A J2EE Syslog Aggregation and Reporting System

Patrick Carroll, Principal, Mobile Java Guy, LLC

Introduction

The purpose of this paper is to show how the Java 2 Enterprise Edition (J2EE) software stack, as implemented in the JBoss Application Server, can be used to develop an application that supports security policy enforcement and management via real-time views, investigative views, and reporting. Furthermore, the paper demonstrates the usefulness of Open Source Software (OSS), and also shows some best practices for enterprise software projects.

Motivation

With the advent of regulations like Sarbanes-Oxley ("Sarbox") and HIPAA (http://www.hipaa.org/) , IT organizations are increasingly under the gun to provide an auditing infrastructure. What was once considered best practice is now becoming law.

As part of the preparation for a Sarbox IT audit, an IT control framework must be identified. There are two federally endorsed publications that any internal or external auditor should be aware of if they are familiar with controls-based audit methodologies.

- Committee of Sponsoring Organizations of the Treadway Commission (COSO), available from http://www.coso.org), which addresses financial systems controls.
- Control Objectives for Information and related Technology (COBIT), a subset of COSO. This document is available at http://www.isaca.org/Template.cfm?Section=COBIT6&Template=/TaggedPage/TaggedPageDisplay.cfm&TPLID=55&ContentID=7981 and details controls from the information technology perspective.

Key to the IT control framework is the process of gathering of information about the IT infrastructure.

Although the regulations do not mandate automated systems-based controls, such controls may ease the compliance process. Auditors look not only for process consistency, but also for the consistent use of controls over those processes. For this reason, auditors may be more critical of manual or paper-based processes in large or distributed organizations. In many cases, using software solutions is the best way to implement consistent audit policy enforcement and management.

The world's better-known names in software – companies like Oracle and SAP – are providing policy enforcement and management solutions specific to their platforms. At the same time, existing internet security companies are retrofitting such capabilities into their existing products. However, the bottom line for now is that the requirements are new, the solutions are either incomplete or immature, and an organization could spend a lot of money without recognizing much benefit.

Fortunately, at this point in history the OSS community has made available tools that compete in terms of quality and capability with anything the commercial world has to offer. The fact that the software is available for free is a low barrier to entry. That there are no non-disclosure agreements, or licenses with financial penalties, means that barriers to exit are also low. Couple these advantages with standards compliance, and you have a collection of off-the-shelf software that allows relatively naïve users to "roll their own" so they can try out technologies to see what best fits them before making large financial commitments.

Recognizing this, many organizations might be better off adopting an enterprise-ready open source product, in order to try before buying and get a better sense of their individual needs. This paper is written to support that process.

Why J2EE?

The Java 2 Platform, Enterprise Edition (J2EE) is a set of coordinated specifications and practices that together enable solutions for developing, deploying, and managing multi-tier server-centric applications. Building on the Java 2 Platform, Standard Edition (J2SE), the J2EE platform adds the capabilities necessary to provide a complete, stable, secure, and fast Java platform to the enterprise level. It provides value by significantly reducing the cost and complexity of developing and deploying multi-tier solutions, resulting in services that can be rapidly deployed and easily enhanced.

The J2EE platform provides the following:

- Faster solutions delivery time to market. The J2EE platform uses "containers" to simplify development. J2EE containers provide for the separation of business logic from resource and lifecycle management, which means that developers can focus on writing business logic - their value add - rather than writing enterprise infrastructure. For example, the Enterprise JavaBeans (EJB) container (implemented by J2EE technology vendors) handles distributed communication, threading, scaling, transaction management, etc. Similarly, Java Servlets simplify web development by providing infrastructure for component, communication, and session management in a web container that is integrated with a web server.

- Freedom of choice. J2EE technology is a set of standards that many vendors can implement. The vendors are free to compete on implementations but not on standards or APIs. Sun supplies a comprehensive J2EE Compatibility Test Suite (CTS) to J2EE licensees. The J2EE CTS helps ensure compatibility among the application vendors which helps ensure portability for the applications and components written for the J2EE platform. The J2EE platform brings Write Once, Run Anywhere (WORA) to the server.

- Simplified connectivity. J2EE technology makes it easier to connect the applications and systems you already have and bring those capabilities to the web, to cell phones, and to devices. J2EE offers Java Message Service for integrating diverse applications in a loosely coupled, asynchronous way. The J2EE platform also offers CORBA support for tightly linking systems through remote method calls. In addition, the J2EE platform has J2EE Connectors for linking to enterprise information systems such as ERP systems, packaged financial applications, and CRM applications.

- By offering one platform with faster solution delivery time to market, freedom of choice, and simplified connectivity, the J2EE platform helps IT by reducing TCO and simultaneously avoiding single-source for their enterprise software needs.

J2EE is an open, standards-based, development and deployment platform for building n-tier, Web-based and server-centric enterprise-strength applications. It's proven, it's pervasive, it's portable, and it's powerful. It's the right way to deliver Web services, no matter what your market, or where your development effort is headed for the long term.

This paper assumes that the reader is familiar with J2EE application servers.

Why JBoss?

Now in its 4th generation, the JBoss Application Server has become a recognized leader in the Java application server market and is to date the only major application server on the market to deliver a production-ready J2EE 1.4 certified product. With well over 5 million downloads to date, JBoss Application Server has become the most popular product among Java developers and independent software vendors alike.

JBoss provides the following advantages:

- JBoss is free to download, deploy, and embed.

- The JBoss Application Server is the only major production-ready application server to achieve J2EE 1.4 certification.
- Performance. Based on their own internal benchmark tests, companies like La Quinta Corporation have found that JBoss offers improved performance and superior server utilization to other leading J2EE application servers.

Why Syslog?

The BSD Syslog Protocol is described in RFC 3164 (http://www.faqs.org/rfcs/rfc3164.html). As the RFC has it,

> This document describes the observed behavior of the syslog protocol.
> This protocol has been used for the transmission of event
> notification messages across networks for many years. While this
> protocol was originally developed on the University of California
> Berkeley Software Distribution (BSD) TCP/IP system implementations,
> its value to operations and management has led it to be ported to
> many other operating systems as well as being embedded into many
> other networked devices.

If there can be said to be a universal IT information reporting system, then Syslog is it. Syslog comes with most Unix systems. Microsoft Windows has a similar "EventLog" system, and adapters exist for converting EventLog messages to Syslog Messages. For example: https://engineering.purdue.edu/ECN/Resources/Documents/UNIX/evtsys/ or this: http://www.mikrotik.com/download.html. Hence, any information gathering system would be wise to center itself on Syslog.

State of the Practice Architecture for Syslog Aggregation and Reporting

The internet security industry has standardized on a process flow whereby security events from many monitored devices are aggregated into a central database before being presented to a user.

The following diagram depicts the general setup.

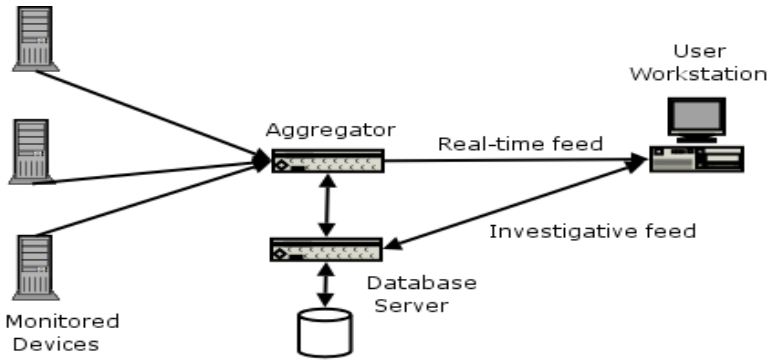

Figure 1: State of the Practice Architecture

For example, the GuardedNet neuSECURE product is depicted as follows:

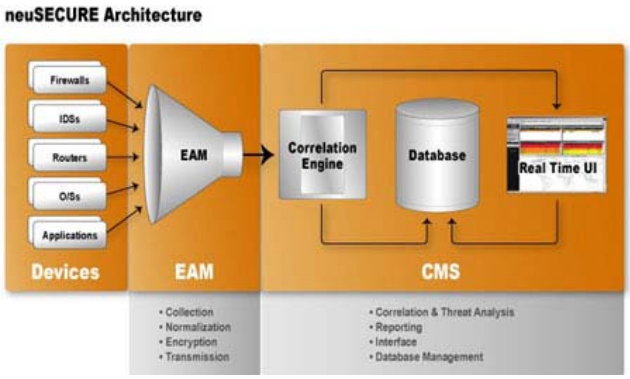

Figure 2: GuardedNet neuSECURE Architecture

Here we see all the components listed in the generic architecture: aggregation of data from multiple sources, storage of information in a database, real-time and non-real time interfaces into the system.

The J2EE stack provides technologies to support all aspects of such an architecture.

An Architecture for a J2EE Syslog Aggregation and Reporting System

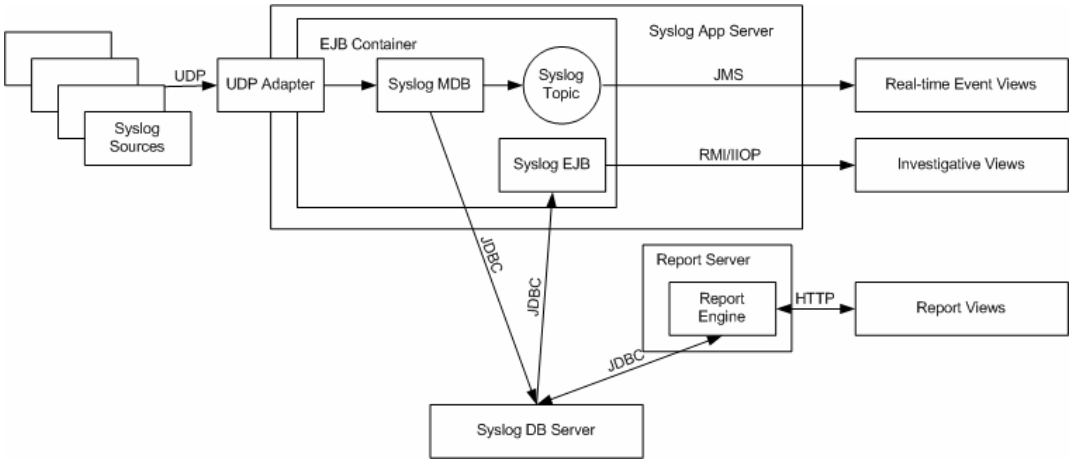

Figure 3: J2EE Syslog Aggregation and Reporting System

Figure 3 shows the general outline for a J2EE Syslog Aggregation and Reporting System.

In a low-cost deployment, the Syslog Application Server, Syslog DB Server, Real-time Event Viewer, Investigation, and Report Viewer could be hosted on the same computer. Architecture highlights include the UDP Adapter, the Real-time and Investigative Views, and the Reporting subsystem.

UDP Adapter

With the latest release of the J2EE Connector Architecture (JCA) specification (JCA 1.5), it is now possible for J2EE applications to manage ports. This allows the J2EE application to listen for Syslog UDP messages on port 514 and deliver them to the J2EE stack for further processing. This paper will provide an in-depth description of this adapter, as it's a new and very useful addition to the J2EE specification.

Real-Time and Investigative Event Views

Once Syslog messages are in the system, they can be presented to users, either as part of a real-time of Syslog events, or as a set of events retrieved as part of a user investigation. J2EE provides different mechanisms to support these different modes of operation.

For real-time viewing, J2EE provides a messaging service that allows for point-to-point messaging via queues and publish/subscribe messaging via topics. Using a topic here allows us to plug in other modules which might take advantage of the real-time Syslog event feed. For example, as well as feeding events to a real-time user interface, we might add an expert system engine into the mix, putting us on the road to rule-based business process execution.

For investigative reports, J2EE provides – via Enterprise Java Beans – a means of providing remote access to the database, via RMI/IIOP. In this situation, the user specifies queries to narrow down the set of Syslog events that are of interest, and only those events are retrieved.

Report Subsystem

The system exists in order to generate reports for auditors. In this system, the reporting engine is the open source OpenReports project.

The rest of this paper is a more-detailed exposition on these subjects.

Packaging Details

This system is composed of a set of packaged components, as depicted below.

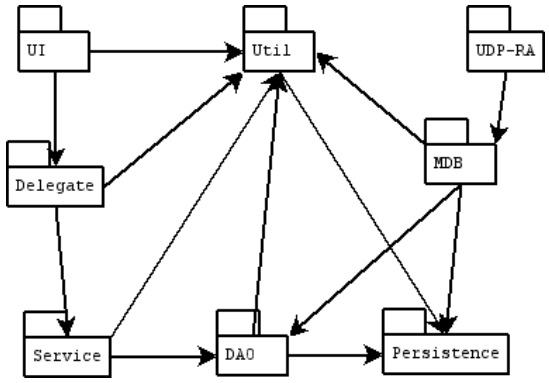

Figure 4: System Packaging Structure

So, building the system involves starting at the lowest level package, and then building up to the highest package.

Package Descriptions

The project is organized into a set of sub-projects.

- UI – The user interfaces for the system.
- UDP-RA – A J2EE Connector Architecture package that integrates UDP into the J2EE stack.
- MDB – A J2EE Message-driven Enterprise Java Bean (EJB) that converts UDP packets to Syslog messages, and passes the messages to the rest of the system.
- Delegate – A wrapper around the EJB service layer to hide the details of the service layer remoting
- Service – An stateless session EJB-based set of services
- DAO – A Data Access Object layer, to hide the details of lower-level database access
- Persistence – A Hibernate-based DBMS access layer.
- Util – A set of utility classes.

The Ant `build.xml` in the syslog-aggregator project has three targets: "clean", "build", and "dist". These will invoke the "clean" and "build" targets on all the sub-projects, and finally combine the results into an enterprise archive file.

Select packages include some interesting technology demonstrations:

- The Util package includes JUnit-based unit tests that are executed inside a JUnit GUI.
- The Service package demonstrates the use of Ejbdoclet to generate EJB code
- The Persistence layer uses Hibernate tools and mapping files to generate a DBMS schema corresponding Java objects
-

UDP Adapter – J2EE Connector Architecture

JCA was introduced (as JCA 1.0) in J2EE 1.3 to provide outbound communications from application servers to Enterprise Information Systems (EISs).

Under the Connector Architecture, EIS vendors no longer have to customize their product for interaction with each compliant J2EE application server. Similarly, application server vendors needn't make modifications whenever they need connectivity to yet another EIS. Instead, application server vendors implement the Connector Architecture framework only once, and EIS vendors develop one standard resource adapter based on this architecture. Under these circumstances, a compliant EIS can plug into any application server that supports the Connector Architecture, and an application server can connect to any EIS that provides a standard resource adapter. This seamless integration between compliant application servers and enterprise information systems is depicted below.

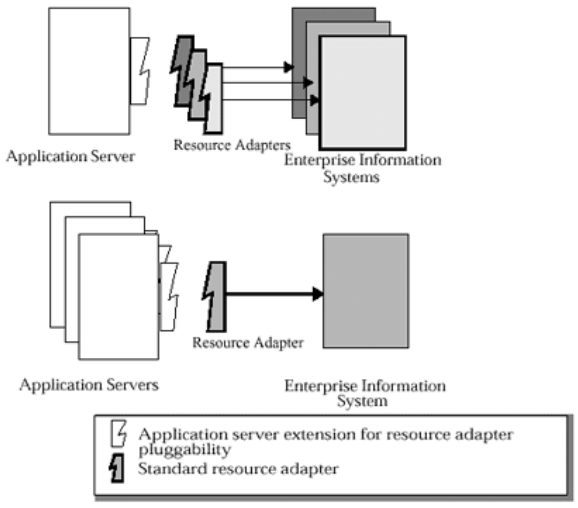

Figure 5: J2EE Connector Architecture

JCA 1.0 did not provide a mechanism for inbound communications from EISs to application servers, such as asynchronous message delivery or event notification. In version 1.5 of the J2EE Connector Architecture, there are more contracts that a resource adapter must support, as new functionality and features made their way into the specification. A resource adapter can support these four new contracts by implementing the required interfaces defined in the specification for each contract.

- Lifecycle Management Contract: This lets the application server manage the lifecycle of the resource adapter.
- Work Management Contract: This allows the resource adapter to do work by submitting it to an application server for execution. Since the application server does the work for the resource adapter, the resource adapter needn't worry about thread management. Instead, the application server manages this aspect efficiently and can use thread pooling if necessary.
- Transaction Inflow Contract: This allows a resource adapter to propagate an imported transaction to an application server, as well as flow-in transaction completion and crash recovery initiated by an EIS.
- Message Inflow Contract: This allows the resource adapter to synchronously or asynchronously deliver messages to endpoints in the application server, regardless of message style, semantics, and infrastructure.

This system builds on two of the new contracts: work management and message inflow. Code samples show how parts of the interfaces can be used. The application includes a special UDP resource adapter as well as a Message Driven Bean (MDB), and functions as a real-time event handler accessible to UDP clients. It accepts UDP messages to the application server and adapts the input into invocations of a backend real-time event MDB. This application demonstrates two key concepts of JCA 1.5:

1. The ability to do work by submitting it to an application server for execution.

2. The ability to invoke Message Driven Beans (MDBs) without JMS.

This application is based on JBoss 4.0, which is J2EE 1.4 compliant and fully supports JCA 1.5.

High-Level Description

Figure 6 shows that inbound message flow has three main components: the sources of Syslog events, the UDP Resource Adapter and the Syslog event MDB.

These components work tightly together, with their interactions managed in part by the Application Server. The Syslog event MDB represents the business logic, i.e. it has the information on how to process incoming events. It publishes a message listener interface, which will be invoked by those resource adapter instances interested in this particular MDB. The UDP Resource Adapter adapts an external message from a UDP client into an invocation on the MDB. In the diagram below, a UDP client accesses the Syslog event MDB through the UDP Resource Adapter.

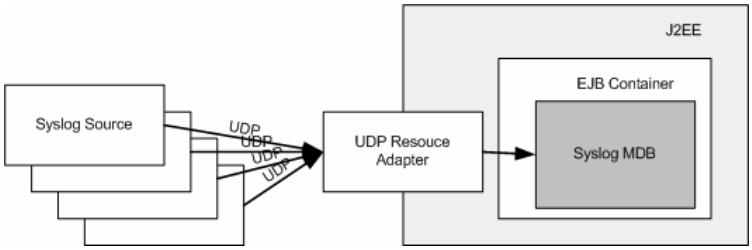

Figure 6: Resource Adapter Components

The message filtering and routing mechanism available through JCA 1.5 inbound connectors is not apparent in this application. In most cases where the inbound connectors are used, messages are filtered and routed in the resource adapter through the Activation Specs of the MDBs involved. In this example, every message that comes into the resource adapter is sent to the same MDB.

The UDP Resource Adapter

The UDP Resource Adapter allows access to the Syslog event MDB. It runs as a thread that listens for UDP datagrams, and spawns threads to handle client input and invoke the `SyslogMessageProcessor` described below. The class `com.mjg.jca.udp.UDPResourceAdapter` implements `javax.resource.spi.ResourceAdapter`.

Work Management

One of the main additions to JCA 1.5 that allows the development of inbound connectors (or listeners) is the introduction of the work API's in the `javax.resource.spi.work` package. The work management APIs allow the resource adapter to schedule work to be preformed by container-managed threads. This substitutes the explicit spawning of a worker threads per client request, which is imperative in building any server. After the datagram has been received, it is then time to invoke the `SyslogMessageProcessor` MDB.

Message Delivery

The `SyslogMessageProcessor` MDB is exposed to the Syslog event listener Resource Adapter as a message endpoint. When the J2EE container is started and the `SyslogMessageProcessor` MDB is deployed, the container looks for deployed resource adapters that can support the `UDPListener`

messaging type (the supported messaging type for a particular resource adapter can be found in its deployment descriptor). For each resource adapter which supports the `UDPListener` messaging type, the method

```
endpointActivation(javax.resource.spi.endpoint.MessageEndpointFactory, javax.resource.spi.ActivationSpec)
```

is called.

This method allows the `SyslogMessageProcessor` to register itself with the various resource adapters deployed that support the `UDPListener` messaging type. It also associates an ActivationSpec with the `SyslogMessageProcessor` MDB, which lets the UDPResourceAdapter know when to invoke the `SyslogMessageProcessor` MDB. The ActivationSpec is a Java Bean that allows an MDB to declaratively specify what kind of events or messages it is looking for. The name of the Java Bean class that implements the ActivationSpec and the Java Bean property names are specified in the deployment descriptor of the Resource Adapter. The property values are specified in the `SyslogMessageProcessor` MDB's deployment descriptor.

When datagram is received, and the Syslog Event Resource Adapter is prepared to invoke the `SyslogMessageProcessor` MDB, the resource adapter uses the list of endpoint activation specs to decide which endpoint factory to use to create the message endpoint, creates a message endpoint, and invokes the `UDPListener`'s `processMessage(Date timestamp, InetSocketAddress socketAddress, ByteBuffer byteBuffer)` method.

Below is a snippet of code that shows how the `SyslogMessageProcessor` MDB is invoked from a Work unit scheduled by the `UDPResourceAdapter`. In this example, the UDP message has been received, the origin of the UDP message has been noted, and a timestamp has been created.

```
Iterator activationsKeySetIterator =
    udpResourceAdapter.getActivations().keySet().iterator();

while (activationsKeySetIterator.hasNext()) {

  UDPActivationSpec udpActivationSpec =
      (UDPActivationSpec) activationsKeySetIterator.next();


  if (udpActivationSpec.accept(udpMessage)) {

    MessageEndpointFactory factory =
        (MessageEndpointFactory) udpResourceAdapter.getActivations().get(udpActivationSpec);

    try {
      UDPListener endpoint =
          (UDPListener) factory.createEndpoint(null);
      // invoke the method
      endpoint.processMessage(timestamp, inetSocketAddress, udpMessage);
    }
    catch (Throwable t) {
      logger.error("Endpoint error: ", t);
    }
  }
}
```

In complex routing schemes there might be many Activation Specs, so the application checks whether the incoming event is acceptable to a given Activation before building an endpoint and invoking the message processing logic.

Sequence Diagram

Here's how messages get delivered to the message driven bean. This sequence diagram illustrates the highlights of the message delivery processing for the UDP resource adapter.

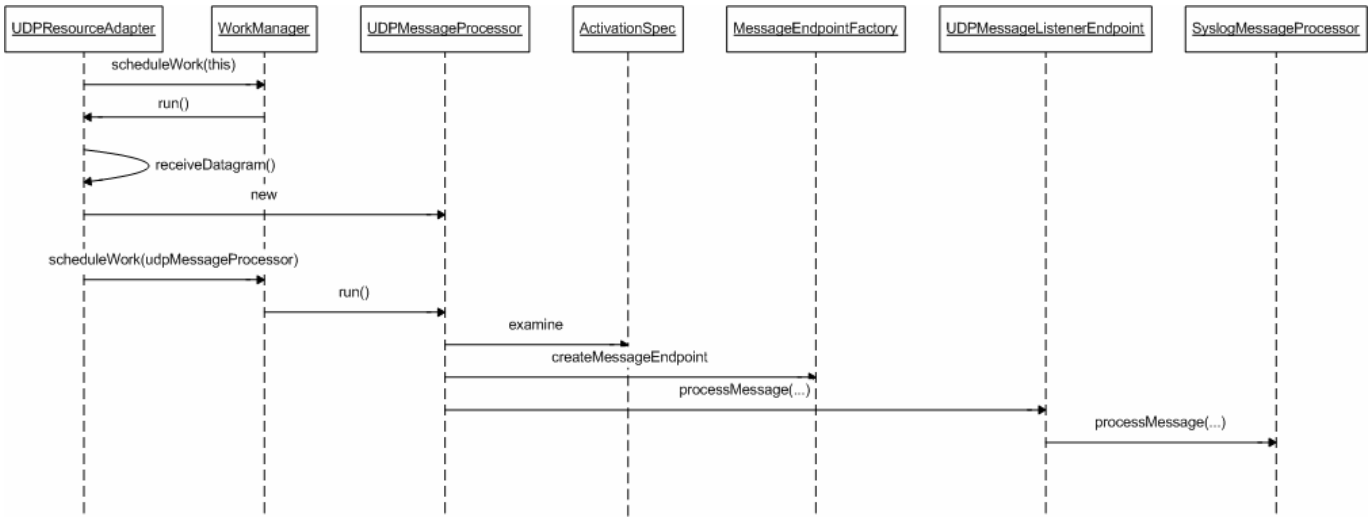

Figure 7: JCA Layer Sequence Diagram

Upon activation the UDPResourceAdapter schedules itself as a piece of work inside the WorkManager.

When run, it listens for incoming datagrams on a specific port (514 for Syslog messages). When a datagram is received, the resource adapter defers processing a datagram to another worker thread. It creates a UDPMessageProcessor and schedules it to process the message.

The UDPMessageProcessor needs to pass the message to appropriate message listener. First it identifies target listeners based on the activation specification values. Then it acquires access to the `MessageEndpoint` (which is the application server's proxy of our message listener) and invoke the `processMessage(…)` method to pass it to the message driven bean

Deployment Descriptor

A look at the Resource Adapter below also provides some useful insights on some of the internal wiring that is exploited by the Application Server at deployment time. Some of the tags of importance in the deployment descriptor are:

1. <resourceadapter-class> tag -- the tag lets the container know the name of the resource adapter class, in this case `com.mjg.jca.udp.UDPResourceAdapter`.
2. <config-property> tag- the tag lets the container know the Java Bean configuration properties to be set on the `UDPResourceAdapter` (in this case it sets the port of the listener to 514).

3. <messagelistener-type> tag -- the tag lets the container know what type of message listeners the resource adapter can invoke. Specifically, the fact that the message listener type is specified to be `com.mjg.jca.udp.UDPListener` means that when the Syslog Event Listener MDB is deployed, the endpointActivation method of the `UDPResourceAdapter` will be called by the container.
4. <activationspec> tag -- the tag lets the container know the form of the ActivationSpec associated with the MDB and the resource adapter.

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://java.sun.com/xml/ns/j2ee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
           http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd"
           version="1.5">

   <description>UDP resource adapter</description>
   <display-name>Inflow of UDP Resource Adapter</display-name>
   <vendor-name>Mobile Java Guy, LLC</vendor-name>
   <eis-type>UDP Adapter</eis-type>
   <resourceadapter-version>1.0</resourceadapter-version>

   <license>
      <description>
      COPYRIGHT AND PERMISSION NOTICE
      Copyright (c) 2004 Mobile Java Guy, LLC
      </description>
      <license-required>true</license-required>
   </license>

   <resourceadapter>
      <resourceadapter-class>com.mjg.jca.udp.UDPResourceAdapter</resourceadapter-class>

    <config-property>
      <config-property-name>Port</config-property-name>
      <config-property-type>java.lang.Integer</config-property-type>
      <config-property-value>514</config-property-value>
    </config-property>
      <inbound-resourceadapter>
         <messageadapter>
            <messagelistener>
               <messagelistener-type>com.mjg.jca.udp.UDPListener</messagelistener-type>
               <activationspec>
                  <activationspec-class>com.mjg.jca.udp.UDPActivationSpec</activationspec-class>
               </activationspec>
            </messagelistener>
         </messageadapter>
      </inbound-resourceadapter>
   </resourceadapter>
</connector>
```

JBoss-specific Configuration

In addition to the standard J2EE connector specification, the system also requires a vendor-specific description of the resources used in the connector. For JBoss, this takes the form of a jboss.xml file that is deployed along with the connector.

JBoss-specific MDB configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss>
   <enterprise-beans>
      <message-driven>
         <ejb-name>SyslogMDB</ejb-name>
         <destination-jndi-name>mjg/SyslogMDB</destination-jndi-name>
         <resource-adapter-name>syslog-aggregator.ear#udp-ra.rar</resource-adapter-name>
      </message-driven>
   </enterprise-beans>
</jboss>
```

One thing to note is that deployment of this MDB depends explicitly on deployment of the resource adapter. Since the entire system is delivered as an Enterprise Archive, the names of both files are used to specify the resource adapter name.

The Syslog Event MDB

As stated above, the Syslog Event MDB contains the business logic of this JCA real-time event application, which is to decorate an incoming event, and forward it to be published to interested subscribers. What is of interest here is not the particulars of the real-time event decoration and forwarding, but rather the contract between the resource adapter and the application server that allows the resource adapter to invoke the Real-time Event MDB.

Custom Messaging Type

Prior to JCA 1.5 Message Driven Beans could only be accessed through JMS, and therefore had to implement the `javax.jms.MessageListener` interface, which allowed it to listen for and perform business logic in response to JMS messages. The introduction of inbound resource adapters in JCA 1.5 allows EIS vendors to build their own listeners in a J2EE container and therefore being able to invoke MDB's with any message type published by the MDB. Thus, the MDBs in EJB 2.1 can implement any custom interface, and whichever EIS would like to access the MDB would have to write a resource adapter to invoke the MDB's listener interface, which is also referred to as the Messaging Type of the MDB. In our JCA application, the real-time event MDB implements the `UDPListener` interface, which has the following signature:

```
public interface UDPListener {
  void processMessage(Date timestamp, InetSocketAddress socketAddress, ByteBuffer byteBuffer) throws Exception;
}
```

The Message Driven Bean handling the real-time events is implemented in the `com.mjg.mdb.syslog.SyslogMessageProcessor` class and has the signature:

```
package com.mjg.mdb.syslog;
…
import javax.ejb.MessageDrivenBean;
…
import com.mjg.jca.udp.UDPListener;

public class SyslogMessageProcessor
    implements MessageDrivenBean, UDPListener {
…
}
```

Deployment Descriptor

The Syslog event MDB is packaged in a jar file as a standard J2EE EJB module. It informs the container of its messaging type and properties through the EJB module's deployment descriptor (ejb-jar.xml) file. Below is the deployment descriptor for the Syslog event MDB. The points of interest are:

1. <message-driven> tag -- this tag lets the container know that the EJB module is a Message Driven Bean
2. <ejb-class> tag -- this tag lets the container know the class that implements the interface javax.ejb.MessageDrivenBean
3. <messaging-type> tag -- this tag lets the container know what messaging type the MDB supports. This is the way by which the container identifies which resource adapters are suited to send messages to this MDB.
4. <activation-config> tag -- this tag configures the activation spec JavaBean associated with this MDB. As described below, the activation spec JavaBean allows the resource adapter to determine whether a particular MDB should be invoked.

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd"
         version="2.1">

   <enterprise-beans>
      <message-driven>
         <ejb-name>SyslogMDB</ejb-name>
         <ejb-class>com.mjg.mdb.syslog.SyslogMessageProcessor</ejb-class>
         <messaging-type>com.mjg.jca.udp.UDPListener</messaging-type>
        <transaction-type>Container</transaction-type>
      </message-driven>
   </enterprise-beans>

   <assembly-descriptor>
      <container-transaction>
         <method>
            <ejb-name>SyslogMDB</ejb-name>
            <method-name>*</method-name>
         </method>
         <trans-attribute>NotSupported</trans-attribute>
      </container-transaction>
   </assembly-descriptor>
</ejb-jar>
```

Views

The system defines two kinds of views: real-time and investigative.

In the example system, the events that drive the message-driven EJB are persisted into a database and propagated through a JMS topic. For now, the Topic is used to drive a real-time view, though it could just as easily drive a correlation engine. The events persisted into the database are used in a non real-time way by the investigative views. The rest of this section describes the working of the two views.

Real-Time Views

A console application is an essential part of a complex enterprise application. Veteran security analysts set great stock by tailing logs to a console, watching the logs stream by, and then detecting patterns in the stream. Once such an analyst understands the general “pattern” of the logs that stream by, deviations in the pattern will trigger the analyst to action.

However, as laws and regulations require more and more monitoring, the supply of analysts, and their ability to extract useful information from a free-scrolling console, will simply not keep up to the demand. Consequently, real-time console views will have to get better at presenting information. The question becomes, what sort of view allows for the summarization of large quantities of information, the better to guide analysts to problem areas.

One candidate is the TreeMap, as described here: http://www.cs.umd.edu/hcil/treemap/. TreeMaps were first invented in order to present a compact visualization of directory tree structures. The initial object was to monitor the use of a disk drive shared by 14 users in order to make quick decisions about moving or deleting large files. A current example of a TreMap is SmartMoney Magazine's "Map of the Market" (http://www.smartmoney.com/marketmap/), which summarize a lot of market information into a very small place:

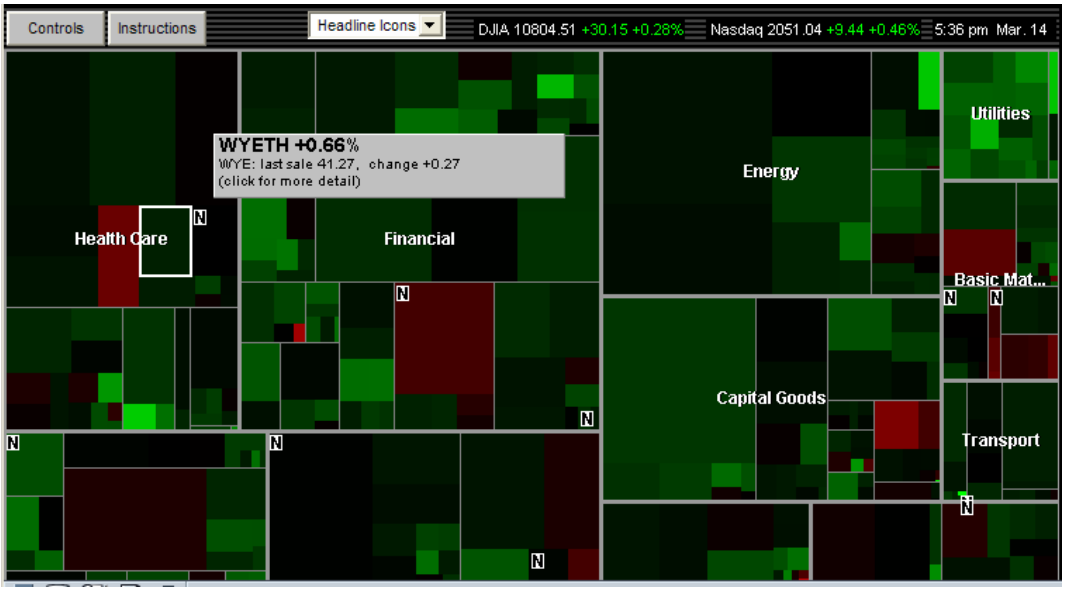

Figure 8: Smart Money Magazine's Map of the Market

TreeMaps present tree levels as alternating horizontal and vertical bars. The algorithm was first presented in a Technical Report (http://www.cs.umd.edu/local-cgi-bin/hcil/rr.pl?number=91-03) in March of 1991, and was published in the ACM Transactions on Graphics (http://www.acm.org/pubs/citations/journals/tog/1992-11-1/p92-shneiderman/) in January of 1992.

This project uses an open source TreeMap library from http://treemap.sourceforge.net/.

The important point about treemaps is their ability to summarize a lot of information in a small amount of space. This is key to leveraging analyst skills in a world where requirements and threats are rapidly evolving.

Real-Time View Details

The real-time view presents information that is fed to it in real-time over a JMS connection. The details of this connection are presented below.

JMS

The Java Message Service (JMS) API provides a common way for Java programs to create, send, receive and read an enterprise messaging system's messages. The messaging system defines two sorts of messaging schemes: point-to-point and publish/subscribe.

A point-to-point (PTP) product or application is built around the concept of message queues, senders, and receivers. Each message is addressed to a specific queue, and receiving clients extract messages from the queue(s) established to hold their messages. Queues retain all messages sent to them until the messages are consumed or until the messages expire.

- Each message has only one consumer.
- A sender and a receiver of a message have no timing dependencies. The receiver can fetch the message whether or not it was running when the client sent the message.
- The receiver acknowledges the successful processing of a message.

Use PTP messaging when every message must be processed successfully by one consumer.

In a publish/subscribe (pub/sub) product or application, clients address messages to a topic. Publishers and subscribers are generally anonymous and may dynamically publish or subscribe to the content hierarchy. The system takes care of distributing the messages arriving from a topic's multiple publishers to its multiple subscribers. Topics retain messages only as long as it takes to distribute them to current subscribers.

Pub/sub messaging has the following characteristics.

- Each message may have multiple consumers.
- Publishers and subscribers have a timing dependency. A client that subscribes to a topic can consume only messages published after the client has created a subscription, and the subscriber must continue to be active in order for it to consume messages.

The JMS API relaxes this timing dependency to some extent by allowing clients to create *durable subscriptions*. Durable subscriptions can receive messages sent while the subscribers are not active. Durable subscriptions provide the flexibility and reliability of queues but still allow clients to send messages to many recipients. Use pub/sub messaging when each message can be processed by zero, one, or many consumers.

Since the intent in this system is to publish system message events to many subscribers, the system uses the JMS pub/sub model to broadcast messages.

The system defines a Syslog topic as follows:

```
<mbean code="org.jboss.mq.server.jmx.Topic"
       name="jboss.mq.destination:service=Topic,name=syslogAggregatorTopic">
   <depends optional-attribute-name="DestinationManager">
      jboss.mq:service=DestinationManager
   </depends>
   <depends optional-attribute-name="SecurityManager">
      jboss.mq:service=SecurityManager
   </depends>
   <attribute name="SecurityConf">
      <security>
         <role name="guest" read="true" write="true"/>
      </security>
   </attribute>
</mbean>
```

When this XML file is deployed inside the J2EE stack a JMS topic, syslogAggregatorTopic, is created and made available for subscriptions.

Real-time View Sequence Diagram

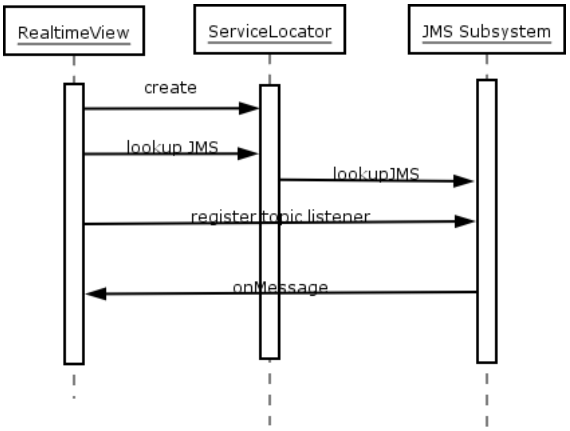

Figure 9: Real-time View Sequence Diagram

The message-driven bean interacts with the topic to propagate Syslog event messages to the rest of the system, while the view implements the JMS MessageListener interface, and uses it to drive the TreeMap view.

The application initialization subscrbes as follows:

```
private void appInit() throws Exception {

  // Lookup the managed connection factory for a queue
  topicConnectionFactory =
      (TopicConnectionFactory) ServiceLocator.
        getInstance().getObject("ConnectionFactory");

  // Create a connection to the JMS provider
  topicConnection = topicConnectionFactory.createTopicConnection();

  topicConnection.start();

  // Creat a queue session
  topicSession = topicConnection.createTopicSession(true, Session.AUTO_ACKNOWLEDGE);

  // Lookup the destination you want to publish to
  Topic topic = (Topic) ServiceLocator.getInstance().getObject("topic/syslogAggregatorTopic");

  topicSubscriber = topicSession.createSubscriber(topic);

  topicSubscriber.setMessageListener(this);
}
```

and receives messages as follows:

```
public void onMessage(Message message) {
  try {

    ObjectMessage objectMessage = (ObjectMessage) message;

    SyslogMessageEvent syslogMessageEvent =
        (SyslogMessageEvent) objectMessage.getObject();

    //  Update the Tree Map object
    …

  } catch (Throwable t) {
    t.printStackTrace();
  }
}
```

Sample TreeMap

The real-time view contains a tree based on the source of the Syslog message, the host being reported on, the facility, and the severity of the Syslog event.

Here's a sample of the TreeMap in action:

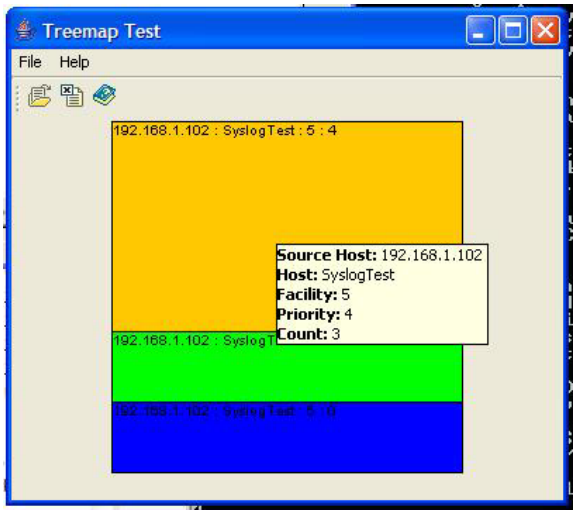

Figure 10: TreeMap Real-time View

In this case, we see that Source Host 192.168.1.102 has sent three facility messages of priority 4.

Investigative Views

While the real-time view is useful to detect trends in the monitored network, an analyst will also need an investigative view to examine events of interest during a particular time range. For that reason, the system also provides a framework for building views that allow a user to specify search criteria, which are then used to search the database.

Investigative View Details

Sequence Diagram

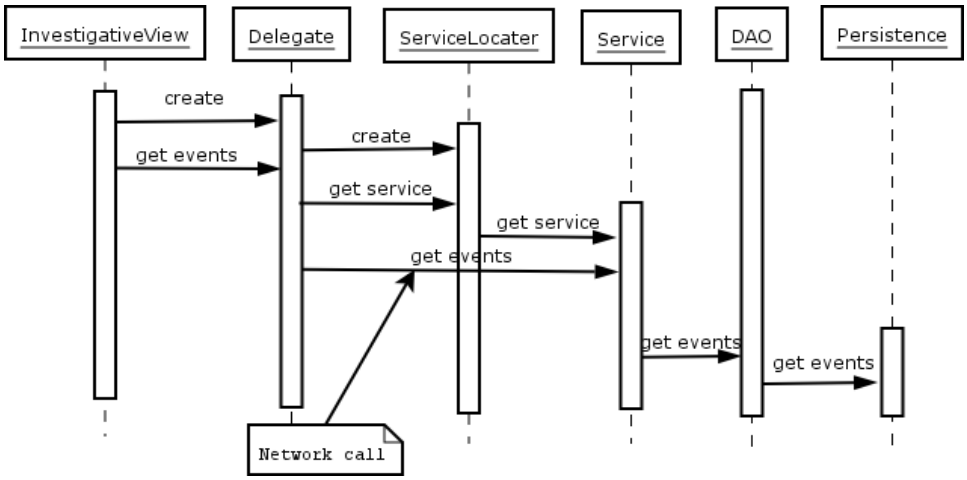

Figure 11: Investigative View Sequence Diagram

When an analyst wishes to investigate a certain collection of events, a set of search criteria is first specified. These criteria are then used to search the database.

Since this system is distributed, the investigative view uses a service layer that is "remoted." That is, at some stage in the search for events, the program will make a call for information over a network network connection. To do this, the investigative view employs a number of standard software patterns.

Delegate

The delegate layer hides the details of the remoted services from the software developer. Among other things, this provides the option of not using a network call if a high-performance interface is required. Of course, in that case the investigative view would have to be executing on the actual Syslog server.

Service Locator

The service locator is another standard pattern that's used to look up services deployed on the J2EE server. Among other things, this allows the service implementation to be changed without impacting code that depends on the service.

Service

The service layer is implemented as a Stateless Session Enterprise Java Bean. This allows services to be invoked remotely using a Java facility called Remote Method Invocation.

Reports

This system uses the OpenReports reporting system, version 0.7. This has the advantage of being freely-available. The downside is it needs a bit of tailoring to get it to work on the one-box solution. As the project matures, the amount of tailoring required is bound to decrease.

The OpenReports wiki notes that deploying OpenReports as a Web Archive (.war) file causes exceptions. So, deploying the Syslog aggregator and the Syslog reporter on the same box means that they must run in different servers. This means, in turn, that the OpenReports Tomcat configuration must be changed so it doesn't conflict with the JBoss Tomcat configuration. This can be accomplished by updating the OpenReports `.../conf/server.xml` file and changing the `80XX` ports to something like `180XX`.

Database Considerations

It's important to distinguish two database concerns in the reporting system. OpenReports needs access to a database in order to store administration information. It also needs access to a database to generate reports. The two databases are logically separate concerns, even if they're physically in the same database.

OpenReports Administration Database Configuration

Since the system uses MySQL, the OpenReports `.../webapps/openreports/WEB-INF/classes/hibernate.properties` file must be updated to work with MySQL. Here's a suggested list of entries for this system:

```
# MySQL dialect settings
hibernate.dialect net.sf.hibernate.dialect.MySQLDialect

# MySQL connection properties
hibernate.connection.driver_class=com.mysql.jdbc.Driver
hibernate.connection.url=jdbc:mysql://syslogdbhost/syslog
hibernate.connection.username=admin
hibernate.connection.password=password

# Commons-DBCP connection pool properties
hibernate.dbcp.maxActive=5
hibernate.dbcp.whenExhaustedAction=1
hibernate.dbcp.maxWait=5000
hibernate.dbcp.maxIdle=2
hibernate.dbcp.ps.maxActive=5
hibernate.dbcp.ps.whenExhaustedAction=1
hibernate.dbcp.ps.maxWait=5000
hibernate.dbcp.ps.maxIdle=5
hibernate.dbcp.validationQuery=select 1
hibernate.dbcp.testOnBorrow=true
hibernate.dbcp.testOnReturn=false
```

In addition, the sql statements in the `.../docs/database/or_ddl_mysql.sql` file must be sourced into the MySQL database so the OpenReports server can run with MySQL. For the purposes of the Syslog system, this file has been updated to include the line:

```
INSERT INTO REPORT_USER (NAME, PASSWORD, ADMIN, EXTERNAL_ID, EMAIL_ADDRESS, PDF_EXPORT_TYPE)
                VALUES ('admin','password',1,'','',0);
```

This creates an admin user, which can be used to configure the reporting process.

Once the database is properly created, and OpenReports is properly configured to use that database, the user can create reports.

OpenReports Report Data Database Configuration

Before creating reports, a database connection must be created. The database connection described above is for OpenReports administration. The database connection for reports is used to retrieve data for reports. An administrator can create a database connection via the administration interface.

Figure 11 shows the creation of such a database connection:

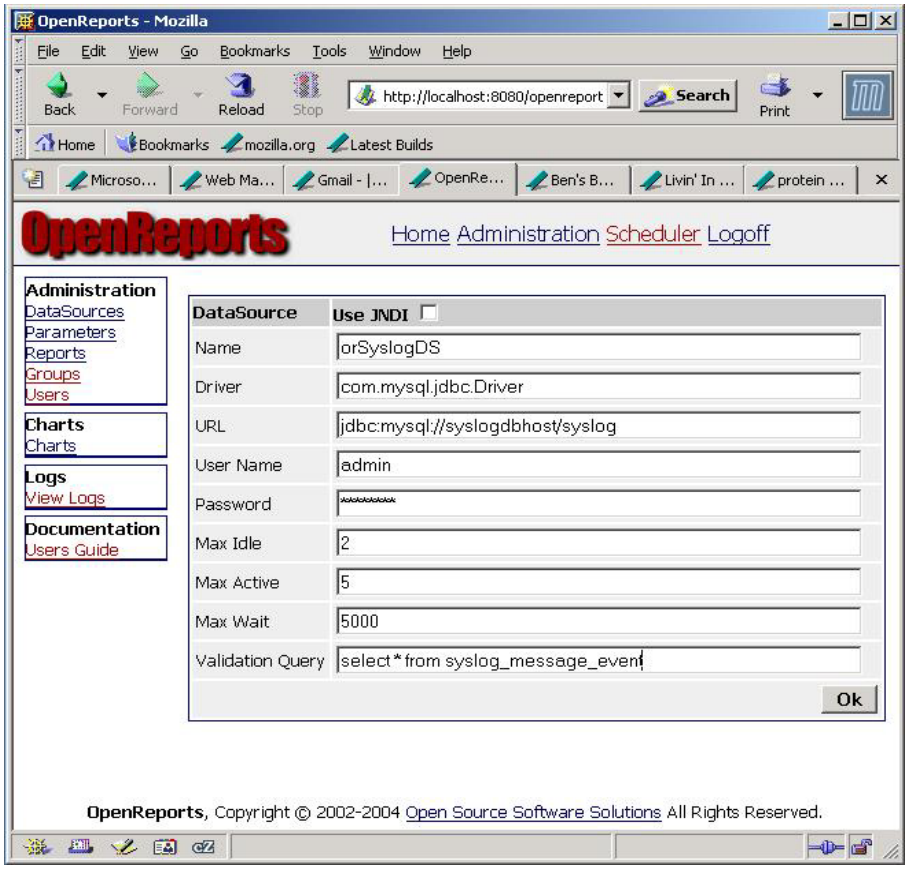

Figure 12: OpenReports Database Connection

Once this is created, then the reports and charts can refer to "orSyslogDS" as the database connection.

System Implementation

Syslog Hosts

The system assumes the existence of two hosts: an application server host and a database server host. Both logical hosts can be the same physical machine, with the proper settings in the machine hosts file.

On most Unix machines, the host file is /etc/hosts. For Windows machines, the location depends on the version of the operating system:

- Windows XP: C:\WINDOWS\SYSTEM32\DRIVERS\ETC
- Windows 2000: C:\WINNT\SYSTEM32\DRIVERS\ETC

Placing the following entries in the hosts file:

```
127.0.0.1       syslogappserverhost
127.0.0.1       syslogdbhost
```

will set up the local machine as both the application server and database host.

Database Setup

As supplied, the system assumes a MySQL database, with a URL of `jdbc:mysql://syslogdbhost/syslog`. This can be changed by edits to the files in the syslog-persistence project. The system uses Hibernate (http://www.hibernate.org/) to do Object/Relational mapping and persistence operations, so no SQL need be changed to migrate to a new database.

User

The system assumes there's a syslog-admin user with access to the `jdbc:mysql://syslogdbhost/syslog` database. The process for creating new users is described in the MySQL manual, at http://dev.mysql.com/doc/mysql/en/Adding_users.html.

Schema

The schema is built into the file …\syslog-persistence\dist\syslog.sql which can be used to create the database. Once the syslog-admin user is set up in the database, the normal build process will create the schema by default.

Application Server Setup

The system has been developed and tested under JBoss 4.0.0. Deployment under a different J2EE application server will require tailoring of deployment descriptors.

Under JBoss, the JDBC driver for the chosen database must be placed in the lib directory of the server where the Syslog aggregator enterprise archive (EAR) is deployed. The rest of the deployment files are in the EAR.

Build it Yourself

This system is composed of a set of packaged components, as depicted below.

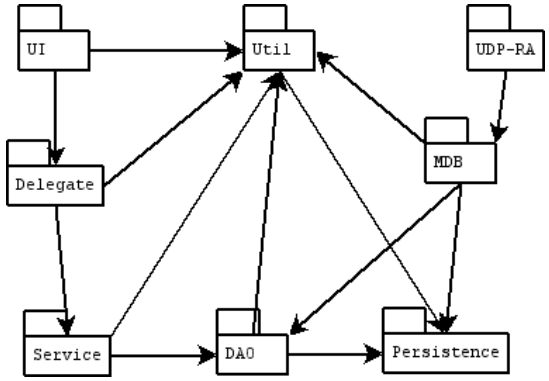

Figure 13: System Packaging Structure

So, building the system involves starting at the lowest level package, and then building up to the highest package.

Select packages include some interesting technology demonstrations:

- The Util package includes JUnit-based unit tests that are executed inside a JUnit GUI.
- The Service package demonstrates the use of Ejbdoclet to generate EJB code
- The Persistence layer uses Hibernate tools and mapping files to generate a DBMS schema corresponding Java objects.

Package Descriptions

The project is organized into a set of sub-projects.

- UI – The user interfaces for the system.
- UDP-RA – A J2EE Connector Architecture package that integrates UDP into the J2EE stack.
- MDB – A J2EE Message-driven EJB that converts UDP packets to Syslog messages, and passes the messages to the rest of the system.
- Delegate – A wrapper around the EJB service layer to hide the details of the service layer remoting
- Service – An stateless session EJB-based set of services
- DAO – A Data Access Object layer, to hide the details of lower-level stabase access
- Persistence – A Hibernate-based DBMS access layer.
- Util – A set of utility classes.

The Ant `build.xml` in the syslog-aggregator project has three targets: "clean", "build", and "dist". These will invoke the "clean", "build" and "dist" targets on all the sub-projects, finally combining the results into an enterprise archive file.

Building the system depends on three sets of resources: source code, third-party components, and directives – via build files – that determine how the components and source code are combined.

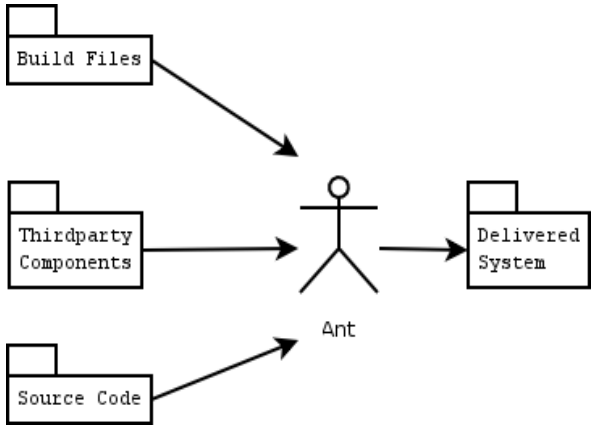

Figure 14: Ant Build System

Ant makes it easy to manage these dependencies. As described previously, the system includes a project, syslog-aggregator, that uses Ant to manage all the details of building subsystems in the proper order.

Third-party Components

Third-party components are maintained in a third-party components project. The project consists of a collection of directories containing jars, and a properties (thirdparty-components.properties) file that allows for abstraction of access to the jars.

A typical entry in the thirdparty-components.properties file is

```
#
# commons-collections: http://jakarta.apache.org/commons/collections/
#
commons-collections-version=3.1
commons-collections-dir=${thirdparty-components-dir}/commons-collections-${commons-collections-version}
commons-collections-lib-dir=${commons-collections-dir}
commons-collections-jar=${commons-collections-dir}/commons-collections-${commons-collections-version}.jar
```

This shows where the third-party component jar came from, and allows access based on the version of the component.

Ant defines a `user.home` variable which is the user's login home. For Windows, this is `C:\Documents and Settings\User Name`. In it a user can locate properties files that control the loading of third-party components. For this project, there's a `syslog.build.properties` file that contains the entries:

```
jboss.home=C:/Development/thirdparty-components/app-servers/jboss-4.0.0
jboss.config=default
thirdparty-components-dir=E:/syslog-workspace/thirdparty-components/
```

Thus, configuration parameters are established for JBoss and third-party components. In the Ant `build.xml` files for each component of the project, the `syslog.build.properties` and `thirdparty-components.properties` files are accessed as:

```
<!-- Load references to third-party components. -->
<property file="${user.home}/syslog.build.properties"/>
<property file="${thirdparty-components-dir}/thirdparty-components.properties"/>
```

Allowing us finally to say things like:

```
<path id="build.classpath">
  <pathelement path="${commons-collections-jar}"/>
  …
</path>
```

J2EE Application Server (JBoss)

As we might guess from the `syslog.build.properties` above Syslog aggregator project is deployed in the JBoss default server. The project also includes client applications for real-time and investigative viewing of Syslogs. These applications depend on jars in the JBoss client directory for proper operation.

Future Directions

As presented, this document describes how to integrate Syslog messages into a J2EE environment. Included in the demonstration are real-time and investigative views, as well as a reporting subsystem.

Plans for ongoing development include integration of a rules engine (http://herzberg.ca.sandia.gov/jess/), upgraded reporting system, and vendor-specific Syslog message parsing.

Summary

This paper has shown how the Java 2 Enterprise Edition (J2EE) software stack, as implemented in the JBoss Application Server, can be used to develop an application that supports security policy enforcement and management via real-time views, investigative views, and reporting. Furthermore, the paper has demonstrated the usefulness of Open Source Software (OSS), and also shown some best practices for enterprise software projects.

Glossary

DAO	Data Access Object
DB	Database
DBMS	Database Management System
EIS	Enterprise Information System
EJB	Enterprise Java Bean
GUI	Graphical User Interface
HIPAA	Health Insurance Portability and Accountability Act
IIOP	Internet Inter-Orb Protocol
JCA	J2EE Connector Architecture
JMS	Java Message Service
MDB	Message Driven Bean
OSS	Open Source Software
RA	Resource Adapter
RMI	Remote Method Invocation
UDP	User Datagram Protocol