OpenNMS qosdaemon Documentation

This document describes the qosdaemon which implements the OpenNMS OSS/J Qos Interface and has been contributed to OpenNMS by the University of Southampton.

Author: Craig Gallen
Version:1.4
Date: 29-11-06

Table of Contents

1 Introduction to OSS/J

Operational Support Systems through Java (OSS/J) is an initiative initially championed by Sun Microsystems but now incorporated into the Telemanagement Forum Prospero program. (see www.ossj.org and www.tmforum.org). OSS/J provides a means to implement the TMForum New Generation Operational Support Systems (NGOSS) framework in Java/J2EE environments.

The OSS/J program has developed specifications for a number of Java API's to ease the integration of Telecoms Operational Support Systems. The API's are collaboratively developed using the Java Community process and released as both Java Interface specifications and supporting XSD definitions. Fully implemented OSS/J interfaces support both Java Value Type (JVT) interactions using interfaces defined using an ejb facade pattern and Message Orientated Middleware (MOM) style interfaces using XML messages transported using JMS. Going forwards new API's will be defined with profiles for Web Service (WSDL) style interactions and the Web Services profile will be retrospectively applied to existing interfaces over time.

The earliest and, to date, the most widely deployed of the OSS/J API's are the Quality of Service (Qos) and Trouble Ticket (TT) interfaces. Subsequent to the initial definition of these interfaces, OSS/J finalised a Common Business Entities (CBE) model in order to provide a more consistent data model which was aligned with the TMForum SID. The newer OSS/J API's closely reference the CBE but the earlier TT and Qos interfaces are not 100% aligned with the current CBE. OSS/J are in the process specifying an aligned Trouble Ticket interface and separate Fault Management and Performance Management interfaces which will supersede the Qos api. However at the time this project started, the new specifications had not been released and it was preferred to work with an already finalised specification However the new Fault API is substantially similar to the Qos API and future support of the Fault API should be possible without a major re-write of the interface.. In our design we have chosen not to implement Qos interface functionality which we know will not be supported by the new Fault interface. (In particular we have avoided using the XmlSerialiser methods which have been depreciated in OSS/J).

The OSS/J Quality of Service (Qos) specification defines API's for accessing Performance Management (PM) and Fault Management (FM) data on a Network management System (NMS). It also defines a set of events which the NMS can publish to a Topic using a JMS provider to inform clients of changes in state. The PM and FM sides of the interface can be implemented separately The FM interface exposes an alarm list formatted according to the ITU-T X.733 standard and also generates events as JMS messages corresponding changes in the alarm list (for example NewAlarmEvent, ClearedAlarmEvent, AlarmAcknowledgeEvent and AlarmChangedEvent). Clients of the Qos interface can register with the JMS provider to receive only alarms corresponding to certain filter criteria such as PerceivedSeverity or ManagedObjectType or Instance. This allows for a truly distributed system where clients only register to receive messages relevant to their purpose. It is possible for Clients to maintain a local copy of the state of an alarm list using events alone - which avoids the performance hit and potential latency involved in regularly querying the alarm list. The query interface is required however if the client needs to resynchronise it's view of the alarm list, say when it is starting up for the first time.

2 Design Philosophy

The qosdaemon project was initiated with a view to make the OpenNMS project more attractive for Telecoms applications where integration to other OSS systems was an important consideration. As a research project, we also wanted to offer a platform where OSS/J could be demonstrated as providing useful functionality and which would provide a vehicle for the wider community to implement and experiment with NGOSS type solutions in an Open Source environment.

The JCP process used by OSS/J requires that the specifications be released with reference implementations and Technology Compatibility Kits to test other implementation's against the specification. Thus the design goals of the OSS/J reference implementations are to provide complete and accurate implementations of the OSS/J specification but not necessarily to provide re-usable libraries or end user functionality. Also once complete, the OSS/J Reference Implementations are not necessarily subject to ongoing enhancement or maintenance by the original developers.

By way of contrast, the design goals of this project have been to utilise the OSS/J specifications to realise useful functionality in order to allow practical use cases to be demonstrated. Thus the focus has not been on accuracy or completeness of specification implementation but on the realisation of a viable end-user use case using OSS/J functionality within a community sustained open source project. It is very important to realise that two key mantras of successful open source projects are; 'Release early and release often.' and 'Make it easy for potential users or contributors to assess and begin using your offering'.

Therefore the design philosophy of the project has been; Firstly, to choose a use case for OpenNMS which would leverage OSS/J and would deliver immediate value to OpenNMS community. Secondly, to contribute the solution in such a way as it is fully sustainable as a mainstream contribution to the OpenNMS project even in it's initial release. Thirdly, to only implement enough OSS/J functionality as was necessary to support the use case. And finally, to structure the design in such a way as it will be possible for future contributions to go back and complete or address any non-conformances in the design of the interface.

A key aspect of the design has been the separate packaging of libraries (OSSbeans) which could be generally useful for OSS/J implementations from the OpenNMS specific interface code (qosdeamon). The OSSbeans libraries are in a separate Apache 2 licensed project on Source forge. The qosdaemon is fully integrated into the OpenNMS code tree and uses maven to incorporate the OSSbeans dependencies into the OpenNMS build. By this means it will be possible for the development of the OpenNMS and the OSSbeans projects to proceed asynchronously and it will also be possible for other projects to leverage OSSbeans. However the dependence of OpenNMS Qos interface on the OSSbeans libraries should ensure that there is sufficient community interest to sustain and carry forwards the OSSbeans project regardless of other users.

A key advantage of this design approach has been that the discipline of having to integrate into a real application (OpenNMS) has made OSSbeans more useful as a library. Throughout the design we have gone through multiple re factoring steps to partition the functionality so that it can easily be picked up by an external application. A second advantage arises from the fact that designing OSSbeans 2.1.0 as an initial offering has been something of an education. It is possible to see numerous areas where the design could be improved. Thus given sufficient interest, future release of OSSbeans will be able to learn the lessons of the first implementation in order to provide a more generally useful library. In addition the issues around OSS/J compliance are confined to the OSSbeans library which can have it's own roadmap towards compliance as contributions fill in the gaps.

3 OSS/J Conformance

This project provides partial implementation of the OSS/J QoS interface for OpenNMS. It is offered as an illustrative and training tool to explain OSS/J and to gauge interest from the OpenNMS community in taking the project forwards.

This project leverage's the OSSbeans project which provides the core classes for the OSS/J implementation. OSSbeans are a separate project hosted as part of the University of Southampton OpenOSS initiative at http://sourceforge.net/projects/openoss

The interface is based upon the OSS/J QoS specification available at www.ossj.org. The basic principles and design patterns of the the specification are implemented however not all of the mandatory functionality is complete and the interface has not been tested against the OSS/J SDK.

Where functionality is provided it does so using classes implementing interfaces conforming to the javax.oss interface tree and the XML messaging uses messages conforming to the OSS/J Qos XSD's. This provides a firm basis for moving towards full OSS/J compliance in future releases.

Some work was done previously by the Budapest University of Business and Technology (BUTE) to demonstrate that the PM interface could be implemented for OpenNMS. This work has not been incorporated into the present Qos interface but could be taken forwards later. The present Qos interface only implements FM functionality. (Note however that it is still possible for OpenNMS performance threshold crossing events to be converted into OSS/J faults reported by OpenNMS)

4 Production Use

The interface should be considered experimental and is not optimised for high load environments. Although included with OpenNMS, it can be completely disabled and will not then interfere with other OpenNMS components. However even in it's present form the interface may still be useful for some production solutions. Envisaged uses include;

* Integration of OpenNMS alarms with other Operational Support Systems using J2EE or JMS

* Monitoring important alarms from remote OpenNMS systems - potentially on a customer's site. (Note that to circumnavigate firewalls JbossMQ can be configured to send JMS messages using HTTP - although this has not been tested.)

As an example, the interface has been successfully used to integrate OpenNMS with an Alarm / Topology correlation engine from Sidonis. www.sidonis.com as part of a proof of concept for managing a Digital TV network

5 Licence

The qosdaemon project builds an OSS/J interface for OpenNMS. It is released with OpenNMS under the GPL licence and uses code contributed to OpenNMS by the University of Southampton.

OSSbeans (http://sourceforge.net/projects/openoss) are released under the Apache-2 licence by the University of Southampton.

6 Functionality Provided

The current release of the qosdaemon module leverage's OSSbeans Release 2.1.0 and provides the following functionality. The module provides two daemons which can be used independently or together. The daemons share a data access layer, OssDao which is a data access object which maps the OSS/J Qos interface onto OpenNMS's internal alarm list exposed by the OnmsDao.

The qosd daemon publishes the internal OpenNMS alarm list as an OSS/J alarm list. The qosdrx daemon allows an OpenNMS system to connect using the OSS/J interface to remote OpenNMS systems running qosd. This allows a 'master' OpenNMS to monitor the state of the alarms lists in 'slave' openNMS systems. The present implementation is almost exclusively JMS event driven with limited alarm list query functionality provided as a J2EE option on qosd.

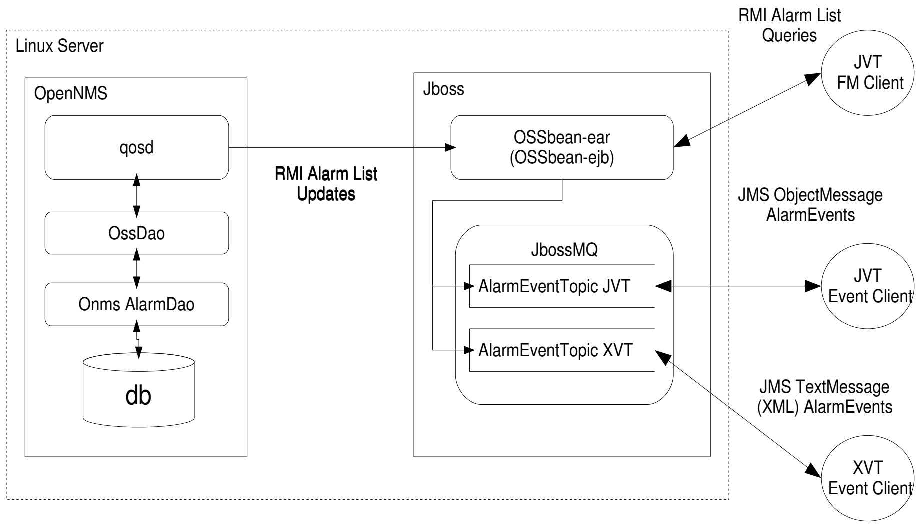
The implementation leverages JbossMQ as the JMS provider. In theory other JMS providers could be used but these have not been tested.

6.1 QoSD daemon

The QoSD daemon monitors the OpenNMS alarm list and generates OSS/J JMS events corresponding to changes in the state of the alarms in the list. It can run in two modes; natively on OpenNMS or in conjunction with a separate J2EE application.

6.1.1 Native OpenNMS provided interface.

OpenNMS does not run natively in a J2EE container but leverages the spring framework and JMX to provide a container like environment for it's daemons. The qosd daemon code can run natively as a spring application within OpenNMS. In this case it uses the OSS/J XVT (XML over JMS) profile to publish alarm list changes . The qosd daemon publishes OSS/J AlarmEvents as both JMS TextMessages and as JMS ObjectMessages containing AlarmEvent objects.



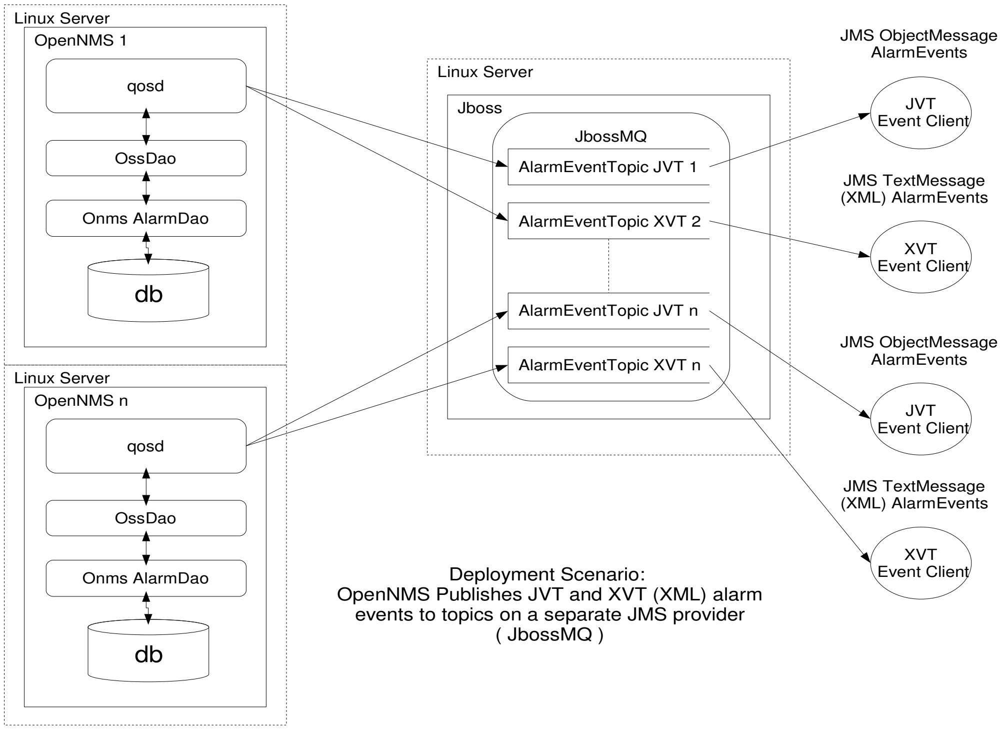
Deployment Scenario :
OpenNMS exposes OSS/J JVT interface using separate J2ee Application

6.1.2 Separate J2EE server provided interface.

An alternative configuration is possible where the qosd daemon connects to an ejb application running in a separate J2EE server. This application is known as OSSBeans-qos-ear and is available from the OSSbeans site. In this mode the ejb exposes OSS/J semantics and allows external applications to connect with the ejb as an OSS/J JVT interface. Note that only a very limited alarm list query functionality is currently provided (query for all alarms).

Note that this configuration requires a J2EE server (Jboss) to be hosting a OSSBeans-qos-ear locally to each OpenNMS implementation which is running qosd in this mode. In most circumstances, it is easier to use the native interface for the remote machines they can all use a single JbossMQ deployment and a local J2EE server is not required for each OpenNMS.

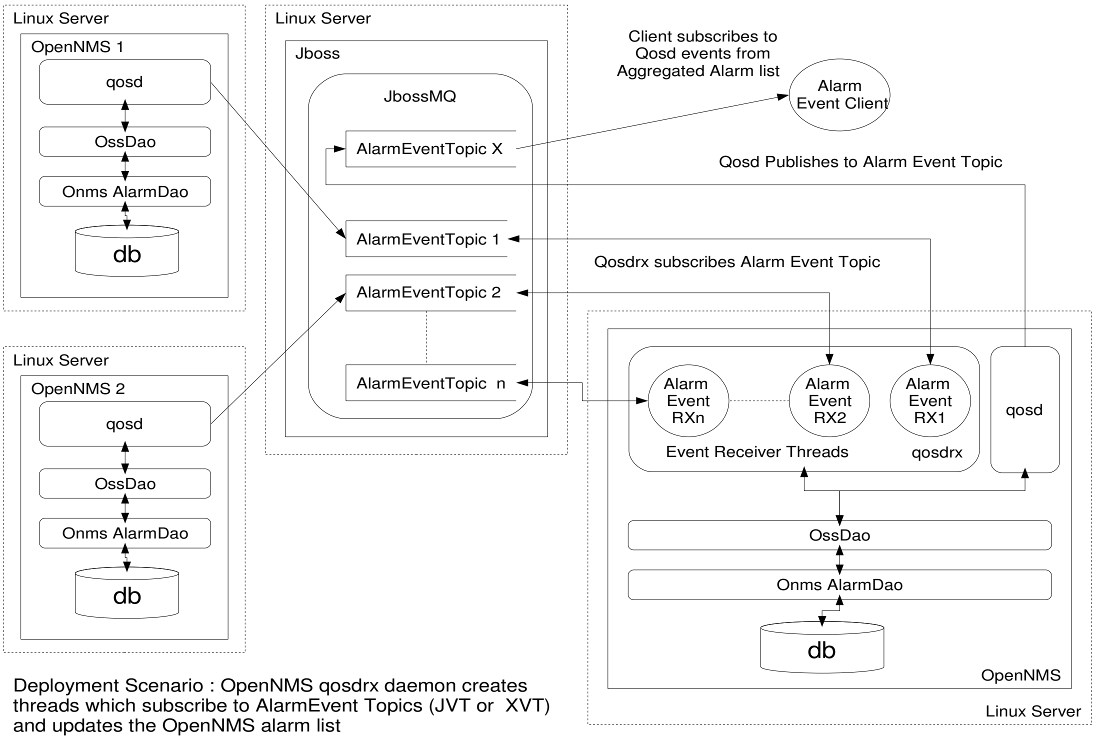
Which mode qosd is running is determined by a setting in the opennms.conf file:
To use the native OpenNMS provided interface use -Dqosd.usej2ee=false
To use the separate J2EE server provided interface use -Dqosd.usej2ee=true



Deployment Scenario:
OpenNMS Publishes JVT and XVT (XML) alarm events to topics on a separate JMS provider (JbossMQ)

6.2 QoSDrx

The qosdrx daemon can connect to multiple OSS/J event topics hosted on a JbossMQ server and receive OSS/J alarm events from remote OpenNMS systems running qosd. The local alarm list will be updated to reflect the remote alarm lists. Note that no resynchronization capability is provided at this time so it is possible for the alarm lists to get out of alignment if messages are lost. However in practice, the JMS messaging system should provide a reliable transport.



Deployment Scenario : OpenNMS qosdrx daemon creates threads which subscribe to AlarmEvent Topics (JVT or XVT) and updates the OpenNMS alarm list

7 OpenNMS qosdaemon Package Description

7.1 Functionality Overview

The qosdaemon provides an elementary OSS/J Qos interface for OpenNMS.

For more information on the OSS/J Qos Specification please see the OSS/J site www.ossj.org. For more details on configuration and functionality provided please see the documentation and example configuration in the $opennms_home/contrib/qosdaemon directory.

The interface can run natively in OpenNMS and use OSS/J XML messages to reflect the state of the OpenNMS internal alarm list or it can use an external ejb running in JBOSS to expose the alarm list as JVT objects. Currently only OSS/J XVT and XML AlarmEvents have been implemented. A future release will service OSS/J XML queries. The interface uses the services of an external JMS provider. We use the JbossMQ implementation of JMS however it should be fairly easy to substitute another JMS implementation.

This interface is in two parts which can be run separately as two opennms daemons;

qosd is an OSS/J server which exposes the OpenNMS alarm list as an OSS/J alarm list to other systems. The changes to the local alarm list are exposed as JMS XML messages which can be used to update an alarm list in a remote client.

qosdrx is an OSS/J event client which can listen for alarm update events from other servers. It updates the local OpenNMS alarm list based upon the received events.

During the development of the qosdaemon, OpenNMS migrated to using the Spring framework. This has been adopted extensively in the design of the qosdrx daemon. However some of the code in the qosd daemon pre-dated the introduction of spring and has been adapted to use spring rather than re-written. This has lead to a somewhat convoluted initialization process which will be tidied up in a later release. In general most classes use injection of control and spring wiring to get their dependencies and most classes are exposed as implementations of interfaces with the 'impl' suffix to their name. Where classes have been re-implemented or gone through several versions, the implxx suffix is used where xx indicates the version. This has allowed several versions of the same class to co exist and be wired in for testing etc during development. (As an example see QosDimpl2.java)

7.2 Package Overview

7.2.1 OssDao

Overview

The OssDao package provides a buffer between the internals of OpenNMS and the requirements of the OSS/J code. The internal alarm definition in OpenNMS contains different fields and severity values to the OSS/J X.733 definitions. This mapping is (mostly) confined to the OnmsAlarmOssjMapper class.

The OpenNMS alarm definition has been extended to allow it to represent alarms from remote systems. Each OpenNMS alarm now contains an ApplicationDN which identifies the unique name of the remote system which generated the alarm and an OssPrimaryKey which identifies the local identifier for that alarm on the remote machine. When an alarm update event is received, the OssDao searches for an existing alarm with matching ApplicationDN and OssPrimaryKey in order to do the update. (If ApplicationDN is blank then the alarm is from the local machine).

Each OSS/J alarm contains an identifier for the managed object which issued the alarm. This is always mapped to an OpenNMS node. The Asset Register entry for nodes now has extra fields to uniquely identify the managed object system wide (ManagedObjectInstance and ManagedObjectClass) . The OssDao provides methods to do a rapid reverse look up on the Asset register to find nodes with a unique id which is given by the concatenation of the ManagedObjectInstance and ManagedObjectClass.

Note that when the OssDao is initialized, the QoSDrx log file will record any nodes having no data or duplicate data in the ManagedObjectInstance and ManagedObjectType fields of the Asset table. If the ManagedObjectInstance and ManagedObjectType fields of nodes are empty or duplicated the OSS/J Alarm management will not work correctly.

Build Directory / package	***Class / XML configuration***	***Description***
org.openoss.opennms.spring.dao	OnmsAlarmOssjMapper	This class provides methods to map OSS/J AlarmValue to OpenNMS alarms. If the OpenNMS alarm model changes, then most of the changes to the qosdaemon will happen here
	OssDaoOpenNMSImpl	This class provides a single point of access to the OpenNMS alarm list and node asset table for both the Qosd and Qosdrx. It provides a cache which allows for rapid look up of alarms by ApplicationDN and OssPrimaryKey and of nodes by managedObjectInstance and managedObjectType
	OssDaoOpenNMSImplSingleton	This class provides a wrapper for OssDaoOpenNMSImpl which allows it to be created and initialized as a singleton by either qosd or qosdrx depending on which daemon starts first. Once initialized both daemons can use the same instance of the dao.

7.2.2 QoSD Daemon

Overview

The QoSD Daemon realises an OSS/J QoS server interface by mediating between the OpenNMS internal alarm management model and an OSS/J implementation realized using the OSSBeans OSS/J Qos library. The OSSbeans AlarmMonitor bean can run natively in the OpenNMS spring container or in a remote the Jboss J2EE server. If it is running in the remote J2EE server then it exposes it's alarm list according to OSS/J semantics as JVT objects through a facade bean. Note that presently only a very simple query is currently supported to get the entire alarm list. The AlarmMonitor bean also sends out XVT (XML) and JVT AlarmEvents as the alarm list changes. In the future it is intended to extend the OSSBeans alarm monitor to also support full JVT and XML queries. If the AlarmMonitor bean is running natively in OpenNMS, it cannot expose the OSS/J JVT interface and only generates the JMS events however it is envisaged that this is the way most users will want to use the interface.

How it works

The qosd.properties file contains the address information for the Qosd to connect to it's JMS queues. The configuration for the QoSD daemon is picked up from the /etc/qosd.properties file using the singleton PropertiesLoader class. (In the future this could be picked up using the spring application context in a similar manner to the qosdrx code.)

The QosD Daemon registers with the OpenNMS eventd daemon to receive events using the call;
eventIpcManager.addEventListener(this, ueiList);

The events which qosd responds to are determined by the ueiList object which is configured by the QosDConfigFactory using and the QoSDConfiguration.xml file in the OpenNMS /etc directory. Castor is used to un-marshal the XML into the ueiList according to the QoSDConfiguration.xsd file.

When an event which qosd has registered for occurs its onEvent() method is called. Qosd responds to two types of events. Firstly any events corresponding to changes in the asset register are used to force an update in the OssDao node cache. Secondly an 'alarm changed event' causes the qosd to review the local OpenNMS alarm list for changes.

The qosd depends upon the vacuumd daemon which must be configured with an automation to look for new or changed alarms in the local alarm list and throw an 'alarm changed event' if a change is detected. An example vacuumd-configuration.xml file is included in /contrib/qosdaemon/qos-example-configuration.

The vacuumd automation also implements the X733 alarm life cycle whereby OpenNMS correlates alarm raising and alarm clearing SNMP traps and removes cleared and acknowledged alarms from the alarm list.

Qosd responds to an alarm changed event by causing the OssDao to update it's cache of OnmsAlarm values with a new snapshot of the alarm list in the database, translates the OnmsAlarm values to OSS/J AlarmValues using the OnmsAlarmOssjMapper and then forwards this new list of alarms to the OSSBeans AlarmMonitor bean. The AlarmMonitor bean compares this new list with it's existing list and sends out AlarmEvents as alarms in its current list are added, deleted or changed to match the new list.

The AlarmListConnectionManager interface determines whether the remote J2EE or local spring implementation of the AlarmMonitor is used. Two implementations of the AlarmListConnectionManager are available to be wired in using spring; AlarmListConnectionManagerJ2eeImpl connects to the remote bean using RMI. AlarmListConnectionManagerSpringImpl.java implements the bean locally.

This choice is determined by the setting of the system variable -Dqosd.usej2ee={true}|{false}' in the /etc/opennms.conf file which is read when the qosd is initialized.

In a heavy alarm storm, vacuumd could trigger multiple alarm changed events such that qosd initiates multiple parallel look ups to the database which would be very costly. The OpenNMSEventHandlerThread is used to avoid this eventuality by latching any new alarm changed events while a database look up is being performed. Thus a lookup has to complete before a new one is initiated. All of the changes which occur during this time will be picked up on the next alarm retrieval.

Build Directory / package	*Class / XML configuration*	*Description*
org.openoss.opennms.spring.qosd:	AlarmListConnectionManager	Interface defining how qosd interfaces with OSSBeans AlarmManager bean
org.openoss.opennms.spring.qosd.ejb	AlarmListConnectionManagerJ2eeImpl AlarmListJ2eeConnectionManagerThread	Implementation of interface to remote AlarmMonitor bean on Jboss. ConnectionManagerThread establishes rmi connection to J2EE server and waits for server to be available if connection is lost
org.openoss.opennms.spring.qosd.spring	AlarmListConnectionManagerSpringImpl	Implementation of interface to AlarmMonitor bean running locally in OpenNMS
	QoSD.java QoSDimpl2.java	Interface and implementation of QoSD daemon.
	OpenNMSEventHandlerThread	Provides a latch for events which occur while a database lookup is happening
	QoSDConfigFactory.java UEIHandler.java PropertiesLoader.java	Classes for reading in configuration of QoSD
./src/main/castor	castorbuilder.properties QoSDConfiguration.xsd	Configuration for Castor XML marshalling framework. Defines the format and contents of the QoSD-configuration.xml file which sets up which OpenNMS events the qosd daemon listens for
org.openoss.opennms.spring.qosd.jmx:	QoSD.java QoSDMBean.java	All OpenNMS daemons are started as JMX beans in the same way. The QoSD.java and QoSDMBean.java in the qosd.jmx package are used by opennms to initiate new threads for running the daemon and to pass in it's spring application context in order to allow access to OpenNMS Daos and other daemons.
./src/main/resources/org/openoss/opennms/spring/qosd	OssjTypeSpecificationApplicationContext.xml	This spring application context file is used as a factory to configure the default settings for the OSS/J AlarmValues before they are populated with alarms.

Build Directory / package	*Class / XML configuration*	*Description*
	qosd-j2ee-context.xml qosd-spring-context.xml	These application context files are used to either set up a local AlarmMonitor bean or a remote AlarmMonitor bean on a J2ee server. Which application context is used is determined by the setting of the system variable -Dqosd.usej2ee={true}|{false}' in the /etc/opennms.conf file
org.opennms.web.alarm	Alarm	A copy of the org.opennms.web.alarm.Alarm class is included in this project to provide a severity mapping for OpenNMS alarms. This is done because at the time of writing OpenNMS did not generate a webapp jar which could be accessed through MAVEN. If the webapp is made into a maven package then this can be replaced with a reference in the pom file. Alternatively (and better) OpenNMS could include the severity mapping centrally in the OnmsAlarm model.

7.2.3 QoSDrx Deamon

Overview

The QoSDrx daemon implements multiple OSS/J JMS event listener clients for OpenNMS. Each client connects to a different OSS/J JMS AlarmEvent Topic on a JMS provider (tested with JbossMQ but other providers could be used) . Alarm Events are used to update the local OpenNMS alarm list such that it mirrors the alarm event list in the remote OSS/J servers. The clients are fully configurable using an XML configuration file in the /etc directory. Planned future extensions to the Daemon will allow it to also synchronize with remote servers using OSS/J XML queries.

How It Works

The QoSDrx Daemon realises an OSS/J QoS client interface using the OSSBeans OSS/J Qos library. The Library provides a collection of classes for implementing clients and servers for receiving or transmitting OSS/J messages as XML or JVT objects. Fundamental to the library is the concept of an OSSbean which is spring class which can be run as a separate thread for handling OSS/J messages and is configured using a spring application context XML file.

As with the QoSD daemon, the qosdrx.jmx classes (QoSDrx and QoSDrxMBean) are used by opennms to spawn the qosdrx daemon and initialise an OssDao if it does not already exist. The daemon passes the OpenNMS application context to the OSSBeans OSSBeanRunner class which reads the /etc/QoSDrxOssBeanRunnerSpringContext.xml file to generate an application context and spawn separate threads for each defined OssBeanAlarmEventReceiver class. Each OssBeanAlarmEventReceiver then registers to listen to it's defined JMS AlarmEventTopic and waits to receive OSS/J AlarmEvents (both XVT and JVT events are supported). If any receiver

cannot attach or gets disconnected from its JMS Topic, it waits for a defined timeout and then tries to reconnect.
The OssBeanAlarmEventReceivers need to have AlarmEventReceiverEventHandler classes assigned which are used to perform appropriate actions on reception of each event. The QoSDrx daemon defines an AlarmEventReceiverEventHandler implementation which interacts with the OssDao to update the OpenNMS Alarm list database on each received event. (The present QoSDrx demon AlarmEventReceiverEventHandler implementation is called QoSDrxAlarmEventReceiverEventHandlerImpl2)
The intention of this design is to extend the OSSbean concept in the future to make each OSSbean also a JMX Mbean. This would allow finer real time monitoring and management of each receiver client. However in the present design all of the OSSbeans are spawned off the root OpenNMS JMX bean which is able to report the statistics from all of the beans using the OpenNMS MX4J console.

Alarm Life cycle

Note that the present use case only utilizes NewAlarmEvent and AlarmCleared events to update the Alarm list. Other AlarmEvents are simply logged. However the QoSDrxAlarmEventReceiverEventHandlerImpl can easily be extended to use more of the OSS/J alarm event types if a tighter coupling between client and server is required. The following process is followed by QoSDrx to process AlarmEvents;

1. When an NewAlarmEvent message is received, it's ApplicationDN and OssPrimaryKey are checked against the OpenNMS alarm list. If the alarm exists, there is an error which is logged and the NewAlarmEvent message is ignored (the NewAlarmEvent messages should refer only to a new not an existing alarm).
2. If the alarm is new (i.e. Not in the AlarmList), QoSDrx has two modes of operation based upon how it is desired to handle the managedObjectInstance and ManagedObjectType fields in the incoming messages.
   a) Alarm can be logged against a node which is named to represent the AlarmTopic. Each OssBeanAlarmEventReceiver is given a unique name in the QoSDrxOssBeanRunnerSpringContext.xml file. In this case OpenNMS should be configured with nodes named after the name of each OssBeanAlarmEventReceiver. When an alarm comes in to a given topic, the alarm is logged in OpenNMS against a node named with the same name as the OssBeanAlarmEventReceiver. This is a good configuration if you simply want to look at alarms from subordinate OSS/J servers on a per server basis but you don't want to put every managed object in the OpenNMS database corresponding to the managedObjectInstance and ManagedObjectType of all the objects being monitored remotely.
   b) Alarms can be logged against nodes having the same managedObjectInstance and ManagedObjectType fields as the incoming message. In this case the listening OpenNMS must have a database populated with all of the nodes (with the same Asset data for managedObjectInstance and ManagedObjectType) as populated in the subordinate OpenNMS's
   Note that in both the above cases, regardless of how the alarm is displayed locally, it will be forwarded through QosD with the same managedObjectInstance and ManagedObjectType as the original message. (i.e. OpenNMS will not modify the origin of the message even though

it displays it against a different node locally)

3. When an AlarmClearedEvent message is received it's ApplicationDN and OssPrimaryKey are checked against the OpenNMS alarm list. If the alarm does not exist, there is an error which is logged and the ClearedAlarmEvent message is ignored (the ClearedAlarmEvent messages should refer to an existing alarm).
4. If the corresponding alarm exists in the Alarm List, it will be set to cleared and acknowledged. The vacummd process will subsequently remove the cleared and acknowledged alarm from the alarm list.
   Note that the behavior of the interface could easily be extended to allow usage of separate acknowledgment messages of cleared alarms before removing them from the list. However this has not been done as it appeared to complicate the use case without much extra value. In the future we will make alternative alarm life cycle behaviour's a configurable item.

Build Directory / package	***Class / XML configuration***	***Description***
org.openoss.opennms.spring.qosdrx.jmx	QoSDrx.java QoSDrxMBean.java	As with Qosd used by OpenNMS to start the Daemon.
org.openoss.opennms.spring.qosdrx	QoSDrx.java	Class which initialises the OssDao and loads the OSSbeanRunner in order to start the OSS/J AlarmEvent clients.
	QoSDrxAlarmEventReceiverEventHandlerImpl2 QoSDrxAlarmEventReceiverEventHandlerImplShell	QoSDrxAlarmEventReceiverEventHandlerImpl2 provides a handler for OSS/J events received which updates the OnmsAlarm list through the OssDao. Note that an unused QoSDrxAlarmEventReceiverEventHandlerImplShell simple logs all events received. This provides a template for writing any other AlarmEvent handling behaviour.
./src/main/resources/org/openoss/opennms/spring/qosdrx	qosdrx-spring-context.xml	Initial spring context used to start the OSSbeanRunner which then loads the QoSDrxOssBeanRunnerSpringContext.xml file and launches the AlarmEventReceiver clients.

8 Appendix 1: Example Configuration of the OSS/J interface

An example configuration for the OSS/J interface is provided in the OpenNMS /contrib/qosdaemon directory. The following table describes the contents of each of the files.

The simplest way to get the OpenNMS Qos interface working is to build and install opennms of Fedora Core 4 along with tomcat55 and then run the qos installation script, opennms_1_3_2_example_deploy_xdotx.sh, prior to starting OpenNMS. (See appendix on installation on FC4 below)

Configuration File	Purpose
/qosdaemon	
README.txt	
LISCENCE(gpl).txt	
/testscrips	
opennms_IF.sh opennms_IFOpenOSS1.sh opennms_IFOpenOSS2.sh opennms_IFOpenOSS3.sh	opennms_IF.sh runs a small client program called SentinalIF.java. This uses the properties in qosclient.properties to connect to the AlarmEvent Topic queue and receive AlarmEvents. The client can display received events and can also forward much simplified XML representation of the X733 Alarm fields to a remove socket for interfacing to other applications. For usage information type sh opennms_IF.sh -help. The opennms_IFOpenOSSx.sh scripts are convenience scripts for starting the same client as opennms_IF.sh using the properties in the qosclientOpenOSSx.properties files)
qosclient.properties qosclientOpenOSS1.properties qosclientOpenOSS2.properties qosclientOpenOSS3.properties	qosclient.properties sets up the configuration for the client interface to connect to anAlarmEvent Topic
/qos_example_configuration	This example configuration provides a simple example of how to set up the OpenNMS qosd application. This should be used to help you work out how to incorporate the qosdaemon into your local configuration.
README.txt	
opennms_1_3_2_example_deploy_1dot0.sh	This script provides a simple method to deploy all of the example configuration files into the correct directories. Notes: 1. This script has been designed for use on a Fedora core 4 installation using a jpackage

	installation of tomcat55. A standard installation of Jboss 4.0.2 is expected to be simlinked from /opt/jboss and OpenNMS is expected to be installed at /opt/OpenNMS. 2. The resulting configuration will leave jboss configured to run on port 8080 and the OpenNMS tomcat at 8081. This is to allow tomcat and jboss to run on the same machine. If they are running on separate machines, the tomcat setting need not be changed. 2. You must also change the hosts file to create a hostname jbossjmsserver1 pointing to your running jboss server. To do this from the kde toolbar: 3. open /system settings/ network select the hosts tab 4. select new and add a host with Hostname: jbossjmsserver1 Address 127.0.0.1 WARNING: without modification this script will overwrite tomcat55 and OpenNMS configuration files in $OPENNMS_HOME/etc so only use on a new install or if you are happy you have backed up your local OpenNMS configuration files
/jboss	This folder contains configuration files to set up JbossMQ messaging (and optionally OSSbeans-qos-ear-xx.ear if it is installed)
log4j.xml	Sets up logging to minimise to INFO log messages from OSSbeans-qos-ear-xx.ear if installed
openoss-jms-service.xml	Sets up 10 separate example AlarmTopics and Message Queues to allow up to 10 OpenNMS qosd deamons to publish to separate topics. It should be fairly obvious how to increase / decrease the number of topics or otherwise change this configuration. Notes 1. The names of the queues match the naming conventions of the OSS/J qos specification. This is merely a convention. The names are actually treated as free format strings. 2. It should be easy to see how the Topic/ Queue names are matched to the names in the qosd.properties, qosdclient.properties and QoSDrxOssBeanRunnerSpringContext.xml
openoss_qos_jboss_start.sh	Starts jboss with the following system settings: -Djava.rmi.server.hostname=jbossjmsserver1 -DqosbeanpropertiesFile= /opt/jboss/server/default/conf/props/qosbean.properties (TODO - this appears not to be used – why?)
uil2-service.xml	Sets up the uil2 transport configuration for JbossMQ.

	This could be changed to set up different JbossMQ transport configuration
/opennms	This folder contains the core configuration files to get the qosdaemon running
opennms.conf	This file is used to set system properties for OpenNMS. The following properties are required for qosdaemon: -Djava.security.policy=/opt/OpenNMS/etc/rmi.policy The security policy is required to allow OpenNMS to make an rmi connection to the OSSbeans-qos-ear-xx.ear application if installed on jboss. -Djava.naming.provider.url= jnp://jbossjmsserver1:1099 This give the name of the jndi naming provider it is set to jbossjmsserver1 which should be the host name of the Jboss server running JbossMQ. (This name can of course be changed if all the other configuration references to jbossjmsserver1 are changed) -Djava.naming.factory.initial= org.jnp.interfaces.NamingContextFactory -Djava.naming.factory.url.pkgs=org.jboss.naming Points to the Jboss naming factory (this class is in the Jboss Client library This could in theory be changed to point to another J2ee provider if the correct client classes are available. However this has not been tested) -DpropertiesFile=/opt/OpenNMS/etc/qosd.properties qosd.properties provides the configuration for the JMS topics which opennms will publish to. -Drx_propertiesFile= /opt/OpenNMS/etc/qosdrx.properties (Note :This configuration is not used as the current qosdrx configuration is contained in QoSDrxOssBeanRunnerSpringContext.xml. However qosdrx properties could be used if the commented out example referencing lines in QoSDrxOssBeanRunnerSpringContext.xml are enabled. -Dqosd.usej2ee=false If set false , this property tells qosd to run internally to opennms and publish to the external JbossMQ topics. The interface then only supports AlarmEvents but runs natively in OpenNMS. This is the simplest configuration and recommended for normal use.

	If set true, this property tells qosd to connect to the OSSbeans-qos-ear-xx.ear running in a local Jboss server. This then allows JVT access to the alarm list.
service-configuration.xml	This file is used by OpenNMS to startup it's deamons. To run qosd uncomment the section beginning <service> <name>OpenNMS:Name=QoSD</name> To run qosdrx uncomment the section beginning: <service> <name>OpenNMS:Name=QoSDrx</name> qosd and qosdrx can be run at the same time. However make sure that qosdrx is not configured to listen to the output from qosd – otherwise you will have a very effective positive feedback loop and an ever increasing list of alarms! In this example configuration qosd publishes to Topic .../OpenOSS/... and qosdrx listens to topic .../OpenOSS1/...
QoSD-configuration.xml	This file sets the events which the qosdaemon code will listen to. By default it listens for the event generated by the vacuum configuration when the alarm list changes; uei.opennms.org/vacuumd/alarmListChanged and for events signifying changes in the node inventory which it uses to update it's local managed object instance and managed object type cache: uei.opennms.org/nodes/assetInfoChanged uei.opennms.org/nodes/nodeAdded uei.opennms.org/nodes/nodeLabelChanged uei.opennms.org/nodes/nodeDeleted
QoSDrxOssBeanRunnerSpringContext.xml	This file is a spring application context to set up the configuration for the qosdrx. Separate threads are set up to connect to each topic. A list of processes to run is in the segment <bean id="OssBeanRunnerList" The definition of each thread is given below. e.g. <bean id="Outstation_OpenOSS1" TODO – full description of this config file
qosdrx.properties	Optional – see note in opennms.conf above

log4j.properties	This file sets up logging for the daemons in openNMS. Two additions are made for QoSD and QoSDrx.
	# QoS daemon server
	log4j.category.OpenOSS.QoSD=DEBUG, QOSD
	...
	# QoSrx daemon server
	log4j.category.OpenOSS.QoSDrx=DEBUG,

	QOSDRX ... Note: In production you should set the logging options in this file to INFO as DEBUG is extremely verbose and will quickly create very large log files and also slow down the QoSD and QoSD deamons significantly..
rmi.policy	Used to allow remote rmi connections to Jboss. (See opennms.conf above). Note that this setting is wide open. You might want to tighten up access security in a production environment.
rrd-configuration.properties	Sets rrd to use jrobin as default instead of rddTool format. (Note that this was needed for the BUTE Qos performance code and not needed for the present qosdaemon)
web.xml	Sets up opennms to use tomcat on port 8081 and also to refresh the alarm list display at 10 second intervals.
/opennms_fault_config	
eventconf.xml events	Configures how opennms treats incoming traps and events. Note in this configuration this is the same as eventconf_NoOpennmsalarms.xml This file references additional events in /ossj_events.xml
eventconf_WithOpennmsalarms.xml	This is default eventconf.xml with additional ossj events referencing /events/ossj_events.xml
eventconf_NoOpennmsalarms.xml	This is default eventconf.xml with additional ossj events but without the opennms generated alarms. This means the alarm list only contains alarms which have been generated as the result of traps
vacuumd-configuration.xml	VERY Important. Contains automation which looks for new alarms in the alarm list and calls the Qosd daemon using the new uei.opennms.org/vacuumd/alarmListChanged event when new alarms are found. Also reconciles raise with clear alarms and deletes cleared and acknowledged alarms
/events	Contains additional event definitions referenced by eventconf.xml
ossj_events.xml	Contains the ossj specific events created for the qosdaemon interface and some test snmp trap definitions to allow the scripts in /testtraps to work
/testtraps	
ossjtesttraps_raise.sh	Simple script to raise an alarm on node 127.0.0.1 Alarm raised is definred in ossj_events.xml <uei>uei.opennms.org/ossjTestEvent/newAlarm/1</uei>

ossjtesttraps_clear.sh	Simple script to raise an alarm on node 127.0.0.1 alarm cleared is defined in ossj_events.xml <uei>uei.opennms.org/ossjTestEvent/clearAlarm/1</uei>
trapgen	Simple utility for generating traps from scripts. Written by http://www.ncomtech.com/trapgen.html Note to run this on fc4 you must also have the library compat-libstdc++-296-2.96-132.fc4.i386.rpm installed
/tomcat55	
server.xml	Sets up tomcat to run on port 8081
tomcat55.conf	Various setting to get tomcat55 running on fedora core 4 with opennms; sets JVM, Tomcat user=root etc
/images	The images directory was used for the BUTE performance management application. Not presently used.
OSS_logo_final.gif	
WEB-INF	
web.xml	

9 Appendix 2: Notes on setting up qosdaemon on Fedora 4

These notes are intended to provide some help in getting OpenNMS runnning with the qosdaemon on Fedora Core 4. They can be adapted to other distributions.

9.1 Installing OpenNMS 1.3.2-SNAPSHOT

The following summarises the instructions for installing OpenNMS Fedora 4 based on http://www.opennms.org/index.php/Building_OpenNMS

preliminary set up of FC4

1. It is useful to have yumex installed as a visual package manager. This makes it easier to pick packages you need for the next steps. (yum install yumex)
2. Ensure subversion is installed on your machine (yum install subversion)
3. Ensure java 1.5 is installed on your machine. This is best done on Fedora core using Jpackage which allows you to easily download and install java packages from the jpackage repo. The following note explains how to use jpackage to install sun java 1.5 on fedora core http://fedoranews.org/mediawiki/index.php/JPackage_Java_for_FC4
4. After installing jpackage, the jpackage repo and java 1.5, install tomcat55 using yumex (do this after jpackage repo is installed so that tomcat gets the right dependencies). Start up tomcat to check it is running and has correct dependencies before proceeding;
   sudo /sbin/service tomcat55 start
   browse to http://localhost:8080 - if tomcat splash is displayed you are in business
   stop tomcat before proceeding; sudo /sbin/service tomcat55 stop
5. Ensure postgres is installed (yum install postgresql postgresql-devel)
6. Ensure rrdtool is installed on your machine (yum instal rrdtool rrdtool-devel)
7. Ensure maven2 is installed on your machine (Note that maven2 is included in the opennms build and opennms will build without it installed. However we need maven2 if you want to build the OSSbeans package separately
   a) download maven 2.0.4 from http://maven.apache.org/
   b) unpack into /usr/local/maven-2.0.4/
   c) place the maven /bin directory on your classpath and set up JAVA_HOME to point to your java 1.5 jre. On Fedora core 4 the following .bashrc login script sets this up when you login;

```
# .bashrc
# User specific aliases and functions
PATH=$PATH:/usr/local/maven/maven-2.0.4/bin
export PATH
export JAVA_HOME=/usr/lib/jvm/java
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
```

```
fi
```

d) after setting up the classpath check maven is installed correctly by typing mvn --version

Installing OpenNMS (this is a standard install of opennms on fc4)

8. cd to directory where you want to download and build opennms source code and issue the command;
   svn co https://svn.sourceforge.net/svnroot/opennms/opennms/trunk opennms
9. cd to the downloaded opennms directory (cd opennms)
10. Type the following commands
    sh build.sh clean   # this should be done before any build to clean out any previous data
    sh build.sh install -Dopennms.home=/opt/OpenNMS package assembly:attached
    This command will use maven to build opennms. On first use it will take some time as it will download all of the dependencies into a local maven repository on your machine ($home/.m2).
    Note: once the dependencies are downloaded, you can do a build off-line using
    sh build.sh -o install -Dopennms.home=/opt/OpenNMS package assembly:attached
11. this builds a zip file called opennms-1.3.2-SNAPSHOT.tar.gz in the target directory. Copy this zip file to /opt/OpenNMS and unpack the contents (need to be root to do this)
12. Follow the instructions in the opennms config guide in /opt/OpenNMS/docs/install.html to set up the postgres database if it is not already set up.
13. Set up the open nms application using following steps;
    a) cd /opt/OpenNMS/bin
    b) make the scripts runnable; sudo chmod 777 *
    c) point openNMS at the jvm; sudo sh runjava -s
    d) upgrade /install the database tables;
       start postgresql if not started (; sudo /sbin/service postgresql start)
       sudo sh install -disU
       (this will upgrade an existing database or install a new one if needed)
    e) link tomcat55 to the opennms web app;
       sudo sh install -y -w /usr/share/tomcat55/conf/Catalina/localhost
14. Start open nms and tomcat and check all is running;
    to start opennms
       cd /opt/OpenNMS /bin directory ; sudo sh opennms -v start
    the consol should show all of daemons starting up correctly
    start tomcat;
       sudo /sbin/service tomcat55 start
    browse to http://localhost:8080/opennms and login using account: admin; password: admin
15. once you are satisfied all is working, shutdown opennms and tomcat before trying to get the qosdaemon to work
    sudo sh opennms -v stop
    sudo /sbin/service tomcat55 stop

9.2 Running the qosd application

The previous steps provides a standard install of opennms of fc4. Having installed opennms the following steps will allow you to test qosd on your system

16. install jboss 4.0.2
    a) jboss provides the jms messaging service. It can be installed on a remote machine but in this example we are installing it locally. Note that the qosdaemon test scripts need some jars from the jboss install to be copied to the /opt/OpenNMS/lib directory. Ideally these should have been included in the OpenNMS build but unfortunately they are not available in any Maven repository.
    b) download jboss 4.0.2 from http://labs.jboss.com/portal/jbossas/download
    (jboss4.0.2.zip)
    c) unpack jboss4.0.2.zip into /opt/
    d) create a symbolic link from /opt/jboss/ to /opt/jboss2.0.2 ;
    sudo ln -s /opt/jboss-4.0.2 /opt/jboss
    e) For this configuration you need to create a host name 'jbossjmsserver1' pointing to this server. To do this from the kde toolbar open /system settings/ network and select the hosts tab. Select new and add a host with Hostname: jbossjmsserver1 Address 127.0.0.1
17. a simple example configuration and install script to get the qosdaemon running can be found in the OpenNMS contrib directory.
    a) cd /opt/OpenNMS/contrib/qosdaemon/qos_example_configuration
    b) A script is provided to move all of the configuration files into the appropriate directories and get you started.
    WARNING: Note - this configuration is provided as an example and will overwrite the opennms default configuration or any other configuration you have installed. It will also change the tomcat55 configuration so that it runs on port 8081 to allow jboss to run on port 8080. (If you will not be running jboss on the local machine the tomcat configuration can be omitted). You can easily adapt this configuration to work with your local configuration but you will need to merge the configuration files appropriately.
    c) To install the qosdeamon configuration ;
    sudo sh opennms_1_3_2_example_deploy_1dot0.sh
18. Test that the OSS/J test clients work with jboss
    a) open a terminal window and move to the Jboss bin directory
    cd /opt/jboss/bin
    b) run the newly installed startup script;
    sudo sh openoss_qos_jboss_start.sh
    You should see the jboss consol log. If all is well Jboss will start up .
    c) Leave this window open as Jboss will stop if you close it. (You can run jboss as a daemon but this is not covered here). To stop jboss properly type control-c in this window.
    d) To see if Jboss is working open a new terminal and try;
    telnet jbossjmsserver 1099
    You should see something like;

```
Trying 192.168.2.4...
Connected to jbossjmsserver1 (192.168.2.4).
Escape character is '^]'.
.srjava.rmi.MarshalledObject|?>IhashlocBytest[BobjBytesq~xp?Fur[Txp&?
http://bitterne:8083/q~q~uq~??rs
org.jnp.server.NamingServer_Stubxr?java.rmi.server.RemoteStub????xrjava.rmi.server.RemoteObject?
a3xpw:
Unicast?
Connection closed by foreign host.
```

e) Use the client test program to connect to jboss
cd /opt/OpenNMS/contrib
sh opennms_IF.sh -xreceive1
You should see something like;

```
***starting sentinal interface program***
***Option: receive - testing OSS/J connection only***
Initialise Session:
Client Properties File Loaded
Using JNP: jnp://jbossjmsserver1:1099
java.naming.provider.url= jnp://jbossjmsserver1:1099
java.naming.factory.initial= org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs= org.jboss.naming
Initial context created
Trying to connect to AlarmMonitorBean
Connecting to AlarmMonitorBean:System/OpenOSS/ApplicationType/AlarmMonitor/Applica
Obtained home, and created Session
Trying to connect to message queues
Topic Connection Factory:System/OpenOSS/ApplicationType/AlarmMonitor/Application/1
Event Topic    :System/OpenOSS/ApplicationType/AlarmMonitor/Application/1-0;0-0;Op
XVT Event Topic:System/OpenOSS/ApplicationType/AlarmMonitor/Application/1-0;0-0;Op
XML Message Queue:System/OpenOSS/ApplicationType/AlarmMonitor/Application/1-0;0-0;
Session Initialised
Subscribing to OSS/J XVT jms session:
Subscribed - Waiting for events ;
Waiting: Event Counts:- ListRebuilt:0   NewAlrmEvt:0    ClrAlrmEvt:0    AlrmCngEvt  ObjMsg:0
TxtMsg:0
***Waiting for OSS/J XVT XML JMS text message event***
```

When OpenNMS starts up you will see the client displaying events and he Event Counts will go up as each event arrives.
To stop the client use Control-c
To see other options for the client try; sh opennms_IF.sh -help
Leave the client running while you start up OpenNMS

19. Test OpenNMS
   a) Note the alarm table has been extended to accommodated OSS/J alarm fields. After an upgrade to opennms and before using the qosdeamon, you should delete any old alarms in the alarm table otherwise QosD may fail at start up.

You can use the postgres gui application pgadmIII to provide an sql terminal to look at the alarm table. Once you have logged into the OpenNMS database you can use;
delete from alarms; # to delete alarms in table
select * from alarm; # to view all of the alarms in the table.

b) When OpenNMS starts up for the first time it does not have node data in it's database. For the following tests to work it is necessary for OpenNMS to at least have it's own node in the database (127.0.0.1 – localhost). If the trap generating scriptr tests are run before this is created, the QoSD may fail the next time it is started as it does not have a default node to reference. (This can be resolved by clearing the OpenNMS alarm table of old alarms and ensuring that localhost exists).
OpenNMS should discover localhost but can be forced to add the node using the OpenNMS /Admin/AddInterface web page.

c) start opennms (/opt/OpenNMS/bin/ sudo sh opennms.-v start). After a short time you should see the following; QoSD and QoSDrx are the two daemons for the qosddeamon interface.

```
OpenNMS.Eventd          : running
OpenNMS.Trapd           : running
OpenNMS.Dhcpd           : running
OpenNMS.Actiond         : running
OpenNMS.Capsd           : running
OpenNMS.Notifd          : running
OpenNMS.Scriptd         : running
OpenNMS.Rtcd            : running
OpenNMS.Pollerd         : running
OpenNMS.Collectd        : running
OpenNMS.Threshd         : running
OpenNMS.Discovery       : running
OpenNMS.Vacuumd         : running
OpenNMS.EventTranslator: running
OpenNMS.PassiveStatusd : running
OpenNMS.QoSD            : running
OpenNMS.QoSDrx          : running
```

Note - the logs for QoSD and QoSDrx are in the /logs library. By default the logging setting is very very verbose. To reduce the log output edit before starting opennms the /opt/OpenNMS/etc/log4j.properties change the lines from DEBUG to INFO;
QoSrx daemon server
log4j.category.OpenOSS.QoSDrx=DEBUG, QOSDRX
QoSrx daemon server
log4j.category.OpenOSS.QoSDrx=DEBUG, QOSDRX

d) Look at the running opennms_IF.sh client. You should see it has received at least an 'AlarmListRebuilt event and possibly others if there were already alarms in the OpenNMS alarms list

e) Open a browser and look at the OpenNMS alarm list at http://localhost:8081/opennms/alarm/

f) Try injecting test alarm raise and alarm clear traps using the following scripts.
cd /opt/OpenNMS/contrib/qosdaemon/qos_example_configuration/testtraps
sudo sh ossjtesttraps_raise.sh
sudo sh ossjtesttraps_clear.sh
You should see alarms added and deleted from the web alarm list as it refreshes
You should also see XML alarm events on the opennms_IF.sh terminal.

20. To test that the qosdrx works you need to install another OpenNMS on a seperate server and change its configuration so that it injects alarms onto the opennms running qosdrx

a) Install and test OpenNMS as above

b) edit /opt/OpenNMS/etc/service-configuration.xml and comment out the section which starts up qosdrx. The remote OpenNMS should not be running qosdra as it will get into a feedback loop.

```
<!--
	<service>
		<name>OpenNMS:Name=QoSDrx</name>
		<class-
name>org.openoss.opennms.spring.qosdrx.jmx.QoSDrx</class-name>
		<invoke at="start" pass="0" method="init"/>
		<invoke at="start" pass="1" method="start"/>
		<invoke at="status" pass="0" method="status"/>
		<invoke at="stop" pass="0" method="stop"/>
	</service>
-->
```

c) Edit /opt/OpenNMS/etc/qosd.properties. Change 'OpenOSS' to OpenOSS1 as below. The OpenNMS qosd will now send alarms to the OpenOSS1 topic;

```
org.openoss.opennms.spring.qosd.naming.provider=jnp://jbossjmsserver1:1099
org.openoss.opennms.spring.qosd.naming.contextfactory=org.jnp.interfaces.Na
mingContextFactory
org.openoss.opennms.spring.qosd.naming.pkg=org.jboss.naming

org.openoss.opennms.spring.qosd.jvthome=System/OpenOSS1/ApplicationType
/AlarmMonitor/Application/1-0;0-
0;OpenNMS_OpenOSS_AM/Comp/JVTHome

org.openoss.opennms.spring.qosd.jms.topicconnectionfactory=System/OpenOS
S1/ApplicationType/AlarmMonitor/Application/1-0;0-
0;OpenNMS_OpenOSS_AM/Comp/TopicConnectionFactory
org.openoss.opennms.spring.qosd.jms.topic=System/OpenOSS1/ApplicationTyp
e/AlarmMonitor/Application/1-0;0-
0;OpenNMS_OpenOSS_AM/Comp/JVTEventTopic

org.openoss.qosd.jms.xvttopic=System/OpenOSS1/ApplicationType/AlarmMon
itor/Application/1-0;0-0;OpenNMS_OpenOSS_AM/Comp/XVTEventTopic

org.openoss.qosd.jms.queueconnectionfactory=System/OpenOSS1/ApplicationT
ype/AlarmMonitor/Application/1-0;0-
0;OpenNMS_OpenOSS_AM/Comp/QueueConnectionFactory
org.openoss.qosd.jms.messagequeue=System/OpenOSS1/ApplicationType/Alar
mMonitor/Application/1-0;0-
0;OpenNMS_OpenOSS_AM/Comp/MessageQueue

org.openoss.opennms.spring.qosd.url=http://OpenOSS1:8081/opennms
```

d) To test if this OpenNMS is sending messages correctly open a new terminal and start a client to listen to its queue. For convenience this can be done using the script
cd /opt/OpenNMS/contrib
sh opennms_IFOpenOSS1.sh -xreceive1
e) Start the second opennms you should see it start up with only the qosd daemon running . You should also see the opennms_IFOpenOSS1.sh terminal register an alarm list rebuilt event
f) Try injecting alarms using the trap scripts above into the second opennms. You should see the first opennms alarm list match the changing remote opennms alarm list.

21. Adding managedobjectinstance, and managedobjectclass data to alarms
So far we have sent out alarms with no unique object reference. In normal operation the OSS/J interface should have a unique object references for each managed object. This reference will be reflected in the OSS/J XML message fields 'managed object class' and 'managed object instance'. New fields have been added to the OpenNMS assets table to allow each node in OpenNMS to thus be uniquely identified. In normal operation every

node in the table should have a unique match for the concatenation of these fields . The assets table is referenced by nodeid (which is the OpenNMS internal key for all nodes in the nodes table). The OSS/J unique reference is given by the fields managedobjectinstance and managedobjecttype (= OSS/J managed object class) . To see the entries in the database use;

```
select nodeid, managedobjectinstance, managedobjecttype
from assets;
```

Presently the only way to insert this data is to modify the Assets table directly in the database using SQL. (In a future release the OpenNMS Import commandshould be update to allow the assets table to be imported using XML scripts and the OpenNMS UI will allow users to directly manipulate these fields). For now you will manually have to add the assets data for each node using pgsqladminIII. Using the following process;

1. Follow the documented procedures to add nodes to OpenNMS or allow OpenNMS to discover the nodes in your network. Each node will be labeled with it's IP address by default.
2. For each node referenced in the Nodes table , ensure that it also has a reference in the assets table. (put in some dummy data using the assets UI)
3. For each node in the Assets table use pgsqladmin to add the managedobjectinstance, managedobjecttype data. This can be scripted using the following type of sql command;
   ```
   update assets set
     managedobjecttype='BroadcastEquipment',
     managedobjectinstance='Caldbeck-BX-AIS-TX'
   where nodeid = (select nodeid from node where
       node.nodeType <>'D' AND nodelabel='10.100.0.1');
   ```
4. Having added the assets managedobjectinstance, and managedobjectclass data this will show up in every OSS/J alarm issued by OpenNMS.

This completes the installation and testing of the Qos interface. See the sections above for more information on how to configure the interface in you environment.