Chemical Early Warning System in Java

Developing emergency response systems

by Kathy Lee Simunich, Gordon Lurie, Michelle Kehrer, and Tom Taxon

The need for a chemical/biological early warning system within mass transportation sites is real and was being researched before the September 11 terrorist attacks. Since then, this research has become an operational necessity.

Led by Argonne National Laboratory, along with teams from Sandia National Laboratories (SNL) and Lawrence Livermore National Laboratory (LLNL), researchers created an early warning system for subways within a major metropolitan area. The entire system entailed the processing of sensor data, live meteorological data, video feeds, and real-time track/train data; execution of below- and aboveground dispersion models; and the timely display of results to subway and emergency personnel. The system is based extensively on Java technologies, and this article examines the merits of each J2EE package used.

The Chem-Bio Emergency Management Information System (CB-EMIS) is an enterprise-level distributed system. The decision to write it in Java was made early because of Java's maturity, scalability, and maintainability, as well as the availability of the industry-supported J2EE technologies. This article discusses the software aspects of the CB-EMIS and presents the architecture in detail.

Industry Standard Technologies

The CB-EMIS runs on four dual-processor Dell PCs running the Red Hat 8 Linux Operating System (the servers). Because the system must always be up and able to recover from communication failures, etc., three J2EE technologies were used to gain this reliability: Java Messaging Service (JMS), Remote Method Invocation Object Activation Daemon (RMID), and persistence through Java Database Connectivity (JDBC). A relational database was used to persist data for forensic playback and simulation modes (MySQL for development and training, and Oracle 9*i* for operational use).

The Sun ONE (previously iPlanet) Message Broker was used for the JMS system, and Novell's Lightweight Directory Access Protocol (LDAP) implementation (eDirectory) was set up for use as the directory service and for the authentication of users. It's important that the server processes across all CPUs remain synchronized in time; a Network Time Protocol (NTP) server was used to keep the system times in sync. Since the system relied on JMS messages, it was important to have the times match for message time-out purposes. Also, the Time Server process reads the system time to keep a consistent time stamp for all data and messages passing through the other server processes.

CB-EMIS System-Level Services

This section describes the design of the foundational services layer with which all server processes extend, implement, or interface. Figure 1 shows the services layer that supports the CB-EMIS.

Directory Service

The LDAP server was used for authentication of user accounts as well as a directory service for object binding. The schema was also extended to allow the storage of attributes unique to the system on a per-user basis, allowing control over what each user could see or do in the client graphical user interface (GUI). The Java Naming and Directory Interface (JNDI) package was used throughout the CB-EMIS to bind and look up object references stored in the LDAP server. Figure 2 shows the various objects bound into the directory. Not only were the JMS Topics and Topic Connection Factory references bound in the server, but the various JDBC data sources were as well. In this way, any of the CB-EMIS processes can look up the references to all the connection factories, topics, and JDBC data sources without having provider-specific references within the code. Saving these objects within an LDAP server gains persistence and security, and avoids the need for a shared file system. The team also decided to bind the CB-EMIS server process Remote Method Invocation (RMI) stub references within the LDAP server. Because there's not a JNDI class for binding stub references within a directory server, it was necessary to write our own javax.naming.spi.ObjectFactory and javax.naming.Referenceable classes to serialize/deserialize the java.rmi.server.RemoteStubs.

The Archive

A relational database management system (RDBMS) is used as the persistent store (referred to as the Archive) for the static and dynamic data during either a simulation or normal operation. The purpose of persistence is for error recovery and playback mode. Static system description data is imported into the database for use during normal operation or simulations. During playback mode of either a training simulation or a normal operational session, the data archived in the database is retrieved and executed for the user. Archived live data can be played back for forensic study of an incident.

Each server process looks up the data source via JNDI and connects to the appropriate database through the JDBC. All events and data are saved in the database for playback or for error recovery. Rather than having SQL statements scattered throughout the code, a Persistent Object class was created that other classes extend. By defining the table name and the column names, each class inherits the ability to insert, retrieve, update, and delete itself from the database. JDBC Prepared Statements were used to speed up transactions to the database.

A Persistence Broker class does the JNDI look-up, maintains the reference to the data source, and automatically reconnects to the database if the connection is lost. Most of the data records are time-stamped, which is usually part of the primary key.

Activatable Server Processes

Each of the server processes is restartable via RMI activation. Activation allows programs to register information about remote objects so they can be started up when they are first accessed or when the RMI Daemon (rmid) starts up. This provides the ability to start the entire server, which consists of processes across multiple machines by registering one process to be started when rmid starts up, as well as the ability to recover from a Java Virtual Machine (JVM) failure by restarting a failed process. If the JVM of an activatable object does not exist, the RMI Daemon creates a new JVM for that object.

All of the CB-EMIS server processes are restartable by extending the java.rmi.activation.Activatable class. The process stubs are registered with the rmid process (which is already running) and also binds its stub reference into the LDAP server. The RMI Daemon provides a Java Virtual Machine from which other JVM instances may be spawned. This process actually starts up the other server processes "on-demand" as they are requested. Therefore, if a process goes down it will be reactivated upon the next reference to it via rmid (barring an actual hardware failure).

A stub reference is looked up by a CB-EMIS process via JNDI and passes that reference to the RMI Daemon, which in turn activates the process.

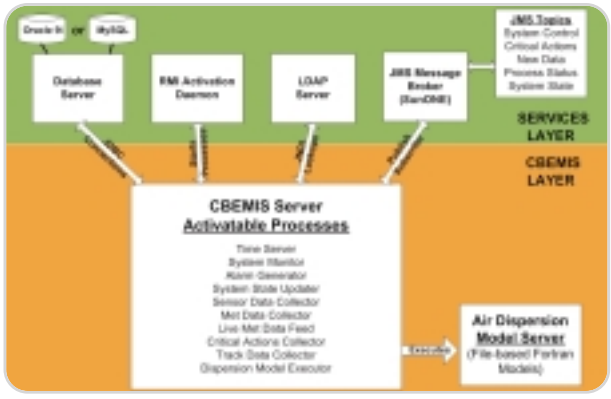

Figure 1 CB-EMIS Software Elements

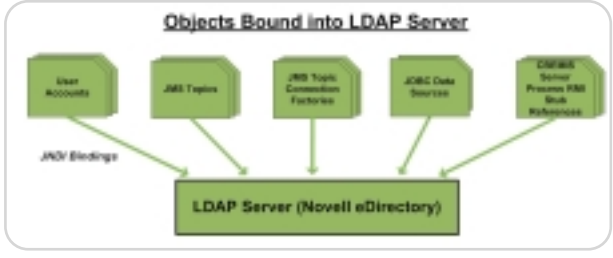

Figure 2 Objects bound into LDAP server

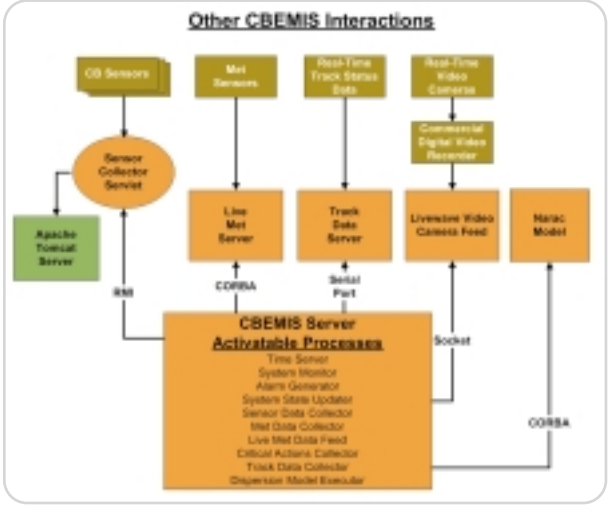

Figure 3 CB-EMIS interactions with other hardware/software elements

JMS Messaging

All the server processes communicate via a publish/subscribe methodology with JMS Topics. This communication mechanism was chosen because JMS has several advantages over Remote Method Invocation for this type of application, most notably:

- It is asynchronous in nature, so if one component goes down, the process that's sending the data is not tied up waiting for the other process to restart.
- Messages can be guaranteed to be delivered even if the process that needs it is not running. It will get all waiting messages when it starts back up.
- It's fairly simple to have another process register to receive data from the same source as others without having to contact the source of the data.

The server processes look up references to the JMS Topics that they're interested in and "sign up" as a publisher and/or a subscriber to various topics. Like the database connections, the JMS Utility class tries to reconnect if a connection is lost or dropped. Most classes in the CB-EMIS that are persistent usually also implement the Publishable interface. Instead of making RMI calls to pass data or having the processes make JDBC calls to pass data via a database, the data is wrapped up in a JMS message and published to a topic.

CB-EMIS Server Processes

Figure 1 shows the CB-EMIS Server Processes interacting with the services layer. The CB-EMIS "server" has been broken down into the various components (or processes) described in this article based on the various tasks the server needs to do.

The processes fall into the following categories: data collection, monitoring, and utilities. The data collection processes are the Meteorological Data Collector, Live Met Data Feed, Track Data Collector, Sensor Data Collector, and the Critical Actions Collector. The monitoring processes are the Alarm Generator, System State Updater, and System Monitor. The utility processes are the Time Server and the Model Executor.

Thus, the server consists of 10 separate processes running within their own JVM across the four dual-processor PCs. They all are activatable via the RMI Daemon. When they start up, they make a JNDI look-up to get references to the JDBC data source and make a connection to the database as well as to the JMS Broker to sign up for publishing and/or subscribing to the available topics. All of the processes employ multithreading to provide the fastest possible processing of information through the system, while monitoring all the data topics and archiving to the database.

This section describes each server process, its basic purpose, and the relevant Java technology it employs to do its work. As stated earlier, the processes communicate via JMS Topics. These include:

- ***System Control:*** The System Monitor publishes to this topic whenever the administrator changes the mode of the server (normal, playback, or training). All processes subscribe to it and change their processing according to the current mode of operation.
- ***Process Status:*** The System Monitor also publishes to this topic approximately every two seconds so that the administrator GUI can show the current state of the server processes.

Kathy Lee Simunich has been an object-oriented software engineer for over 15 years, progressing from C++ to Smalltalk and now writing almost exclusively in Java since 1998. Her concentration has been in cross-platform model integration and simulation systems. She has a BS in meteorology and an MS in computer science.

Gordon Lurie has been an object- oriented architect and software engineer for over 15 years. He was the software architect lead developer for the CB-EMIS system. His work has focused on object-oriented geographic information systems. He has a BS in computer engineering.

Michelle Kehrer has been a software engineer for over 10 years developing standalone client, Web, and enterprise applications. She has a BS in computer science.

Tom Taxon has been an object-oriented software engineer for over 13 years. He worked with Smalltalk and C++ and has been working in Java since 1996. His background is in GIS and model integration and simulation systems. He has a BS in computer science and engineering.

partners@anl.gov

- ***New Data:*** All server processes archive data and events to the database and then publish the data on the New Data topic for the other processes to ingest (if interested in the particular data item).
- ***Critical Actions:*** The CB-EMIS client process publishes to this topic whenever the emergency response users declare a critical action (e.g., trains have stopped, alarms triggered). The purpose of this topic is to provide a non-RMI method of having the client processes communicate critical information back to the server processes.
- ***System State:*** The System State Updater publishes eight different data types to this topic. Each data type runs in its own thread, publishing frequently enough to update GUI elements within the client process. By publishing the state of the system onto this topic, any number of client processes can log in and display the current state by subscribing to the topic. As a consequence, clients can also recover from communications interruptions without missing data.

The Time Server

The Time Server is responsible for holding and maintaining the system time. It's the central point for all the server processes to obtain the synchronized system time, since it is the process that is synchronized with the NTP server. The other server processes access the time from the Time Server process via an RMI call. Having a single process keep track of the system time allows all processes to use the same methodology for getting current time regardless of the current mode (normal, training, or playback).

The System Monitor

The System Monitor is responsible for monitoring the state of the software system. It restarts the components if there is a problem and notifies the user if it cannot restart a process. The status of the system processes (components) is displayed via a GUI for the system administrator.

The System Monitor is responsible for verifying that all required processes are running and maintaining the current run state (mode, simulation run status, simulation speed, etc.). It pings a process via RMI every two seconds. When a process is pinged, the activation system (RMI Daemon) starts up the process if it is not already running. The individual processes must also ping the system monitor back within five seconds. This allows the administrator to be notified if a process starts up and then crashes immediately. In addition, if a process detects or recovers from a nonfatal error, it can notify the System Monitor of this state change via an RMI call. If the status of a process changes, the current status is published on the Process Status Topic.

The current run state is maintained via RMI calls from the System Monitor GUI. When changes are made to the state, they are published to the System Control Topic so that all other interested processes can handle the change.

System State Updater

The System State Updater is responsible for maintaining the system state (train location, track status, meteorological conditions, sensors, plumes, alarms, etc.) and publishing it periodically to the System State Topic. There is currently one updater for high-bandwidth connections to the server, as there is a full, detailed system state sent every second. It listens to the New Data topic for when any process saves data to the database and then publishes the new data on the appropriate System State Topic for consumption by client processes that are currently connected. The design allows for other levels of detail of the System

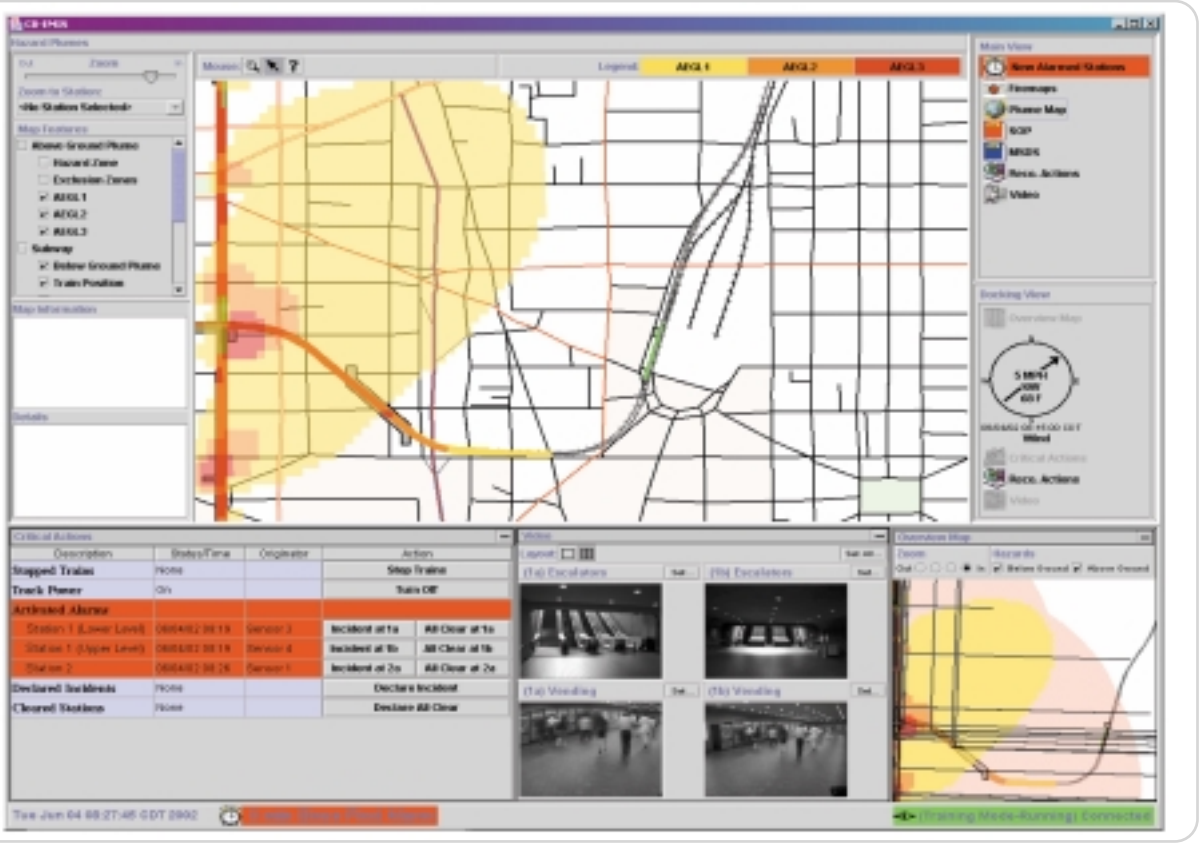

Figure 4 CB-EMIS Client GUI main window showing plume dispersion map with contamination, critical actions table, and various video feeds

State to be published. One such use of this would be a hand-held computer used by an emergency responder; it might get only system state data relative to the responder's location. Thus, a low-bandwidth updater can be created that publishes updates less frequently for clients connecting over a dialup or wireless connection.

Alarm Generator

The Alarm Generator is responsible for monitoring the New Data Topic for sensor events, determining if an alarm condition has been reached, and storing the data in the Archive/DB. It also publishes any alarms to the New Data Topic.

Model Executor

This process is responsible for responding to sensor alarms in the New Data Topic. It generates input files based on data received from the New Data Topic and starts to execute the appropriate below- and aboveground dispersion model. When the model(s) are finished, this process publishes the contamination results to the New Data Topic and archives the data in the database.

Critical Actions Collector

The main purpose of this process is to provide a non-RMI method of having the client process(es) communicate with the server. This is essentially a pass-through for Critical Action Data messages. It tracks who created a critical action and when it was generated.

Sensor Data Collector

This process is responsible for monitoring all the chemical/biological agent sensor readings. It stores the readings in the Archive and publishes the data on the New Data Topic. This process registers with the SNL Sensor Server via RMI.

Live Met Data Feed

This process polls the LLNL Met Server, which monitors real-time weather observations, and stores the data in a local database for access by the Meteorological Data Collector.

Meteorological Data Collector

The Met Data Collector is responsible for retrieving meteorological data from a server (or from a file if in training mode), storing the data in the Archive/DB, and publishing to the New Data Topic. This data is needed for the dispersion models to correctly predict plume concentrations.

Track Data Collector

The Track Data Collector is responsible for receiving real-time track status data and train location data, archiving it in the database, and publishing the data on the New Data Topic. It subscribes to the Critical Actions Topic for notification if the trains stop running.

CB-EMIS Interactions with Other Hardware/Software

Figure 3 shows the CB-EMIS interactions with various servers that have the physical connections to the sensors as well as other parties' monitoring software with which the CB-EMIS must communicate. Four types of sensor hardware are needed for the system:

1. Chemical/biological agent detectors
2. Meteorological sensors
3. Real-time train location and track status sensors
4. Real-time video camera feeds

Each of these systems has associated servers that were installed by parties other than Argonne. The video cameras were part of a legacy system installed for security purposes that CB-EMIS tapped into for an integrated display within the client GUI. The train location and track status data feed were also part of a legacy system used by other monitoring software within the subway's control room. This data, along with

current meteorological conditions and sensor readings, contains the critical information needed for accurate predictions of agent plume dispersion and concentrations.

Argonne has written a belowground dispersion model specifically for the subway where the CB-EMIS has been installed. The system also employed an existing EPA aboveground model (INPUFF) for the prediction of plumes if the agent escapes from underground. These models are written in Fortran and therefore use flat files for input and output. The CB-EMIS Model Executor reads/writes the output/input and makes a system call to actually start the models running on a separate CPU.

The CB-EMIS also uses the LLNL real-time meteorological observation server. Data is updated every 15 minutes, and the Live Met Data Feed process polls the server for current observations. The Model Executor can also access and run LLNL's high-resolution aboveground dispersion model (Narac) at a preset time after an alarm to obtain more accurate results. The default INPUFF model is useful for quick assessments, but more detailed predictions can be calculated with the Narac model (although it takes longer to run and get the results).

Sandia was responsible for installing the sensor detection software interface to log the actual readings of the chemical/ biological agent detectors. They implemented their sensor collection software as a Java servlet and, therefore, the servlet must run within a servlet container. The Apache Tomcat Server is used as the servlet container and Web server and runs on one of the nodes in the Linux cluster. The servlet continuously monitors the sensors.

CB-EMIS Client

Figure 4 shows one view of the Swing-based GUI for the emergency response coordinators. Each panel on the display can be either minimized and shown in icon form or maximized to show more detail. Panels include:

- GIS-like overhead map view with zoom and pan capability
- Overview map of subway system and train status
- Plume concentration overlay
- Sensor concentration-level graphs
- Critical actions and alarms
- Video camera views

Any number of remote client processes can be communicating with the CB-EMIS server located in the control center of the subway system. This communication link is done through Secure Socket Layers (SSL) and uses the Java Secure Socket Extension (JSSE) package to make the connection through the Java client process to the proxy server. As mentioned in the JMS Topic section, JMS messaging is used to pass data between the client process and the CB-EMIS server over the SSL channel. Critical actions are posted to the server via the Critical Action Topic, and each client subscribes to the System State Topic for updates. These updates are viewable in near real time on a high-bandwidth connection.

Conclusion

Emergency response systems, such as the Chem/Bio Emergency Management Information System, should be deployed throughout many critical infrastructures such as subway systems, airports, or other mass-transportation hubs. This article presented the distributed processes, the need for asynchronous communication through messaging, and secure communication and user access mechanisms provided by the CB-EMIS employing various Java technologies.

Reliability was achieved by using the RMI Activation Daemon to reactivate any processes that may have failed, and persistence of JDBC Data Sources and JMS Topic Factories within an LDAP server allowed restarted processes to easily look up the references via JNDI and reconnect to the database and message broker automatically.

Security was maintained through user authentication within the LDAP server on both the CB-EMIS server side as well as access to the client processes. RMI security policy files were used to restrict access rights for any RMI calls throughout the server processes. Further security was achieved by using an SSL connection via JSSE from client processes to a proxy server and then to the server processes behind a firewall.

This project has proven that Java technologies not only addressed the various needs of the system, but helped all the components of the system integrate seamlessly and efficiently. With all the monitoring processes, the continuous real-time updates (not only of the system status, but of the train and track data and sensor readings), the database archiving of all data, and message publishing and subscribing, the performance of the multithreaded processes running in multiple Java Virtual Machines was never a problem.

Q&A

Joe Ottinger, ***JDJ***'s editor-in-chief, had the opportunity to have a discussion about the Chem-Bio Emergency Management Information System with the team at Argonne National Laboratory.

JDJ: *What other technologies were considered? What were the measurements for validity for the technologies being considered?*

ANL: We wanted to use the established Java technologies. The first principle of the entire system was reliability.

The early prototype system was a Smalltalk/Java program using RMI and some custom socket-based communication between Smalltalk and Java. The Smalltalk version of GeoViewer was existing code that is now converted to Java for the final product.

Once we converted to production development, we needed to manage the synchronization of the communications between all the server processes and clients, and each process had to be restartable. We chose RMID activation for its restarting ability and JMS for its asynchronous communications and guaranteed delivery of the messages.

These technologies greatly reduced our development time.

JDJ: *I notice you're duplicating a lot of the functionality of JMX – how has the JCP worked to affect your development process and architecture (why wasn't it used, in other words)?*

ANL: We felt JMX was not a mature enough technology when we started on the final system. We started the final development stage in December 2001 and it had to be delivered/ installed by January 2003 and operational by June 2003.

JDJ: *What licenses did you use for your custom code, specifically for the RMI stubs in your LDAP server?*

ANL: We used LDAP for authentication and the JNDI lookup of the data sources as well as the RMI stub references. The use of eDirectory was dictated by the sponsor.

JDJ: *What kind of persistence layers did you consider, if any? Are they in-house, commercial, or open source?*

ANL: We had already written an in-house persistence layer for other projects, so we adapted it for use in our final prototype. The database was used more as a logger than needing the full features of an RDBMS. We used generic SQL to switch between using MySQL for development and Oracle for deployment.

JDJ: You mention you're using iPlanet for some services – why, then, did you choose Tomcat for a servlet container, when iPlanet also has a servlet container available for it?

ANL: We bought just the Sun ONE Java Message Queue since that's all we needed it for. We chose it after testing a couple of other vendors' products. The servlet used to monitor the sensors was a prototype developed by another party that has since been replaced by a more robust system, which is *not* based on servlet technology. It's much more tightly integrated with the rest of the system.

JDJ: In the "System State Updater" section, what is a "plume"? I have some idea because of context, but clarification would be nice.

ANL: The "plume" is a contour plot of the predicted concentration levels of the released agent that could pose health risks.

The plumes are updated every minute and are plotted as an overlay over the map on the client machines.

JDJ: What client platforms are supported?

ANL: Windows 2000 and XP have been extensively tested for the client software. However, plans are in the works for supporting Red Hat Linux (which is what we run on the server side). All the code runs on the major OSes, including Solaris, Red Hat Linux, and Mac OS X. Basically, any platform that supports J2SE can support our system. There is a small amount of JNI code included in our system, so the biggest hurdle is just compiling that code for a new platform. Plus, tweaking for optimal bandwidth usage for the video streaming needs to be done on any new platform.

References

- www.anl.gov
- http://java.sun.com
- www.dell.com
- www.redhat.com
- www.mysql.com
- www.oracle.com
- wwws.sun.com/software/products/message_queue/ home_ message_queue. html
- www.novell.com/products/edirectory/
- www.eecis.udel.edu/~ntp/
- http://java.sun.com/products/jndi/
- http://java.sun.com/products/jdk/rmi/
- http://java.sun.com/products/jdbc/
- http://java.sun.com/j2se/1.4/docs/guide/rmi/activation.html
- Coke, L. R., Sanchez, J.G., and Policastro, A.J. "A Model for Dispersion in the Subway Environment." 10th International Symposium on Aerodynamics and Ventilation of Vehicle Tunnels, Principles, Analysis, and Design. Boston, Massachusetts, 1-3 November, 2000
- Petersen, W.B., and Lavdas, L.G. "INPUFF 2.0 A Multiple Source Gaussian Puff Dispersion Algorithm User's Guide." EPA/600/8-86/024, U.S. Environmental Protection Agency, August 1986.
- http://narac.llnl.gov
- http://java.sun.com/products/servlet/
- http://jakarta.apache.org
- Lurie, G.R., Sydelko, P.J., and Taxon, T.N. "An Object-Oriented Geographic Information System Toolkit for Web-Based and Dynamic Decision Analysis Applications." *Journal of Geographic Information and Decision Analysis*. 2002, Vol. 6, No. 2, p.108–116
- Korp, P.A., Lurie, G.R., and Christiansen, J.H. "A Smalltalk Based Extension to Traditional Geographic Information Systems." Proceedings of the ParcPlace/Digitalk International Users Conference, San Jose, CA, ANL/DIS (July 31–Aug. 12, 1995).