UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 5

"PROPUESTA DE TECNOLOGÍA ADECUADA PARA MEJORAR LA VERSIÓN 1.0 DEL MIDDELWARE DEL GUARDIÁN DEL ALBA"

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: Luis Angel Ravelo Hernández

Tutor: Ing. Maikel Pérez Javier

Ciudad de la Habana

Junio 2009

DATOS DE CONTACTO

Tutor:

Ing. Maikel Pérez Javier

Graduado de Ingeniería Informática.

e-mail: mperezj@uci.cu

Profesor instructor con tres años de experiencia docente en la Universidad de las Ciencias Informáticas (UCI) y tres años de experiencia en el desarrollo de software, específicamente en el desarrollo del Middleware del Sistema de Supervisión, Control y Adquisición de Procesos Industriales (SCADA).

AGRADECIMIENTOS

Son muchas las personas que han hecho posible esta tesis, y por ello quiero dejar constancia de mi agradecimiento a todos ellos en esta primera página, que sin embargo es la última que escribo.
En primer lugar, quiero agradecer esta tesis a mi tutor Maikel Pérez Javier por su apoyo y consejos durante la realización de este trabajo.
A mis compañeros del proyecto SCADA Guardián del Alba y a los antiguos compañeros de aula por escuchar y contribuir a las diferentes propuestas e ideas que dieron como resultado esta tesis.
A mis amigos que están lejos y a los que están a mi lado cada día y me apoyan.
A los que siempre confiaron en mí y a los que no, porque esos provocaron que me esforzara más cada día.
En el plano personal quisiera agradecer y dedicar esta tesis a mi familia, en especial a mis padres, Ismary y Pedro Félix, porque si he llegado hasta aquí ha sido gracias a ellos y porque siempre me han apoyado y animado a seguir con mis estudios a pesar de que esto suponga estar lejos de casa. A mi novia Yaima por ser mi mayor apoyo en estos últimos años y darme las fuerzas que necesito en el momento preciso, por confiar siempre en mí.
A mis tíos Carlos Alberto y Juan Carlos por guiarme y animarme a entrar en el mundo de la informática. A mis abuelas Erelia y Lidia, por sus consejos sabios.
A mis tíos Norelvis y Sory por confiar y creer que este día iba a llegar.
A mi primo Yaniel por tenerme como ejemplo obligándome a ser cada día mejor profesional y mejor persona.
Gracias a todos.

DEDICATORIA

A mis padres Ismary y Pedro Félix por su amor y apoyo en todo momento.
A mi novia Yaima por su comprensión y paciencia.
A tío juanqui y tío carlos por creer siempre en mí.
A mi abuela Lidia por su amor y preocupación.
A mi primo Yaniel.

RESUMEN

El proyecto SCADA Nacional o Guardián del ALBA que está siendo desarrollado en la Universidad de las Ciencias Informáticas, en la facultad 5, implementa en una de sus líneas, un Middleware, instrumento usado como sistema de comunicación entre sus módulos. En la versión 1.0 del Guardián del ALBA se detectaron algunos problemas en el Middleware, por lo que se realiza esta investigación para estudiar las tecnologías que su utilizan en la actualidad para el desarrollo de este tipo de software y entre ellas seleccionar la más adecuada, para mejorar las condiciones de la versión actual.

Además de seleccionar la tecnología, se implementa un prototipo funcional que demuestra las ventajas y mejoras que reportaría el uso de la misma.

PALABRAS CLAVES

Middleware, tecnologías y prototipo funcional.

ÍNDICE

INTRODUCCIÓN

La Universidad de las Ciencias Informáticas (UCI) surgida como uno de los programas de la Revolución en la Batalla de Ideas es una universidad de excelencia, única en su tipo, dedicada al estudio y a la producción del software, que tiene como principal objetivo la formación de ingenieros informáticos altamente calificados y además pretende convertirse en uno de los principales centros del país en la producción y distribución del software. Para ello, en la misma se vincula el estudio directamente con la producción, por lo que se desarrollan proyectos productivos de gran importancia. El desarrollo de estos proyectos, representa un significativo paso de avance en el ámbito de la industria del software, en el cual nuestro país pretende desarrollarse ampliamente.

En la Universidad actualmente existen una gran cantidad y variedad de proyectos productivos, los cuales se encuentran distribuidos en los distintos polos productivos que han sido creados en cada facultad. Uno de los polos de la Facultad 5 es el de Hardware y Automática, en el cual se desarrollan varios proyectos relacionados con esta línea de trabajo. En el año 2006 se comenzó a desarrollar un software que une a la república bolivariana de Venezuela y a nuestro país, el SCADA (Sistema de Supervisión, Control y Adquisición de Datos) Nacional o Guardián del Alba como se le conoce actualmente, el cual es desarrollado para PDVSA. Desde sus inicios se define una línea de trabajo nombrada Middleware que es la encargada de la trasmisión de eventos y que requiere un rendimiento óptimo en cuanto a su tarea. Al terminar la versión 1.0 del Guardián del ALBA y puesto en marcha un piloto en un patio tanque en la hermana república bolivariana se detecta que la velocidad de trasmisión de eventos, del cual el responsable es el Middleware, no es el óptimo por lo que se debe mejorar para que el SCADA tenga un mayor rendimiento y pueda controlar el proceso del petróleo en lugares más grandes que el actual. También se detecta que el mecanismo de persistencia no es lo suficientemente fiable, lo que trae como consecuencia que cuando se suspende y se reinicia el servicio, en ocasiones, no reconoce las conexiones que antes tenía. Además presentó problemas con las comunicaciones cuando se usa más de una interfaz de red, imposibilitando las mismas.

Por lo antes explicado surge la principal problemática:

Problema Científico: ¿Cómo mejorar las condiciones de la versión 1.0 del "MIDDLEWARE" implementado en el "Guardián del ALBA" atendiendo a los problemas detectados?

Como **Objeto de estudio** de este trabajo se tiene las tecnologías libres que permitan desarrollar sistemas de comunicación distribuida (Middleware) y como **campo de acción** las características y funcionalidades que proveen las tecnologías libres para el desarrollo de "Middleware". De acuerdo con el Problema Científico planteado la **Idea a Defender** definida es la siguiente:

Implementando el Middleware del SCADA actual, con otra tecnología, se puede lograr un mayor rendimiento dando una solución de comunicación distribuida más robusta y eficiente entre los módulos del SCADA.

El Objetivo General que se desea alcanzar es: Desarrollar un prototipo funcional del Middleware basado en otra tecnología que permita corregir los problemas detectados en la versión 1.0 desarrollada en el "Guardián del Alba".

Para llevar a cabo este trabajo y dar cumplimiento al objetivo propuesto se concibieron las siguientes **Tareas de Investigación**:

1. Análisis de las tecnologías libres existentes que permiten desarrollar sistemas de comunicación distribuidos.
2. Definición de las características que debe soportar la solución esperada.
3. Selección de la tecnología adecuada que cumpla con las condiciones requeridas.
4. Definición de las funcionalidades que soportará el prototipo funcional basado en la tecnología seleccionada.
5. Diseño de clases para las funcionalidades especificadas.
6. Implementación del prototipo funcional.
7. Desarrollo de pruebas.
8. Evaluación de los resultados.

Para darle cumplimiento a las tareas se guiará la investigación a través de los métodos científicos de investigación Teóricos y Empíricos, los cuales posibilitarán descubrir, analizar y sistematizar los resultados obtenidos, para llegar a conclusiones confiables que permitan resolver el problema. Dentro de los métodos teóricos se utilizará el método "Analítico-sintético", con éste se podrá analizar cada uno de los elementos de manera independiente, posibilitando una mayor capacidad de comprensión y de síntesis sobre los aspectos más importantes. También se utilizará el método "Inductivo-deductivo" durante toda la investigación, para llegar a conclusiones y hacer generalizaciones acerca la tecnología

a usar. Otro método que será referenciado es el de la “Modelación”, el cual permitirá realizar abstracciones con el objetivo de explicar la realidad y la posible solución del problema de investigación.

El método empírico que será referenciado es la “Observación”, este método permite adquirir información necesaria y puede utilizarse en cualquiera de las fases de la investigación, además ofrece un gran acercamiento a la realidad y permite ver la posible solución del problema desde diferentes ángulos.

Este documento está estructurado en tres capítulos, en los que se expone todo el proceso de desarrollo de este trabajo:

CAPÍTULO 1: Fundamentación teórica y evaluación de las tecnologías.

CAPÍTULO 2: Diseño de la solución propuesta.

CAPÍTULO 3: Implementación de la solución y evaluación de los resultados.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA Y EVALUACIÓN DE LAS TECNOLOGÍAS

1.1. Introducción.

Desde los inicios de la computación esta ha ido evolucionando con gran rapidez, en sus inicios una computadora solamente podía realizar operaciones matemáticas básicas y tenían un tamaño exageradamente grande. A medidas que fue transcurriendo el tiempo, las funciones de una computadora iban siendo más cada día y su tamaño fue disminuyendo, pero eran usadas solamente por algunas empresas muy selectas. Los ordenadores actuales, ya sean portátiles o de sobremesa, tienen mayores prestaciones a medida que trascurre el tiempo y a su vez están introducidos en el quehacer cotidiano de una persona.

Existen dos razones que permitieron esta acelerada evolución y fue desde la década del setenta, cuando se desarrollaron los primeros microprocesadores permitiendo reducir el tamaño y el costo a los ordenadores y además aumentaron considerablemente las capacidades de los mismos y que un número mayor de personas tuvieran acceso. Otra razón fue el desarrollo de las redes de área local y de las comunicaciones que permitieron la conexión de varias computadoras con posibilidad de transferencia de datos a altas velocidades.

A partir de estos avances es que se comienza a tratar el término de sistemas distribuidos que en la actualidad se ha popularizado tanto, y que tiene como ámbito el estudio de redes, ejemplo de ellas son Internet, redes de telefonía móvil, redes de empresas, redes de corporaciones, etc.

1.2. Sistemas Distribuidos

Una definición de un sistema distribuido muy acertada es la que proponen Tanenbaum y Van Renesse, 1985:

"*Un sistema distribuido es aquél al que sus usuarios ven como un ordinario sistema operativo centralizado; sin embargo, se ejecuta en diferentes e independientes CPUs. El concepto clave aquí es la transparencia; en otras palabras, el uso de diversos procesadores deberá ser invisible (transparente) al usuario. Otra forma de expresar esta misma idea es diciendo que el usuario verá al sistema como un uniprocesador virtual y no como una colección de máquinas diferentes.*"

Además por sistema distribuido se puede entender:

"Sistema en el cual múltiples procesadores autónomos, posiblemente de diferente tipo, están interconectados por una subred de comunicación para interactuar de una manera cooperativa en el logro de un objetivo global." **(Lelann, 1981)**

"Conjunto de computadores independientes que se muestran al usuario como un sistema único coherente." **(Tanenbaum, 2001)**

"Sistema en el cual componentes de hardware y software, localizados en computadores en red, se comunican y coordinan sus acciones sólo por paso de mensajes." **(Coulouris, 2002)**

"Colección de ordenadores autónomos enlazados por una red y soportados por aplicaciones que hacen que la colección actúe como un servicio integrado" **(Universidad Politécnica de Cataluña, 2007)**

Se puede resumir que un sistema distribuido no es más que un conjunto de elementos, tanto de procesamiento como de almacenamiento que están conectados a través de una red y que para un usuario del sistema se comparta como un único computador.

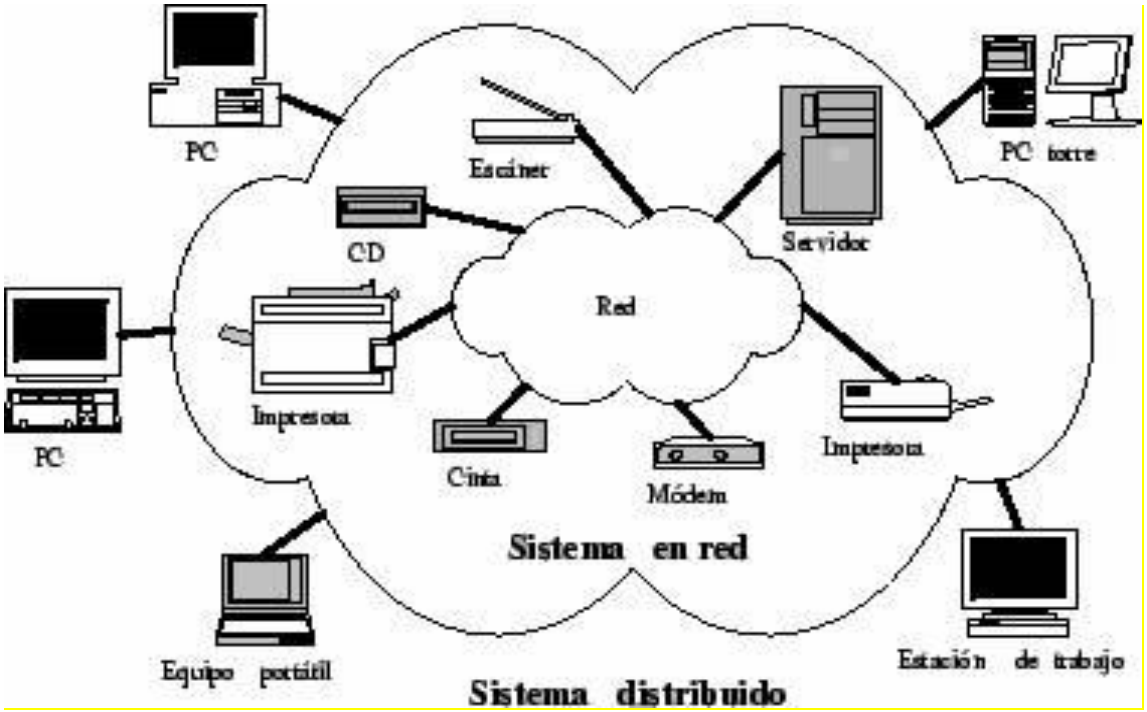

Figura 1: Ejemplo de sistema distribuido.

1.2.1. Características de los sistemas distribuidos.

Existen seis características que son responsables de la utilidad de los sistemas distribuidos: compartir recursos, apertura, concurrencia, tolerancia a fallas y transparencia. Se hace notar que estas características no son consecuencia automática de la distribución; el software del sistema y de las aplicaciones debe ser cuidadosamente diseñado para asegurar que ellas sean conseguidas. Para mayor entendimiento de cada una de las características mostramos un resumen.

Compartir recursos :

El término recurso caracteriza el conjunto de cosas que pueden ser compartidas. El rango va desde componentes de hardware (discos e impresoras) a entidades de software (archivos, bases de datos u otros objetos de datos).

Los recursos de un computador multiusuario son normalmente compartidos entre todos los usuarios, pero los usuarios de estaciones de trabajo individuales y PC conectadas en red no obtienen automáticamente los beneficios de compartir recursos.

Los recursos en un sistema distribuido están físicamente encapsulados dentro de uno de los computadores y pueden ser accesados por otros mediante comunicación.

Para compartir efectivamente cada recurso, éste debe ser administrado por un programa que ofrece una interfaz de comunicación que posibilita que el recurso sea accesado, manipulado y actualizado confiable y consistentemente.

El término administrador de recurso es usado para denotar el módulo de software que administra un conjunto de recursos de un tipo particular.

Sistema abierto:

La apertura de un sistema computacional es la característica que determina si el sistema puede ser ejecutado en varias formas.

Un sistema puede ser abierto o cerrado con respecto a extensiones de hardware (por ejemplo, periféricos, memoria o interfaces de comunicación) o con respecto a extensiones de software (adiciones de características al sistema operativo, protocolos de comunicación y servicios para compartir recursos).

La apertura está asociada a la especificación y documentación de las claves en las interfaces de software de un sistema y dejarlas disponibles a los desarrolladores de software. En una palabra, las claves de interfaces son publicadas.

Los sistemas distribuidos abiertos están basados en la provisión de un mecanismo de comunicación interprocesos e interfaces publicadas. El término SD abierto sirve para enfatizar que son extensibles.

Pueden ser extendidos al nivel de hardware agregando computadores a la red y en el nivel de software por la introducción de nuevos servicios.

Concurrencia:

Cuando varios procesos se ejecutan en un único computador diremos que se están ejecutando concurrentemente.

Si el computador está equipado con una CPU, esto implica la intercalación de la ejecución de porciones de cada proceso (timesharing).

Si el computador tiene N procesadores, entonces N procesos pueden ser ejecutados simultáneamente (en paralelo).

En un SD que está basado en el modelo de compartición de recursos descrito, las oportunidades de la ejecución en paralelo ocurren por dos razones:

- Muchos usuarios invocan simultáneamente comandos o interactúan con aplicaciones.
- Muchos procesos del servidor corren concurrentemente, cada uno respondiendo a diferentes solicitudes desde procesos clientes.

Para resumir, la concurrencia y la ejecución en paralelo caen naturalmente en SD. Los accesos concurrentes y actualizaciones de recursos compartidos deben ser sincronizados.

Escalabilidad:

Los sistemas operan efectiva y eficientemente en muchas diferentes escalas.
El sistema distribuido práctico más pequeño, probablemente consista de dos estaciones de trabajo y un servidor de archivos. Un sistema distribuido construido como LAN puede contener varios cientos de estaciones de trabajo y muchos servidores de archivos, servidores de impresión y otros tipos de servidores.

Varias LAN frecuentemente son interconectadas para formar interredes, y estas pueden contener miles de computadores que forman un único sistema distribuido, habilitando recursos para ser compartidos entre ellos.

El software de sistema y aplicaciones no debería necesariamente cambiar cuando el sistema incrementa su escala.

Tolerancia a fallas:

Los sistemas a veces fallan. Cuando fallan, es debido a software o hardware.

El diseño de sistemas tolerantes a fallas está basado en dos enfoques:

1. Redundancia de hardware: Uso de componentes redundantes.

2. Recuperación del software: Diseño de programas para recuperarse de fallas.

Para producir sistemas tolerantes a fallas de HW, dos computadores interconectados se emplean para una misma aplicación.

En un sistema distribuido la redundancia puede ser planificada (pueden ser replicados servidores individuales que son esenciales en una operación continuada o que contengan aplicaciones críticas).

Cuando falla un servidor los clientes son redirigidos a la réplica. La recuperación de SW involucra diseño de programas que pueden recuperar el estado de datos permanentes cuando una falla es detectada.

Transparencia:

La transparencia está definida como el encubrimiento al usuario y al programador de aplicaciones de la separación de componentes en un sistema distribuido, tal que el sistema es percibido como un todo más que como una colección de componentes independientes.

Las implicaciones de transparencia son la mayor influencia en el diseño de software de sistema. La ISO identifica 8 formas de transparencia:

1. El *acceso transparente* que habilita objetos de información local y remotos para ser accesados usando operaciones idénticas.

2. La *transparencia de ubicación* que habilita objetos de información local para ser accesados sin conocer sus ubicaciones.
3. La *transparencia de concurrencia* que habilita varios procesos para operar concurrentemente usando objetos de información local compartidos sin interferencias entre ellos.
4. La *transparencia de replicación* que habilita múltiples instancias de objetos de

información local para ser usados para incrementar la confiabilidad y rendimiento sin conocimiento de las réplicas por los usuarios o programas de aplicación.

5. La *transparencia a fallas* que habilita el encubrimiento de fallas, permitiendo a los usuarios y aplicaciones completar sus tareas a pesar de las fallas de componentes de hardware o software.
6. La *transparencia de migración* permite el movimiento de los objetos de información local dentro de un sistema sin afectar la operación de usuarios o aplicaciones.
7. La *transparencia de rendimiento* que permite al sistema ser reconfigurado para mejorar el rendimiento de acuerdo a como varían las cargas.
8. La *transparencia de escalamiento* que permite al sistema y aplicaciones expandirse en su escala sin cambiar a la estructura del sistema o a los algoritmos de aplicaciones.

Las dos más importantes son las transparencias de acceso y de localización; su presencia o ausencia afecta fuertemente la utilización de los recursos distribuidos. A menudo se las denomina a ambas transparencias de red. La transparencia de red provee un grado similar de anonimato en los recursos al que se encuentra en los sistemas centralizados.

1.2.2. Ventajas de los Sistemas Distribuidos.

Por su estructura y sus características los sistemas distribuidos ofrecen grandes ventajas a los usuarios. La primera de ellas es que se adecuan a un concepto común en la vida de los seres humanos: la distribución. Es decir, el hombre comparte información y recursos sin necesidad de complejos protocolos de comunicación u otros requerimientos, simplemente se comunica. Así que los sistemas distribuidos se acercan más al concepto que tiene el hombre de compartir recursos.

El costo y el rendimiento de los sistemas son reducidos, debido a que cada dispositivo del sistema se desenvuelve como una entidad. Es programado por separado y no dependiendo de otros dispositivos. No se requieren costosos sistemas de redes que requieran de una configuración exhaustiva. Además los sistemas de comunicación han mejorado su rendimiento, implementando protocolos que permiten la rápida y efectiva transmisión de datos, incluyendo multimedia.

Otra ventaja importante es la modularidad, ya que un sistema distribuido deja de ser centralizado y cada una de las entidades que lo componen tiene integrado su propio sistema de control. Es decir cada entidad es independiente y es programada cuidadosamente para que tenga un óptimo desempeño dentro de una comunidad de servicios.

Además, un sistema distribuido es expandible y escalable. Por la misma razón que cada entidad es programada de tal forma que sea independiente pueden ser añadidas o removidas de un sistema, a esto se refiere la escalabilidad. En cuanto al concepto de expansión dentro de un sistema significa que, no sólo se pueden agregar nuevos dispositivos periféricos, sino también se pueden agregar procesadores y servidores que incrementen la capacidad de almacenamiento del sistema.

También ofrece como ventaja la disponibilidad de los dispositivos, gracias a que cada entidad debe de estar programada dentro del concepto de redundancia de esta manera sus servicios permanecen al alcance de quien los solicite a pesar de que ocurra alguna falla en ellos.

Por último dentro de las ventajas de un sistema distribuido existe la confiabilidad. Esta cualidad se debe a que implementa todos los conceptos anteriores, en especial la disponibilidad, ya que si un componente está disponible a pesar de que exista alguna falla el usuario tiene como garantía que podrá seguir usándolo.

1.2.3. Desventajas de los Sistemas Distribuidos.

A pesar de que las ventajas que nos proporcionan los sistemas distribuidos son más que las desventajas, si se administran seriamente, no podemos dejar de prestarle atención a estas dificultades pues el conocimiento de las mismas nos permite trabajar sobre ellas y tratar de erradicar las fallas para lograr un producto final con buena calidad.

La principal desventaja que presenta este tipo de sistemas es el software. El diseño, implementación y uso del software distribuido presenta numerosos inconvenientes. Entre los especialistas existe gran diversidad de criterios en cuanto al sistema operativo, lenguaje de programación y aplicaciones adecuados para los sistemas distribuidos, así como cuánto deben saber los usuarios de la distribución y lo que deben hacer los usuarios y el sistema.

Otro problema que presentan estos sistemas tiene que ver con las redes de comunicación y es precisamente la pérdida de mensajes, saturación de la red, etc. Además un problema que surge a partir de compartir datos es la seguridad de los mismos.

1.2.4. Desafíos de los sistemas distribuidos.

Muchos de los desafíos que se discutirán a continuación ya están resueltos pero los futuros diseñadores necesitan estar al tanto y tener cuidado de considerarlo.

- **Heterogeneidad:**

Internet permite que los usuarios accedan a servicios y ejecuten aplicaciones sobre un conjunto heterogéneo de redes y computadoras. Esta heterogeneidad (es decir, variedad y diferencia) se aplica a todos los siguientes elementos:

1. Redes.
2. Hardware de computadoras.
3. Sistemas operativos.
4. Lenguajes de programación.
5. Implementaciones de diferentes desarrolladores.

Para lograr esta heterogeneidad se desarrolla el *middleware* que no es más que el estrato de software que provee una abstracción de programación, así como un enmascaramiento de la heterogeneidad subyacente de las redes, hardware, sistemas operativos y lenguajes de programación.

- **Extensibilidad:**

La extensibilidad de un sistema de cómputo es la característica que determina si el sistema puede ser extendido y reimplementado en diversos aspectos. La extensibilidad de los sistemas distribuidos se determina en primer lugar por el grado en el cual se pueden añadir nuevos servicios de compartición de recursos y ponerlos a disposición para el uso por una variedad de programas.

- **Seguridad:**

La seguridad de los recursos de la información es uno de los aspectos más importantes y tiene tres componentes:

1. Confiabilidad: protección contra individuos no autorizados.
2. Integridad: protección contra la alteración o corrupción.
3. Disponibilidad: protección contra la interferencia que impide el acceso a los recursos.

Aún existen dos desafíos que no han sido resueltos en su totalidad:

1. Ataques de negación de servicio.
2. Seguridad del código móvil.

- **Escalabilidad:**

Se dice que un sistema es escalable si conserva su efectividad cuando ocurre un incremento significativo en el número de recursos y en el número de usuarios.

- **Tratamiento a Fallos:**

Consiste en la posibilidad que tiene el sistema de seguir funcionando después de la ocurrencia de un fallo en alguno de sus componentes. Para esto debe tenerse en cuenta algunos aspectos como la detección de fallos, el enmascaramiento o la tolerancia de estos, la recuperación y la redundancia.

- **Concurrencia:**

Permite que varios clientes accedan a un recurso compartido a la vez.

- **Transparencia:**

Se define transparencia como la ocultación al usuario y al programador de las aplicaciones de la separación de los componentes en un sistema distribuido, de forma que se perciba el sistema como un todo más que como una colección de componentes independientes. Las implicaciones de la transparencia son de gran calado en el diseño del software sistema.

1.2.5. Aplicaciones de Sistemas Distribuidos.

Se puede decir que los campos en los que se aplican este tipo de sistemas son muy diversos debido a las ventajas que nos ofrecen, principalmente por su distribución, que como se ha visto anteriormente los componentes pueden estar bien dispersos. Es por esto que un área en la que ha tenido un impacto muy importante es en los Sistemas Comerciales que por sus características de distribución geográfica y necesidad de acceso a sistemas distintos son ideales para implementarlos siempre que se tengan en cuenta algunas características como la fiabilidad, seguridad y protección. Algunos ejemplos que podemos mencionar son: Sistemas de reservas de líneas aéreas, aplicaciones bancarias, sistemas de gestión de almacenes o sistemas de automatización industrial.

Otra de las áreas en las que podemos ver el desarrollo de estos sistemas es en las Redes WAN ya que debido al gran aumento de este tipo de redes se ha incrementado el intercambio de información a través de ellas utilizando algunos de los servicios que brinda Internet como Correo Electrónico, Servicio de Noticias, Transferencia de Archivos o la propia World Wide Web.

Últimamente se han incorporado a utilizar este tipo de sistemas las áreas relacionadas con las Aplicaciones Multimedia, las mismas imponen ciertas necesidades de hardware para garantizar una velocidad y regularidad de transferencia de gran cantidad de datos en sistemas de videoconferencias, tele vigilancia, juegos multiusuario, entornos virtuales de aprendizaje entre otros. Por último no podemos dejar de mencionar las Áreas de la Informática aplicada a los Sistemas Distribuidos en dónde se tienen en cuenta todas las variedades de aplicaciones de los sistemas distribuidos tales como comunicaciones, sistemas operativos distribuidos, bases de datos distribuidas, servidores distribuidos de ficheros, lenguajes de programación distribuidos o sistemas de tolerancia de fallos.

Con todo lo anteriormente visto se puede concluir con que los Sistemas Distribuidos abarcan una cantidad de aspectos considerables lo que determina que el desarrollo de los mismos sea muy complejo. En el siguiente epígrafe se abordará un sistema distribuido más específico, SCADA (Supervisory Control And Data Acquisition).

1.3. SCADA.

Como se pudo apreciar en el epígrafe anterior uno de los ejemplos de aplicación de sistemas distribuidos son los sistemas de automatización industrial. Uno de estos sistemas se denomina SCADA que es el acrónimo de Supervisory Control And Data Acquisition (Supervisión, Control y Adquisición de Datos).

Un SCADA es un sistema basado en computadores que permite supervisar y controlar a distancia una instalación de cualquier tipo, donde el lazo de control es generalmente cerrado por el operador aunque hoy en día es fácil hallar un sistema SCADA realizando labores de control automático en cualquiera de sus niveles, aunque su labor principal sea de supervisión y control por parte del operador. **(Modesti, 2008)**

1.3.1. Definición de SCADA.

Un sistema SCADA se define como una aplicación especialmente diseñada para controlar a través de un ordenador y con dispositivos de campo, las operaciones de control, supervisión y registro de datos de cualquier proceso industrial gobernado por autómatas programables o redes de autómatas.**(Wikipedia, 2009)**

Los sistemas SCADA utilizan la computadora y tecnologías de comunicación para automatizar el monitoreo y control de procesos industriales. Estos sistemas son partes integrales de la mayoría de los ambientes industriales complejos o muy geográficamente dispersos ya que pueden recoger la información de una gran cantidad de fuentes rápidamente, y la presentan a un operador en una forma amigable. Los sistemas SCADA mejoran la eficacia del proceso de monitoreo y control proporcionando la información oportuna para poder tomar decisiones operacionales apropiadas.

Los primeros SCADA eran simplemente sistemas de telemetría que proporcionaban reportes periódicos de las condiciones de campo vigilando las señales que representaban medidas y/o condiciones de estado en ubicaciones de campo remotas. Estos sistemas ofrecían capacidades muy simples de monitoreo y control, sin proveer funciones de aplicación alguna. Los ordenadores agregaron la capacidad de programar el sistema para realizar funciones de control más complejas.

Los primeros sistemas automatizados SCADA fueron altamente modificados con programas de aplicación específicos para atender a requisitos de algún proyecto particular.

La mayoría de los sistemas SCADA que son instalados hoy en día, se están convirtiendo en una parte integral de la estructura de gerenciamiento de la información corporativa. Estos sistemas ya no son vistos por la gerencia simplemente como herramientas operacionales, sino como un recurso importante de información. En este papel continúan sirviendo como centro de responsabilidad operacional, pero también proporcionan datos a los sistemas y usuarios fuera del ambiente del centro de control, que dependen de la información oportuna en la cual basan sus decisiones económicas cotidianas. La arquitectura de los sistemas de hoy integra a menudo muchos ambientes de control diferentes, tales como tuberías de gas y aceite, en un solo centro de control.

Para alcanzar un nivel aceptable de tolerancia de fallas con estos sistemas, es común tener ordenadores SCADA redundantes operando en paralelo en el centro primario del control, y un sistema de reserva del mismo situado en un área geográficamente distante. Esta arquitectura proporciona la transferencia automática de la responsabilidad del control de cualquier ordenador que pueda llegar a ser inaccequible por cualquier razón, a una computadora de reserva en línea, sin interrupción significativa de las operaciones.

Resumiendo la definición, se puede decir entonces que un SCADA es un sistema industrial de mediciones y control que consiste en una computadora principal o máster (generalmente llamada Estación Principal, Máster Terminal Unit o MTU); una o más unidades de control obteniendo datos de campo (generalmente llamadas estaciones remotas, Remote Terminal Units, o RTU's); y una colección de software estándar usados para monitorear y controlar remotamente dispositivos de campo. El control puede ser automático, o iniciado por comandos de operador.

Dentro de las funciones básicas realizadas por un sistema SCADA están las siguientes:

a) Recabar, almacenar y mostrar información, en forma continua y confiable, correspondiente a la señalización de campo: estados de dispositivos, mediciones, alarmas, etc.

b) Ejecutar acciones de control iniciadas por el operador, tales como: abrir o cerrar válvulas, arrancar o parar bombas, etc.

c) Alertar al operador de cambios detectados en la planta, tanto aquellos que no se consideren normales (alarmas) como cambios que se produzcan en la operación diaria de la planta (eventos). Estos cambios son almacenados en el sistema para su posterior análisis.

d) Aplicaciones en general, basadas en la información obtenida por el sistema, tales como: reportes, gráficos de tendencia, historia de variables, cálculos, predicciones, detección de fugas, etc.

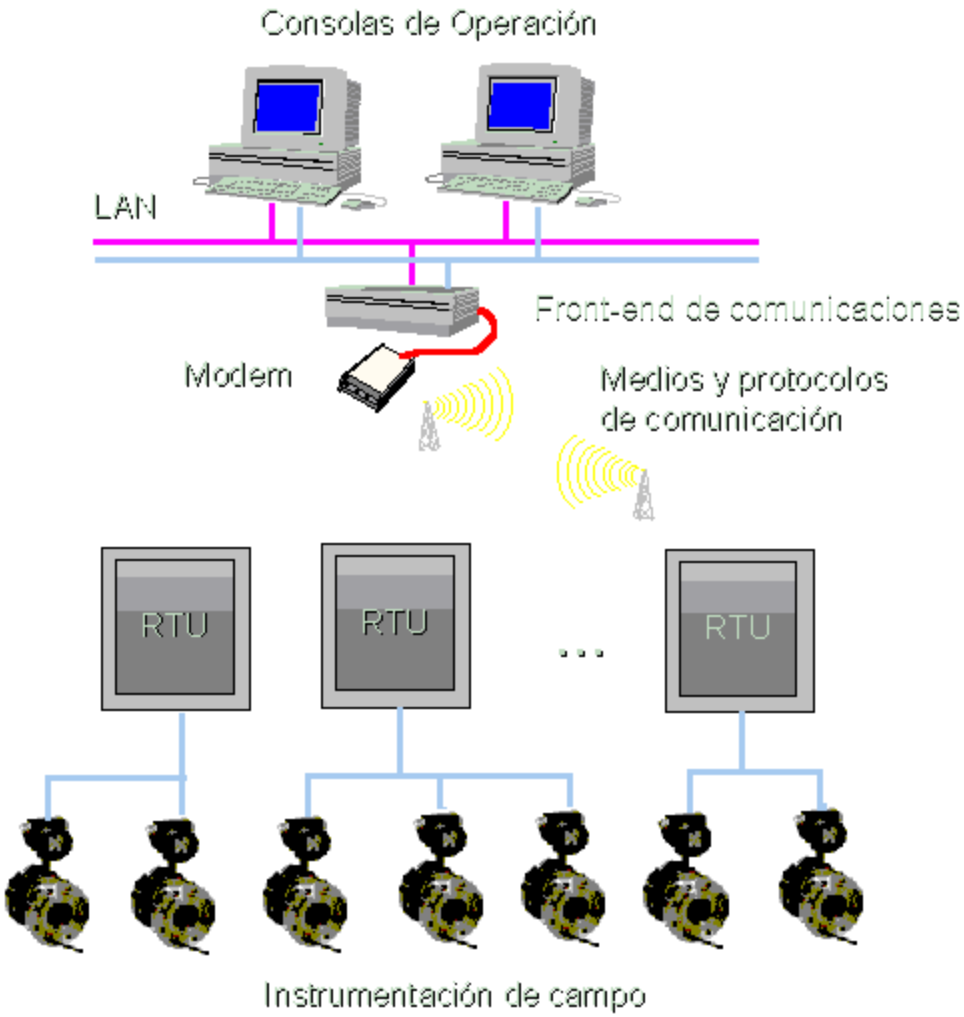

Figura 2: Ejemplo de un SCADA.

1.4. Middleware.

Como se dijo anteriormente, una parte muy importante de los Sistemas Distribuidos es su Subsistema de Comunicación. Éste es el encargado de gestionar los mensajes y el traslado de información de un componente del sistema a otro. Esto permite que se organice el trabajo entre todos los elementos que lo componen alcanzándose un mayor uso de los recursos y obteniéndose resultados concretos en un corto período de tiempo. La mayoría de los Sistemas SCADA que más se comercializan en la actualidad, utilizan como Subsistema de Comunicación un Middleware.

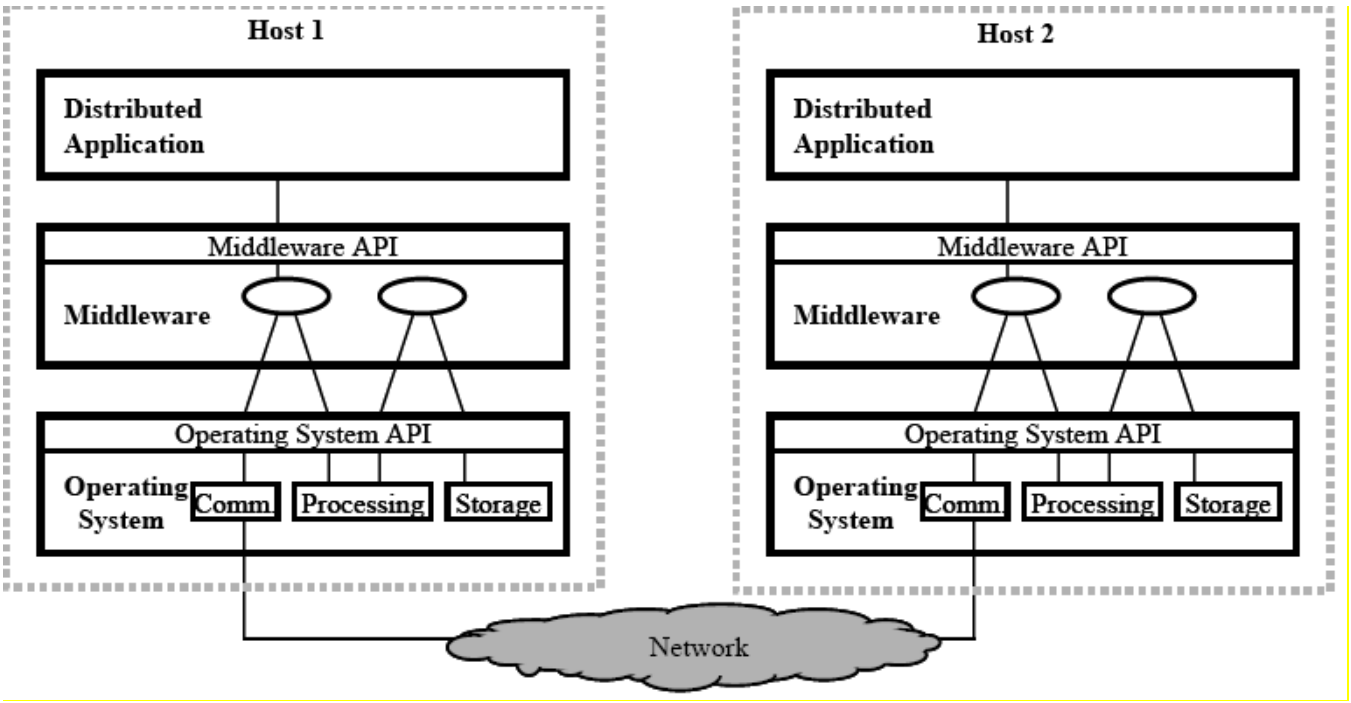

Figura 3: Ejemplo de middleware.

1.4.1. Definición de Middleware.

Se puede decir que un middleware no es más que un software que se encuentra entre la capa de aplicaciones y la capa de red y el sistema operativo y que tiene como principal tarea la abstracción de la heterogeneidad de las redes, los sistemas operativos y los lenguajes de programación.

El Middleware es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red). El Middleware nos abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API para la fácil programación y manejo de aplicaciones distribuidas. Dependiendo del problema a resolver y de las funciones necesarias, serán útiles diferentes tipo de servicios de middleware. **(Wikipedia, 2008)**

Es muy común ver que los middleware se agrupen en diferentes clasificaciones en dependencia de para qué tipo de sistemas se desarrolle el mismo. Además pueden clasificarse a partir de su escalabilidad y su tolerancia a fallos.

1.5. Tecnologías libres para desarrollar un Middleware

En la actualidad existe gran variedad de tecnologías con las que se pueden desarrollar un Middleware por lo que resulta muy difícil seleccionar una de ellas debido a que cada una fue desarrollada en diferentes etapas y con el objetivo de resolver alguna deficiencia de las que existían hasta ese momento para poder dar solución a nuevos requerimientos que definían los desarrolladores. En lo que sí se puede coincidir es en que cada una intentó crear diferentes mecanismos para resolver los problemas de comunicación entre elementos distribuidos, independientemente de la heterogeneidad de las arquitecturas, protocolos, sistemas operativos y lenguajes de programación, abstrayendo al usuario de la complejidad. Algunos de estos mecanismos pueden ser, la arquitectura que utilizan, como por ejemplo: las arquitecturas cliente/servidor y publicación/subscripción; también están los compiladores utilizados o las diferentes calidades de servicios, incluso muchos otros elementos que es necesario mencionar pero todos tienen un objetivo común, el de garantizar la comunicación.

1.5.1. DCOM

1.5.1.1. Descripción

DCOM surgió a partir de la tecnología *OLE* de Microsoft la cual vio la luz en el año 1990 con el fin de hacer la funcionalidad de “cortar y pegar” en los entornos Windows. Después se extendió a *OLE2* la cual se le cambió el nombre por *COM* y constituyó una infraestructura genérica para la comunicación entre componentes. Al extenderse esta infraestructura con la funcionalidad de aplicarla a varias máquinas comunicadas en red, surgió *DCOM*. **(Booch, 2007)**

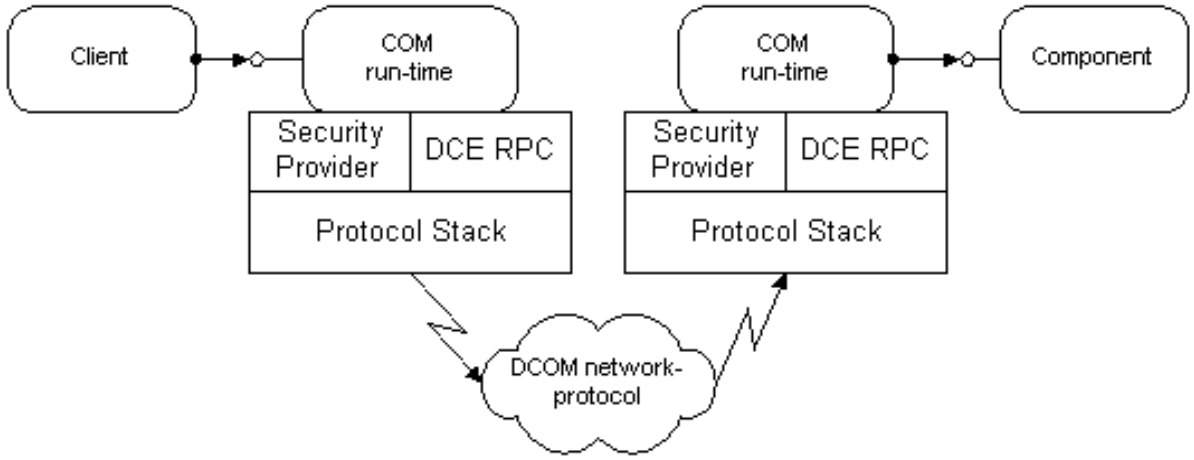

Figura 4: Esquema estructural de DCOM

1.5.1.2. Características

Esta es una tecnología que se ha ido desarrollando y actualmente forma parte esencial de *ActiveX*, *Microsoft Transaction Server*, *COM+ y OPC*.

DCOM no solo brinda todos los componentes básicos necesarios para el diseño y el desarrollo de sistemas sofisticados, también brinda escalabilidad y robustez.

Los *ActiveX* son heredados, al igual que DCOM, de la tecnología *COM* de Microsoft, son muy utilizados para hacer componentes reutilizables entre aplicaciones y gran cantidad de aplicaciones SCADA soportan su uso para dotar a los usuarios de un medio de interactuar con sus objetos de negocio.

1.5.1.3 Ventajas

- Amplia utilización por los usuarios y componentes registrados.
- Muy utilizado, por medio de los *ActiveX*, en los SCADAS existentes para la plataforma Windows.
- Integración de software binario.
- Reutilización del software a gran escala.
- Reutilización del software a través de los lenguajes.
- Actualización de software en línea.

- Permite actualizar un componente en una aplicación sin recompilación.
- Múltiples interfaces por objetos.
- Selección de las herramientas de programación habilitadas, muchas de las cuales brindan automatización del código estándar.

1.5.2. RMI

1.5.2.1. Descripción

RMI es el mecanismo ofrecido en Java que permite a un procedimiento *(método, clase, aplicación)* poder ser invocado remotamente, o sea que este puede existir en cualquier espacio de direcciones de la red incluso en la misma computadora en la que se hace la invocación. *RMI* es básicamente un mecanismo *RPC* orientado a objetos.**(Campo Vázquez, 2004)**

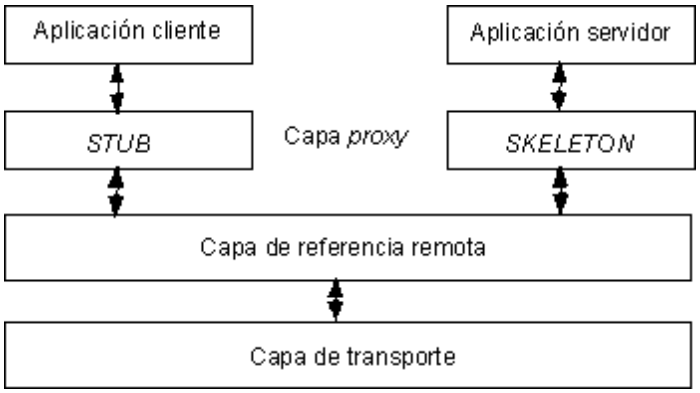

Figura 5: Esquema estructural de Java RMI

1.5.2.2. Características

Java/RMI está basado en un protocolo llamado *JRMP*. Una de las características fundamentales de la arquitectura Java es *Java Object Serialization*, la cual permite que los objetos sean traducidos o transmitidos como un stream (conjunto de datos). Desde que *Java Object Serialization* es específico del entorno, tanto el objeto servidor como el objeto cliente tienen que ser escritos en Java. Cada objeto servidor define una interfaz, la cual puede ser usada para acceder a este objeto desde la misma máquina virtual o desde otra en cualquier lugar de la red. La interfaz brinda un grupo de métodos, los cuales indican los servicios ofrecidos por el objeto servidor. Para que un cliente pueda localizar un objeto servidor, RMI depende de un servicio nombrado RMI-Registry que corre en el servidor y da información acerca de los objetos que se encuentran en la misma. Cuando un cliente adquiere la

referencia de un objeto, invoca sus métodos con total transparencia a su ubicación, su espacio de direcciones puede estar en cualquier lugar de la red.

La referencia de objetos es brindada a los clientes en forma de direcciones URL y ellos acceden a la invocación del objeto servidor de la misma forma que si accedieran a una dirección Web.

1.5.2.3. Ventajas

- Soporta invocaciones remotas de objetos en diferentes máquinas virtuales.
- Soporta llamadas desde los servidores a los applets.
- Integra el modelo de objetos distribuidos dentro del lenguaje Java en un modo natural.
- Preserva la seguridad brindada por el ambiente de trabajo de Java en tiempo de ejecución.
- Hace escrituras confiables de aplicaciones distribuidas tan sencillas como es posible.
- Lenguaje simple.
- Es libre.

1.5.3. CORBA

1.5.3.1. Descripción

CORBA es actualmente una de las opciones tecnológicas más importantes a la hora del desarrollo de sistemas software distribuidos. Es un estándar que establece una plataforma de desarrollo de sistemas distribuidos, facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos. *CORBA* fue definido y está controlado por la *OMG* que define las APIs, el protocolo de comunicaciones y los mecanismos necesarios para permitir la interoperabilidad entre diferentes aplicaciones escritas en diferentes lenguajes y ejecutadas en diferentes plataformas, lo que es fundamental en computación distribuida. En un sentido general la especificación CORBA "envuelve" el código escrito en otro lenguaje en un paquete que contiene información adicional sobre las capacidades del código que contiene, y sobre cómo llamar a sus métodos. Los objetos que resultan pueden entonces ser invocados desde otro programa (u objeto CORBA) desde la red.

En este sentido *CORBA* se puede considerar como un formato de documentación legible por la máquina, similar a un archivo de cabeceras pero con más información. (**Elvira Valenzuela, 2000)**

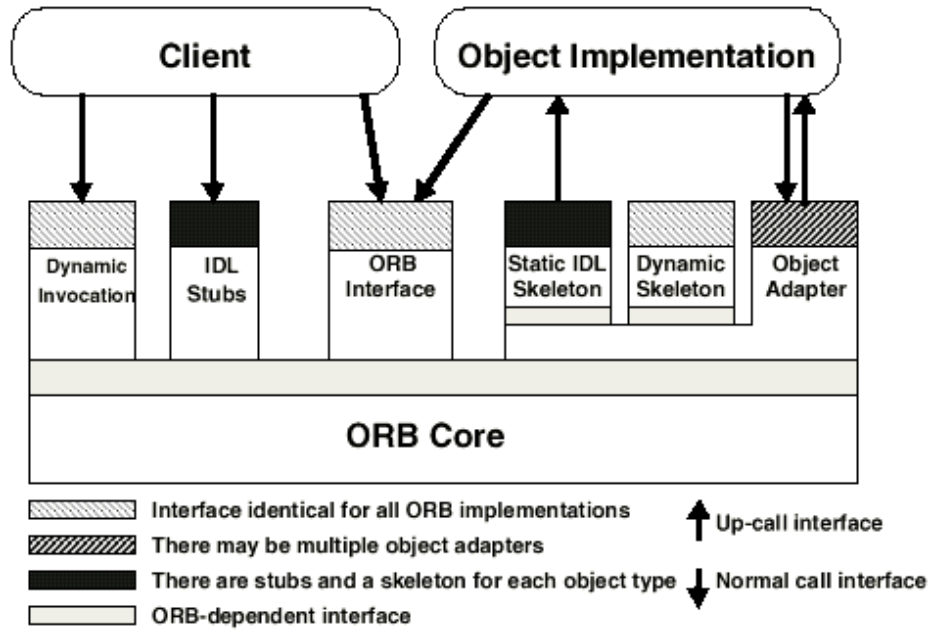

Figura 6: Esquema estructural de CORBA

1.5.3.2. Características

CORBA utiliza un *IDL* para especificar las interfaces con los servicios que los objetos ofrecerán. Puede especificar a partir de este IDL la interfaz a un lenguaje determinado, describiendo cómo los tipos de datos CORBA deben ser utilizados en las implementaciones del cliente y del servidor. Implementaciones estándar existen para Ada, C, C++, Smalltalk, Java y Python. Hay también implementaciones para Perl y TCL. Al compilar una interfaz en IDL se genera código para el cliente y el servidor (el que implementa el objeto). El código del cliente sirve para poder realizar las llamadas a métodos remotos. Es el conocido como stub (cabo), el cual incluye un proxy (representante) del objeto remoto en el lado del cliente. El código generado para el servidor consiste en unos skeletons (esqueletos) que el desarrollador tiene que rellenar para implementar los métodos del objeto. *CORBA* es más que una especificación multiplataforma, también define servicios habitualmente necesarios como es el caso de seguridad.

CORBA es una tecnología basada en la clásica arquitectura encuesta/respuesta. Existe una implementación de objetos en el servidor, al cual el cliente encuesta para ejecutar. Las implementaciones de los objetos cliente y servidor no tienen ninguna restricción de espacio de direcciones, por ejemplo el cliente y el servidor pueden existir en el mismo espacio de direcciones, o pueden ser localizados en diferentes espacios de direcciones en la misma máquina o en espacios de

direcciones separados en diferentes máquinas.

La colección de servicios por niveles del sistema, empaquetados con las interfaces *IDL* son llamados servicios *CORBA*. Ellos son usados para aumentar y complementar la funcionalidad del *ORB*.

El *ORB* es el Middleware que establece las relaciones cliente/servidor entre objetos. Usando un *ORB*, un cliente puede invocar de forma transparente un método en un objeto servidor, que puede estar en la misma máquina o en otra red. El *ORB* intercepta la llamada y es responsable de encontrar un objeto que implemente la petición, pase los parámetros, invoque su método, y devuelva los resultados. El cliente no tiene que saber donde se encuentra el objeto, su lenguaje de programación, su sistema operativo o interfaz. Haciendo esto, el *ORB* proporciona ínter-operatividad entre aplicaciones en diferentes máquinas en entornos distribuidos heterogéneos y conecta múltiples sistemas de objetos.

Los servicios de *CORBA* representan un conjunto de funcionalidades que complementan el desarrollo funcional básico de los objetos que dan lugar a una aplicación. El número de servicios se amplía continuamente para añadir nuevas facilidades a los sistemas desarrollados con *CORBA*. Esto no quiere decir que se encuentren implementados en los *ORB* comerciales. El estándar de *OMG* ha definido dieciocho de estos servicios.

Servicios de CORBA:

- Ciclo de Vida: define operaciones para crear, copiar, mover y eliminar objetos.
- Eventos: permite registrarse para recibir eventos que pueden ser producidos por otros objetos.
- Nombre: permite localizar objetos por un nombre.
- Registro: permite localizar objetos por sus propiedades.
- Persistencia: ofrece una interfaz para almacenar objetos.
- Transacciones: proporciona coordinación transaccional.
- Concurrencia: proporciona un gestor de bloqueos.
- Externalización: permite obtener y producir datos como streams (flujos).
- Seguridad: da soporte a la autenticación, listas de control de acceso y confidencialidad.
- Tiempo: permite definir y gestionar eventos temporizados.
- Propiedades: permite asociar propiedades a un objeto.
- Encuesta: ofrece una interfaz SQL para realizar operaciones en objetos.

- Licencia: ayuda a medir el uso de los objetos.
- Relaciones: permite crear asociaciones dinámicas entre objetos.
- Colección: proporciona las interfaces para crear y manipular colecciones de objetos (listas, conjuntos...)
- Notificación: es una extensión del Servicio de Eventos, añadiendo a la misma: la transmisión de eventos estructurados, subscripción y descubrimiento de eventos, QoS, y un repositorio de tipos de eventos.
- Descubrimiento: soporta el encuentro de objetos CORBA basado en propiedades, describiendo el servicio ofrecido por el objeto.

1.5.3.3. Ventajas

La especificación de *RT-CORBA* define mecanismos independientes de la plataforma para controlar la prioridad de las invocaciones. La especificación *RT-CORBA* define dos tipos de prioridades, las prioridades *CORBA* y las nativas, con el fin de enfrentar la heterogeneidad de los sistemas operativos. Cada operación *CORBA* obtiene una prioridad entre 0 y 32767. Cada sistema final ORB que se encuentre en el camino de la invocación puede asociar las prioridades *CORBA* a las prioridades nativas.

Otras ventajas:

9. Interfaz independiente del lenguaje de programación
10. Integración de herencia
11. Infraestructura de objetos distribuidos
12. Transparencia en la localización
13. Transparencia en la red
14. Capacidades de tiempo real
15. Comunicación directamente con los objetos
16. Interfaces de invocación dinámica
17. Conjunto de funcionalidades representadas por los servicios explicados en el punto anterior.

1.5.4. ORBit

1.5.4.1. Descripción

El surgimiento de *ORBit* está relacionado con el momento en que el proyecto *GNOME* comenzó a hacer uso de *CORBA* con la utilización del *ORB MICO* pero este no se ajustaba muy bien a las necesidades de *GNOME* por lo que Elliot Lee y Dick Porter decidieron escribir un nuevo *ORB* desde cero naciendo de esta forma ORBit.

Hoy en día *ORBit* es la implementación *CORBA* que se utiliza para muchos de los componentes de *GNOME*. Es rápido y ágil permitiendo el uso de *CORBA* en áreas en las que normalmente no parece práctico. Soporta gran parte del estándar 2.2 de *CORBA* y cuenta con características que permiten una fácil integración con los programas de *GNOME*. **(Graham Lewis, 2008)**

1.5.4.2. Características

ORBit cumple con los dos requisitos del proyecto *GNOME*, es libre y es uno de los *ORB* más rápido que existe independientemente de que es muy ligero lo cual lo hace ideal para ser usado en un escritorio como *GNOME*. Consigue una velocidad casi idéntica a una simple llamada a función cuando se usa localmente pues detecta automáticamente las comunicaciones que se están realizando en la misma máquina, desactivando en ese caso gran parte de la complejidad del protocolo de comunicaciones usado en *CORBA* (GIOP/IIOP). Como el resto de las partes básicas de la arquitectura *GNOME*, ORBit está implementado en C y el lenguaje que soporta es precisamente C aunque es bueno destacar que existe soporte para otros lenguajes de programación como C++, Perl y PHP entre otros.

Otro elemento a tener en cuenta a favor de *ORBit*, además de los que se mencionan anteriormente, es que usa menos memoria que otros *ORB*. Aunque es bueno aclarar que no es una implementación completa de *CORBA* ya que solo implementa 14 de sus servicios, aunque esto no sólo es característico de *ORBit* pues ningún estándar los implementa todos; pero si está especialmente indicada para cubrir las necesidades de *GNOME* y de muchas otras aplicaciones que no necesitan un uso ultra-extensivo de *CORBA*, lo cual quiere decir que no necesita todos los servicios de los que esta dispone. ORBit es capaz de comunicarse con otros *ORB* que cumplan el estándar *CORBA* sin ningún problema.

1.5.4.3. Ventajas

- Es rápido y ágil permitiendo el uso de *CORBA* en áreas en las que normalmente no parece práctico.
- Tiene soporte para lenguajes como C++, Perl y PHP entre otros, aunque esta implementado en C.
- *ORBit* es una implementación ligera por sus cortos tiempos de respuesta y que usa menos memoria que otros *ORB*.
- Soporta GIOP/IIOP.
- Es libre.

1.5.5. ACE + TAO

1.5.5.1. Descripción

ACE + *TAO* ha sido desarrollado por el grupo *DOC*, dirigido por Douglas C. Schmidt. Este grupo después de una década de investigación ha desarrollado *ACE*, que es un framework orientado a objetos que implementa varios patrones básicos para software de comunicación concurrente. Además han aplicado los patrones y componentes de *ACE* para desarrollar el *ORB* de *ACE* que es un estándar basado en un framework de Middleware *CORBA* que permite a los clientes invocar operaciones en objetos distribuidos por la ubicación del objeto, lenguaje de programación, sistema operativo, los protocolos de comunicación y las interconexiones, y el hardware. Diseñado utilizando las mejores prácticas de software y los patrones que descubrió el grupo de desarrollo durante su trabajo en *ACE* con el fin de automatizar la entrega con alto rendimiento y en tiempo real de calidad de servicios (QoS) para aplicaciones distribuidas.

Algunas de las motivaciones para el desarrollo de *ACE+TAO* fueron: Combinar las estrategias en arquitecturas de subsistemas en tiempo real y de entrada/salida para proporcionar sistemas *ORB* integrados verticalmente que pueden soportar rendimiento end-to-end, latencia, fiabilidad y QoS. Capturar y documentar los principales patrones de diseño y patrones de optimización necesarios para desarrollar estándares compatibles, portables y QoS extensibles permitidas por los *ORB*. Proporcionar una plataforma Middleware basada en *CORBA* con una alta calidad, disponible gratuitamente y de código abierto para que sea usado por investigadores y desarrolladores. **(Schmidt, 2008)**

ACE y TAO comercialmente cuentan con el apoyo de varias empresas mediante un modelo de negocio “open source”. Esto ha servido para que varios proyectos y patrocinadores estén aplicando

ACE+TAO ayudando a reducir su costo de desarrollo, mejorar sus QoS y disminuir su tiempo de salida al mercado. Además de que sus desarrolladores siguen trabajando en mejorar la calidad, la previsibilidad y el rendimiento del Middleware añadiendo soporte para nuevos servicios y funciones en nuevos estándares OMG.

1.5.5.2. Características

ACE

ACE es un framework disponible libremente, de código abierto y orientado a objetos que implementa patrones básicos para software de comunicación concurrente. *ACE* proporciona un rico conjunto de envoltura de fachadas C++ reutilizables y un marco de componentes que realizan tareas de comunicaciones comunes de software a través de un rango de plataformas de sistemas operativos. Estas tareas de comunicación de software previstas por *ACE* incluyen demultiplexar eventos, manejar y enviar eventos, manejo de señales, inicialización de servicios, comunicación entre procesos, gestión de memoria compartida, enrutamiento de mensajes, reconfiguración dinámica de servicios distribuidos, ejecución concurrente y sincronización.

Algunos de los beneficios de la utilización de *ACE* son:

- *Aumento de la Portabilidad:* Los componentes de *ACE* hacen fácil escribir aplicaciones concurrentes en red en una plataforma de sistema operativo y rápidamente a muchas otras plataformas de sistemas operativos. Además, por ser ACE de código abierto y software libre, nunca habrá que preocuparse sobre la obtención de bloqueos en una determinada plataforma de sistema operativo o la configuración del compilador.
- *Aumento de la Calidad de Software:* Los componentes de *ACE* han sido diseñados utilizando muchos patrones claves que aumentan las cualidades fundamentales como la flexibilidad, extensibilidad, reutilización y la modularidad en los software de comunicación.
- *Aumento de la eficiencia y previsibilidad: ACE* está cuidadosamente diseñado para apoyar una amplia gama de calidad de servicios en aplicaciones, incluyendo la baja latencia para aplicaciones con retraso sensible, alto rendimiento en aplicaciones con uso intensivo del ancho de banda y la previsibilidad en aplicaciones de tiempo real.
- *Fácil transición al estándar de más alto nivel Middleware:* ACE proporciona los componentes reutilizables y patrones usados en el *ORB de ACE* (TAO) que es un estándar de código abierto y compatible con la especificación de CORBA, que es optimizado para el alto rendimiento y

sistemas de tiempo real. De este modo *ACE y TAO* están diseñados para trabajar bien juntos con el fin de proporcionar soluciones de Middleware.

Para proporcionar el desarrollo de software de comunicación con estas características el nivel más alto de aplicaciones Middleware es agrupado con la siguiente liberación de *ACE.*

TAO

Para los desarrolladores de aplicaciones distribuidas e integradas que tienen demandas de prestaciones muy estrictas, *TAO* está disponible libremente, de código abierto y compatible con estándares en tiempo real de implementaciones de *CORBA* que proporcionan eficiencia, previsibilidad, escalabilidad y calidad de servicios "end-to-end". A diferencia de implementaciones convencionales de *CORBA* que son ineficientes, impredecibles, no escalables y a menudo no portables, *TAO* aplica las mejores prácticas y patrones de software para automatizar la entrega de la data en tiempo real.

Generalmente el obstáculo para viabilizar *CORBA* en tiempo real ha sido que muchos problemas en tiempo real han sido asociados con los aspectos del diseño "end-to-end" del sistema, que trascienden las capas límites tradicionalmente asociadas con *CORBA*. Esta es la razón por la que *TAO* integra las interfaces de red, subsistema E/S del sistema operativo, *ORB* y servicios Middleware con el fin de proporcionar una solución "end-to-end". *TAO* aumenta el servicio de eventos estándar de *CORBA* para proporcionar características importantes como el envío y programación de eventos en tiempo real, procesamiento periódico de eventos, filtrado eficiente de eventos y mecanismos de correlación así como protocolos *multicast* requeridos por aplicaciones en tiempo real.

Actualmente *TAO* es un *ORB* implementado en C++ que es compatible con la mayoría de las funciones y servicios definidos en las especificaciones 3.X de *CORBA*, puede descargarse de internet y puede ser usado y redistribuido libremente. Múltiples proveedores tienen disponible soporte comercial, documentación, capacitación y consultorías para *TAO*.

Ha sido portado para plataformas de sistemas operativos incluyendo Windows, muchas versiones de *UNIX*, en sistemas operativos de tiempo real, en definitiva, está previsto *TAO* para todas las plataformas en las que corre *ACE*, además ha llegado a ínter-funcionar con otros *ORB* por lo que se tiene mucha confianza en que su aplicación es robusta e interoperable.

1.5.5.3. Ventajas

TAO brinda muchos de los servicios del estándar de *CORBA* entre los que se encuentran:

- Servicio "streaming" de audio y vídeo.

- Servicio de concurrencia.
- Servicio de eventos.
- Servicio de ciclo de vida.
- Servicio de autenticación.
- Servicio de nombres.
- Servicio de notificación.
- Servicio de persistencia de estado.
- Servicio de propiedad.
- Servicio de seguridad.
- Servicio de tiempo.
- Servicio de intercambio.

Otras ventajas de *ACE+TAO:*

- *TAO* está disponible libremente, de código abierto y compatible con estándares en tiempo real de implementaciones de *CORBA* con altas prestaciones.
- Proporciona eficiencia, previsibilidad, escalabilidad y calidad de servicios.
- *TAO* es un *ORB* implementado en C++.

1.5.6. ICE

1.5.6.1. Descripción

ICE es un middleware, es decir, un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas con soporte para C++, C#, Java, Python, Ruby, PHP, y Visual Basic. *ICE* proporciona herramientas, APIs, y soporte de bibliotecas para construir aplicaciones cliente/servidor orientadas a objetos y entre muchos, un servicio de publicación/suscripción, *IceStorm*, que actúa como un distribuidor de eventos entre clientes y servidores.

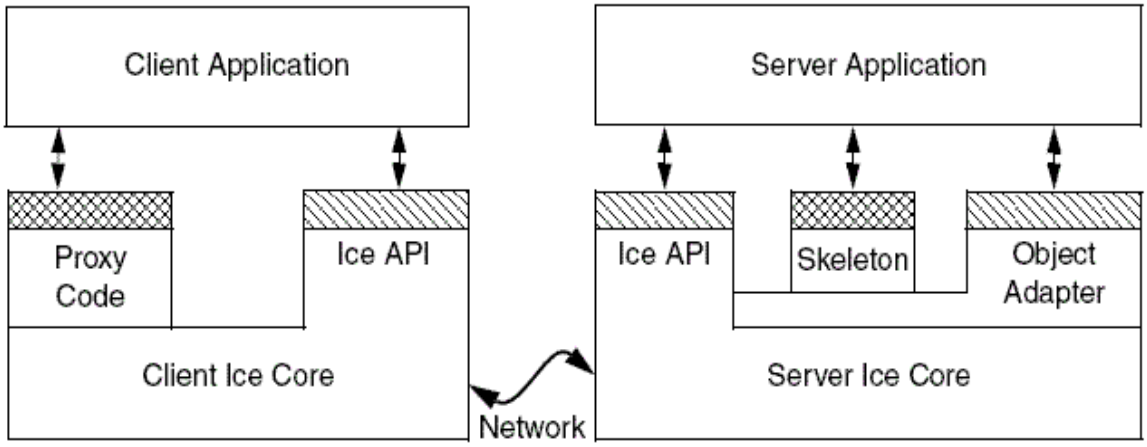

Figura 7: Estructura cliente / servidor de ICE

1.5.6.2. Características

ICE no es más que un Middleware para el programador práctico, una plataforma para desarrollo de aplicaciones de comunicación para Internet de alto rendimiento que incluye varias capas de servicios y plugin, y se compone de los siguientes paquetes:

- Plataforma de objetos distribuidos basados en enfoque cliente/servidor con persistencia.
- *Slice:* Es el lenguaje de especificación de *ICE* que establece un contrato entre clientes y servidores y también se utiliza para describir los datos persistentes.
- *Slice Compilers:* Las especificaciones *SLICE* pueden ser compiladas en diferentes lenguajes de programación. Soporta C++, Java, C#, Visual Basic, Python y Ruby. Los clientes y servidores de *ICE* trabajan juntos independientemente del lenguaje de programación.
- *Ice:* Es la librería núcleo de *ICE*, que entre otras funciones gestiona todas las tareas de comunicación utilizando un protocolo de gran eficiencia (incluyendo el protocolo de compresión y apoyo para *TCP* y *UDP*), proporciona un pool de hilos flexibles para servidores multiproceso y una funcionalidad adicional que apoya la extrema escalabilidad con millones de potencialidades de objetos ICE.
- *IceUtil:* Una colección de funciones de utilidad tales como la manipulación de un estándar de codificación e hilos de programación. (Solamente C++).
- *IceBox*: Un servidor de aplicaciones específicamente para aplicaciones *ICE*. Puede ejecutar y administrar los servicios *ICE* que están cargados dinámicamente como una *DLL*, librería

compartida o una clase Java.

- *IceGrid*: Un sofisticado servidor de activación y despliegue de herramientas avanzadas para computación grid. Además de ser un servicio de localización, *IceGrid* tiene muchas otras características dentro de las que se encuentran la replicación (redundancia de servicios y servidores) y balanceo de carga.
- *Freeze*: Proporciona persistencia automática para servidores *ICE*. Con solo unas pocas líneas de código una aplicación puede incorporar un evictor altamente escalable que gestiona de manera eficaz la persistencia de objetos.
- *FreezeScript*: Es común para cambiar tipos de datos persistentes, sobre todo en grandes proyectos de software. Con el fin de minimizar los impactos de estos cambios proporciona inspecciones y herramientas de migración *Freeze* para las bases de datos. Las herramientas soportan un *XML* basado en su capacidad de *scripting* que es a la vez poderoso y fácil de usar.
- *Ice SSL:* Un *SSL* dinámico que transporta plugin para el núcleo de *ICE*. Ofrece autenticación, cifrado e integridad en los mensajes utilizando el protocolo estándar industrial SSL.
- *Glacier2*: Uno de los retos más difíciles para los sistemas Middleware es la seguridad y los cortafuegos. *Glacier* es la solución de cortafuegos para *ICE*, que simplifica el despliegue de aplicaciones seguras. Glacier autentica y filtra las solicitudes del los clientes y permite llamadas de retorno al cliente en un modo seguro. En combinación con Ice *SSL*, *Glacier* ofrece una potente solución de seguridad que no permite la entrada de intrusos y es fácil de configurar.
- *IceStorm:* Es un servicio de mensajería que soporta federación. En contraste con la mayoría de los tipos de mensajerías o servicios de eventos, IceStorm soporta tipos de eventos, en el sentido de que el *broadcasting* de un mensaje a través de una federación es tan fácil como invocar un método en una interfaz. Se basa en el paradigma de comunicación publicación/suscripción.
- *Ice Patch:* Es un servicio de parches para distribuciones de software. Mantener un software actualizado es a menudo una tarea tediosa. Ice Patch automatiza la actualización de archivos individuales así como jerarquías completas de directorios. Sólo los archivos que han cambiado son descargados a la máquina cliente, utilizando algoritmos de compresión eficiente.

1.5.6.3. Ventajas

- Mantiene una semántica orientada a objetos.

- Proporciona un manejo síncrono y asíncrono de mensajes.
- Soporta múltiples interfaces.
- Es independiente de la máquina.
- Es independiente del lenguaje de programación.
- Es independiente de la implementación, es decir, el cliente no necesita conocer como el servidor implementa sus objetos.
- Es independiente del sistema operativo.
- Soporta el manejo de hilos.
- Es independiente del protocolo de transporte empleado.
- Mantiene transparencia en lo que a localización y servicios se refiere (*IceGrid*).
- Es seguro *(SSL y Glacier2).*
- Soporta comunicaciones cliente/servidor y publicación/suscripción.
- Permite hacer persistentes las aplicaciones (*Freeze*).
- Tiene disponible el código fuente.
- Manejo de un elevado número de variables.
- Ya está siendo usado en sistemas empotrados.

1.5.7. DDS

1.5.7.1. Descripción

DDS es una especificación realizada por *RTI* y aprobada por *OMG*, bajo el paradigma publicación/suscripción. *RTI DDS* es un Middleware orientado a datos que implementa las comunicaciones bajo el modelo antes mencionado con capacidad para tiempo real y permite a los procesos distribuidos compartir datos sin conocer la localización física de los interesados e independientemente de la arquitectura en la cual están soportados. **(Pardo Castellote, 2008)**

Unas pocas soluciones propietarias de *DDS* han estado disponibles durante varios años. Desde el año 2004 dos empresas, la americana *RTI* y la francesa *Thales Group* ambas han participado en las especificaciones aprobadas por la *OMG* en un documento denominado *Data Distribution Service for Real-time Systems.*

Las especificaciones *DDS* describen dos niveles de interfaces:

- Uno menor DCPS *(Data-Centric Publish-Subscribe)*: Este nivel tiene por objeto hacer un reparto de la información de forma eficiente a los recipientes apropiados.
- Uno superior opcional DLRL *(Data Local Reconstruction Layer)*: Este nivel permite una sencilla integración de DDS en la capa de aplicación.

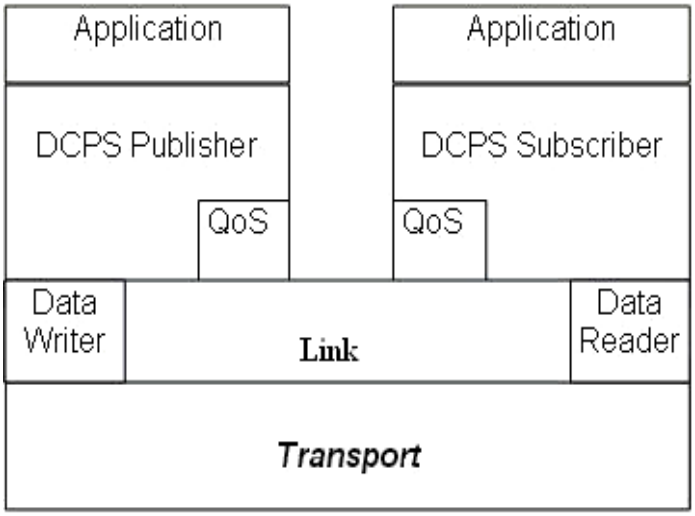

Figura 8: Integración de la capa DCPS en DDS

1.5.7.2. Características

DDS es un Middleware de red que simplifica la compleja programación de red. Implementa un modelo de publicación/suscripción para enviar y recibir datos entre los nodos. Los nodos que producen información (publicadores) crean "*topics*" (tópicos) y publican "*samples*" (muestras) de estos tópicos. *DDS* entrega estas muestras a todos los suscriptores que declaren su interés en el tópico en cuestión.

DDS gestiona todas las fases de la transferencia: direccionamiento de los mensajes, la serialización y deserialización a formato estándar, entrega, control de ancho de banda, reintentos, etc. Cualquier nodo puede ser publicador, suscriptor, o ambas cosas simultáneamente.

El modelo publicación/suscripción de *DDS* elimina virtualmente la compleja programación de red para las aplicaciones distribuidas. Esta tecnología soporta mecanismos que van más allá del modelo básico de publicación/suscripción. El beneficio clave es que las aplicaciones que la usan para sus comunicaciones están enteramente desacopladas. El tiempo de diseño de sus interacciones mutuas es

mínimo. En particular, las aplicaciones no necesitan información sobre el resto de las aplicaciones participantes, incluyendo su existencia o localización. *DDS* gestiona automáticamente todos los aspectos de la entrega de mensajes sin requerir ninguna intervención por parte de la aplicación, incluyendo:

- Determinar quien debe recibir los mensajes.
- Dónde están localizados los destinatarios.
- Qué ocurre si los mensajes no pueden ser entregados.

Esto es posible por el hecho de que *DDS* permite al usuario especificar parámetros de calidad de servicio como una forma de configurar los mecanismos de descubrimiento automático y de especificar el comportamiento deseado en el envío y recepción de mensajes. Estos mecanismos se configuran inicialmente y no requieren un esfuerzo posterior por parte del usuario. Intercambiando mensajes de una forma completamente anónima, simplifica enormemente el diseño de aplicaciones distribuidas y conduce a programas modulares y bien estructurados.

Además gestiona de forma automática el cambio en caliente de publicadores redundantes si el primario falla. Los suscriptores siempre obtienen los datos que son aun válidos con la prioridad más alta (es decir, si el período de validez del dato especificado por su publicador no ha expirado). También automáticamente cambia de nuevo al publicador primario cuando este se ha recuperado.

DDS está disponible con APIs para C, C++ y Java, y en multitud de plataformas (Linux, Solaris, Windows, Integrity, LynxOs, VxWorks…).

A continuación se muestran las principales componentes de *DDS*:

- *Topic:* se asocia a un nombre único en el dominio, a un tipo de dato, y una QoS relacionados con los datos en sí. Este nombre de tópico se utiliza de los lados del *publisher* y del *subscriber* para enviar y recibir los datos.
- *Publisher:* El publicador es el objeto responsable de la distribución de los datos. Puede publicar diferentes tipos de datos.
- *DataWriter:* Es el objeto que la aplicación debe utilizar para comunicar a un publicador la existencia y el valor de los objetos de un tipo determinado.
- *Subscriber:* El suscriptor es el objeto responsable de recibir los datos publicados y ponerlos a disposición (de acuerdo con las QoS del suscriptor) a la aplicación receptora. Es posible recibir y despachar datos de los diferentes tipos especificados.

- *DataReader:* Permite a la aplicación acceder a los datos recibidos, la aplicación debe utilizar un DataReader adscrito al suscriptor. Así, la suscripción se define por la asociación de un Data Reader con un suscriptor.

1.5.7.3. Ventajas

- Disminución entre el acoplamiento de entidades debido en parte al empleo de la filosofía publicación/suscripción.
- Arquitectura flexible y adaptable gracias al empleo del descubrimiento automático (*RPTS*).
- Eficiencia debido a la comunicación directa entre el publisher y el subscriber.
- Determinismo en la consigna de los datos.
- Escalabilidad debido en parte a la disminución del acoplamiento entre entidades.
- Calidad de servicio altamente parametrizable.
- Independencia de la plataforma por el uso de estándares como IDL.

1.5.8. OpenDDS

1.5.8.1. Descripción

OpenDDS es una implementación C++ de código abierto de la especificación de *DDS* para Sistemas de Tiempo Real versión 1.0 de la *OMG*. Implementa la mayor parte de los perfiles mínimos de la capa DCPS de la especificación *DDS*.

OpenDDS se basa en la capa de abstracción de *ACE* para facilitar la portabilidad de la plataforma. Además aprovecha las capacidades de *TAO*, como su compilador de *IDL* y la base del Repositorio de Información del *DCPS* (DCPSInfoRepo). Adicionalmente aprovecha *MPC* para aliviar la carga de mantenimiento apoyando la construcción de múltiples entornos y plataformas. *OpenDDS* cuenta con el apoyo de la *OCI* que es una compañía de ingeniería de software basada en los principios de la Tecnología Orientada a Objetos (OO) para producir software de calidad.**(Object Computing Inc, 2007)**

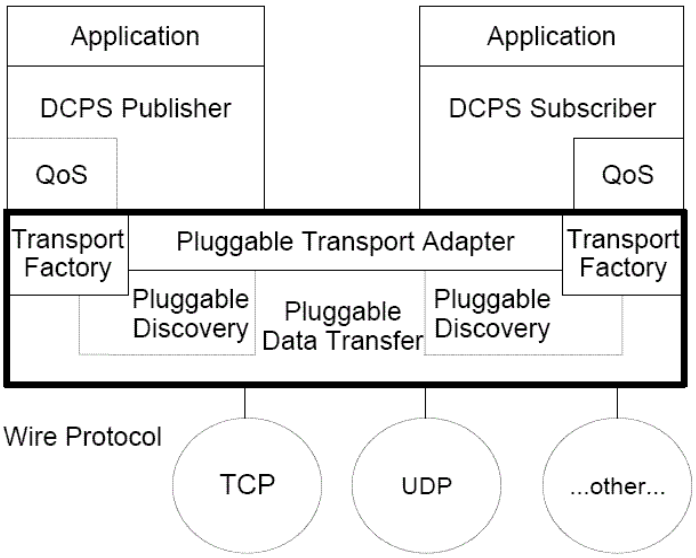

Figura9: Framework de Transporte en OpenDDS

1.5.8.2. Características

OpenDDS está basado en la especificación 1.0 de *DDS* y ofrece los siguientes protocolos de transporte por defecto: *TCP*, *Reliable Multicast, Unreliable Multicast y UDP.*

El framework de transporte permite a cualquiera crear un transporte para adaptarse a los requerimientos personalizados.

OpenDDS se ha encontrado para obtener mejores resultados que otros servicios de *TAO* (notificación y canal de eventos en tiempo real) por un factor de dos o tres. Las características ofrecidas por el *RTEC* y el *NS* son similares a *DDS* pero no idénticas, por lo que se debe revisar cuidadosamente los casos de uso antes de elegir un servicio u otro. Además se debe tener en cuenta que la velocidad no es el único criterio a tener en analizar.

OpenDDS se relaciona con *CORBA* siendo un híbrido de la comunicación *CORBA*/*GIOP* para tareas administrativas y transporte personalizado de tipo seguro para la transferencia de datos.

Como cuerpo de código es vagamente unido con *TAO*, esto permite a *DDS* evolucionar con rapidez y sin embargo permite a *TAO* evolucionar a su propio ritmo. Cualquier dependencia, entre las versiones de *TAO* y *DDS* deben ser administradas cuidadosamente con miras a reducir al mínimo sus efectos.

En las zonas en las que no hay utilización de ningún núcleo *ORB*, *OpenDDS* utiliza *ACE* como capa de abstracción o portabilidad.

Si se compara *OpenDDS* con el canal de eventos en tiempo real de *TAO*, que tiene un servicio de publicación/subscripción, parece indicar una mejora tres veces mayor en el rendimiento, en una mensajería de punto a punto.

1.5.8.3. Ventajas

- *OpenDDS* está basado en la especificación 1.0 de *DDS*.
- Se basa en la capa de abstracción de *ACE* para facilitar la portabilidad de la plataforma.
- Ofrece los siguientes protocolos de transporte por defecto:

1. *TCP*,
2. *Reliable Multicast,*
3. *Unreliable Multicast*
4. *UDP*.

- El framework de transporte permite a cualquiera crear un transporte para adaptarse a los requerimientos personalizados.
- El uso de transporte personalizado permite a obtener las ventajas de eficiencia y baja latencia en protocolos de transmisión de datos, manteniendo los beneficios de interfaces estándares de alto nivel.
- *OpenDDS* se ha encontrado para obtener mejores resultados que otros servicios de *TAO* (notificación (*NS*) y canal de eventos en tiempo real (*RTEC*)) por un factor de dos o tres.
- Transferencia de datos alta y a gran velocidad.
- *OpenDDS* proporciona un framework de transporte que facilita añadir un nuevo transporte.

CAPÍTULO 2: SELECCIÓN DE TECNOLOGÍAS Y MODELADO DE LA SOLUCIÓN

2.1 Introducción

El presente capítulo muestra la descripción de las funcionalidades del sistema, la selección de las tecnologías para mejorar la versión de 1.0 del Middleware del Guardián del ALBA y el modelado de un prototipo funcional basado en la tecnología seleccionada. También se describen las herramientas y metodologías a usar para modelar la solución y su implementación a tratar en el próximo capítulo.

2.2. Descripción de las funcionalidades del Sistema

A continuación se muestran los requerimientos del subsistema para mejorar el Middleware en su versión 1.0.

2.2.1. Requisitos Funcionales

Los requisitos funcionales (RF) se pueden definir como capacidades o condiciones que el sistema debe cumplir, y que se mantienen invariables sin importar con qué propiedades o cualidades se relacionen. *A continuación la lista de requisitos del sistema Middleware como un todo a pesar de que la solución solo va modelar e implementar algunas de estos a partir de un prototipo funcional:*

RF1. El sistema debe permitir el intercambio de variables (puntos) de forma desacoplada.

RF1.1. Brindar un mecanismo para solicitar el envío de variables.

RF1.2. Brindar un mecanismo para solicitar recibir variables.

RF1.3. Enviar variables de forma desacoplada.

RF1.4. Recibir variables de forma desacoplada.

RF2. El sistema debe permitir el intercambio de alarmas de forma desacoplada.

RF2.1. Brindar un mecanismo para solicitar el envío de alarmas.

RF2.2. Brindar un mecanismo para solicitar recibir alarmas.

RF2.3. Enviar alarmas de forma desacoplada.

RF2.4. Recibir alarmas de forma desacoplada.

RF3. El sistema debe permitir el intercambio de eventos de forma desacoplada.

RF3.1. Brindar un mecanismo para solicitar el envío de eventos.

RF3.2. Brindar un mecanismo para solicitar recibir eventos.

RF3.3. Enviar eventos de forma desacoplada.

RF3.4. Recibir eventos de forma desacoplada.

RF4. El sistema debe permitir el intercambio de log de usuarios de forma desacoplada.

RF4.1. Brindar un mecanismo para solicitar enviar log de usuarios.

RF4.2. Brindar un mecanismo para solicitar recibir log de usuarios.

RF4.3. Enviar log de usuarios de forma desacoplada.

RF4.4. Recibir log de usuarios de forma desacoplada.

RF5. El sistema debe permitir el intercambio de bitácoras de forma desacoplada.

RF5.1. Brindar un mecanismo para solicitar enviar bitácoras.

RF5.2. Brindar un mecanismo para solicitar recibir bitácoras.

RF5.3. Enviar bitácoras de forma desacoplada.

RF5.4. Recibir bitácoras de forma desacoplada.

RF6. El sistema debe permitir el intercambio de comandos de forma desacoplada.

RF6.1. Brindar un mecanismo para solicitar enviar comandos.

RF6.2. Brindar un mecanismo para solicitar recibir comandos.

RF6.3. Enviar comandos de forma desacoplada.

RF6.4. Recibir comandos de forma desacoplada.

RF7. El sistema debe permitir el intercambio de respuesta de comando de forma desacoplada.

RF7.1. Brindar un mecanismo para solicitar enviar respuesta de comando.

RF7.2. Brindar un mecanismo para solicitar recibir respuesta de comando.

RF7.3. Enviar respuesta de comando de forma desacoplada.

RF7.4. Recibir respuesta de comando de forma desacoplada.

RF8. El sistema debe permitir invocar métodos remotos implementados por el módulo de Seguridad.

RF8.1. Brindar un mecanismo para obtener un cliente de Middleware para Seguridad.

RF8.2. Invocar métodos remotos implementados por Seguridad.

RF9. El sistema debe permitir resolver llamadas remotas realizadas al módulo de Seguridad.

RF9.1. Proporcionar un mecanismo para obtener un servidor de Middleware para Seguridad.

RF9.2. Registrar implementación de los métodos que soporta Seguridad.

RF9.3. Ejecutar métodos de Seguridad y entregar respuesta a los clientes de Seguridad.

RF10. El sistema debe permitir invocar métodos remotos implementados por el módulo de Configuración.

RF10.1. Brindar un mecanismo para obtener un cliente de Middleware para Configuración.

RF10.2. Solicitar configuración al módulo de Configuración.

RF11. El sistema debe permitir resolver llamadas remotas realizadas al módulo de Configuración.

RF11.1. Proporcionar un mecanismo para obtener un servidor de Middleware para Configuración.

RF11.2. Registrar implementación de los métodos que soporta Configuración.

RF11.3. Ejecutar métodos de Configuración y entregar respuesta a los clientes de este.

RF12. El sistema debe permitir enviar consultas SQL al módulo de Históricos.

RF12.1. Brindar un mecanismo para obtener un cliente de Middleware para BDH.

RF12.2. Enviar consultas al módulo de Históricos.

RF13. El sistema debe permitir enviar respuesta de consultas a los clientes de BDH.

RF13.1. Brindar un mecanismo para obtener un servidor de Middleware para BDH.

RF13.2. Enviar respuesta de consultas a los clientes de Históricos.

2.2.2. Requisitos No Funcionales

Las características o requisitos no funcionales (RNF) del sistema, son propiedades o cualidades que el producto debe tener. Son las características que hacen al producto atractivo, usable, rápido y confiable. Estos requisitos son muy importantes para que los clientes y usuarios puedan valorar características no funcionales del software como: seguridad, confiabilidad y usabilidad.

RNF1. Software

RNF1.1. Sistema operativo GNU/Linux, distribución Debian Lenny (5), Kernel 2.6.26.

RNF2. Usabilidad

RNF2.1. El sistema debe brindar interfaces de comunicación fáciles de integrar y de usar.

RNF3. Fiabilidad

RNF3.1. Debe existir una comunicación segura y eficiente, garantizando no existan perdidas de información ni acceso indebido de usuarios o aplicaciones.

RNF3.2. Deben proveerse mecanismos de tolerancia a fallas que permitan recuperarse antes e errores de comunicación.

RNF3.3 La invocación de métodos implementados por Seguridad y la transferencia de comandos debe ser dependiente de los mecanismos de comunicación segura implementados.

RNF4. Rendimiento

RNF4.1. El intercambio de puntos, alarmas, comandos, eventos, log de usuarios, bitácora y estados de la comunicación debe ser en tiempo real.
RNF4.2. El sistema debe garantizar la transferencia de un elevado número de variables y tiempos bastante bajos, 50000 variables en 1 segundo.

RNF5. Soporte

RNF5.1. El desarrollo del sistema orientado a la exposición de interfaces de comunicación debe brindar alta interoperabilidad para que aplicaciones del SCADA Guardián del ALBA implementadas en diferentes lenguajes y sistemas operativos, puedan comunicarse de manera rápida y eficiente.
RNF5.2. Se debe implementar el sistema sobre la base de una arquitectura que introduzca atributos de flexibilidad, escalabilidad y adaptación y que permita agregar y/o modificar tipos de datos y funcionalidades con bastante facilidad y de forma eficiente sin impactar en los clientes y sin introducir grandes cambios en la solución.

RNF6. Portabilidad

RNF6.1. La solución adoptada debe poder ser implantada en cualquier sistema distribuido que necesite intercambiar información entres los subsistemas o módulos que lo componen.

2.3. Selección de las Tecnologías

Si se toma como premisa los estándares y tecnologías actuales utilizados para implementar soluciones de este tipo y que fueron abordados en el capítulo 1 y como elemento principal, los requisitos especificados al inicio de este capítulo, básicamente relacionados con brindar interfaces de información en tiempo real entre los subsistemas de Guardián del ALBA, entonces se puede proceder a los diferentes análisis.

2.3.1. Selección de los criterios de evaluación

A continuación se exponen uno idéntico, disponible con las mismas funciones y que mantenga las comunicaciones de forma transparente para a los clientes.
Soporte de mecanismos de seguridad: en todo sistema de este tipo es importante la seguridad, y la comunicación no está fuera de ello, es por eso que se debe evaluar los mecanismos de seguridad que provee cada tecnología a fin de seleccionar los más adecuados a serie de criterios que deben ser valorados por cada tecnología a fin de seleccionar varias opciones, compararlas y decidir por la que mejores resultados sea capaz de arrojar.

- *Facilidad de implementación:* este criterio pretende evaluar cuanto de difícil sería implementar la

solución con las tecnologías analizadas. Evaluar este criterio es de suma importancia porque permite realizar una mejor estimación del tiempo de desarrollo de la solución y aporta seguridad al cliente teniendo en cuenta que el equipo de desarrollo está preparado en la tecnología analizada y puede cumplir con las entregas en el tiempo estimado.

- *Documentación disponible y alcance del personal:* define la posibilidad que tiene el equipo de avanzar de forma rápida y segura en el desarrollo de la solución usando una tecnología que tiene documentación disponible. Contar con la documentación disponible aporta seguridad al equipo de trabajo y al cliente. La combinación con el criterio anterior puede traer muy buenos resultados.
- *Desarrollo de aplicaciones distribuidas:* este criterio pretende evaluar la capacidad de desarrollar aplicaciones distribuidas que aportan las tecnologías analizadas siendo este criterio el punto clave a analizar en aplicaciones de este tipo.
- *Capacidad de tiempo real:* es de mucha importancia medir la capacidad de transacciones en tiempo real en cada tecnología a fin de garantizar que la información viaje de forma rápida y se entregue en tiempo real cumpliendo con los tiempos establecidos en las especificaciones de requisitos del SCADA.
- *Consumo de recursos:* este criterio tiene un valor importante y determinante en la selección permitiendo determinar aproximadamente cómo se comporta el consumo de CPU y Memoria usando cada tecnología.
- *Soporte por parte de los desarrolladores:* analizar este criterio permite tener claridad y seguridad en cuanto al soporte que tiene cada tecnología y saber si es seguro usar una u otra dependiendo de quién da el soporte y de qué forma se realiza el mismo.
- *Transmisión de un elevado número de variables:* este punto es de elevada importancia en la solución de Middleware del SCADA teniendo en cuenta que los requisitos exigen que la data debe viajar en cantidades de 50000 variables por segundo lo que obliga a contar con una tecnología que soporte mecanismos de comunicación sumamente eficientes que permitan cumplir con estos tiempos.
- *Persistencia a las conexiones:* una de las grandes diferencias entre los sistemas centralizados y los distribuidos es que en estos últimos tiene un centro de comunicación que intercomunica cada módulo logrando que si se cae un módulo no afecte al resto porque no tienen dependencias directas entre ellos sino a través de un centro o Middleware. Una de los mecanismos que garantiza la tolerancia a fallos relacionados con las comunicaciones es la persistencia a las conexiones. Si se cae el servicio de comunicación cada módulo continúa

haciendo sus operaciones, algunos van a dejar de recibir datos pero en el momento de restablecerse los servicios, los módulos continuarán comunicándose sin haber tenido que reiniciarse y de forma transparente. Por todo lo anteriormente planteado es necesario evaluar este criterio en cada una de las tecnologías.

- *Soporte de mecanismos de redundancia:* es importante analizar este criterio en la evaluación porque otra de las formas de tener tolerancia a fallas durante las comunicaciones es contando con Servicios o Middleware redundantes garantizado que cuando deje funcionar uno de estos, exista otro idéntico, disponible con las mismas funciones y que mantenga las comunicaciones de forma transparente para a los clientes.
- *Soporte de mecanismos de seguridad:* en todo sistema de este tipo es importante la seguridad, y la comunicación no está fuera de ello, es por eso que se debe evaluar los mecanismos de seguridad que provee cada tecnología a fin de seleccionar los más adecuados.

2.3.2. Importancia y calificación de los criterios de evaluación

A cada uno de los criterios se le debe asignar un valor de importancia en el rango de 1 a 100 donde la suma de los valores de importancia asignados no debe pasar de 100. En el Anexo I se pueden encontrar los valores de importancia asignada a cada criterio de evaluación.

También es necesario asignar a cada criterio una calificación que vas desde 0 hasta 4 o lo que es lo mismos desde “No Aceptable” hasta “Excelente”. Las calificaciones por criterios se pueden encontrar en el Anexo I.

La multiplicación de la calificación dada a cada tecnología en un criterio determinado por la importancia asignada al mismo, arroja el puntaje de una tecnología en ese criterio, al final se suman todos los puntajes permitiendo seleccionar las mejor tecnología de las opciones tecnológicas analizadas.

2.3.3. Opciones Tecnológicas

Después de analizar la información relacionada con las diferentes tecnologías descritas en este documento y los criterios de evaluación, llegamos al punto de definir que tecnologías cumplen con estos criterios y determinar que opciones se van a analizar en la matriz de decisión.

Para comenzar es necesario tener presente las características que debe cumplir la tecnología con el fin de lograr una solución adecuada para este tipo de aplicaciones y que permita eliminar o disminuir

considerablemente los problemas existentes en la implementación actual. En el proceso de estudio de varias tecnologías se pudo apreciar que a pesar de las grandes facilidades que brinda ***CORBA*** como la interfaz independiente del lenguaje de programación, la infraestructura de objetos distribuidos así como la transparencia en la localización y en la red y soporte para casi todos los criterios, esta presenta dos grandes desventajas: su complejidad y las deficiencias de rendimiento. Uno de los aspectos que amplía su complejidad es que hay que estar consciente a la hora de diseñar que objetos van a ser remotos y cuáles no, los primeros sufren restricciones en cuanto a sus capacidades que los objetos normales no tienen. Tiene otra desventaja que no solo afecta a *CORBA*, sino a todas las implementaciones de esta como: *TAO*, *MICO*, *ORBit* y *omniORB*. La desventaja radica en la incompatibilidad que existe entre las diferentes implementaciones. Aunque las diferencias entre los diversos *ORB* radica en detalles que hacen imposible aplicar en uno el mismo diseño de un programa pensado para otro, hacen que la adaptación sea difícil y propensa a fallas. También es importante saber que *CORBA* es una especificación que tiene muchas implementaciones.

Continuando con el análisis del resto de las tecnologías puede verse que ***ORBit*** posee prestaciones que son convenientes para su utilización en un sistema como el que se necesita debido a que es una implementación de *CORBA*. Se destaca por ser ligera, por sus cortos tiempos de respuesta y baja utilización de memoria. Debe aclararse que este *ORB* mantiene algunas de las desventajas de *CORBA* como es la incompatibilidad con otros *ORB*. Se puede decir que ninguna de las implementaciones de *CORBA* implementa todos los servicios de los que dispone la misma, ya que sólo utilizan los que crean los desarrolladores que son más importantes utilizar en el área para la cual está destinado fundamentalmente el *ORB*. Un ejemplo lo podemos ver en *ORBit*, que está destinado para que se utilice en el proyecto *GNOME* y sólo implementa 14 servicios. Además otra desventaja que presentan las implementaciones de *CORBA*, principalmente el caso de *ORBit* es que tienen una frecuencia de transmisión de datos baja. Todo esto determina que no sea muy conveniente la utilización de *ORBIT* en este sistema por lo que se deberá continuar analizando el resto de las tecnologías.

En el caso de ***DDS*** debe señalarse que su utilización de un modelo publicación/subscripción y su DCPS le permiten ser flexible y adaptable. También posee una baja sobrecarga lo que le permite ser usado en sistemas de alto funcionamiento, es escalable dinámicamente y tiene una entrega de datos determinística entre otras características interesantes. Sin embargo DDS tiene una gran desventaja y es que no posee un mecanismo de seguridad muy eficiente algo que realmente es muy importante para cualquier aplicación, en especial para una destinada a mantener la comunicación. Se puede agregar también que está en constante desarrollo, las implementaciones existentes aún carecen de varias características esenciales para Middleware en sistemas distribuidos y en este caso en particular no soporta muchos de las criterios planteados.

Otra de las tecnologías que ha alcanzado un cierto reconocimiento en los últimos tiempos es ***OpenDDS*.** Integra varios componentes de otras tecnologías para lograr su funcionamiento lo que la convierte en una nueva herramienta muy potente. Ella además de que es una implementación en C++ de *DDS* y de código abierto, implementa varios perfiles del *DCPS* basándose en la capa de abstracción de ***ACE*** para facilitar la portabilidad. También integra algunas capacidades de *TAO* como su compilador *IDL* entre otros elementos. Todo lo anteriormente mencionado le permite obtener mayores resultados que *TAO* en la notificación y el canal de eventos en tiempo real. No obstante, no soporta plenamente todas las políticas de QoS que tiene *DDS* por lo que todavía existen algunos elementos que son incompatibles. Todo esto, desde el punto de vista de esta investigación, la convierte todavía en una tecnología dependiente de las especificaciones de las demás en las que se basa por lo que es necesario administrar cuidadosamente las dependencias entre las versiones de *TAO* y *DDS* con vista a reducir al mínimo sus efectos. Por tanto esta tecnología, aunque presenta mucho potencial para un futuro, no ofrece la estabilidad que necesita el sistema de comunicación del Guardián del ALBA.

En el caso de la tecnología ***TAO*** es una implementación del estándar CORBA que posee muchas prestaciones y son muy convenientes para su utilización en un sistema como el que se necesita. TAO está concebido para el desarrollo de sistemas en tiempo real debido a la eficiente implementación que realiza del estándar *RT CORBA*, además soporta la mayoría de las características definidas. No obstante, debe aclararse que este *ORB* mantiene algunas de las desventajas de *CORBA* como es la incompatibilidad con otros *ORB*. Se puede decir que ninguna de las implementaciones de *CORBA* implementa todos los servicios de los que dispone la misma, ya que sólo utilizan los que crean los desarrolladores que son más importantes utilizar en el área para la cual está destinado fundamentalmente el *ORB*. *TAO* sin embargo es la implementación de *CORBA* que más servicios implementa y esto constituye otra de sus desventajas pues lo hacen un poco lento en algunas interacciones. Podemos agregar que el equipo de desarrollo de la tecnología está trabajando en lograr mayor solidez con la tolerancia a fallos, agregarle nuevas funcionalidades que le permitan el trabajo en nuevas plataformas y lograr mayor integración con otros *ORB*.

Además de las ventajas y desventajas que posee esta tecnología Middleware entre ellas que es una de las pocas de las estudiadas que cumple con todos o casi todos los criterios de evaluación definidos, es necesario agregar todos los problemas afrontados en la implementación del Middleware del SCADA Guardián del ALBA relacionadas problemas para implementar la persistencia de las conexiones, realizar intercambio de información por diferentes interfaces de red, no existencia de mecanismos de Redundancia y lentitud en las trasmisiones de un elevado número de variables cuando aumenta a medida que crece el número de clientes.

Por último nos queda por analizar la tecnología ***ICE*** que ha tomado mucha fuerza en los últimos años. ICE fue desarrollada por varios de los desarrolladores de CORBA con el objetivo de resolver muchos de los problemas de esta última. Se dice que fue tal el empeño puesto en su desarrollo que ningún otro ORB implementa todas las funciones que ICE contempla. Esto le permite grandes ventajas como el manejo de mensajes síncronos y asíncronos, soporta múltiples interfaces, mecanismo de comunicación cliente-servidor y publicación-subscripción, es independiente de la máquina, del lenguaje de programación, de la implementación y del sistema operativo; soporta el manejo de hilos, es independiente del protocolo de transporte, mantiene la transparencia en la localización y en los servicios, es seguro, hace persistentes las aplicaciones y redundante los servicios ICE y tiene disponible el código fuente. Además tiene dos ventajas sobre TAO, una de las tecnologías antes analizadas y con mayor por ciento de aceptación a su favor, es el rendimiento en cuanto a la velocidad de transmisión de los eventos y que ICE permite la comunicación con otros ORB. Si alguna desventaja se pudiera mencionar de esta tecnología es que no responde a ningún estándar sino que su especificación es de los creadores del proyecto ZeroC.

2.3.4. Justificación de la Evaluación de Criterio – Alternativas

Después de la valoración anterior se puede determinar que hay dos tecnologías candidatas que cumplen con los requisitos o con la mayoría de los criterios de evaluación, *TAO* y *ICE*, es por eso que se desarrolla la matriz de evaluación basada en estas dos tecnologías y se realizan un grupo de pruebas de concepto y rendimiento a ambas tecnologías a fin de tener más elementos para una decisión final.

Para observar los detalles de la evaluación de cada uno de los criterios en las opciones tecnológicas y determinar el porqué de los valores otorgados en caso, se debe remitir a los **Anexo I**. A continuación de hace realiza un análisis crítico por cada alternativa evaluada en la matriz.

2.4. Tecnología(s) Seleccionada(s)

Después de analizar las descripciones de cada tecnología hechas a lo largo del documento, hacer una valoración general sobre cada una de ellas y analizar los criterios de evaluación se pudo determinar que opciones tecnológicas son las mejores candidatas, lo que arrojó que solo las tecnologías TAO y ICE cumplen con todos o la mayoría de estos criterios de evaluación. Para seleccionar la mejor tecnología se realizó un análisis crítico de los criterios de evaluación en una matriz de decisión que de acuerdo al valor de importancia asignado a cada criterio por la calificación otorgada a cada tecnología

en cada uno de ellos, arrojó un puntaje final que permitió determinar la mejor tecnología para implementar el Middleware 2.0.

Para obtener los resultados anteriores se realizaron un grupo de pruebas de concepto y rendimiento a ambas tecnologías a fin de tener más elementos para una decisión final.

Las pruebas de concepto de TAO están dentro de la implementación del Middleware actual y las pruebas de concepto realizadas a ICE demostraron que los servicios probados cumplen con las características que describen sus desarrolladores. Esta potente tecnología soporta todas las potencialidades que tiene TAO entre otras más novedosas, pero más aún, se pudo constatar que ICE no tiene algunos de los problemas que presenta TAO como son la complejidad, problemas con las interfaces de red y lentitud ante el envío de gran cantidad de eventos en tiempo real. También tiene soporte para persistencia de los objetos, mecanismos de seguridad más flexibles y robustos que los de TAO y redundancia de servicios y aplicaciones.

Las pruebas de rendimiento demostraron la superioridad de ICE sobre TAO en cuanto al intercambio de eventos (variables o puntos) en tiempo real donde ICE llega a ser más rápido que TAO, 8 y 10 veces de forma local y remota respectivamente.

Si además de los elementos anteriores, se toma con premisa los problemas detectados en TAO durante el desarrollo de la versión actual de Middleware, se tiene un elemento más que ayuda a decidir por ICE como la tecnología base para mejorar la versión 1.0 de la capa de comunicaciones del Guardián del ALBA.
Para más información o la información más importante sobre los resultados de las pruebas que hacen de ICE la tecnología ideal para mejorar la versión 1.0 del Middleware se puede revisar el **Anexo II**:

2.5. Metodologías, herramientas y lenguajes de programación a utilizar

Las metodologías y lenguaje de programación a utilizar que se exponen a continuación no son objeto de selección en este trabajo, las mismas fueron analizadas y evaluadas desde los inicios del proyecto SCADA es por ello que solo se describen sus principales características.

2.5.1. Metodología RUP

La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, es la metodología de desarrollo de software estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Es un producto del proceso de ingeniería de software que proporciona un enfoque disciplinado para asignar tareas y responsabilidades dentro de una organización del desarrollo. Su meta es asegurar la producción del software de alta calidad que resuelve las necesidades de los usuarios dentro de un presupuesto y tiempo establecidos.

El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización. Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Posee un ciclo de vida el cual está formado por 4 fases secuenciales, donde en cada una se realiza una evaluación para determinar si los objetivos de la fase se han cumplido, y de ser satisfactoria, pasar a la próxima fase. **(Booch, y otros, 1999)**

2.4.2. Lenguaje de Modelado Unificado (UML)

El Lenguaje Unificado de Modelado UML, es una notación estándar para el modelado de sistemas software, es decir, un lenguaje de propósito general para el modelado orientado a objetos, que combina notaciones provenientes desde: Modelado Orientado a Objetos, Modelado de Datos, Modelado de Componentes, Modelado de Flujos de Trabajo. UML surge como respuesta al problema de contar con un lenguaje estándar para escribir planos de software. Permite documentar y especificar los elementos creados mediante un lenguaje común describiendo modelos. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Es importante resaltar que UML es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. **(Booch, y otros, 2007)**

2.4.3. RSA como herramienta CASE

IBM Rational Software Architect, RSA es una herramienta de desarrollo y diseño integrada que

fortalece el desarrollo dirigido por modelos UML para la creación de servicios y aplicaciones con arquitecturas sólidas.

RSA ofrece un ambiente poderoso de diseño y construcción que ayuda los arquitectos de software a entender, diseñar y manejar las soluciones empresariales y los servicios a través del equipo, a través del mundo y a través de diferentes áreas de la experticia técnica. Ofrece significativas ventajas como: modelado de aplicaciones más productivo que nunca con pues incluye nuevas características fáciles de usar y la productividad en modelos de flujos de trabajo; soporte de gran variedad de diagramas; utilización de lo más nuevo en estándares de lenguajes de modelado, permitiendo al usuario definir múltiples niveles de modelos acoplados con transformaciones de usuario entre modelos y código, resultando en una clara separación de asuntos a través del ciclo de vida; ofrece flexibilidad en el manejo de modelos UML y capacidad para fusión de modelos que se pueden adaptar a las necesidades; y por último automatizaciones que generan poder basadas en modelos, es decir, el modelado agrega valor y ayuda a reducir riesgos de proyecto, pero mayores beneficios son obtenidos cuando los modelos son utilizados para automatizar la creación de otros artefactos de desarrollo incluyendo otros modelos, código y otros. **(IBM, 2009)**

2.4.4 Lenguaje de Programación C++

C++ es un lenguaje para la programación que se utiliza ampliamente en la industria del software. Es un lenguaje imperativo orientado a objetos, la definición oficial dice que C++ es un lenguaje de propósito general basado en C, al que se han añadido nuevos tipos de datos, clases, plantillas, mecanismo de excepciones, sistema de espacios de nombres, funciones online, sobrecarga de operadores, referencias, operadores para manejo de memoria persistente, y algunas utilidades adicionales de librería que lo convierten en un candidato especial para el desarrollo del middleware. En el diseño de la librería estándar C++ se ha utilizado la dualidad de ser mezcla de un lenguaje tradicional con POO lo que ha permitido un modelo muy avanzado de programación extraordinariamente flexible llamada programación genérica.

2.4.5. Eclipse como IDE de Desarrollo.

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar todo tipo de aplicaciones libres. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE). Es una herramienta para el programador que facilita al

máximo la gestión de proyectos colaborativos mediante el control de versiones y es posible también con subversion, exportar e importar proyectos.

Es posible añadir nuevas funcionalidades al editor, a través de nuevos módulos ('plugins'), para programar en otros lenguajes de programación como C/C++, PHP, Python, Ruby, etc.

Entre sus principales características se puede destacar que es multiplataforma, aplicable a sistemas operativos como GNU/Linux, Solaris, Mac OSX, Windows; soportado para distintas arquitecturas; posee una estructura de plug-in que hace sencillo añadir nuevas características y funcionalidades; es posible realizar un control de versiones con cvs o con subversion; presenta funciones como resaltado de sintaxis, autocompletado, tabulador de un bloque de código seleccionado, entre otras utilidades de edición que ayudan enormemente al programador.

2.5. Diseño de la solución

Aunque el trabajo está orientado a la selección de una tecnología para mejorar la versión 1.0 del Middleware no es objetivo implementarlo completamente, sino implementar un prototipo funcional basado en la tecnología ICE que permita la transferencia de puntos, comandos e invocación de métodos remotos a Seguridad como se explica en los requisitos.

Por las razones anteriores todavía no se define la arquitectura para el módulo de Middleware simplemente se muestran los subsistemas y paquetes de diseño arquitectónicamente significativos y sus relaciones.

El modelo de diseño del prototipo del Middleware intenta modelar dos formas de comunicación entre los subsistemas del SCADA Guardián del ALBA, cliente-servidor para la comunicación con el módulo de Seguridad y publicación-subscripción para la transferencia de puntos desde la BDTR hasta los clientes interesados en este tipo de información. El mismo describe las clases y sus relaciones distribuidas en subsistemas y paquetes de diseño bien definidos y con una responsabilidad determinada.

2.5.1. Modelo de Comunicación Cliente-Servidor

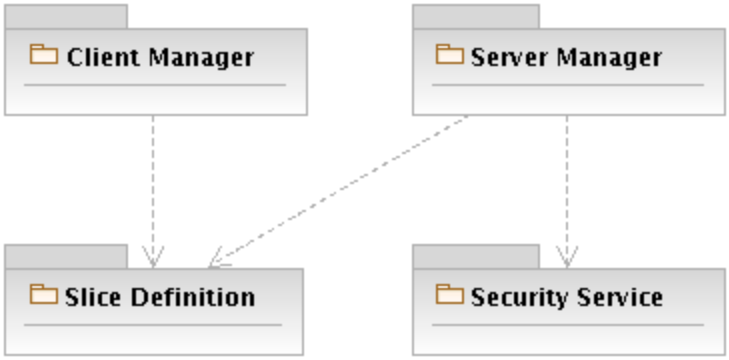

Figura 10: Relación entre Subsistemas de Diseño del Modelo Cliente-Servidor

La figura anterior muestra los diferentes subsistemas del modelo Cliente-Servidor y sus relaciones. Cada subsistema expone una interfaz que encapsula sus principales funcionalidades.

2.5.1.1. Subsistema Client Manager

El subsistema Client Manager define la lógica para un cliente de Middleware basado en la tecnología ICE, sin embargo expone a los módulos el acceso a las funcionalidades del subsistema a través de una interfaz virtual pura desacoplando a los mismos de la tecnología y de los mecanismos de comunicación utilizados. Para ello utiliza una factory “ClientResolver” encargada de devolver un objeto del tipo de la interfaz “Client” que en si llevará un objeto de la clase que implementa esta interfaz “IceClient”, ejecutando sus métodos por polimorfismo en el momento que son invocados desde la interfaz que hace de fachada, “Client”.

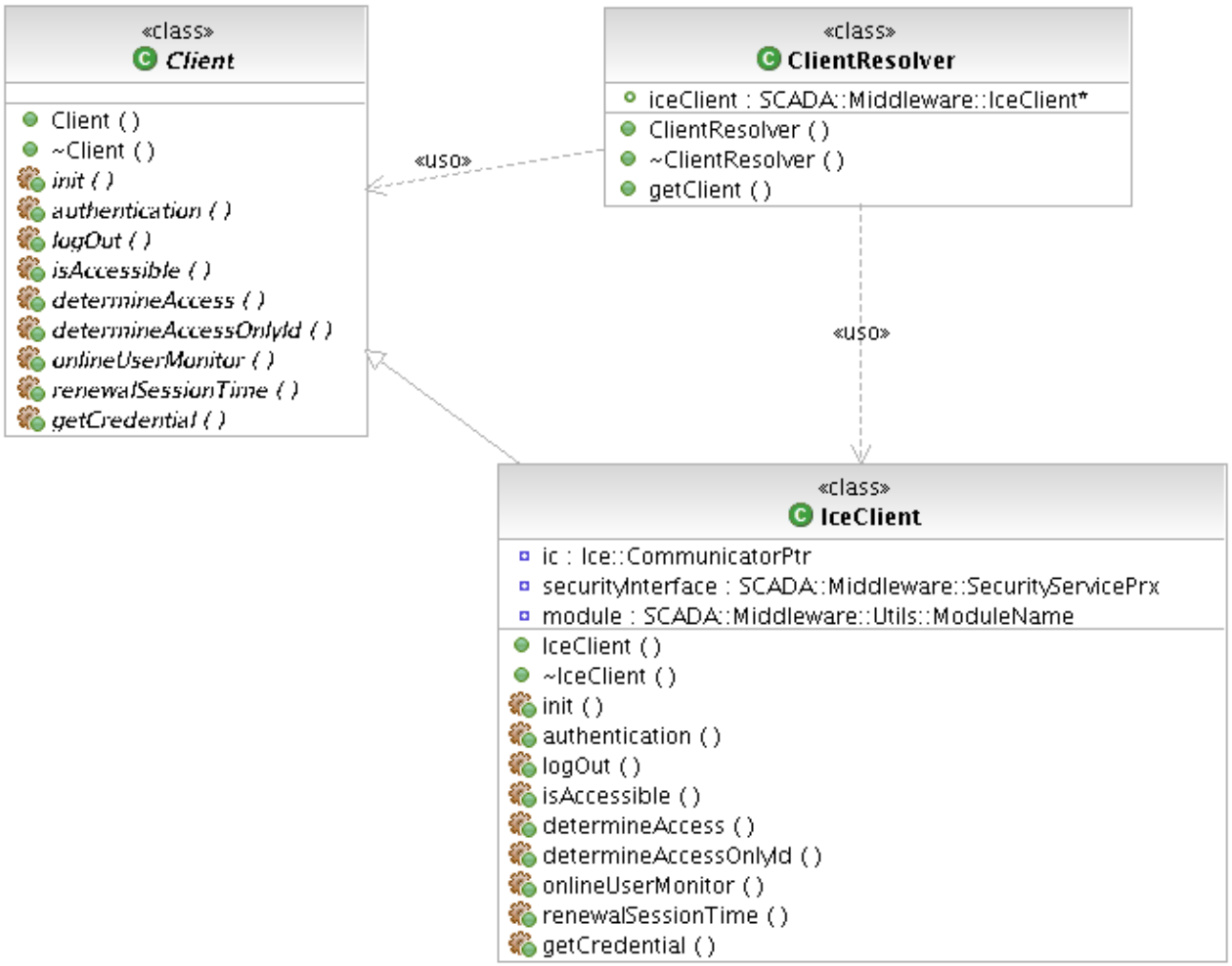

Figura 11: Diagrama de Clases del Subsistema Client Manager

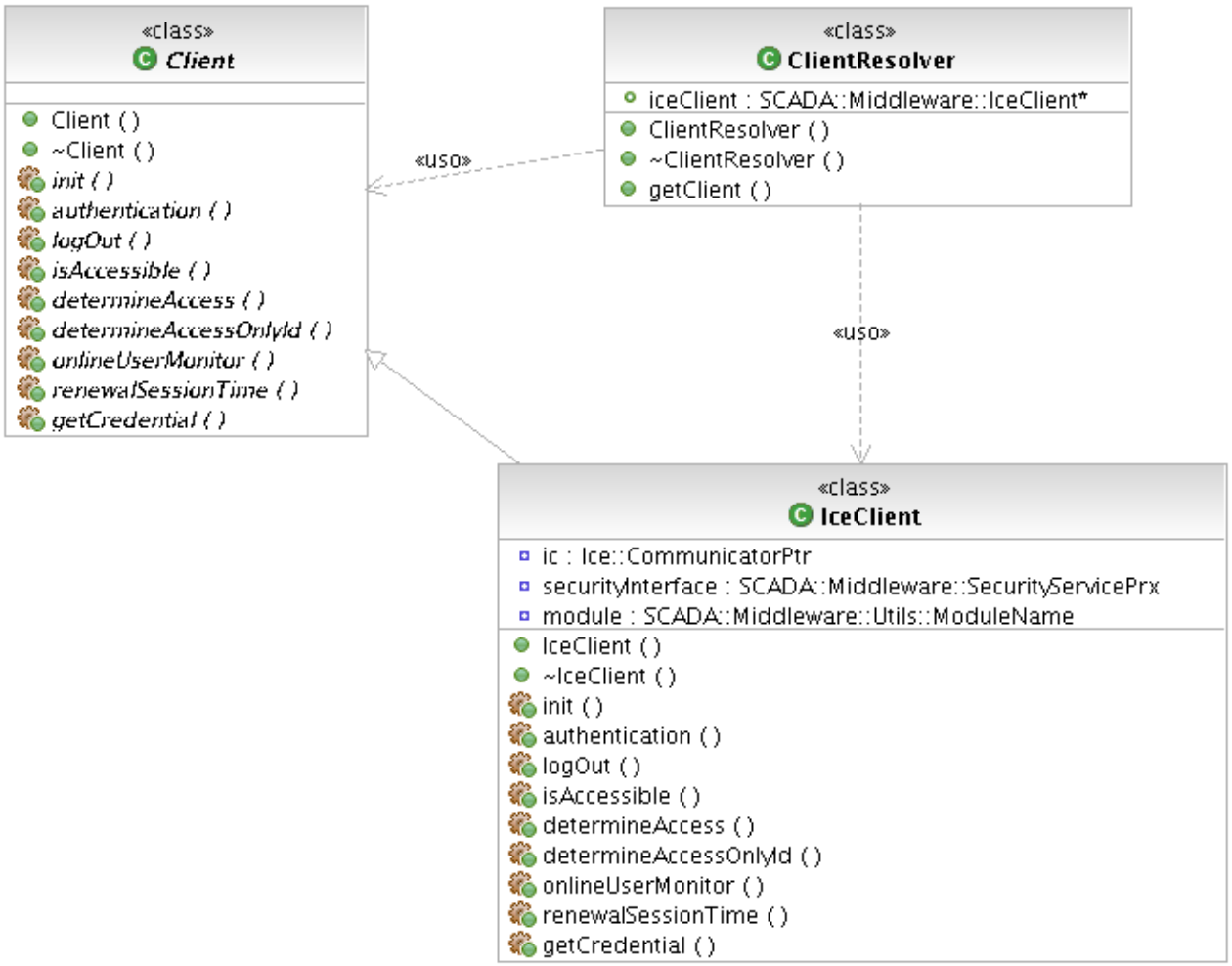

Figura 12: Diagrama de Clases del Subsistema Client Manager

Tabla 1: Descripción de la Clase ClientResolver

Tabla 2: Descripción de la Interfaz Client

Tabla 3: Descripción de la Clase IceClient

2.5.1.2. Subsistema Server Manager

El presente subsistema define la lógica para un servidor de Middleware basado en la tecnología ICE, sin embargo expone a los módulos el acceso a las funcionalidades del subsistema a través de una interfaz virtual pura desacoplando a los clientes de la tecnología y de los mecanismos de comunicación utilizados. Para ello utiliza una *factory "ServerResolver"* encargada de devolver un objeto del tipo de la interfaz *"Server"* que en si llevará un objeto de la clase que implementa esta interfaz *"IceServer",* ejecutando sus métodos por polimorfismo en el momento que son invocados desde la interfaz que hace de fachada, *"Server"*.

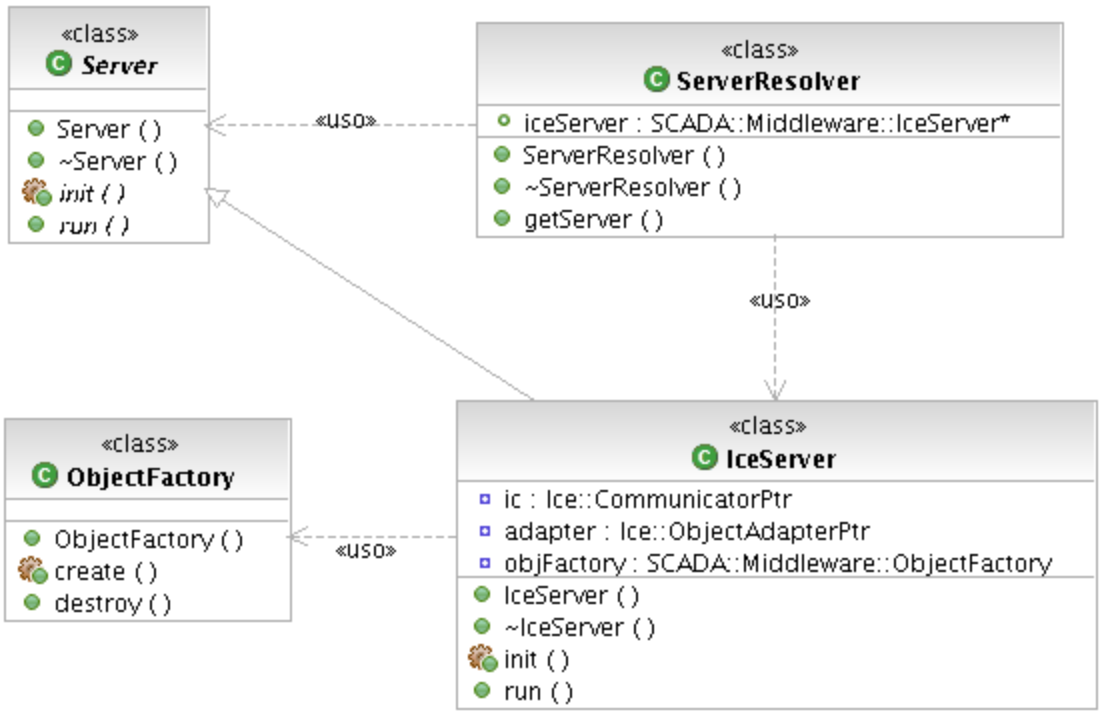

Figura 1: Diagrama de Clases del Subsistema Server Manager

Tabla 4: Descripción de la Clase ServerResolver

Tabla 5: Descripción de la Interfaz Server

Tabla 6: Descripción de la Clase IceServer

2.5.1.3. Subsistema Slice Definition

Este presente subsistema expone las interfaces, stub y skeleton generados a partir de la definición de interfaces en el SLICE *sync.ice*. Los mismos son utilizados para comunicar clientes y servidores basados en ICE y lograr un entendimiento entre ellos.

Slice es el un lenguaje de definición o especificación de interfaces como lo es IDL para TAO. En él se definen los tipos de datos que se van a intercambiar y las interfaces que van a utilizar los clientes y servidores para comunicarse.

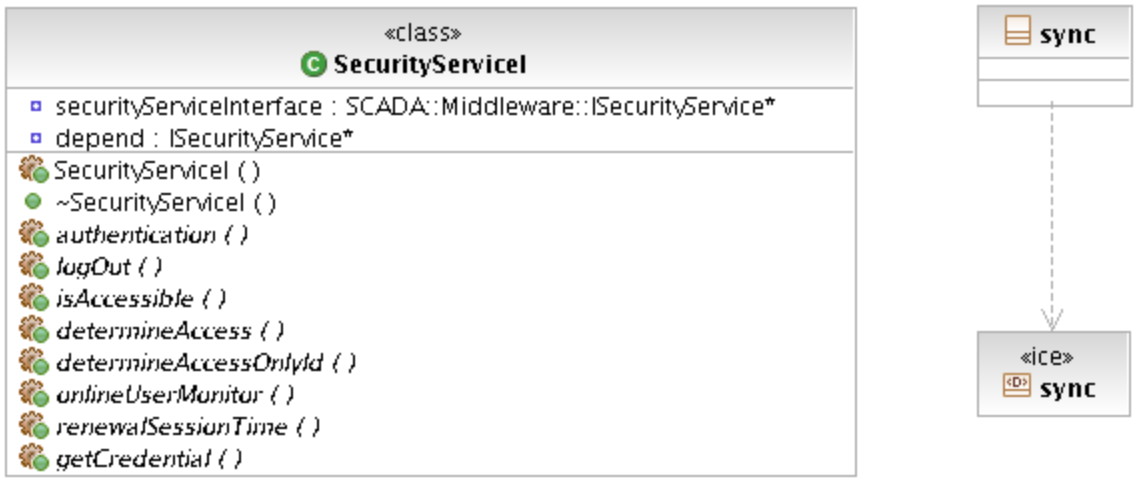

Figura 14: Diagrama de Clases del Subsistema Slice Definition

Sync.ice: es el fichero *SLICE* que define los datos y las interfaces para la comunicación entre clientes y servidores.

Sync: es el fichero que se genera a partir de compilar el fichero ***Sync.ice*** usando el compilador SLICE de ICE. Define los stub y sleleton, proxies para clientes y servidores.

SecurityServiceI: es la implementación que hace el Middleware de las definiciones que se encuentran *en los stub y los skeleton.*

2.5.1.4. Subsistema Security Service

El presente subsistema expone al servidor de Seguridad la interfaz ISecurityService, la misma define todos los métodos que pueden ser invocados desde el cliente y el módulo de Seguridad es el encargado de implementarlos.

Figura 15: Diagrama de Clases del Subsistema Security Service

2.5.2. Modelo de Comunicación Publicación-Subscripción

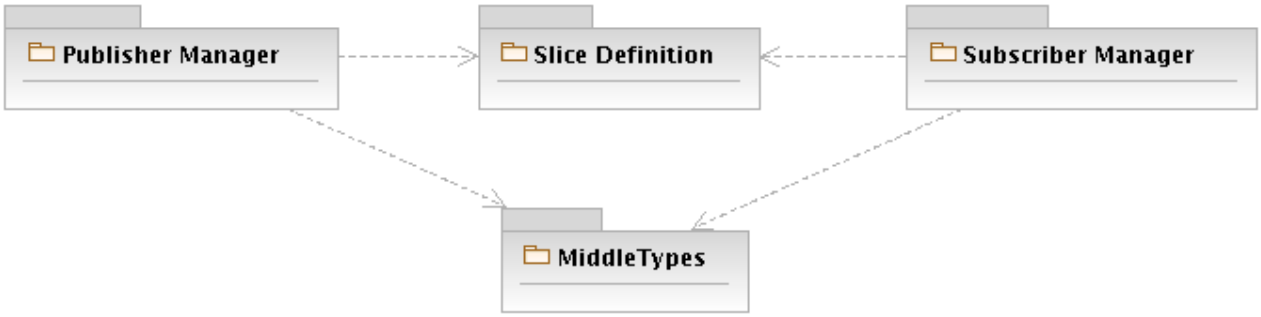

Figura 16: Relación entre Subsistemas de Diseño del Modelo Publicación-Subscripción

La figura anterior muestra los diferentes subsistemas del modelo publicación-subscripción y sus relaciones. Cada subsistema expone una interfaz que encapsula sus principales funcionalidades.

2.5.2.1. Subsistema Publisher Manager

El presente subsistema define la lógica para un publicador de Middleware basado en la tecnología ICE, sin embargo expone a los módulos el acceso a las funcionalidades del subsistema a través de una interfaz virtual pura desacoplando a los clientes de la tecnología y de los mecanismos de

comunicación utilizados. Para ello utiliza una *factory "PublisherResolver"* encargada de devolver un objeto del tipo de la interfaz *"Publisher"* que en si llevará un objeto de la clase que implementa esta interfaz *"IcePublisher",* ejecutando sus métodos por polimorfismo en el momento que son invocados desde la interfaz que hace de fachada, *"Publisher"*.

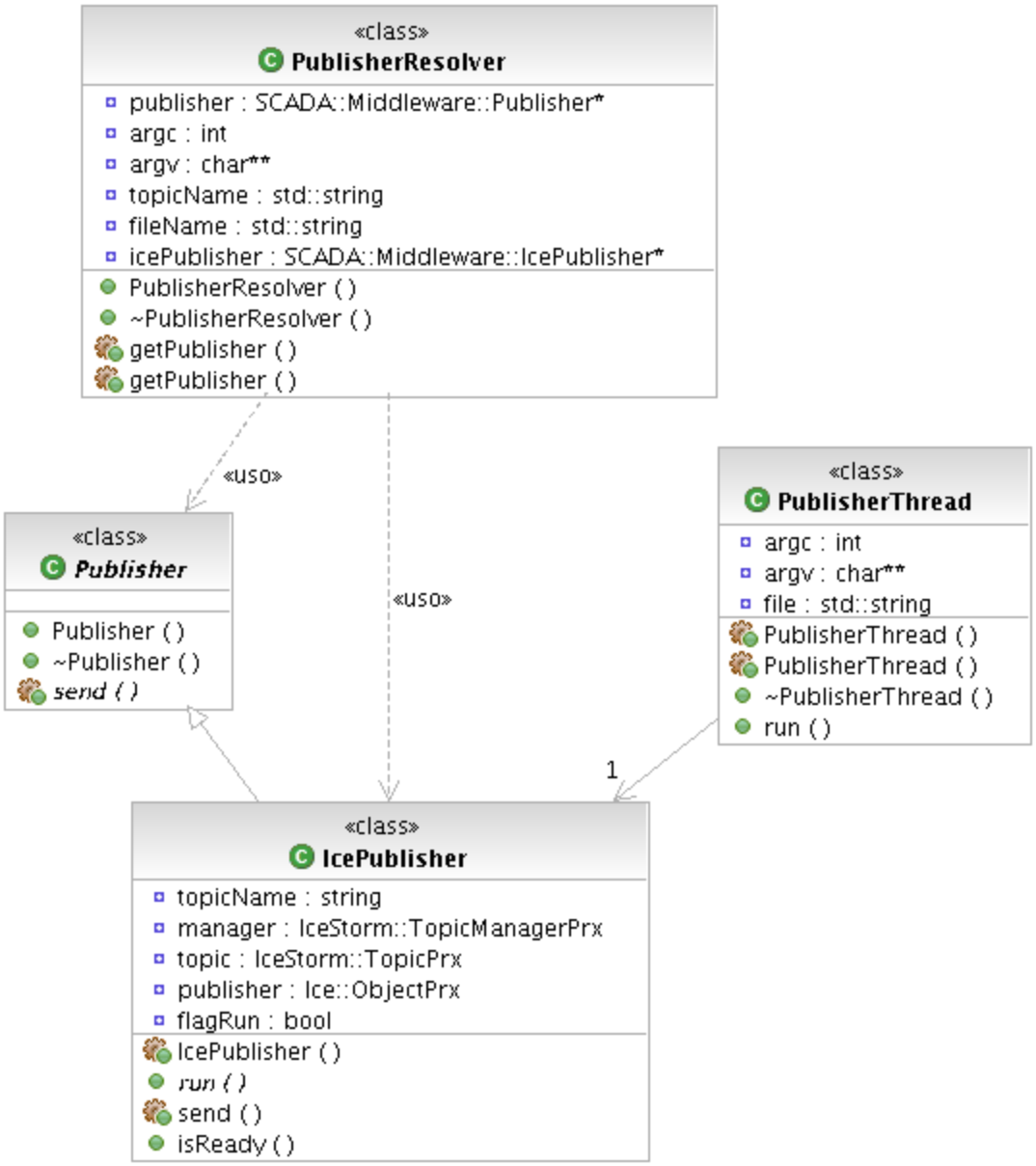

Figura 17: Diagrama de Clases del Subsistema Publisher Manager

Tabla 7: Descripción de la Clase PublisherResolver

Tabla 8: Descripción de la Interfaz Publisher

Tabla 9: Descripción de la Clase IcePublisher

2.5.2.2. Subsistema Subscriber Manager

El presente subsistema define la lógica de un subscriptor de Middleware basado en la tecnología ICE, sin embargo expone a los módulos el acceso a las funcionalidades del subsistema a través de una interfaz virtual pura desacoplando a los clientes de la tecnología y de los mecanismos de comunicación utilizados. Para ello utiliza una *factory "SubscriberResolver"* encargada de devolver un objeto del tipo de la interfaz *"Subscriber"* que en si llevará un objeto de la clase que implementa esta

interfaz *"MiddleSubscriber"*, ejecutando sus métodos por polimorfismo en el momento que son invocados desde la interfaz que hace de fachada, *"Subscriber"*.

Figura 18: Diagrama de Clases del Subsistema Subscriber Manager

Tabla 10: Descripción de la Clase SubscriberResolver

Tabla 11: Descripción de la Interfaz Subscriber

Tabla 12: Descripción de la Clase MiddleSubscriber

2.5.2.3. Subsistema MiddleTypes

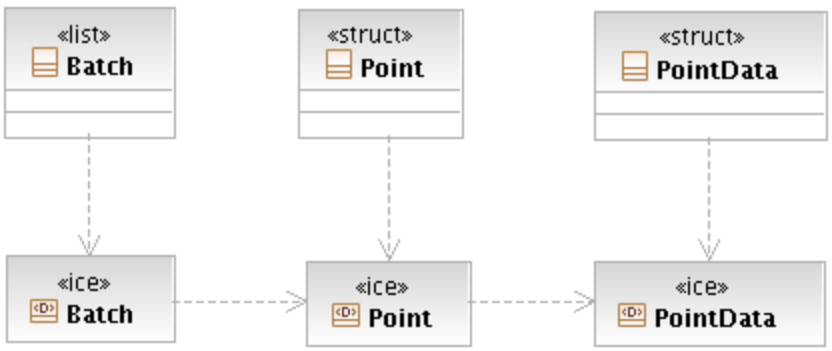

Figura 19: Diagrama de Clases del Subsistema MiddleTypes

La figura anterior muestra los tipos de datos fundamentales definidos en la solución de Middleware 2.0. Entender este punto es importante porque de acuerdo a las últimas definiciones se acordó que el Middleware sea el encargado de definir los tipos de datos que se manejan en el Guardián del ALBA, como puntos, alarmas, etc. Esta solución permite mejorar considerablemente el rendimiento del

subsistema de comunicación porque no obliga al Middleware a convertir los datos que envía la BDTR aumentando así la velocidad de transmisión de la data.

La tecnología ICE al igual que TAO tiene un lenguaje de definición de datos y/o interfaces, en este caso se conoce como ***SLICE*** y los ficheros creados deben tener la extensión ***.ice***. La figura anterior define con el estereotipo de **<<ice>>** los datos *Point*, *Batch* y *PointData* que a la hora de ser implementados deben tener la extensión ***.ice***. Usando el compilador *SLICE* de la tecnología se generan los stub que definen el dato como tal y son los que cada modulo va a utilizar para el manejo de puntos, alarmas, etc. Las estructuras Point, Pointdata, y la lista de Batch son las definiciones que pueden ser utilizadas.

En el momento de la implementación se podrá entender cómo se generan y se usan cada uno de los tipos de datos definidos por el Middleware 2.0 en los SLICE.

2.5.3. Diseño de Comportamiento por Subsistemas

2.5.3.1. Subsistema Client Manager

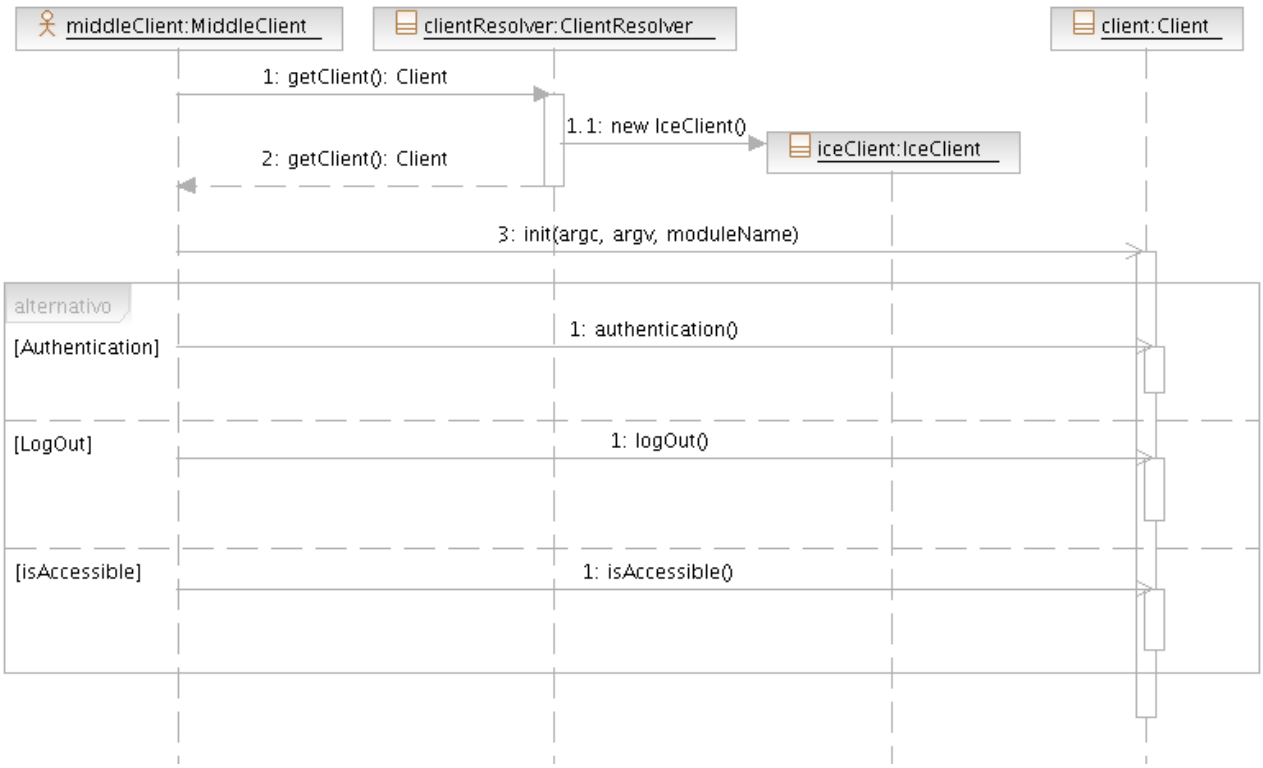

Figura 20: Diagrama de Secuencia Obtener Cliente e Invocar sus Métodos

El diagrama de secuencia anterior describe como los clientes del Seguridad pueden obtener la interfaz cliente que provee el Middleware para invocar los métodos que implementa el módulo de Seguridad. También muestra como después de invocar el método *init()* encargado de inicializar el cliente se pueden invocar cualquiera de los métodos que define la interfaz *Client()*. El resto de los métodos no se incluye en el diagrama pero para ello se sigue la misma lógica.

2.5.3.2. Subsistema Server Manager

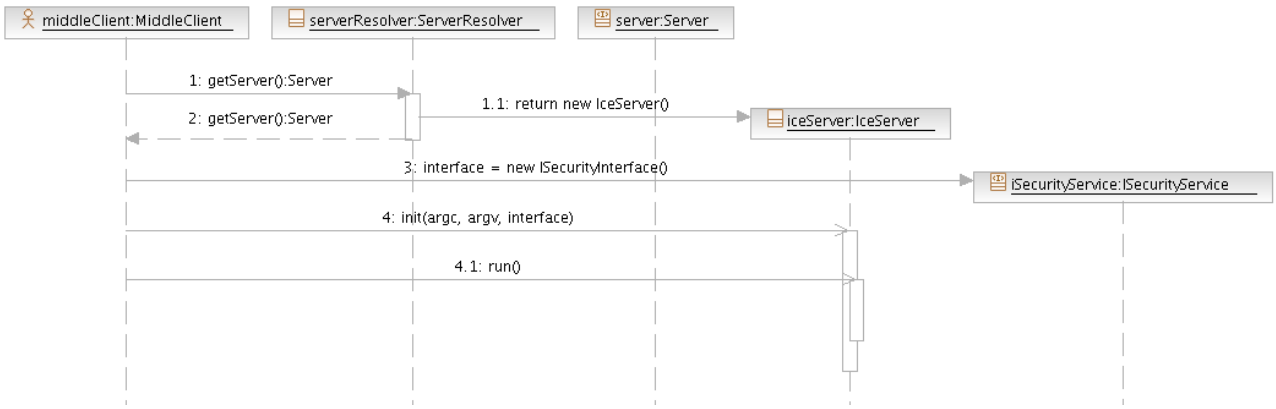

Figura 21: Diagrama de Secuencia Obtener e Iniciar Servidor

El diagrama de secuencia anterior describe como el módulo de Seguridad puede obtener la interfaz servidor que provee el Middleware. También como registrar mediante el método *init()* la clase que implementa los métodos que serán invocados desde los clientes y la ejecución del servidor a través del método *run()*, ambos a través de la interface *Server*.

2.5.3.3. Subsistema Publisher Manager

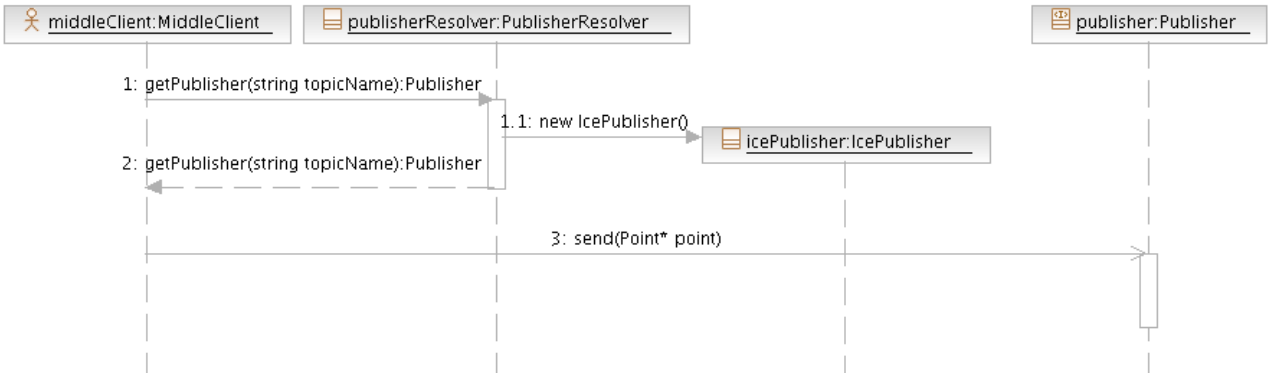

Figura 22: Diagrama de Secuencia Obtener Publicador y Enviar Puntos

El diagrama de secuencia de la figura anterior describe como un cliente de Middleware, BDTR por ejemplo puede obtener la interfaz *Publisher* e invocar el método *send()* para publicar puntos.

2.5.3.4. Subsistema Subscriber Manager

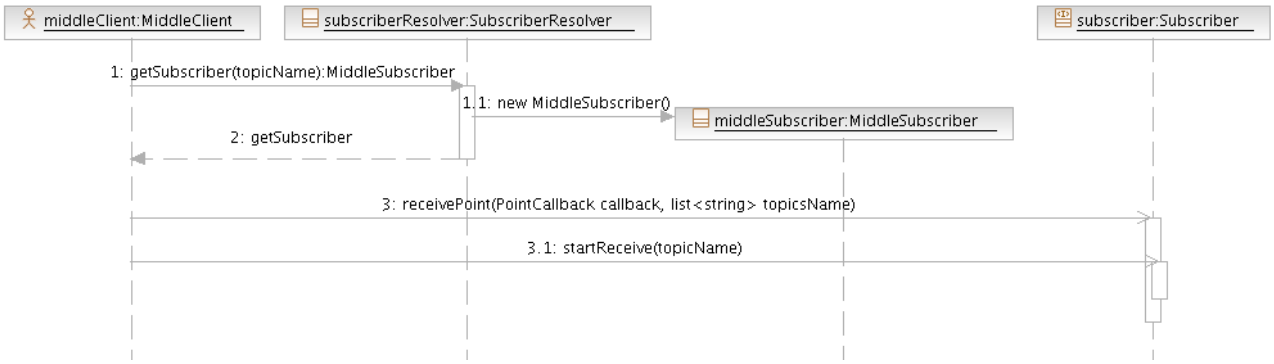

Figura 23: Diagrama de Secuencia Obtener Subscriptor y Recibir Puntos

El diagrama de secuencia anterior describe la interacción entre las clases del subsistema y como un cliente de Middleware como la BDH puede obtener la interfaz *Subscriber* e invocar el método *receivePoint()* para indicar que quiere recibir puntos y posteriormente iniciar la recepción de puntos usando el método *startReceiver()*.

CAPÍTULO 3: IMPLEMENTACIÓN DE LA SOLUCIÓN Y EVALUACIÓN DE LOS RESULTADOS

3.1. Introducción

El presente capítulo presenta el prototipo funcional de Middleware en términos de componentes, es decir, archivos de código fuente, scripts, binarios y similares, con el objetivo de mostrar los componentes que se obtienen de implementar las clases y los subsistemas definidos en el diseño. También describe la vista de implantación del sistema y los casos de pruebas desarrollados para validar la solución implementada.

3.2. Vista de Implementación

El principal objetivo de la implementación del prototipo funcional de Middleware es proveer mecanismos de comunicación entre los subsistemas del SCADA Guardián del ALBA permitiendo la comunicación asincrónica para el intercambio de variables (puntos), y comandos y la comunicación sincrónica para realizar peticiones clientes-servidor al módulo de Seguridad. Con el fin de lograr estos objetivos y brindar las funcionalidades requeridas por el cliente, la implementación se realiza basada en la tecnología ICE comentada en los capítulos 1 y 2 siendo la tecnología Middleware con mejores prestaciones en la actualidad y que permite implementar la solución sobre C++. Esta implementación logra la exposición interfaces de comunicación que permite la interacción entre los subsistemas distribuidos del SCADA como principal objetivo del cualquier Middleware o sistemas de comunicación distribuida.

3.3. Modelo de Implementación

Para lograr un modelo de comunicación abstracto para los clientes, normalmente se divide la implementación en niveles o capas de abstracción, donde cada nivel encapsula una parte importante del sistema, permitiendo modificar una de ellas, sin afectar la arquitectura, y en menor medida la interacción con el cliente. Cada nivel tiene una responsabilidad específica en el sistema, pero solo una está en constante interacción con otras aplicaciones o clientes, el nivel de aplicación o de interfaces. Esta solución no implementa una arquitectura de este tipo porque se está desarrollando un prototipo funcional y no un sistema como un todo, aún así brinda una serie de interfaces de comunicación que abstraen al cliente de los mecanismos implementados.

Aunque se realizó un diseño no significa que exista un ejecutable o biblioteca por cada paquete, subsistema o clase del mismo. De la solución salen tres componentes o bibliotecas que serán usadas por los módulos del SCADA para intercambiar información de forma distribuida, IceDataTypes.so, SyncComm.so, y AsynComm.so. Las mismas se pueden observar en dos modelos de implementación separados teniendo en cuenta que el prototipo de comunicación implementa dos mecanismos de comunicación, sincrónico y asincrónico.

3.3.1. Modelo de implementación sincrónico

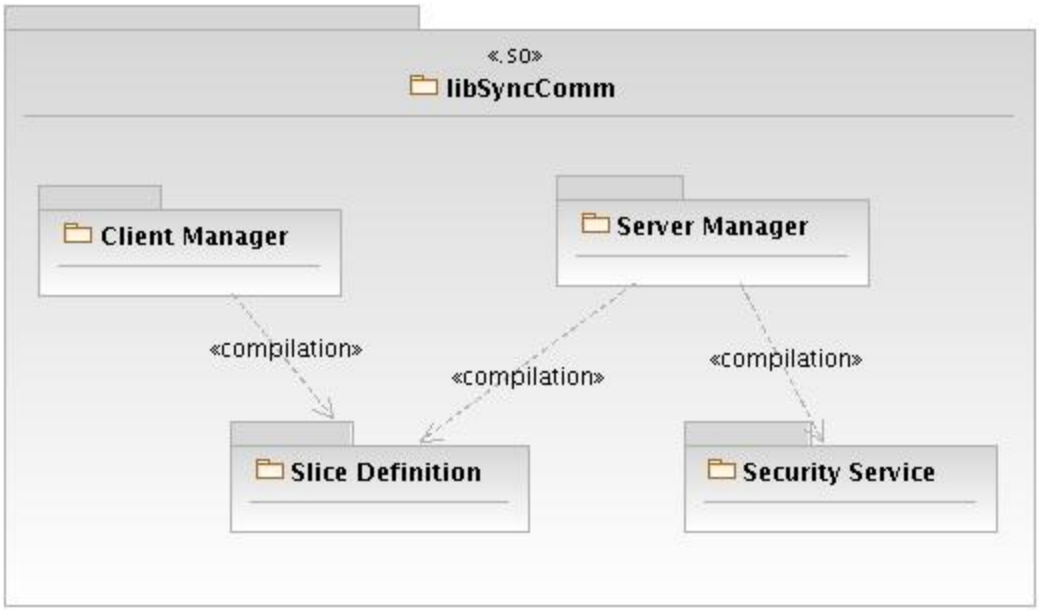

Figura 24: Diagrama de Componentes de la Biblioteca libSyncComm

3.3.1.1. Descripción de Componentes

libSyncComm.so: Este componente (biblioteca) implementa los mecanismos de comunicación sincrónica brindando interfaces de comunicación que permiten la interacción de los subsistemas del SCADA con el módulo de Seguridad de este. Esta biblioteca se genera a partir de la implementación de cada uno de los componentes agrupados en los subsistemas de implementación Client Manager, Server Manager, Slice Definition y Security Service.

Componentes agrupados en Client Manager

Este subsistema encapsula un grupo de componentes que implementan los mecanismos de comunicación de la parte del cliente y brinda una interfaz de comunicación que expone los métodos que pueden ser invocados desde los módulos del SCADA enviando una solicitud al servidor de Seguridad. El resto de los componentes del subsistema se encargan de resolver que estas peticiones lleguen al servidor usando un cliente de la tecnología ICE basado en el paradigma cliente-servidor.

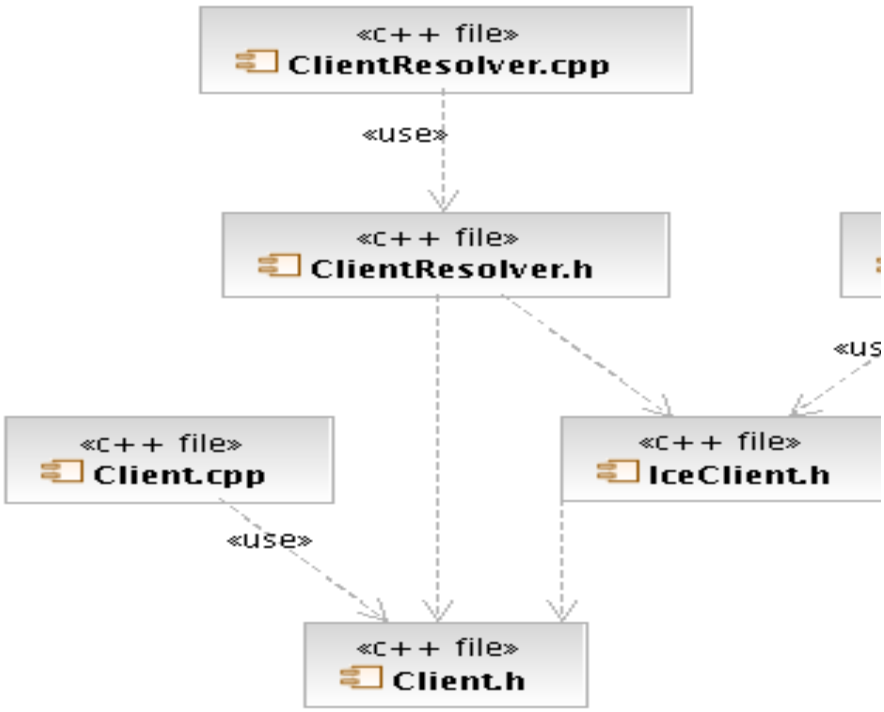

Figura 25: Diagrama de Componentes del Subsistema Client Manager

Componentes agrupados en Server Manager

Este subsistema encapsula un grupo de componentes que implementan los mecanismos de comunicación de la parte del servidor y brinda una interfaz de comunicación que expone los métodos que pueden ser invocados desde el servidor de Seguridad para registrar su implementación, la que va a ser usada por el Middleware para resolver las peticiones realizadas desde las aplicaciones cliente. El resto de los componentes del subsistema se encargan de resolver que estas peticiones y dar una respuesta a los clientes usando el paradigma cliente-servidor propuesto por la tecnología ICE.

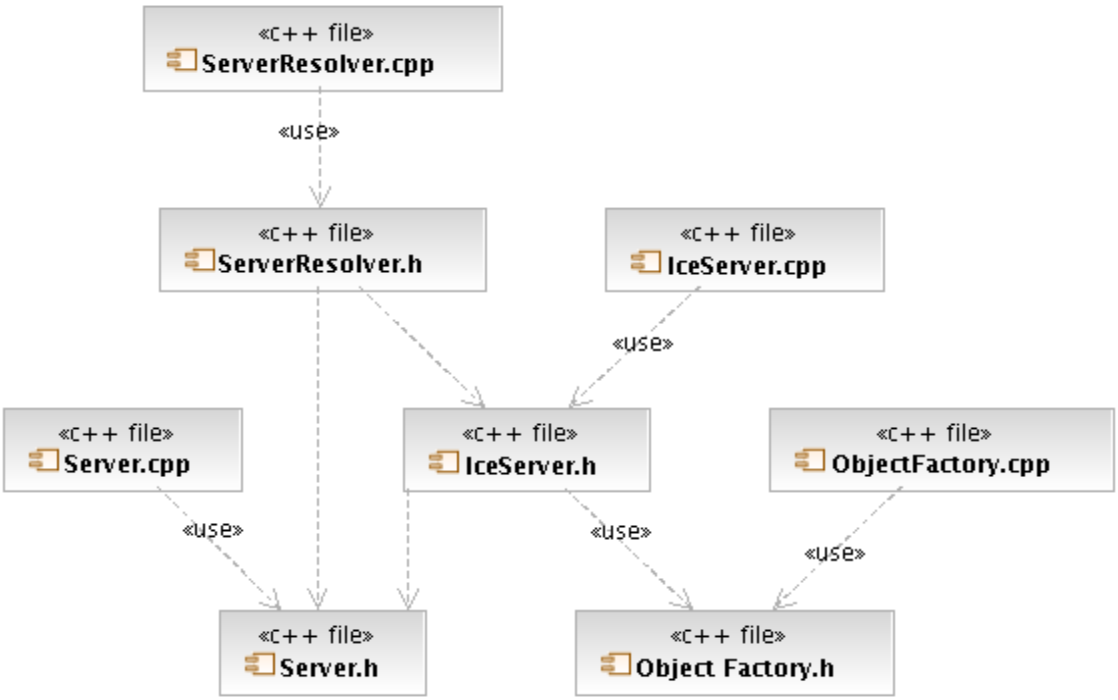

Figura 26: Diagrama de Componentes del Subsistema Server Manager

Componentes agrupados en Slice Definition

Este subsistema encapsula un grupo de componentes que implementan las definiciones de datos e interfaces que van a ser usadas por las aplicaciones que usen el Middleware basado en ICE. Para estas definiciones, la tecnología provee un lenguaje de definición de interfaces o SLICE donde se definen los datos que va a entender clientes y servidores. También se generan los stub y skeleton, proxies, a partir de compilar los ficheros con extensión .ice usando el compilador slice2cpp. Los stub y skeleton son los que van a permitir que aplicaciones desarrolladas en diferentes lenguajes de programación puedan entenderse usando estas definiciones e interfaces.

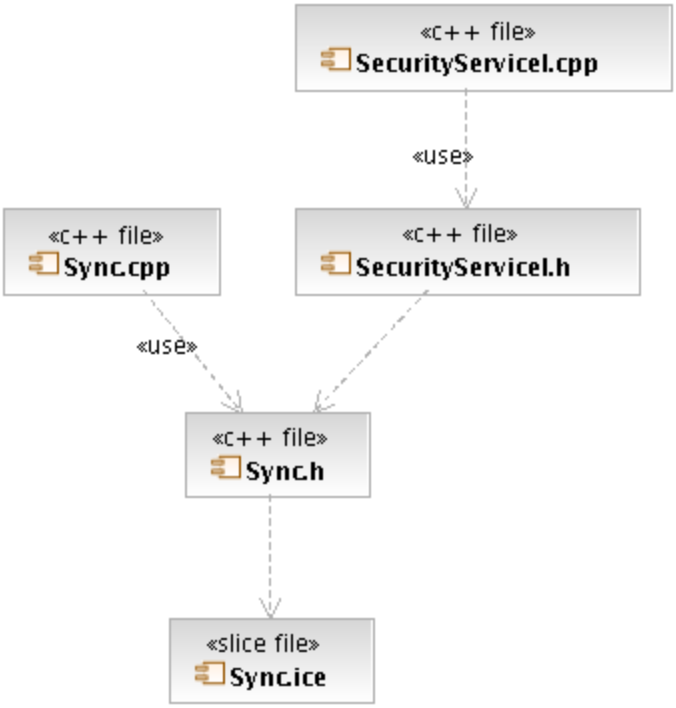

Figura 27: Diagrama de Componentes del Subsistema Slice Definition

3.3.2. Modelo de implementación asincrónico

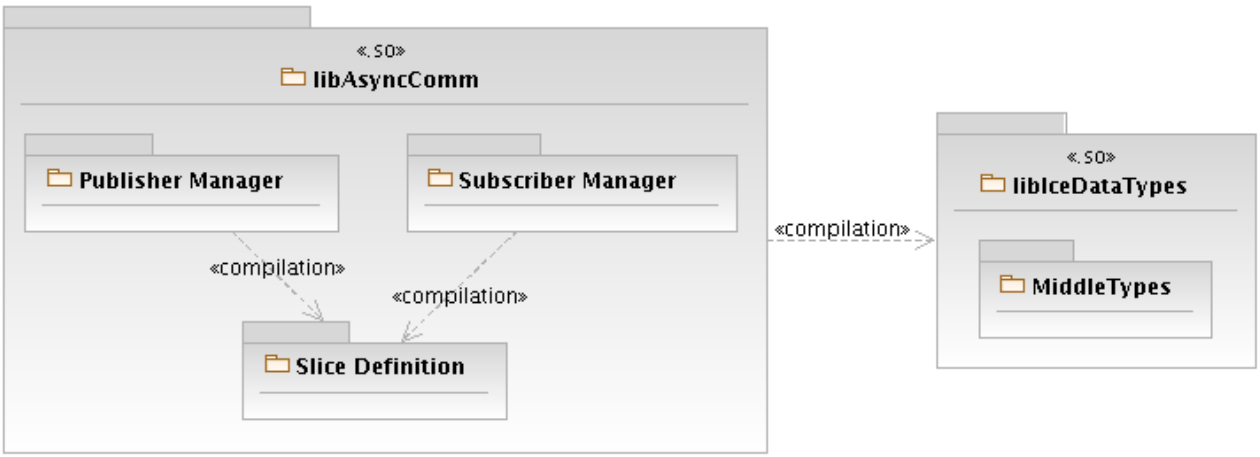

Figura 28: Modelo de Implementación Asincrónico

3.3.2.1. Descripción de componentes

libIceDataTypes.so: Este componente (biblioteca) implementa las definiciones de todos los tipos de datos que va a manejar la solución y los cuales serán manipulados por los diferentes subsistemas del SCADA y transmitidos por la red. Las definiciones existentes se corresponden con las definiciones de puntos y comandos del SCADA Guardián del ALBA. Esta bliblioteca se obtiene a partir de implementar los componentes agrupados en el paquete MiddleTypes.

libAsyncComm.so: Este componente (biblioteca) implementa los mecanismos de comunicación asincrónica brindando interfaces de comunicación que permiten la interacción de los subsistemas del SCADA para el intercambio de variables (puntos) y comandos. Esta biblioteca se genera a partir de la implementación de cada uno de los componentes agrupados en los subsistemas de implementación Publisher Manager, Subscriber Manager y Slice Definition.

Componentes agrupados en Publisher Manager

Este subsistema encapsula un grupo de componentes que implementan los mecanismos de comunicación de la parte del publicador y brinda una interfaz de comunicación que expone los métodos que pueden ser invocados desde los módulos del SCADA para enviar puntos, comandos y respuestas de comandos. El resto de los componentes del subsistema se encargan de resolver que estas peticiones lleguen al resto de los módulos usando el paradigma publicación-suscripción propuesto por la tecnología ICE.

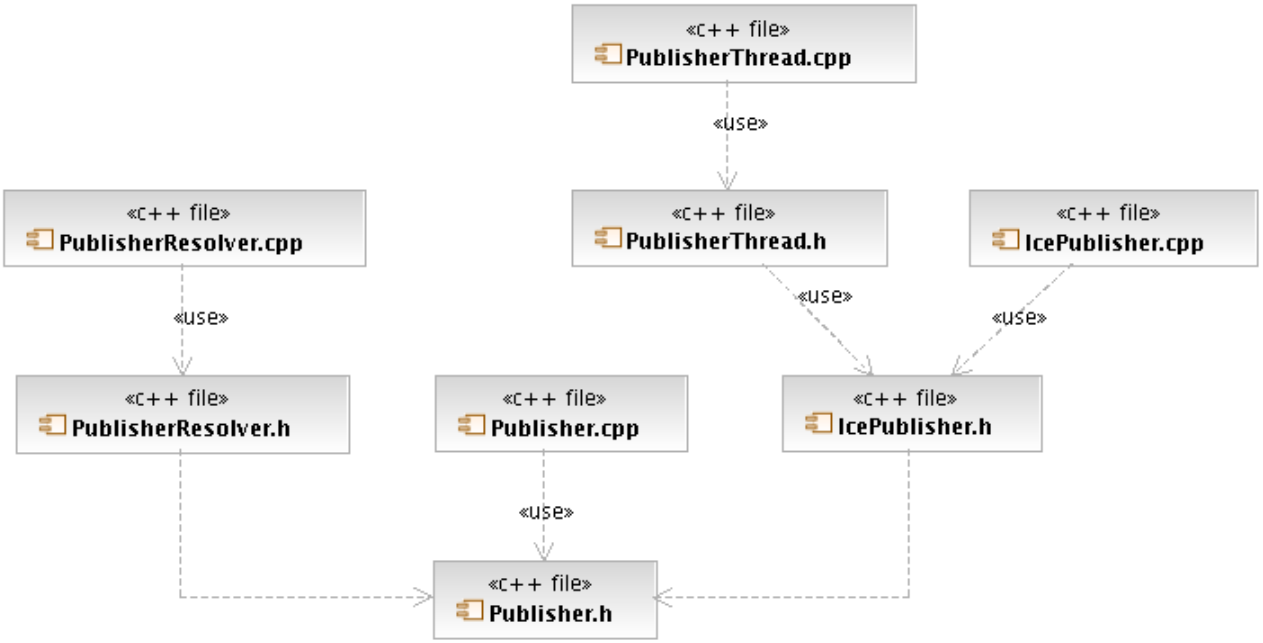

Figura 29: Diagrama de Componentes del Subsistema Publisher Manager

Componentes agrupados en Subscriber Manager

Este subsistema encapsula un grupo de componentes que implementan los mecanismos de comunicación de la parte del suscriptor y brinda una interfaz de comunicación que expone los métodos que pueden ser invocados desde los módulos del SCADA para recibir puntos, comandos y respuestas de comandos. El resto de los componentes del subsistema se encargan de resolver estas peticiones usando el paradigma publicación-suscripción propuesto por la tecnología ICE.

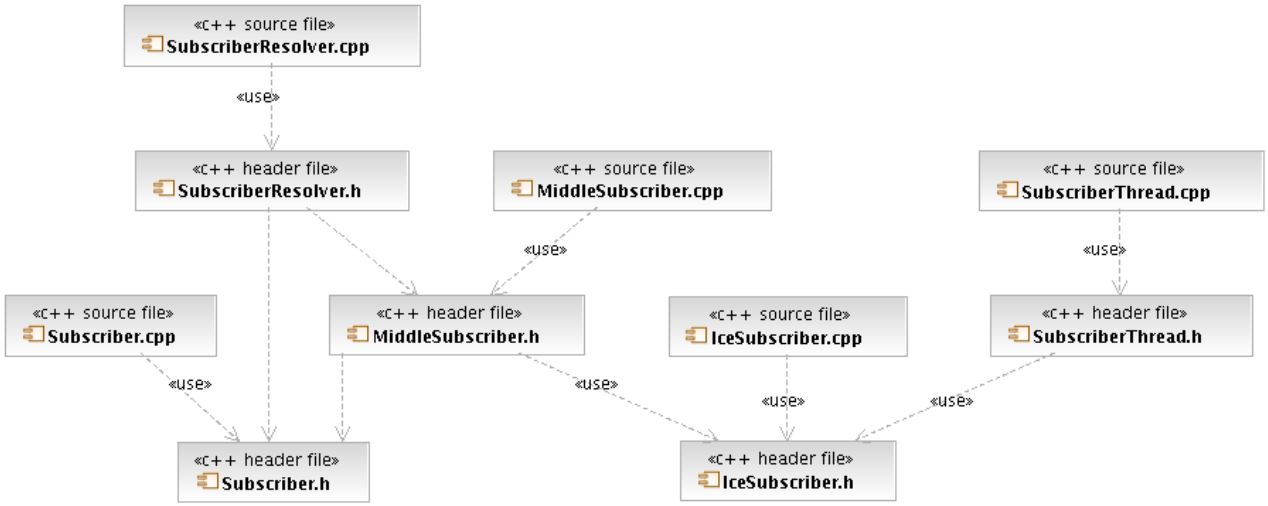

Figura 30: Diagrama de Componentes del Subsistema Subscriber Manager

Componentes agrupados en Slice Definition

Este subsistema encapsula un grupo de componentes que implementan las definiciones de datos e interfaces que van a ser usadas por las aplicaciones que usen el Middleware basado en ICE. Para estas definiciones, la tecnología provee un lenguaje de definición de interfaces o SLICE donde se definen los datos que va a entender publicadores y suscriptores. También se generan los stub y skeleton, proxies, a partir de compilar los ficheros con extensión .ice usando el compilador slice2cpp. Los stub y skeleton son los que van a permitir que aplicaciones desarrolladas en diferentes lenguajes de programación puedan entenderse usando estas definiciones e interfaces.

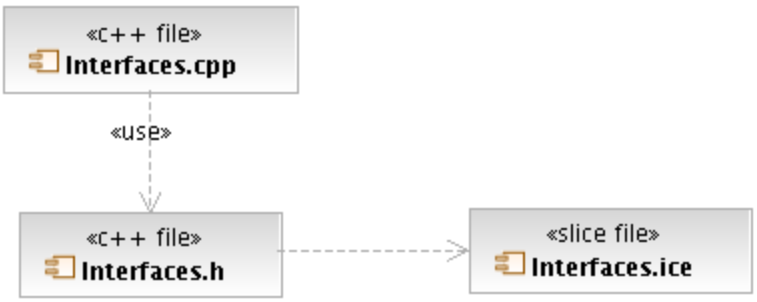

Figura 31: Diagrama de Componentes del Subsistema Slice Definition

Componentes agrupados en MiddleTypes

Este subsistema encapsula un grupo de componentes que implementan las definiciones de todos los tipos de datos que va a manejar la solución y los cuales serán manipulados por los diferentes subsistemas del SCADA y transmitidos por la red. Las definiciones existentes se corresponden con las definiciones de puntos y comandos del SCADA Guardián del ALBA.

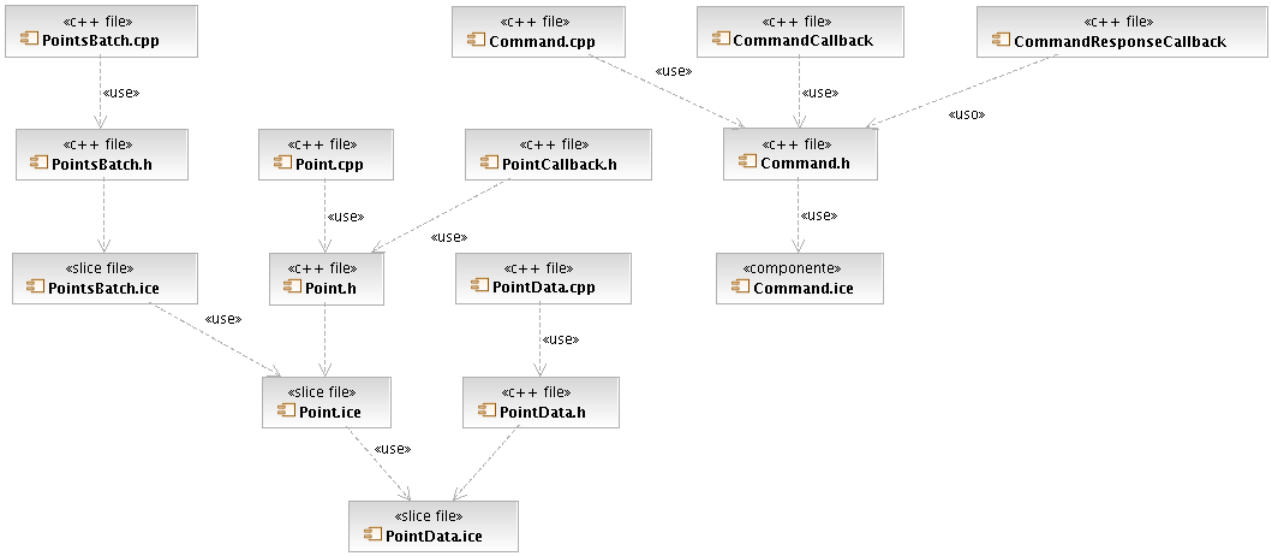

Figura 32: Diagrama de Componentes del Subsistema de MiddleTypes

3.4. Estilo de Código utilizado

La codificación siguiendo un estilo o estándar de código garantiza el desarrollo de un producto siguiendo un estilo de codificación único y uniforme.

3.4.1. Definición y Usabilidad de los Nombres

- Los nombres de las clases son sustantivos singulares.
- Los nombres deben reflejar el qué y no el cómo.
- Los nombres no deben revelar detalles de implantación.
- Escoger nombres lo suficientemente largo para ser expresivos, pero evitando manejar nombres que dificulten la labor de implantación.
- Evitar nombre que permitan una interpretación subjetiva (evitar ambigüedad y asegurar abstracción).
- Evitar redundancia no repitiendo nombre de clases en sus elementos.
- Concatenar calificadores de cómputo a las variables que almacenen el producto de tal cómputo

(avg, sum, min, max, index).

- Dado que los nombres generalmente son el producto de concatenar varias palabras, se debe emplear mayúscula para el inicio de cada palabra y minúscula para el resto de las letras para el caso de los nombres de métodos y funciones. Para el caso de los nombres de variables y atributos debe aplicarse la misma convención, con excepción de la primera letra del nombre, la cual debe ser en minúscula.
- Variables booleanas deben contener "ls" en su nombre.
- Los nombres de constantes deben contener solo letras mayúsculas.
- Minimizar el uso de abreviaciones. En caso de ser requeridas, se debe ser consistente en su uso y cada abreviación debe significar solo una cosa. En general agregar a la documentación las abreviaturas.
- Los nombres de los métodos son frases que incluyen verbos.
- Los nombres de los atributos y parámetros son frases con sustantivos.
- Evitar el reuso de nombres para distinto propósito.

3.4.2. Manejo de Errores

Se pueden manejar los errores mediante mecanismos de excepciones o mediante valores de retorno, aunque esto debe ser uniforme dentro de un mismo objeto.

Es buena práctica emplear herramientas para identificar errores en la codificación en caliente.

3.4.3. Documentación y Comentarios

- En el código debe documentarse en forma explicativa los pasos que se van ejecutando.
- Emplear oraciones completas al documentar código.
- Documentar mientras se programa.
- Documentar cualquiera cosa que no sea obvia en el código.
- Documentar eliminación de errores y cambios sobre el código.
- Al modificar el código se deben actualizar todos los comentarios y documentación asociada.
- Documentar cada rutina agregando: nombre del desarrollador, fecha, parámetros de entrada, valores de retorno, precondiciones, postcondiciones, dependencia con otros métodos o funciones y descripción general del algoritmo. Además, de realizarse cambios al código, debe indicarse el nombre de la persona que realizó el cambio, la fecha y la descripción del cambio, comenzando desde el o los cambios más recientes.

- Evitar agregar comentarios al final de líneas de código, salvo en el caso de declaraciones. En este caso tales comentarios deben estar alineados.
- Antes de la entrega de la aplicación, eliminar todos los comentarios superfluos y/o temporales con la finalidad de evitar confusiones en su mantenimiento.

3.4.4. Codificación

- Se establece un tamaño de identación estándar de tres (3) espacios, sin tabulaciones. Alinear secciones del código.
- Alinear verticalmente llaves de apertura y cierre.
- Usar espacios antes y después de los operadores que el lenguaje de programación permita.
- Emplear líneas en blanco para organizar el código, permitiendo crear ?párrafos? de código para una mejor lectura.
- Evitar colocar más de una sentencia por línea.
- Emplear constantes en sustitución de números o cadenas de caracteres literales.
- Minimizar el alcance de las variables para evitar confusión y facilitar el mantenimiento.
- Emplear cada variable y rutina solo para un propósito.
- Evitar el uso de variables públicas, sustituirlas por variables privadas y métodos que provean el valor de tal variable, para mantener el encapsulamiento.
- Minimizar el uso de conversiones de tipo forzadas (castings), cuando se requiera su uso, debe ser comentada la justificación.
- Emplear select-case o switch en sustitución de if anidados sobre la misma variables.
- Liberar apuntadores de manera explícita.
- Emplear i, j, k, l, p, q, r para contadores en ciclos.
- Comentar siempre las llaves que cierran.
- Emplear al máximo operadores del tipo: +=, *=, /=, -=, ++, --, etc.
- Mantener la modularidad del código bajo el criterio de la lógica que encierra, no exagerar la modularidad.
- Emplear correctamente los tipos de ciclos: si es al menos una vez usar do-while, si es ninguna o más veces usar while-do, y si se conoce el número exacto de ciclos usar for.
- Inicializar todas las variables.
- Emplear líneas en blanco para separar pasos lógicos (declaraciones, lazos, etc.).
- Siempre asignar NULL a los apuntadores luego de ser destruidos (solo aplica para C)
- Evitar prácticas que incrementan explosivamente la complejidad, como lo son: objetos y

variables globales y saltos tipo goto.

3.5. Vistas más significativas del código

Este epígrafe muestra fragmentos de códigos esenciales para el correcto funcionamiento del prototipo de Middleware 2.0.

3.5.1 Comenzar a recibir por una lista de tópicos.

Este fragmento de código muestra la implementación del método run() encargado de dejar listo al subscriptor para recibir todo a lo que esta subscrito(punto, comando y/o respuesta de comando).

```
int IceSubscriber::run(int argc, char** argv, const CommunicatorPtr& communicator)
{
Ice::ObjectPrx obj;
std::string data = parserArg(argc, argv);
obj = communicator->stringToProxy(data.c_str());
IceStorm::TopicManagerPrx manager;
manager = IceStorm::TopicManagerPrx::checkedCast(obj);
if(!manager)
{
cerr << " Invalid Proxy" << std::endl;
}
retriveTopic(manager);
Ice::ObjectAdapterPtr adapter;
adapter = communicator->createObjectAdapterWithEndpoints("Send.Subscriber",
"tcp:udp");

Ice::Identity subId;
string id;
subId.name = id;

if(subId.name.empty())
{
subId.name = IceUtil::generateUUID();
}
Ice::ObjectPrx subscriber;
subscriber = adapter->add(send, subId);
subscriber = subscriber->ice_oneway();
```

```
IceStorm::QoS qos;
try
{
```

```
std::list<IceStorm::TopicPrx>::const_iterator it;
for(it = topics->begin(); it != topics->end(); it++)
{
(*it)->subscribeAndGetPublisher(qos, subscriber);
}
}
catch(const IceStorm::AlreadySubscribed&)
{
if(id.empty())
{
throw;
}
}
start = true;
std::cout << "Ready to receive...." << std::endl;
adapter->activate();
communicator->waitForShutdown();
std::list<IceStorm::TopicPrx>::const_iterator it;
for(it = topics->begin(); it != topics->end(); it++)
{
(*it)->unsubscribe(subscriber);
}
return EXIT_SUCCESS;
}
```

3.5.2. Dejar listo al publicador para el envío.

Este fragmento de código muestra la implementación del método run() encargado de dejar listo el publicador para el envío, ya sea de puntos, de comandos o de respuesta de comando.

```
int IcePublisher::run(int pargc, char* pargv[])
{
argc = pargc + 2;
argv = new char*[pargc + 2];
for(int i = 0; i < pargc; i++)
{
argv[i] = pargv[i];
}
char* timeout = "--Ice.Override.Timeout=1000";
char* retry = "--Ice.RetryIntervals=0 100 500 1000 5000";
argv [pargc] = timeout;
argv [pargc + 1] = retry;
CommunicatorPtr communicator;
Ice::InitializationData initData;
communicator = initialize(argc, argv);
std::string data = parseArg(argc, argv);
Ice::ObjectPrx obj;
```

```
obj = communicator->stringToProxy(data.c_str());
IceStorm::TopicManagerPrx manager;
bool man = false;
while(!man)
{
try
{
manager = IceStorm::TopicManagerPrx::checkedCast(obj);
man = true;
}
catch(...)
{
man = false;
}
}
if(!manager)
{
cerr << "Invalid Proxy" << std::endl;
throw;
}
IceStorm::TopicPrx topic;
try
{
topic = manager->retrieve(topicName);
}
catch(const IceStorm::NoSuchTopic&)
{
try
{
topic = manager->create(topicName);
}
catch(const IceStorm::TopicExists&)
{
cerr << "Temporary failure. Try again" << std::endl;
throw;
}
}
Ice::ObjectPrx publisher;
publisher = topic->getPublisher();
sender = SendPrx::uncheckedCast(publisher->ice_oneway());
flagRun = true;
std::cout << "ready o send" << std::endl;
communicator->waitForShutdown();
return 0;
}
```

3.6. Ejecución de las Pruebas

Las pruebas de software constituyen una parte importante de cualquier desarrollo de software garantizando la calidad del producto final desarrollado. Los métodos de pruebas más utilizados son los de Caja blanca y Caja Negra. Los métodos de caja blanca están orientados a probar el código y su funcionamiento haciendo necesario que los probadores conozcan el código y la lógica interna del mismo; las pruebas de caja negra evaluan las funcionalidades a nivel de interfaz, los resultados de las entradas y salidas de datos al sistema, en fin, como se comporta la aplicación ante la interacción con los usuarios u otros sistemas.
A continuación se muestran los casos de prueba de caja negra confeccionados para probar las principales funcionalidades del prototipo funcional de Middleware 2.0 especificadas como requisitos.

3.6.1. Ambiente de pruebas

A continuación se muestran los datos de las máquinas y el sistema operativo utilizado en la ejecución de las pruebas.

Recursos Físicos

Las computadoras utilizadas para la ejecución de las pruebas tienen las siguientes características:

- **PC1**

  Procesador Intel(R) Core(TM)2 CPU T5500 @ 1.66GHz

  Memoria RAM 1GB

- **PC2**

  Procesador Intel(R) Pentium(R) Dual CPU E2140 @ 1.60GHz

  Memoria RAM 1GB

- **PC3**

  Procesador Intel(R) Core(TM)2 Duo CPU T5870 @ 2.00GHz

  Memoria RAM 1GB

Recursos Lógicos

- Debian 5.0(lenny), Kernel Linux 2.6.26-2-68
- Código de AsynComm, biblioteca asincrónica del prototipo.

- Código del Publisher, stub de prueba.
- Código del Subscriber, stub de prueba.

3.6.2. Especificación de Escenarios de Pruebas

El desarrollo de estas pruebas tiene como objetivo fundamental determinar si el rendimiento en cuanto a la transferencia de puntos resuelve los problemas de la solución actual. Para realizar las pruebas que permitan cumplir con el objetivo anterior se ejecutó el siguiente escenario de pruebas.

3.6.2.1. Escenario de Pruebas

Este escenario de prueba permite validar la velocidad de transferencia de la data que soporta la tecnología y la solución del Prototipo Funcional de Middleware 2.0. El escenario aplicado fue el más real posible pero teniendo en cuenta solamente la transferencia de puntos. De forma general el orden de ejecución de las pruebas y la distribución de las máquinas fue la siguiente:

- En la PC3 se ejecutan los servicios de ICE, IceBox y IceStorm.
- En la PC1 se ejecuta un Subscriber escuchando puntos por tres tópicos(canales) diferentes para recibir 20 mil puntos enviados por cada Publisher para un total de 60 mil. Se mide el tiempo en que el Subscriber recibe los 60 mil puntos por cada envío o iteración. Así con 1000 muestras.
- En la PC2 se ejecuta un Subscriber escuchando puntos por tres tópicos(canales) diferentes para recibir 20 mil puntos enviados por cada Publisher para un total de 60 mil. Se mide el tiempo en que el Subscriber recibe los 60 mil puntos por cada envío o iteración. Así con 1000 muestras.
- En la PC3 también se ejecuta un Publisher (como una BDTR) y se pone a enviar puntos por tres tópicos diferentes. Por cada tópico va enviar 20 mil puntos, va esperar 1 segundo y volverá a enviar. Así hasta obtener 1000 muestras.

3.6.3. Diseño del caso de prueba.

3.6.3.1. CPR1 Envío y Recepción de puntos.

Descripción

El caso de prueba permite comprobar el envío y recepción de puntos por varios canales utilizando el servicio ICEStorm de ICE. Dos suscriptores se conectan al servicio de ICE escuchando por tres tópicos diferentes, en total recibirán 60 000 puntos, o sea 20 000 por canal. Tres publicadores se conectan al servicio para publicar cada uno por un tópico diferente.

Flujo central

- Se obtiene un subscriptor al servicio de ICE.
- Se le dice a este subscriptor por los tópicos que va a escuchar puntos y además la cantidad que va a escuchar en total.
- Se comienza a recibir puntos.
- Se obtiene un publicador al servicio de ICE.
- Se le dice por el tópico que va a publicar y la cantidad de puntos que va a enviar por ese tópico.
- Se crea un script que iniciará tres instancias de los publicadores por tres canales diferentes.

Condiciones de ejecución

Debe estarse ejecutando en una de las computadoras el IceBox, servicio de aplicaciones Ice. Este levanta el IceStorm.

Resultados

Caso #	Clases Válidas	Clases Inválidas	Resultados Esperados	Resultados Obtenidos	Observaciones
1	Cuando se levanta la aplicación está recibiendo puntos por los tres tópicos y marca el tiempo cuando recibe la cantidad de puntos que se le configuró.		Cada Subscriber debe recibir todos los puntos enviados por los Publisher para un total de 60 mil y el tiempo de recepción en cada muestra no debe superar el tiempo de 1 segundo.	Después de 1000 muestras recolectadas se pudo determinar que cada Subscriber recibió los 60 mil puntos en cada envío y el tiempo promedio de recepción fue: • El subscriber de la PC1 recibió los 60 mil puntos de cada envío en un tiempo promedio de 318, 74	

				milisegundos. • El subscriber de la PC2 recibió los 60 mil puntos de cada envío en un tiempo promedio de 359, 27 milisegundos. Es importante destacar que entre más grandes sean los tiempos entre un envío y otro, es decir 500 ms, 1 s, etc; más grandes o defasados pueden ser los tiempos de recepción en los Subscriber. Si tenemos en cuenta que estos últimos marcan el tiempo de recepción cuando le llega el último punto de los 60 y recibe de 3 Publisher que no están sincronizados, entonces la diferencia que existe entre los envíos de los Publisher puede traer los resultados anteriores. Eso no afecta porque en la realidad en un SCADA los que reciben puntos de dos BDTR distintas no van a esperar que le lleguen el total de los puntos que publica cada una para almacenarlos o hacer cualquier otra operación.	

Tabla 13: Resultados de las pruebas para envío y recepción de puntos.

CONCLUSIONES

Al finalizar el desarrollo de este trabajo se han logrado cumplir los objetivos, concluyendo que:
Luego de haber analizado y comparado las tecnologías actuales para el desarrollo de un Middleware y definido las características que este debe soportar se seleccionó la tecnología Ice como la más adecuada para la implementación de un Middleware que supere las condiciones de la versión 1.0 del Middleware del SCADA Guardián del ALBA. Se diseñaron las clases para las funcionalidades especificadas y se implementó un prototipo funcional al cual se le realizaron numerosas pruebas que corroboraron esta decisión.

RECOMENDACIONES

- Se recomienda integrar este prototipo con varios módulos del SCADA Guardián del ALBA.
- Implementar la versión 2.0 del Middleware del Guardián del ALBA usando la tecnología ICE.
- Profundizar en el estudio de la tecnología ICE para futuros trabajos.

REFERENCIAS BIBLIOGRÁFICAS

1. **Modesti, Mario R.** Sistemas de supervisión y monitoreo. Introducción a los sistemas SCADA. *Sistemas de supervisión y monitoreo. Introducción a los sistemas SCADA.* [Online] [Cited: Octubre 6, 2008.] http://www.profesores.frc.utn.edu.ar/industrial/sistemasinteligentes/UT4/4.htm.

2. **Wikipedia.** Electrónica de control. SCADA. *Electrónica de control. SCADA.* [Online] 2009. [Cited: Mayo 22, 2009.] http://es.wikipedia.org/wiki/SCADA.

3. **Wikipedia**. La enciclopedia libre. *Middleware.* [En línea] [Citado el: 12 de Febrero de 2008.] http://es.wikipedia.org/wiki/Middleware.

4. **Booch, Grady, Jacobson, Ivar and Rumbaugh, James. 2007.** *El Lenguaje Unificado de Modelado.* s.l. : ADDISON-WESLEY, 2007. 978-84-7829-087-1.

5. **Campo Vázquez, María Celeste. 2004.** *Tecnologías Middleware para el desarrollo de servicios en entornos de computación ubicua.* [En línea] 2004. [Citado el: 12 de Febrero de 2008.] http://www.gast.it.uc3m.es/theses/phd_celeste.pdf.

6. **Elvira Valenzuela, José Luis. 2000.** *¿Qué es CORBA?* [En línea] 2000. [Citado el: 6 de Mayo de 2008.] http://iteso.mx/~jluis/sd/corbaesp.ppt.

7. **Graham Lewis, Todd y Zoll, David "Gleef".** Linux Lots. *CORBA, ORBit y Baboon.* [En línea] [Citado el: 6 de Mayo de 2008.] http://www.linuxlots.com/~barreiro/spanish/gnome-es/faq/corba.html.

8. **Schmidt, Douglas C., y otros.** *TAO: a High-performance Endsystem Architecture for Real-time CORBA.* [En línea] [Citado el: 14 de Mayo de 2008.] http://www.cs.wustl.edu/~schmidt/PDF/ORB-endsystem.pdf.

9. **Henning, Michi y Spruiell, Mark. 2009.** *Distributed Programming with Ice.* [En linea] 2009. [Citado: Mayo 11, 2009.] http://www.zeroc.com/download/Ice/3.3/Ice-3.3.1.pdf.

10. **Pardo Castellote, Gerardo.** *OMG Data Distribution Service: Architectural Overview.* [En línea] [Citado el: 2007 de Mayo de 2008.] http://www.ll.mit.edu/HPEC/agendas/proc04/abstracts/pardo-castellote_gerardo.pdf.

11. **Object Computing Inc. 2007.** OpenDDS. [En línea] 2007. [Citado el: 21 de Mayo de 2008.] http://www.opendds.org/index.html.

12. **IBM. 2009.** *RSA: Rational Software Architect.* [En linea] 2009. [Citado: Abril 28, 2009.] http://www-01.ibm.com/software/awdtools/architect/swarchitect/index.html.

13. **—. 1999.** *El Proceso Unificado de Desarrollo de Software.* s.l. : ADDISON-WESLEY, 1999. ISBN 84-7829-036-2.

14. **Herrera Vázquez, Moisés. 2008.** *Introducción a la arquitectura del "Guardián del ALBA".* [En linea] 2008. [Citado: Febrero 5, 2009.]

15. **Douglas C. y Parsons, Jeff.** *Overview of the OMG Data Distribution Service.* [En línea] [Citado el: 7 de Mayo de 2008.] http://www.cs.wustl.edu/~schmidt/DDS.ppt.

16. **Schmidt, Douglas C., y otros.** *TAO: a High-performance Endsystem Architecture for Real-time CORBA.* [En línea] [Citado el: 14 de Mayo de 2008.] http://www.cs.wustl.edu/~schmidt/PDF/ORB-endsystem.pdf.

17. **Squeak Swiki. 2006.** *CORBA.* [En línea] 16 de Enero de 2006. [Citado el: 6 de Mayo de 2008.] http://wiki.squeak.org/squeak/2519 - 9k.

BIBLIOGRAFÍA

Basanta Gutiérrez, David y Tajes Martínez, Lourdes. 1999. *Tecnologías para el Desarrollo de Sistemas Distribuidos: Java versus Corba.* [En línea] Septiembre de 1999. [Citado el: 11 de Febrero de 2008.] http://di002.edv.uniovi.es/~lourdes/publicaciones/bt99.pdf.

Buades, Gabriel. 2002. *Sistemas Distribuidos.* [En línea] 21 de Febrero de 2002. [Citado el: 12 de Febrero de 2008.] http://dmi.uib.es/~bbuades/sistdistr/sistdistr.PPT.

Busch, Don. 2006. *The TAO Data Distribution Service and Advanced Acoustic Concepts' Data Casting Technology.* [En línea] 5 de Diciembre de 2006. [Citado el: 14 de Mayo de 2008.] http://www.omg.org/.../dds-info-day/dds_2006_12_11_Don_Busch_OCI_DDS-Vendor-Presentation-2006Dec05.pdf.

Departamento de Arquitectura de Computadoras UPC. 2005. *Conceptos generales de sistemas distribuidos.* [En línea] Septiembre de 2005. [Citado el: 11 de Febrero de 2008.] http://studies.ac.upc.edu/EPSC/FSD/FSD-ConceptosGenerales.pdf.

Departamento de Sistemas, Universidad EAFIT Colombia. *Principios de middleware orientado a objetos.* [En línea] [Citado el: 11 de Febrero de 2008.]

Díaz Díaz-Chirón, Tobías. 2005. *Control de concurrencia en sistemas distribuidos con middleware.* [En línea] 26 de Febrero de 2005. [Citado el: 6 de Mayo de 2008.] http://www.lacasadetruca.org/system/files/doc.pdf.

eProsima. eProsima: Soluciones en Tiempo Real-Expertos en Middleware de Red. *RTI Data Distribution Service.* [En línea] [Citado el: 7 de Mayo de 2008.] http://www.eprosima.com/dnneprosima/Software/Distribuci%C3%B3nDatos/Middleware/tabid/62/language/es-ES/Default.aspx.

Fundación GNOME. 2002. *Programación en el Entorno GNOME.* [En línea] 2002. [Citado el: 6 de Mayo de 2008.] http://www.calcifer.org/documentos/librognome/index.html.

García Castro, Alejandro y Sánchez Penas, Juan José. *ORBit2 como middleware para una aplicación multicapa, usando el servicio de nombres de CORBA.* [En línea] [Citado el: 12 de Febrero de 2008.] http://2005.guadec-es.org/.../articulos/Articulo%2008%20-%20Alejandro%20Garcia%20-%20Orbit2%20como%20middleware.pdf.

Guinea de Salas, Alejandro y Jorrín Abellán, Sergio. *Arquitectura SOA para la integración entre software libre y software propietario en entornos mixtos.* [En línea] [Citado el: 11 de Junio de 2008.] http://www.sigte.udg.es/jornadassiglibre2007/comun/1pdf/13.pdf.

Henning, Michi y Spruiell, Mark. 2007. *Distributed Programming with Ice.* [En línea] Agosto de 2007. http://www.zeroc.com.

Hurtado Jara, Omar. Monografías.com. *Sistemas Distribuidos.* [En línea] [Citado el: 15 de Abril de 2008.] http://www.monografias.com/trabajos16/sistemas-distribuidos/sistemas-distribuidos.shtml.

iespaña.es. 2002. *Redes de Comunicación.* [En línea] 19 de Mayo de 2002. [Citado el: 12 de Febrero de 2008.] http://elsitiodetelecomunicaciones.iespana.es/redescomunicacion.htm.

Instituto Nacional de Estadísitcas e Informática de Perú. inei.gob.pe. *¿Qué es Middleware?* [En línea] [Citado el: 12 de Febrero de 2008.] http://www.inei.gob.pe/biblioineipub/bancopub/inf/Lib5038/midd.HTM.

Joshi, Rajive. 2006. *A Comparison and Mapping of Data Distribution Service (DDS) and Java Message Service (JMS).* [En línea] 2006. [Citado el: 7 de Mayo de 2008.] http://www.dsp-fpga.com/articles/white_papers/joshi/.

López, G., y otros. Diseño de aplicaciones con Arquitectura Orientada a Servicios. Modelos de Ciclos de Vida ad-hoc. [En línea] [Citado el: 11 de Junio de 2008.] http://www.itba.edu.ar/capis/rtis.

Martínez Gomáriz, Enric. 2008. *Diseño de Sistemas Distribuidos.* [En línea] 2008. [Citado el: 12 de Febrero de 2008.] www.lsi.upc.edu/~gomariz/index_archivos/IntroduccionSD-EnricMartinez.pdf.

Martinez, Marcela. 2005. *Modelo de Madurez en Arquitectura Orientada a Servicios (SOA).* [En línea] 2005. [Citado el: 10 de Junio de 2008.] http://portal.veracruz.gob.mx/pls/portal/url/ITEM/199A79BBCBEE6318E044080020B27120.

Mastermagazine. 2004. *Conceptos básicos de la Arquitectura Orientada a Servicios.* [En línea] 8 de Noviembre de 2004. [Citado el: 3 de Marzo de 2008.] http://www.mastermagazine.info/articulo/3391.php.

Microsoft Corporation. 2005. *Service-oriented Architecture and the Paradigm Shift in Client Architecture.* [En línea] Enero de 2005. [Citado el: 3 de Marzo de 2008.] http://www.microsoft.com/interoperability.

Millán, José Antonio. Vocabulario de ordenadores en Internet. *Más wares.* [En línea] [Citado el: 12 de Febrero de 2008.] http://jamillan.com/v_wares.htm.

Monje, Raúl. 2006. *Sistemas Distribuidos.* [En línea] Agosto de 2006. [Citado el: 11 de Febrero de 2008.] www.inf.utfsm.cl/~rmonge/sd/apunte01x6.pdf.

Moya, Rodrigo. 2002. *Taller práctico de CORBA en GNOME.* [En línea] 2002. [Citado el: 23 de Mayo de 2008.] http://es.tldp.org/Presentaciones/200211hispalinux/moya/corba-hispalinux-2002.html.

Navarro Marset, Rafael. 2006-2007. *REST vs Web Services.* [En línea] 2006-2007. [Citado el: 5 de Junio de 2008.] http://www.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf.

Object Computing Inc. 2007. OpenDDS. [En línea] 2007. [Citado el: 21 de Mayo de 2008.] http://www.opendds.org/index.html.

Objet Computer. Inc. ociweb.com. *CHAPTER 31 Data Distribution Service.* [En línea]

Paramio Danta, Carlos Alberto. *REST: Representational State Transfer, en Ruby on Rails.* [En línea] [Citado el: 5 de Junio de 2008.] http://flossic.loba.es/Contenidos/actas/rest.pdf.

Pardo Castellote, Gerardo. *OMG Data Distribution Service: Architectural Overview.* [En línea] [Citado el: 2007 de Mayo de 2008.] http://www.ll.mit.edu/HPEC/agendas/proc04/abstracts/pardo-castellote_gerardo.pdf.

Quemada, Juan y Salvachúa, Joaquín. *Desplegando Arquitecturas Rest con RoR.* [En línea] [Citado el: 5 de Junio de 2008.] http://2007.conferenciarails.org/archivos/joaquin_salvachua_rest.pdf.

Querzoni, Leonardo, y otros. 2006. *Quality of Service in Publish/Subscribe Middleware.* [En línea] 26 de Abril de 2006. [Citado el: 7 de Mayo de 2008.] http://www.cse.wustl.edu/~corsaro/papers/pubsubChapter.pdf.

Rojo, J. Oscar. 2003. AUGCyL. *Introducción a los Sistemas Distribidos.* [En línea] 2003. [Citado el: 11 de Febrero de 2008.] http://www.augcyl.org/?q=glol-intro-sistemas-distribuidos - 46k.

Rouaux, Martín. 2005. *Diseño y prototipo de middleware basado en CORBA para el BUS CAN.* [En línea] Octubre de 2005. [Citado el: 12 de Febrero de 2008.] http://www.fi.uba.ar/materias/7500/rouaux-tesisingenieriainformatica.pdf.

Salvachúa, Joaquín. *Aplicaciones y Servicios Web II (REST).* [En línea] [Citado el: 5 de Junio de 2008.] http://s3.amazonaws.com/ppt-download/scom5-ws-ii-24954.ppt.

Sánchez López, Emilio y Muñoz Ferrara, Javier. *El Protocolo CORBA en GNOME.* [En línea] [Citado el: 23 de Mayo de 2008.] http://www.aditel.org/jornadas/02/ponencias/bonobo/bonobo-doc/corba-gnome.html.

Schmidt, Douglas C. 2007. The ACE ORB (TAO). [En línea] 25 de Junio de 2007. [Citado el: 14 de Mayo de 2008.] http://www.cse.wustl.edu/~schmidt/TAO.html.

The Adaptive Communication Environment (ACE). [En línea] 25 de Junio de 2007. [Citado el: 14 de Mayo de 2008.] http://www.cse.wustl.edu/~schmidt/ACE.html..

Signes Andreu, Juan Miguel. 2005. *Integración y middleware.* [En línea] 10 de Mayo de 2005. [Citado el: 12 de Febrero de 2008.]

Squeak Swiki. 2006. *CORBA.* [En línea] 16 de Enero de 2006. [Citado el: 6 de Mayo de 2008.] http://wiki.squeak.org/squeak/2519 - 9k.

The Server Labs. *Integración al Servicio de la Empresa.* [En línea] [Citado el: 11 de Junio de 2008.] http://www.theserverlabs.com/folletos/SOA%20whitepaper.pdf.

Tolosa, Gabriel H. y Bordignon, Fernando R. A. 2006. Telématique. *GNUTWARE: Miidleware para soportar servicios distribuidos en redes compañero a compañero.* [En línea] 2006. [Citado el: 12 de Febrero de 2008.] http://redalyc.uaemex.mx/redalyc/pdf/784/78450103.pdf.

UPM. 2007. Morfeo-EzWeb. *Protocolos de comunicación para transmitir el contexto.* [En línea] 19 de

Noviembre de 2007. [Citado el: 5 de Junio de 2008.] http://forge.morfeo-project.org/wiki/index.php/D4.1.4_Protocolos_de_comunicaci%C3%B3n_para_transmitir_de_contexto_de_informaci%C3%B3n.

Vallejo Fernández, David. 2006. Oreto. *Documentación de ZeroC ICE.* [En línea] Diciembre de 2006. http://oreto.inf-cr.uclm.es.

Voglino, Laura. 2006. Blog Halo3. *Arquitectura Orientada a Servicios (SOA), La nueva Generación del Software.* [En línea] 29 de Septiembre de 2006. [Citado el: 3 de Marzo de 2008.]

Wikipedia. La enciclopedia libre. *Representational State Transfer.* [En línea] 1 de Junio de 2008. [Citado el: 5 de Junio de 2008.] http://es.wikipedia.org/wiki/Representational_State_Transfer.

ZeroC Inc. 2008. ZeroC. *The Internet Communications Engine.* [En línea] 2008. [Citado el: 22 de Mayo de 2008.] http://www.zeroc.com/ice.html.

ANEXOS

Anexo I

Matriz de decisión elaborada para la evaluación de tecnologías candidatas para implementar un prototipo funcional de un Middleware que mejore las condiciones del Middleware del SCADA "Guardián del Alba". Esta matriz permite la toma de decisiones durante la fase de desarrollo de la arquitectura de la aplicación. Los elementos como, criterios, opciones, importancia y puntajes fueron analizados durante el desarrollo del trabajo.

¿Como calificar una opción?

Calificación	Descripción
0	No Aceptable
1	Poco Aceptable
2	Aceptable
3	Sobresaliente
4	Exelente

Tabla 14: Descripción de la calificación.

Evaluación de las alternativas.

		ALTERNATIVAS			
Modelo de Decision		TAO		ICE	
Criterios	Importancia	Calificacion	Puntaje	Calificacion	Puntaje
Facilidad de implementación.	5	1	5	2	10
Documentación disponible y a la alcance del personal	5	1	5	2	10
Desarrollo de aplicaciones distribuidas.	15	4	60	4	60
Capacidad de tiempo real.	10	4	40	4	40
Consumo de recursos.	10	2	20	3	30
Soporte de los desarrolladores de las tecnologías.	10	4	40	3	30
Transmisión de un elevado número de variables(50000)	15	3	45	4	60
Persistencia a las conexiones.	10	2	20	4	40
Soporte de mecanismos de Redundancia.	10	1	10	3	30
Soporte de mecanismos de Seguridad.	10	2	20	3	30
Total	100	24	265	32	340

Tabla 15: Modelo de decisión de la tecnología más apropiada.

Puntaje = Valor * Importancia

Importancia = Rango entre 1-100

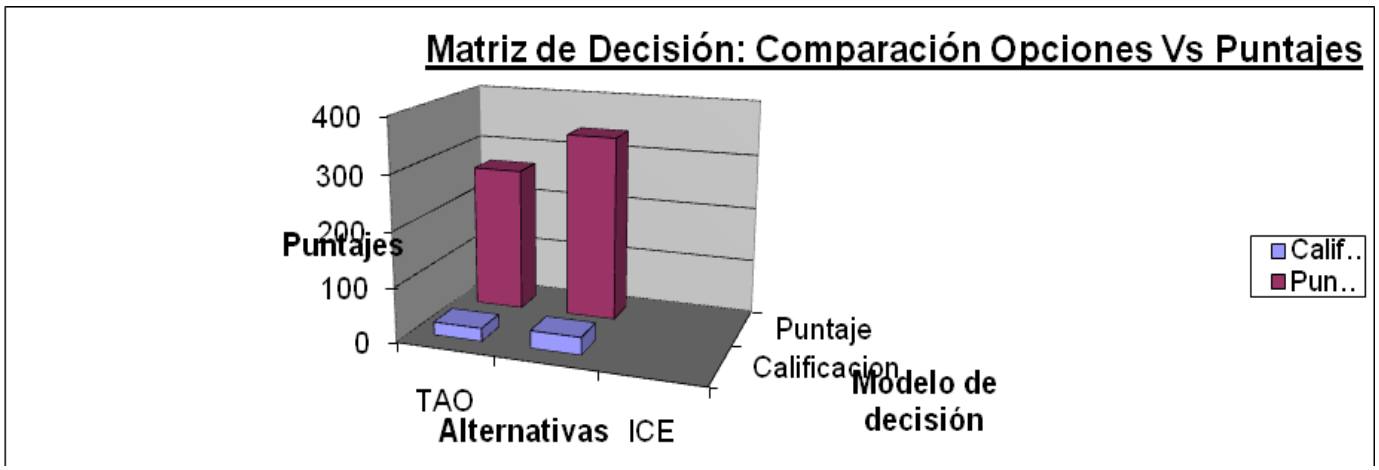

Figura 33: Gráfico de comparación entre tecnologías Ice y TAO.

Anexo II

Pruebas de concepto realizadas a las tecnologías ACE+TAO y ICE en el envío y recepción de puntos de forma local y remotas para valorar el desempeño de ambas tecnologías en cuanto a su rendimiento.

Resultados locales de Ice de Zeroc

Puntos	Tiempos (ms)			Memoria (Mb)			Procesador (%)		
	Min	Max	Media	Min	Max	Media	Min	Max	Media
1000	49	103	68,8	9,82031	9,92969	9,86328	36	50	41,6
5000	374	516	441,6	9,93359	10,2383	10,013282	54	60	56,4
10000	645	1170	943	9,77344	10,8594	10,090624	50	63	57,6
15000	1252	1525	1349,6	9,94922	12,3398	10,437492	54	60	56,6
20000	1705	2274	1992	9,77734	10,4414	10,017192	55	66	59,4
25000	1649	2313	2021,2	10,0586	10,6094	10,35626	60	77	65,4
30000	1988	3114	2567,4	9,83203	12,1016	11,030466	54	61	59
35000	2908	3584	3250,4	10,0391	11,8086	10,59064	54	70	63,2
40000	3248	4068	3660,6	9,88672	13,082	10,699994	60	70	64,4
45000	3385	4599	3849,8	9,99219	13,7148	11,771858	60	66	61,8
50000	4485	5270	4781	9,76172	9,92969	9,868752	60	66	62,2
55000	4312	6018	5189,4	10,0586	13,668	10,84844	60	70	64,4
60000	5038	5463	5286,4	10,1367	10,3242	10,22422	60	70	65,6

Tabla 16: Resultado de las pruebas locales de Ice.

Resultados locales de TAO

Puntos	Tiempos(ms)			Memoria(Mb)			Procesador(%)		
	Min	Max	Media	Min	Max	Media	Min	Max	Media
1000	458	500	487,4	9,92969	12,9883	11,546118	27	44	37,4
5000	2350	2598	2483,6	9,9375	13,0859	11,57032	45	60	49
10000	4931	5060	4994,2	9,96	13,1289	11,58342	43	50	46,6
15000	7154	7527	7432,2	9,9375	13,1328	11,6	50	54	50,8
20000	9338	9589	9467,4	10,5039	13,4219	11,9164	45	66	53,2
25000	12407	12640	12528,6	9,9375	13,1211	11,65236	50	60	53,8
30000	13976	14218	14104,4	9,94141	13,0469	11,585162	50	55	51,8
35000	16282	16622	16467,8	9,9375	12,9922	11,55158	50	70	56,4
40000	19740	20362	19970	10,0195	13,1641	11,61484	45	55	52,6
45000	22304	22679	22416,2	10,0039	13,0898	11,5867	45	63	54,6
50000	23271	23685	23494	10,0156	13,168	11,61562	50	60	55,6
55000	25768	27540	27082,6	10,0977	13,2148	11,71248	50	60	54,6
60000	27879	28520	28174	9,94141	13,8125	11,863282	50	80	59,8

Tabla 17: Resultado de las pruebas locales de TAO.

Resultados de las pruebas remotas de ICE de Zeroc

Puntos	Tiempos (ms)			Memoria (Mb)			Procesador (%)		
	Min	Max	Media	Min	Max	Media	Min	Max	Media
1000	87	137	116,8	9,71094	9,75	9,724218	33	50	44,8
5000	625	901	779,2	9,78516	10,1836	9,959368	45	50	48
10000	1100	1534	1381,6	9,71484	9,73438	9,728908	45	66	56,2
15000	1857	2367	2081,4	9,72	10,1445	9,843212	53	60	56,4
20000	2526	2842	2678,6	10,3867	10,7969	10,5539	60	66	61,2
25000	3326	3537	3459,6	9,77734	10,3984	9,906242	60	70	63,2
30000	3967	4387	4175	9,74609	11,5078	10,3039	60	66	62,4
35000	4543	4984	4745,4	11,5977	11,9531	11,79922	60	66	62,4
40000	5527	5891	5721,8	9,74609	10,6484	10,158578	60	70	63,2
45000	6234	6409	6354,8	11,2461	12,1602	11,64298	58	66	60,8
50000	7077	7900	7307,8	10,5078	12,832	11,6625	60	70	64,4
55000	7413	7954	7627,8	12,3711	13,7773	12,97576	60	66	63,6
60000	7813	8257	8034,4	12,1797	13,0469	12,44298	60	66	64,8

Tabla 18: Resultado de las pruebas remotas de Ice.

Resultados Remotos de TAO

Puntos	Tiempos (ms)			Memoria (Mb)			Procesador (%)		
	Min	Max	Media	Min	Max	Media	Min	Max	Media
1000	1272	1505	1370	10	11,7461	10,86876	23	30	26,4
5000	6202	6716	6399,6	10,0078	11,7461	10,86954	30	40	35,8
10000	12137	12600	12276	10,0117	11,7461	10,8711	40	50	42,2
15000	17139	18733	17899	10,0078	11,2773	10,7656	34	44	38
20000	21715	24794	23781	10,0078	10,8945	10,51326	40	55	44
25000	25300	30353	28207	10,0078	11,7461	10,86796	44	50	46,8
30000	31055	45192	37085	10,0039	11,2812	10,7156	36	50	45,4
35000	49954	54227	52493	10,0078	11,3164	10,72266	36	50	42,4
40000	49725	65866	59556	11,6914	13,2383	12,43594	40	50	44,8
45000	74728	76360	75431	10,043	11,7422	10,8789	40	45	42
50000	82285	85380	84296	10,0039	11,7383	10,86326	40	50	44,2
55000	91083	95440	93619	10,0078	11,7422	10,86718	36	45	40,2
60000	100782	103152	101867	10,0078	11,7422	10,86718	40	45	42,2

Tabla 19: Resultado de las pruebas remotas deTAO.

GLOSARIO DE TÉRMINOS

- **ActiveX:** Entorno de aplicaciones desarrollado por Microsoft. Incluye un gran número de componentes que inter-operan y enlazan diferentes aplicaciones en una computadora o red de computadoras.
- **ALBA:** Alternativa Bolivariana para las Américas.
- **API (Application Programming Interface):** Conjunto de rutinas y definiciones de funciones que abstraen los detalles de la implementación y hace más fácil el desarrollo de aplicaciones.
- **COM:** Component Object Model, es una plataforma de Microsoft para componentes de software desarrollada en 1993, es utilizada para permitir la comunicación entre procesos y la creación dinámica de objetos, en cualquier lenguaje de programación que soporte dicha tecnología.
- **CORBA:** Common Object Request Broker Architecture, es un estándar que establece una plataforma de desarrollo de sistemas distribuidos, facilitando la invocación de métodos remotos bajo el paradigma orientado a objetos.
- **DCOM:** Distributed Common Objects Model o Modelo de Objetos de Componentes Distribuidos, es una tecnología propietaria de Microsoft para desarrollar componentes software distribuidos sobre varios ordenadores y que se comunican entre sí.
- **Framework:** Es un conjunto de clases que encapsulan diseños abstractos de soluciones a un determinado números de problemas en relación. Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.
- **Hilos:** Los hilos son las distintas partes en las que un programa se puede dividir para ejecutarse en varias tareas simultáneas o casi simultáneas.
- **IDE:** Integrated Development Environment o Entorno de Desarrollo Integrado, es un software que provee facilidades a los desarrolladores en lenguaje de programación y compatibilidad con el sistema operativo o plataforma en que se trabaje.
- **LAN:** Local Area Network o Red de área local, es una red que conecta los ordenadores en un área relativamente pequeña y predeterminada (como una habitación, un edificio, o un conjunto de edificios).
- **Mensajes:** Mecanismo de comunicación entre los servicios. Los datos que componen el mismo están contenidos en formato XML Schema.

- **OLE:** Object Linking and Embedding, es un sistema de objeto distribuido y un protocolo desarrollado por Microsoft.
- **OMG:** Object Management Group o Grupo de Gestión de Objetos, es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos. Es una organización sin ánimo de lucro que promueve el uso de tecnología orientada a objetos mediante guías y especificaciones para las mismas.
- **ORB:** Object Request Broker, constituye el núcleo de la tecnología CORBA.
- **Plug-in:** Funcionalidad que se le puede agregar a un programa.
- **Software Libre:** Las cuatro reglas esenciales que deben cumplir las aplicaciones para ser consideradas como software libre son:
  - Libertad 0: Libertad de ejecutar el programa como quieras.
  - Libertad 1: Libertad de estudiar le código fuente y cambiarlo para realizar lo que desees.
  - Libertad 2: Libertad de realizar copias y distribuirlas cuando quieras.
  - Libertad 3: Libertad de distribuir o publicar versiones modificadas cuando desees.
- **TCP/IP:** Es un conjunto de protocolos de red en la que se basa Internet y que permiten la transmisión de datos entre redes de ordenadores. En ocasiones se le denomina conjunto de protocolos TCP/IP, en referencia a los dos protocolos que la componen: Protocolo de Control de Transmisión (TCP) y Protocolo de Internet (IP), que fueron los dos primeros en definirse, y que son los más utilizados de la familia.
- **UDP:** User Datagram Protocol, es un protocolo del nivel de transporte basado en el intercambio de datagramas, permitiendo el envío de estos a través de la red sin que se haya establecido previamente una conexión.
- **World Wide Web:** Es el sistema de información basado en hipertexto, cuya función es buscar y tener acceso a documentos a través de la red de forma que un usuario pueda acceder usando un navegador web.