Fall 2007

Distributing Real Time Data From a Multi-Node Large Scale Contact Center Using Corba

Joe Goggins
Regis University

Regis University
College for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

Distributing Real Time Data from a multi-node large scale Contact Center using CORBA

Joe Goggins
joegoggins@yahoo.com
[05118778]

A Project Report submitted in partial fulfillment of the requirements for the degree of Master of Science in Software and Information Systems

School for Professional Studies
Regis University
Denver, Colorado

National University of Ireland
Galway

31st August 2007

Advisor: Dr. Des Chambers

Declaration

I herby declare that this material, which I now submit for assessment on the programme of study leading to the award of Master of Science in Software and Information Systems is entirely my own work and has not been taken from the work of others, save to the extent that such work has been cited and acknowledged within the text of my work. This body of work has not previously been used to obtain any other award in this institution, or elsewhere.

Joe Goggins

_____

National University, Galway

Regis University, Denver

August 2007

Abstract

An Abstract of a Professional Practical Report Submitted to Regis University School for Professional Studies in Partial Fulfilment of the Requirements for the Degree of Master of Science in Software and Information Systems.

Distributing Real Time Data from a multi-node large scale Contact Center using CORBA

by

Joe Goggins

31st August 2007

This report is based on the experiences of the author while participating in the final year of a Masters of Science in Software and Information Systems.

Acknowledgements

Thanks to Mr Des Chambers for affording me his experience, his time and for his guidance throughout the process of writing this paper.

Table of Contents

List of figures:

List of Tables

Organization of this Thesis

This paper is divided into the following sections:

Section 1 : Introduction

This section contains Chapter 1 providing an introduction to the thesis topic. Chapter 1 defines the purpose of this thesis paper, the research and evaluation topics the paper aims to resolve, the planning strategy used and the deliverables required.

Section 2 : Literature and Technology Review

This section is made up of Chapters 2 and 3. Chapter 2 will cover the background to the thesis, the software topic being researched and evaluated, the requirements of the thesis and a brief description of high level technologies that will be part of the research. Chapter 3 will review the State of the Art technologies in the area of Distributed Computing.

Section 3 : Technology Evaluation

This section is made up of Chapters 4 and 5. Chapter 4 will review the technologies discussed in Section 2, while Chapter 5 evaluates the vendors that implement the CORBA Notification specification.

Section 4 : Case Study

Chapter 6 is the Case Study, where a description of the design and implementation of the CORBA Notification Service is detailed. This chapter also includes the various ways in which the implementation may be configured and how it meets the requirements specified.

Section 5 : Case Study Evaluation

Chapter 7 is an evaluation of the Case Study, where the CORBA Notification Service is evaluated against the existing implementations and the requirements presented in Section 2. Chapter 8 completes the thesis paper with a conclusion of the paper and a description of further work that may be done.

Abstract / Executive Summary

This thesis researches and evaluates the current technologies available for developing a system for propagation of Real-Time Data from a large scale Enterprise Server to large numbers of registered clients on the network. The large scale Enterprise Server being implemented is a Contact Centre Server, which can be a standalone system or part of a multi-nodal system.

This paper makes three contributions to the study of scalable real-time notification services. Firstly, it defines the research of the different technologies and their implementation for distributed objects in today's world of computing. Secondly, the paper explains how we have addressed key design challenges faced when implementing a Notification Service for TAO, which is our CORBA-compliant real-time Object Request Broker (ORB). The paper shows how to integrate and configure CORBA features to provide real-time event communication. Finally, the paper analyzes the results of the implementation and how it compares to existing technologies being used for the propagation of Real-Time Data.

SECTION 1: INTRODUCTION

Chapter 1: Introduction

Chapter 1

1.1 Statement of the problem

The paper involves the research and evaluation of technologies related to the propagation of Real Time data from a Contact Center Server. A Contact Center Server is a server based application that handles Inbound and Outbound communications with customers. A Contact Center Server is connected to both the telecommunications network and the IP network with Contact Center Agents configured to handle communications from customers. Contact Center Agents are individuals that represent a company to whom a customer is contacting. Each Agent is configured with a skillset or group of skillsets allowing for the Contact Center Server to be configurable for skill-based routing and call treatment flexibility, and possesses extensive management reporting capabilities. The latter helps companies to respond quickly and effectively to the customer call-flow dynamic by constantly changing priorities. The server hence allows for superior call management capabilities to be delivered.

A Contact Center Server provides both Inbound and Outbound communications with customers. Inbound communications refers to any contact that is initiated by the customer while Outbound is where a Contact Center Agent initiates contact directly to a potential customer.

The Server can have a maximum 3500 agents configured, allowing for the configuration of Contact Center Agents thereby enabling a quick and efficient response to customer demands. The term Contact Center refers to the fact that the server has the ability, not only

to support voice calls but also many different types of multimedia, such as email and instant messaging. A ‘contact’ refers to any telecommunications or IP connected customer. As each contact enters the Contact Center Server, basic reporting on agents, skillsets and other statistics are available. This information needs to be propagated to third party customers quickly and efficiently to provide data for the reporting capability in third-party applications. At peek times, large Contact Center Server deployments, with a high number of configured agents, will be processing around seventy thousand calls per hour. This means that the statistical information been propagated from a Contact Center is extensive.

Currently, the Contact Center Server supports two ways in which third-party applications can obtain real-time statistics from the Server for use in basic contact center status reporting applications. A typical example of a third-party application would be a reader-board or agent desktop application where the number of contacts in the system, number of contacts answered, number of contacts queued etc., can be visible to all within the organization.

1.2 Purpose

The purpose of this thesis is to determine a way to solve the propagation of Real-Time Data from a Contact Center Server to third-party applications. The solution to the problem must

- provide the information in a defined and structured message format,
- allow for the client to receive / retrieve information from the server,
- provide platform and language independence for the customer applications receiving / retrieving the information,

- relieve the server of processing and managing individual connections for each client wanting to receive information,
- provide asynchronous message to the client application in a loosely coupled system.

1.3 Scope

The goal of this thesis is the development of an application that can communicate between different types of computers and programming languages. In the world of computing this is referred to as distributed object or heterogeneous computing. This type of computing requires middleware to facilitate communication between disparate hardware and software. It is purpose is to stand between diverse applications and handle low-level communications, while eliminating platform and language barriers. This thesis paper, the Case Study and the evaluation is focused primarily on the platform and language independence technologies and standards available in the current market.

During the scope of this paper, the issues with current implementations was part of the initial thinking behind using set standards that are supported and stable in the common telecommunications environment. The thesis evaluates the importance in the telecommunications industry to consider the standards being used, to allow for current and future interoperability between different services supplied by different vendors. This ensures a ‘fruitful’ array of services for the customers, which can be provided by multiple vendors.

1.4 Planning

The planning for the project followed the Uniform Software Development and Release Program (USDRP). It is an initiative aimed at creating a framework for the processes, tools and metrics used to develop software. USDRP is related to Life Cycle Management providing a formal and highly structured way of defining a workflow.

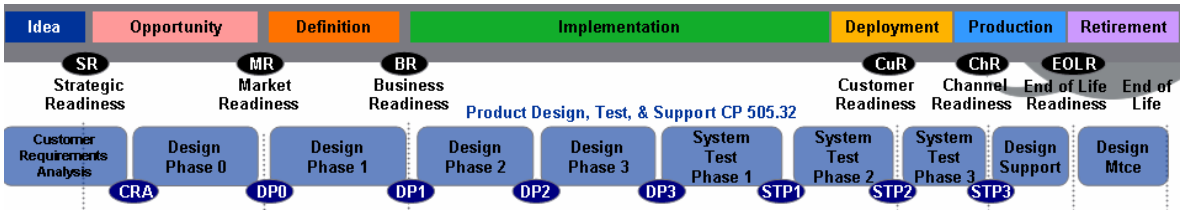

Figure 1: USDRP Workflow

The first step towards gathering the information presented here involved a process of requirements gathering and design decisions between customer requirements and design. The result of this process was a set of core assumptions and requirements related to the propagation of Real-Time Data efficiently from a Contact Center Server.

The next phase involved the definition of the system architecture, the selection of the technologies to be used in implementing a solution, the creation of the scenarios under which the solution would be tested and the creation of a schedule for the delivery of the defined solution.

The implementation phase, which is detailed in the case study, provides the implementation details where the software development is completed, software inspections are carried out and customer documentation is started.

The evaluation of the case study carries the paper into the Design Phase 3 stage of the USDRP life cycle, which involves the testing of the software solution, the completion of bug fixes to the software and the preparation for verification of the solution before validation.

1.5 Research Methodology

Initial research for the project involved researching the technologies available to meet the requirements of the project. On researching these technologies, this allowed for the evaluation of the technologies to define one that best suit is the requirements laid out for the implementation of a solution to the propagation of Real-Time Data from a Contact Center Server. This paper covers the initial research into the existing and emerging technologies in a thematic research manner and in order of chronological review. The paper also evaluates these technologies in order to provide a suitable distributed technology that best matches the requirements.

1.6 Deliverables

The following is a list of deliverables that will be completed as part of the thesis.

- A background into the operation of a Contact Center Server and it is existing technologies.

- A review of the current State of the Art (SOTA) technologies used in Distributed technologies in today's environment.
- Case study of the design and implementation of a solution for the propagation of Real Time Data from the Contact Center Server to third party customer applications running on different platforms and different programming languages.
- An evaluation of the technology used; how it is the best solution and any pitfalls that may have been encountered.
- A review of the technology chosen and recommendations on further research that may be carried out.

1.7 Thesis Overview

This thesis is organized into five different sections with 8 chapters. Each section reviews different parts of the paper. Section 2 is made up of two chapters, Chapter 2 and Chapter 3 that review the current technologies, a background to the paper and a literature review of technologies. Chapter 2 will introduce Contact Center Server, it is operation, it is current implementations for data propagation and their associated limitations. It will also introduce a broad outline of the various technologies discussed in the paper.

Chapter 3 introduces the SOTA technologies in the area of distributed systems while describing their operation and features.

Section 3 is an evaluation of the SOTA technologies. Chapter 4 reviews the technologies described in Chapter 3 and defines which technologies best suit the potential solution to the

issues defined in Chapter 2. Chapter 5 is an evaluation of the different vendors that implement the technology chosen for the solution.

Section 4 defines the case study. Chapter 6 is the design and implementation of the solution that shows how the chosen technologies used to resolve the issues mentioned in Chapter 2.

Section 5 is the evaluation of the case study from Section 4. Chapter 7 looks at the desired characteristics of the technology, while evaluating the outcome of the implementation. Chapter 8 is a summary of the paper, describing the operation of the solution and how it meets the requirements specified in Chapter 2. Future work will also be examined in this chapter.

SECTION 2: LITERATURE & TECHNOLGY REVIEW

Chapter 2 – Concepts & Background Information

Chapter 2

Although the audience for this paper is intended for the technical development community, this paper will allude briefly to the description of the core system; it is current operation and the issues being dealt with in this paper.

The following sections outline the background to fields of interest core to this thesis. To begin, we will start with an overview of Contact Center Server that produces the Real Time data and continue to review the different technologies, the standards involved and the SOTA technologies that implemented the standards.

2.1 Contact Center Server

During normal operation, the Contact Center Manager Server generates a variety of Real-Time Data statistics. These statistics may be grouped into the following statistical packages:

- Application statistics
- Skillset statistics
- Agent statistics
- Nodal statistics
- IVR statistics
- Route statistics

Each statistical package has a defined data type of Cumulative, State, or Admin.

- **Cumulative**: The statistics are accumulated over a specified period of time (for example, the number of calls answered during an interval).
- **State:** The value depends on the instantaneous state of the system (for example, the state of an agent at a given time).
- **Admin**: The value is entered by a data administrator and is not affected by call events (for example, a skillset ID).

For cumulative statistics, data can be collected in two ways:

- **Moving window** The data is collected within the fixed size time window of 10 minutes that moves forward as time progresses. The fixed size time window is divided into a number of equal data sampling periods. As each sampling period

expires, data collected in the current sampling period is added to the totals of the current time window, while the values from the oldest sampling period within the current time window are subtracted from the totals. Therefore, the totals always represent the last 10 minutes of activity.

- **Interval-to-date** The data is collected on an interval basis. The interval is in increments of 15 minutes up to a maximum of 24 hours. When the specified interval is reached, all data fields are reset to zero, and collection starts for the next interval.

This is the means by which Real Time data propagation is currently achieved by the Contact Center Server. There are two interfaces defined by the Contact Center Server that can be used by the third-party applications receiving the Real-Time Data. Each of the two interfaces published to allow access to the Contact Center Server will be discussed in the coming paragraphs. These interfaces are:

- Real Time Multicast
- Real Time Data API

2.1.1 Real Time Multicast

The Real-Time Multicast interface uses IP multicast technology to propagate Real Time Event Data to third-party application developers. It provides a data interface between third-party applications and the component in Contact Center Manager Server responsible for collecting and maintaining real-time event data.

The IP multicast-based interface provides efficient distribution of real-time data to multiple customers. The IP multicast interface does not mandate a specific programming language or platform. IP multicast (RFC1112) defines a mechanism for efficient distribution of data to a group of users. It enables the deployment of scalable and platform-independent receivers of real-time data from Contact Center Manager Server. The concept is inherit from the Internet Protocol (IP), providing a mechanism to enable a one-to-many relationship when sending data, thus improving efficiency when sending data. This, in turn, means that the number of users in the data receiver group is unlimited.

Communication is based on the client/server paradigm. The client references any software application connected to the RSM server on Contact Center Manager Server using one or both of the interfaces defined in this document. The RSM server acts as the source of data, and the data consumers are the third-party applications (or clients) connected to the RSM server. Communication between client and server is through IP networks (LANs and WANs), based on a connection-based (point-to-point) protocol for CORBA communication, and IP multicast data for the real-time data. The fact that the concept is inherent of the IP means that the User Datagram Protocol (UDP) packets sent by multicast will not carry any reliability thus resulting in a potential issue with guaranteed delivery and retransmission of lost packets.

The majority of customers have the following issues with this method of gaining information from the Call Center Server:

- The network routers need to be upgraded in most cases to allow for multicasting.

- The data is being multicast over an open network causing bandwidth problems on busy Call Centers.
- The data is visible to all.

2.1.2 Real Time Data Interface API

The Real-Time Data API is an interface provided for the Windows environment and is referred to as the Real Time Data (RTD) API client. The API client accesses a Contact Center Manager Server by way of a TCP/IP connection. The API allows a single application to connect to a single server providing a one-to-one connection, which in turn provides reliability of message delivery, but adds further processing from the server providing the messages.

To display a continuous stream of data from multiple servers, applications must connect to each server through a different process. Each server requires the client application to provide login details, which will allow that client to access the Real Time Data in two different manners:

- They can make one-time requests for data.
- They can register with a server for a continuous stream of data updates.

If a continuous stream of updates is requested, an update rate must be specified. Updates will not occur more frequently than the rate specified. Depending on system load, however, updates may occur less frequently than requested. The performance issue with multiple client connections is that the server is responsible for the load. The interface also has the following disadvantages:

- No interoperability, programs must use a Windows Dynamic Linked Library (DLL) (C++ and .NET)
- Limited client applications due to the fact it is a multi-threaded push service.
- Developers can only develop client applications on Windows machine.
- Client applications are not portable to other environments.

2.1.3 Requirements

The requirements gathering process resulted in the following list of requirements.

- Structured message and specifications
- Scalability
- Interoperability
- Asynchronous Messaging

2.1.3.1 Structured Messages and Specification

The idea of a structured message is to have conformancy in the messages being sent from the server to the client applications affording the delivery of different types of messages in the same structure. This can be an internally defined message structure allowing different types of events to be sent using the same structure. This enables filtering of messages to be done along with decoding of messages received by clients.

A specification, on the other hand is the idea of a structured message that is already defined by a supporting standards organization, allowing for different vendor implementations of this standard to be used in conjunction with the project. An example of this would be a defined specification for a structured message that a vendor has implemented a filtering

service for; this would in turn mean that we could use this filtering service to save development costs and continue to integrate these services in building our service.

2.1.3.2 Scalability

This is an important factor in the design of a new service that will be used for delivery of all events, both internal and external to the server. The design and implementation must have the ability to support a large number of client connections that may be dependent on the Real Time data propagated by the Contact Center Server. Services that only provide the ability for a small number of server and client connections will suffer in the process of ensuring the development of a scaleable server.

2.1.3.3 Interoperability

A system of modular parts is not as desirable as a system of interoperable modular parts. The ability to break systems down into stand-alone modules is useful, but being able to build these modules into something new is more valuable. It is the interoperability of modular parts that allows developers to leverage their existing work and cut their development costs. Furthermore, interoperability grants a sense of assurance that a system built to work within one environment will work across multiple environments.

In the context of the development of the new service, it ensures that one system can talk to others using defined and supported specifications. It is a rather broad term, but in the world of communications it provides the ability for different services to exchange data, via a common set of procedures that are known and implemented by all of the connected services.

According to ISO/IEC 2382-01, *Information Technology Vocabulary, Fundamental Terms,* interoperability is defined as follows: "The capability to communicate, execute programs, or transfer data among various functional unit is in a manner that requires the user to have little or no knowledge of the unique characteristics of those unit is". [29]

2.1.3.4 Asynchronous Messaging

Asynchronous Messaging is sometimes known as the "fire-and-forget" way of delivering messages, ensuring that the server does not need to be in a blocking state, while it await is an acknowledgment from the receiving client. Having this feature adds to the scalability option, but also provides the server with a mechanism where it does not need to worry about the connection status of the clients.

This means that we must guarantee that the mechanism used for the delivery of messages ensures the delivery of those messages to the connected client, to prevent the pitfall mentioned when discussing the disadvantages of using multicast delivery.

2.2 Object Management Group

The Object Management Group (OMG) is a computer industry consortium setup in 1989, which develops enterprise integration standards for a wide range of technologies, and an even wider range of industries. It is a non-profit worldwide consortium consisting of software vendors and members that are dedicated to promoting the theory and practice of object technology (OT) for the development of distributed computing systems. OMG's

middleware standards and profiles are based on the Common Object Request Broker Architecture (CORBA) and support a wide variety of industries.

2.2.1 Common Object Request Broker Architecture (CORBA)

CORBA, the Common Object Request Broker Architecture, is the OMG's open, vendor-neutral architecture and infrastructure that computer applications use to work together over networks. It was first released in 1991, with the promise of a platform and software independent architecture for writing distributed, object-oriented applications. The technology is best described as a distributed, heterogeneous collection of objects that interoperate. Figure 2, shows a high level view of how CORBA is used on the network. In addition to alleviating the communication and migration barriers, CORBA provides services and facilities that enhance client/server computing.

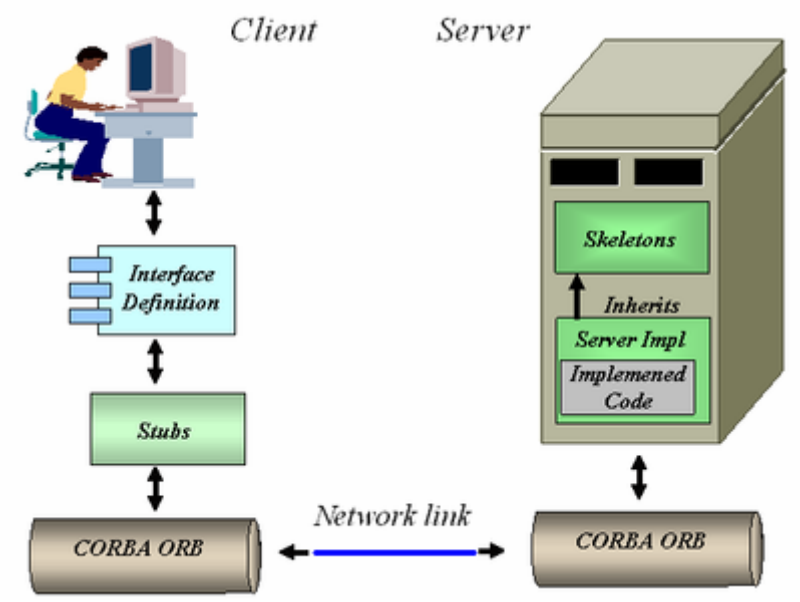

Figure 2: High Level View of CORBA Operation [39]

The main part of the architecture is the Object Request Broker (ORB). It is described best by Dr. Doug Schmidt, a Professor of Computer Science at Vanderbilt University. His description states that the ORB

> "provides a mechanism for transparently communicating client requests to target object implementations. The ORB simplifies distributed programming by decoupling the client from the details of the method invocations. This makes client requests appear to be local procedure calls. When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller"*[3]*

Figure 3 shows the internal architecture of CORBA and how it works.

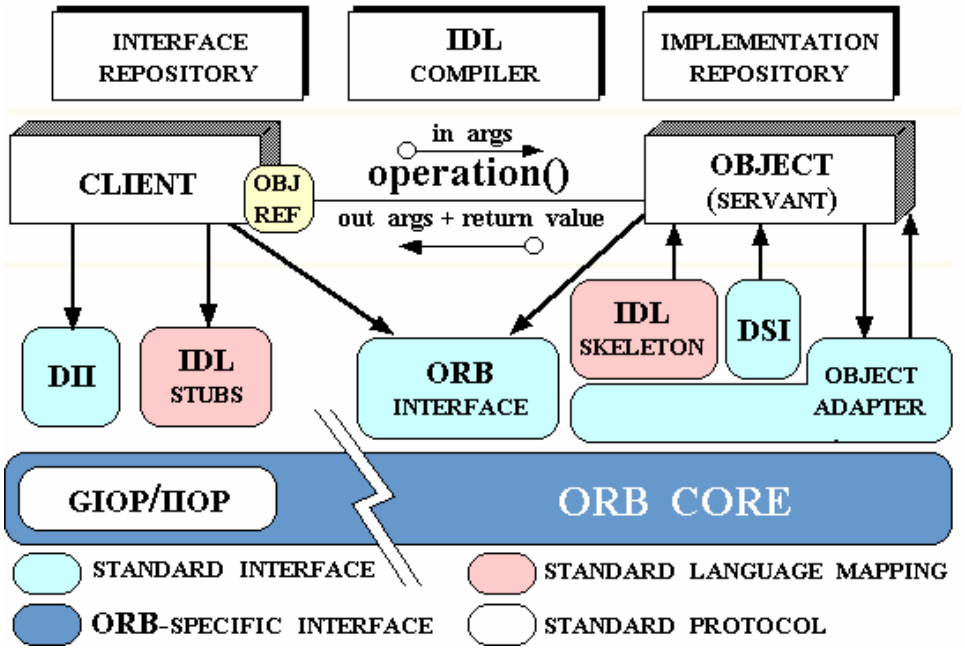

Figure 3: CORBA ORB Architecture [3]

As you can see from the diagram, the primary protocol used by CORBA is GOIP/IIOP.

From the OMG CORBA directory; “using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any other computer, operating system, programming language, and network can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network.”[2]

As CORBA is an OMG defined infrastructure and the structure of OMG is defined by those individuals representing enterprise companies, OMG primary market is for large scale middleware market where it is most important usage is that it must handle large number of clients, at high hit rates, with high reliability.

2.2.2 The ACE ORB (TAO)

TAO is an open-source implementation of a CORBA Object Request Broker (ORB) built using components and patterns from the Adaptive Communication Environment (ACE) framework. ACE is an open-source framework that provides components and patterns for developing high-performance, distributed real-time and embedded systems. TAO is best described by one of the companies that provide commercial support for this open-source implementation; Prismtech describes it as a “a high performance, real-time CORBA ORB end-system called The ACE ORB (TAO), which is open-source software that supports end-to-end quality-of-service assurance over high-speed networks and embedded interconnects.” [4]

2.3 Java

As CORBA is a standard defined by the OMG and implemented by numerous vendors including TAO, Java is not a standard but a Programming Language, which is used to implement a wide variety of middleware that will be discussed in the next chapter. Java's main concept of 'Write Once, Run Anywhere' means that Sun intended Java to be a platform-neutral language, allowing for software to be run on any number of computer architectures. This is achieved using the Java's Virtual Machine (JVM) as a abstraction layer above the computer architecture. The JVM allows for the Java Language to be compiled to byte code, which is interpreted by the Virtual Machine, which converts it to machine specific byte code at runtime thus allowing it to be platform independent.

2.4 Distributed Systems

Firstly, a Distributed System is defined as "a combination of several computers with separate memory, linked over a network, and on which it is possible to run a distributed application" [11]. This is not to be taken in the context of a Contact Center Server as it is possible to have a multi-node network of Contact Center Servers where activities within these Servers must be monitoring to allow for a resilient "always up" system. When this configuration is in place we want to ensure that all Contact Center Event data can be centralized at a single location allowing for a single point of contact for gathering this information.

Criteria for a successful Distributed System would be the following:

- Geographical decoupling
    - There is no need for the server / supplier and client / consumer to know about each other in the case of registering / deregistering or connect / disconnect. Everything should work seamlessly.
- Time decoupling
    - The server does not need to be active or up for the client and vice versa.
- Synchronization decoupling
    - Asynchronous messaging should be supported to prevent messages from blocking.

2.5 Web Services

Defined by Sun Microsystems a Web Service is an "application that uses open, XML-based standards and transport protocols to exchange data with clients". [21] They enable application to application communication via a web interface. There are many different groups of tools allowing for the development of Web Services each supplying the very same end results. Web Services are seen to be the next revolution of distributed programming allowing for both platform and language independent communication between applications. Web Services are not standardized but the Web Services Interoperability Organization (WS-I) has published specifications in an attempt to have interoperability between the many different approaches taken by different vendors.

Chapter 3 – Literature & Technology Overview

Chapter 3

The preceding chapter described the background of the technologies and how they are used in today's current environment. This chapter introduces and assesses some of the State of the Art technologies in the area of distributed systems. The technologies presented have been chosen for a variety of reasons, some are current and have been selected to flaunt the power of modern technology while older technologies have been chosen are they represent stable and standardized technologies.

The following will be presented in a thematic research manner and in order of chronological review.

3.1 CORBA Event Service

The OMG defined Event Service was first introduced in 1995 and was the OMG representation of a standard for pushing data from a supplier to a consumer. It defined a service for decoupling of suppliers of events from consumers of those events. Having a decoupled system meant that it provided a much more appropriate communications model for large scale applications that the typical client server paradigm of request / reply that CORBA could already supply via object calls. The Event Service defined a basic interface for the suppliers and consumers of events but it also defined the concept of an Event Channel to provide the decoupling of suppliers from consumers.

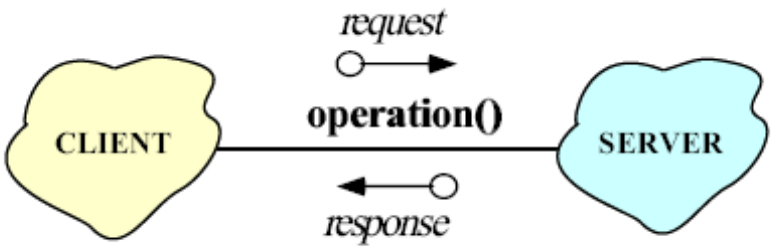

Figure 4: CORBA synchronous execution [5]

In a standard CORBA invocation the request results in the synchronous execution of an operation by an object. In short, the calling object will block until a response is received, thus being tightly coupled. In this situation, for the request to be successful, both the client and the server must be available. If a request fails because the server is unavailable, the client receives an exception and must take some appropriate action. Although CORBA

does support synchronous method invocation (AMI) it would still require the server to be operational. The Event Service removes this necessity as it decoupled entirely for the client / consumer as it defines two roles for objects: the supplier role and the consumer role. Suppliers produce event data and consumer's process event data. This is not a synchronous execution as the supplier only needs to pass the event data to the channel. To reference the OMG specification,

> "An event channel is an intervening object that allows multiple suppliers to communicate with multiple consumers asynchronously. An event channel is both a consumer and a supplier of events. Event channels are standard CORBA objects and communication with an event channel is accomplished using standard CORBA requests."

This type of specification is moving our initial client / server paradigm further towards a publish / subscribe paradigm where the client / consumer is completely decoupled from the server / supplier. This is where suppliers publish events and consumers receive only events for which they have subscribed. As you can see from this, neither the supplier nor the consumer knows of each others existence.

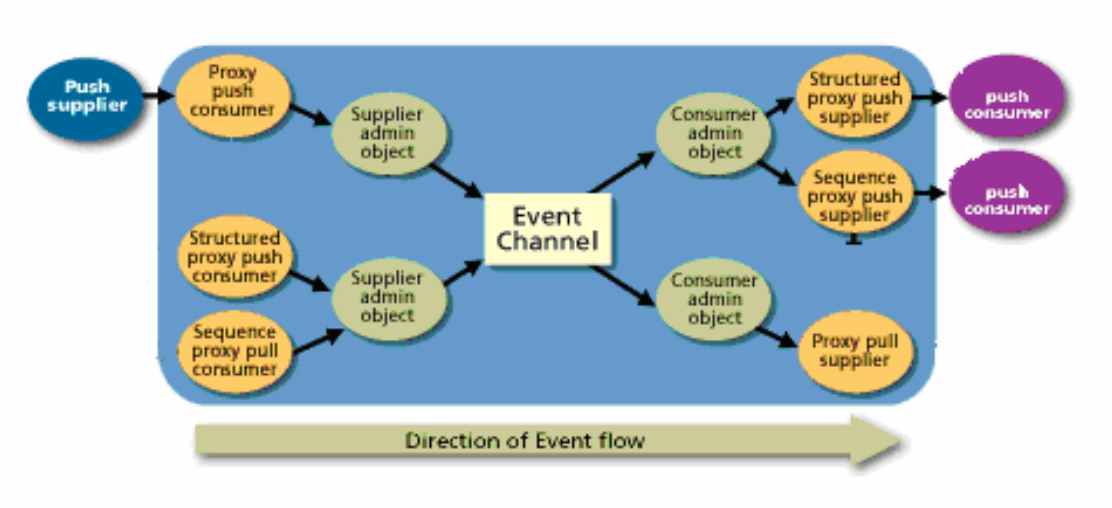

Figure 5: Typical Event Channel Usage [6]

As you can see in Figure 5 above, the relationship between suppliers and consumers are decoupled through the use of the event channel. The PushConsumer interface is a very simply defined interface shown below.

```
interface PushConsumer {
    void push (in any data) raises(Disconnected);
    void disconnect_push_consumer();
};
```

This is implemented by the supplier and allows the supplier to push events to the consumer. When the supplier invokes the push operation the data is sent ot the event channel who takes care of pushing the data to all the consumers that requested that information. The disconnect_push_consumer operation terminates the event communication; it releases resources used at the consumer to support the event communication. The PushSupplier interface is more simply in that it must only receive the events thus it is interface is

```
interface PushSupplier {
    void disconnect_push_supplier();
};
```

Again, the disconnect_push_supplier operation terminates the event communication; it releases resources used at the supplier to support the event communication. Figure 6 shows a typical example of the event service been used on a network with suppliers and consumers.

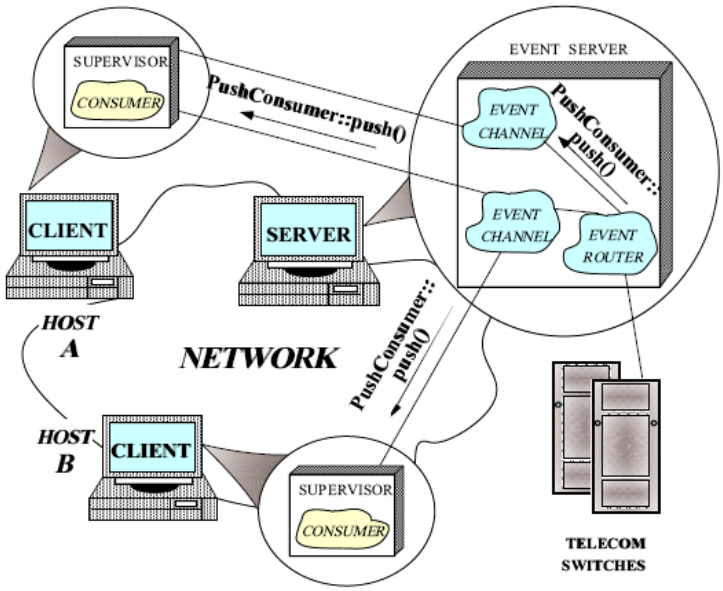

Figure 6: Network View of the Event Service [5]

The Event Channel provides the following benefit is:

- An event channel may provide many-to-many communication
- The channel consumes events from one or more suppliers, and supplies events to one or more consumers
- Subject to the quality of service of a particular implementation, an event channel
- provides an event to all consumers
- An event channel can support consumers and suppliers that use different communication models.[6]

3.2 CORBA Notification Service

The OMG defined Notification Service was first introduced in 1997 and was the extending of the initial OMG Event Service. To quote Dr. Doug Schmidt

> *"*The CORBA Notification Service provides a publish/ subscribe mechanism that is designed to support scalable event-driven communication by routing events efficiently between many suppliers and consumers, enforcing various QoS properties (such as reliability, priority, ordering, and timeliness),and filtering events at multiple points in a distributed system*"[7]*

The main difference here is the addition of Quality of Service (QoS) support along with event filtering mechanisms, which are not available with the Event Service. OMG defines a list of additions to the Event Service as:

- The ability to transmit events in the form of a well-defined data structure, in addition to Anys and Typed-events as supported by the existing Event Service.
- The ability for clients to specify exactly which events they are interested in receiving, by attaching filters to each proxy in a channel.
- The ability for the event types required by all consumers of a channel to be discovered by suppliers of that channel, so that suppliers can produce events on demand, or avoid transmitting events in which no consumers have interest.

- The ability for the event types offered by suppliers to an event channel to be discovered by consumers of that channel so that consumers may subscribe to new event types as they become available.
- The ability to configure various qualities of service properties on a per-channel, per-proxy, or per-event basis.
- An optional event type repository, which if present, facilitates the formation of filter constraints by end-users, by making information about the structure of events which will flow through the channel readily available.[9]

As previously mentioned, the Notification Service is a direct enhancement of the Event Service thus it remains backward compatible with the Event Service. The OMG defined Notification Service supports all of the interfaces and functionality supported by the OMG Event Service. The OMG specification tells us that the Notification Service also supports new features that are introduced by directly extending the interfaces defined by the Event Service. Both the original Event Service interfaces, and these new extended interfaces specific to Notification, are made available to Notification Service clients in order to preserve backward compatibility. [9] As the Notification Service is the same as the Event Service, all operational discussed in 3.1 are relevant to explaining this Service. As part of the Event Service specification the OMG defined that it supports two types of event communication: untyped and typed. Untyped communication involves transmitting all events in the form of CORBA Anys types. While untyped event communication is generic and easy-to-use, many applications require more strongly typed event messages. This is where the Notification specification has defined the Structured Event message type. With

the introduction of filtering, untypes events can be used for filtering events to the consumer but the Structured Event message type has a well defined data structure comprising of a header to allow for filtering options.

The most important part of the Notification Service is the Event Channel. It is role is to propagate events from suppliers to consumers. Once an event has been delivered to the Channel, the Channel takes responsibility for delivering it to each subscribed consumer.

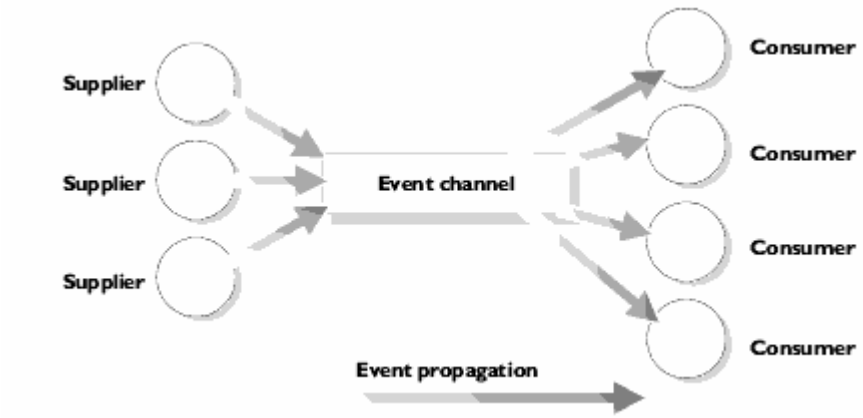

Figure 7: Event Channel [40]

The default behaviour of the Notification Channel is to deliver every event it receives to every subscribed consumer. This is also the behaviour of the Event Channel. However, the Notification Channel has the facility to filter events and thereby provide selective delivery. To use this facility, consumers specify which events they are interested in receiving by registering a filter expression with the Notification Channel. The Channel then applies the filter expression to each event to determine whether it should be delivered to that consumer [32]. The Figure below shows a high level architecture of the components that make up the

CORBA Notification Service and also includes the filters that may be applied to the Notification Service.

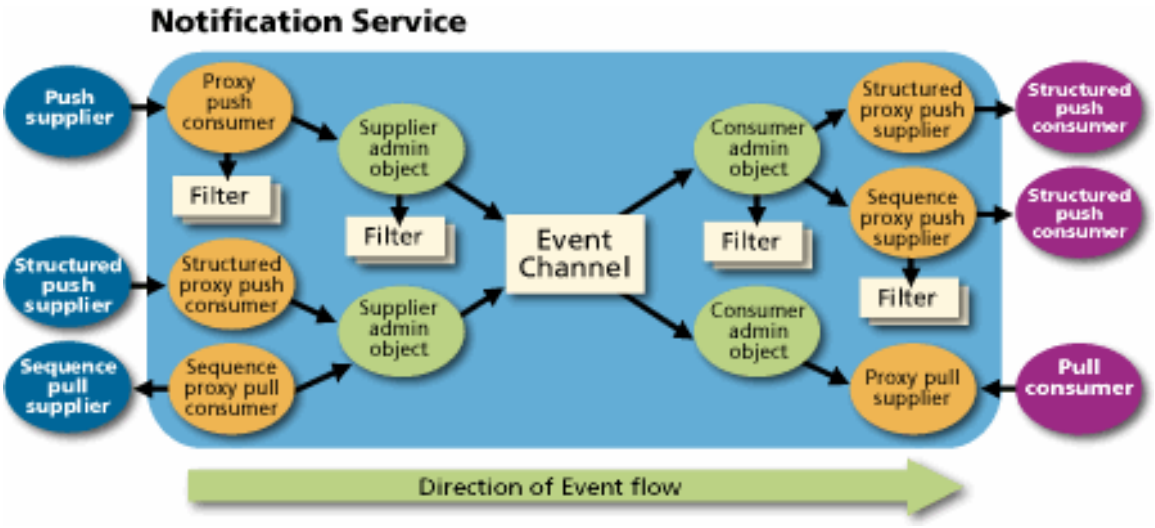

Figure 8: Notification Service components [6]

As you can see for the Figure above, the event filtering is one of the more notable additions to the Notification Service. Event filtering allows the consumer to subscribe to a precise set of events. These types of filters are known as forwarding filtering as they are used to determine which events should be forwarded to the next component in the architecture. There are two type of filtering included in the OMG specification for the Notification Service. The first is the forwarding filters mentioned above while the other is called mapping filters. These are filters which can be used to modify an events property as it passes through the component. As the filtering is done by the consumer (in the push architecture) performance is increased as only the events required are received by the consumer thus there is no transmission of invalid events to the client and no client side

filtering of received data. The network bandwidth is also decreased due to the fact that the events are not propagated forward to the consumer.

The Quality of Service (QoS) added by the OMG in the CORBA 3.0 specification is also a major feature for the Notification Service is an important factor that adds functionality. CORBA supplies these by means of interfaces that include the ability to get and set QoS properties at the event channel, admin, proxy and event levels such examples of these properties are:

- Timeout which allows a per event basis support to ensure that the message is discarded if not delivered in a said period of time.

- Priority which allows for an event to be given a special priority thus allowing for ordering of events to consumers.

There is also an Order Policy, which allows for Any, FIFO, Priority and Deadline ordering of events to the consumer. This same selection also applies to the Discard Policy.

The last addition to the Notification Service that makes it different from the Event Service is the mechanism provided to allow for offers and subscriptions. These allow the consumer to be notified whenever the set of offered event types change but also allow for the suppliers to be notified whenever the set types required by the consumer changes. This allows the creation of adaptive suppliers and consumers that can change their filtering constraints dynamically to adapt to changes in the system.

In summary, the Notification service like the Event service does have the consequence of application elements using the Notification Channel is that they no longer communicate directly with each other but indirectly via the Channel. Many benefit is arise from this decoupling, including the following:

- Supplier elements can deliver events at different rates than that at which consumer elements process them. Therefore, they can produce events at a different rate as well. In this respect, the Channel acts as a buffer, accommodating and levelling out peaks in an application's processing activity.
- The absence or unavailability of consumer elements does not prevent supplier elements from delivering events. In this respect, the channel allows an application to continue functioning even when parts of that application are unavailable.
- A supplier can send an event to every consumer by creating a single event and delivering it to the Channel. In this capacity, the Channel acts as a broadcast medium for the application. If filtering is used in the Notification Channel, then the Channel acts as a multicast medium.
- The identity of consumers is not needed by suppliers in order to reach them; only the identity of the Channel is needed by consumers and suppliers. Because of this, suppliers and/or consumers can be introduced to a system without requiring reconfiguration of existing suppliers or consumers in order to accommodate them. This has enormous benefit for large distributed applications.

Distributed application architectures can use these characteristics of decoupled communication to improve their performance, reliability, scalability, and adaptability [32].

3.3 Remote Method Invocation (RMI)

RMI is Java's introduction to distributed computing allowing Java objects running on different machines on a network or different processes on a machine to communicate via remote method calls. This concept is based on an earlier technology for procedural programming called Remote Procedure Calls (RPC's), which was developed in the 1980's and mainly used the 'C' / 'C++' programming language. It allowed a procedural program transparency when calling another object within a different process, be it a local or remote process. This provided a transparent communication layer allowing the programmer to concentrate on the required tasks of the application. There were some obstructions when dealing with local versus remote 'calls' to be made. When a 'call' was local, a reference to the object in memory would be passed to the local process while when the 'call' was remote; a copy of the object would be passed to the remote process.

Java, on it is implementation of remote calls covered some of the disadvantages of RPC such as IDL, expansion on data types, object passing and callback mechanisms. On it is initial design, the implementation was based on a Java object to Java Object distributed communication thus preventing interoperability with other languages was not available but this will be discussed later. Even dealing with the initial a Java object to Java Object communication, issues such as computer architecture had to come into play as Java's 'Write Once, Run Anywhere' required that communication between machines on a heterogeneous network meant that the communication model (RMI in this case) could no longer rely on the fact that the internal representation of data from one machine to another would be sufficient. This meant that data to be copied had to be converted to a platform

independent format, such as XDR (External Data Representation) and then converted back to an internal representation on the receiving side.

As RMI supports the Object Orientated architecture, it uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism. [13] To note at this point, serialization and marshalling are different processes, serialization refers to the converting of data to a byte stream while marshalling is the process of encoding the data used by RMI and transferring it across the network. William Grosso, an author for O'Reilly Authors writes the following when describing the Marshalling and Unmarshalling process of RMI as "Marshalling is a generic term for gathering data from one process and converting it into a format that can be used either for storage or for transmission to another process (correspondingly, unmarshalling involves taking the converted data and recreating the objects). In RMI, marshalling is done either via serialization or externalization. Marshalling and unmarshalling occupy a strange role in designing a distributed application. On the one hand, the means by which you perform marshalling and unmarshalling is a technical detail: once you've decided to send information to another process, how you do so shouldn't be a primary design consideration. On the other hand, it is a very important technical detail, and the way you do it can often make or break an application." [14].

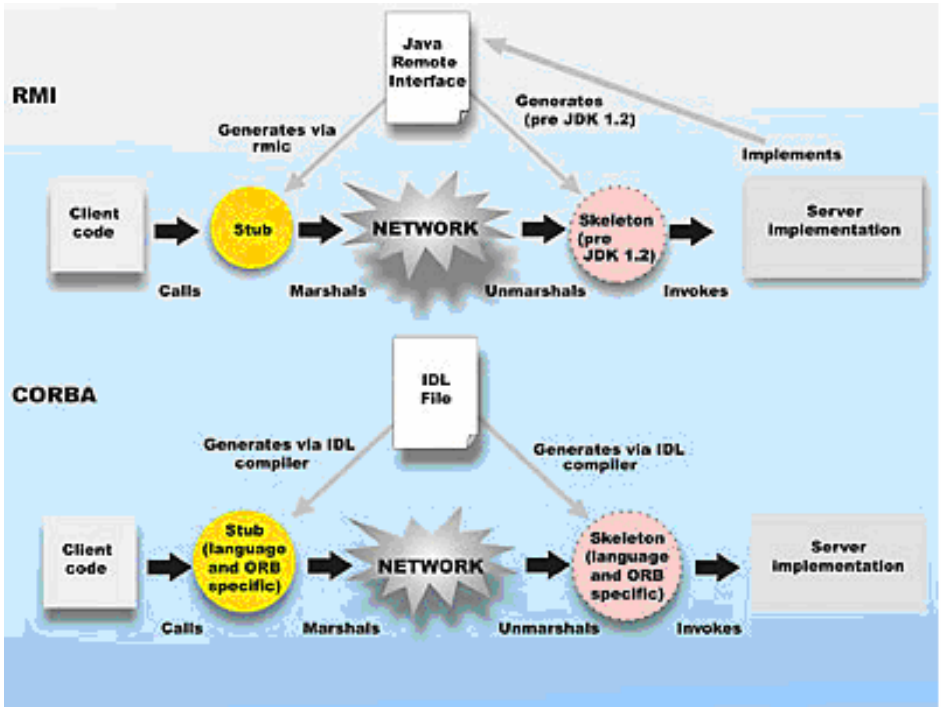

Figure 9: RMI versus CORBA Operation. [15]

From the figure above, at a High level view you can see the main concepts versus that of CORBA discussed earlier in this chapter. The concepts behind RMI and CORBA are the same since both have great similarity and achieve the same thing on the conceptual level. In the end, both allow the ability to invoke the features of an object server remotely, be it local on another process or remotely on another network machine.

From Sing Li, "In both RMI and CORBA, a distributed application is segregated into a server side and a client side that agree on how to communicate through a common interface (or set of common interfaces). In RMI, this common interface is expressed in terms of a Java language interface that extends the java.rmi.Remote interface. In CORBA, this common interface is expressed inside an Interface Description Language (IDL) file. Using this common interface, it is possible to use software tools to generate glue code that makes

the method invocation across the network relatively transparent to the application developer."

Although RMI is primarily Java and supports Java Object to Java Object communicate, Java's support for native languages may be used for C/C++ support therefore allowing for native language support. From java in a NutShell by O'Reilly they state that "using Java's Native Interface API, it is possible to wrap existing C or C++ code with a Java interface and then export this interface remotely through RMI" [16]. Along with supporting native language, RMI may also be used to support the CORBA IIOP protocol with RMI IIOP, which enables the programming of CORBA servers and applications via the `RMI` API. It is the RMI compiler, the rmic compiler can be used to generate the Java Remote Method Protocol (JRMP) as the transport, or work with other CORBA-compliant programming languages using the Internet InterORB Protocol (IIOP). The main disadvantage of this is that the server must be written in Java to generate the CORBA stubs using the rmic compiler, which prevents this from been used in legacy systems but does allow RMI greater functionality as CORBA is feature rich, fast and a supported specification.

3.4 Java Message Service (JMS)

The Java Message Service more resembles the CORBA as it too it a specification. It was developed under the Sun's Java Community Process as JSR 914 and the specification was first published in August 1998 and released in April 2002. The specification defines a messaging service that is implemented by Java in the JMS API and is defined by Sun as Messaging Oriented Middleware (MOM) API for sending messages between two or more clients. MOM is universally recognized as an essential tool for building enterprise applications [17].

It is important when building enterprise applications that the system be loosely coupled to prevent inter dependencies in the application and this is provided by MOM. When we talk of Messaging Systems, there are generally two types:

1. Point-to-Point
2. Publish / Subscribe

A Point-to-Point messaging model allows suppliers to send messages to a message queue; the sender intends the message for a single consumer. This resembles the CORBA Event Service discussed earlier in that a supplier supplies messages to the consumer through a channel / queue. In the JMS Specification, when a consumer connects to the queue it will receive all messages not yet consumed. In this model, one client consumes a message and acknowledges that the message was received.

The Publish / Subscribe messaging model allows suppliers to publish messages to a message topic. This type of messaging resembles the CORBA Notification Service discussed earlier in that it allows zero or multiple consumers to receive a message if they

are interested in the message. In the CORBA Notification Service, this is done through filtering as the consumer can supply a filtering object on the types of events required.

JMS in both types of messaging models discussed has a message that contains a

- header which contains the destination address and sending time;
- properties which allow for filtering of messages on the client side
- and the body of the message

Like the CORBA Notification Service, the filtering of the message is done at the Server side although the Notification Service provides numerous layers of filtering as discussed.

Figure 8 shows the JMS Programming model showing the two different JMS models. The point-to-point interfaces are shown on the left and the publish/subscribe interfaces are shown on the right. The arrows leading from top to bottom in the figure represents the typical steps that a JMS developer performs developing client applications:

1. Resolve a connection factory and a destination from JNDI. A destination is either a queue or a topic.
2. Create a connection using the connection factory.
3. Create a session with the desired properties from the connection.
4. If the application is a supplier, create a MessageProducer; if the application is a consumer, create a MessageConsumer from the session.
5. Start to send or receive messages using the producer or consumer object. A producer will use the session to create different kinds of messages.

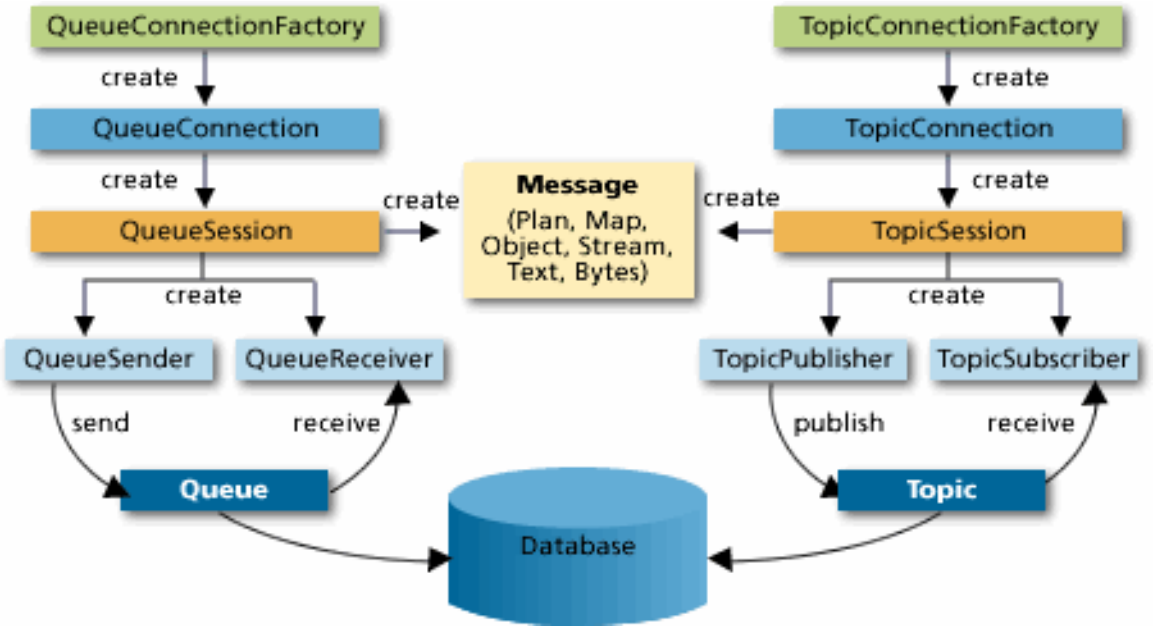

Figure 10: The Java Message Service programming model [6]

As we have discussed the JMS model, the elements that make up JMS architecture:

- JMS provider
  - The entity that implements JMS for a messaging product.
- JMS clients
  - Programs or components, written in the Java programming language, that produce and consume messages.
- JMS Message
  - JMS defines a set of message interfaces.
  - Clients use the message implementations supplied by their JMS provider.
- JMS producer
  - A JMS client that creates and sends messages.
- JMS consumer

  - A JMS client that receives messages.
- JMS Domain
  - Messaging products can be broadly classified as either point-to-point or publish / subscribe systems.

As JMS is a specification and is open generally to Java development, there are a number of different implementation mainly ActiveMQ by Apache. This is an Open Source implementation of the JMS Specification and is widely used in the Enterprise Application industry.

Steve Trythall explains to us that "JMS supports six different kinds of messages, which are used to carry different types of payload. The header of a message is the same regardless of the payload, which means that filtering is the same for all six message types. A message supports a number of properties to set priority, reliability, and other QoS properties, which will be interpreted and handled by the JMS server". The different types of message are

- ByteMessage
- MapMessage
- ObjectMessage
- StreamMessage
- TextMessage

The JMS specification is very clear that it does not support Load Balancing, Error Notification, Security, Repository or a wore protocol. This in turn means that there can be alot of differences in the implementation of the specification. The JMS specification states

that “It is expected that JMS providers will differ significantly in their underlying messaging technology. It is also expected there will be major differences in how a provider’s system is installed and administered”.

Also provided by JMS is the guaranteed delivery of message which is similar to that provided by the CORBA Notification QoS properties of a message. In JMS this is provided using a store and forward mechanism. This mechanism means that the underlying message server will write the incoming messages out to a persistent store if the intended consumers are not currently available and delivery these message when the intended consumers reconnect, dealing with any loss of connection due to network issues that may occur.

3.5 Open Grid Services Infrastructure (OGSI) Notification

OGSI is a more recent specification developed by the Global Grid Forum more recently renamed to be the Open Grid Forum, which was published in June of 2003. The Open Grid Forum define there mission to "accelerate grid adoption to enable business value and scientific discovery by providing an open forum for grid innovation and developing open standards for grid software interoperability". The purpose of the specification was to define mechanisms for creating, managing, and exchanging information among entities called Grid services. A Grid Service is part of Grid computing, which is a group of resources that can be flexibly and dynamically allocated and assessed in solving problem that required a large number of resources such as processors, storage or bandwidth. The specification for OGSI provides only the basic level of function allowing Grid Services to be created, managed, discovered and destroyed but does not relate to resources.

The OGSI specification is where Grid technologies are integrated with Web Service mechanisms to create a distributed system framework based on the *structure of the OGSI*. The Web Service definition language (WSDL) interface, which is used to provide XML defined protocols in a structure format, is also seen as a set of conventions by which a Grid Service must conform. A Grid Service therefore can be viewed as a Web Service since it conforms to a WSDL interface. Karl Czajkowski from Globus, one of the largest implementers of OGSI tells us in his paper that "at the core of OGSI is a Grid service [Physiology], a Web service that conforms to a set of conventions for such purposes as service lifetime management, inspection, and notification of service state changes" .

From the OGSI Specification, Notification is described as a framework which (OGSI specification (2003)) "*allows for both direct service-to-service notification message delivery and the integration of various intermediary delivery services*". The intermediary delivery services might include messaging services and message filtering services. The specification defines the purpose of notification as being a mechanism "*to deliver interesting messages from a notification source to a notification sink*" where the source represents a service and a sink a client. The framework allows for asynchronous, one-way delivery of messages and any services implementing this must conform to the framework's interface. A sink will subscribe with the service for receiving these notifications, which is also accompanied with an XML definition of the message types requested allowing for service side filtering of messages thus preventing unnecessary bandwidth usage.

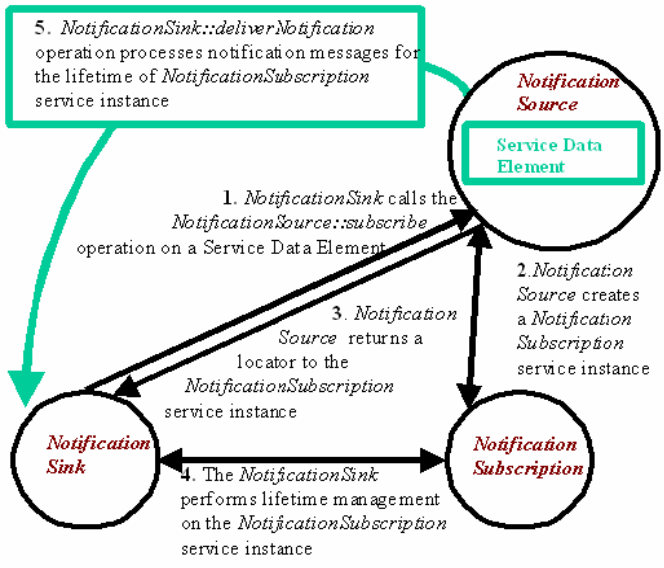

Figure 11: OGSI Notification [20]

Figure 11 shows a typical client subscription with a service. The request from the client causes the creation of a Subscription Grid Service Instance that can be used by the client to manage the subscription and to discover it is properties. The service returns a Locator

object to the client upon subscription allowing for the completion of the subscription process. The OGSI specification (2003) describes each of the components involved:

- A *notification source* is a Grid service instance that implements the NotificationSource portType, and is the sender of notification messages. A source MAY be able to send notification messages to any number of sinks.
- A *notification sink* is a Grid service instance that receives notification messages from any number of sources. A sink MUST implement the NotificationSink portType**,** which allows it to receive notification messages
- A *notification message* is an XML element sent from a notification source to a notification sink. The XML type of that element is determined by the subscription expression.
- A *subscription expression* is an XML element that describes what messages should be sent from the notification source to the notification sink. The subscription expression also describes when messages should be sent, based on changes to values within a service instance's serviceDataValues.
- In order to establish what and where notification messages are to be delivered, a *subscription* request is issued to a source, containing a subscription expression, the locator of the notification sink to which notification messages are to be sent, and an initial lifetime for the subscription.
- A subscription request causes the creation of a Grid service instance, called a *subscription* that implements the NotificationSubscription portType. This portType MAY be used by clients to manage the (soft -state) lifetime of the subscription, and to discover properties of the subscription.

3.6 Web Service (WS) Specifications.

There are a number of Web Service (WS) specifications which are designed to provide a rich set of tools to provide security in the Web Services environment. All of the specifications where published by the World Wide Web Consortium (W3C) March of 2006 adding to the original specification for WS-Eventing submitted by IBM, BEA Systems, TIBCO Software, Sun Microsystems, Computer Associates and Microsoft but defined in August 2004. The first to be discussed is the WS-Eventing specification, which is used to provide secure, reliable, and/or transacted message delivery and to express Web service and client policy.

3.6.1 WS-Eventing

WS Eventing specification is defined by the W3C as "a protocol that allows Web Services to subscribe to or accept subscriptions for event notification messages". As discussed in the description of the other topics in this chapter, the same relates in Web Services that may want the ability to receive messages when events occur in other services or applications. To enable a Web Service to receive these messages, a mechanism must be put in place that allows a Web Service to register for receiving these messages. This is the purpose of this specification. In general terms it defines a protocol that will allow a Web Service acting as a subscriber to register a subscription with another Web Service acting as a message (event) source. We are provided with a publisher/subscriber communication model that allows decoupling of the subscriber from the message (event) source. These models allows for a push and pull mechanism to be implemented as we seen in the CORBA specification for the CORBA Notification Service earlier, which is also provided using JMS, which is

not a standard. However, the specification only defines the mechanism for a Push model to be implemented. It is an asynchronous push mechanism meaning that we do a “fire and forget” on the message (event) been sent.

This may lead to questions about the Quality of Service (QoS) provided by the specification but other WS Specifications such as WS-Reliability and WS-Transaction deal with this issue and may also be implemented as the Eventing Specification defines a Delivery Mode which acts as an extension point allowing for the creation of tailored delivery modes.

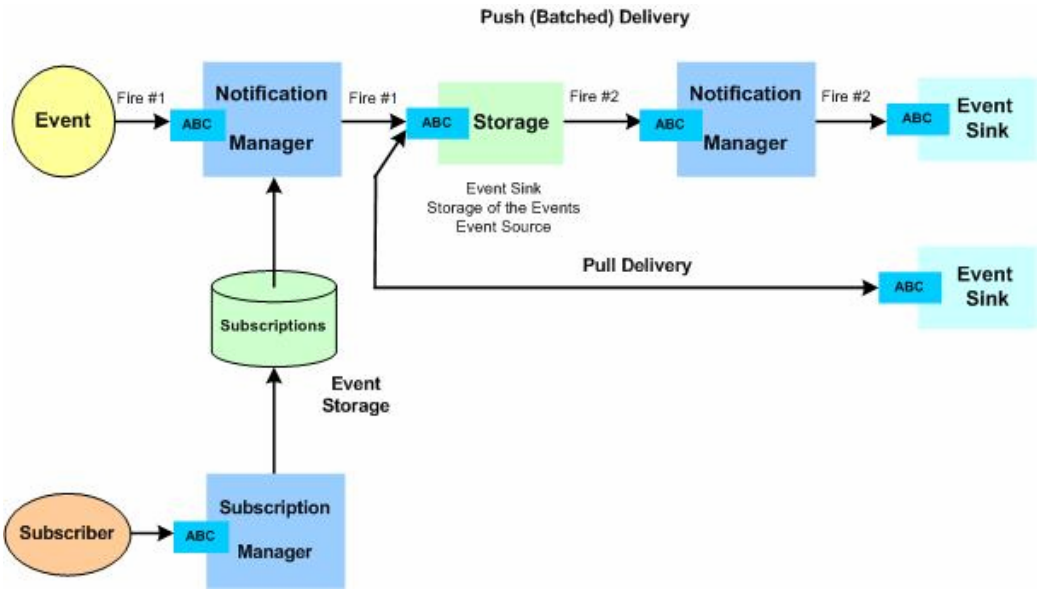

Figure 12: Delivery process for events in WS Eventing [37]

As with the QoS issue security of messages on the communication network may be provided using the WS-Security Specification. From the Microsoft Developer Network on Security considerations, it states that “Different security mechanisms may be desired depending on the frequency of messages. For example, for infrequent messages, public key technologies may be adequate for integrity and confidentiality. However, for high-

frequency events, it may be more performant to establish a security context for the events using the mechanisms described in WS-Trust and WS-SecureConversation".

3.6.2 WS-Notification

This Specification was written by IBM, Sonic, TIBCO, Akamai, SAP, CA, HP Fujit isu and Globus in March 2004 which is directly competing with the WS-Eventing Specification. It consists of the WS -BaseNotification, WS-BrokeredNotification and WS-Topics that define the mechanisms to be used for notification producers, consumers and brokers. The components are described *Building Web Services with Java as:*

- Publish-Subscribe Notification for Web Services—A whitepaper that defines the base concepts, roles, and so forth within the WS-Notification set of specifications.
- WS-BaseNotification—A specification that defines the basic interfaces in WS-Notification. These include Web service interfaces to describe the behavior of producers of notification messages, consumers of notification messages, and subscriptions that relate producers with consumers.
- WS-Topics—A specification that defines topics, a means to categorize notifications.
- WS-BrokeredNotification—A specification that defines the Web service interface to an intermediary or message broker Web service.

As with the WS-Eventing specification the primary goal of WS-Notification is to allow for Web Services to be notified off events that occur. WS-Notification encapsulates all the same functionality as WS-Eventing as it is a competing specification offering the ability to pause and resume subscriptions on top of the WS-Eventing Specification. The two

specifications will not provide any interoperability with each other but provide the same transport layer and message structure. A paper called Publish-Subscribe Notification for Web services describes the WS-Notification approach as “The Event-driven, or Notification-based, interaction pattern is a commonly used pattern for inter-object communications. Examples exist in many domains, for example in publish and subscribe systems provided by Message Oriented Middleware vendors, or in system and device management domains. This notification pattern is increasingly being used in a Web services context”. The authors continues to say that “in the notification pattern a Web service, or other entity, disseminates information to a set of other Web services, without having to have prior knowledge of these other Web services”. A list of the characteristics of this pattern is listed below:

- The Web services that wish to consume information (which we call NotificationConsumers) are registered dynamically with the Web service that is capable of distributing information. As part of this registration process the NotificationConsumers may provide some indication of the nature of the information that they wish to receive.
- The distributing Web service disseminates information by sending one-way messages to the NotificationConsumers that are registered to receive the information. It is possible that more than one NotificationConsumer is registered to consume the same information. In such cases, each NotificationConsumer that is registered receives a separate copy of the information.
- The distributing Web service may send any number of messages to each registered NotificationConsumer; it is not limited to sending just a single message. Note also

that a given NotificationConsumer may receive zero or more NotificationMessages throughout the time during which it is registered.

3.6.3 WS-EventNotification

This is the integration of the WS-Eventing and the WS-Notification Specifications. The companies involved in the creation of this new specification are HP, IBM, Intel, and Microsoft. In a paper titled Toward Converging Web Service Standards for Resources, Events, and Management written by Kevin Cline we learn that the new capabilities that WS-Notification supports are:

1. Subscription policy – WS-Eventing and WS-Notification introduce the concept of subscribing to resource/services for events. Different services/resources may have different approaches to implementing subscriptions and notifications. Subscribers may wish to set different requirements or directive on subscriptions. WS-EventNotification defines concrete policies that allow a resource/service to describe it is approaches for subscriptions and subscription management, and allows a subscriber to specify directives to the event source. This allows extensibility for WS-EventNotification and capability description that other specifications can use.
2. Richer filter languages – WS-Eventing introduced a simple filtering language. The language allows a subscriber to specify a filter that describes which events the subscriber wishes to receive. WS-EventNotification introduces a richer filter language, which enables functions that WS-Notification supports.
3. Wrapped Notification – WS-Eventing describes events as output operations/messages on a WSDL portType. The output messages correspond to

input message/operations on the event sink. Some scenarios, especially those building on existing publish/subscribe systems require an explicit notification message that contains the event data. This is 'wrapped' notification. The output message/operation for the event is contained within an outer notify operation /message. Wrapped notification also provides a generic interface for receiving notifications. This allows defining subscribers that can receive events from any notifier. There is no need for an operation that matches the output operation of the event emitter.

4. Subscription resources – WS-EventNotification, like WS-Notification, treats a subscription's state as a resource in WS-ResourceTransfer. A subscription may have a lifetime, and the subscriber can use Get', Put', and Delete' to read or update the subscription's state, for example to change a filter or expiration lifetime. This better integrates concepts defined in WS-Eventing with similar concepts in WS-ResourceTransfer, and WS-ResourceFramework.
5. Pausing subscriptions – WS-EventNotification, like WS-Notification, introduces the notion of 'pausing' a subscription. This allows for the temporary halting the flow of Notifications to a particular subscriber. The exact QoS properties, e.g. whether the new notifications are cached or simply ignored, will be controlled by the Subscription Policies.

The new specification is a superset of the existing specifications and provides backward compatibility to these specifications meaning that these specifications will be continued to be supported.

SECTION 3: TECHNOLOGY EVALUATION

Chapter 4 –Technology Evaluation

Chapter 4

The preceding chapter described the various State of the Art technologies that can be used for Notification of Events in different environments. This chapter will group each of those technologies into specific groupings and analysis these technologies from the perspective of that group. It will be used to show the different abilities and advantages of each of the technologies discussed previously.

4.2 CORBA Services.

In the preceding chapter we discussed the CORBA Event and CORBA Notification Services. CORBA, the Common Object Request Broker Architecture, is OMG's open, vendor-neutral architecture and infrastructure that computer applications use to work together over networks. It was first released in 1991 with the promise of a platform and software independent architecture for writing distributed, object-oriented applications. In basic terms the technology is best described to allow a distributed, heterogeneous collection of objects to interoperate. Both the Event and Notification services were defined by the Object Management Group (OMG) with the Event Service having been introduced in 1995 while the Notification Service followed on 1997. The Notification Service being the more recent version of the specification extends the functionality of the Event Service with features such as event filtering, structured event types and Quality of Service properties.

The concept of CORBA allowing for both platform and language independence makes it very powerful and the specifications listed above prevent the synchronous execution of an operation by an object. The introduction of the COPRBA Event and the CORBA Notification Services that are implemented by the CORBA vendors allows for the decoupling of communication between objects by providing the roles of suppliers and consumers as defined in the OMG specification of these services. In an overview of both services however, the Notification Service provides all the same functionality as the Event Service but with the addition of simplicity in using the structured events, in built filtering and the ability to ensure Quality of Service. Both services also provide the ability for both Generic and Types communications allowing for any CORBA object to be used in Generic

type event communication model and for structured message types defined as part of the specification to be used in the Typed event communications model. Both services also provide the push and pull communication models allowing for CORBA clients or consumers to interact with the service(s) differently depending on the operation and use of the application. An example of the push mechanism is shown below:

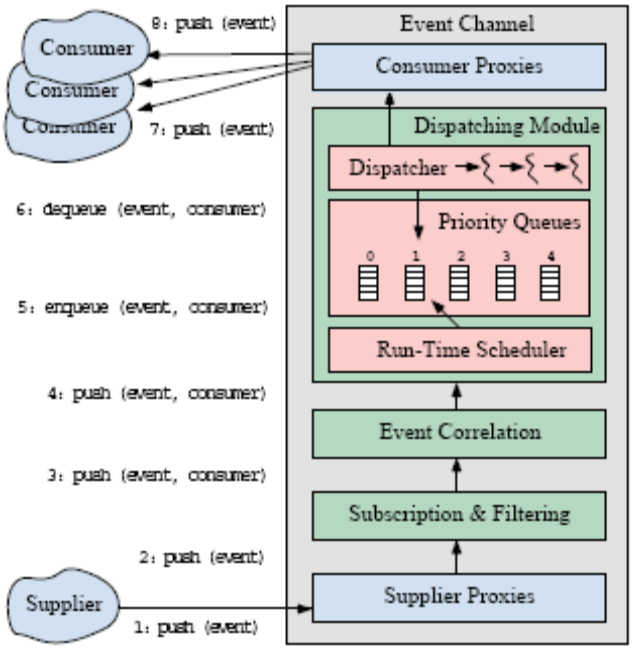

Figure 13: CORBA Service push model [5]

The primary advantages provided by these services are:

- Anonymous consumers/suppliers
    - Publish and subscribe model
- Group communication
    - Supplier(s) to consumer(s)

- Decoupled communication
    - Asynchronous delivery
- Abstraction for distribution
    - Can help draw the lines of distribution in the system
- Abstraction for concurrency
    - Can facilitate concurrent event handling

As you can see from the above list, both services allow for multiple suppliers of events along with the expected feature of multiple consumers meaning that in a multi-node system a single service can be used for supplying events to consumers providing a many-to-many communication model.

The add advantages provided by the Notification Service are listed below but these are also seen to be the limitations of the Event Service.

- Structured Events
    - Ability to include filtering and QoS parameters that influence delivery of events.
- Filtering
    - Built ion event channel filtering allowing consumers to specify which events they are interested in receiving.
- QoS Properties

- Allow both Suppliers and Consumers of events to configure properties associated with delivery including reliability, priority, ordering and time on a per-channel, per event basis.

- Subscription information
  - Suppliers are aware of the events requests by the Consumer and therefore only need to supply those events. The same applies to the Consumer of events; the consumer is notified if a Supplier is providing new event data types.

From a paragraph by Dave Bartlet when he describes the CORBA Notification Service "Using the Notification Service, your applications can be built more effectively by leveraging a proven middleware solution that is standards-based, flexible, and optimized for high performance and scalability" best describes my thought on using this service over the Event Service provided by OMG. This might seem to be enough but the OMG has a new specification introduced more recently called the "Notification / JMS Interworking" that will be discussed later in this chapter.

4.2 Java Technology.

The preceding chapter introduced the Java's Remote Method Invocation (RMI) and Java's Message Service (JMS). From Sun they describe Java's Remote Method Invocation as enabling "the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines". This was Java's answer to the older Remote Procedural Calls (RPC) used in the C and C++ programming language. Sun describe the Message Service as

"a messaging standard that allows application components based on the Java 2 Platform, Enterprise Edition (J2EE) to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous".

When we compare the two, there is a difference in the usage of the two implementations. JMS carries on from our CORBA discussion where it is a predefined specification that may be implemented by a service, thus providing that service. RMI on the other hand is the ability for developing distributed systems by providing a mechanism to compile and communicate via the JMRP protocol or the OMG defined IIOP protocol. Like RPC, RMI provides the ability for a program to seemingly carry out operations as if the distributed code was running in a single process. The main advantage apart from the ability to provide lightweight, easy to implement and easy to maintain distributed applications is that the Java based components and services have the ability to communication with any other language that is supported by CORBA using the IIOP protocol but the main disadvantage when it comes to the messaging structure required is that RMI has a synchronous communications model that means a tightly coupled system design between the Supplier of events and the Consumer of those events. This in turn brings the following problems:

- Exceptions when the producer is not available or out-of-service.
- The consumer is always in a waiting state until the server is finished processing the events and there is only communication with one server unless registered with multiple servers individually, which must be all managed connections.
- RMI marshalling and de-marshalling of network calls will cause latency where speed is essential

JMS like the CORBA services provide all this as part of it specification and thus provides a simpler service that can be implemented by our Suppliers and Consumers in a loosely coupled system. By using an implementation of JMS provided by multiple vendors we can utilize this work allowing us to deal more independently with developing our application.

4.4 Web Technology

Although the Open Grid Services Infrastructure (OGSI) is a more recent publication, it is now considered to be obsolete having being replaced by the Web Service - Resource Framework (WS-RF). OGSI was a set of extensions to Web Services that provided abilities for Web Services such as the ability for a Web Service to have “stateful” interactions between clients. Essentially, it provided extensions to the Web Services but with the definition of WSDL 2.0 WS-RF emerged as the evolution of OGSI partitioning the framework into five distinct, composable specifications. These specifications are attempting to define a generic and open framework for modeling and accessing stateful resources using Web services in the same manner as the WS specifications for Eventing and Notification. These are shown in the table provided by Karl Czajkowski from Globus, one of the largest implementers of OGSI tells us in his paper “From Open Grid Services Infrastructure to WSResource Framework”. The table is provided in the next page.

Table 1: WS Specifications

Name	Description
WS-ResourceProperties	Describes associating stateful resources and Web services to produce WS-Resources, and how elements of publicly visible properties of a WSResource are, retrieved, changed, and deleted.
WS-ResourceLifetime	Allow a requestor to destroy a WS-Resource either immediately or at a scheduled future point in time.
WS-RenewableReferences	Annotate a WS-Addressing endpoint reference with information needed to retrieve a new endpoint reference when the current reference becomes invalid.
WS-ServiceGroup	Create and use heterogeneous by-reference collections of Web services.
WS-BaseFault	Describes a base fault type used for reporting errors.

The WS-RF and the other technologies listed in the preceding chapter will not compete, they complement each other. The three WS-* specification discussed however do have a lot of similarities; WS-Eventing and WS-Notification are competing specifications but notification is said to be very difficult to implement while eventing is simpler to implement. In saying this, I am not looking to implement the specification but to use the service that provides the specification. As the WS-Notification specification is implemented in the enterprise service bus provided by Globus and Apache ServiceMix, this makes it a more attractive option as these are proven and reliable servers. The relatively new introductions of these specifications and finding it difficult to find reliable implementations of the ideal WS-EventNotification, this makes this technology very difficult to use as competing specifications could lead to a lot of future issues as the currently defined specifications of Eventing and Notification are not interoperable in a project that requires speed and scalability along with reliability and Quality of Service, these specifications will not suit the requirement.

4.4 CORBA versus Java

Looking at the challenging technologies of the CORBA Notification Service and the Java Message Service both of which can be described as Message Oriented Middleware (MOM) providing asynchronous QoS delivery of messages along with platform independence and both of which are proven a reliable services. At this stage, both services providing different implementations of two popular specifications need to be analysis to which best suit is as a service that can be implemented successfully within our server.

In starting the analysis of these technologies we will first look at the requirements required and further discuss how these technologies can be implemented in meeting these requirements. The final paragraph will then look at the best approach before design and implementation are considered.

The requirements listed at the start of this paper included:

- Guaranteed delivery of event message to any number of client applications
- Allow client applications language and platform independence
- Speed and Reliability in the delivery of messages.
- Interoperability between difference technologies.
- Standardized specifications implemented in stable and reliable services.

The ability for both services to act as a standalone service in a multi-nodal system was not an initial requirement but adds valued features to the system design along for large scale grid based distributed servers.

The Java Message Service (JMS) is a Sun specification which differs from the Notification specification since the OMG specification covers both the client interface and the messaging engine while the JMS specification defines and standard API with no mention of the implementation of the messaging engine, it is architecture or it is communication protocol which means that vendors implementation differ. This does not defer away from the features provided by the service but it does add the issue of interoperability between different implementations supplied by different vendors. This is not the case for the OMG defined specification where difference vendor implementations are interoperable.

JMS is widely implemented by a large number of vendors providing platform independence and asynchronous delivery of event messages. This in turn provides the ability for supporting large number of client connections along with the guaranteed delivery of event message while utilizing network bandwidth. Over the past number of years, this has been a growing messaging infrastructure due to it is tight integration into the J2EE architectures and the growing of web based architectures in the Java environment, which makes it a very powerful MOM.

CORBA on the other hand is a legacy specification and is widely implemented by various vendors and like JMS is also available as a Open Source implementation cutting the cost of adding these services. CORBA Notification Service also provides providing platform independence and asynchronous delivery of event messages with the advantages of this been the same as those for JMS. The Notification Service however does inherently gain the ability to be language independent since it is a CORBA implementation.

So as we can see at this point, JMS and the Notification can provide the same features with JMS failing to meet the language independence requirement but having a simpler interface and the ability for point-to-point communications, which is not a feature of the Notification Service. The last issue which is a powerful requirement is that of speed. In reviewing the two services the Notification Service does provide a much quicker delivery of message with message delivery been ten fold that of the JMS service when tested with a single Supplier Consumer test-bed. This added to the fact that the Contact Center Server is primarily C++ based makes the Notification Service a better choice for implementation since a JNI bridge would be required in getting C++ event into the Java environment for use with the JMS.

As our Contact Center Server moves forward however, we are starting to see that much of our future services will be Java based components, these will have the ability to use CORBA and a Java implementation of the CORBA specification can replace the C++ implementation as it provides the interoperability between implementations as it is defined in the CORBA specification. Along with that the ability for JMS messages is also available as the OMG have noticed the growing usage of JMS and have provided a 'Notification / JMS Interworking' specification that will allow for JMS messages to be interoperable with the Notification Service allowing for future work to integrate with the J2EE architecture and not limiting the preferred implementation.

Chapter 5 – CORBA Vendor Implementation

Chapter 5

From the preceding chapter we evaluated the various technologies referenced in Chapter 3 and discovered that the CORBA Notification Service satisfied the requirements needed by the Contact Center Server. This chapter will review the design and implementation details for implementing that service, how the messages are delivered to the Notification Service and how the clients can receive those messages in a reliable way.

5.1 Evaluation of the Notification Service

The decision to use the Notification Service leads to the finding of the suitable Notification Specification vendor implementation. As discussed in Chapter 3 in the Literature and Technology Overview the Notification specification is part of the CORBA specification, which is specified by the OMG. This in turn leads to multiple difference vendors having implementations of CORBA and the Notification Service, which in turn leads to the decision of which vendor implementation should be used in the implementation of the new service

5.1.1 Vendor Implementations

There are currently a large number of CORBA implementations on the market. The primary leaders are listed below:

- Inprise Visibroker
- Iona Orbix
- The ACE ORB
- JacORB

5.1.1.1 Inprise Visibroker

This is one of the leading commercial CORBA ORB's that supports all of the most popular programming languages such as Java, C++ and .NET. It is fully compliant with the CORBA 2.6 specification. Visibroker also provides a rich set of tools for implementing CORBA and a rich set of services that support the OIMG specifications for CORBA services. VisiBroker was the first CORBA ORB to support the Java language.

5.1.1.2 Iona Orbix

This is another one of the leading commercial CORBA ORB's that also supports Java, C++ and the .NET environment. It is fully compliant with the CORBA 2.1 specification and was released in 2005.

5.1.1.3 The ACE ORB

This is one of the leading free ORB's that supports the C++ language. . It is fully compliant with the CORBA 2.6 specification and is always under continued development with CORBA 3.0 specification features been released into the most recent version of 1.5. OCI states that "The ACE ORB (TAO), pronounced "dau", is a CORBA V3.0 compliant" implementation. [35] TAO also has full compliance with the "Defense Information Infrastructure Common Operating Environment (DII COE)" defined by the U.S. Defense Information Systems Agency (DISA). DII COE was developed in late 1993. DII COE was designed to eliminate duplication of development (in areas such as mapping, track management, and communication interfaces) and eliminate design incompatibility among Department of Defense (DoD) systems. [36]

5.1.1.4 JacORB

This is one of the leading free ORB's which supports the Java Language. It is fully compliant with the CORBA 2.3 specification and is always under continued development. It is a 100% pure Java implementation of the specification

5.1.2 Cost of Ownership

The cost of ownership will mainly apply to the commercial products which require payment not only for the development tools but also for the runtime files supplied with the implementation of the services provided by using that vendor implementation. Both TAO and JacORB are both Open Source projects allowing for free distribution of the runtime and work with all development tools currently available under different environments.

5.1.2.1 Costs

Royalty costs relate to the cost of distribution of the CORBA runtime files required by services that implement the vendor's implementation of the CORBA specification. It is stated that the cost of Iona Orbix "costs 5000 dollars on Unix systems PER developer. The multi-threaded version costs 6500 dollars a seat! Other platforms typically cost 2500 dollars for a developer's license. In addition, run-time licenses cost around 100 - 200 dollars for each machine which will be making use of the developed CORBA technologies." This also contains information that states that Visibroker costs "close to 5000 dollars a seat for a developer's license"[29] and there are also run-time licenses costs involved

5.1.2.2 Support

As with all large scale server distributions, every company must support the software that it ships so Open Source Software must be supported if it is to be used at industry level to ensure customer satisfaction. In the case of commercial software, this is naturally supported but at large costs in both development and distribution. For our selected Open Source software there are a number of commercial software companies who support the

distribution of both TAO and JacORB ensuring that we have supported distributions delivered to the customer. The tow main support offering comes from Object Computing Inc (OCI) and Prismtech which both offer different types of support contracts allowing for natural savings in development and distribution but with the assurances of support.

5.1.3 Performance

One of the more important factors of selecting an appropriate vendor is ensuring that we are getting a high performance and largely scaleable implementation of CORBA. From the performance evaluation carried out by VTransIT on TAO 1.3, Orbix 6.1 and Visibroker 6.0, the results show that the overall better implementation is Iona Orbix but TAO is closely second followed by Visibroker. [30] There are a number of benchmarks done and all three comes closely linked together in various ordering which different benchmarking test cases. The following figure shows the results from VTransIT:

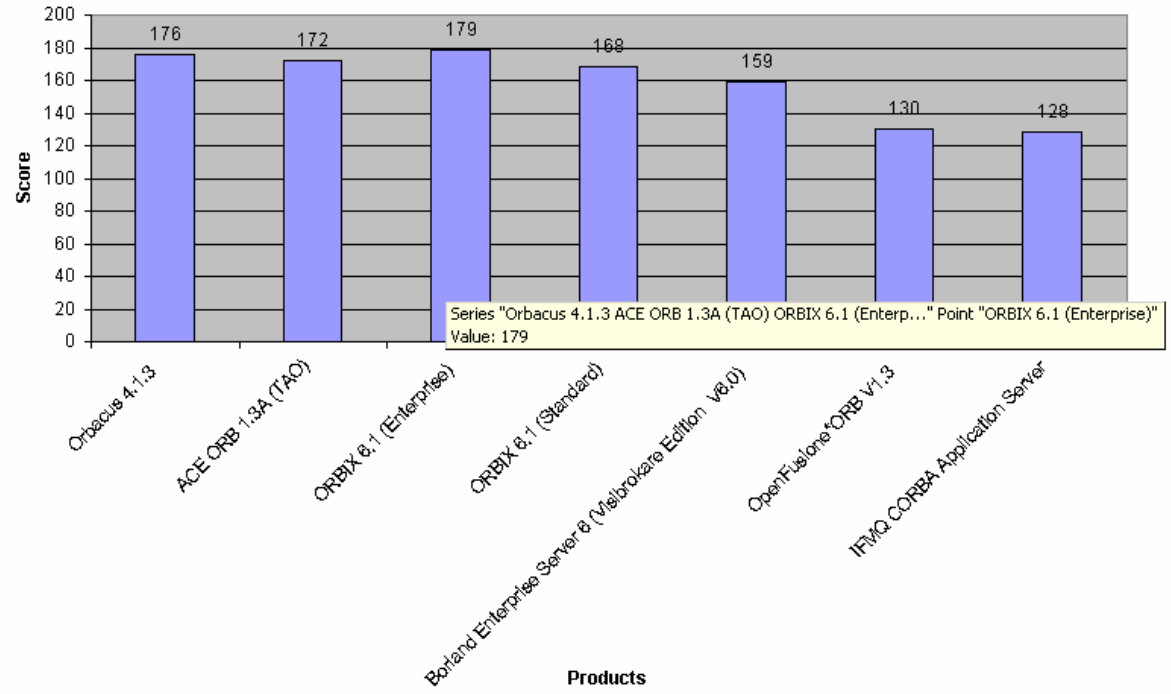

Figure 14: VTransIT results [30]

As Figure 14 shows the TAO ORB which is commented as “good for performance, scalability, cost of ownership is very low, has source code availability” [30]. It falls slightly behind Iona but has the advantage of low cost of ownership.

5.2 Summary

To summarize the analysis of the vendor implementation the two main runners are the TAO ORB for Open Source versus the Iona Orbix solution for commercial. The option of choosing a commercial implementation does have advantages of continued support but the costs are excessive and the Open Source Version is fully funded and does have multiple commercial support companies giving it an added extra as it is constantly under development, fully supported and has a higher level of compliance with the CORBA specification than the commercial implementation discussed.

TAO may be downloaded and distributed under an open source license and is completely free of development- and run-time license fees. It may be downloaded from: http://download.dre.vanderbilt.edu/

SECTION 4: CASE STUDY IMPLEMENTATION

Chapter 6 – Case Study: Design & Implementation

Chapter 6

From the preceding chapters we evaluated the various technologies referenced in Chapter 3 and discovered that the CORBA Notification Service satisfied the requirements needed by the Contact Center Server. Chapter 5 discussed the vendor implementation that was to be used and this chapter will involve the design and implementation for sending the events from the Contact Center Server to the clients.

6.1 Server Design

As stated in Chapter 2 of this paper, the Contact Center Server publishes two interfaces that allow for third party applications to receive Real-time Data via Multicast and via a Windows API. The Real-time Data produced involves the following statistical packages:

- Application statistics
- Skillset statistics
- Agent statistics
- Nodal statistics
- IVR statistics
- Route statistics

Each type of statistic is collected in two different ways—interval-to-date and moving window. There are a total of twelve different types of event messages that must be supplied top the Notification Service. Before we approach this subject it must firstly be decided how we will send this information. There are three different approaches that may be taken at this level.

- Using the multicast implementation to send the data from the Contact Center Server to a multicast client which will forward the Real Time Data to the Notification Service.
- Using the Real Time Data Windows API which will allow for a third party application to receive this information and forward the Real Time Data to the Notification Service.

- Having the Contact Center Server act as a supplier of the Real Time Data directly to the Notification Service.

These options will be discussed in detail in the next sections of this Chapter.

6.2.1 Real Time Multicast

The disadvantages of using multicast mentioned in Chapter 2 were

- The network routers need to be upgraded in most cases to allow for multicasting.
- The data is being multicast over an open network thus it causes bandwidth problems on busy Contact Centers.
- The data is visible to all.

All of these disadvantages will not apply here as having multicast enabled locally will not imply that the routers need to be upgraded or that we will suffer bandwidth problems but does imply that the data is visible to all on the network. This can be prevented by ensuring that any network routers disable multicast but this does add another element to the running of the server thus it will not be considered appropriate.

6.2.1 Real Time Data API

The disadvantages of using the API mentioned in Chapter 2 were

- No interoperability, programs must use a Windows DLL (C++ and .NET)
- Limited client applications due to the fact it is a multi-threaded push service.
- Developers can only develop client applications on Windows machine.
- Client applications are not portable to other environments.
- Asynchronous Delivery of Messages.

This API ensures that any service acting as a client of this API must be implemented in a Windows environment but we it no longer has the limitations of limited clients or asynchronous delivery of messages as it will be acting a mechanism to forward Real Time Data to the Notification Service which will in turn be the source of all data for clients. A main advantages of this is that

- The API client application can be used to connect to multiple Contact Center Servers in a networked environment thus been able to supply all events to a single Notification Service.
- A single API client application to supply information directly to multiple Notification Services where scalability of client connections may be an issue for performance of the Notification Service.

6.1.3 Server Implementation

This would involve the implementation of a CORBA Notification supplier inside the core of the server allowing for it to send information directly to the Notification Service inherently having reliable of messaging within the CORBA environment. It would also allow for the supply of information directly to multiple Notification Services and could be configured to have each server send information to a single Notification Service solving any problems that may arise in the future.

6.1.4 Summary

The Real Time Data Windows API and the Server Implementation do success in the same area's with both requiring a similar amount of development do be done but the Server Implementation would result in a more tightly coupled system and may result in

performance related issues on the server. The creation of a separate independent service that may be run on any machine in the local network provides us with a solution to the problem.

Also, if we are to implement the Notification Service as part of the Contact Center Server and not as a separate server, this will involve validation and verification of the entire Contact Center Server while the implementation of a new service will involve the validation and verification of only the single service, therefore providing a faster time to market and less system test of the overall system.

6.2 Design Implementation

The design implementation is taken from our conceptual design "How to propagate event data from the server?" where the actual components in the design have been decided and act as a solution to the problem. This is a design that developers can code the actual software component that will be implemented by the system. The design of the service will take a component design methodology allowing for the service to be broken down into behavioral pieces exposing each area of functionality in the implementation.

6.2.1 Conceptual Design

The Server contains a propagation service that is used to propagate data within the Server and to the external clients connected via Multicast or the Real Time Data API. Since this service is already supporting the RTD interface we will be using this to allow for further propagation of data to be achieved outside of the Server processor. This is shown in Figure 12. It will allow for the Server to send the Real Time Data to the new RTD Client service

either remotely or locally thus allowing for processing of this information either remotely or locally.

The information flow will be as follows:

- The new Service will subscribe with the Contact Center Server to receive Real Time Data messages.
- The Contact Center Server will monitor Real Time Data messages.
- When values change or on periodic time slices, Contact Center Server will publish the Real Time Data to the interested client applications.
- This data will be received by the new Service and a new structured message built with QoS options added
- Structured message will be sent to the Notification Service.

6.2.2 Design Overview

The functionality of the RTD Client Service is built around four software components, which will be described here in detail. Two of the software components are Executables (EXEs), one that runs as a Windows service and the other constitutes the registry configuration for the operation of the service. Two of the software components are Dynamic Link Libraries (DLLs), and will provide the connections, and much of the functionality, to the CORBA Notification Service and the Contact Center Server. These DLL hides the actual communication methods from the users of the DLLs, the service and configuration tool, and provides a simple “C++” API to send data. Figure 12 above shows the location of the various components in the design.

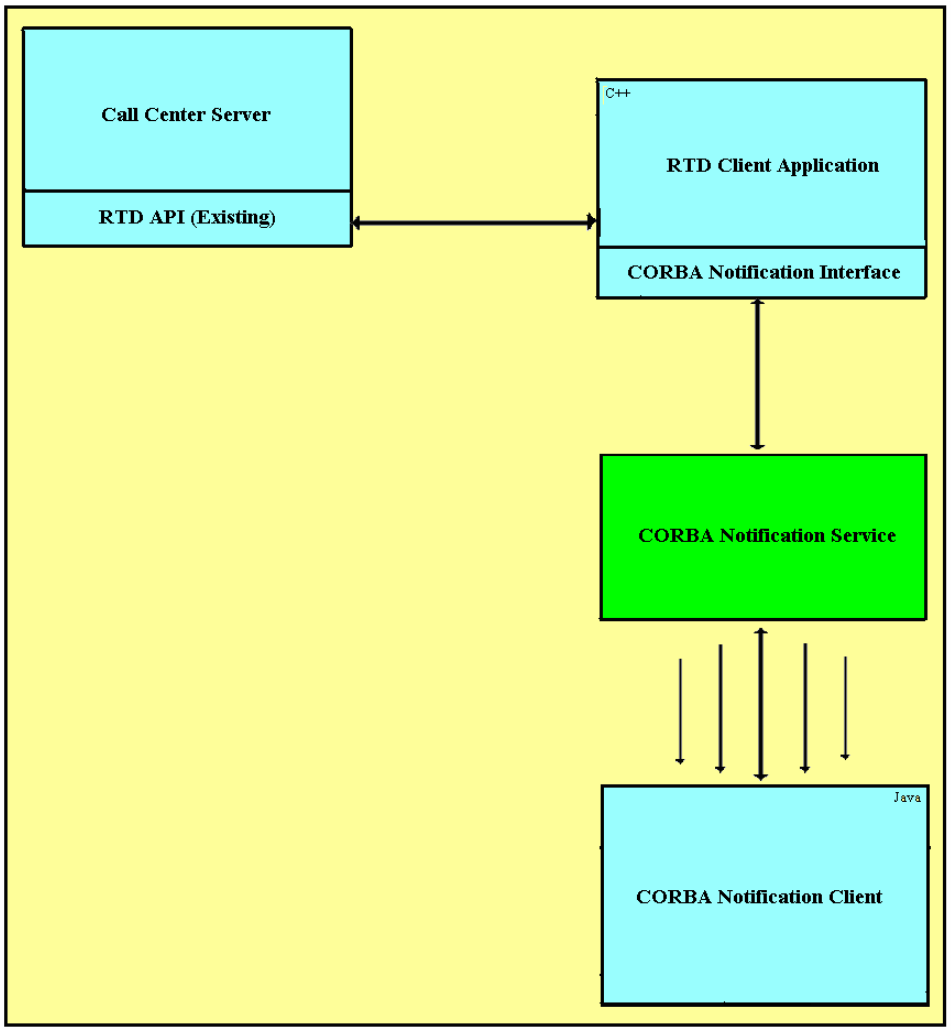

Figure 15: Design Implementation

As shown in Figure 15 above, the implementation of the supplier interface will be contained within a Dynamic Linked Library (DLL) which will be used by the multi-threaded RTD Client Application as a point of propagation to the Notification Service Event Channel.

6.3 Functional Overview

As mentioned above, the new Real Time Data CORBA (RTDC) service provides for the interaction of Contact Center Server and with TAO Notification Service.

6.3.1 Feature Overview

- RTDC design is component based in an effort to divide the problem domain. It consists of four software components:
    - nirtdc.exe: Windows Service
    - nirtdc.dll: Driver for connection to the Contact Center Server.
    - nirtdc_notify.dll: Driver for connection to the TAO Notification Service
    - nirtc_config.exe Configuration Setup
- It acts as a RTD Client interacting with the Server and consuming one RTD connection
- RTDC runs as a Windows Service using the Control Service for startup and shutdown
- It uses the RTD runtime for communication with the Server.
- Configuration
    - Base configuration is performed using the Configuration tool which writes configuration data to the Windows Registry..
    - Logging is configured during install and can be reset through the Windows Registry.

6.3.2 Operation Overview

This section describes the basic operations of RTDC. It is intended to give the reader a high level concept of operations.

RTDC is designed as a client of the RTD API provided by the Contact Center Server. The RTD API provides an interface for:

- Describe the set of statistics desired.
- Translate names (agent, skillset, application, nodal, IVR and route names) into id values that can be used in the description of the desired set of statistics.
- Request either Asynchronous or Synchronous transmission of the requested data.
- Access to just what has changed (new rows, deleted rows, changed rows).
- Cycle through the series of rows and columns of data transmitted.
- Translate table id values into (agent, skillset, application, nodal, IVR, route) names which can be displayed to the end user.
- Handle recovery when communications fail.
- Handle both the Windows 95 and Windows NT environments by providing two different DLLs.

The RTD API application is built on top of three ICCM components:

- Toolkit: used for threading, synchronization and communication features
- Security Server (SS): used for ensuring access to the ICCM server
- OAM: used for translation of names

6.3.3 Configuration

The RTDC Service will run accordingly to the configuration set using the configuration tool. This tool will write the following information to the Windows Registry for connecting to the Contact Center Server.

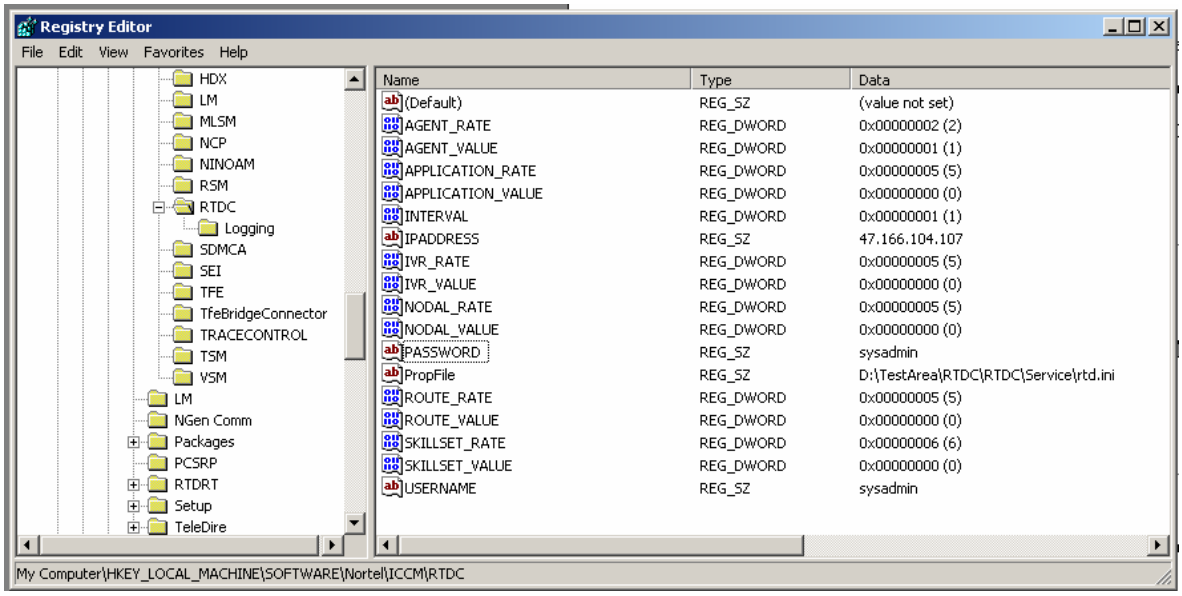

Figure 16: Configuration Data

The configuration data shown in Figure 16 will allow for the connection to the Contact Center Server and allow for the dynamic allocation of Real Time Data from the Server. The Service will use the toolkit runtime provided to communicate with the Contact Center Server in an attempt to login to the Security Server. It attempts to log onto the Contact Center Service using the NIrtd_login and supplies the configuration information from the Windows Registry. The connection to the Notification Service is part of the DLL and reads a configuration file for it connection. This configuration file will be discussed later in this chapter.

6.3.4 Functional Operation

After initialization of the service is complete and a successful connection to the Contact Center Server is achieved and a successful connection to the Notification Service is established, the internal operation of the RTDC Service contains the single listener thread that will read the registry and listen for changes on the registry values for the service. This thread is used to dynamic create and destroy worker threads running in the service. Each worker thread will correspond to one of the 12 event types that can be received from the Server. When the value of the corresponding registry value is enabled, a worker thread will be created allowing for a request to be made to the server. Each thread will call into the rtdc.dll which provides an interface which in turn calls the RTD API to the server. This DLL contain a callback function that will be called by the Server when an event value changes or is required.

The callback function in will create a new StructuredEvent messages structure, populate it with the necessary header information, event data and QoS properties and call out to the notification DLL which will forward the message to the Notification Service. This call out will ensure that there is a valid connection to the Notification Service and if not it will attempt a connection. The code segment that sends the data to the Notification Service is a singleton that must be acquired before sending to the single event channel. This design allows for the multiple threads to structure the message without having a synchronize message queue thus delaying the initial sending of the Real Time Data.

6.3.4.1 RTDC Service

Base HAI functionality exists in the HAI Service – nihai.exe.

- This executable runs as a Contact Center Server and Windows Service.
- It is called "Real Time Data CORBA Service" in the 'Services Control Manager' (SCM) Window.
- It is Description in SCM reads: "Integration of Contact Centre Server with CORBA Notification."
- It is startup and shutdown are controlled by Contact Center Server or via the Service Control Manager.
- Status is available via the 'Contact Center Server System Window' in the 'Start Menu'.
- Service fatal errors are reported to the fault manager and are visible in the event viewer.
- Is designed to pickup changes affected by the Configuration Wizard automatically without requiring a service shutdown.

Upon startup request:

- Starts logging – default is informational level, which logs startup and shutdown functionality.
- Initializes toolkit framework.
- Attempts to connect to Contact Center Server and the TAO CORBA Notification Service. If connection to any of the above fails the service continues to start, and wait is for a configuration change, or one of the services to become available, before trying to reconnect.
- Creates a listener thread which wait is on configuration changes.

Listener Thread:

On startup:

- Registers for notification if there is any change of the registry values associated with RTDC, and then gets the settings for RTDC, such as RTD API Login details, Contact Center Server IP address from the registry so it can check values when a change is notified.
- Creates a connection to the CORBA Notification Service using the supplied properties file for configuration.

While looping:

- Checks for a notification of registry value change. If a registry value is changed the new value is read and the connection associated with the change is recovered.

On shutdown:

- Exit is the thread run function.
- Disconnects from the Contact Center Server using the RTD API, disconnects from the CORBA Notification Service and closes any handles and deletes any memory still allocated.

Worker threads:

On startup:

- Creates a new request for the Contact Center Server and sends this to the Server via the RTD API.
- Registers a callback for the event data to be pushed to.

Callback:

- Receives data via a callback from the Contact Center Server, processes this information, packages it as a Structured Event message and forwards this to the rtdc_notify DLL that will send the information to the CORBA Notification Service.

On shutdown:

- Exit is the thread run function.
- Closes any handles and deletes any memory still allocated.

Upon shutdown request:

- De-registers and de-initializes all connections.
- Deletes listener and worker threads.
- Stops logging.

6.3.4.2 RTDC Logging

All RTDC components except the GUI use a local Generic logger library to log messages. The wizard as a GUI sends informational and error messaging to screen as prompts, if required. Logging in RTDC has the following properties:

- Each component uses an individual log file to log it is messages.
- All log entries are date and time stamped with millisecond precision.
- Logging variables, which are set in code, in the constructor to the logger object, include:
    - Location log file is to be placed.

  - Name log file is to be given.
  - Append mode.
  - Maximum size of log file.
  - Level at which messages are to be logged.

- Maximum size of log files and 'Level at which messages are to be logged' are also set in the registry. Initial registry values are set during install. A registry checking object is instantiated which checks these values. It is also notified when a change occurs to them. The constructors default values are overridden by the registry values. The registry location for logging information is: [HKEY_LOCAL_MACHINE\SOFTWARE\Nortel\ICCM\RTDC\Logging]
- Logging for the service can also be sent to a console which can be toggled on/off in the logging location of the registry.
- Messages can be logged on eight levels with priorities ascending from zero: LOG_FATAL (0); LOG_WARN (1); LOG_INFO (2); LOG_DEBUG (3); LOG_TRACE (4); LOG_TRC_1 (5); LOG_TRC_2 (6); LOG_TRC_3 (7). (Only the first five are currently used). Levels are set in the constructor and overridden by the registry. When a level is set - messages at that level and higher are logged.

- When a log file size exceeds it is configured limit the old file is retained and the file re-initialized. The previous file name is renamed with the filename prefixed by the string "Previous_".

- Any errors which occur while trying to log are logged to a file name with the name of the file where logging was intended prefixed by the string "FatalError_".

6.4 Notification Service Overview

The connection between the RTDC Service and the Notification Service is managed via the nirtdc_notify.dll module. This dynamic link library is loaded by the RTDC service. It provides the following interfaces that may be called:

- StartNotificationService
- StopNotificationService
- pushNotifyData

The listener thread at startup with provide a call to the connection class which initials a connection to the Contact Center Server but also invokes a call to StartNotificationService to receive a connection to the Notification Service. The connection is destroyed by invoking a call to StopNotificationService only when the service is been shutdown.

The pushNotifyData method is invoked from the RTD API connection driver DLL that receives callback's from the Contact Center Server. When a callback method is invoked the Real Time Data received is organized as new event data, deleted event data or updates to current event data. Once this data has been grouped into the specific table group, the data is packaged into a StructuredEvent message and forwarded to the driver of Notification Service.

6.4.1 CORBA Connection

The CORBA Notification connection is initialized by the listener thread on startup of the service. This would initial a single instance of a thread to set up a connection with the Naming Service and retrieve a connection to the running Notification Service. The thread would read the registry in finding the location of the properties file required in starting the service. To get the location of the Notification Service, we would first have to find the CORBA Naming Service. This is an implementation of the OMG Interoperable Name Service (INS) specification. Basically, it provides the principal mechanism through which most clients of an ORB-based system locate objects that they intend to use. In finding the location of the Naming Service from which the Notification Service can be found the following parameters where used

```
## Naming Service Discovery is done using:
##      [<ipaddress> is the IP Address of the Server running the Naming Service
##      which has a configured Notification Server]
##      ORBInitRef NameService=iioploc://<ipaddress>:<NameServerPort>/NameService

IPAddress=47.166.104.94
NameServerPort=4422
```

Once we finding the Naming Service we must know the Naming of the Service to which we want to connect. Again the properties file contains this information.

```
## Name of the Notification Service to Connect too
##      [This is configured using Registry on Server running Notification Server]
Notify_Service_Name=CCMS_Notification_Service
```

This allows for our Service to have a connection to the Notification Service. Next is the decision if we wish to create a new channel or use the existing channel. This is the configuration used

```
## Create a Notify Factory Channel or Bind to the existing Factory Channel on
## the Notification Server [ Recommend NO ]
## 'No' means we bind to the Notification Service Channel.
##      [If one does not exist we will create one on the Notification Server]
## 'Yes' means we create a new Notification Channel that references the local
## Server..
##      [This must also register with Naming Service to allow it to be
##      identified as <Notify_Service_Name>\\CCMS\\<Server Name>\\<Channel_Name>]
##              Server Name is retrieved as the site name configured for CCMS -
##              if not found it will use <DefaultServerName>
Create_Notify=No
DefaultServerName=defaulted
```

The option to allow for the service to create a new event channel allows for the service to send messages via it is own separate event channel. In basic terms, a supplier will supply events to an event channel and a consumer will receive events from that same event channel. The supplier is unaware of the consumer and the supplier supplies this information to the Notification Service which then propagates this information to the consumers on that event channel. If we will not create a new event channel and connect to the default, this allows for multiple Contact Center Servers to supply Real Time Data to a single event channel ensuring that clients will not need to connect to multiple event channels to receive information from different Contact Center Servers. This is configurable as some implementation might prefer for information to be available to specific clients which must connect to different event channels. The creation of a new event channels will invoke a call to register the new event channel name on the Naming Service. For the creation of a new event channel and the naming of that channel this information is too configurable via the properties file as shown below.

```
## Do you want to create a Notify Channel
##      'No' means that we will bind to the Notification Default Channel [This
##      is already bound to the Notify Factory Channel]
##      'Yes' means that we will create a new Channel using "Channel_Name"
##      that is bound to the Notify Factory.
Create_Notify_Channel=No
Channel_Name=RTDC
```

In Theory “An event channel is a factory that creates consumer admin and supplier admin objects. This differs slightly from the CORBA Event Service event channels, which only have one instance of admin objects. QoS and admin properties can be set on the event channel during it is creation. These parameters are passed as default values to any admin object created by the channel. These parameters can be changed subsequently by consumers and suppliers.”[8]

So the initial connection to the Notification Service is created on service startup. If the connection fails to start at this point the service will continue to start however. On sending of each event status to the Notification Service, the connection status is checked and a re-connection is tried to ensure robustness in our system design.

6.4.2 Structured Event

The callback function within the RTD API connection driver receives a NIrtd_stTableGroup message from the Contact Center Server. This is made up of three table structures NIrtd_stTable;

1. deletedValues
2. newValues
3. deltaValues

Each table is then taken and a local table is updated with the values. Each table is made up of:

- number of rows
- number of columns
- table

Once the local tables have been updated in the RTDC Service, changes to the original table are then propagated forward to the Notification Service. Each new, deleted or updated event is forwarded individually to allow for filtering of each event message without a Notification client having to receive large bulk packets of data. The Structured Event message has the following structured:

Table 2: StructuredEvent Structure

<table>
<tr><td colspan="2">

```
struct StructuredEvent {
   EventHeader header;
   FilterableEventBody filterable_data;
   any remainder_of_body;
};
```

</td></tr>
<tr><td>

```
struct EventHeader {
    FixedEventHeader fixed_header;
    OptionalHeaderFields variable_header;
};
```

</td><td>

```
PropertySeq FilterableEventBody
```

</td></tr>
<tr><td>

```
struct FixedEventHeader {
    EventType event_type;
    string event_name;
};
```

</td><td>

```
sequence<Property> PropertySeq;
```

</td></tr>
<tr><td>

```
struct EventType {
    string domain_name;
    string type_name;
};
```

</td><td>

```
struct Property {
    PropertyName name;
    PropertyValue value;
};
```

</td></tr>
</table>

The StructuredEvent message fixed header will have the following parameters set:

- Domain_name: IP Address of the Contact Center Server responsible for sending the event.
- Type_name: Moving Window or Interval to Date event type
- Event_name: one of the twelve event types i.e.: Agent, Skillset, Nodal, Application etc

The StructuredEvent is shown in greater detail in Figure 17.

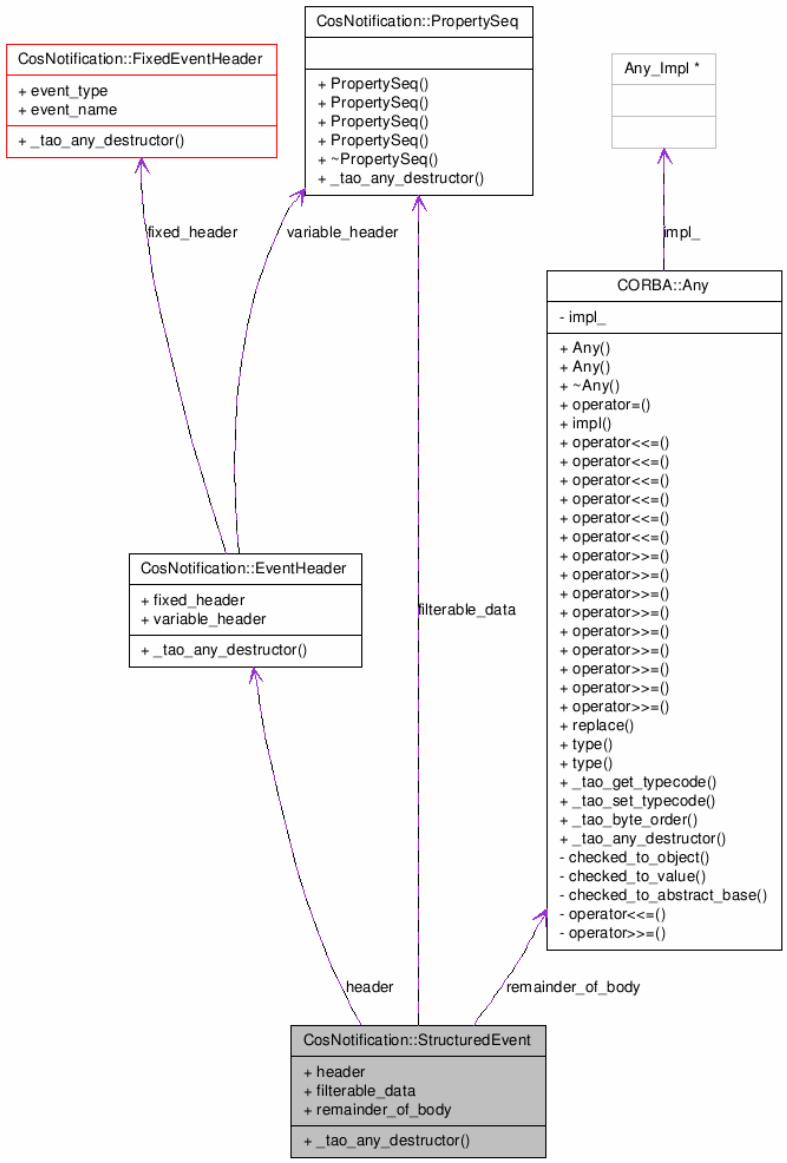

Figure 17: Structured Event [38]

The variable header information is not used in this case as priority and timeout of messages are not necessary. These properties are supported on a per event basis allowing for a message to be discarded after a set timeout period or a message may be of a higher priority meaning this message will be delivered quickly to consumers. After the header information has been set, the filterable event data is set. The event data here contains the individual

event elements of the various event messages. An example of this would be an Agent event message that is made up of the following elements.

- AgentID
- SupervisorID
- Agent State
- TimeInState
- AnsweringSkillset
- DNInTimeInState
- DNOutTimeInState
- SupervisorUserID
- PositionID
- Not Ready Reason
- DN Out Num
- Skillset Call Answered
- DN In Call Answered
- DN Out Call Made
- Answering App
- Answering CDN
- Answering DNIS

Each of these elements are stored as name / value pairs as part of the Filterable event body. This is the part of the StructuredEvent upon which the consumer is most likely to base filtering decisions. Structured events are defined as “a standard data structure into which a wide variety of event messages can be stored. The schema for structured events is known to the Notification Service and it is clients. Consumers can install different filters that use the “filterable body” fields of the structured event definition to match with the filter constraint expressions efficiently.”[8]

6.4.3 Quality of Service (QoS)

The QoS options available are can be attached to the StructuredEvent on a per event basis or can be options set for the vent channel. The options supported on a per event basis are the Timeout and Priority. The timeout property refers to the amount of time a message should wait before being delivered. If this time is exceeded the message will be discarded. This property is set by the CORBA Notification connection handler when sending the StructuredEvent to the Notification Service. The timeout set is fifteen seconds as moving window values are updated every fifteen seconds which means any old messages received after this time would be incorrect.

The following is a list of the QoS supported by TAO.

Table 3: QoS Properties

QoS Property	Support
EventReliability, ConnectionReliability	Best Effort
Priority	Supported Per Message
StartTime, StopTime	Not Supported
Timeout	Supported Per Message
MaxEventsPerConsumer	Supported
OrderPolicy, DiscardPolicy	PriorityOrder,AnyOrder, FifoOrder, LifoOrder, DeadlineOrder
MaxEventsPerConsumer	Supported
MaximumBatchSize	Supported
PacingInterval	Supported

When we connect or create an event channel we add EventReliability, ConnectionReliability and an OrderPolicy of FifoOrder using set_qos to the CosNotifyChannelAdmin object. The event reliability and connection reliability allow the specifying of fault tolerance properties to the Notification Service. When these properties

are set, then after a Notification Service is restarted after a crash, it must reconnect to all it is clients and deliver all events that have not expired to the consumers. The OrderPolicy ensures that the messages are arranged correctly in the dispatch queue. Having the property value set to the FifoOrder ensures that the messages are delivered as they arrive to the Notification Service.

SECTION 5: CASE STUDY EVALUATION

Chapter 7 – Case Study: Evaluation

Chapter 7

This chapter presents an evaluation of the implementation of the CORBA Notification Service. It will evaluate the software solution and the use of the CORBA Notification Service compared to the existing functionality provided by the Contact Center Server. The emphasis of the evaluation will conclude whether the solution is a suitable replacement for the existing interfaces and to ensure that all of the initial problems were resolved.

7.1 Desirable Characteristics

The main issues as described in Chapter 2 of this paper were to include the following requirements in the solution to propagating Real-Time Data from the Contact Center Server:

- provide the information in a defined and structured message format,
- allow for the client to receive / retrieve information from the server,
- provide platform and language independence for the customer applications receiving / retrieving the information,
- relieve the server of processing and managing individual connections for each client wanting to receive information,
- provide asynchronous message to the client application in a loosely coupled system.

The Notification Service Specification proposed by the OMG satisfied the above criteria which will be discussed throughout this chapter.

7.1.1 Structured message format

The specification for the Notification Service also defined a specification for the StructuredEvent message type used by the implementation of the CORBA Notification Service. It is a data structure that allows a wide variety of event messages to be stored and is generic in it is structure. It uses a header in the identification of the message type and instance while allowing for a sequence of name value pairs to be incorporated into the body of the message allowing for easy filtering of data.

As the CORBA Notification Service allows for the service to send CORBA::Any data type this allowed for the creation of a new more suitable message structure. The CORBA::Any data type is an untyped type thus allowing for any object or struct type message to be sent through the Notification Service. In evaluating the message format to be used the advantages of using defined message structures that are part of a specification allow for these message structures to be implemented by multiple vendors and supported by different technologies increasing interoperability with other technologies. An example of this is the JMS Message structure which conforms to a similar structure as the StructuredEvent. Both of the message event structures allow for the StructuredEvent data to be mapped directly to a JMS message as shown below in Figure 18.

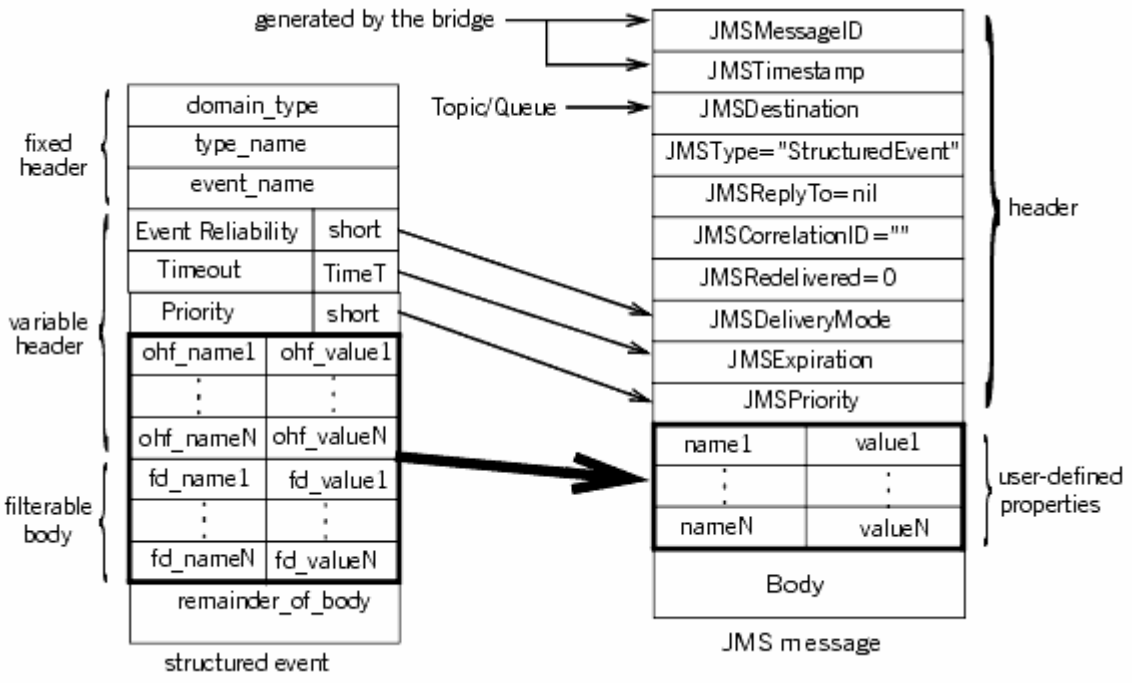

Figure 18: Mapping a StructuredEvent to a JMS Message

A significant advantage of using the Typed Event Notification Channel is that an application's elements communicate via strongly typed application-level interfaces and therefore will not need to encode or decode to and from an event data type when publishing

or consuming events [32]. If any other type was to be used there would exist an interface requiring that events be formatted into an explicit event structure, delivered using an infrastructure-level operation, and extracted from the event data type.

7.1.2 Communication Model

The CORBA Notification Specification supports both the 'push' and 'pull' models allowing for Publish / Subscribe architecture. This has the following advantages:

- one-to-many or many-to-many propagation of messages
- Subscription to topic or channel having no restriction on the consumer
- All subscribed clients receive the message, if available

The initial client / server paradigm is moved further towards a publish / subscribe paradigm where the consumer is completely decoupled from the supplier. Suppliers publish events and consumers receive only events for which they have subscribed. This is the push model which is the implementation being used in the service created. The vendor selected does not support the 'pull' model as of their latest release, this is not a concern for this implementation but does hinder on allowing for clients to retrieve information from the Contact Center Server.

7.1.3 Platform and Language independence

As the Notification Specification is part of the CORBA Specification it is inherently language independent as it is defined by a specification allowing for any language to communicate via the interfaces defined. In the case of TAO it is also platform independent as it is built on top of the ADAPTIVE Communication Environment (ACE). This is a

freely available, open-source object-oriented (OO) framework that implements many core patterns for concurrent communication software.

7.1.4 Connection Management

The idea of connection management is to relieve the Server of processing and managing individual connections for each client registering for event information. The operation of the RTDC Service allows for a single connection to the Contact Center Server ensuring that the Server is minimized to a single connection while allowing for multiple consumers to connect to the Notification Service to receive Real Time Data. The Service can also be setup to ensure that all processing of event data can be carried out locally or remotely allowing for intensive processing of event data to be carried out on a separate machine relieving the Server processor of time consuming work. As the Notification Service uses an event channel this decouples the supplier of events from the consumers and also provides a mechanism of supplying multiple consumers faster. This is achieved by the Notification Service which uses an event channel that can implement group communication thus serving as a replicator, broadcaster, or multicaster that forwards events from one or more suppliers to multiple consumers.

There is a disadvantage however to the implemented design in that the Contact Center Server is not completely decoupled from the design. The implementation of the RTDC Service does mean that the service will register securely with the Contact Center Server that is responsible for call back notifications to the service.

The Notification Service can also be setup locally or remotely from the RTDC Service and / or the Contact Center Server providing a scalable system.

7.1.4.1 Configuration & Scalability

The implementation design allows for the following configurations:

- There can be a single RTDC Service connected to a single Contact Center Server which will then send the information to a single Notification Service. This allows the client to get information from a single Server but would require the client to connect to multiple Notification Services to receive information from multiple Contact Center Servers.

- The above configuration can be used where multiple RTDC Services send information to a single Notification Service allowing client to receive information which can be filtered from multiple Contact Center Servers.

- The RTDC Service can also be setup to connect to multiple Contact Center Servers enabling it to process a large number of events. This data is then passed to the Notification Service allowing for Notification clients to receive information which can be filtered from multiple Contact Center Servers. This setup would have to ensure that the RTDC Service can process the event data in a timely fashion to ensure Real Time Data and no bottlenecking of data.

As shown above there are multiple different setup and configuration options but the scalability of the Notification Service must also be considered. The considerations here

included the hardware platform and operating system being used, the capabilities of the underlying TAO ORB, the performance of the processor and efficiency of the event filtering and dispatching modules. A paper by Douglas C. Schmidt titled Designing and Optimizing a Scalable CORBA Notification Service reviews how the TAO ORB has implemented the optimization of event filtering. The paper states that "In TAO's Notification Service, we pass a hint to proxy objects to skip filter evaluation of their parent admin object if this has already been performed. We can optimize the filtering of a given event by a group of proxies since each member of the group logically applies the same filters to the same event. Thus, the results of the evaluation of a given event against a given filter can be shared by all proxy objects managed by a given admin object" [8]. The configuration of the Notification Service also allows for the setting of various resources. Examples of some of the more important options are listed below:

Table 4: Notification Properties

-ORBRunThreads nthreads	Number of threads to run the ORB::run method.
-UseSeparateDispatchingORB 1|0	Indicates whether the service should create use a separate ORB dedicated to dispatching of events.
-DispatchingThreads [thread_cnt]	Enables MT dispatching with the specified number of threads
-ListenerThreads	How many threads for listener filter evaluation
-AsynchUpdates	Send subscription and publication updates asynchronously
-AllocateTaskperProxy	Allocate worker tasks per proxy

An explanation taken from the ReadMe file that is deployment with the TAO Source files states that “A Task here implies a thread pool that performs a fixed work in the Notify”. E.g.: when you specify "DispatchingThreads 1". It means that there is 1 thread to perform the event dispatching to consumers irrespective of the number of proxy suppliers. It also means that events destined for each consumer will be queued to a buffer for that consumer. Therefore, you can also think of this option as enable Consumer-side Buffering of Events. This is the default case. When you specify "-AllocateTaskperProxy" it asks notify to create a dispatching task (with the specified thread pool size) per proxy supplier. So if you use this option and connect 50 consumers with 1 thread for the dispatching task you will have created 50 dispatching threads. This option should be used with care, it will not be needed in most cases.

Why has this feature in the first place?

The intent is to allow the software architect of a Notify based system, fine control over where and how much thread resources are deployed. E.g.: a channel could have 2 proxy suppliers - the first one delivers an important event in huge quantities. A dedicated thread pool to this proxy will ensure better throughput to it is consumers. Similarly "-ListenerThreads 2" specifies a thread pool for use by the supplier-side processing. This enables Buffering on the Supplier-side, with the thread pool being used to process supplier side filters and push the events to the Consumer side.

7.1.5 Asynchronous messaging

The definition of asynchronous messaging is to 'fire and forget' where the supplier of events does not need to wait for an acknowledgment of receiving the data passed. The RTD API already supported by the Contact Center Server and used by the RTDC Service in this implementation allows for the registering of asynchronous data. This is the implementation used where the Contact Center Server uses a callback method to send the data to the registered client, in this case the RTDC Service. The RTDC Service can then forward this data to the Notification Service which uses an event channel allowing for asynchronous messaging while ensuring the supplier and consumer are decoupled from each other.

7.1.6 Existing Technology

The Contact Center Server implements the two existing interfaces that allow for the propagation of Real Time Data. One of these implementations is leveraged in the implementation of the new service that provides Real Time Data. In evaluating the new service against the existing solutions we will examine their advantages and disadvantages and how the new service compares.

7.1.6.1 Real Time Multicast

The issues recorded previously were:

- The network routers need to be upgraded in most cases to allow for multicasting.
- The data is being multicast over an open network thus it causes bandwidth problems on busy Contact Centers.
- The data is visible to all.

The implementation of the new service will no longer result in any issues with multicast enabling the routers on the network. Bandwidth issues will only relate to the number of clients connected to the Notification Service but will be based on point to point connection to the Notification Service. The data is no longer present on the network thus a client must connect to the Notification Service to receive data. The ability for multicast to supply the data network wide to multiple clients can also be achieved by the Notification service as it is event channel can be configured for group communication.

7.1.6.2 Real Time Data API

The issues recorded previously were:

- No interoperability, programs must use a Windows Dynamic Linked Library (DLL) (C++ and .NET)
- Limited client applications due to the fact it is a multi-threaded push service.
- Developers can only develop client applications on Windows machine.
- Client applications are not portable to other environments

As the Notification Service is a CORBA based specification it allows for both language and platform independence which resolves the problems of the RTD API. The new design allows for multiple different configurations ensuring that only a single connection to the Contact Center Server is required allowing for the Notification Service to propagate Real Time Data to multiple consumers connected.

A limitation here is that any client application can connect to the Notification Service as there is no security or registration interface associated with the specification.

7.2 Open Source

The Notification Service specification is implemented by the ACE ORB (TAO) which is an Open Source implementation of the specification. Development of the TAO ORB began in 1993 by Douglas C. Schmidt's who is now part of the research group at Vanderbilt University, Washington University, St. Louis, and the University of California, Irvine. It is a standards-compliant real-time C++ implementation of CORBA based upon the Adaptive Communication Environment (ACE). It attempts to provide efficient, predictable, and scalable quality of service (QoS) end-to-end.

ACE+TAO have been funded by the DARPA Quorum program, NSF, and many visionary industrial sponsors. TAO has the following advantages:

- It can be downloaded as binaries or as source code.
- It supports numerous different build formats (including Windows)
- It is beta tested before release.
- It is continually being improved
- It is CORBA 3.0 Complaint.

7.2.1 Community Support

TAO has a large community online and the following groups provide freely from a wide range of developers, designers, and commercial support vendors. Registration for online community can be achieved using the below links.

http://groups.yahoo.com/group/tao-users/

http://groups.google.com/group/comp.soft-sys.ace/topics

7.2.2 Commercial Support

As TAO matured over the years, a number of companies began to support it commercially. Open-source commercial support, documentation, training and consulting for TAO is available from the following commercial vendors:

- Object Computing Inc
- Remedy IT
- Prismtech
- Systematic Designs International

This provides for reliable and stable distribution of supported TAO binaries. The above commercial vendors will not charge for the distribution of TAO but do charge for support of the distributed binaries.

7.3 Documentation

This is an area in which TAO is lacking. Documentation is by way of papers written by Douglas C. Schmidt on how TAO is best used in design of large scale systems. The online documentation does not appear to be very good or well supported.

7.3.1 Commercial Support

The companies that support TAO commercially do have good documentation on TAO and this is available to purchase from any of the vendors at a cost.

Remedy IT has a free version of their first TAO programmers guide available as a PDF download. This is a limited beginner's guide to TAO with little or no specific detail included.

OCI produces high quality, shrink wrapped CDs and documentation sets. They are designed to enable the rapid evaluation and subsequent use of TAO but this is product for sale at a cost. This is a much lager distribution of TAO and has a very detailed Programmer guide for TAO. There is no online information available as the purchase of hard copies is all that is made available.

Prismtech provide commercial support for TAO also and have recently hired TAO's designer Douglas C. Schmidt.

7.4 Installation

Installation of the TAO Notification service is very straightforward. Once the NT_Notify_Service.exe executable is built it can be installed on a Windows environment very simply as it is built as a Windows Service. The execution of the command "NT_Notify_Service.exe –i" will install the executable as a Windows Service which can then be started and stopped via the Windows Control Manager. The TAO Naming Service must be installed and running to allow the Notification Service to register with the Naming Service allowing it to be discovered by the supplier and consumer interfaces.

Neither the Naming Service configuration nor the Notification Service configuration is documented for Windows but looking at the source code shows that the creation of new registry keys is required to allow for the detailed configuration settings discussed earlier to be used.

```
HKEY_LOCAL_MACHINE\SOFTWARE\ACE\TAO\TAONamingServiceOptions

        -m 1
        -ORBEndPoint iiop://<IP Address>:<Port>
        -o tao_name_service.ior
        -ORBDottedDecimalAddresses 1

HKEY_LOCAL_MACHINE\SOFTWARE\ACE\TAO\ TAONotifyServiceOptions

        -ORBInitRef NameService=corbaloc:iiop:<IP Address>:<Port>/NameService
        -ORBListenEndpoints iiop://<IP Address>:0
        -Factory My_Notification_Service
        -Channel -ChannelName MyNotifyChannel
        -IORoutput tao_notify_service.ior
        -ORBRunThreads 4
```

Server side and client side deployment require only the TAO runtime files which can be downloaded from the distribution or can be built from the source code. These are Windows DLL files and are simple and unproblematic on Windows.

7.5 Reliability

The TAO Notification Service is CORBA 3.0 compliant but it does not implement all aspects of the CORBA Notification Specification; for example, pull interfaces and typed events are not yet supported. It does, however, implement several QoS properties, including per-message event priority, order policy, discard policy, maximum batch size, pacing interval, and maximum events per consumer. It also supports the administration properties maximum queue length, maximum consumers, maximum suppliers, and reject new events.

7.5.1 Compatibility

As the Notification Service is built on top of the Event Service, it enables backward compatibility to any applications currently implemented using the Event Service. The

interfaces used by the Notification Service are inherited from the Event Service interfaces so all of the capabilities of the Event Service are inherited by the Notification Service.

7.5.1.1 ORB Compatibility

Unfortunately, as mentioned earlier in this paper, TAO is built on the ACE framework. ACE stands for A Communication Environment, which is the large communication framework on which TAO is based. TAO performance can be increased using the ACE framework but ACE macros make the code un-portable to a different ORB, which results in more work hours should we wish to use another vendor in the future. Therefore, the RTDC Service does not use any of the ACE Marco's but instead uses standard CORBA code. [34]

7.5.2 Filtering

TAO's implementation of the Notification Service allows the use of your own constraint grammar or by default you can use the Extended Trader Constraint Language (ETCL) which is defined by the OMG. It allows application to create complex expressions to describe which events should be allowed to pass thorough an element of the Notification Service. The original Trader Constraint Language is also supported as defined by the Object trader Service specification. [33] The filter described above refers to forwarding filters. The specification for the Notification Service defines forwarding and mapping filters. Mapping filters are not implemented by TAO.

7.5.3 Runtime execution

In the testing of the Notification Service and the associated runtime files there was no reliability issues encountered. In the research stages of this paper, the community group

associated with TAO had said that there had been a large number of updates to the Notification Service in the latest release so testing with this release was done.

7.6 Performance

The solution was tested by configuring a large scale Contact Center Server up to send event data to the RTDC Service which propagated the data to the Notification Service. Testing consisted of increasing the call rate to a maximum supported rate of 70K call per hour and receiving this information on a single consumer.

7.6.1 Interface Limitations.

The Contact Center Server currently implements two interfaces to allow third party applications to receive data. The RTD API interface acts as a callback mechanism where the client registers for events, the server holds an object to that client and uses the callback mechanism in pushing events to the client. The RSM Multicast Interface sends UDP packets over the TCP network.

When dealing with the RTD API Interface, an issue encountered with this type of solution is that the client must have a defined callback interface which is used by the server. If we have 1,000 clients registering with the server for events, the system will incur performance issues as the server must have object references for each of the 500 clients and must send a separate message to each in turn. If this number of clients are registered with the RTD API Interface this will directly impact Multicast clients as the server performance will be down and the UDP packets will not be sent. This scenario would also directly impact our new RTDC Service as it would be one of the 500 registered clients and would not be receiving

the event data due to the performance of the Contact Center Server. In the case where the RTD API clients are connected to the Notification Server, there is no impact on the Multicast data; there is no impact on the RTDC Service and no impact on the performance of the Contact Center Server. The 500 registered clients of the Notification Service receive there event data in a timely fashion while we have the added advantage of filtering and QoS properties.

The number of consumers registered with the Notification Service was incremented with no performance issues encountered.

The operation of the Notification Service proves to be much more efficient and faster than the RTD API but slower than multicast which is expected as CORBA is built on top of TCP while multicast is part of the protocol.

7.6.2 Execution

The execution of the TAO Notification Service, while propagating Real-Time Data from an active Contact Center Server, proved successful. The design of a new service that implemented an existing interface while having the limitation of the example above results in the following advantages:

- Testing of only the new service. It will not impact the performance of the Server nor will it impact any of it is existing components.
- Having a service that can be deployed separate to the Contact Center Server.
- Ability to create a new SDK to support this new service.

7.7 Licensing

From OCI, “TAO is made available under the "open source software" model. The source is freely downloadable, open for inspection, review and comment. Copies may be freely installed across all your systems and those of your customers. The source code is designed to be compiled and used across a wide variety of hardware and operating systems architectures. Target systems include UNIX systems, including Linux; MS Windows platforms, and real time platforms such as VxWorks, Integrity and LynxOS.

The ACE ORB source code is copyrighted by Dr. Douglas C Schmidt and his research group at Washington University, University of California - Irvine and Vanderbilt University Copyright (c) 1993-2007, all rights reserved.“

7.8 Migration

When migrating from one version of the TAO runtime to a newer version of the same runtime, it is as simple as replacing the runtime files on the local host that is running the CORBA application. The application does not require any rebuilding of components (unless new features added in the more recent release need to be integrated into the new application). The reason that no rebuild of the application is required that the CORBA interfaces are part of a defined specification which means that the application calling into the runtime interface has not been changed and provides backward capabilities

In testing, executables can be built using an early release of TAO and the same executables will use newer versions of the runtime without recompilation at later dates.

7.9 Interoperability

MOM is a proven communication model for developing large-scale, distributed enterprise integration solutions. It provides more flexibility and scalability because senders and receivers of messages are decoupled. The Notification Service is a mature and stable standard that has been one of the OMG's success stories. As TAO is a C++ ORB and does not support Java we will look at the interoperability of TAO and the JacORB and also with the Java Message Service.

The JacORB is an Open Source 100% pure Java implementation of the CORBA specification and can be used with the TAO implementation. Any CORBA application that is written using Java can use the JacORB interfaces to compile in Java but can also use the interfaces to find the TAO Naming Service or TAO Notification Service and connect to these services. The interoperability of the two implementations is due to the CORBA specification which allows all implementations to communicate using the defined protocol and defined interfaces.

For interoperability with other services provided by Java, such as the JMS we must consider that it is a defined specification. It is an API for messaging in an EJB environment, which is now becoming the platform of choice for server-side Java development within the Java Enterprise Architecture thus, interoperability should be powerful. JMS is an important API because it provides simplified access to enterprise messaging systems from Java applications.

OMG published a specification in 2004 titled “Notification/JMS Interworking Service”. The specification defines a new bridge that will be used to manage and interconnect an event channel with a JMS destination. It will provide backward capabilities with both of the existing specifications and will provide automatic mapping of event types. This is not implemented by TAO or JacORB but Iona have been working on a bridge over the past number of years. I have not attempted this implementation as registering the JMS Provider as a CORBA Notification consumer and restructuring the message format would provide a similar solution to that defined in the specification.

Chapter 8 – Conclusion

Chapter 8

The focus of this paper was the research and evaluation of technologies related to the propagation of Real Time data from a Contact Center Server to third party client applications. The thesis paper is titled “Distributing Real Time Data from a multi-node large scale Contact Center using CORBA”. The aim was to research and evaluate the existing State of the Art technologies relating to Distributed Systems while designing and implementing a solution to the propagation of Real-Time data from a Contact Center Server.

8.1 Overview

The author found this thesis to be an informative and a worthwhile experience. It allowed for the gathering of information on a topic of great interest to the author thus providing greater knowledge and in depth experience of a number of new and emerging technologies. It also allowed for an introduction to life cycle management using USDRP for product implementation while working on this project. This proved to be an important learning experience in designing a new component as part of a large Server deployment.

The purpose of the thesis was the research and evaluation of technologies related to the propagation of Real Time data from a Contact Center Server to third party client applications.. This paper researched the existing and the emerging State of the Art technologies used from the propagation of data while ensuring that all technologies researched met the required criteria list below:

- Structured message and specifications
- Scalability
- Interoperability
- Asynchronous Messaging

The criteria resulted from the USDRP process which involved requirements gathering to be carried out. This involved analyzing the existing implementation, there advantages and disadvantages, while also reviewing customer requirements. The CORBA Notification Service defined by the OMG and implemented by TAO met all of the requirements while

providing a service that is highly reliable and well supported and has very low cost of ownership. The design also has the added advantage of being highly configurable. It can be centralized in a multi-node environment, can be dynamically integrated into any existing deployment and does not require for the validation and verification of the Contact Center Server as it is decoupled in it is entirety from the system. Overall, from a design and implementation perspective and from the fact that the solution provided meets all the requirements, the thesis project is highly successful.

The conclusions relating to the evaluation are:

General:

There are a number of vendors that implement the CORBA Notification Specification. After having reviewed the different vendors, the TAO implementation of CORBA shows the best results; having a continuous development cycle, commercial support, a faster and more efficient service and a low cost of ownership.

Standardisation:

There are many benefit is in having a service that is designed to a specification. These benefit is include ease of integration, standardized structures, flexibility in vendor’s components, improved confidence in the quality of the product and avoidance of vendor lock-in.

Interoperability:

The Notification Service is part of CORBA thus it is inherently interoperable. It is this interoperability that grants a sense of assurance that a system built to work within one environment will work across multiple environments.

In the context of the development of the new service it ensures that one system can talk to others using defined and supported specifications.

Scalability:

The implemented design allows for the solution to be scalable in a number of different ways. In a multi node environment where we have multiple Contact Center Servers in operation, a single RTDC Service can be connected to a single Contact Center Server where each RTDC Service is a supplier to a single Notification Service. The configuration can also support a single RTDC Service connected to multiple Contact Center Servers which supplies a single Notification Service Both solutions allow third party application to receive all information at a central location offering a much improved solution when compared to existing technologies.

Integration:

The TAO Notification Service can be used in conjunction with the Java implementation of the CORBA specification, thus allowing for the Java environment to use interfaces which connect to TAO Services. This is inherent in the CORBA specification.

Portability:

As the TAO implementation of CORBA is built on the ACE framework it will provide a portable implementation of the specification, as both ACE and TAO are portable onto multiple platforms.

Performance:

The performance of the TAO Notification Service is much better than that of the RTD API, as it prevents the Contact Center Server from having to obtain, store and execute multiple client interfaces in call-backs for supplying events in real time.

Ease of use:

The TAO Notification Service allows for a quick and easy to deploy service that is readily available as a Windows service requiring only the installation of a CORBA Naming service and a registry key for configuration.

Costs:

TAO may be downloaded and distributed under an open source license and is completely free of development and run-time licensing fees.

Support:

Support for the Open Source deployment of TAO is widely available with a number of commercial companies, supplying support contracts throughout the world.

8.2 Future Research

Throughout the research and evaluation of the CORBA Notification Service the newly defined Data Distribution Service specification has been compared to the CORBA Notification Service. The specification also defined by the OMG is now implemented and distributed as part of TAO. Like CORBA, this is an open specification which contains a Data Centric Publish-Subscribe layer and a Data-Local Reconstruction Layer. However, this is not a loosely coupled implementation as there is no event channel; the supplier and consumer are directly linked in a Publish-Subscribe design. This confers a disadvantage in scalability, but an advantage in the area of performance. As the performance of the TAO Notification service is far beyond any of the other technologies described in this paper, the DDS implementations are said to be quicker than CORBA as it is a tightly coupled design with no event channel.

References

List of References:

Web Sites:

1 . About the Object Management Group
http://www.omg.org/

2. The CORBA Directory
http://corba-directory.omg.org/

3. CORBA ORB Architecture by Douglas C. Schmidt.
http://www.cs.wustl.edu/~schmidt/corba-overview.html

7. Research on High-performance CORBA by Douglas C. Schmidt.
http://www.cs.wustl.edu/~schmidt/corba-research-performance.html

29. CORBA Availability
http://www.csdl.tamu.edu/ohs/tech/framework/corba/availability.html

37. Delivery process for events in WS Eventing
http://www.codeproject.com/soap/WSEventing.asp

38. Mapping a StructuredEvent to a JMS Message.
http://www.iona.com/support/docs/orbix/6.1/develop/messaging/java/bridge_message3.htm

39. Wikipedia on CORBA
http://en.wikipedia.org/wiki/CORBA

40. Middleware Platforms: CORBA Event Service
http://sar.informatik.hu-berlin.de/teaching/_previous-years/2006-w%20Middleware,%20Platforms/lab/lab-3%20(CORBA%20events)/lab-3.htm

Standards and Specifications:

6. Version 1.2 of the Event Service Specification published by the OMG on 4th October 2002
http://www.omg.org/docs/formal/04-10-02.pdf

9. Version 1.1 of the Notification Service Specification published by the OMG on 13th October 2004
http://www.omg.org/docs/formal/04-10-13.pdf

17. Java Message Service Specification Final release 2002
http://sdlc-esd.sun.com/ESD4/JSCDL/jms/1.1-fr/jms-1_1-fr-spec.pdf?AuthParam=1181410685_ce80c7f715772f6ef4a644a26ec9f4d3&TUrl=an1npDpbKod7kSYrROhENTonIuc4W0D1Lc4nXz+pGFFranixdCdgxDTPbW4=&TicketId=dVF4OA9NNOoy+w==&GroupName=SDLC&BHost=sdlc1i.sun.com&FilePath=/ESD4/JSCDL/jms/1.1-fr/jms-1_1-fr-spec.pdf&File=jms-1_1-fr-spec.pdf

22. Web Services Eventing Specification, W3C Member Submission 15th March 2006. Published by W3C.
http://www.w3.org/Submission/WS-Eventing/

23. Microsoft Developer Network on Global XML Web Services Specifications
Web Services Eventing (WS-Eventing) 2004
http://msdn2.microsoft.com/en-us/library/ms951233.aspx

31. EDUCOM/NLII Instructional Management Systems - Specifications Document Version 0.5 (April 29, 1998)
http://www.desire2learn.com/patent/IMS/educom-nlii-instructional-management.pdf

35. Object Computing Inc (OCI)
http://www.theaceorb.com/

36. Defense Information Infrastructure Common Operating Environment (DII COE) http://www.sei.cmu.edu/str/descriptions/diicoe.html

Evaluations:

30. **VTransIT** CORBA Performance Evaluation www.vtransit.net/ref/CORBA%20Product%20Evaluation%20Matrix1.0.xls

Literature:

11. Java in Distributed Systems written by Marko Boger and published by Wiley (2001). ISBN: 0-471-49838-6 Chapter 1 Java on Distributed Systems & Chapter 4 on RMI

12. Developing Distributed and E-commerce Applications 2nd Edition by Darrel Ince and published by Pearson (2004) ISBN: 0-321-15422-3 Chapter 11 on Internet Security

14. Java Enterprise Best Practices 1st Edition written by The O'Reilly Java Authors (December 2002) ISBN: 0-596-00384-6 Chapter 6 on RMI Best Practices

16. Java Enterprise in a NutShell by David Flanagan, Jim Farley, William Crawford and Kris Magnusson and Published by O'Reilly and Associates (September 1999) ISBN 1-56592-483-5E Chapter 3 on Remote Method Invocation

18. The J2EE 1.4 Tutorial written by Eric Armstrong written by Jennifer Ball, Stephanie Bodoff, Debbie Bode Carson, Ian Evans, Dale Green, Kim Haase, Eric Jendrock. Released on December 5th 2005. Chapter 33 The Java Message Service API

24. Building Web Services with Java: Making sense of XML, SOAP, WSDL, and UDDI 2nd Edition written by Steve Graham et al. ISBN: 0672326418. Chapter 8

33. TAO Developer Guide 1.3a Building a standard in Performance. Published by Object Computing Inc. OCI Part Number 530-02 (2003)

Articles:

5. An Overview of the CORBA Event Service by Douglas C. Schmidt
http://www.cs.wustl.edu/~schmidt/PDF/coss4.pdf

10. JMS and CORBA Notification Interworking by Steve Trythall on 12th December 2001
http://www.onjava.com/pub/a/onjava/2001/12/12/jms_not.html

13. Java Remote Method Invocation - Distributed Computing for Java by Sun MicroSystems http://java.sun.com/javase/technologies/core/basic/rmi/whitepaper/index.jsp

15. Distributed Network Coding without the Pain by Sing Li Published: 6 April 2004
http://www.itarchitect.co.uk/articles/display.asp?id=1

19. Open Grid Services Infrastructure (OGSI)
Version 1. June 23rd 2003.
http://www.ogf.org/documents/GFD.15.pdf

20. Open Grid Service Infrastructure Primer by Global Grid Forum (2004)
www.ogf.org/documents/GFD.31.pdf

21. Java Web Services at a Glance
http://java.sun.com/webservices/ by Sun Microsystems.

25. Publish-Subscribe Notification for Web services (Version 1) written by Steve Graham (IBM) et al (June 2004)
http://www.ibm.com/developerworks/library/ws-pubsub/WS-PubSub.pdf

26. Toward Converging Web Service Standards for Resources, Events, and Management written by Kevin Cline (Intel) et al (March 2006)
http://msdn2.microsoft.com/en-us/library/Aa480724.aspx

27. From Open Grid Services Infrastructure to WSResource Framework: Refactoring & Evolution written by Karl Czajkowski et al (June 2004)
http://www.globus.org/wsrf/specs/ogsi_to_wsrf_1.0.pdf

28. CORBA Junction: CORBA 3.0 Notification Service By Dave Bartlet (01st May 2001)
http://www.ibm.com/developerworks/webservices/library/co-cjct8/

34. CORBA's Notification Service By Letha Etzkorn, Joel Sherrill, and Ron O'Guin Embedded.com (09/16/02)
http://www.embedded.com/story/OEG20020913S0058

Papers:

8.

Paper Title: Designing and Optimizing a Scalable CORBA Notification Service

Authors: Pradeep Gore, Douglas C. Schmidt, Carlos O'Ryan, and Ron Cytron

Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001), Snowbird, Utah, June 18, 2001.

Link: http://www.cs.wustl.edu/~schmidt/PDF/notify.pdf

32.

Paper Title: Strategies for integrating systems with CORBA® - A Borland White Paper

Authors: Brenton Camac, Ke Jin, and Dave Stringer (March 2004)

Link: http://www.borland.com/resources/en/pdf/white_papers/corba_strategies_integrating_ossj_systems.pdf

News:

4. Prismtech News on ‘Dr. Douglas C. Schmidt Joins PrismTech’
http://www.prismtechnologies.com/section-item.asp?sid6=&sid5=&sid4=&sid3=&sid2=14&sid=29&id=673

Appendix A - Annotated Bibliography

Pradeep Gore, Douglas C. Schmidt, Carlos O'Ryan, and Ron Cytron (2001). Designing and Optimizing a Scalable CORBA Notification Service. Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems. Retrieved August 2007 from: http://www.cs.wustl.edu/~schmidt/PDF/notify.pdf

Dr. Douglas C. Schmidt, a former professor of Computer Science and an Associate Chair of Computer Science and Engineering in Vanderbilt University, Nashville who is currently working for Prismtech in the role of Principal Technologist. He is internationally renowned and widely cited expert on distributed computing middleware, object-oriented patterns and frameworks, and distributed real-time and embedded (DRE) systems. In this paper he describes the CORBA Event Service and tell us the design principles of how the CORBA Notification Service extended the Event Service functionality.

Brenton Camac, Ke Jin, and Dave Stringer (2004). Strategies for integrating systems with CORBA®. A Borland White Paper. Retrieved August 2007 from http://www.borland.com/resources/en/pdf/white_papers/corba_strategies_integrating_ossj_systems.pdf

Brenton Camac is a Consultant with Camac IT Ltd and author of “Code examples showing J2EE and CORBA interoperability”. In this paper the authors review different strategies for interoperability of CORBA and J2EE platforms for the purpose of Operational Support Systems developed in Java (OSS/J). The authors discuss the complicated integration of the CORBA Notification Service in there analysis.

Steve Trythall (2001). JMS and CORBA Notification Interworking. Retrieved August 2007 from http://www.onjava.com/pub/a/onjava/2001/12/12/jms_not.html

Steve Trythall is a senior manager for Prismtech Ltd, and has over two decades experience in the computing industry including a variety of senior positions on large successful product developments. He has spent the last 7 years in product management bringing to market several CORBA and J2EE middleware products that established the PrismTech as the market leader in it is segment. In this article the author describes the requirements integration of the Java Message Service and the CORBA Notification Service. It compares the messaging structure of both specifications and defines similarities in the structure and filtering and how they can be integrated.

Sing Li (2004). Distributed Network Coding without the Pain. Retrieved August 2007 from http://www.itarchitect.co.uk/articles/display.asp?id=1

Sing Li is a consultant, system architect, open source software contributor, and freelance writer specializing in Java technology, embedded and distributed systems design. He has authored or co-authored several books on the topic, including Professional Apache Tomcat (Wrox), Professional JSP 2 (Apress), Early Adopter JXTA (Wrox) and Professional Jini (Wrox). The author in this article discusses Java Remote Method Invocation and how it can be used with the IIOP protocol allowing for interoperability with CORBA.

Dave Bartlet (2001). CORBA Junction: CORBA 3.0 Notification Service. Retrieved August 2007 from http://www.ibm.com/developerworks/webservices/library/co-cjct8/

Dave Bartlet is the author of “Hands-On CORBA with Java”, a 5-day course presented via public sessions or in-house to organizations. The author in this article discusses the Quality of Service (QoS) and filtering features in the CORAB Notification Service. It also defines the Notification StructuredEvent message type and it is internal structure.

Letha Etzkorn, Joel Sherrill, and Ron O'Guin (2002) CORBA's Notification Service. Retrieved August 2007 from http://www.embedded.com/story/OEG20020913S0058

Letha Etzkorn is an assistant professor at the University of Alabama in Huntsville, where she has recently worked on two CORBA contracts for the US Army. Joel Sherrill is director of R&D at OAR. He has 15 years experience in commercial and military real-time embedded applications and related research. He is active in the free software community as the maintainer of the RTEMS RTOS and as a member of the Steering Committee for the Free Software Foundation's GNU Compiler Collection. Ron O'Guin is executive vice president of OAR. He has more than 25 years of experience in the development of real-time applications, visual simulations, and trainers in the video production, telephony, and military domains. He is a principal author of the open-source RTEMS RTOS. In this article the authors describe different choices of middleware in the distributed computing environment and compare them in a project developed for the army.