**Air pollution monitoring system**

**J2EE, JBoss, JMS, JPA, JDOM, EJB, Hypersonic DB, JUnit, Ant and Java**

[1. ABSTRACT 3](#_Toc203212192)

[2. INTRODUCTION 3](#_Toc203212193)

[3. OBJECTIVE 4](#_Toc203212194)

[4. PROPOSED SOLUTION 4](#_Toc203212195)

[4.1. Technologies and Approach 4](#_Toc203212196)

[4.2. Architectural Concept 9](#_Toc203212197)

[4.3. Design Patterns Used 11](#_Toc203212198)

[4.3.1. Message Façade 11](#_Toc203212199)

[4.3.2. Session Façade 11](#_Toc203212200)

[4.3.3. File transfer 13](#_Toc203212201)

[4.4. Screenshots 14](#_Toc203212207)

[4.5. Testing 17](#_Toc203212208)

[5. Assumptions & Future Enhancements 18](#_Toc203212211)

[6. Hardware & Software Used 18](#_Toc203212212)

[7. Steps in running the project 19](#_Toc203212213)

[8. References 19](#_Toc203212214)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | *layout_center* | **1. ABSTRACT** |   ACME is participating in a joint project with the California Department of Transportation (Cal DOT), and the California Dept of Public Health. The pilot project California Highway Emissions Monitoring System (HEMS) is collecting data to assist in the understanding on the health effects and risks to people who use and live near the state’s freeway and interstate highways. This data is collected by deploying sensors across three freeways 101,680 and 880.This will be done by employing a network of sensors for collecting the traffic congestion and uses a monitoring system to understand the pollution levels based on the values of Co2, So2, NO, CO and Particulate matters. The aim of our project is to design and develop the HEMS.  This document presents an overview of our design approach, assumptions, considerations, limitations and future enhancements*.*   |  |  | | --- | --- | | *layout_center* | **2. Introduction** |   Exhaust pollutants from trucks and cars pose a health risk to people with heart or lung disease, asthma or other respiratory problems. Generally, the children and the elderly are most susceptible to health risks. The purpose of the HEMS system is to understand emissions along transportation corridors and to better understand long-term potential health risks.  The system is divided into three components. HEMS-CCC provides visualization and controls. The HEMS-DATA system collects processes raw sensor data, and storage/archival system .The HEMS-PHS investigates the effect of highway emissions to health effects. PHS requests data to perform analysis outside of the HEMS system.  The pilot project has installed state-of-the-art air sampling (particulate) units (ASUs) and chemical (emission) sensor units (CSUs) at fixed locations along the test area’s freeways. The CSU collects and reports CO,CO2,S02, and NO values, and the ASU collect particulate samples. Both the ASU and CSU data is electronically sent to the collection system for processing and storage. The SMS data is a comma-delimited record, which includes the sensor, material (sample) and a value. |

|  |  |
| --- | --- |
| layout_center | **3. Objective** |

The project has the following core functionalities:

1. Design the HEMS-CCC system in a component based model.
2. Develop the HEMS system by using JMS, EJBs and JPA technologies.
3. Use the mock data feeds for demonstrating the processing and congestion indicators.
4. Provide the snap shots of the traffic usage in the KML file to the HEMS-CCC system which uses Google Maps/Google Earth for visualization.

# layout_center 4. PROPOSED SOLUTION

## Technologies and Approach

This section presents the technologies, used for implementing the HEMS system.

**Java Messaging Service (JMS)**

JMS is used for sending asynchronous messages between two or more clients. We have used JMS in HEMS system to send and receive asynchronous messages in the following two different ways:

1. To send the mock data that has been retrieved to a Message Driven Bean (MDB) class.
2. To receive the data from the client and to send the acknowledgement and messages to a log file by using callback method.

**Enterprise Java Beans (EJB)**

EJB is server-side component architecture for java. We have used EJB 3.0 in this project since the HEMS system is a distributed application. We have used MDB which is one of the EJB technologies for writing the entire business logic of the HEMS system. This MDB class is used for writing the logic for calculating the traffic congestion, sending the messages to a log file.

**Java Persistence API (JPA)**

JPA is part of EJB 3.0 which is a framework that allows managing relational data in java. JPA is used in HEMS system in the following two different ways:

1. To retrieve the mock data from the database since we created mock data in database.
2. To persist the calculated traffic congestion and signal timing into the default database used in the JBoss and querying using Java Persistence Query Language (JPQL).

## Supporting Information for Technologies Used

**EJB 3.0**

We have used EJB 3.0 for HEMS System implementation because it is a distributed application which should be portable and secured. Since EJB 3.0 can be easily integrated with the legacy systems and provides all the above functionalities by enabling rapid and simplified development, EJB 3.0 is the major component of our project.

This specification defines the new simplified EJB API targeted at ease of development. It also includes the new Java Persistence API for the management of persistence and object/relational mapping with Java EE and Java SE.

In a typical J2EE application, Enterprise JavaBeans (EJBs) contain the application's business logic and live business data. Although it is possible to use standard Java objects to contain your business logic and business data, using EJBs addresses many of the issues you would find by using simple Java objects, such as scalability, lifecycle management, and state management. When there is a requirement of reusability, distributed, transactional, secured server-side component, EJB’s are used.

Applications can incorporate EJBs to house processing and mapping between different applications. Again, this is an encapsulation of the business logic that is needed when transferring data between applications.

There are three different types of EJB that are suited to different purposes:

* *Session EJB*—A Session EJB is useful for mapping business process flow (or equivalent application concepts). There are two sub-types of Session EJB — stateless and Stateful— that are discussed in more detail on Day 5. Session EJBs commonly represent "pure" functionality that is created as it is needed.
* *Entity EJB*—An Entity EJB maps a combination of data (or equivalent application concept) and associated functionality. Entity EJBs are usually based on an underlying data store and will be created based on that data within it.
* *Message-driven EJB*—A Message-driven EJB is very similar in concept to a Session EJB, but is only activated when an asynchronous message arrives.

**JPA**

JPA is the major enhancement and part of EJB 3.0. We have used JPA in our project as an object-relational mapping to the database. Since the HEMS System architecture is component based and independent of one another, the ORM should also be independent and should be used outside the container or with any third party persistence providers. These features increase the flexibility and will be very easy to add any enhancements in future. Since JPA provides all the above features, we have used JPA as an object-relational mapping to the database in our project.

The Java Persistence API is a Java Specification for persisting the Java Objects to relational databasegrey_loader

1. using popular ORM technology. JPA API provides enough tools to enable the java developers to create database driven applications quickly. The JPA API can be used to persist the business object (POJO) to the relational database. Retrieving the data from database in the form of business objects (Java Objects) is so simple with the help of JPA API.

Java developers who need to store and retrieve persistent data already have several options available to them: serialization, JDBC, JDO, proprietary object-relational mapping tools, object databases, and EJB 2 entity beans. JPA combines the best features from each of the persistence mechanisms listed above. Creating entities under JPA is as simple as creating serializable classes. JPA supports the large data sets, data consistency, concurrent use, and query capabilities of JDBC. Like object-relational software and object databases, JPA allows the use of advanced object-oriented concepts such as inheritance. JPA avoids vendor lock-in by relying on a strict specification like JDO and EJB 2.x entities. JPA focuses on relational databases. And like JDO, JPA is extremely easy to use. ORM frameworks that can be used with JPA are Hibernate, Top link, Open JPA etc.

**JMS**

JMS is used as a Message-Oriented Middleware for HEMS System. Since HEMS System architecture is component based and independent of one another as described above, the messaging system also should be very loosely coupled between the components. The messages that will be sent between the components of the HEMS System should be asynchronous, reliable and secured even for future enhancements. Since JMS provides all the above mentioned features, we have used JMS as MOM in our project.

JMS provides a reliable, flexible service for the asynchronous exchange of critical business data and events throughout an enterprise. JMS, the Java Message Service, is an API that allows Java applications to access a wide range of Message Queuing servers through a standardized interface.

The Java Message Service is a Java API. It is used to create, read, send, and receive messages. It is designed by Sun Systems and other partner companies. The JMS API provides a common set of associated semantics and interfaces that enable programs written in Java Language to communicate efficiently with other Messaging Implementations.

The JMS API enables a loosely coupled communication and also provides asynchronous delivery of messages and reliability in delivering of these messages.

Some of the features of the JMS API are:

* The web components, application clients, and the EJB components can send or receive JMS messages synchronously.
* Message-Driven Beans, a new kind of EJB enables asynchronous processing of messages. JMS can implement processing of messages by MDB concurrently.
* Messages sent though JMS can be used to participate in distributed transactions

A JMS Application is composed of the following parts:

* JMS Provider – it implements the JMS Interfaces and also provides controlling and administrative features.
* JMS Clients – They are the components written in Java Programming Language. These are the programs which consume and produce messages.
* Messages – these are the objects which communicate the information between the JMS Clients.
* Administered Objects – These are the initially configures objects of the JMS. These are created by the administrator. These are used by the clients.
* Native Clients – These are the clients which use a different communicating product.

**JUnit Test Tool**

JUnit is a simple open source Java testing framework. It is used to write and run repeatable automated tests. Eclipse supports creating test cases and running test suites, so it is easy to use for the Java applications. Using the JUnit test tool, functionality can be tested at very unit level. While the development is in progress before all pieces work together we want to make sure each unit is working properly. So, it is very helpful for debugging at unit level. It is because of the above reasons we have selected JUnit test tool and performed the testing.

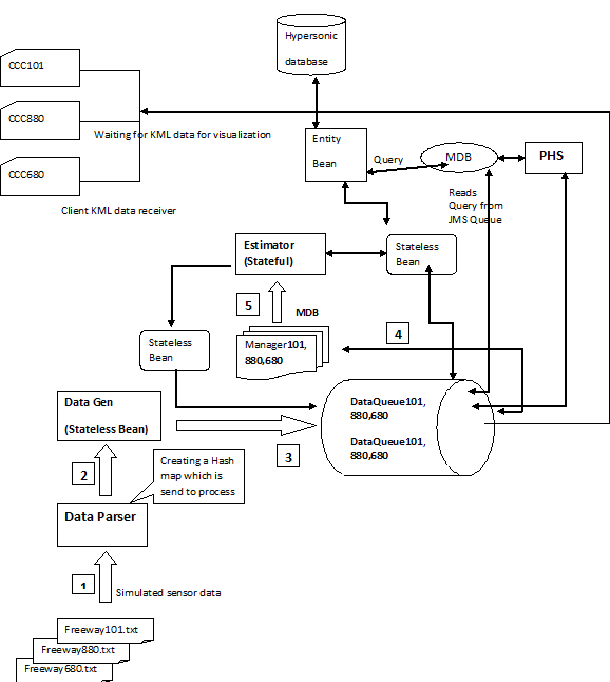
**JBoss Application Server**

JBoss is an application server which provides all the basic features of our project like EJB 3.0, JPA, and JMS. One of the goals of the HEMS System is to send and receive reliable messages between the components. Since JBoss is a provider of the above features, we have used it as an application server for our project.

**ANT**

Apache ANT is a Java based build tool. It can be used for automating software build processes. Ant uses XML to describe the build process and its dependencies. For our HEMS system, we have used ANT tool to build the application. It has been used to deploy and run our application on JBOSS Application Server

4.2 Architecture Design

In order to simulate the smoke data capture we created 3 sets of text file and it is used as a raw data feed into the system to measure and analyze the data. We created a text parser which will parse these text file and create a hashmap of array list in which all the data are stored for transmission across the wire. Then hashmap object is sent to the message bean which will read these hashmap object and will call the stateful session bean which will use stateless bean to evaluate the level of smoke in the freeway and read the kml file and update the file based on the analyzed data. We randomly selected a value for CO, CO2, NO, SO2 and particulate. If these value is less than defined value then we assume that normal smoke in the freeway and some time it is offline.

The phs will request data from the data center and which will query the persisted data and sent it to the phs client.

***Working of Each Component***

**Text file**: As seen in the above architecture there are three text files“freeway101.txt”,”freeway880.txt”,”freeway680.txt” each for three road ways 101,880 and 680.Based on the given format we have considered the values to be sent in the following order for sensor one as S1 Co CO2 So2 No2 Pa and then timestamp as given in the problem statement .The values of the material in text file are considered to be same as coming from a real sensor deployed on the freeway hence it’s a simulated data.

**Data Parser**: The data parser parses the data in the three text files based on an algorithm and creates a hash map of array list which is send to the processor(data generator) to process.

**Processor data generator**: The processor analyses, maps the data and sends it to the JMS queue.

**Manager (Message Bean):** There are three managers for each roadways 101,880,680.The manager is actually message bean that listens the object message coming from the JMS queue using "On message" and sends the data to the Estimator .

**Estimator (SSB**): The Estimator is a stateful session bean which uses JDom to analyze the data create KML files and map data in database using a stateless Bean.

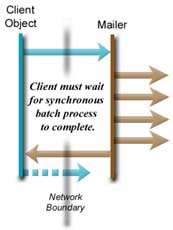
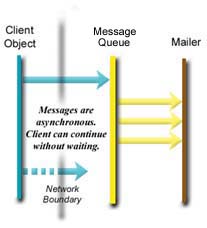
**PHS**: The PHS reads query from the JMS Queue sends it to a Message Driven Bean where the data stored in the default Jboss database is accessed and analyzed by Stateless bean from the entity log and the result is sent back to PHS in the specified format.

**CCC101, 880,680: There** are Command & Control center for each roadway as seen in the above diagram which waits for KML data for visualization.

|  |  |
| --- | --- |
| layout_center | **4.3 Design Patterns** |

**Message façade Pattern:**

A server-side object provides client objects with an interface to the methods of one or more EJBs. However, unlike the Session Façade, where messages are synchronous, the Message Façade uses asynchronous messaging. This design pattern is useful in situations where a client message triggers a process such as a batch update or a mass emailing, and the client doesn't want or need to wait for a return value before continuing its flow of execution.

HEMS DATA receives the mass SMS updates asynchronously using queues by the HEMS sensors which are mimicked by the data-generator module. This pattern is chosen because:

* It allows bandwidth management
* It makes the components asynchronous, so that they are independent for processing
* It hides differences in interfaces if any

Also, message queuing pattern is used by HEMS data to interface with the client application for the same reasons of *decoupling* and *scalability* as stated above.

**Session façade**

The session facade pattern defines a higher-level business component that contains and centralizes complex interactions between lower-level business components. A Session Facade is implemented as a session enterprise bean. It provides clients with a single interface for the functionality of an application or application subset. It also decouples lower-level business components from one another, making designs more flexible and comprehensible.

Fine-grained access through remote interfaces is inadvisable because it increases network traffic and latency. The "before" diagram in Figure 1 below shows a sequence diagram of a client accessing fine-grained business objects through a remote interface. The multiple fine-grained calls create a great deal of network traffic, and performance suffers because of the high latency of the remote calls.

The session beans in HEMS Data provide remote and local business interfaces which are coarse-grained hiding the underlying complex interaction between the components. This provides flexibility to change underlying implementation.

|  |  |
| --- | --- |
| **Without Session Façade** | **With Session Façade** |
| Lots of network traffic without the Session Facade. | Session Facade reduces network traffic. |

**Data Access Object Pattern (DAO)**

The Java Persistence API used for persistent storage of data is an example of the DAO pattern implemented by HEMS DATA. This pattern is chosen to implement because:

* It separates a data resource's client interface from its data access mechanisms
* It adapts a specific data resource's access API to a generic client interface

This means that using the same API for mapping objects remains consistent irrespective of the underlying data source or the persistence provider. For instance, currently the system uses JPA with Hibernate as the persistence provider and HyperSonic as the DataSource. JPA provides flexibility and scope for scalability since it allows data access mechanisms to change independently of the code that uses the data.

**Transfer Object Pattern**

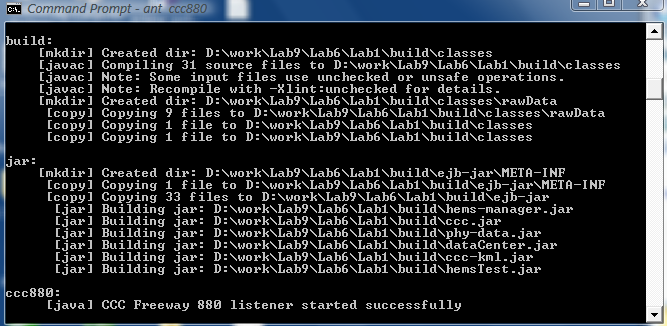
The TO pattern is used for efficient use of bandwidth since the client can invoke coarse grained methods and retrieve objects for calling its attributes locally. The Log object sent to the client is an example of this pattern used by HEMS data.

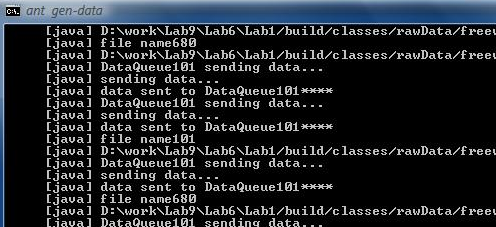
**MVC Pattern**

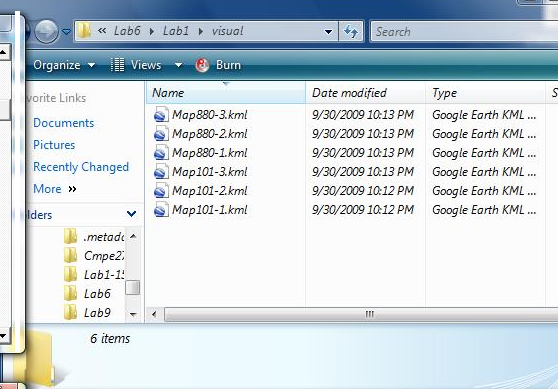
Finally, HEMS Data is built upon the foundation pattern which is the Model-View-Controller pattern to facilitate isolation of functionality and reduce dependency.

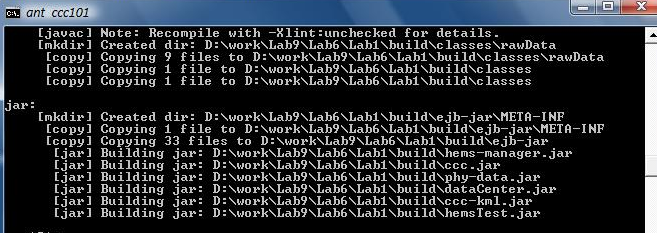
Stateless beans are injected in MDB’s which interface with the client and sensor systems so that they can provide scalability with external PHS and sensor systems.

|  |  |
| --- | --- |
| layout_center | **4.4 Screenshots** |

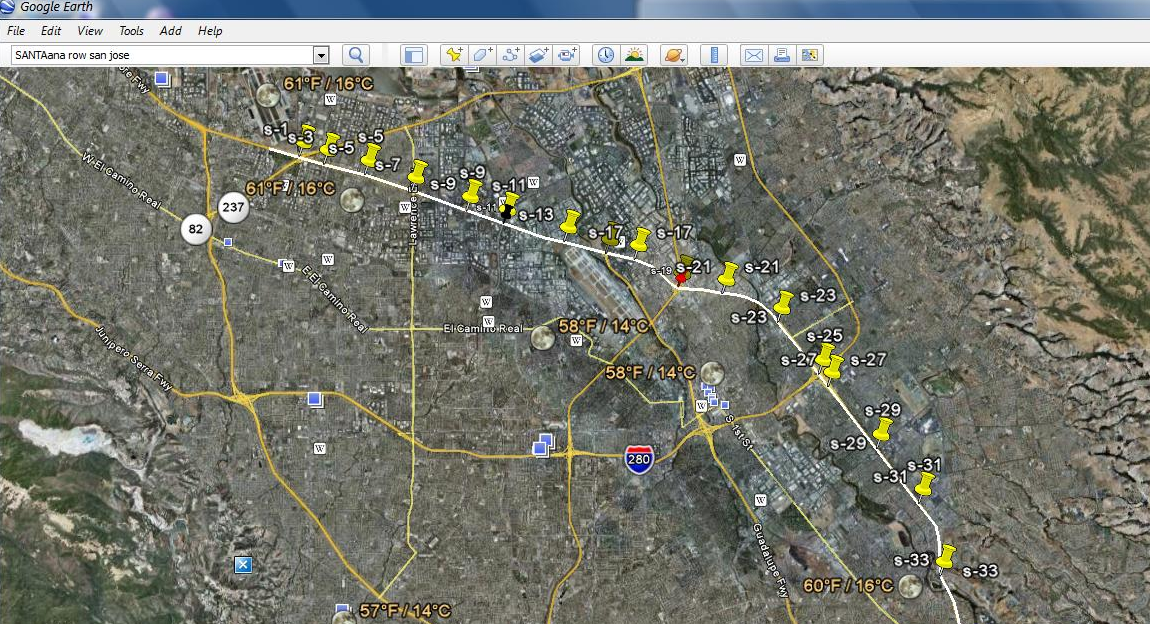








|  |  |
| --- | --- |
| layout_center | **4.5 Testing** |



`

s-1:10,11,10,11,10,2009,02,12,06,20,00

s-11:00,00,00,00,00,2009,02,12,06,20,00

s-15:15,11,20,11,10,2009,02,12,06,20,00

s-19:66,77,77,76,65,2009,02,12,06,20,00

The above diagram shows sensor-1 in different states. These data are sent to the evaluator and which will mark the styleurl in the map according to the calculated value. And it is evident that the actual value matches with the assumed data set.

|  |  |
| --- | --- |
| layout_center | **5. Assumptions & Future Enhancement** |

1. In the project we are sending an object message assuming that all the 18 sensors send the message at the same time to the CCC.
2. We have taken assumptions of the values of Co, Co2, So2, No and Pa as 15, 10,25,10,10 to be normal. Any deviation of the +5 will be considered to be increase in pollution while -5 will be considered to be decrease in the pollution level.
3. If the values are 0 then the sensor is not operating, either faulty or switched off.
4. The sensor is sending messages every 30 sec at present and can be calibrated to any hours, minutes or seconds.
5. There are three text files includes the simulated data from the sensors. The mock data in each text file represents data coming from sensors of the three roadways 101,880 and 680.

**Future Enhancement**

1. In this project the sensors are limited to 18 but can be increased and likewise for the data also which proves that it is scalable.
2. The project can be implemented in the real time environment where real sensors send data, the message will be processed sent to CCC and also any offline sensor can also be detected.

|  |  |
| --- | --- |
| layout_center | **6. Hardware & Software Used** |

**Hardware:**

Processor: Intel Core 2 duo 2.0 GHz

Ram: 4 GB

HD: 180 GB

**Software**:

JBoss Application Server 4.2.3

Eclipse GA

Java, JEE / EJB 3.0/JPA/JMS

JUnit

Apache ANT

|  |  |
| --- | --- |
| layout_center | **7. Steps for running the project** |

1. Run jboss 4.2.2
2. Ant ccc101, ant ccc880, ant ccc680 – to run the ccc client
3. Ant gen-data – to generate data
4. Ant phy – to request data from data center
5. Ant test – to junit testing on the system
6. For step 3 and 4 it is not necessary to perform step 2 and 3

|  |  |
| --- | --- |
| layout_center | **8. References** |

[www.java.sun](http://www.java.sun) .com

<http://www.roseindia.net/ejb/>

[www.oracle.com/.../**design pattern**/](file:///C:\Users\Nehru\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\SAENG01N\www.oracle.com\...\design%20pattern\)

<http://www.oracle.com/technology/tech/java/ejb30.html>