STATE OF COLORADO

Intelligent Transportation Systems
700 Kipling Street, Suite 2500
Lakewood, Colorado 80215
Phone (303) 512-5834
FAX (303) 239-0848

Colorado Department of Transportation

CTMS/CTIS INTEGRATION
Contract Routing No. 04 HAA 0063

Software Architecture Assessment Document

Date: 12-May-2004

Version 1.2

This particular document identifies the various technologies considered for the CTMS/CTIS architecture and the final choice.

Approved By

Robert Wycoff
CDOT ITS Office

Signature: ___________________________

Date: _______________________________

John Williams
CDOT ITS Office

Signature: ___________________________

Date: _______________________________

Prepared By:

EnRoute Traffic Systems, Inc.

Revision History

Date	Version	Description	Author
09-Jul-2003	0.1	Introduction to the Architecture elements	Sachin Saindane Pawan Kharbanda
16-Jul-2003	0.2	Significant Updates	Sachin Saindane Pawan Kharbanda
28-Jul-2003	0.3	Significant Updates	Sachin Saindane Pawan Kharbanda
01-Aug-2003	0.4	Additions, MDB, Recommendation section,	Sachin Saindane Pawan Kharbanda
11-Aug-2003	0.5	Section 3 added, all other sections modified	Sachin Saindane Pawan Kharbanda
12-Aug-2003	0.6	Portion on Datex added	Sachin Saindane Pawan Kharbanda
20-Oct-2003	0.7	Findings from new prototype development added. Appendix changed	Sachin Saindane Pawan Kharbanda
28-Oct-2003	0.8	Sections 3,4,5,7 updated	Sachin Saindane Pawan Kharbanda
06-Nov-2003	0.9	Updates to Metrics, Scoring sheet and model 2.	Sachin Saindane Pawan Kharbanda
10-Nov-2003	1.0	Final Version	Sachin Saindane Pawan Kharbanda
12-Nov-03	1.1	Added Findings, Executive Summary, front page introduction, minor formatting.	Sachin Saindane
19-July-04	1.2	Updates with JBoss findings.	Sachin Saindane

Table of Contents

1. INTRODUCTION

The Software Architecture Assessment (SAA) document introduces the various technologies that were chosen and evaluated in developing the overall CTMS/CTIS architecture. This document also describes the four architectural alternatives that were prototyped as a part of the architecture assessment. Lastly it evaluates the architectures based on well-defined metrics and concludes with a recommendation.

1.1 Purpose

The purpose of this document is:

- To serve as a prerequisite reading before any architecture related demonstrations or discussions.
- To supplement the technical information provided in the Software Architecture Document.
- To capture and convey the significant architectural issues and decisions which have been made during meetings and discussions.
- To describe the evaluation process used to arrive at the final CTMS/CTIS architecture.
- To describe the final choice of architecture after evaluation.

1.2 Scope

The scope of this document is the CTMS/CTIS system.

1.3 Audience

This document describes the CTMS/CTIS architecture from various perspectives. The initial sections should help ITS professionals understand the technologies that were considered, the technologies that were evaluated and the reasons for doing so. The technical overview section is aimed at Software Professionals and assumes prior participation in architecture related discussions or demonstrations at TOC. It also assumes knowledge of software design, development, knowledge of Object Oriented Paradigm and basic object fundamentals (object life cycle, interfaces, classes and object instances).

1.4 References

1. The CTMS/CTIS Document Index.
2. Several definitions have been used from The Software Engineering Institute website:

   http://www.sei.cmu.edu/str/indexes/glossary/
3. Java technology related information found on Java homepage at:

   http://java.sun.com
4. Description on Quality of Services (QoS) archived in VSS at:
   $/ATMS_ATIS/Architecture Assessment/Architecture Assessment/QOS.doc
5. Various JMS related presentations archived in VSS at:
   $/ATMS_ATIS/Tech Docs/JMS/JMS INTRODUCTION/JMS – presentation.ppt
6. Documents created for the MDOT CHART system, particularly, System Architecture Document

   http://www.chart.state.md.us/readingroom/readingroom.asp
7. Choosing a Center-to-Center Communications Protocol: An Overview of DATEX, CORBA and XML by U.S. Department of Transportation.
8. The ITS glossary for CTMS/CTIS

Important Note: The architecture assessment was conducted using Borland Enterprise Server (BES) and evaluation copies of JMS brokers. A post evaluation decision was made to use JBoss and Jboss MQ instead of BES. The diagrams and explanations in this document will still use Borland Enterprise Server. For explanation on use of JBoss and JBoss MQ over Borland Enterprise Server please refer to section 10.4.1 and 10.4.2 addendum.

2. EXECUTIVE SUMMARY

2.1 Introduction

This document describes the various technologies considered for the CTMS/CTIS software architecture, identifies known architectural elements, and describes the alternatives being considered where there are questions or unknowns.

An early evaluation of the proposed CTMS/CTIS architecture requirements revealed an area of potential risk in the communications architecture component of the system. To mitigate this risk and better understand the issues, several architectural prototypes were constructed. The basic choice was between a CORBA based architecture, or a J2EE-XML based architecture for communication.

The project has chosen to use a J2EE-XML communication architecture.

2.2 Background

The CTMS/CTIS project architecture is based upon open standards as well as existing production systems architectures. Most of the CTMS/CTIS architectural elements are standard, such as J2EE, Java Swing, EJB, and Oracle. The communication architecture component that supports interaction between the CTMS/CTIS and field deployed devices (C2F) had the most uncertainty. CORBA and J2EE both offered potential solutions, so prototypes were developed to better understand each of the alternatives.

The prototypes were evaluated based upon CDOT-ITS architectural goals:

General Goals:

- *Scalability:* To evaluate the potential of the system to grow and support future needs.
- *Maintainability:* To analyze the maintainability of the system in the future with respect to technology evolution and the introduction of new features and requirements.
- *Availability:* To evaluate the availability of support products involved in the overall architecture and the availability of programmers required to implement the architecture.
- *Prevalence:* To check the prevalence of the technology choices in US and other state held DOT's.
- *Ease of Implementation:* To study the ease of implementing the architecture.
- *Standards:* To study the support available for the standards.

Project Specific Goals:

- Use three tier (or n-tier) application architecture.
- Manage all application communication using Model View Controller (MVC) pattern.
- Place as much of the application as possible in the Application Server Container.
- Let the application layer manage communication between architectural components. (This is related to keeping as much of the application in the container as possible.)

Benefits:

- Application server based architecture will ensure future portability to other EJB Containers.
- Current CDOT staff is proficient with J2EE and EJB.
- Application Server based architecture will ensure ease of future maintenance since application logic and business logic will be kept in one place, instead of being distributed to various architectural components.

2.3 Conclusion

Based upon the results of the Software Architecture Assessment, the communications architecture of the CTMS/CTIS system will be J2EE-XML based.

3. INTRODUCTION TO ARCHITECTURAL ELEMENTS

This section describes the various technologies being considered while developing the CTMS/CTIS core architecture. The technologies have been classified into two sections. The **Existing Technologies** section lists and describes all technologies currently used at CDOT-TOC. The **Technologies Under Evaluation** section lists and describes all technologies that may become a part of the CTMS/CTIS architecture, however need to be evaluated.

3.1 Existing Technologies

Following is a list of technologies that currently exist at CDOT. It was decided to use these technologies for CTMS/CTIS. Applications currently used at CDOT–TOC such as the 'www.COTRIP.org' and device drivers for RTMS have been developed using these technologies. Using the same technologies for CTMS/CTIS will help define a standard set of technologies, and minimize integration issues.

- **Java:** Java is an object-oriented programming language that features "write once, run anywhere" deployment to many popular computing platforms, without the need to redevelop or recompile. Due to the numerous benefits of Java, and its increasing acceptance in the IT circles, Java has been the preferred programming language at CDOT–TOC. The Courtesy Patrol application has been developed using Java. Having a standard programming language will simplify the integration process.
- **Java Virtual Machine (JVM):** The JVM is an engine on which Java applications run (or are interpreted). A JVM is required for each particular platform on which the application is to be run. The JVM is what allows Java applications to be moved from platform to platform with no changes to the application itself.
- **Java Foundation Classes (JFC)/ Swing:** JFC is a set of Java class libraries provided to support building graphics user interface (GUI) and graphics functionality for Java technology-based client applications. With Java as a preferred programming language, JFC/Swing is the natural choice of any java related GUI developments.
- **Java2 Platform, Enterprise Edition (J2EE):** J2EE is a standard architecture from Sun Microsystems that defines and supports a multi-tiered programming model where client applications invoke business logic that executes on an application server, which in turn interact with a separate layer of devices to store and retrieve data. COTRIP.org and CPDS have designs that conform to the J2EE specifications.
- **Enterprise Java Beans (EJB):** EJB's are a major portion of the J2EE platform. They are Java components developed for the server. There are three types of enterprise beans: Entity EJBs, Session EJBs, and Message Driven Beans (MDB). EJB's have been widely used in COTRIP.org and CPDS.
- **Session EJB's:** There are two types of Session EJBs – Stateful and Stateless. A stateful session bean contains conversational state on behalf of the client. Stateless session beans do not have any state information for a specific client.
- **Entity EJB's:** An entity bean represents data in a database and the methods to act on that data. There are two types of data access for entity beans: Container Managed Persistence (CMP), and Bean Managed Persistence. CMP's is the standard method for data access in most Java based application at TOC.
- **Borland Enterprise Server (BES):** BES is a J2EE/CORBA compliant application server. It is a multithreaded server that listens on the network for a client request. On the back-end, the Borland Enterprise Server can connect to virtually any network-accessible service. The Borland Enterprise Server can be a primary web server or it can process requests redirected to it by an existing web server. CDOT–TOC currently uses BES for its existing operations systems.
- **XML:** An eXtensible Markup Language that allows programmers to design their own markup language for creating Web pages with dynamic meta-information. XML format is currently used by the CoTrip.org website.

ESRI Technologies

Following is a list of products developed by ESRI that support the management and manipulation of spatial and graphical information. These products are currently used for development of various applications in TOC.

- **MapObjects:** a set of Java API's for developing Map based GUIs.
- **ArcSDE (Spatial Database Engine):** a tool that supports the management and manipulation of spatial and geographic information.
- **ArcIMS (Internet Mapping Service):** a tool that supports the management and manipulation of spatial and geographic information in a internet environment.

3.2 Technologies Under Evaluation

Following is a list of technologies that are being evaluated for the CTMS/CTIS architecture.

- **Message Oriented Middleware (MOM):** Message-oriented middleware is software that resides in both portions of client/server architecture and typically supports asynchronous calls between the client and server applications. Message queues provide temporary storage when the destination program is busy or not connected. MOM is an alternative to CORBA, hence it is important to analyze the tradeoffs.
- **SonicMQ:** SonicMQ is a Message Oriented Middleware from Sonic Software. It allows two entities to communicate by sending and receiving messages with SoniqMQ managing the background complexity and message volume.
- **Common Object Request Broker Architecture(CORBA):** The Common Object Request Broker Architecture (CORBA) allows distributed applications to interoperate (application-to-application communication), regardless of what language they are written in or where these applications reside.
- **CORBA Interface Definition Language(IDL):** A standard notation language for defining component interfaces.
- **CORBA Notification Service:** The main design goal of the Notification Service architecture is to define the service as a direct extension of the existing Object Management Group (OMG) Event Service, enhancing the latter with important features, which are required to satisfy a variety of applications with a broad range of scalability, performance, and quality of service (QoS) requirements.
- **CORBA Trading Service:** The Trading Service provides a sophisticated 'yellow pages' style object directory that supports complex look-up queries.
- **CORBA Naming Service:** The Naming Service provides the principal mechanism through which most clients of an Object Request Broker (ORB) -based system locate objects that they intend to use (make requests of).
- **Visinotify:** Borland VisiNotify is an industrial strength implementation of OMG Event/ Notification Service. Instead of on implementing on the user level, VisiNotify is implemented on ORB level.
- **Portable Object Adapter (POA):** is a particular type of object adapter that is defined by the CORBA 2.3.1 specification that connects a request using an object reference with the proper code to service that request. POA replaces an obsolete object adapter called a Basic Object Adapter, or BOA.

- **PSA:** Publish/Subscribe Adapter (PSA) is a programming model and software component supported by VisiBroker 5.1. It is simply a wrapper of the MG Notification and Typed Notification Service. The idea of PSA is to present a high level object-oriented abstraction for publish/subscribe and queued communications.
- **QoS:** Quality of Service is used by OMG for providing and controlling reliability, queue management, and event management.
- **SOAP:** Simple Object Access Protocol is a lightweight protocol for exchange of information in a decentralized, distributed environment.
- **XML:** An eXtensible Markup Language that allows programmers to design their own markup language for creating Web pages with dynamic meta-information. XML format is currently used by the CoTrip.org website.

4. OVERVIEW OF CTMS/CTIS ARCHITECTURE:

4.1 Architectural Goals

The goals of the CTMS/CTIS/ATIS architecture are defined below and will be used to guide this evaluation:

General Goals:

- *Feasibility:* To assess the feasibility of the technology approach to successfully construct the application.
- *Scalability:* To evaluate the potential of the system to grow and support future needs arising from introduction of new features and requirements.
- *Maintainability:* To analyze the maintainability of the system after assessing the overall complexity and network overheads.
- *Cost:* To study the cost of the technology approach.
- *Availability:* To evaluate the availability of support products involved in the overall architecture and the availability of programmers required to implement the architecture.
- *Ease of Implementation:* To study the ease of implementing the architecture.
- *Standards:* To study the vendor and industry support available for the standards.
- *Prevalence:* To check the prevalence of the technology choices in US and other state held DOT's. Further to assess the future of the technology choices.

Specific Goals:

- Use three tier (or n-tier) application architecture.
- Manage all application communication using Model View Controller (MVC) pattern.
- Place as much of the application as possible in the Application Server Container.
- Let the application layer manage communication between architectural components. (This is related to keeping as much of the application in the container as possible.)

Application server based architecture will ensure future portability to other EJB Containers. Currently CDOT staff is proficient with J2EE and EJB, this will ensure successful development. Application Server based architecture will ensure ease of future maintenance since application logic and business logic will be kept in one place, instead of being distributed to various architectural components.

Figure 1, shows the CTMS/CTIS architecture framework, proposed by the team at TOC. This framework will serve as a basis for any prototype developments. The architecture framework identifies the core pieces of the architecture based on the functional requirements of the CTMS/CTIS system software. This architecture framework will be comprised of six main components. Each component is essential to the overall system software. The six components are briefly described.

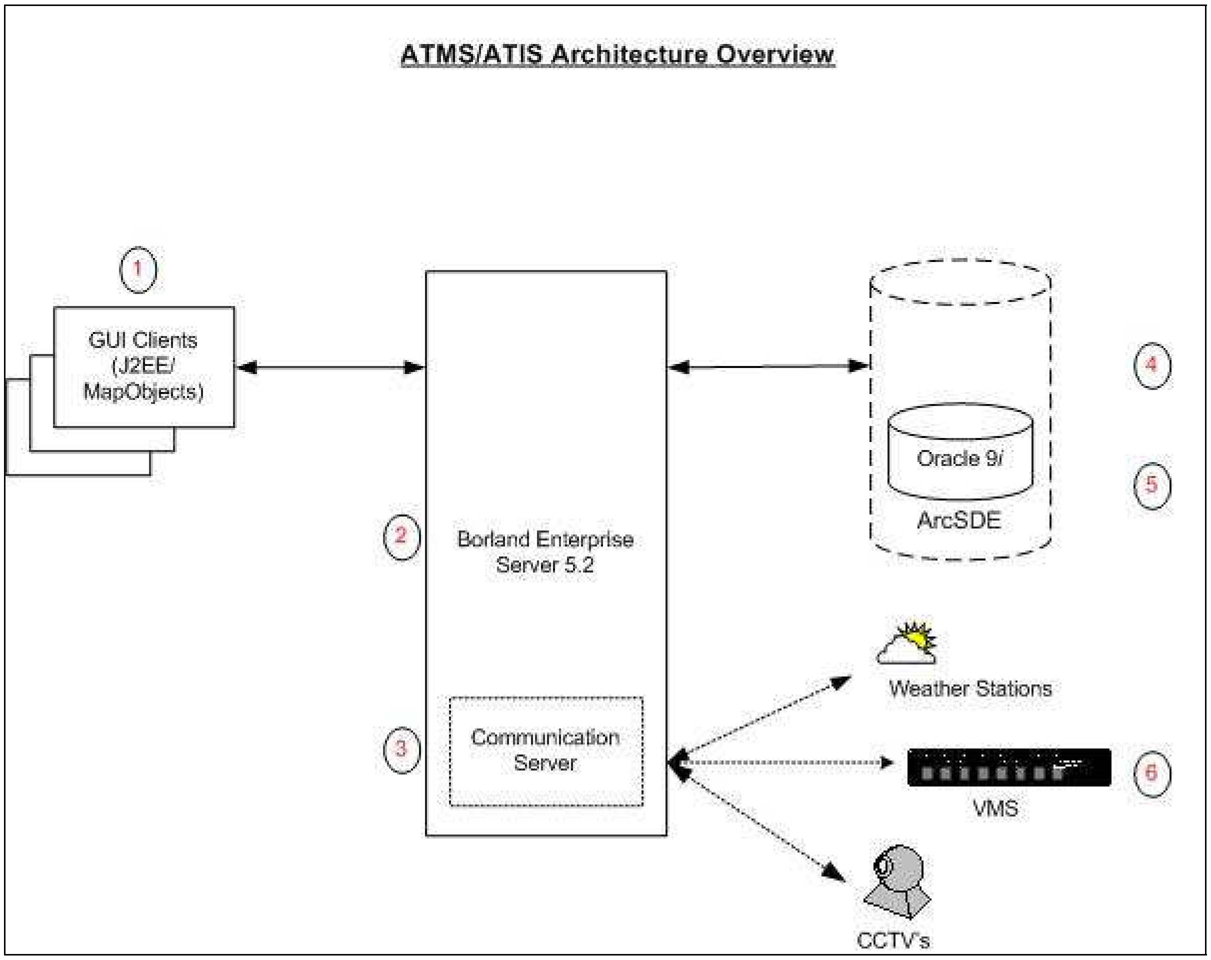
ATMS/ATIS Architecture Overview
1
GUI Clients (J2EE/ MapObjects)
2
Borland Enterprise Server 5.2
3
Communication Server
Oracle 9i
ArcSDE
4
5
Weather Stations
VMS
6
CCTV's

Figure: 1. CTMS/CTIS architecture framework **Figure 1**

4.2 Architectural Components and Assumptions

1. **GUI Clients:** The GUI Clients will provide a Map based interface to the user/operator. The interface will also act as an operator console to poll DMS signs, post messages on DMS signs and perform other device specific operations. A fully functional GUI client will display events, logs, and generate operator reports. It will also enable selection and display of messages on all other devices.

   The user interface will be developed using Java, Java Swing/ Java Server Pages and MapObjects. All device coordinates will be obtained from the spatial database, ArcSDE. MapObjects and ArcSDE will help obtain real time information on the status of the devices.

2. **Enterprise Application Server:** Borland Enterprise Server is a market leader in Enterprise Servers that conform to the J2EE specifications from Sun Microsystems. Currently, CDOT-TOC uses Borland Enterprise Server BES, version 5.2 for most of its applications. BES is therefore a very appropriate fit to the framework and will be used in the overall architecture.
3. **Communication Server:** This component will be responsible for establishing communication with the devices and managing the communication. The communication architecture of the CTMS/CTIS system is under review because of a requirement for creating and managing a thread to communicate with a field device. The BES EJB Container (and the EJB 2.1 Specification) prohibits explicit thread control in the container forcing us to consider two different communication mechanisms: CORBA and JMS/MOM.

   The CORBA (Common Object Request Broker Architecture) Model is an object-oriented model. In addition to the basic CORBA model, several CORBA services will be required. CORBA services follow specific OMG standards, however implementations of services vary from vendor to vendor. Borland and PrismTech are two main vendors of these implementations and both will be evaluated during the process of prototype development.

   JMS/MOM (Java Message Service/Message Oriented Middleware) is a message-oriented model. JMS API's are part of the J2EE specification from Sun Microsystems. As with CORBA, implementation of these standards varies from vendor to vendor. For purposes of architecture assessment products of Sonic Software (SonicMQ) and TIBCO will be evaluated.
4. **Spatial Database Engine:** Arc SDE is a separate architectural component that sits on top of Oracle 9i and manages spatial data. This has caused us to have a dual representation of the datastore object in the system. Spatial data is managed in one table, and non-spatial data is managed in another table. Since we use EJB CMP to manage persistence, it is unknown how we will integrate SDE and EJB.
5. **Database (Oracle 9i)**: Oracle has been the chosen database for the architecture under assessment. The database will have modem pool related information (e.g. Baud Rate, Com Port number etc.). Any information obtained from polling the devices will also be stored in the database.
6. **Devices:** As integration software, the main intent of the CTMS/CTIS system is to communicate with various ITS devices. Some of these ITS devices include DMS, ATR, RTMS, AWOS, HAR and others. (Refer to Glossary for acronyms)

5. CTMS/CTIS COMMUNICATION ARCHITECTURE ALTERNATIVES

This section describes the alternative models evaluated for the communication architecture within the CTMS/CTIS core architecture. As described earlier, for the CTMS/CTIS application architecture, technology choices have been made for the GUI Client (J2EE, MapObjects), Spatial Database Engine (ArcSDE), the database (Oracle), and Enterprise Application Server (BES). For the communication server, however, four alternative models have been identified. Each model uses different technologies, programming model and product implementations.

Alternative1: CORBA – Visinotify – EJB Model.

The first alternative is the CORBA – Visinotify – EJB model. CORBA is the broker architecture. Visinotify is Borland's implementation of CORBA Notification service and will be used for asynchronous Notification to the clients. PrismTech's CORBA Notification service is also an alternative to Visinotify. While the overall model remains the same, both Visinotify and PrismTech's products can be used for asynchronous notification.EJB will be used for database access. Alternative 1 has been prototyped and a working model is available for demonstration.

Alternative 2: CORBA – JMS Model

This particular model is different from Model 1 and will use the Common Object Request Broker Architecture (CORBA) for remote object invocation. CORBA Notification service has been replaced with Java Message Service. Although both services were developed to solve different problems, both achieve similar goals. JMS enables loose coupling of client and server and insures asynchronous communication. The JMS broker used for developing this prototype is SonicMQ from Sonic Software.

Alternative 3: JMS – MDB Model

This model is different from models 1 and 2 and follows a message-oriented paradigm. The model conforms to the JMS/J2EE specifications from Sun Microsystems. The implementations of these specifications vary from vendor to vendor, however for purposes of prototyping SonicMQ and TIBCO have been used as the JMS brokers. EJB is still used for persistence management. Alternative 3 has been prototyped and a working model is available for demonstration.

Alternative 4: J2EE - XML Model

This model uses EJB's for messaging, data access and business logic. The idea is to shift all functionalities inside the Web container and check feasibility of the model. All device and socket communication will be message based using a JMS Server. Message will be a standard XML packet. Option 3 differs from option 4, wherein, modules such as alarm handler, scheduler are implemented as bean façade. Alternative 4 has been prototyped and a working model is available for demonstration.

DATEX Protocol

The Datex protocol was not evaluated on this project. Rather, results from a US DOT study were used which recommended XML over Datex.

For the communication between device server and the field devices, ITS standards conformance is important. NTCIP protocol standards will be used for any such Center to Field communication. There were several considerations in choosing the protocols. A major consideration was DATEX. However, DATEX fails to prove its merit for the following reasons:

1. DATEX suffers from the fact that it is a protocol used exclusively for center–to–center ITS applications. The market may simply be too small and it that jeopardizes the survival of this technology.
2. Currently, there is only one company, TRANSCOM that supports DATEX.
3. The availability of programmers familiar with this technology is very low.
4. There are not many implementations of DATEX nationwide.
5. A study by US DOT suggests that XML is a better option than DATEX.

5.1 Alternative 1: CORBA – Visinotify – EJB Model

(Additional information in Appendix A)

Objectives behind this development were:

1. To explore/understand the Java – CORBA programming combination.
2. To explore/understand CORBA Services, especially the Notification service.
3. To experiment with Borland's implementation of Notification service - Visinotify
4. To check the feasibility of PrismTech's CORBA Notification Service, called Open Fusion, as an alternative to Borland's Visinotify.
5. To document any issues.

Visinotify is Borland's implementation of CORBA Notification Service while Open Fusion is PrismTech's implementation of CORBA Notification Service. The team's evaluation of Visinotify and Open Fusion have concluded that both products conform to CORBA specifications and can be used for CORBA based model. For this particular prototype the team chose to use PrismTech Notification.

This application prototype is structured into 3 packages **Visinotify.driver, Visinotify.map** and **Visinotify.idl.**

Source codes of these packages have been archived in Visual Source Safe, $/CTMS/CTIS_ATIS/Architecture Assessment/src folder.

1. Visinotify.Map: *Client*

This package contains classes that do the following:

- *Start.java* sets the frame for Map (start.java)
- *MapFrame.java* Initializes the Object Request Broker, PSA and resolves POA references.
- *ConsumerNotificationHandler.java* creates the structured event and publish/subscribe to the notification channel

2. Visinotify.Idl: *Device Server*

- *DS_VMS.idl* is the main idl file that defines the operations that can be performed on the device.
- Conversion of idl file to java generates 7 files (Stub, POA, POATie, Operations, Holder, Helper). Each file has different objectives depending on the Java-CORBA communication.
- DS_VMSImpl.java has the implementation for the "idl" file that defines the main operations to be performed on the particular device (e.g. poll, check, connect etc)
- StartServer.java contains classes required for instantiating the Notification service, creating channels, activating POA manager and setting the POA policies.
- EventGenerate.java creates the structured event and publishes the actual event onto PSA.

3. Visinotify.Driver: *Communication API's*

- Contains java classes that constitute the communication API's for communication with the devices through a modem pool.
- For purposes of this application prototype, the Visinotify.driver also contains driver classes that establish communication with the Display Solution VMS signs.

5.1.1 STRUCTURE:

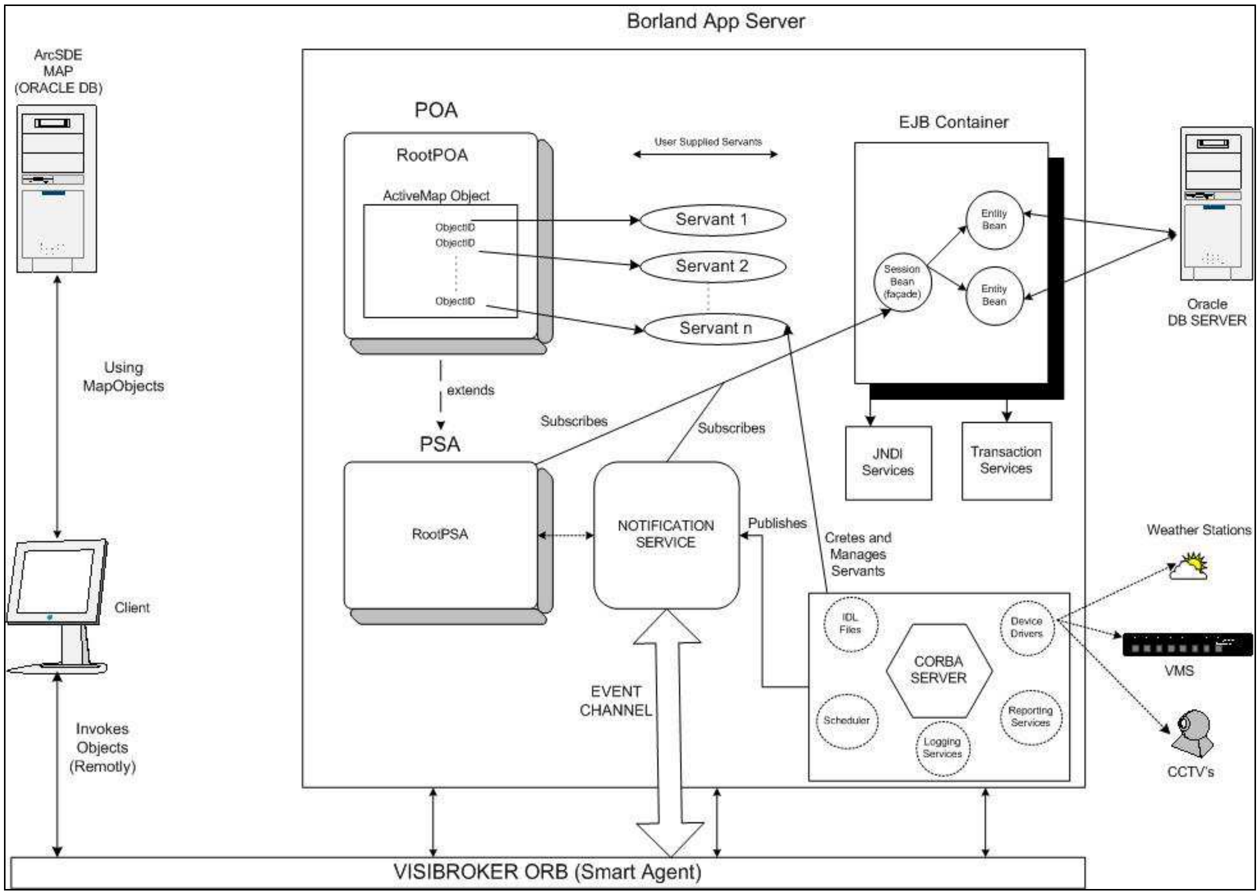

Figure 2. CORBA – Visinotify – EJB model

5.1.2 EXECUTION

This portion elaborates the execution sequence of this CORBA application prototype.

1. Initializing various services:

It is important to have Borland Application Server, Visinotify (or any other Notification service), Visibroker (or any other ORB) and the Visinotify.idl (the device server) up and running. These pieces form the basic blocks of the communication framework for this particular architecture.

2. Starting MapObjects:

With the services running, the MapObjects client can be executed. Each device is associated with an object id, references of which are obtained through the POA Manager.

5.1.3 ISSUES

1. Any communication between the client and server requires the client and server to be subscribers or publishers to a channel. Each time the client or server is disconnected, the subscription is lost; hence channel and subscription management is important.
2. To guarantee delivery of structured events to the client and/or server, additional QoS properties have to be set. Not all notification services implement these QoS properties. E.g. Visinotify support fewer QoS properties than PrismTech Notification service.
3. Each time a client is closed (shutdown) and restarted, the proxies created during previous subscription persist and continue to consume resources. These proxies have to be managed.
4. This architecture requires transaction management to be in the client for synchronized updates to SDE and database through entity beans. This also indicates a thicker client, which will make design and implementation complex.

5.2 Alternative 2: CORBA – JMS Model

Objectives behind this development were:

1. To explore/understand the CORBA – JMS programming combination.
2. To explore/understand the Java Message Service as an alternative to CORBA Notification service.
3. To explore/understands the features of a Message Oriented Middleware.

Packages used in constructing this particular prototype were also used in the previous model. The JMS related packages have been designed keeping possible reuse in mind. Prototypes 3 and 4 may very well use packages designed in this particular model. Source codes of these packages have been archived in Visual Source Safe, $/CTMS/CTIS_ATIS/Architecture Assessment/src folder.

1. **Visinotify.Map:** *Client*

   This package contains classes that do the following:

   - *Start.java* sets the frame for Map (start.java)
   - *MapFrame.java* Initializes the Object Request Broker and resolves POA references.
   - *DS_VMS.idl* is the main idl file that defines the operations that can be performed on the device.
   - Conversion of idl file to java generates 7 files (Stub, POA, POATie, Operations, Holder, Helper). Each file has different objectives depending on the Java-CORBA communication.

2. **jmstester.Driver:** *Communication API's*

   - Contains java classes that constitute the communication API's for communication with the devices through a modem pool.
   - For purposes of this application prototype, the jmstester.driver also contains driver classes that establish communication with the Display Solution VMS signs.

3. **jmstester.MDB:** *MDB and Entity Beans*

   - MDB_first.java implements the MDB façade.
   - CommTypeBean.java is the Entity bean (Container Managed Persistence).
   - CommTypeHome.java is the home interfaces for the CMP.

5.2.1 STRUCTURE:

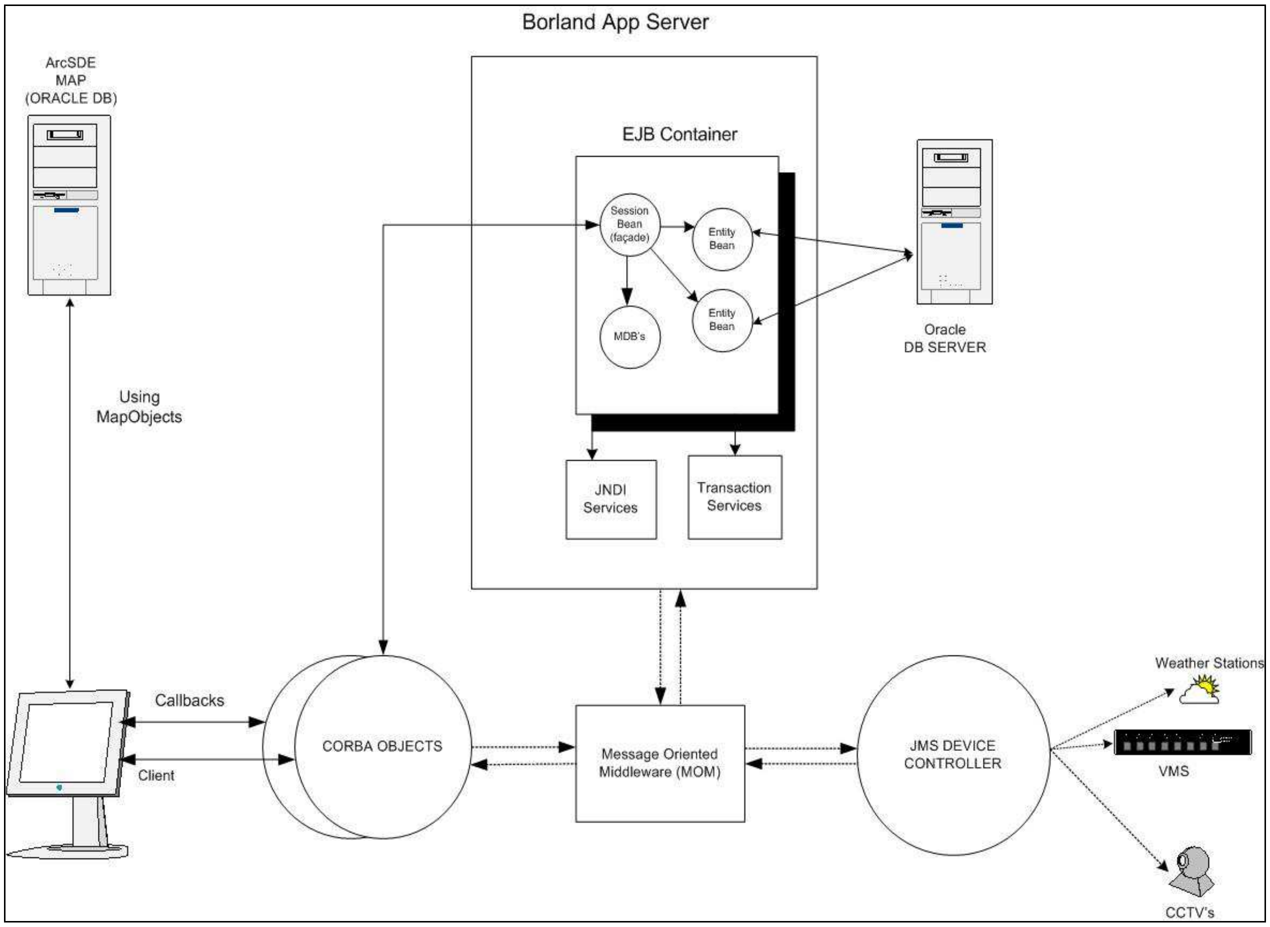

Figure 3. CORBA – JMS Model

5.2.2 EXECUTION

1. It is essential to have SonicMQ and BES running before executing this application.
2. With the services running, the MapObjects client can be executed. Each device is associated with an object id, references of which are obtained through the POA Manager.
3. Two topics: ClientToServer and ServerToClient have been created in mainJMS.java
4. The text message created by *leftpanel.java* is published onto topic ClientToServer. *Mainjms.java* interprets the messages and calls the driver api's.
5. Message received upon polling the VMS is published onto the topic ServerToClient as a JMS object message and is interpret by the *ClientSubscriber.java*
6. ClientSubscriber.java forwards the message to leftpanel.java which displays the message.

5.2.3 ISSUES

1. Each time a client is closed (shutdown) and restarted, the proxies created during previous subscription persist and continue to consume resources. These proxies have to be managed.
3. This architecture requires transaction management to be in the client for synchronized updates to SDE and database through entity beans. This also indicates a thicker client, which will make design and implementation complex.
4. For larger applications, it is very important to design the topics and queues looking at the scope of the application.

5.3 Alternative 3: JMS – MDB Model

(Additional information in Appendix B)

JMS are portable API's that set standards for any Java based client to connect to an MOM. MOM's are therefore at the heart of any JMS architecture. This application prototype was intended at:

1. Exploring/Understanding features provided by the MOM.
2. Exploring/Understanding a JMS based architecture.
3. List issues involved.

This JMS application prototype was built using JMS broker – SonicMQ from Sonic Software. This application prototype is structured into 3 packages **jmstester.driver, jmstester.map** and **jmstester.MDB.** Source codes of these packages have been archived in Visual Source Safe, $/CTMS/CTIS_ATIS/Architecture Assessment/src folder.

1. jmstester.Map: *Client*

This package contain classes that do the following:

- *Start.java* sets the frame for Map (start.java)
- *MapFrame.java* Initializes the Object Request Broker, PSA and resolves POA references.
- *ConsumerNotificationHandler.java* creates the structured event and publish/subscribe to the notification channel

2. jmstester.Driver: *Communication API's*

- Contains java classes that constitute the communication API's for communication with the devices through a modem pool.
- For purposes of this application prototype, the Visinotify.driver also contains driver classes that establish communication with the Display Solution VMS signs.

3.Jmstester.MDB: *MDB and Entity Beans*

- MDB_first.java implements the MDB façade.
- CommTypeBean.java is the Entity bean (Container Managed Persistence).
- CommTypeHome.java is the home interfaces for the CMP.

5.3.1 STRUCTURE

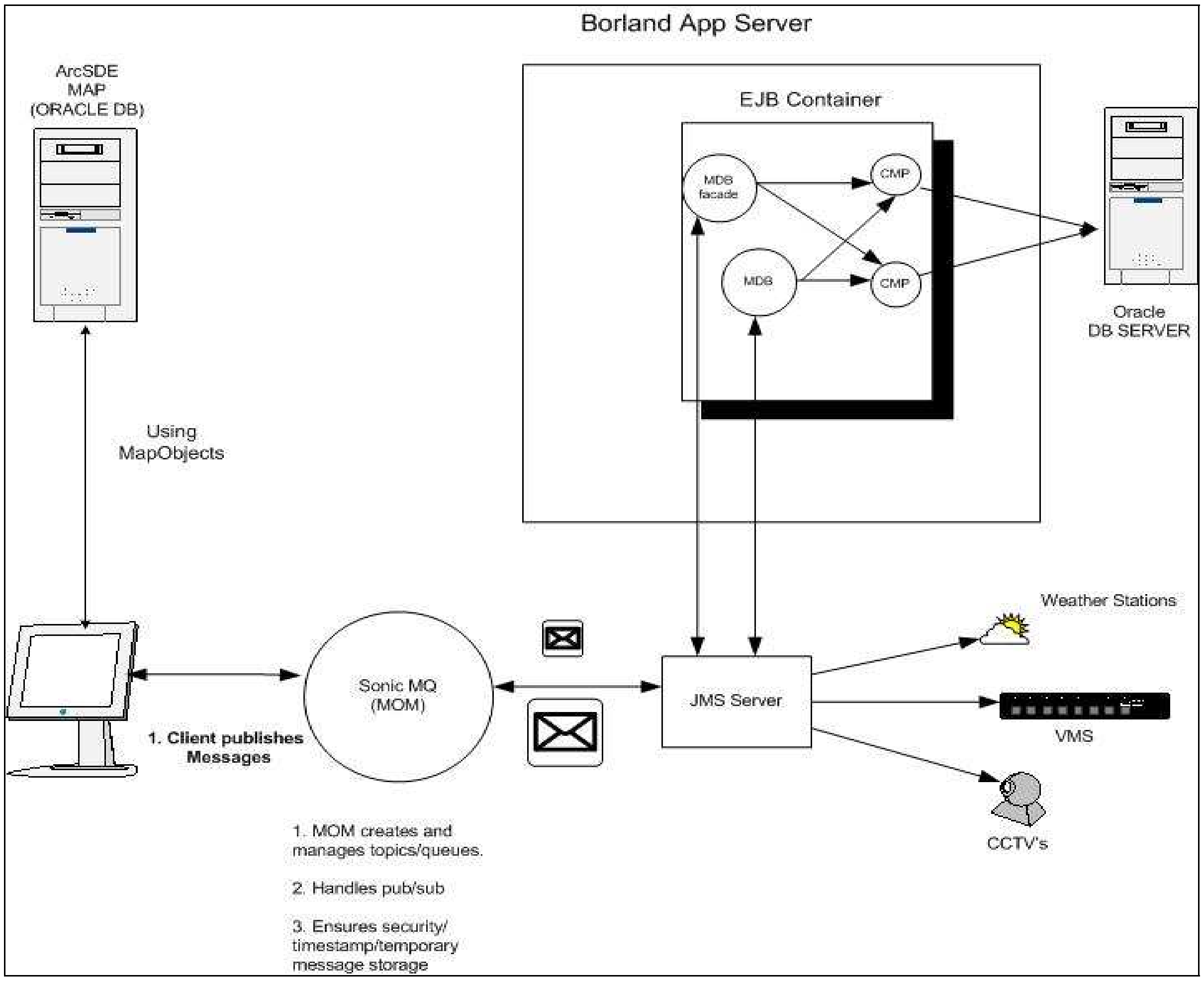

Figure 4. Java – JMS – MDB model

5.3.2 EXECUTION

7. It is essential to have SonicMQ running before running this application.
8. *Start.java* within *jmstester.map* package has the main class. Upon execution, a text message with the appropriate operation (E.g. Poll) is created by *leftpanel.java.*
9. Two topics: ClientToServer and ServerToClient have been created in mainJMS.java
10. The text message created by *leftpanel.java* is published onto topic ClientToServer. *Mainjms.java* interprets the messages and calls the driver api's.
11. Message received upon polling the VMS is published onto the topic ServerToClient as a JMS object message and is interpret by the *ClientSubscriber.java*

MDB Communication:

1. Message received by mainJMS.java is published to MDB façade via topic STOC. MDB is accessed via JNDI lookups.
2. Upon receiving the message OnMessage( ), MDB calls getCommType ( ) from CommTypeBean.
3. The get method executes a Select query to test the MDB – Entity Bean interaction.

5.3.3 *ISSUES*

1. A pure JMS architecture is not possible. For example, API's that extend Connectionfactory depend upon the type of broker used. Hence a JMS client that connects to SonicMQ broker cannot be ported to any other broker (E.g. TIBCO) without changes.
2. For larger applications, it is very important to design the topics and queues looking at the scope of the application.
3. This architecture requires transaction management to be in the client for synchronized updates to SDE and database through entity beans. This also indicates a thicker client, which will make it more difficult to manage during maintenance and operations.

5.4 Alternative 4: J2EE - XML Model

(Additional Information in Appendix C)

JMS are portable API's that set standards for any Java based client to connect to an MOM. MOM's are therefore at the heart of any JMS architecture. This application prototype was intended at:

1. Exploring/Understanding EJB container's ability to handle communication with devices.
2. Exploring/Understanding a JMS based architecture.
3. List issues involved.

This JMS application prototype was built using JMS broker – SonicMQ from Sonic Software. This application prototype is structured into 3 packages **cdot.its.assess4.bean, cdot.its.assess4.server and cdot.its.assess4.util.** Source codes of these packages have been archived in Visual Source Safe, $/CTMS/CTIS_ATIS/Architecture Assessment/src folder.

1. cdot.its.assess4.util: *Client*

This package contain classes that do the following:

- *Message.java* message class that defines the common packet to be sent to the modules.
- *Client_Side.java* Publishes messages to the EJB container and listens for messages retrieved from the devices.

2. cdot.its.assess4.server: *Communication API's (Refer to Appendix entry 4 for message class)*

- *JMS_Server.java* Listens for messages published out of the container and publishes messages obtained from the devices.
- *Message.java* this class represents the common message packet containing routing information.

3. codt.its.assess4.Jmstester.bean: *MDB and Entity Beans (Refer to Appendix entry 3 for details)*

- DelegatorMessageBeanMBFBean.java delegates message published by the JMS Server to session bean for further processing.
- DeviceControllerSSBFBean.java and AlarmHandlerSSBFBean.java are stateless session beans that represent modules and are responsible for the module specific processing of the message packet. E.g. AlarmHandlerSSBFBean is responsible for checking Alarm Handler related logic before message processing.
- RTMSUnitIdCMPBBean.java is a CMP bean to access the relational values of the device from database.
- MessageQueueBeanSSBFBean.java is message driven bean that publishes messages outside the container and to the JMS Server using a JMS queue.
- CmdBean.java accepts message sent by client and delegates it to the bean façade specific to the module.

5.4.1 STRUCTURE

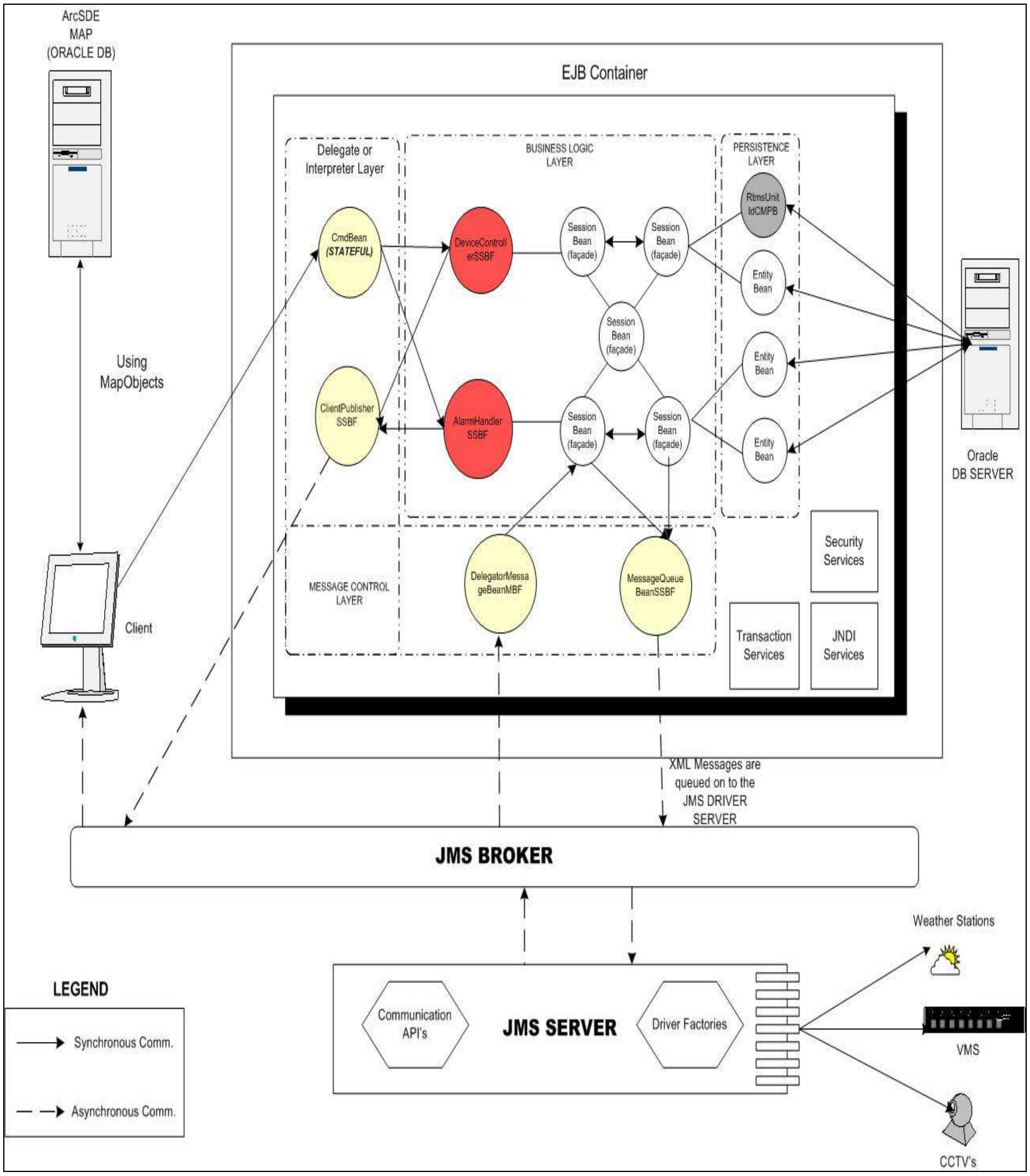

Figure5. J2EE-XML Model

5.4.2 EXECUTION

1. It is essential to have SonicMQ broker and BES running before running this application.
2. *Client_Side.java* is the client program that sends message packet to the CmdBean on JMS queue named SampleQ1.

EJB Communication:

1. CmdBean is an interpreter bean and exposes one remote method executeMessage() which takes a serialize object message as an argument. Once this method is remotely activated by the Client, the CmdBean checks attributes set by the client on the object message and forwards the message to the appropriate session bean.
2. DeviceControllerSSBFBean.java and AlarmHandlerSSBFBean.java are stateless session beans which expose method to executeCommand which accepts message object forwarded by the command bean.
3. RTMSUnitIdCMPBBean.java accesses the relational values of the device specified by the client in the message packet. These value are send by the messagequeuebean to the JMS Server.
4. JMS Server loads the device specific drivers, establishes connection with the devices and performs the required operation such as polling. The message retrieved from the polling operation is send to the ClientPublisher bean via a MDB.
5. ClientPublisherSSBFBean.java finally publishes the retrieved message to the client interface.

5.4.3 ISSUES

1. In this particular architecture flexibility may be an issue. Since most modules are implemented as EJB's, every change to the bean would require redeployment of the bean. Application servers such as JBoss use time efficient techniques for deployment.

6. CRITERIA FOR SELECTION

This section describes various metrics used to compare and contrast the communication architectures. Definitions of each metric are as defined by the Software Engineering Institute. Several criteria may compare the technology as against the overall architecture. For example, complexity considers CORBA and JMS instead of CORBA and J2EE-XML.

6.1 Cost

- **CORBA**

  PrismTech Notification, OpenFusion: (Prices have been obtained from the Sales Quote)

  Development License (Name User based): $6000

  Testing/Production License (Per CPU): $4000

  Oracle Persistence Plug in (Per CPU): $1000

- **J2EE-XML**

  SonicMQ: (Price Quotes were given by the Sales Manager of Sonic Software)

  SonicMQ Professional Developer edition: standards based messaging backbone that supports dynamic routing, RSA security, and server clustering (Licensed for development use only. No Deployment) - $2500 per named user includes client plus functionality
  SonicMQ Enterprise Edition: for deployment – standards based messaging backbone that supports RSA security. No dynamic routing, no clustering, no SonicMQ client plus functionality included - $5000 per CPU
  Sonic MQ Enterprise Edition Plus: for deployment - standards based messaging backbone that supports dynamic routing, RSA security, and server clustering, includes 10 licenses for SonicMQ Client Plus - $7500 per CPU
  Sonic MQ Client Plus: Extended messaging client that supports local persistence and large message support - $2500 per 10 machine license pack.

  A quick evaluation proves that implementation of CORBA notification service such as Open Fusion is more expensive than a JMS broker such as Sonic MQ.

6.2 In-house Expertise

The following expertise is available in house right now.

1. John Williams: Expert in Java, Java Server side technologies, Oracle, Web technologies

2. Greg Arbon: Expert in Java, Server side technologies, Oracle, Web technologies

3. Pawan Kharbanda: Expert in Java, Server side technologies, CORBA, JMS, Oracle, Web technologies.

4. Sachin Saindane: Java, Server side technolgies, JMS and JMS Brokers such as SonicMQ.

5. Kim Hubbard: Java, Oracle, ArcSDE.

To summarize, on the whole the team has significant experience with Java, Java Server side technologies and JMS

6.3 Complexity

Complexity is defined as the degree to which a system or component has a design or implementation that is difficult to understand and verify. It is determined by such factors as the number and intricacy of interfaces, the degree of nesting, and the types of data structures.

CORBA:

- Key to any CORBA based application is the design of the interface definition language (idl) file. The idl file defines the publicly available operations that can be invoked on the CORBA objects. Available operations are implemented as Java classes and are termed 'servants'. Idl file is compiled to generate platform specific classes. Idl2Java compiler creates 7 java specific files, each handles a separate function. Managing these files add to the complexity of the program.
- Default environment set for CORBA objects are different than the ones desired. Behavior of the objects can be changed by setting different POA policies. Managing these POA policies often add to the complexity of the system.
- Programming model used by CORBA Notification service (refer Appendix A – item 2) requires management of admin objects, proxies and consumer objects. As the number of CORBA objects increase, managing the proxies and admin objects add to the complexity of the application.
- Another key requirement of this application is guaranteed delivery of structured events. QoS properties applied to the filters can ensure guaranteed delivery, however all notification services may not support all QoS properties. Choice of service becomes important at this point. Furthermore, setting these properties depends on the number of filters added to the design. (More information on QoS is available in the appendix).
- While a CORBA based application may have a defined structure consisting of idl files and implementation files, design decisions vary depending upon the number of consumers and suppliers. This makes it difficult to come up with a standard design pattern. Furthermore, steps related to management of POA appear to make the programming detailed, no level of abstraction provided here. PSA provides abstraction to a certain extent, however it is proprietary to Borland.

J2EE-XML:

- JMS based applications follow message oriented paradigm. Hence all applications should be designed around the JMS messages that will be passed.
- As the number of messages increase, managing the messages and the topics/queues becomes critical. This task is far less complex in JMS than it is in CORBA Notification service
- Since there are five types of JMS messages available, each message is treated differently by the subscriber. If more than one type of message is published onto the same topic, complexity of the application increases.

In conclusion, CORBA based application is more complex than JMS.

6.4 Scalability

As defined by SEI, Scalability is the ease with which a system or component can be modified to fit the problem area. In simple terms it is the ability of a computer application to continue to function well when it is changed in size or volume in order to meet a user need.

CORBA:

Scalability of the application in this context may require, adding devices, increasing number of client and/or EJB's. Adding devices to the CORBA model requires changes to the communication server. A careful analysis of the overall design and design patterns is important. Further it requires creating additional idl files for each device and implementing operations in additional Java classes. With the standard structure available, scaling up a CORBA application is feasible but tedious.

J2EE-XML:

For JMS architecture, scaling up an application is feasible. If the numbers of clients are increased, the overall architecture can handle increased client requests. A very large number of topics and queues can be managed by increasing the number of brokers. SonicMQ supports 'dynamic routing architecture' to help scale up applications. Brokers can be used in a clustered environment for enterprise wide application.

In conclusion, J2EE-XML model is more scalable than the CORBA model.

6.5 Network Overheads

Overheads are a relative term and may imply additional property bytes or additional properties set for more features.

CORBA MODEL:

- For a CORBA model, overheads are associated with the design of the structured event. It is important to keep a close eye on the headers of these structured events since they directly affect complexity.
- For features such as guaranteed delivery, additional QoS properties may be set on filters. Adding these properties increase overhead in terms of additional bytes to the vector (structured event).

J2EE-XML:

- In the JMS Model message, overheads are associated with the use of message properties. SonicMQ provides proprietary properties to a JMS message. Using these properties will add to the overheads. Further, type of messages itself add to the overheads (e.g. XML message will have more properties than Text Message)

In conclusion, CORBA Model has higher network overheads than the J2EE-XML model.

6.6 Vendors Required

Vendor support involvement implies the number of vendors or vendor products utilized in developing the architecture. More products will increase dependency on the vendor for version upgrades, support and licenses.

CORBA MODEL:

- For Visinotify model, Borland becomes the sole vendor that is involved. While the number of vendor products may not change, the interactions will be limited to Borland and ESRI
- For PrismTech model, Prismtech in addition to Borland is involved. Implications of dependencies on the number of vendors may be worth a consideration.

J2EE-XML:

- For JMS model, Sonic Software becomes involved in addition to Borland.

In conclusion, both models may require interaction with at least three commercial vendors.

6.7 Vendors Supporting the Technology

This particular measure looks at technology from the vendor support perspective. While commercial implementations of both CORBA and JMS are available, the number of vendors supporting these technologies will give an indication of vendor's faith in the technology.

CORBA:

- The advent of CORBA as a specification dates to early 199o's. After the formation of the Object Management Group (OMG), a lot of vendors participated in keeping CORBA open standard and developing CORBA related products. As of today there are at least 200 known vendors supporting CORBA. The number has grown in the past few years.

J2EE-XML:

- The advent of JMS as a specification dates to late 1990's. Although the technology has gained increasing popularity, there are a few well established vendors including IBM, Sun Microsystems, Sonic Software etc. The total number of vendors may amount to about 75. A research report from Wintergreen research has shown that the JMS/MOM market is projected to triple in the next four years.

In conclusion, the number of vendors supporting CORBA is more than the number of vendors supporting JMS. However the JMS market has increased at a growth higher than the market for CORBA products.

6.8 Portability

Portability is defined by SEI, as the ease with which a system or component can be transferred from one hardware or software environment to another.

CORBA MODEL:

- CORBA model was not deployed on another hardware/software environment. However the standards development body (Object Management Group) claims CORBA to be platform independent.

J2EE-XML:

- Portability of JMS based applications depends on JMS service provider. Whether JMS brokers run on different hardware and software platforms depends on the JMS provider.

In conclusion, within the context of CTMS/CTIS, portability may not be an issue with either technology.

6.9 Standards (Acceptance in ITS and/or IT)

This particular metric looks at the acceptance of CORBA and JMS in the overall IT market and later in the ITS market.

CORBA:

- The CORBA development follows specification from Object Management Group. There is considerable support to these standards and a lot of active work going amongst OMG members to continually improve the specifications.
- CORBA as a technology exists since early 1990's and hence has been adopted widely in IT related projects. Of the state held DOT's the CHART II system developed for Maryland Department of Transportation uses CORBA

.
J2EE-XML:

- JMS standards follow specifications from Sun Microsystems. Considering its onset in 1998, JMS has observed several revisions. The most current version is version 1.1 and was released on March 2002.
- Since JMS is a relatively new technology, it has not been used by any of the Department of Transportation within United States. Internet has tremendously encourages adoption of B2B and B2C models. JMS brokers have been extensively used in web based models.

In conclusion, CORBA has been preferred over JMS in ITS, however JMS has been preferred over CORBA in the IT circles. One may account JMS's late penetration in the IT market as a reason for not being widely adopted.

6.10 Support of Asynchronous Communication

CORBA MODEL:

- CORBA was intended to solve computing problems associated with heterogeneous environment. The objective was to enable natural interoperability regardless of platform, operating system, programming language, even network hardware and software. Remote calls on CORBA objects are synchronous calls. Notification service makes asynchronous behavior possible. Hence, to have asynchronous communication, it is important to have CORBA Notification service in the overall architecture.

J2EE-XML:

- JMS was intended to enable legacy applications to integrate with Java based applications. One benefit of JMS is that it provides asynchronous communication with JMS applications. All versions of JMS specification have supported asynchronous consumption of JMS messages.

In conclusion, both JMS supports asynchronous communication and loose coupling while CORBA services such as Notification service supports asynchronous communication.

6.11 Risk

Risk implies the potential loss associated with the usage of a particular technology.

CORBA MODEL:

- CORBA model is definitely a feasible model and can be adopted. Use of Notification Service can help achieve asynchronous communication. However, to the CTMS/CTIS project, use of Notification service poses a potential risk. An evaluation of Visinotify and Prismtech proved that not all commercial implementations of Notification service conform to OMG standards. Using CORBA Notification service can therefore pose risk of increased vendor dependability. CORBA also has restrictions on explicit thread management.

J2EE-XML:

- JMS model fits all architecture requirements. Asynchronous communication is inherent to the JMS technology and is well supported by all JMS providers. An evaluation of SonicMQ and TIBCO proved that risks involved in using these products are far less. Although these implementations provide a lot of vendor proprietary features, they closely conform to JMS specifications. Having chosen one particular product, one may successfully use another product to accomplish the same JMS related tasks.

In conclusion, CORBA poses a potential risk.

6.12 Ease of Implementation

CORBA:

- Although a mature technology, CORBA has been known for the complexities involved in its implementation. Furthermore, CORBA ITS standards have not been completed, including the CORBA ITS object model. Until the object model is complete, deployers must develop their own object model. This complicates implementation, and increases cost.

J2EE-XML:

- JMS technology is compatible with any Java based client. It is easier for an experienced Java programmer to familiarize himself with JMS API's than CORBA technology.

In conclusion, it is easier to use JMS than CORBA.

6.13 Availability of Programmers

CORBA:

- Successful CORBA implementation often requires senior programmers. Specialized programmers may be hard to find. Furthermore, additional training of required CORBA services may also have to be considered.

J2EE-XML:

- Since several companies are adopting JMS technology, programmers are easy to find. Furthemore, since JMS is a technology from sun and uses features similar to that of Java Language specification, a Java programmer will find it easier to learn JMS.

In conclusion, it is relatively easier to find Java/JMS programmers than the specialized CORBA programmers.

6.14 Conjecture about Future

CORBA:

- Since CORBA is used in a variety of applications, its future depends on what happens in the wider IT industry. CORBA's complexity may ultimately be its downfall. There are several proprietary products on the market that can do what CORBA does but without CORBA's complexity. However, even if the broader computer industry opts for these simpler proprietary technologies, companies supporting CORBA with software and technical assistance will probably continue to support it in the short term because of the number of existing CORBA applications in a variety of industries.

J2EE-XML:

- The worldwide middleware messaging market at $415 million in 2002 is expected to reach $822 million by 2006 The market will increase over the 2003-2008 forecast period as new systems are designed to support mission critical transport of integration modules. Mission critical messaging system management accounted for 63% of the markets in 2002. EAI broker system management accounted for 37%. By 2008, the percentages shift.
- http://www.wintergreenresearch.com/reports/MOM_Final.htm

In conclusion, JMS has a bright and encouraging future in comparison to CORBA.

7. COMPARISON OF ARCHITECTURES

7.1 Scoring Scheme

Metrics	Scoring Scheme (0 = low score , 10 = high score)
1. Cost	0 => High Cost, 10=> Low Cost
2. In-House Expertise	0 => No Expertise , 10=> High Levels of Expertise
3. Complexity	0 => High Complexity, 10=> Low Complexity
4. Scalability	0 => Low Scalability, 10=> High Scalability
5. Network Overheads	0 => High Overheads, 10=> Very low Overheads
6. Portability	0 => Not portable, 10=> Highly portable
7. Vendors Required	0 => High dependability on vendors. 10=> Very low dependability on vendors.
8. Vendor Support	0 => Very low support from vendors 10 =>Very high support from vendors
9. Standards (acceptance in ITS and/or IT)	0 => Very low acceptance/ adoption in ITS and/or IT 10 => Very high acceptance / adoption in ITS and/or IT
10. Support of Asynchronous Communication	0 => Poor support of asynchronous communication 10=> Very good support of asynchronous communication
11. Risk	0 => High Risk, 10=> Low Risk
12. Ease of Implementation	0 => Hard, 10=> Very easy
13. Availability of programmers	0 => Low availability, 10=> High Availability
14. Conjecture of future.	0 => No future, 10=> Very bright future

7.2 Scoring Sheet

Metrics	Model 1: CORBA - Visinotify	Model 2: CORBA - JMS	Model 3: JMS - MDBs	Model 4: J2EE– XML
1. Cost	3	4	6	6
2. In-House Expertise	2	3	7	7
3. Complexity	3	3	5	6
4. Scalability	5	5	5	6
5. Network Overheads	3	3	6	6
6. Portability	3	3	6	8
7. Vendors Required	7	7	8	8
8. Vendor Support	7	6	7	8
9. Standards (acceptance in ITS and/or IT)	5	5	4	6
10. Support of Asynchronous Communication	5	5	6	6
11. Risk	3	3	5	6
12. Ease of Implementation	3	3	5	6
13. Availability of programmers	3	3	7	7
14. Conjecture of future.	4	4	5	6
TOTAL SCORE	56	57	82	92

8. SUMMARY:

The Architecture Assessment conducted by the team followed four distinct phases. Each phase has been briefly described below. The team arrived at the core architecture choice after the fourth phase of assessment.

1. **Discovery Phase:** The team started with a brainstorming session to conduct a high level review of the business requirements. There were six elements identified (described in Section 3) that would make the architecture framework. Technology components were identified that would implement the six core requirements. The brainstorming session concluded with possible alternative architectures and a list of technologies. These technologies have been listed in section 2 (Introduction to Architectural Elements)

2. Architecture Review Phase: A series of group meetings and architecture review meetings were planned to discuss and communicate the technologies to the group. Each week, participants presented their views on specific technologies based on prototype development, case studies and research. The architecture assessment document uses the research conducted by the participants. Four prototype models were designed in this particular phase.

3. High Level Code Review Phase: Prototype models were constructed in this phase and were review by the team to gain a better understanding of the technologies involved. Furthermore, issues were identified, discussed and documented. All findings have been documented in the Architecture Assessment document.

4. Final Assessment Phase: In this final phase, all prototypes developed were compared along well defined metrics. The final decision was made based on comparisons and scores given to each model.

9. CONCLUSION/ RECOMMENDATION:

Assumptions:

Before starting prototype development, the technical requirements in terms of synchronous, asynchronous communication, persistence and type of events were gathered. The prototypes take the technical requirements into consideration. The operational pieces aimed at evaluating the technologies and locating their position in the architecture. A few assumptions were made while constructing the prototypes.

1. It was assumed that durable subscription would be required in the final application. Durable subscription implies guaranteed delivery of messages from producer of the messages to the consumer.
2. It was assumed that the communication with devices would be asynchronous.

Conclusions:

- JMS and Notification Service provide the same basic functionalities such as, synchronous communication, asynchronous communication, structured events (JMS messages), and filters. Upon comparison, the difference can be observed in terms of programming complexity. It was observed that it is much easier (fewer steps and classes) to implement Publish Subscribe model in JMS than it is to implement in the Notification model. In this regard (ease of programming), JMS is a preferred choice.
- From a product features standpoint, Visinotify does not conform to all OMG standards (E.g. QoS properties), PrismTech's product does conform to most standards. If Notification service is used, PrismTech is a preferred choice. For JMS Model, SonicMQ and TIBCO were evaluated, while functionalities of JMS brokers remain the similar, Sonic MQ has proved to be faster than TIBCO.
- From a cost standpoint, we concluded that it may be less expensive to use visinotify in a Notification Model. PrismTech / SonicMQ is expensive than Visinotify and would add to the cost.
- From a portability standpoint, CORBA model is more portable as long as proprietary API's are not used. Within the context of Notification Service, CORBA based application can be ported from Notification service of one manufacturer to another if proprietary API's are not used. In a JMS model, portability is still at question. Since only two JMS broker has been evaluated, portability could not be fully tested. However, it was observed that the JMS brokers comply with Sun's JMS specifications.
- Keeping the future of technologies in mind, market share of MOM's is projected to increase to $822 million according to a report from Wintergreen Research.

To conclude with, J2EE-XML is an easier, cost effective solution and will be preferred over CORBA. A score comparison of the four architectures reveals that the J2EE-XML model received the highest scores and hence will be chosen as the core architecture framework.

10. Appendices:

10.1 Appendix A:

1. QoS Document highlighting differences between Visinotify and PrismTech.

   Document is archived in VSS under $/CTMS/CTIS_ATIS/Architecture Assessment//QOS.doc

2. Notification service

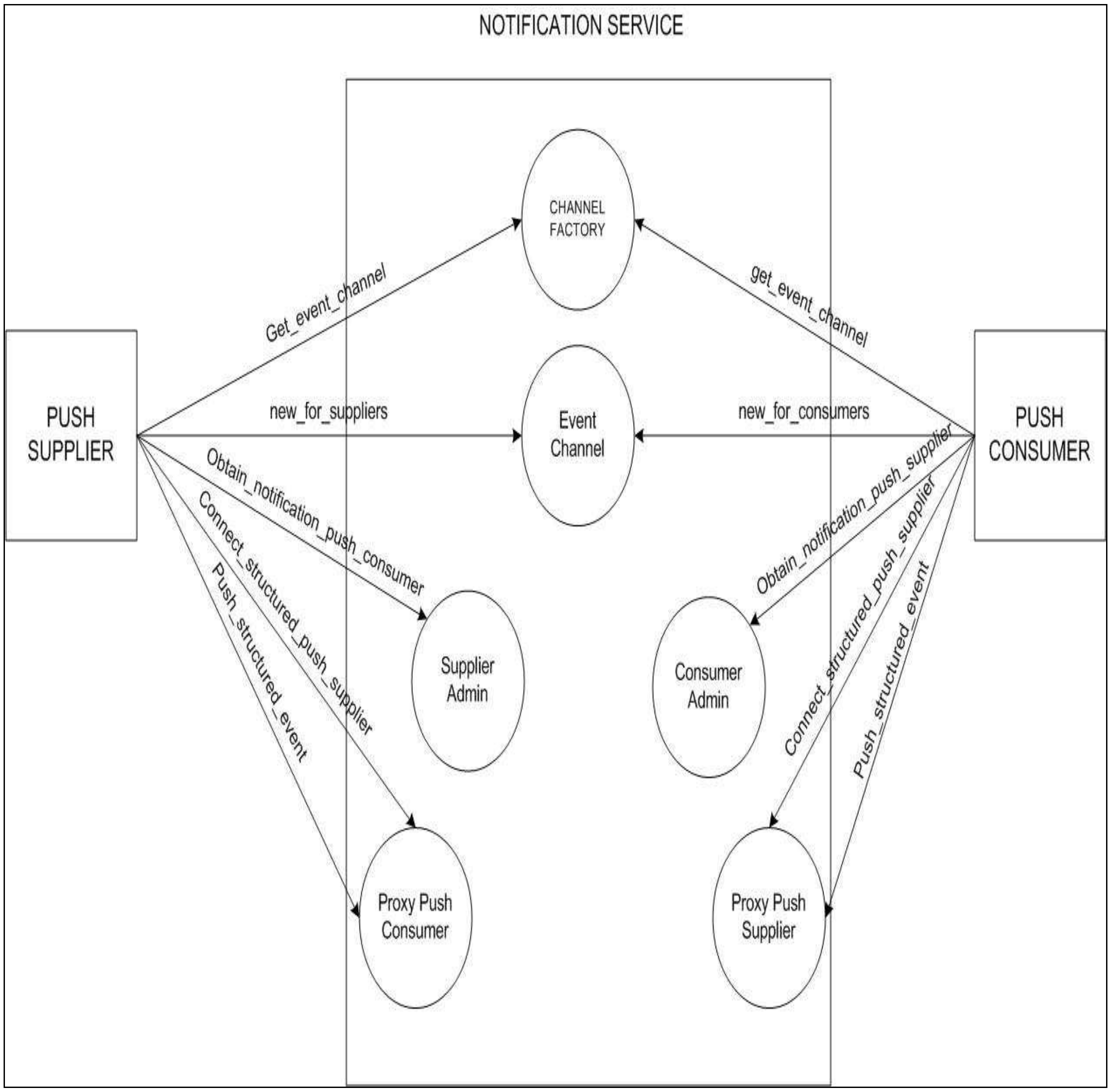

Figure 6: Notification Service

10.2 Appendix B:

1. JMS introductory presentation
2. SonicMQ presentation from Sonic Software

Both are archived under;

VSS $/CTMS/CTIS_ATIS/Tech Docs/JMS/JMS Introduction

10.3 Appendix C:

1. EJB Designs

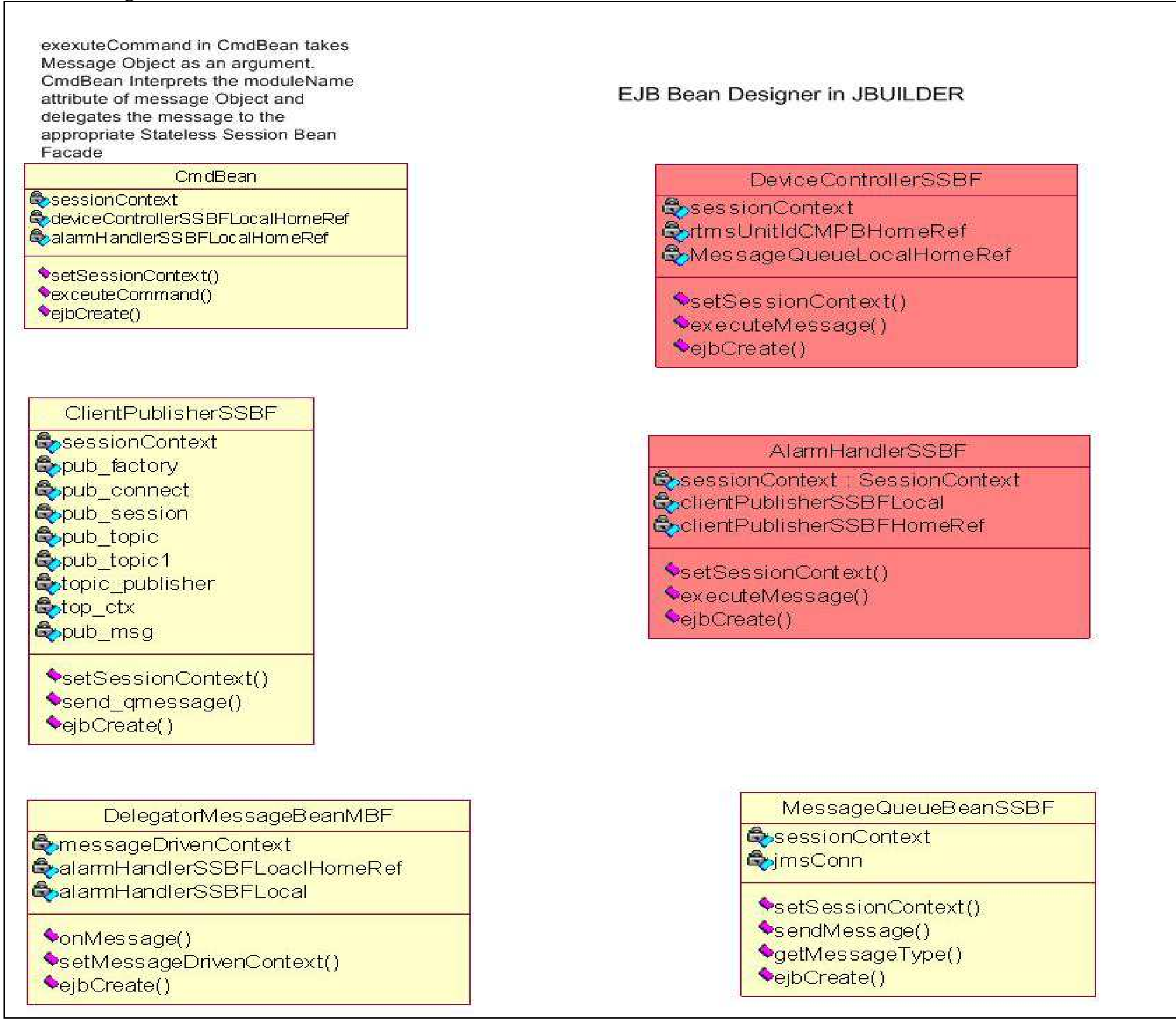

Figure 7: EJB Designs

2. Message Packet

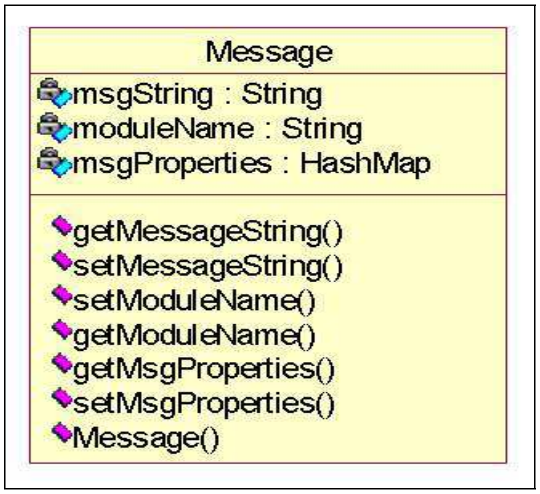

Figure 8: Message Packet

10.4 Addendum:

10.4.1 Rationale for selection of JBoss

The following documents why JBoss was chosen as the preferred vendor for the CTMS project.

Speed of Development

There are numerous ways to measure speed of development. For instance, there is “compilation and build times” as well as the time it takes to implement a technical area like security or clustering. JBoss development time is 6x faster than other application servers requiring stub generation. JBoss is built from the ground-up on Java and its use of Java technology is years ahead of any other vendor product, especially products built on top of older technologies like CORBA (products requiring stub generation include servers from BEA, IBM, and Borland).

Example A: Implementing Security

JBoss’ large developer following and market share has created a wealth of examples in all areas of development and a solid Open Source product. The product and its examples are standard, not proprietary and they are easy to use. For example, the security examples used to incorporate security in CTMS included work from Sun Microsystems documentation, JBoss documentation, and JBoss sample code from both its product and its web forum – contributed by fellow JBoss developers. It took only 1 person 1 hour to build a rudimentary, yet workable security example in JBoss.

On the other hand, for Borland, the security still does not work, even after 2 people tried for several days and several cases were opened with technical support. There are no other examples (e.g., contributions from Borland users) to use because of the lack of adoption of the Borland product. Several outstanding security cases submitted by the Enroute team remain to be solved by Borland even after more than a week.

Example: Implementing Clustering

With JBoss installed on multiple machines, it took only a morning to cluster the servers and another hour to have a test application load balancing requests across the cluster.

Borland requires a separate clustering license, which has yet to be functional in the project environment.

Performance
Products such as WebSphere and Borland that are built on top of existing, outdated CORBA technologies are slower and less integrated. While CORBA is already a large burden, adding a J2EE layer onto the CORBA servers adds additional overhead. Borland is actually changing features in its code-base to be like JBoss because of the performance gains inherent in “dynamic proxying” versus using generated IIOP stubs to make EJB calls.

JBoss’ Licensing: Hassel-free and Cost-Free
JBoss is Open Source and free for all uses (Open Source does not equate to free, so this is important). Its license requires that it remain free, so there is never a fear of lock-in in the future or extra licensing surprises.

Market Position
Market share is a good indicator of a product’s viability and future shelf life. JBoss is not far behind BEA and IBM, whose WebLogic and WebSphere servers hold the #1 and #2 positions in the market. 3-4 years ago, the market was saturated with numerous J2EE vendors including BEA, Bluestone/HP, Borland, IBM, Enhydra, JBoss, Jonas, JRun, Merant, SilverStream, and Unify. Now, there are 3 contenders: BEA, IBM, and JBoss. Together these 3 vendors dominate ~99% of the market leaving 1% to split between the survivors. To go with any vendor besides the top 3 is to be siding with a losing cause. This results include lower levels of support (already seen in Borland support for 6.0), longer release cycles between products versions, less support for latest standards, and worst of all a high probability of the product being discontinued altogether.

JBoss’ Viablility in the Long-term
JBoss is not a new company. It has been in business for several years based on revenues from its consulting, training, and production support. As of February 2004, JBoss is backed by a first-round of $10 million dollars in venture capital .The newly infused $10 million in capital, and a large J2EE market share JBoss will be around in the long term.

Also, JBoss is bundled in product from major vendors across both the hardware and software industries. HP has an “Open Source” architecture that includes Jboss and HP supports the software in this configuration

Support for JBoss
JBoss is a *real* product. It is supported via the company, JBoss, LLC. JBoss, LLC offers various professional support contracts and numerous training courses given around the globe. Also, there are numerous technical books written about the JBoss server including a server administration book and a book on JMX (the technology from Sun Microsystem’s that is the unifying backbone of JBoss). Finally, the wealth of information on the Web about JBoss is critical to its successThe following tables describe the cost structure of J2EE application servers from BEA, IBM, Borland, and JBoss.

Pricing Models

Vendor/Product	**Cost/CPU**	**Number of CPUs**	**Total Cost**
IBM WebSphere	$35,000	10	$350,000
BEA WebLogic - clustered	$14,000 (CSC discount price)	10	$140,000
Borland Enterprise	$12,000	10	$12,000
JBoss	0	10	$0

Development Licenses

Vendor/Product	**Cost**	**Total Cost**
IBM		
BEA	$850/developer	$4250
Borland	$0	$0
JBoss	$0	$0

Support and Maintenance Models

Vendor/Product	**Cost**	**Total Cost**
IBM		
BEA		
Borland		
JBoss	$8,000	$8,000

10.4.2 Rationale for Selection of JBoss MQ

Although the JMS brokers Sonic MQ and TIBCO were used for evaluation, the final architecture will use JBoss MQ. Reasons for this choice is documented below

JBoss MQ is deployed as part of the "all" configuration bundled with the JBoss distribution. Thus JBoss MQ comes at no additional cost. In this regard, SonicMQ and TIBCO are priced above $ 10,000.

Features

JBoss MQ offers the following really attractive features;

1. Automatic server fail-over
2. Remote client connectivity to JBossMQ
3. Lossless recovery after fail-over for messages targeted to durable subscribers
4. Client notification via connection ExceptionListener on fail-over
5. Easily configurable load-balancing setup.