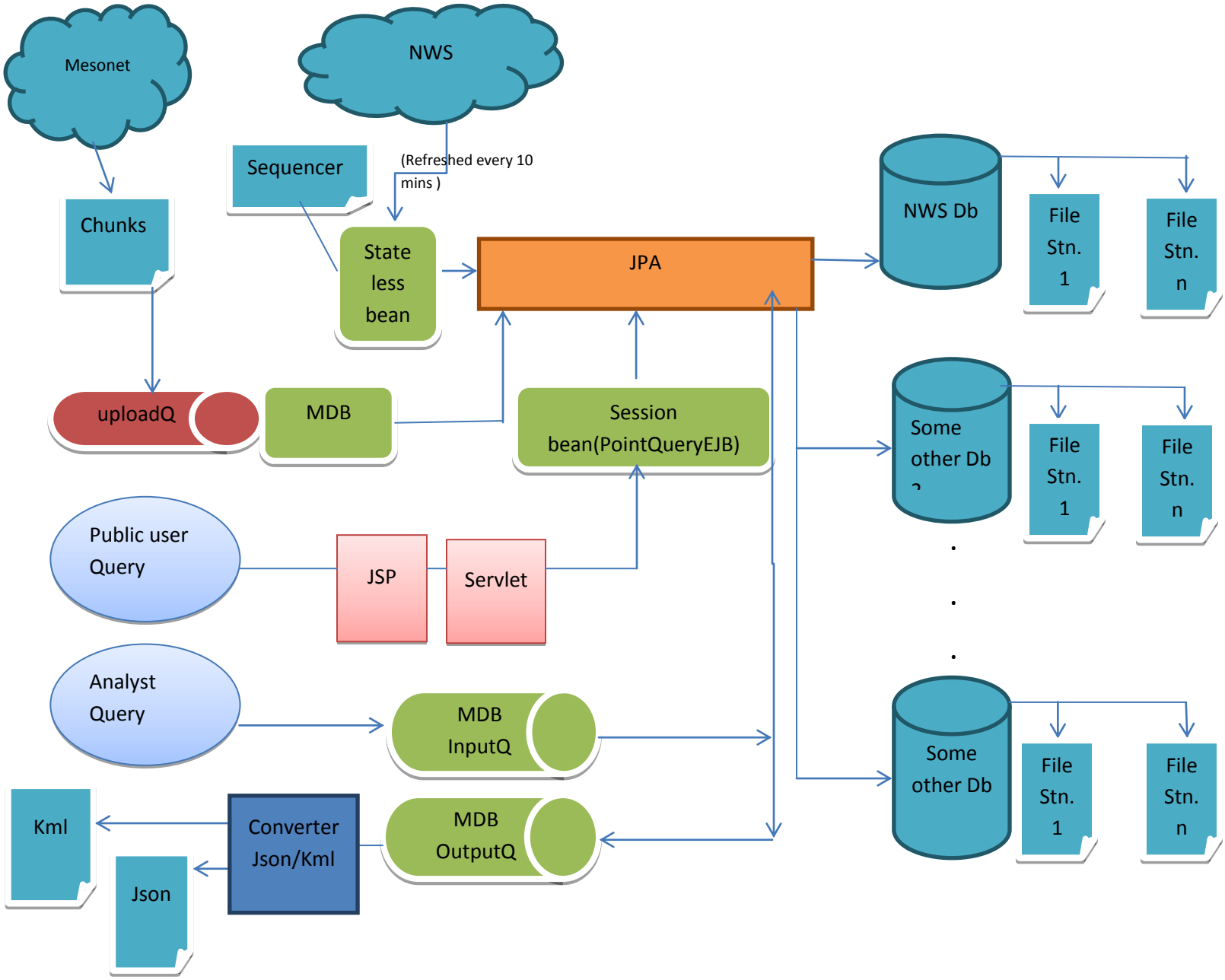


Figure 3: Final design of ClimSpace.

Reasons for chosen Architecture :

We are using Message driven beans to get data from mesonets because the data can come at any time and MDB support asynchronous message management. We are using stateless beans for data upload from NWS as this is a periodic process i.e for every 10 minutes or so. We use session bean for public user query as this process is again stateless. Analysts often need huge data to investigate how to manage the use of water for Northern California agriculture and household use. The analyst queries output is large in size and it is not real time i.e it is asynchronous so we use MDB queues to handle these queries.

Our goal for the design shown in Figure 3 , is to allow the system to be capable of accepting different file formats .The system looks for station name or id and it can query the data. We use a relational database because the user should be able to access data quickly and multiple users can query the data simultaneously. We are using file system to store the archive .Separating the data into current and archive data simplifies the system design. Program uses files as inputs to simplify the design and for easy testing. It also helps the system to be independent of the input format. The input can come in many different formats and so using files is a better option. Currently we can use comma separated or space separated files in our system. We also support parsing zipped files.

We have two tables in the data base. One is the **metadata** and one is the actual **data** table. Metadata is uploaded only once at the starting of the application. Client program (ant load) will upload first and sequence one metadata read and further periodic reads. For data read , once data is read it is stored into both current database and file. Each station has different achieve file . By doing this we have a scalable and efficient system. If a new system like NCDC or other weather websites's data can to be managed easily. Also for example if we have a query for mesowest then only that database is accessed which makes efficient and quick searches too.

Each station file is organized on date basis. For example the folder structure will be like data -> 2012- > Mar -> 04 -> station files. An analyst can specify station names and number of days. So if number of day is given 10 today , program takes station data from first 3 days in march and last 7 days in February.

Output of this system supports KML and JSON formats. We did it because it was one of the requirements from the client. Using KML files we can easily display the weather details on Google Earth. More benefits of KML and JSON are listed in the "technologies used "section. We are creating a servlet to enter station name/id and format details. SJSU can host these files to display the output for the users. We have a servlet, this servlet is just of the basic testing of our pilot system and to see if it works properly and gives proper results for the users. Analyst can use the KML or the JSON format files to get the required data. Our system can be further improved to support more encodings or representations easily.

Technologies used

SubVersion : Sub version is open source version control software. We can share project online. We used subversion to keep our code at one place and collaborate. We used code.google.com as repository location for our code. Older version and the changed versions can be compared using subversion. It is easier to debug the changes made in the code using subversion as we can easily compare the two files side by side with the changes highlighted in the newer version of the file**.**

JPA : JPA stand for Java Persistence Architecture . JPA was defined as part of EJB3. JPA is specification for accessing, persisting, and managing data between Java objects / classes and a relational database. It is easier to model objects to database using JPA. We have used hibernate framework for JPA.

JBOSS: JBOSS is open source application server. We are using JBoss 6 for this project. JBoss is based on Java so it is platform independent. It also implements Java EE part of Java along with server implementation. Jboss is used to develop and deploy EJBs, Web applications and services and portals.

Apache Ant: ANT stands for Another Neat Tool. Ant is a tool to manage, run compiling, jarring and version control, documentation, and testing. Apache Ant is used to build, deploy and test the code. Ant manages the build process and source dependencies. ANT takes instructions in XML format.

EJBs: EJBs stands for Enterprise Java Beans .EJBs are used to develop and deploy robust, portable and component-based distributed applications that are scalable and consistent. EJB is a server side model that encapsulates the business logic .There are three types of EJBs – Session Beans , Entity Beans and Message driven Entity beans.

KML and JSON :

KML or KMZ files are similar to XML format and these files are used for geographical display purposes. Many applications display KML, including Google Earth, Google Maps, and Google Maps for mobile, NASA WorldWind, ESRI ArcGIS Explorer, Adobe PhotoShop, AutoCAD, and Yahoo! Pipes. Data can be easily interpreted when it is in picture format.

JSON stands for JavaScript Object Notation, it is a low-overhead alternative to XML.JSON is easily understandable to both humans and machines. It is programming language independent. String, Boolean ,array ,number etc. types can be used in JSON.

JUnit :

JUnit is a program used to perform unit testing of virtually any software. JUnit testing is accomplished by writing test cases using Java, compiling these test cases and running the resultant classes with a JUnit Test Runner. JUnit API is used to generate test cases for a project. We have written the test cases using JUNIT which are in the source code.

Requirements for the system

Following installations are required for this software to run:

JBoss 6.1.0

Apache ant 1.7.0

JDK 1.6.0

Google earth

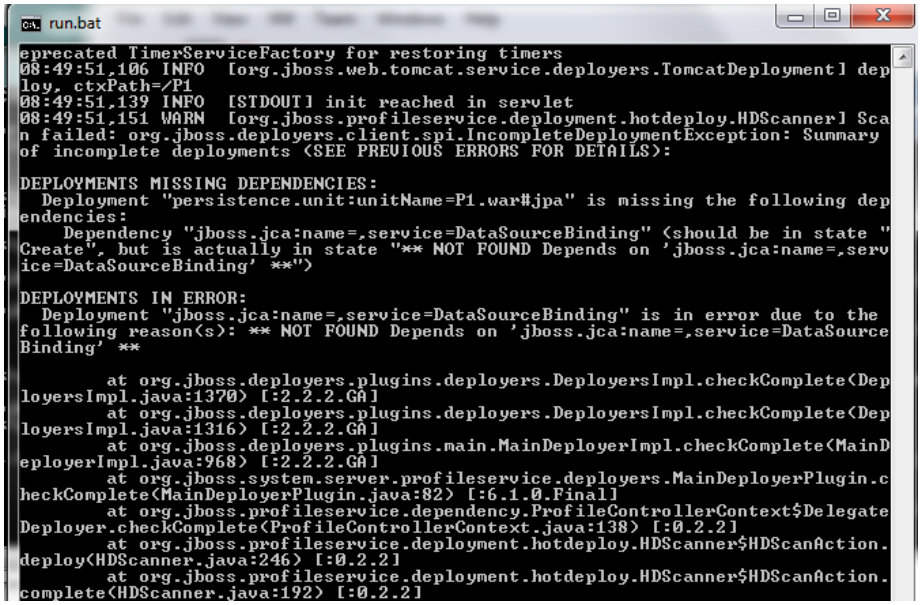
Internet

Limitations of the System

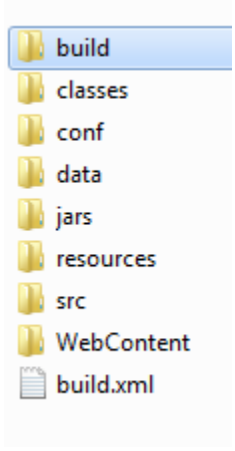
Output display is only for selected columns and not all the fields that are in the file.

Analyst queries are defined for specific stations and further can be sorted down on the dates but we haven't done the time based sorting of the files.

Exception on deployment – The following deployment error has to be resolved



Folder Structure



Install notes

Installation notes

----

1. unzip abp-275-project1.zip to a folder. This will lay down the following folders and files

build - points to "upload-queue-service.xml"

classes - the compiled classes reside here, once the ant script is run

```
    conf            - contains
                            log4j.properties
                            persistence.xml - // change the mysql database url and user/password here

    data            - contains 3 sub folders
                        - fileArchive - //the archive files are stored here
                        - outFiles - the .kml and .json files after and analyst query and point query
                        - realtime - here the nws real time files such as "mesowest,gz" and "mesowest_csv.gz"

   jars/rss         - the external jars such as
                                    'gson-2.1.jar', 'JavaAPIforKml.jar', and
                                      'rssutils.jar' required for compilation

   src              - the java sources
                     - com.client - contains the client programs for "mesonat upload",
                                         "point    query", "range query" and "Data load"(NWS feed)
                     - com.controller.servlet -servlet clode and the client to integrate
                                         with PointQueryEJB
                     - com.ejb.query - contains all the EJB code for query purposes
                     - com.ejb.upload - contains all the EJB code for upload purposes
                     - com.entity.jpa - contains all the jap code for interacting with database as well as NWS
                     - com.ejb.query - contains all theFile read and write code for json and kml file
                                          Read/writes
    Test             - contains JUnit test cases.

    WebContent
                    - contains the query.jsp and web.xml for servlet configuration

    build.xml       - the build file to compile, deploy and initaiting NWS real time feeds the sources


Run notes
---------
pre-requisite - Make sure JAVA_HOME is pointed appropriately in batch files

1. run 'ant build' from installed folder. This compiles the sources to the 'classes' directory.

2. run 'ant war'
            - creates a P1.war file
            - deploys the P1.war to running Jboss

3. run 'ant nws'
            - starts the initial sequence to
                - connect to "http://mesowest.utah.edu/data/mesowest.out.gz" and initiate metadata  read on
startup(once)
                - connect to "http://mesowest.utah.edu/data/mesowest.csv.gz" and initiate data read on every 10
minutes

4. run 'ant analyst'
            - runs sample to do a analyst query and stores read data to file located in .\data\outFiles

5.run 'ant mesonet' -
                  - updates mesonat data to the system.

6. run 'ant point' - gives the output for public user
```

Running

Make sure that the following jars are present in <JBOSS>/deployments folder

mysql-connector-java-3.1.6-bin,

gson-2.1.jar

JavaAPIforKml.jar

Start Mysql server

Add the following contents in jboss-6.1.0.\server\default\deploy\hornetq\hornetq-jms.xml

```
<queue name="station">
<entry name="/station/upload"/>
</queue>

<queue name="queryin">
<entry name="/query/in"/>
</queue>

<queue name="queryout">
<entry name="/query/out"/>
</queue>
```

User manual:

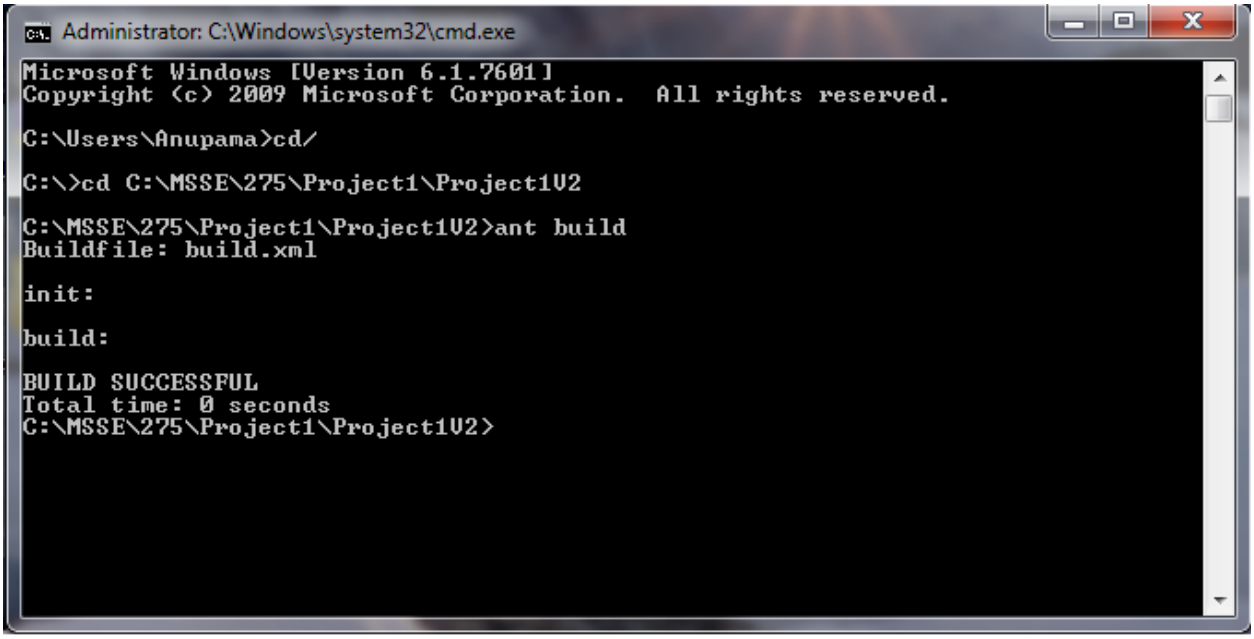
1. Extract the zipped folder CMPE275_Project2_ABP .zip and place it in C:\ CMPE275_Project2_ABP
2. Open the folder and change the values of Jboss home and project folder values to your respective locations of jboss and the project folder.

```
<!-- JBoss 6.x -->
<property name="jboss.home" value="D:/Sem2/CMPE275/work/project/jboss/jboss-6.1.0.Final" />
<property name="home" value="D:/Sem2/CMPE275/work/project/delme/try2/" />
```

3. Change username and password values in conf/persistence.xml to access your mysql server

```
<properties>
    <property name="hibernate.connection.url" value="jdbc:mysql://w2003r2en/db1"/>
    <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
    <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
    <property name="hibernate.connection.username" value="root"/>
    <property name="hibernate.connection.password" value="password"/>
    <property name="hibernate.hbm2ddl.auto" value="update"/>
    <property name="hibernate.show_sql" value="false"/>
    <!--
```

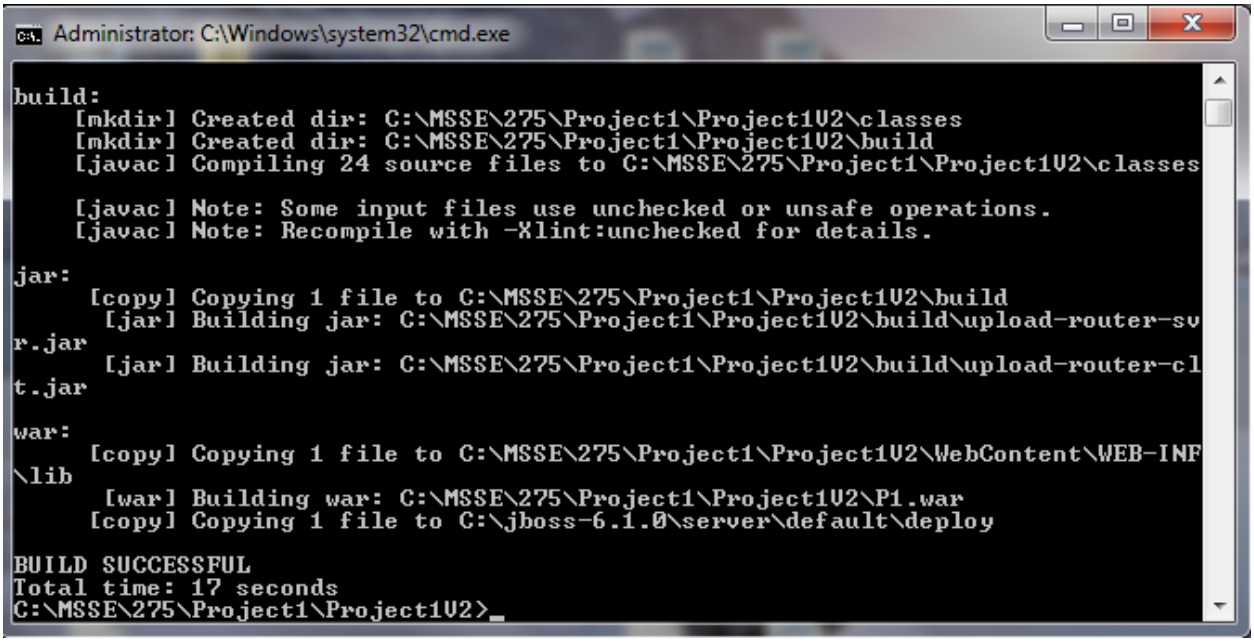
4. Open Command prompt and change directory to point to your Jboss/bin directory. Type run.bat. This should start the Jboss server
5. Open another command prompt and type ant build
6. You should get the BUILD SUCCESSFUL message as shown below .



7. To deploy the application

   >ant war

This will deploy the war file and display BUILD SUCCESSFUL message as shown below.



Deployed EJBs are shown as below

Three MDBs.

One for "upload Mesonat", - UploadEJB

Two for "analyst Query"(One for InQueue and One for OutQueue)

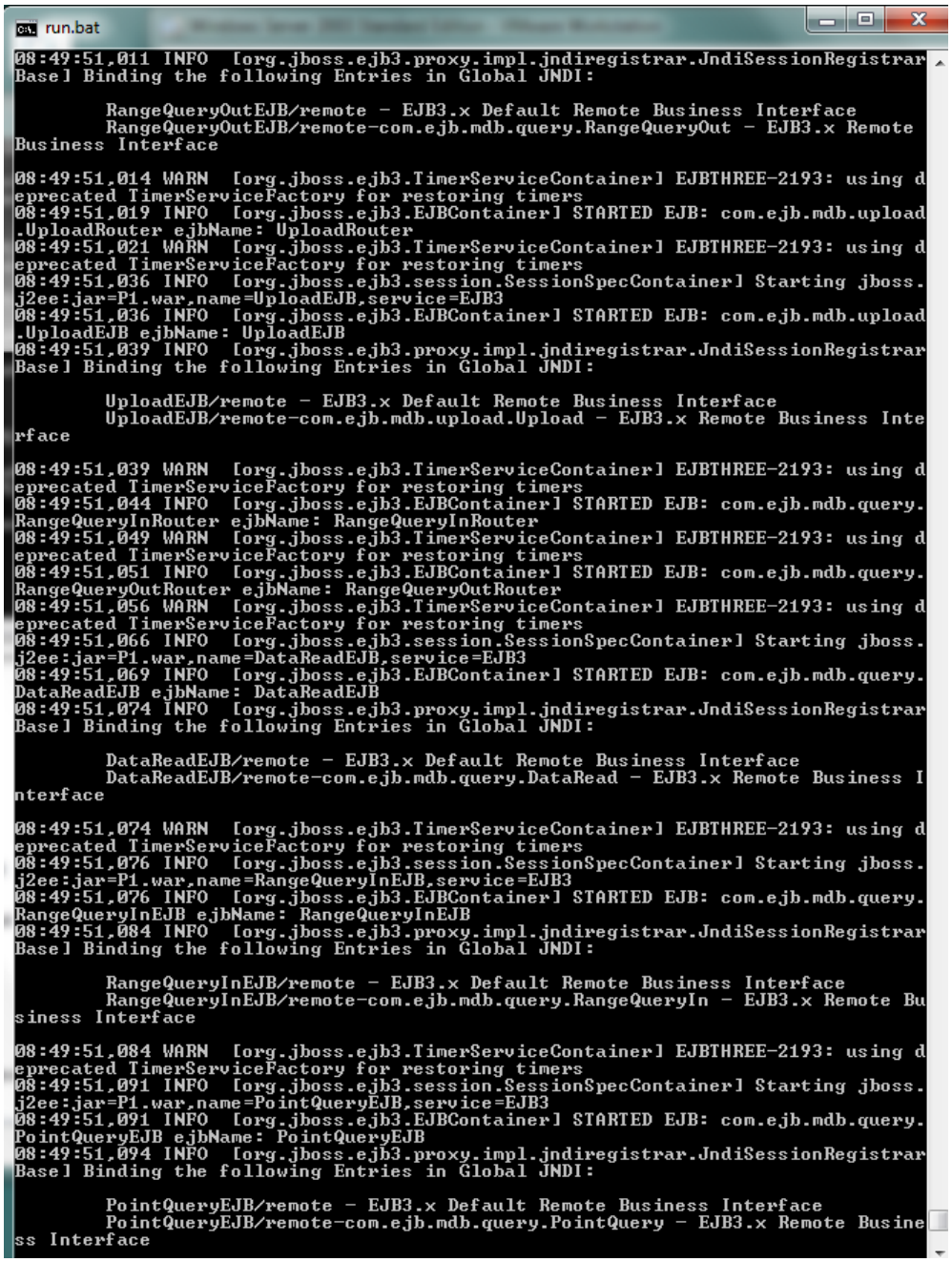
RangeQueryOutEJB and RangeQueryInEJB

Two Session Bean – One for “Live Feed from NWS” and One for “Point Query”

DataReadEJB

PointQueryEJB

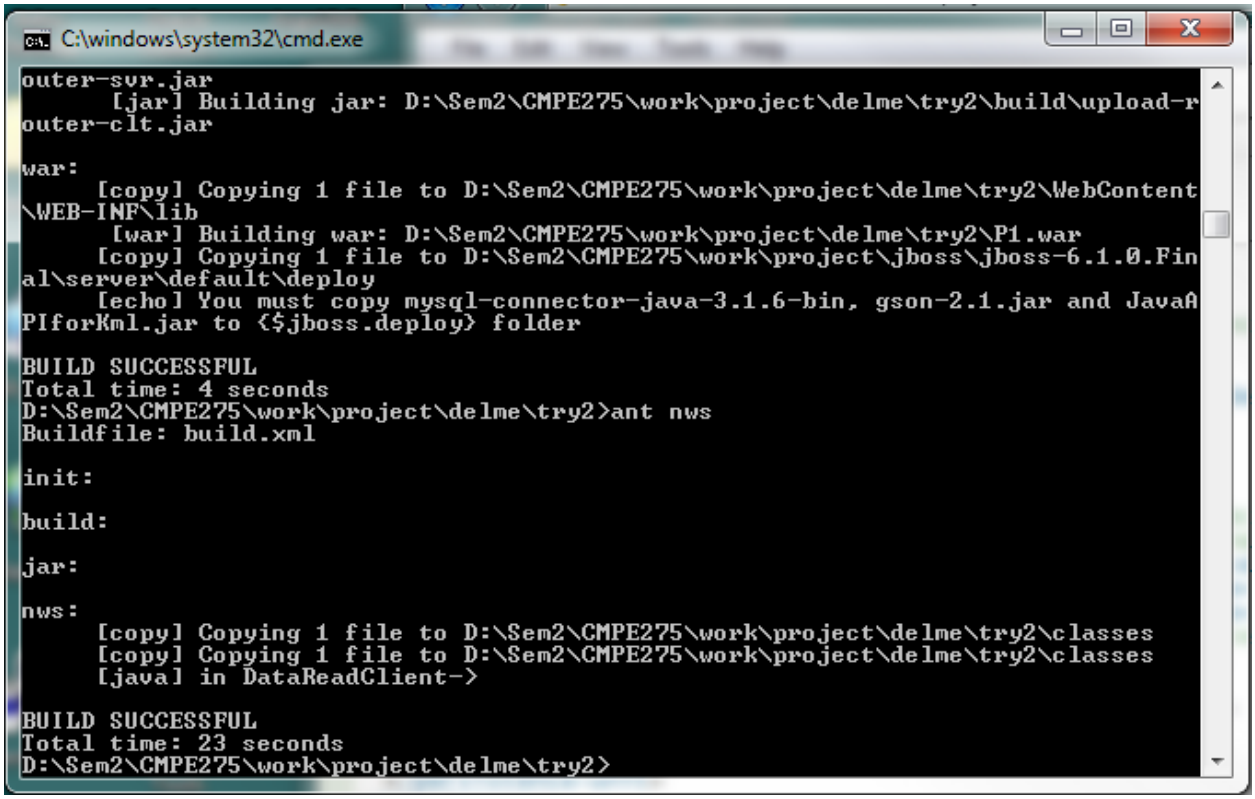
Deployed EJBs are shown below



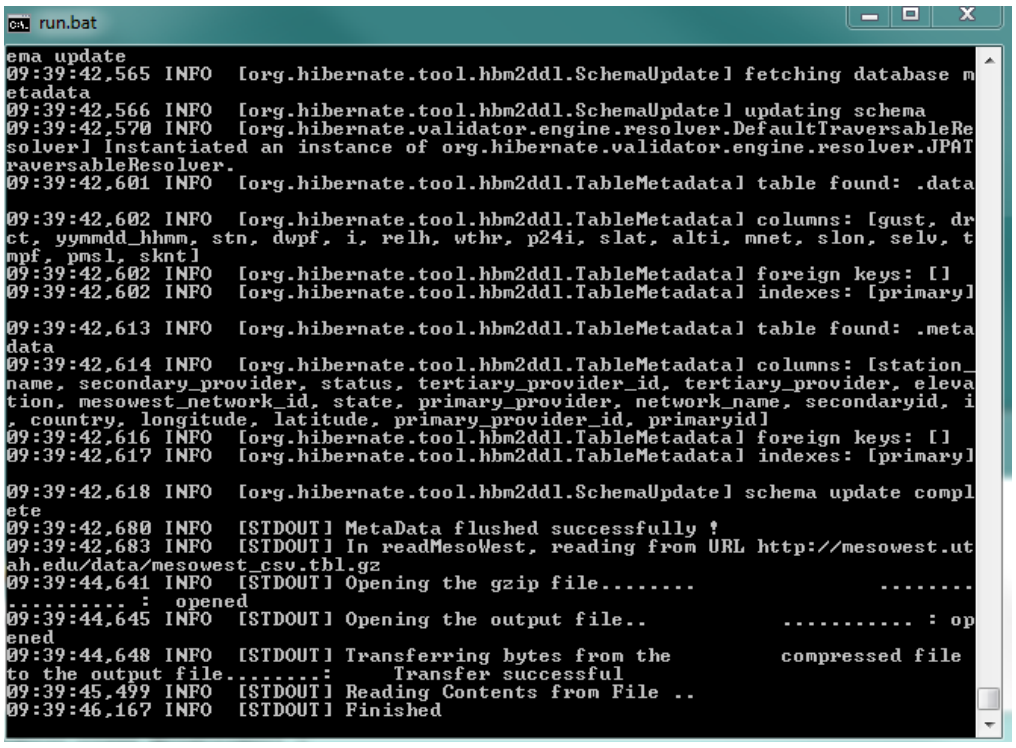
8. Start real time feeds by

   >ant nws

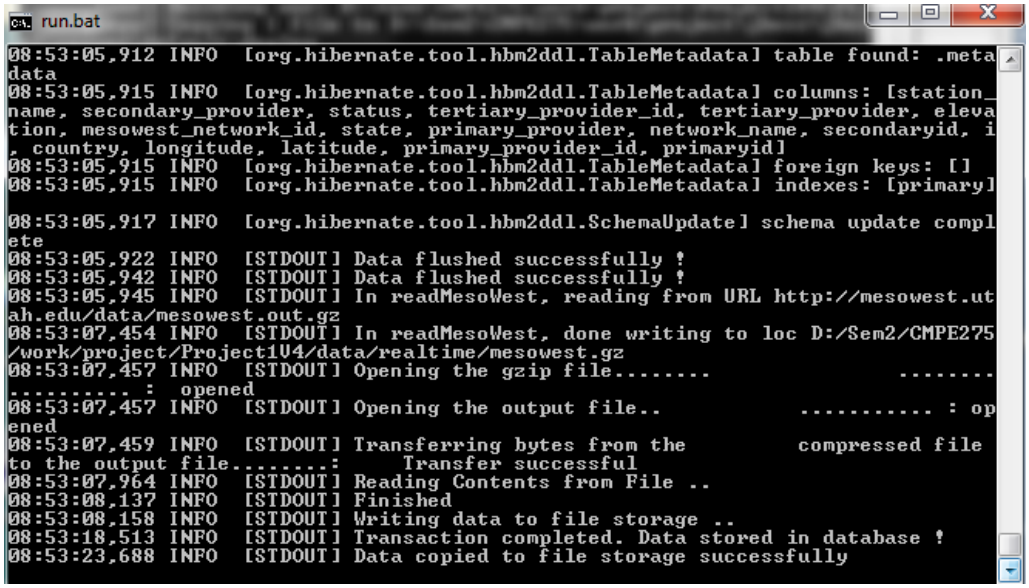
   This should start to get the metadata and data from the live feeds



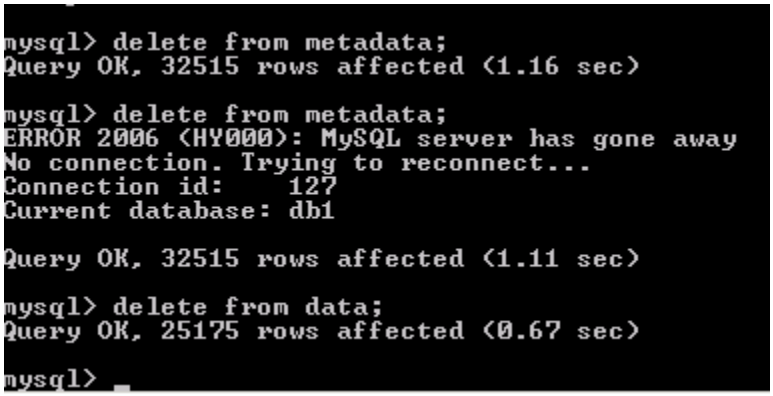
MetaData read initiated



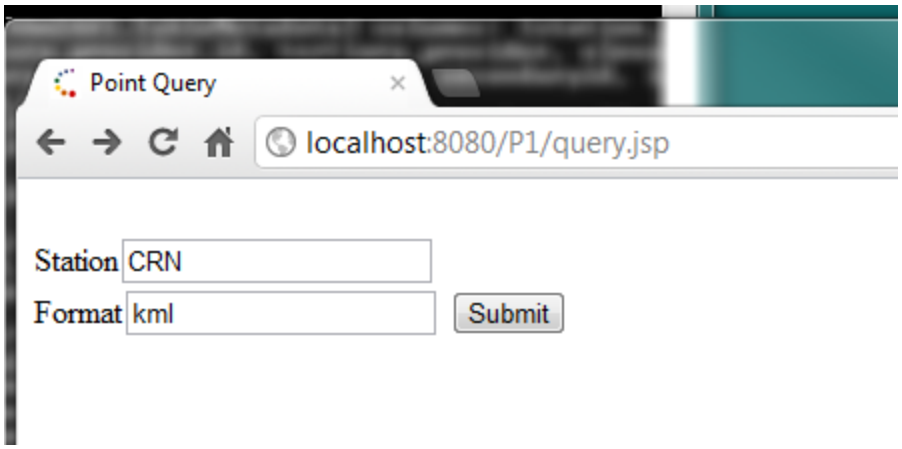
DataRead initiated

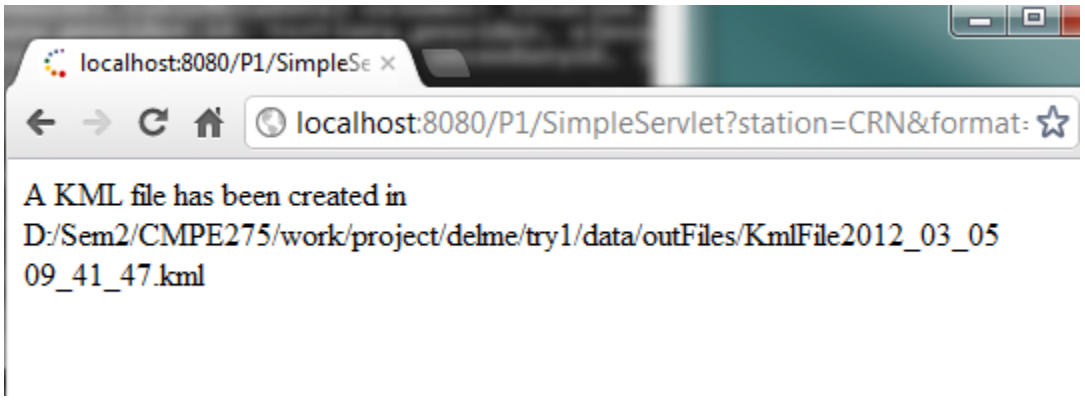


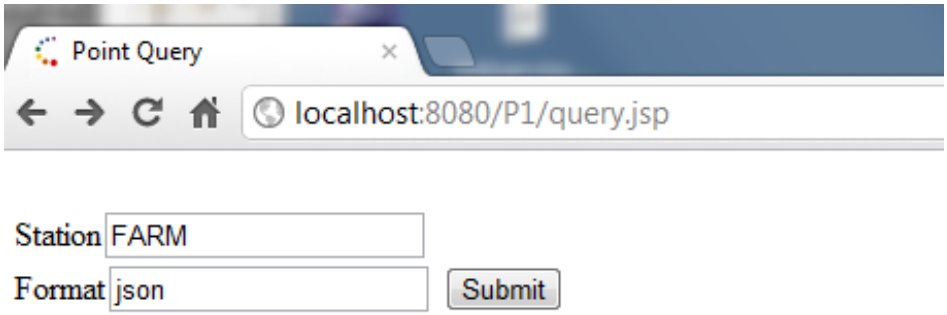
database update



9. Open the browser and type http://localhost:8080/P1/query.jsp . This will display the servlet for public users .The user can enter the station and the format (json or kml).


Point Query
localhost:8080/P1/query.jsp
Station CRN
Format kml
Submit


localhost:8080/P1/SimpleSe
localhost:8080/P1/SimpleServlet?station=CRN&format=
A KML file has been created in
D:/Sem2/CMPE275/work/project/delme/try1/data/outFiles/KmlFile2012_03_05
09_41_47.kml


Point Query
localhost:8080/P1/query.jsp
Station FARM
Format json
Submit


localhost:8080/P1/SimpleSe
localhost:8080/P1/SimpleServlet?station=FARM&format=json
{"latitude":"
-111.891","longitude":"41.004","station":"FARM","state":"UT","temp":"58.11","gust":"11.35"}

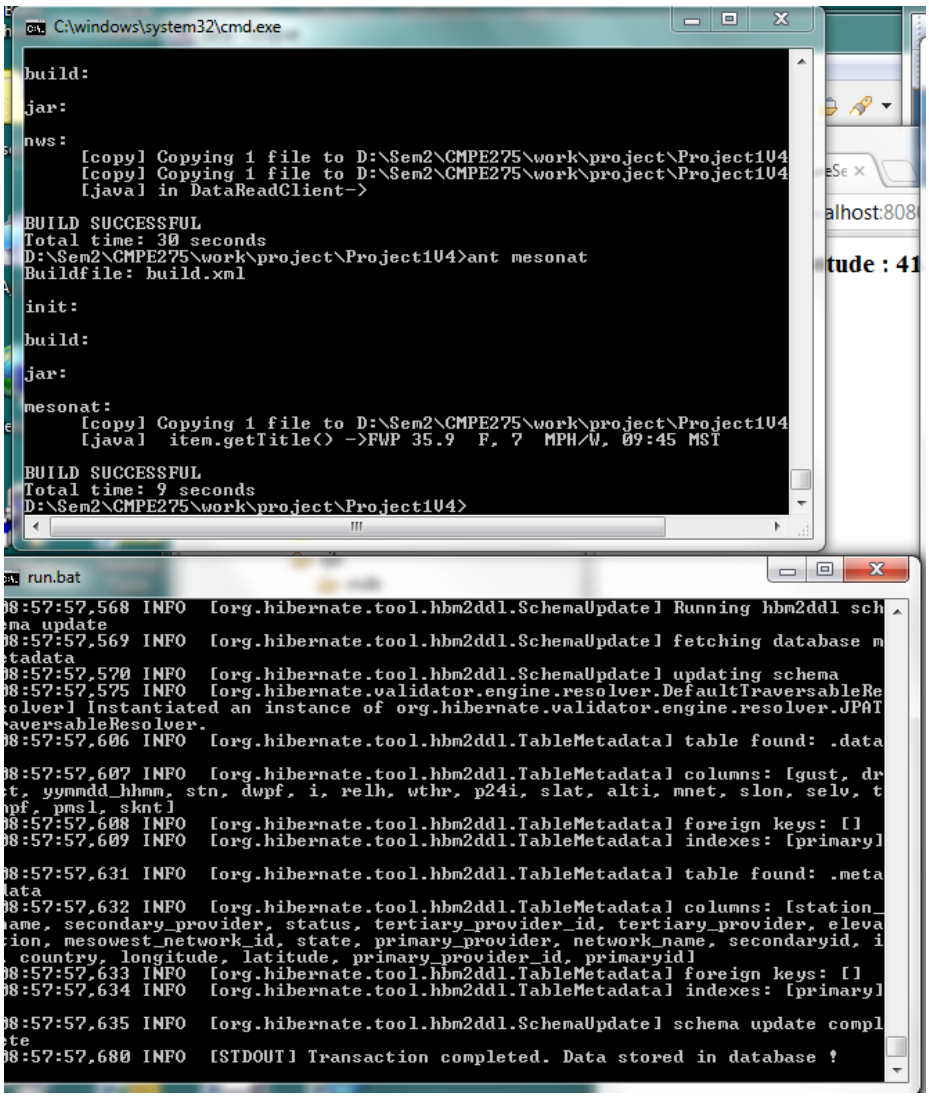
Contents displayed in jsonfile for public user is as follows,

{"latitude":"41.004","longitude":"111.891","station":"FARM",”STATE”:”UT,"temp":"58.11"}

Above output will be displayed depending on your choice of display format. Following is the kml file rendered on google earth.

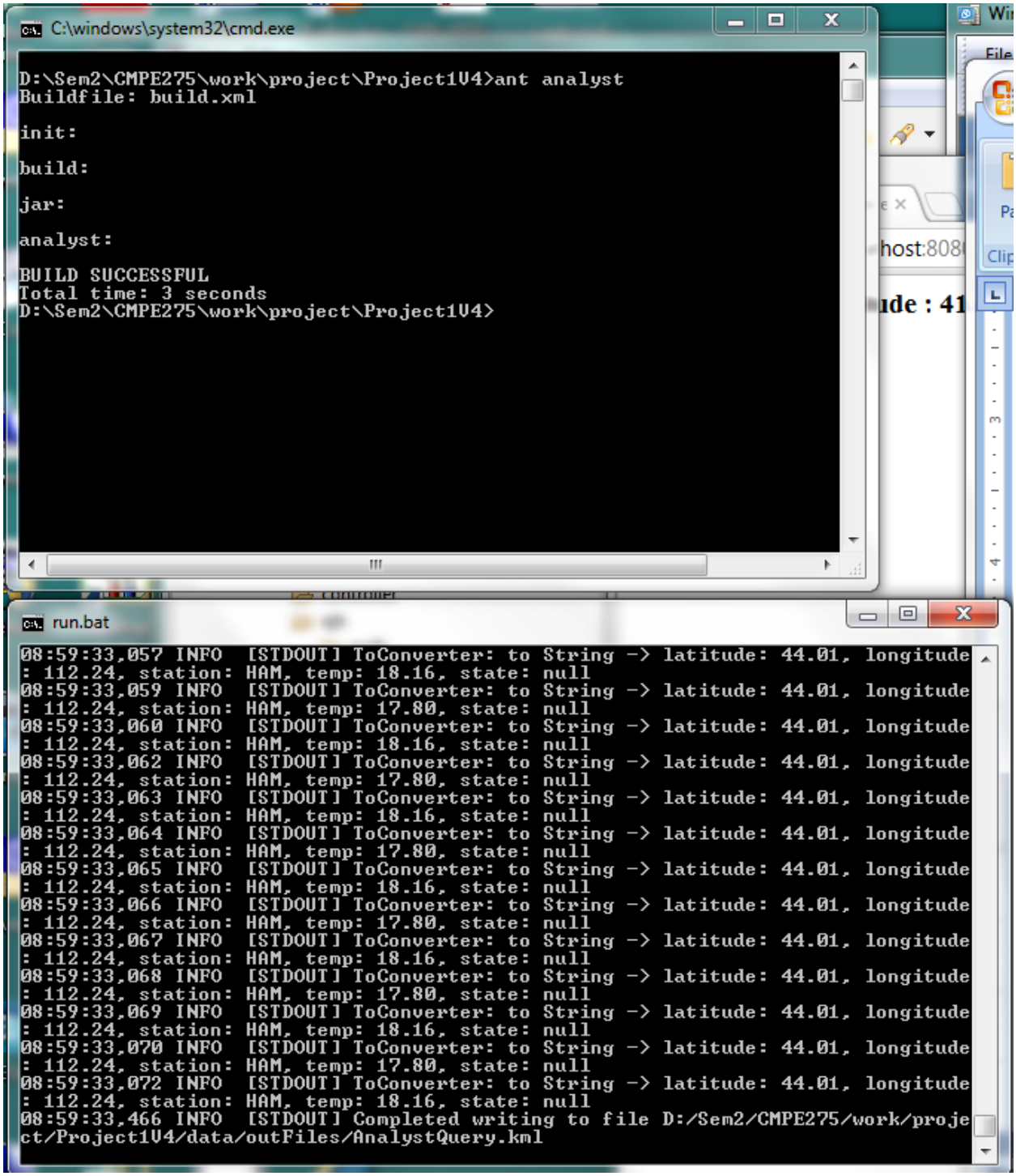
10. Upload

>ant mesonat

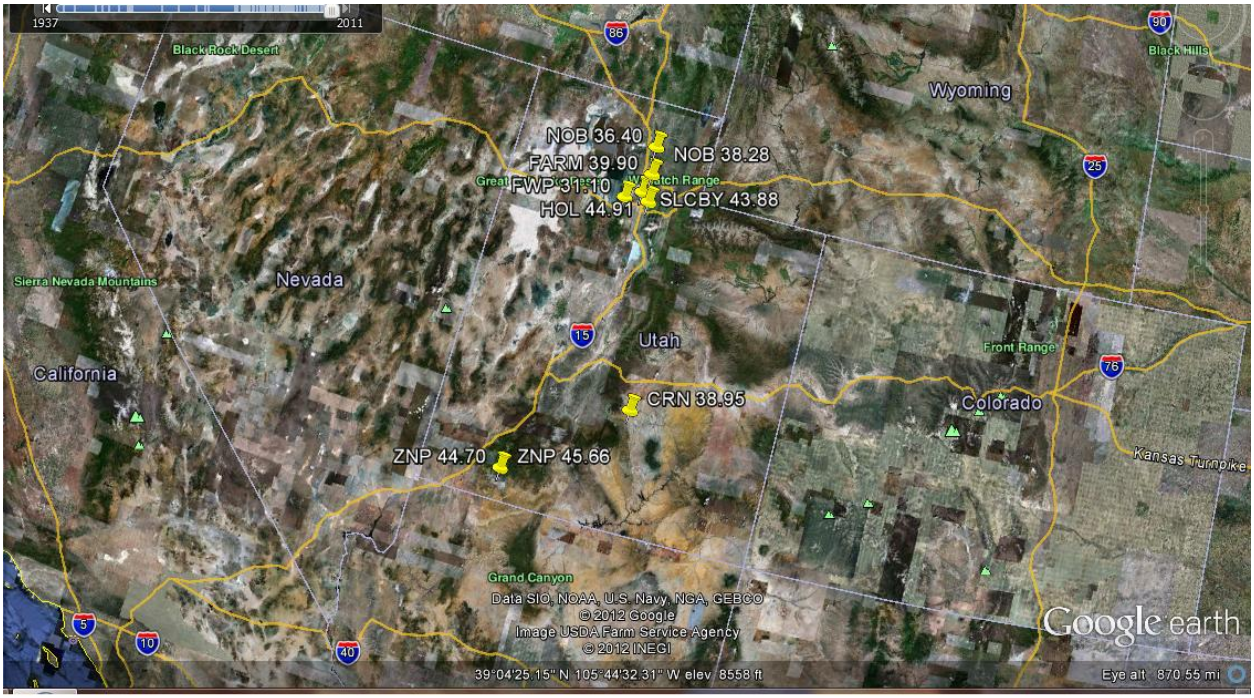


10. Analyst query – This will create a file with the requested stations at respective days to a file

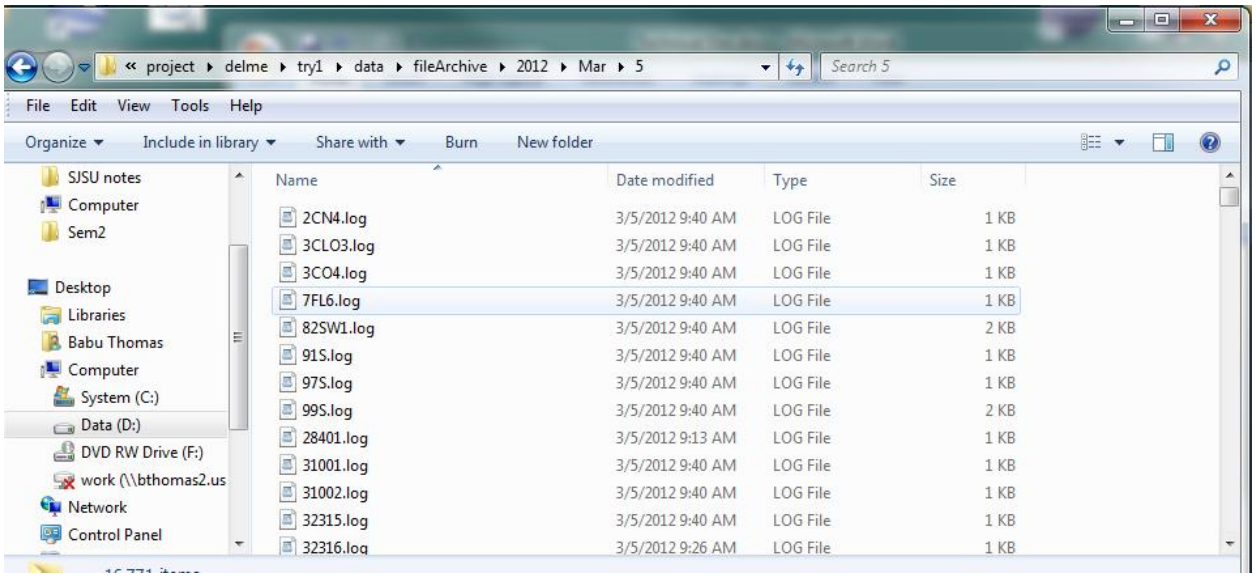
    >ant analyst



Sample KML Rendering is shown below



Archive Files



Tests

JUnit test cases are listed in com.Junit.test folder. Test cases are to validate the Inputs for the query is null and valid or not ,test that the query results are valid or not and generation of KML and JSON files etc. Use ant test command to run the test suite.

Performance tests

Read from http://mesowest.utah.edu/data/ - around 2 sec

Update conversion to internal data structures – 2 sec

Write to database – around 30 sec

So system can accommodate feeds from other stations

Issues faced

Sl	Issue	Resolution
1	For reading data file contents, initially used to read line by line and appending to a string, the file read took 12 mins	later modified to read via Bufered Reader and doing System.arraycopy and read took less than 10 sec
2	Initially tokenized by space but there were columns - " Limon International Airport "	Resolved by so had to processing the csv file and using .tbl format and parsing the exact columns width
3	Data – cleaning - had to fill in some columns were missing	Filled with spaces
4	Mesanat upload different foimat	Converted to database row format, and null columns filled with “unknown” entry
5	some records were repeated	Choose table column with unique primary key as ID
6	some columns had numbers with spaces	had to combine those columns to single column
7	Records in the tables were increasing, so deleted old rows, when new data came. Since delete and insert were originally in separate transactions, there were queries failing	Resolved by keeping delete and update in a single transaction
8	Update realtime feed were originally planned as a separate process. But found that for JPA integration to write to database, as JPA exception were thrown	Since the container support was required for JPA resolved it by keeping the sequencer program and collection of

		NWS feed as a Session Bean and a EJB Client to initiate the NWS feed
9	Plotting kml files on google maps to display on the browser itself	The kml file had to be stored on internet and given to google map as and when the kml file is refreshed.
10	Integration of JPA component with EJB components resulted in run time exception. Since Hibernate validator jar conflicted with another version of present in EJB.	First we added jars of JPA's and tested the system to find other jars required by EJB modules and then added only the required EJB jars.
11	MySQL columns are created with NOT NULL constraint by default if the variables are of type float which resulted with exception while storing NULL values.	Converted all float variable to String to make the columns have NULL values.

Possible Future Enhancements

Future consideration will include the creation of multiple global data centers and planning for cluster or multi-site deployments. We can also support some more file formats based on what other organizations tie up with WeatherSpot and agree to provide their weather data.

Future consideration will include the creation of multiple global data centers and planning for cluster or multi-site deployments. We can also support some more file formats based on what other organizations tie up with WeatherSpot and agree to provide their weather data.

Mem cache – There could be significant increase in the query performance if memcashing was to be introduced. The present query is

```
"select distinct d.STN,d.YYMMDD_HHMM,d.SLAT,d.SLON,d.TMPF,m.state,d.GUST FROM Data d,Metadata m where d.STN=m.primaryid and d.STN=:station"
```

Proposal - This can be converted to

```
function queryStation(int stnId) {
/* first try the cache */
    result = memcached_query("userrow:" + stnId);
if(!result)
    result = pointQuery(""select distinct d.STN,d.YYMMDD_HHMM,d.SLAT,d.SLON,d.TMPF,m.state,d.GUST FROM Data d,Metadata m where d.STN=m.primaryid and "WHERE stnId = ?", station);
    /* then store in cache until next get */
       memcached_update("station:" + stnId, result);
    }
    return result;
 }
```

```
}
```

So that initial try is from the cache and if not the database and memcache is updated and query retrieved

Proofing in coming data - The incoming data is processed and cleaned so that they can be inserted to the database in the tables with the predefined format. Issues such as spaces in individual columns, presence of duplicate rows etc were removed. The issues faced in the data cleaning area is summarized below.

1	some records were repeated	Choose table column with unique primary key as ID
2	some columns had numbers with spaces	had to combine those columns to single column
3	Records in the tables were increasing, so deleted old rows, when new data came. Since delete and insert were originally in separate transactions, there were queries failing	Resolved by keeping delete and update in a single transaction

Individual Contribution :

Babu Thomas	Architecting the overall system. Interface definition for sub-components, and implementation of EJB(MDB and Session), sequencing, NWS, Point Query and Upload integration, file read write interfaces, build scripts, performance tests. Servlet, JSP design, and interfacing with EJB. Driving project to completion. Doc review and update
Prasanna Raaj Kumar	Designed and Implemented JPA(Hibernate ORM), Database and File Storage components in the system. Implemented logic to handle and store mesowest station data and point data (from NWS). Implemented querying logic for point query(users) and Range query(analyst) . Integration and testing the entire system. Document update and review.

Anupama Patil	Setting up subversion. Research on JPA. Trying out given samples by professors and analyzing the requirements. Coding for KML and JSon outputs for both public user queries and Analyst queries. Writing JUnit test cases Writing technical document.

References

http://sathishgoogleapp.blogspot.com/2008/06/using-javamysql-to-create-kml_05.html

http://www.mkyong.com/java/how-do-convert-java-object-to-from-json-format-gson-api/

http://www.ankara-gtug.org/2011/06/04/how-to-create-google-code-project-and-synchronize-files-with-svn-repository-using-subclipse/

http://blog.msbbc.co.uk/2007/06/using-googles-free-svn-repository-with.html

http://www.youtube.com/watch?v=D_CebfrTRpc

http://www.youtube.com/watch?v=ZzT2fI32Z7w&feature=bf_next&list=ULnNUf7EEOkok&lf=mfu_in_order

http://java.sun.com/developer/technicalArticles/javaserverpages/rss_utilities/

http://www.javapractices.com/topic/TopicAction.do?Id=54

http://httpunit.sourceforge.net/doc/servletunit-intro.html

http://www.youtube.com/watch?v=nNUf7EEOkok