Integrating CIM/WBEM with the Java Enterprise Model

Kenneth Carey & Fergus O' Reilly
Adaptive Wireless Systems Group
Department of Electronic Engineering
Cork Institute of Technology, Cork,
IRELAND.
Tel. +353 21 4326100 Fax: +353 21 4326625
Email: kcarey@cit.ie, foreilly@cit.ie

Abstract

Enterprise level system development, integrating a broad range of architectures and legacy systems, has received significant attention over the last few years. Integrated chain management, which allows companies to see their full manufacturing, inventory and sales channels, has driven much of this work. Delivering quality IP based wireless access is forcing similar integration in 2.5G(GPRS) and 3G(UMTS) architectures. Networks are now combining IP core backbones, legacy databases and switching technology. This integration introduces challenges for today's network administrators, particularly in the area of Alarm Management. This paper will discuss the experiences of developing a multi-tiered Heterogeneous Alarm Management System based on the Java2 Enterprise Edition, with CIM/WBEM as the interface to the legacy and mixed protocol architectures. We shall also discuss the simulation of a SNMP/CMIP based heterogeneous network environment.

1. Introduction

Network management today is becoming increasingly difficult because of the need to manage multiple devices each using different network technologies. A particular problem facing network administrators is the detection and tracking of alarms sent from managed devices. In order to observe these alarms telecommunications network management systems must incorporate support for the various network protocols and proprietary systems operating within the heterogeneous environment. Such systems must also be extendable as systems can quickly become outdated due to the introduction of updated or newly designed network protocols and equipment.

Current telecom network management systems are finding it difficult to meet these needs. Incumbents have large legacy systems, which have been built piecemeal over many years. A typical such system has the characteristics of a mature software discipline, such as reference models, communications protocols, design patterns and reference architectures. Consequently the task of repeatedly updating the system to support updated network devices and protocols is arduous and time consuming.

Two popular protocols operating within heterogeneous networks are the Common Management Information Protocol (CMIP) and the Simple Network Management Protocol (SNMP). SNMP is ubiquitous in the IP world, while traditionally, CMIP is responsible for managing telecommunications devices. The convergence of voice and data networks has however, resulted in IP, and subsequently SNMP, becoming popular throughout next generation telecom carrier networks. The Web Based Enterprise Management (WBEM) initiative is a management architecture proposed under the umbrella of the Distributed Management Task Force (DMTF). The architecture is scalable and distributed and designed in such a way so that it should be compatible with all major existing management protocols and proprietary systems.

As illustrated in Figure 1, for a GPRS/UMTS Network, the aim of this research project is to build and test an extendable, scalable, platform independent, telecom management architecture. This will be achieved through the integration of the J2EE architecture and the WBEM initiative. The management framework will be capable of detecting and processing alarms generated within a Heterogeneous Management Network.

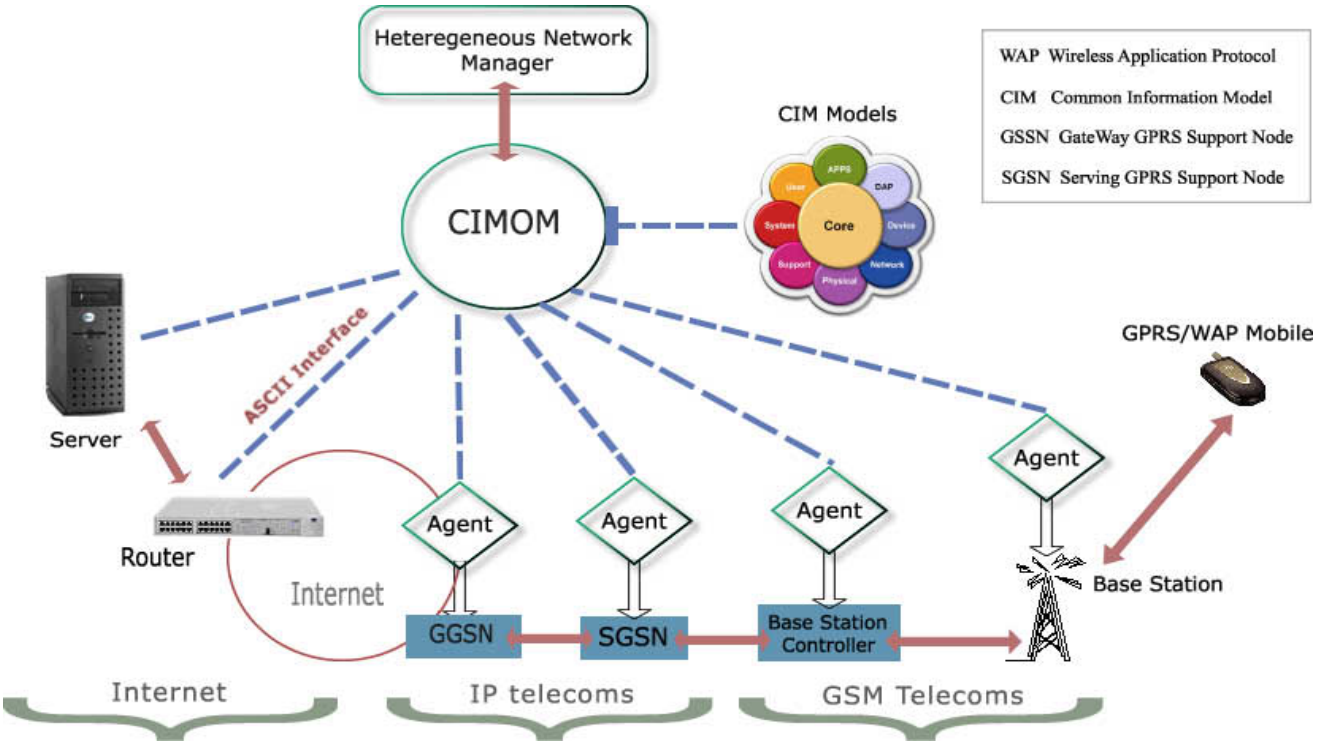

Figure 1. Heterogeneous Manager for GPRS/UMTS network

2. Web Based Enterprise Management

For this research project we will use the WBEM initiative, which is characterised by a distributed architecture, an object oriented information model, and the integration of network, systems, application, service, and policy-based management. For our uses it will provide:

- A data description (CIM)
- On-the-wire encoding (xml/CIM)
- Operations to manipulate data (HTTP)

CIM is a mechanism for modelling managed resources and representing those models in the Managed Object Format (MOF) language[1]. Similar to the Management Information Base (MIB) used to describe SNMP managed resources, it is a textual format for describing management information. Using the CIM and MOF, the components that make up a managed resource or a network of resources can be modelled and viewed in a way similar to that used in an object-oriented software design process.

The DMTF has proposed a layered class hierarchy to help represent manageable elements from a variety of areas. Each standard management area, such as storage or applications, can be represented in a CIM Schema, which inherits from a Core Schema (see Figure 1). Once a CIM model and MOF definition are complete, the package is imported into a CIM Object Manager (CIMOM). The CIMOM provides a central repository where WBEM clients in a network can go gather information about managed resources within the system.

For applications to interoperate with one another it must be possible to represent actual management data in a standard way. Extensible Markup Language (XML) is a markup language for representing information in a standard format. An XML schema is a grammar that describes the structure of an XML document. In the WBEM system an XML schema is used to describe the CIM, and both CIM classes and instances are valid XML documents for that schema [2].

To allow WBEM to operate in an open, standardized manner, the DMTF has defined a mapping of CIM operations onto the Hyper Text Transfer Protocol (HTTP). HTTP, running on top of TCP/IP, can be used for many tasks through the extension of its request methods, error codes and headers. The standard encoding of CIM to XML favours completeness over conciseness and consequently performance is compromised. This target of generality as opposed to efficiency is a possible shortcoming.

3. Selecting the WBEM/CIM Layer

Individual companies provide their own implementation of WBEM, each of which must conform to the specifications, accept the standardized schemas and all associated CIMOMs must be accessible via the standard HTTP operations. Java based implementations are provided by Sun and the Storage Networking Industry Association, in the form of the Sun WBEMServices[3] and the SNIA CIMOM[4] respectfully. While both versions offer excellent client programming APIs, the Sun version has been chosen for the development/testing of our WBEM system because of its excellent documentation. Figure 2 outlines the position of the CIM/WBEM implementation within our Enterprise Framework. The purpose of the CIM/WBEM layer is to provide an interface to the multi-protocol architectures present within a typical Heterogeneous Telecom Network.

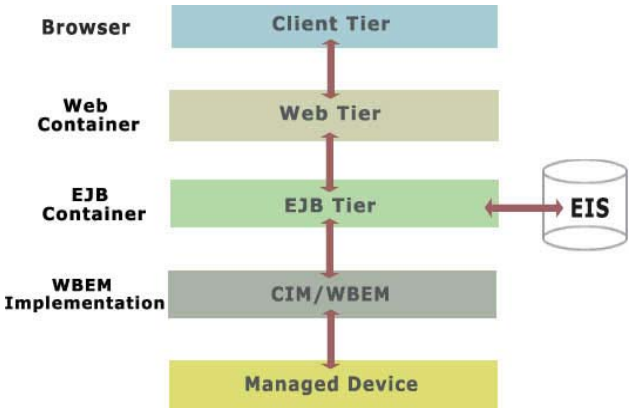

Figure 2. Alarm Management Framework

WBEM implementations use a client – server model as illustrated in Figure 3. The client applications generate requests against a CIMOM using the WBEM client application programming interfaces (API). The CIMOM may interpret these requests using information in its internal store or support SNMP, CMIP and proprietary protocols through the use of Providers. These are Java translator classes responsible for acting as an interface between the CIMOM and the real managed object. The provider uses HTTP to interact with the CIMOM above and whatever protocol is appropriate to interact with the managed object below. Consequently for us, the provider classes are responsible for detecting events and incorporate SNMP or CMIP functionality.

We therefore designed SNMP and CMIP management classes that could be inherited by the various Provider classes/modules we developed. The respective classes were implemented using the AdventNet SNMPAPI[5] and the DynamicTMN GDMO Manager Java API[6].

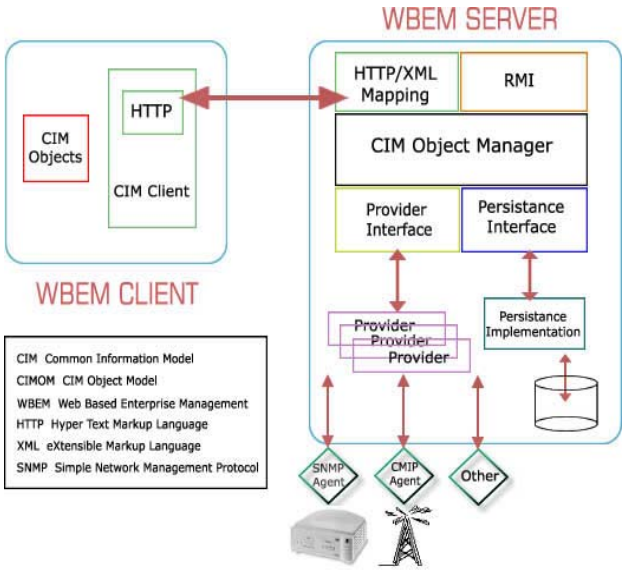

Figure 3. WBEM Client-Server Architecture

Figure 4 shows the operation of the manager classes. They are based on a multithreaded design, permitting them to perform management operations, while simultaneously listening for, and processing alarm/events.

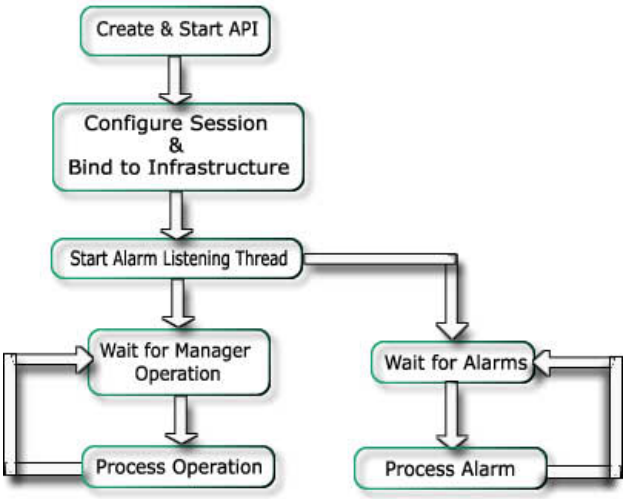

Figure 4. Manager Class

3.1 Alarm/Event Processing in CIM/WBEM

An event is a real world occurrence and is typically assumed to be a change in the state of the environment or a record of the behaviour of some component of the environment. In the case of SNMP enabled devices, Traps are issued to notify a management application of an event. For example, a cold-start trap is issued by an SNMP enabled device when it has started.

There are two types of events in CIM, life cycle events and process events. A life cycle event is a built-in CIM event that occurs in response to a change in data in which a class is created, modified or deleted etc. A Process event is a user-defined event that is not described by a life cycle event. The CIM Indication schema is used to communicate occurrences of events in CIM. A CIM indication is an object that communicates the occurrence of an event and CIM indications are published for subscription[7]. Types of indications (representing different types of events) are denoted by CIM Indication subclasses. These include InstIndication and ClassIndication subclasses for modelling CIM life cycle events and ProcessIndication classes for alert notifications associated with objects that do not correspond to a simple life cycle event; like SNMP traps and CMIP events. Subclasses of ProcessIndication include SNMPTrapIndication and AlertIndication, which map an SNMP Trap and CMIP event to CIM. A client application wishing to receive notifications of events from the CIMOM subscribes for the correct type of Indication.

We developed providers capable of receiving SNMP traps and CMIP events. When, for example, a trap is received by a provider, the provider creates an instance of SNMPTrapIndication. The SNMPTrapIndication class contains properties that describe an SNMP trap, such as Enterprise, AgentAddress and Specific Type. These properties are assigned the appropriate values by extracting the information from the received trap. The indication is delivered to the CIMOM, which routes it to client applications with a subscription to that particular indication class. A similar operation is performed when a CMIP enabled provider detects a CMIP event. In this case however the provider generates an instance of AlertIndication.

4. Simulating a Heterogeneous Network

Before work on the multi-tiered enterprise application could begin, it was first necessary to build a heterogeneous network on which the functionality of the WBEM providers could be tested and evaluated. As discussed earlier, a heterogeneous telecom network typically consists of SNMP and CMIP enabled devices. To allow total flexibility in describing a large network we decided to simulate the heterogeneous environment through the development of Java based SNMP and CMIP agents. The Java Dynamic Management Toolkit (JDMK)[8], AdventNet Agent Toolkit[9] and the Monfox DynamicTMN Agent Toolkit[6] are rapid prototyping and development tools used for building cross platform Java based SNMP and CMIP agents and proprietary protocols.

4.1 Java Based SNMP/CMIP Agents

The Java agents model real network devices and are similar to agents found on SNMP and CMIP enabled devices. They comprise several classes, which are generated from an SNMP MIB or a CMIP GDMO[10] file. These are responsible for handling manager requests, sending alarms and representing the MIB or GDMO information. The agents store statistics and information in a database.

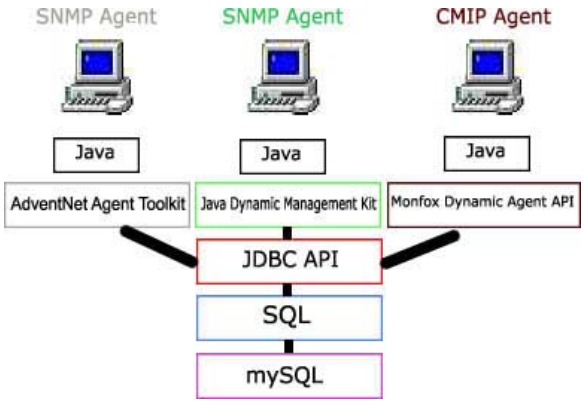

Figure 5. Agent Development Technologies

Figure 5 outlines the technologies working in unison in the simulated Java agents.

4.2 Network Failure Simulation

In a large complicated network, the failure of a single piece of equipment may cause all network elements that depend on that piece of equipment to malfunction. Each malfunction may generate an event notification, leading to an event storm of several hundred notifications, all of which are the result of the same underlying problem.

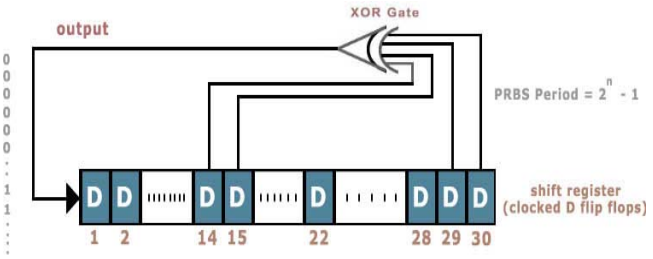

Figure 6. Thirty Stage LFSR.

Therefore a requirement for our simulated heterogeneous network is to generate random event storms. The event storm scenario is created through the use of Pseudo Random Binary Sequences (PRBS).

Pseudorandom binary sequences are a close approximation to white noise and are widely used in

statistical testing[11]. An n stage linear feedback shift register (LFSR), as per Figure 6, governed by the primitive characteristic polynomial $\Phi(s)$, produces a PRBS according to the formulae[12]

$$a_i = \sum_{k=1}^{m} \beta_k a_{i-k}, i \geq 0 \quad (1)$$

where $\beta_k \varepsilon \{0,1\}$ are coefficients determining the feedback taps $fb(x)$ and a_i is the LFSR output. A PRBS is a binary waveform and has two stable states. The number of stages n in a LFSR determines the sequence length L of the PRBS. We have taken two 30 bit primitive polynomials, as per equations (2) and (3), and determined the corresponding feedback tap positions as shown in equations (4) and (5). The LFSR outputs are ANDed together to produce a sequence containing streams of 0s interspersed with bursts of 1s due to the AND operation.

$$s^0 + s^1 + s^{15} + s^{16} + s^{30} \quad mod\ 2 \quad (2)$$

$$s^0 + s^1 + s^{12} + s^{21} + s^{30} \quad mod\ 2 \quad (3)$$

$$x^{30} + x^{29} + x^{15} + x^{14} \quad (4)$$

$$x^{30} + x^{29} + x^{18} + x^{9} \quad (5)$$

Our simulated agents are configured to send events according to this sequence, resulting in periods of inactivity followed by random bursts of events issued over a short period of time as illustrated in Figure 7.

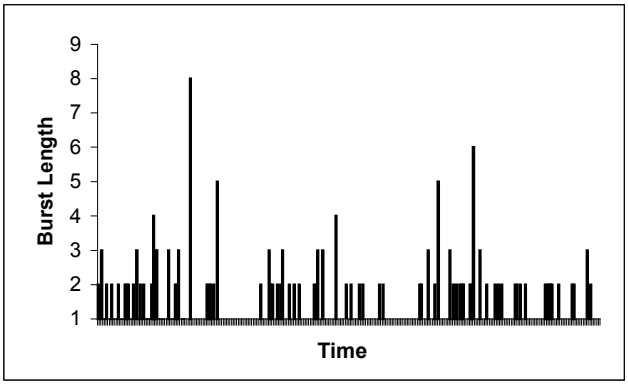

Figure 7. Random Burst Lengths

5. Developing the Multi-Tiered Management Application

To provide for scalability and reliability we have designed a multi-tiered enterprise application based on the Java 2 Enterprise Edition (J2EE) server-side application model and the Enterprise JavaBeans (EJB) component model[13]. The J2EE provides industry-strength scalability; support for existing information systems; a simple flexible security model and allows for seamless integration of the WBEM client API. Figure 8 illustrates the various tiers incorporated into our management application.

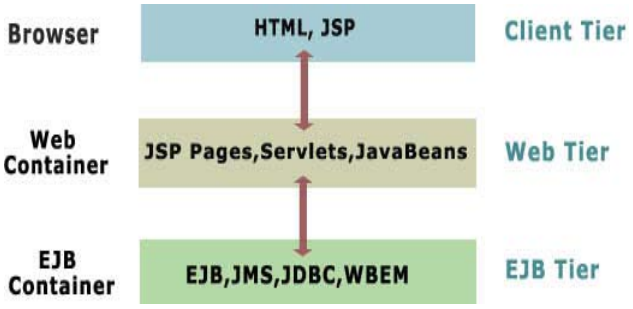

Figure 8. Multi-tier Application

5.1 Tiered Architecture

For the user, the client browser is the application. It must be useful, usable, and responsive. The web-based client we have developed uses HTML and Java Server Pages (JSP)[15]. Being a thin client, it has two main advantages over a thick client application; firstly it allows the network to be managed from any platform capable of running a web-browser, secondly, the resource requirements are much less than that of a thick client manager, so receiving alarm updates on a GPRS enabled handheld device is also a possibility.

Our client browser provides the user with an intuitive user interface, allowing them to view the current state of the heterogeneous network, invoke management operations, create CIM classes and instances, upload Java provider classes, perform alarm monitoring (Figure 9) and view a log of all management operations performed.

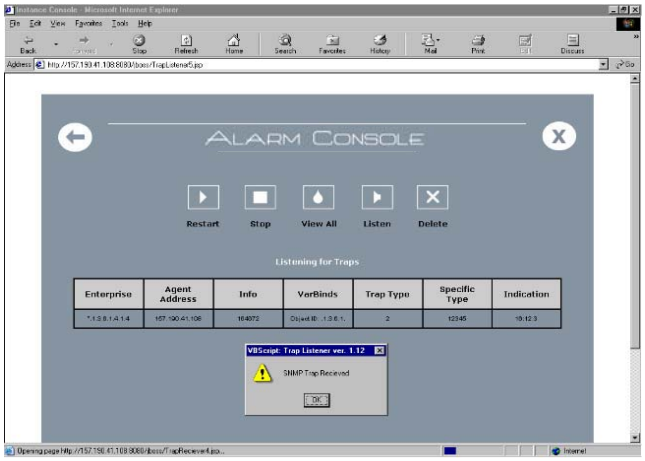

Figure 9. Event Monitoring UI (Client Tier)

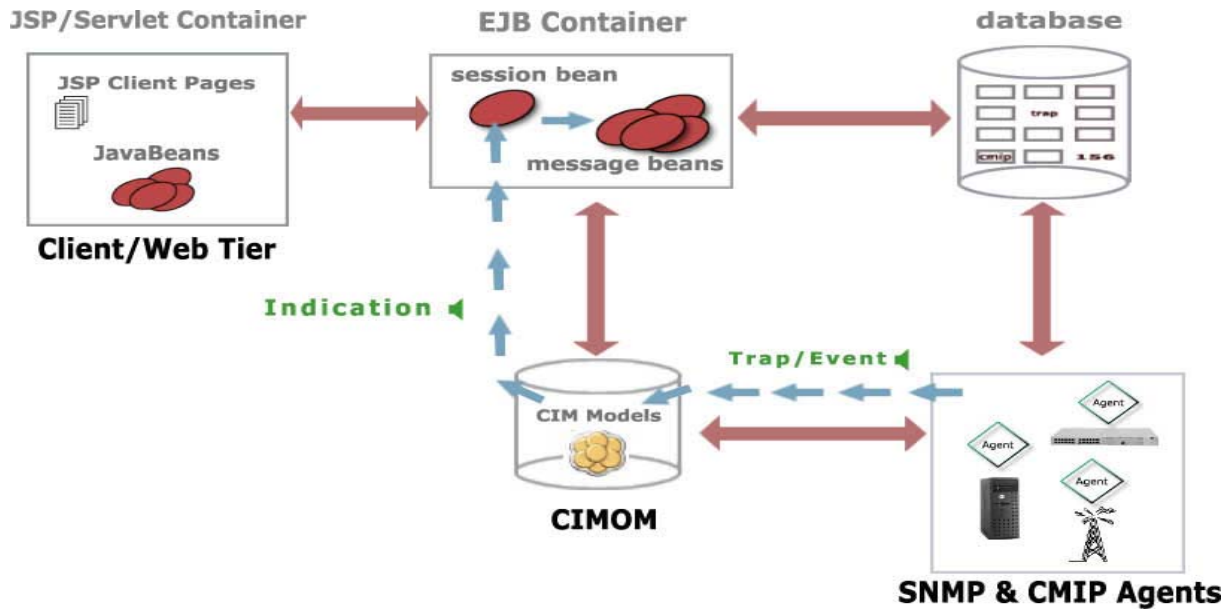

Figure 10. Handling WBEM Indications

The Web-tier is responsible for handling all of the J2EE application's communications with the web client and transmitting data in response to incoming requests. The various web-tier technologies used include JSP, JavaBeans and Java servlets. A JavaBeans component is a Java class that follows certain coding conventions, so it can be used by tools as a component in a larger application [16]. A servlet is a Java class that extends a web server, producing dynamic content in response to requests from the server. The components execute inside a Tomcat JSP server.

In a typical multiple-tier application, the EJB tier hosts any application-specific business logic and provides services such as concurrency control and security. Our server-side components, as Enterprise JavaBeans (EJB), are distributed objects that provide remote services to clients, with the above features. Enterprise beans run in a special environment called an EJB container which manages every aspect of an enterprise bean at run time. There are three types of enterprise beans: session beans, entity beans and message driven beans[14]. We have developed several session type EJBs, responsible for performing operations on the CIMOM. These beans achieve this through the use of the WBEM client API. There are also session EJBs dedicated to alarm management.

5.2 Scalable Management Architecture

The alarm management session bean works in conjunction with a Message System, the Java Message Service (JMS) API[18] and message-driven beans.

Messaging systems provide a way to exchange data asynchronously and the JMS API allows a java application to connect to these systems. Once connected, an application can use the facilities of the underlying enterprise messaging system to create messages and communicate asynchronously with one or more peer applications.

Figure 10 and 11 outline the various application tiers and components involved in the detection and processing of WBEM indications. Referring to Figure 10, an alarm is dispatched from an agent and detected by either the SNMP or CMIP provider. The provider sends an event to the CIMOM, which routes it to the subscribing client, in this case, the EJB session bean. The session bean converts the indication to a JMS Message and places it on a Message System Queue as outlined in Figure 11. Message driven beans retrieve the messages from the queue in parallel. The message contents are processed and the event information is inserted into a database table.

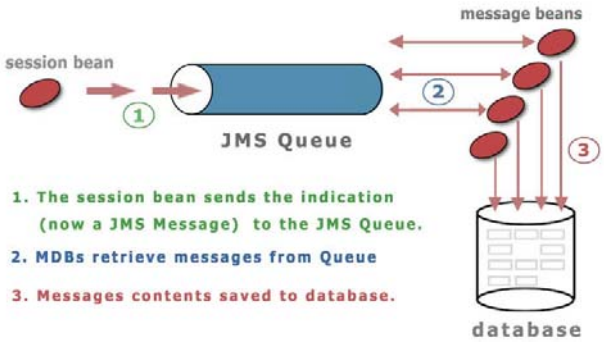

Figure 11. Processing WBEM Indications

At the client tier, a web page monitors the contents of the database table, displaying on screen an alert box and any new additions to the database table since the last poll. The network administrator can view all events received and also delete all alarms/events processed. The database-polling interval can also be dynamically configured from the web page.

The above method of storing and processing is efficient and scalable. The system is initially configured to have a pre-defined number (10) of message driven bean instances created. However if all message beans are busy processing messages and messages are building up on the queue, then the EJB container is made to create another message bean, resulting in a highly scalable mechanism for processing messages. This is confirmed in the results section. The container will continue to instantiate message beans until the queue is empty or a maximum number of beans, specified by the administrator of the system, are instantiated.

6. Results

Table 1 outlines the performance of the architecture, showing its ability to handle bursts of alarms over an interval of 60 seconds. The average number of alarms sent per second was continuously ramped up in order to determine the average burst rate that the system could manage over the interval. The CIMOM was deployed on a Pentium 4 (2.0 GHz, 1024 MB Ram) running Linux Mandrake 8.1. The JBoss EJB container was used to host the various Enterprise Java Beans, while the Tomcat JSP/Servlet container hosts the JSP, HTML and JavaBeans. Both these containers were deployed on a Pentium IV (1.5 GHz, 384 MB Ram) running Windows 2000.

Table 1. Determining Average Burst Rate

Alarms Sent	Time (s)	Ave (s^{-1})	Received (cimom)	Received (EJB)	%lost
10000	60	167	10000	10000	0
10500	60	175	10000	10000	0
11000	60	183	10000	10000	0
11500	60	192	11155	11155	3
12000	60	200	11160	11160	7
12500	60	208	10875	10875	13
13000	60	217	10660	10660	18
13500	60	225	10395	10395	23
14000	60	233	10080	10080	28
14500	60	242	9280	9280	36

From the results above we can see that the system begins to drop alarms when the average burst rate exceeds 183 per second. From our examination of the system we determined that the reason for the drop off in performance is due to the provider classes responsible for detecting the SNMP and CMIP alarms. Not all network events issued by the SNMP and CMIP agents could be detected by the management classes within the respective providers. Figure 12 below outlines how the performance of the systems drops linearly as the average burst rate exceeds 183 alarms per second.

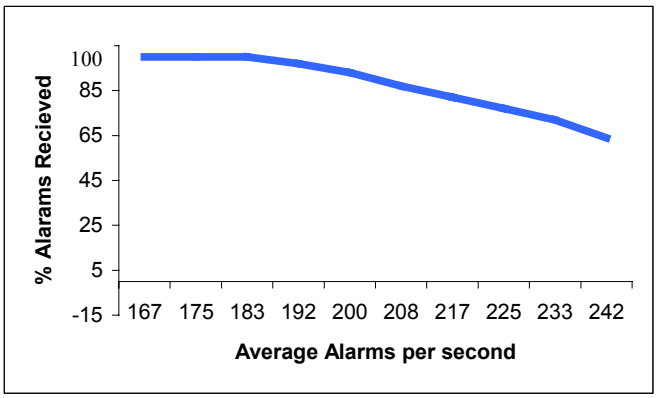

Figure 12. System Performance

The performance of the SNMP and CMIP classes is OS and PC configuration dependant. Therefore performance could be improved by increasing the processor speed or physical memory available. Another possible solution could be achieved through the distribution of alarms to multiple CIMOMs, running on separate machines, each responsible for detecting a certain type of alarm.

The maximum number of alarms that the system could detect and process over an interval of 1 second was also determined. Table 2 outlines the results obtained. It is shown that the system can handle a maximum of 700 alarms in 1 second.

Table 2. Maximum Burst Rate in 1 s

# Alarms Sent	Received	% dropped
550	550	0
600	600	0
650	650	0
700	700	0
750	724	3.46

7. Conclusion

In this paper we have discussed the building of a multi-tiered, platform independent Heterogeneous Alarm Management System based on J2EE architecture. From the outset we set out to design an extendable and scalable system, capable of handling alarms generated by SNMP, CMIP and proprietary agents. This was achieved through the integration of the Web Based Enterprise Management (WBEM) architecture and Enterprise Java Beans (EJB). Results have confirmed this, showing how the system can successfully handle up to 183 alarms per second over an interval of 1 minute. The distributed nature of the architecture is also evident, with the CIMOM server and EJB/Web containers running on separation machines during testing. The system's scalability is achieved through the use of an EJB container, session enterprise beans and message driven beans. Extensibility is incorporated through the use of the WBEM provider architecture, which allows for the seamless integration of network management protocols.

References

[1] Winston Bumpus, Andrea Westerinen – Common Information Model, Wiley, Oct 2000

[2] K. Carey & F. O Reilly – Tools and Toolkits for Heterogeneous Networks, DMTF Developers Conference 2002.

[3] Sun WBEM SDK - http://wbemservices.sourceforge.net/

[4] SNIA CIMOM – http://www.snia.org

[5] AdventNet SNMP API - http://www.adventnet.com/products/snmp/index.html

[6] Monfox DynamicTMN toolkit - http://www.monfox.com/protocol.htm

[7] CIM Event Model White Paper, Mar 2002 http://www.dmtf.org/download/whitepaper/Events

[8] Java Dynamic Management Kit - http://sun.java.com/jdmk4.2

[9] AdventNet SNMP Agent Toolkit - http://www.adventnet.com/products/javaagent/

[10] William Stallings – SNMP, SNMPv2, and CMIP, Addison-Wesley, Oct 1994

[11] S. Rae & R. Guinee, Uniform Random Number Generation Using Pseudorandom Binary Sequences, Cork Institute of Technology, 2001.

[12] V.N. Yarmolik and S.N. Demidenko, Generation and Application of Pseudorandom sequences for Random Testing, John Wiley and Sons.

[13] Java 2 Enterprise Edition – Specification http://java.sun.com/j2ee/docs.html

[14] Enterprise JavaBeans™ Technology Downloads & Specifications - http://java.sun.com/products/ejb/docs.html

[15] Hans Bergsten – Java Server Pages, O'Reilly, Jan 2001

[16] The JavaBeans API Specification - http://java.sun.com/products/javabeans/docs/spec.html

[17] Ed Roman, Mastering Enterprise JavaBeans, Wiley, Sept 2002.

[18] Hapner, Burridge, Sharma, Java™ Message Service API Tutorial and Reference: Messaging for the J2EE™ Platform, Addison-Wesley, Feb 2002