Universidad de las Ciencias Informáticas

Facultad 5: Entornos Virtuales e Informática Industrial

"Sistema de Comunicación (Middleware) aplicable a diferentes entornos distribuidos."

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas.

Autor: Raicel Ger Peña

Tutor: Ing. Maikel Pérez Javier

Ciudad de la Habana

Julio 2008

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas, a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Autor:	Tutor:
Raicel Ger Peña	Ing. Maikel Pérez Javier
_____	_____

DATOS DE CONTACTO

Tutor:

Ing. Maikel Pérez Javier

Graduado de Ingeniería Informática.

e-mail: mperezj@uci.cu

Profesor instructor con dos años de experiencia docente en la Universidad de las Ciencias Informáticas (UCI) y dos 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).

"El deber del hombre virtuoso no está sólo en el egoísmo de cultivar la virtud en sí, sino que falta a su deber el que descansa mientras la virtud no haya triunfado entre los hombres."

José Martí.

DEDICATORIA

A la memoria de mis abuelos Betty y Jesús y de mi tía Rolindes.

Por todo el amor que me dieron y dejar en mí los deseos de ser cada día mejor.

A mis abuelos Lazara y Rogelio, mi mamá Celestina y mi hermana Dainy.

Por su amor y apoyo en todo momento y creer siempre en mí.

AGRADECIMIENTOS

A mis abuelos Lazara y Rogelio por estar siempre a mi lado y ser mi luz en la oscuridad, pero sobre todas las cosas por el amor que me han dado durante estos 24 años.

A mi mamá Celestina por su amor, su apoyo, su confianza y su sacrificio diario para que hoy pueda disfrutar de este gran sueño y poder regalárselo a ella.

A mi hermana Dainy por tenerme como ejemplo y quererme tanto a pesar de nuestras peleas.

A mi papá Ramón por abrirme las puertas de este maravilloso mundo de la informática.

A mis tíos Rogelito y Juan por preocuparse siempre de mí y por estar siempre cerca.

A mi tío Jesús y a mis primas Elizabeth y Geisis por su afecto y por permitirme seguir siendo su guajiro.

A mi tío Jorge por toda su ayuda, todas sus palabras de aliento y sus consejos.

… a toda mi familia por confiar siempre en mí y animarme a seguir adelante.

A mi tutor Maikel por su apoyo en el desarrollo de este trabajo, por guiarme y aclarar todas mis dudas.

A mi Decana Mayra y al Profe Tomás por su ejemplo y dedicación, por su afecto.

A Andrea por todos sus consejos y todo su cariño durante estos 5 cursos. A Ania, Mileydis y Maye.

A la profe Mileidy por su amistad, por su ayuda y apoyo incondicional.

A mis profesores de la infancia y la adolescencia, especialmente Florencia, Rafaela, Marta, Francisco, Glenys, Mariana, Angelita y Mary, por ayudarme con sus conocimientos a estar hoy aquí.

A todos mis profesores de la universidad por su aporte en mi preparación profesional.

… a todos los que contribuyeron en mi formación a lo largo de mi vida estudiantil.

A mis amigos de siempre Onisleydis, Eyleén y Rubén, que a pesar de la distancia seguirán estando cerca.

A Ocilia por haber sido tan especial. A Olga Lidia, Lala, Gallega, Idolka, Yami y Carmen por su aprecio.

A mis compañeros de aula y de apartamento por su compañía estos 5 años.

A mis amigos de la FEU por ser tan especiales y por tantos buenos momentos juntos.

A Aneyvis, Yela, Yari y Dailín por ser mi familia de la UCI.

… a todos los que de una forma u otra me han hecho el infinito regalo de su amistad.

A la Revolución Cubana y a Fidel por permitirme formar parte de este gran proyecto que es la UCI.

A Dios por darme la fe para seguir, la esperanza para confiar en el futuro y su amor para guiarme.

RESUMEN

El presente documento refleja una investigación, realizada a partir de la necesidad de contar con un sistema de comunicación (middleware) que por sus propiedades pueda ser utilizado en cualquier entorno, como pueden ser: sistemas industriales, redes WAN, aplicaciones multimedias, aplicaciones bancarias entre otros. De esta forma se analizan diferentes tecnologías relacionadas con la implementación de este tipo de software así como diferentes arquitecturas, que por sus principios son muy utilizadas para resolver los problemas de la distribución en diversos ambientes. Orientado al Polo de Automatización de la Facultad 5, este trabajo se plantea proponer las características que debe tener un middleware genérico. Para cumplir este propósito luego de hacer dicha propuesta se describen cada una de las características que garantizan sus funcionalidades y se selecciona una tecnología que sea capaz de dar respuesta a dichos requisitos además de una arquitectura que a partir de la utilización de servicios garantice la genericidad de dicho sistema. A partir de esta selección se proporcionan determinados elementos y modelos que muestran una vista de la arquitectura del middleware.

PALABRAS CLAVE

Sistemas Distribuidos, Middleware, Genericidad, Servicios.

ÍNDICE DE CONTENIDOS

INTRODUCCIÓN

Desde el Triunfo de la Revolución Cubana el 1ro de Enero de 1959 en Cuba se han desarrollado diversos programas en diferentes sectores del país, todos con un mismo fin, el de construir una sociedad culta cuya base principal sea el aporte de todos a la formación de los demás. Dentro de estos programas la dirección de la Revolución le ha prestado singular atención al sector de la ciencia el cual ha sido digno ejemplo de cuanto pueden aportar los recursos humanos en el desarrollo de una sociedad justa tanto desde el punto de vista económico como social.

Es por este motivo que desde muy temprano diferentes instituciones del país comienzan a dar los primeros pasos en los temas relacionados con la informática, el más importante de ellos, la construcción de la primera computadora cubana, la CID-201, el 18 de Abril de 1970 con el fin de utilizarse en la industria azucarera. Desafortunadamente para nosotros el período especial hizo que mucho de lo que se había adelantado en la rama de la informática se quedara paralizado y no fue hasta 1996 que comienza nuevamente a fomentarse en nuestra pequeña isla el desarrollo de la computación que posteriormente fue revitalizado con la Batalla de Ideas.

Algunas de las acciones llevadas a cabo con este fin fueron la creación del Ministerio de Informática y Comunicaciones, el programa de los Joven Club de Computación, la introducción de la computación en todos los niveles de la educación y el desarrollo de Multimedias Educativas para la enseñanza primaria y secundaria, el impulso dado a los Politécnicos de Informática y la creación de la carrera de Ingeniería Informática en todas las provincias por sólo mencionar algunos. Todo esto gracias a la extraordinaria visión de nuestro Comandante en Jefe Fidel Castro Ruz, quién desde la década del 70' dijera refiriéndose a la informática *"...Somos un país sin recursos naturales, pero tenemos un recurso muy importante, la inteligencia del cubano, que tenemos que desarrollarla, la Computación logra eso y estoy convencido de que los cubanos tenemos una inteligencia especial para dominar la Computación..."*

Es esta idea uno de los motivos que lo impulsaron, en el año 2002, a crear la Universidad de las Ciencias Informáticas dónde se preparan estudiantes de todos las regiones del país como Ingenieros en Ciencias Informáticas y que tiene como objetivos la formación de profesionales altamente calificados en la rama de

la informática y además el de impulsar la Industria Cubana del Software. Este último objetivo se logra con la vinculación de los estudiantes a proyectos productivos de desarrollo de software reales destinados al mercado nacional e internacional.

Para lograr la organización de este tipo de funcionamiento cada una de las Facultades cuenta con uno o varios polos productivos relacionados, cada uno de ellos, con diferentes áreas de desarrollo de la informática. La Facultad 5, entre otras áreas, trabaja los temas relacionados con la automatización de procesos. Una de las acciones que motivaron a la facultad a trabajar en esta rama fue la creación de un Grupo de Investigación integrado por profesores de la Universidad y coordinado por profesores de la Facultad. Pero sin dudas, el hecho que marcó el génesis del Polo de Automatización fue la asignación a la facultad, a mediados del año 2006, del proyecto de desarrollo SCADA (Sistema de Supervisión, Control y Adquisición de Datos) para PDVSA. En este polo actualmente se desarrollan otros proyectos además del SCADA, un ejemplo de ellos es el proyecto Revolución Energética para la Unión Nacional Eléctrica (UNE).

Estos sistemas generalmente se encuentran distribuidos geográficamente y están compuestos por varios componentes tanto de hardware como de software que se encuentran en varios ordenadores conectados en red y cada uno de estos componentes, tienen una función específica dentro del sistema. Estos elementos se comunican y coordinan sus acciones mediante la transmisión de mensajes en función de lograr un objetivo global. El trabajo con ellos en ocasiones resulta muy complicado debido a que no necesariamente los elementos que lo componen deben estar soportados por una misma plataforma, utilizan el mismo lenguaje de programación o el mismo protocolo de comunicación por lo que el medio de comunicación que utilice debe ser capaz de resolver este tipo de problemas para que el envío y recepción de los datos que cada elemento necesita para trabajar no se perjudique con un retraso, una pérdida o cualquier otro motivo que impida su correcto funcionamiento.

Los software que se encargan de las comunicaciones en los sistemas distribuidos se conocen como “middleware” y su uso es muy amplio a nivel mundial. Los middleware no son más que software de conectividad que ofrecen un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Dependiendo del problema a resolver y de las funciones necesarias, serán útiles diferentes tipo de servicios de middleware. Estas funciones les son asignadas

durante su desarrollo y dependen mucho de la tecnología que se utilice por lo que es muy importante realizar una correcta selección de la misma para que estos puedan satisfacer los servicios que debe brindarle al sistema en el cual van a ser acoplados.

Nuestra facultad se prepara para enfrentar nuevos proyectos de desarrollo de software en cualquier momento, además una de las líneas fundamentales de la misma es continuar desarrollando sistemas automatizados dentro del Polo de Automática, que puedan ser utilizados en diferentes ramas de la industria cubana o para la exportación y de esta forma continuar apoyando el progreso de la economía nacional.

Para enfrentar este reto es importante contar con un sistema de comunicación entre los entes distribuidos de las aplicaciones que permita dos cosas fundamentalmente: primero, cambiar la capa de la tecnología sin incurrir en grandes cambios y sin afectar la interfaz con el cliente y segundo, agregar, modificar y eliminar servicios en dependencia del ambiente en que se va a implantar y las necesidades del cliente. Esto se debe a que no todas las tecnologías que se utilizan son middleware en sí, que permiten configurarlo, y aunque así fuera se tendría que implementar nuevas interfaces para poder obtener un software robusto y fiable, es decir, por muy genérica que sea una tecnología no puede resolver el problema por sí sola sino que se tienen que definir e implementar servicios y funcionalidades.

En la facultad ya se desarrolló un middleware para el SCADA de PDVSA que cumple con la primera de las características que mencionamos anteriormente, pero no con la segunda debido a que las interfaces están orientadas a SCADAs, de forma tal que si se quisiera utilizar en un sistema para un banco u otro se tendrían que realizar grandes cambios en él pues en un banco no se envían puntos, alarmas, comandos etc. Sería mucho más genérico si se pensara en enviar un metadato en vez de puntos y alarmas y de esta forma daba la posibilidad de utilizarlo en otros sistemas diferentes de un SCADA, pero aún así no está orientado a servicios lo que limita mucho la genericidad.

Todo lo anteriormente expuesto es lo que ha motivado la investigación sobre este tipo de software en nuestra facultad de forma tal que a través de su estudio podamos dar respuesta al siguiente **Problema Científico:**

¿Qué características debe soportar un Sistema de Comunicación (Middleware) para que pueda ser integrado en cualquier ambiente distribuido?

Para dar respuesta a esta pregunta se plantea el siguiente **Objetivo:**

Constatar las características, tecnologías y arquitectura que permitan desarrollar un sistema de comunicación de propósito general.

Es por este motivo que se ha definido como **Objeto de Estudio** de la presente investigación a los sistemas de comunicación (middleware) y el **Campo de Acción** lo constituyen las características y funcionalidades de los middleware así como las tecnologías y arquitecturas que permiten desarrollarlos.

Por tanto las **Preguntas Científicas** que guiarán la investigación, atendiendo al problema enunciado anteriormente son:

1. ¿Cuáles son las características que debe tener un middleware para que pueda utilizarse en cualquier ambiente distribuido?

2. ¿Qué tecnología es la más adecuada para dar respuesta a las características definidas?

3. ¿Qué arquitectura de software permite la genericidad en los sistemas de comunicaciones de aplicaciones distribuidas?

Además durante la investigación y con el propósito de cumplir el objetivo del trabajo de diploma se plantean las siguientes **Tareas Investigativas:**

1. Desarrollar la fundamentación teórica del tema en interés dónde se aborden los conceptos de sistemas distribuidos y middleware.

2. Realizar una revisión y evaluación de las tecnologías existentes que permiten desarrollar middleware, tales como CORBA, ACE+TAO, ORBit, DDS, OpenDDS, ICE, SOA y REST.

3. Especificar cuáles deben ser las características que debe soportar un Middleware para que sea aplicable a diferentes ambientes distribuidos.

4. Identificar los requisitos que satisfacen las características que se definan durante el cumplimiento de la tarea anterior.

5. Mostrar una vista lógica de la arquitectura.

Es importante destacar que para el cumplimiento de las tareas científicas se emplearán diferentes **Métodos y Técnicas** que nos facilitarán la búsqueda y el procesamiento de la información, estos son:

Teóricos:

- Histórico-Lógico: Empleado durante la investigación de los antecedentes y las tendencias actuales referidas a los sistemas distribuidos pero específicamente a los middleware, además en la profundización de los conceptos, términos y vocabulario propio del objeto de estudio y el campo de acción.

- Analítico-Sintético: Durante la definición de los requerimientos y funcionalidades básicas que le permitan al middleware utilizarse de forma genérica.

- Inductivo-Deductivo: En la revisión, estudio y justificación de la tecnología seleccionada para desarrollar el middleware pues por medio de análisis individuales de las tecnologías se llegará a una propuesta general de características que se utilizarán en la inferencia de la tecnología más adecuada.

- Modelación: Utilizado durante la elaboración de los modelos encargados de mostrar la vista de la arquitectura que se utilizará para el desarrollo del middleware.

Empíricos:

- Entrevistas: Se aplica al Ing. Maikel Pérez Javier sobre diferentes temas relacionados con la investigación y por su trabajo en la elaboración del middleware del SCADA de PDVSA. Además se emplea también en otros estudiantes integrantes del proyecto mencionado anteriormente.

Por último se muestra como queda estructurado este documento:

En el **Capítulo 1** se realiza el estudio del estado del arte y la fundamentación teórica del tema que compete. Se muestran los conceptos más utilizados a nivel mundial de sistemas distribuidos y middleware. Se hace un recorrido por la evolución histórica de estos últimos así como de los diferentes tipos existentes.

En el **Capítulo 2** se hace alusión a las tecnologías existentes en la actualidad que deben tomarse en consideración a la hora de seleccionar aquella que puede ser utilizada para desarrollar el middleware así como las arquitecturas candidatas.

En el **Capítulo 3** se plantea la propuesta de solución compuesta por las características del middleware así como la tecnología seleccionada y la arquitectura que va a utilizarse. En este capítulo se muestra además la vista de la arquitectura que se empleará en el desarrollo del middleware genérico.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

Introducción.

La informática se ha convertido en una ciencia creciente en los últimos años, uno de los factores que han influido en este crecimiento es el desarrollo de los ordenadores que desde finales de los años 70 han sido objeto de las más grandes transformaciones. Estos equipos, que en esa época solo podían realizar funciones muy limitadas y eran de uso exclusivo de algunas empresas, hoy en día se han convertido en partes indispensables de las personas, ya sean portátiles o personales.

Estos cambios se deben principalmente a dos fenómenos principalmente: primero, el desarrollo de los microprocesadores que permitían con menos costo y menor tamaño aumentar las capacidades de los mismos; segundo, el desarrollo de las redes de área local y las comunicaciones que permiten conectar varios computadores lo que hace posible la transferencia de datos a altas velocidades.

A partir de estos avances es que comienza a utilizarse el termino de sistemas distribuidos que ha alcanzado una amplia popularidad y que tiene como campo de estudio las redes, ya sean corporativas o empresariales, ejemplo de ellas son las telefónicas o internet. Pero un elemento principal dentro del estudio de estos sistemas son los mecanismos de comunicación que utilizan. Sobre todos estos temas se estará profundizando en el desarrollo de este capítulo.

1.1 - Sistemas Distribuidos.

Por sistemas distribuidos puede entenderse:

> "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)

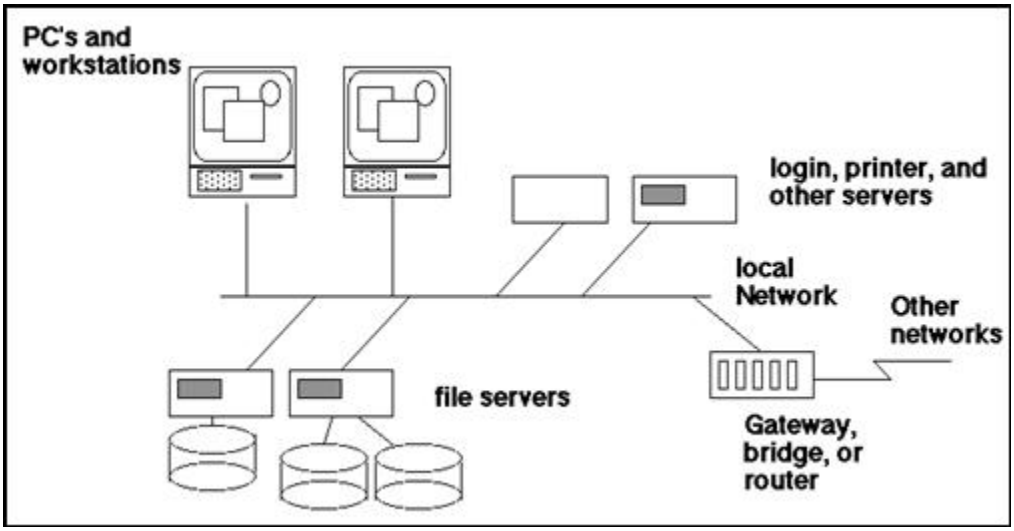

Figura # 1.1: Ejemplo de un Sistema Distribuido Simple.

Podemos resumir que los sistemas distribuidos pueden definirse como aplicaciones que se ejecutan en diferentes estaciones geográficamente distribuidas y que pueden estar bien distantes unas de otras. Estas computadoras se encuentran interconectadas por una red. Los procesos de la aplicación pueden estar ejecutándose en diferentes computadoras de dicha red junto a un constante intercambio de mensajes que permite la transferencia de información. Todo esto garantiza que todos los componentes del sistema funcionen de forma correcta y de forma cooperativa para alcanzar la realización de un objetivo común.

1.1.1 – Características.

Son varias las características que se pueden numerar para describir un sistema distribuido, por ejemplo las relacionadas con la distribución del trabajo en las cuales las computadoras se reparten este último para un mejor funcionamiento, o las que definen las diferentes funcionalidades de los componentes de la red. Podemos mencionar además las económicas, las cuales nos permiten financiar los elementos del sistema, o las físicas que se encargan de la distribución física de este. Ahora bien, en el año 1994 Coulouris estableció seis características principales, responsables de la utilidad de los sistemas distribuidos.

A continuación se presenta un resumen de cada una de ellas para un mejor entendimiento de los sistemas distribuidos, estas son:

- Compartición de Recursos.

El término recurso se utiliza para caracterizar todas las entidades que se comparten dentro de un sistema distribuido ya sean de hardware o software. Los recursos en un sistema distribuido están físicamente encapsulados en una de las computadoras y sólo pueden ser accedidos por otras computadoras mediante la red. Para que la compartición de recursos sea efectiva, ésta debe ser manejada por un programa que ofrezca un interfaz de comunicación permitiendo que el recurso sea accedido, manipulado y actualizado de una manera fiable y consistente. Todo esto puede realizarse a partir de un gestor de recursos que no es más que un módulo software que maneja un conjunto de recursos de un tipo en particular. Cada tipo de recurso requiere algunas políticas y métodos específicos junto con requisitos comunes para todos ellos.

- Apertura:

Esta característica se determina primariamente por el grado hacia que nuevos servicios de compartición de recursos se puedan agregar sin perjudicar o duplicar los existentes. Básicamente los sistemas distribuidos cumplen una serie de características:

1. Las interfaces software claves del sistema están claramente especificadas y se ponen a disposición de los desarrolladores. En una palabra, los interfaces se hacen públicas.
2. Los sistemas distribuidos abiertos se basan en la provisión de un mecanismo uniforme de comunicación entre procesos e interfaces publicados para acceder a recursos compartidos.
3. Los sistemas distribuidos abiertos pueden construirse a partir de hardware y software heterogéneo, posiblemente proveniente de vendedores diferentes. Pero la conformidad de cada componente con el estándar publicado debe ser cuidadosamente comprobada y certificada si se quiere evitar tener problemas de integración.

- Concurrencia:

En los sistemas distribuidos hay muchas maquinas, cada una con uno o mas procesadores centrales. Es decir, si hay M ordenadores en un sistema distribuido con un procesador central cada una entonces puede tener hasta M procesos ejecutándose en paralelo. En un sistema distribuido que está basado en el modelo de compartición de recursos, la posibilidad de ejecución paralela ocurre por dos razones:

1. Muchos usuarios interactúan simultáneamente con programas de aplicación.
2. Muchos procesos servidores se ejecutan concurrentemente, cada uno respondiendo a diferentes peticiones de los procesos clientes.

- Escalabilidad:

Tanto el software de sistema como el de aplicación no deberían cambiar cuando la escala del sistema se incrementa. El diseño del sistema debe reconocer explícitamente la necesidad de escalabilidad o de lo contrario aparecerán serias limitaciones. La demanda de escalabilidad en los sistemas distribuidos ha conducido a una filosofía de diseño en que cualquier recurso simple -hardware o software- puede

extenderse para proporcionar servicio a tantos usuarios como se quiera. Esto significa que si la demanda de un recurso crece, debería ser posible extender el sistema para darle servicio.

Cuando el tamaño y complejidad de las redes de ordenadores crece, es un objetivo primordial diseñar software de sistema distribuido que seguirá siendo eficiente y útil con esas nuevas configuraciones de la red. Resumiendo, el trabajo necesario para procesar una petición simple para acceder a un recurso compartido debería ser prácticamente independiente del tamaño de la red. Las técnicas necesarias para conseguir estos objetivos incluyen el uso de datos replicados, la técnica asociada de caching, y el uso de múltiples servidores para manejar ciertas tareas, aprovechando la concurrencia para permitir una mayor productividad.

- Tolerancia a Fallos:

El diseño de sistemas tolerantes a fallos se basa en dos cuestiones, complementarias entre sí: Redundancia hardware (uso de componentes redundantes) y recuperación del software (diseño de programas que sean capaces de recuperarse de los fallos). En los sistemas distribuidos la redundancia puede plantearse en un grado más fino que el hardware, pueden replicarse los servidores individuales que son esenciales para la operación continuada de aplicaciones criticas. La recuperación del software tiene relación con el diseño de software que sea capaz de recuperar el estado de los datos permanentes antes de que se produjera el fallo.

Los sistemas distribuidos también proveen un alto grado de disponibilidad en la vertiente de fallos hardware. La disponibilidad de un sistema es una medida de la proporción de tiempo que esta disponible para su uso. Cuando uno de los componentes de un sistema distribuido falla, solo se ve afectado el trabajo que estaba realizando el componente averiado, de ahí que un usuario podría desplazarse a otra estación de trabajo y un proceso servidor podría ejecutarse en otra maquina.

- Transparencia:

La transparencia se define como la ocultación, al usuario y al programador de aplicaciones, de la separación de los componentes de un sistema distribuido, de manera que el sistema se percibe como un

todo, en vez de una colección de componentes independientes. La transparencia ejerce una gran influencia en el diseño del software del sistema.

El manual de referencia RM-ODP [ISO 1996a] identifica ocho formas de transparencia. Estas proveen un resumen útil de la motivación y metas de los sistemas distribuidos. Las transparencias definidas son:

1. Transparencia de Acceso: Permite el acceso a los objetos de información remotos de la misma forma que a los objetos de información locales.
2. Transparencia de Localización: Permite el acceso a los objetos de información sin conocimiento de su localización.
3. Transparencia de Concurrencia: Permite que varios procesos operen concurrentemente utilizando objetos de información compartidos y de forma que no exista interferencia entre ellos.
4. Transparencia de Replicación: Permite utilizar múltiples instancias de los objetos de información para incrementar la fiabilidad y las prestaciones sin que los usuarios o los programas de aplicación tengan por que conocer la existencia de las replicas.
5. Transparencia de Fallos: Permite a los usuarios y programas de aplicación completar sus tareas a pesar de la ocurrencia de fallos en el hardware o en el software.
6. Transparencia de Migración: Permite el movimiento de objetos de información dentro de un sistema sin afectar a los usuarios o a los programas de aplicación.
7. Transparencia de Prestaciones: Permite que el sistema sea reconfigurado para mejorar las prestaciones mientras la carga varía.
8. Transparencia de Escalado: Permite la expansión del sistema y de las aplicaciones sin cambiar la estructura del sistema o los algoritmos de la aplicación.

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.

1.1.2 – Ventajas de los Sistemas Distribuidos.

A la hora de desarrollar un sistema distribuido no solo debe conocerse su funcionamiento, comportamiento y estructura sino que para obtener un mayor beneficio de los mismos debe conocerse a profundidad las fortalezas y las debilidades de los mismos, para explotarlas o protegerlas según sea el caso, y de este modo obtener los mejores resultados. A continuación se presentan un conjunto de elementos que nos permitirá ahondar más en este tema. Pero para poder analizar las ventajas de los mismos debemos hacerlo teniendo en cuenta dos aspectos, el primero es con respecto a sistemas centralizados y el segundo con respecto a computadoras independientes para ver las mismas desde varios puntos de vista.

- Ventajas Respecto a Sistemas Centralizados:

1. Una de las principales ventajas que este tipo de sistema ofrece es en la economía pues es mucho más barato agregar al sistema nuevos servidores y clientes cuando es necesario aumentar la potencia de procesamiento del mismo.
2. Otra ventaja es el trabajo en conjunto pues permite que se de respuesta a determinada solicitud en mucho menos tiempo ya que las tareas se reparten entre los componentes del sistema para que se procese más rápido el trabajo y la responsabilidad no recae en un único componente.
3. Existe mayor confiabilidad pues al estar distribuida la carga de trabajo en diferentes máquinas la falla de una de ellas no afecta a las demás permitiendo que el sistema sobreviva como un todo.
4. Otra de estas ventajas es la capacidad de crecimiento incremental debido a que se pueden agregar procesadores al sistema incrementando su potencia de forma gradual en dependencia de sus necesidades.

- Ventajas Respecto a Computadoras Independientes:

1. Una de las ventajas más importantes con respecto a un único procesador es que se pueden compartir recursos, como programas, periféricos que pueden ser muy costosos y de esta forma se puede ahorrar y satisfacer necesidades de muchos usuarios a la vez.
2. Se logra una mejor comunicación entre las personas pues se pueden utilizar servicios como el correo o chat.
3. Existe mayor flexibilidad pues la carga de trabajo se reparte entre diferentes ordenadores.

1.1.3 – Desventajas de los Sistemas Distribuidos.

Pero no solo podemos fijar nuestra atención en los aspectos positivos que estos sistemas nos ofrecen sino que para una mejor utilización de los mismos debemos conocer cuales son sus dificultades para intentar trabajar sobre ellas y erradicarlas si queremos tener un producto verdaderamente con calidad.

- Desventajas de los Sistemas Distribuidos.

1. El problema más importante en este tipo de sistema es el software distribuido en cuanto a su diseño, implantación y uso, es por esto que la comunidad mundial no ha logrado unificar los criterios en cuanto a qué sistema operativo, lenguaje de programación y aplicaciones son adecuados para estos sistemas o cuanto deben saber los usuarios sobre la distribución, incluso muchos no han podido delimitar cuanto debe ser capaz de hacer el sistema y cuanto los usuarios; es por eso que esto constituye su problema fundamental.
2. Otro conflicto a resolver por estos sistemas tiene que ver con las redes de comunicación y está relacionado con la pérdida de mensajes o la saturación de la red.
3. Por último y no menos importante está el tema de la seguridad de los datos que son compartidos para el trabajo de los usuarios ya que por su importancia deben tener una vigilancia constante.

1.1.4 – Desafíos de los Sistemas Distribuidos.

Por todo lo anteriormente expuesto es que se han determinado una serie de retos o desafíos que este tipo de sistemas tiene y a los cuales se les dedica una sección en este trabajo para su conocimiento. Los retos son los siguientes:

Heterogeneidad de los Componentes: Se necesitan ciertos estándares a la hora de realizar la interconexión entre los componentes debido a la variedad de software o hardware de los mismos. Para esto se desarrollan los middleware que son elementos software que permiten una abstracción de la programación y el enmascaramiento de la heterogeneidad subyacente sobre las redes.

Extensibilidad: Esto significa que el sistema pueda extenderse o re-implementarse algunos elementos de él. Además que se puedan integrar los componentes desarrollados por diferentes personas.

Seguridad: Es uno de los aspectos que más preocupa a los usuarios y dentro de ella debe garantizarse la confidencialidad, la integridad y la disponibilidad.

Escalabilidad: Conservar la efectividad del sistema independientemente del aumento de recursos o usuarios.

Tratamiento de 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: Permitir compartir recursos a los usuarios a la vez.

Transparencia: Está relacionada con el ocultamiento de la separación de los componentes a los usuarios y los programadores.

1.1.5 – Aplicabilidad.

Podemos 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 podemos concluir con que los Sistemas Distribuidos abarcan una cantidad de aspectos considerables lo que determina que el desarrollo de los mismos sea muy complejo. Dentro de esos aspectos existen algunos que requieren un extremo cuidado a la hora de desarrollarse o implementarse. Este es un tema en el que se sigue investigando. Además podemos decir que uno de las factores más importantes de estos sistemas es la comunicación que debe existir entre sus componentes, esta se encuentra en el centro del funcionamiento de estos sistemas y de sus características depende su correcto funcionamiento, es por eso que en el siguiente epígrafe se pretende abordar los elementos fundamentales de los Subsistemas de Comunicación (Middleware) de los Sistemas Distribuidos.

1.2 – Middleware.

Como bien se pudo apreciar en el epígrafe anterior una parte muy importante de los Sistemas Distribuidos es su Subsistema de Comunicación. Él 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. Lo mencionado anteriormente es uno de los motivos por lo que

se propone abordar este tema en el presente trabajo. Pero como ya se habrá dado cuenta el lector, el segundo objetivos del trabajo de diploma es proponer la confección de un middleware con funcionalidades genéricas para que pueda utilizarse en diferentes ambientes.

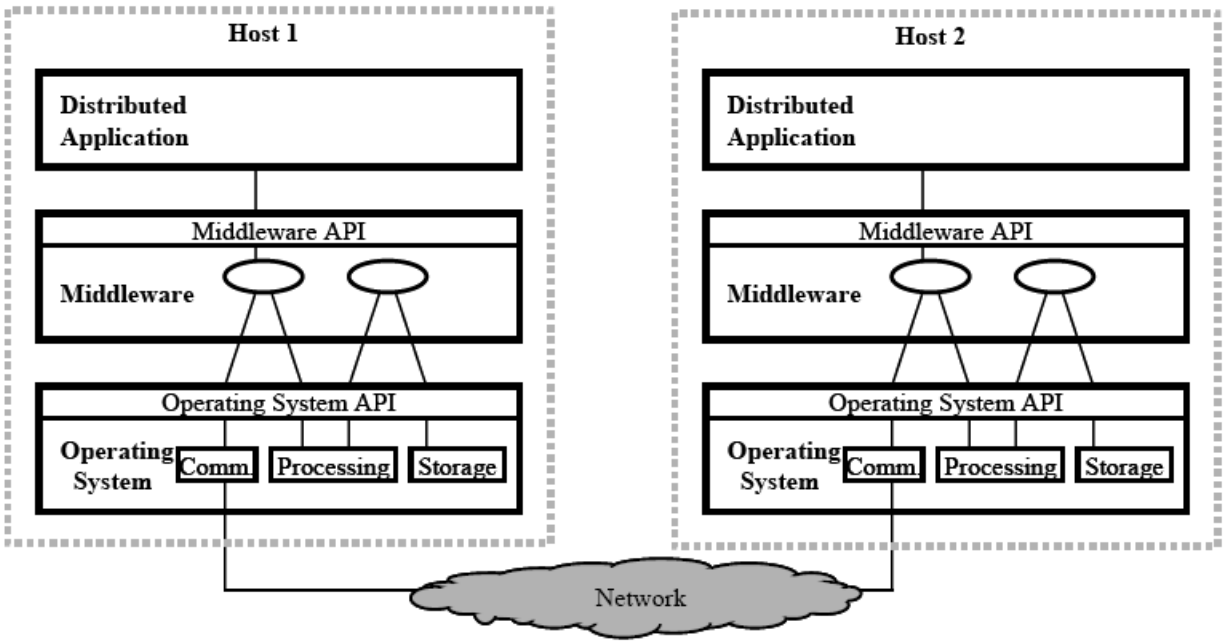

Figura # 1.2: La Capa del Middleware ubicada en contexto.

1.2.1 – Definición.

Luego de dar una pequeña introducción a lo que se estará abordando en el epígrafe nada mejor que ver algunos conceptos de middleware para conocer y entender mejor de que estamos hablando.

> Podemos decir que middleware es el conjunto de servicios que permiten a las aplicaciones distribuidas interoperar en redes LAN o WAN. Enmascara la complejidad del sistema tanto para los usuarios finales como para los desarrolladores de las aplicaciones, proporcionando el acceso transparente a los servicios que se encuentran a través de los recursos del sistema (computadoras, impresoras, módems, software, etc.).

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]

Middleware (elementos de en medio) es un término general para cualquier programación que sirve para unir o trabajar como puente entre dos programas separados, y que por lo general ya existen. La mensajería es un servicio común provisto por programas de middleware de tal manera que diferentes aplicaciones se puedan comunicar. [Daccach, José Camilo]

Middleware es una capa entre los sistemas operativos de redes y las aplicaciones que apunta a resolver la heterogeneidad y la distribución.

Middleware se refiere a una capa de software entre los servicios de la red y las aplicaciones, encargadas de proporcionar servicios como identificación, autenticación, autorización, directorios y movilidad. [López, Eric]

1.3.2 – Origen y Evolución.

Cómo la mayoría de los términos que se utilizan en el mundo de la informática el término Middleware ha aparecido de forma reciente en esta ciencia aunque podemos decir que se utilizó por vez primera en 1968, lo cual está registrado en el Reporte de la Conferencia de Ingeniería de Software NATO dónde fue empleado para nombrar al software mediador entre las aplicaciones y el sistema operativo. (NATO SOFTWARE COMITEE, 1969).

Posteriormente vuelve a retomarse el término en 1970 y en 1972 primero para emplearlo como: “software de fabricantes de computadoras, que ha sido confeccionado para las necesidades particulares de una instalación”; y posteriormente para nombrar a los programas realizadores de cambios de extensión y modificación en los sistemas operativos debido a los requerimientos solicitados por aplicaciones de complejidades únicas. (Gall, 2003)

La verdadera popularidad la alcanzó en la década del 80 ya que constituían la solución de como integrar las nuevas aplicaciones con los sistemas heredados y se fija su utilización ampliada a mediados de los noventa empleándose en casos donde se facilitaba la computación distribuida mediante conexiones de múltiples aplicaciones para crear una mucha mayor sobre una red. Esto hace que el middleware evolucione en un conjunto de paradigmas y servicios más abundantes y maduros, ofrecidos para facilitar y brindar manejabilidad al desarrollo de sistemas distribuidos. (“Encyclopedia of Distributed Computing”, Kluwer Academic Press, 2003) También se le han asignado otros nombres muy similares a los de nuestros días como sistemas operativos de red, sistemas operativos distribuidos, y entornos de computación distribuida. (David E., 2003)

Ahora bien, repasando la historia del middleware desde los años 80 se pueden distinguir tres categorías de productos en los que este ha evolucionado: estaciones de mensajería, motores de integración y buses de integración. A continuación se explican cada una de estas categorías para entender mejor la evolución de dicho software.

- Estaciones de Mensajería (EM):

Las EM proporcionan asistencia a los sistemas en lo referente al medio de comunicación liberando a las aplicaciones de algunos servicios como son mantener los parámetros de conexión de todos sus interlocutores y de gestionar directamente con ellos el envío y recepción de mensajes. Con una EM lo único que necesitan saber las aplicaciones es como intercambiar mensajes con ella y la EM asume el compromiso se hacer llegar los mensajes a su destino de una forma rápida y segura lo cual se especifica mediante parámetros de calidad y acuerdos de nivel de servicio lo cuales pueden ser expresados a través de un tiempo máximo de entrega o reintentar enviar el mensaje un número determinado de veces o

durante cierto tiempo si el sistema remoto no está disponible entre otros. Las EM proporcionan estos servicios mediante procedimientos basados en soluciones propietarias lo cual significa que el módulo de comunicación lo ha de proporcionar el proveedor de la EM.

Las EM suelen ocupar el centro de una topología en estrella en dónde las aplicaciones o sistemas se conectan a través de la EM en lugar de hacerlo directamente entre sí. Es aquí dónde radica el inconveniente principal de estas arquitecturas y es que las EM se convierten en punto único de fallo, pues de caer la misma se acaba la comunicación lo cual constituye un riesgo para muchos entornos y una forma de mitigarlo es instalarlas en equipos dotados de características de alta disponibilidad.

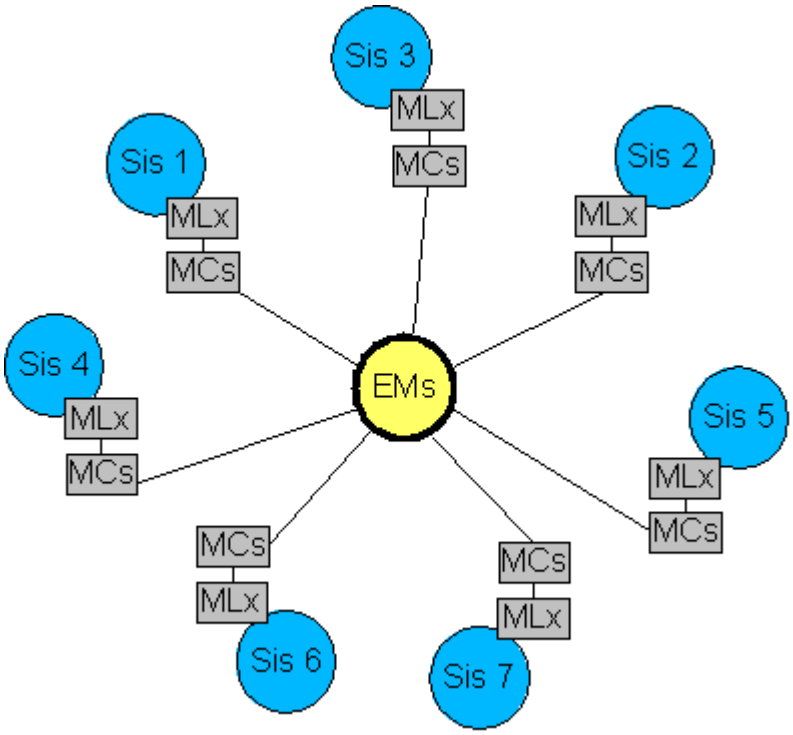

Figura # 1.3: Topología de Red de una Estación de Mensajería.

Normalmente las EM no interpretan el contenido del mensaje compuesto por el módulo del lenguaje de cada sistema sino que lo tratan como un todo que han de hacer llegar a su destino cumpliendo una serie de condiciones.

Las EM aparecieron en el mercado a finales de la década de los 80 y los diferentes proveedores las desarrollaron utilizando soluciones propietarias por tanto la interoperabilidad entre las EM de distintos fabricantes era nula o muy limitada pues cada proveedor tenía su propia forma de gestionar las comunicaciones a bajo nivel y de especificar las políticas de seguridad y calidad. [Signes Andreu, 2005]

- Motores de Integración (MInt):

Los MInt aparecieron a finales de los años 90 como una evolución de las EM. Los MInt heredan todas las características de las EM a las que agregan un nuevo tipo de funcionalidades, basadas en la interpretación y tratamiento de los mensajes. Entre los nuevos servicios se pueden destacar la traducción de mensajes y el enrutamiento basado en el contenido.

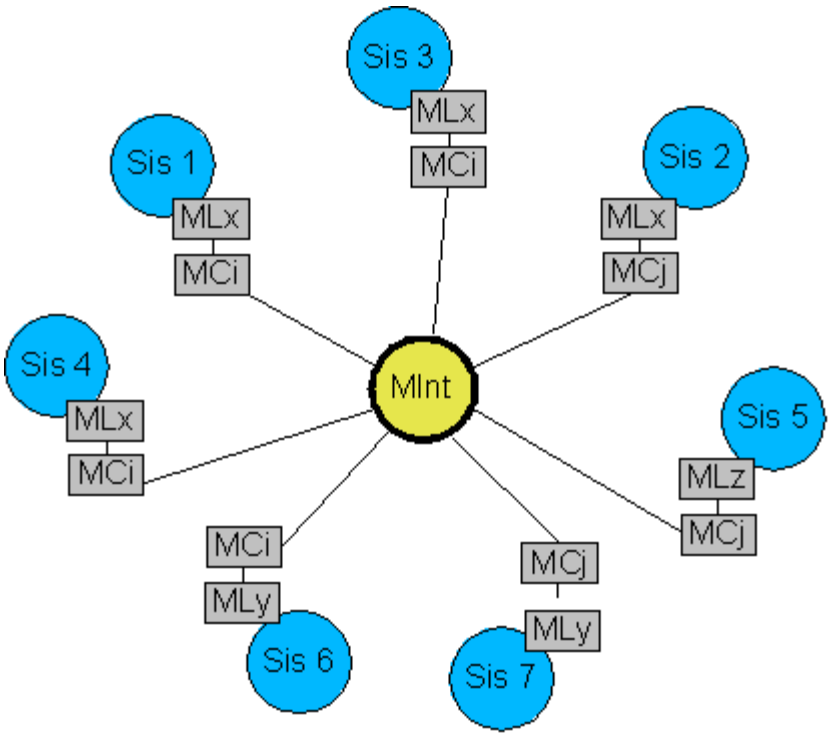

Figura # 1.4: Topología de Red de un Motor de Integración.

La traducción de mensajes es muy importante porque poner de acuerdo a todos los sistemas para que utilicen el mismo lenguaje, incluso la misma versión, puede ser un reto considerable o prácticamente imposible. A medida que vaya aumentando el proyecto de integración menor es el control que se tiene sobre los sistemas participantes y mayor la probabilidad de que el escenario sea multilingüe lo cual puede agravarse aun más teniendo en cuenta que la mayoría de los sistemas sólo son competentes con el manejo de un lenguaje. Por tanto los MInt ayudan a ordenar todo esto haciendo que las barreras lingüísticas no sean un obstáculo para las interacciones.

El enrutamiento de mensajes basado en su contenido permite implementar modelos de interacción más sofisticados en los cuales el destino de los mensajes no está predeterminado sino que se asigna de forma dinámica durante la ejecución en función de su contenido y otras circunstancias que se consideren relevantes como la disponibilidad y la carga de los sistemas de destino, la hora, el tráfico en la red entre otras.

Como se ha señalado anteriormente en el núcleo de todo MInt se encuentra una EM esto hace que además de heredar sus posibilidades también herede sus limitaciones y una de ellas es que su dominio de aplicación natural está limitado a una red de área local (LAN). [Signes Andreu, 2005]

- Buses de Interacción:

El concepto de bus de integración empresarial (o ESB, por Enterprise Service Bus) comenzó a manejarse a partir del año 2002. Este es un tipo de Middleware diseñado para superar las limitaciones de los MInt y responder a las necesidades de integración de las empresas a una escala superior a la considerada hasta ese momento. La medida de esa nueva escala la proporciona el concepto de empresa u organización extendida, es decir aquella con varias sedes, tal vez repartidas por distintos continentes, que establecen procesos de negocio con otras organizaciones sobre cuyos sistemas de información no tiene control.

Tenemos dos conceptos muy importantes a la hora de entender los ESB y estos son bus y servicio. Los ESB pueden estar físicamente distribuidos en distintos segmentos pero constituyen una única unidad lógica y todos los sistemas conectados comparten el mismo espacio de direccionamiento. Los parámetros

de calidad de servicio pueden ser aplicados a todo el bus a cada uno de los segmentos, pero siempre de la misma forma. Los servicios de integración están disponibles para cualquier sistema conectado al bus, no están físicamente ligados a ninguna ubicación y pueden replicarse en aras de la eficiencia y de la seguridad. Entre estos servicios están los de validación, transformación y traducción de mensajes, enrutamiento basado en el contenido, definición y control de procesos de negocio, autenticación, autorización y auditoría, monitorización, administración, y otros.

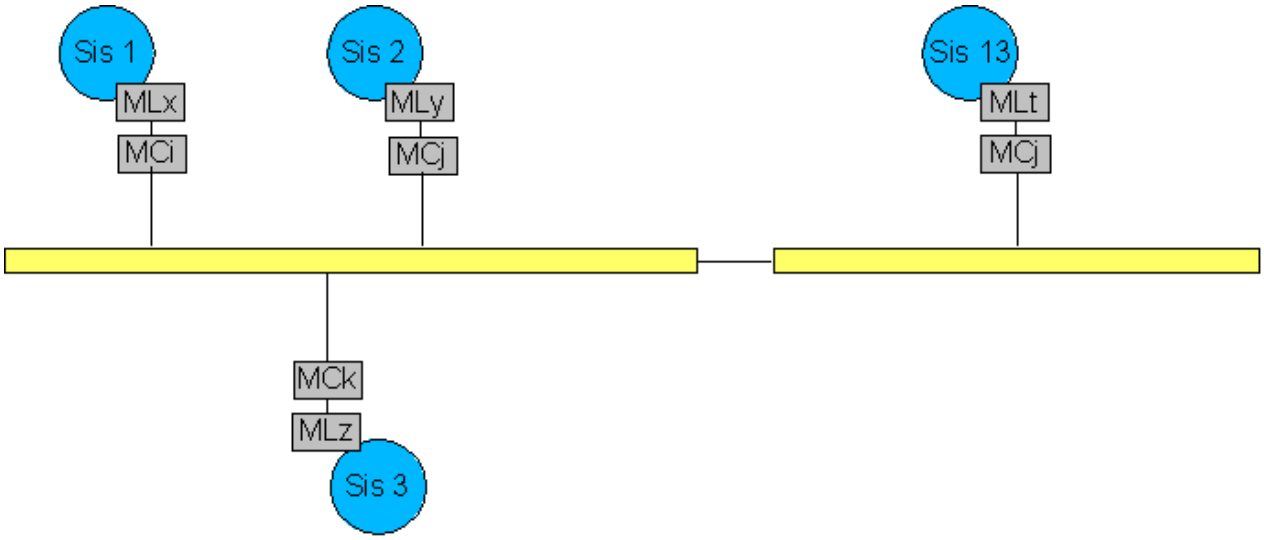

Figura # 1.5: Topología de Red de los Buses de Interacción.

En cuanto a los servicios todos los sistemas conectados al ESB son proveedores o consumidores de servicios dónde los ESB ofrecen el recubrimiento adecuado en caso de que algún sistema sea ajeno a este concepto. De esta forma los ESB hacen posible el despliegue de Arquitecturas Orientadas a Servicios (SOA) sin exigir que los sistemas participantes ofrezcan una interfaz basada en servicios.

La columna vertebral de todo ESB es un MOM aunque dicha tecnología siga siendo propietaria pero es muy robusta y está muy probada. Lo único que debe saber cada sistema participante es cómo enviar y recibir mensajes al ESB. A diferencia de lo que sucedía con las EM, los procedimientos se basan en estándares y pueden ser muy variados, desde un acceso directo a una base de datos al uso de servicios

web, pasando por protocolos propietarios o incluso el intercambio de ficheros. En cualquier caso el ESB es el que hace el esfuerzo de adaptarse a lo que el sistema en cuestión sea capaz de manejar. Para este propósito los ESB disponen de una amplia variedad de adaptadores: JDBC, FTP, SMTP, HTTP, RMI/IIOP, acceso a ficheros planos etc.

En un ESB la caída de un segmento no compromete el funcionamiento de los demás. Los ESB pueden configurarse de manera que existan servicios, rutas y segmentos alternativos, lo que junto a la ausencia de puntos de control y administración únicos hace que sean infraestructuras capaces de funcionar en régimen de alta disponibilidad.

Todos los servicios de integración que aportan los MInt se pueden encontrar en los ESB, pero más y mejor resueltos, en particular los que tienen que ver con los procesos de negocio. Algunos proveedores incluyen también una capa que permite diseñar interfaces de usuario, con lo que abren la puerta a un nuevo producto de integración que son las aplicaciones compuestas que se construyen combinando los servicios disponibles a lo largo del bus.

Debido a la cantidad de especificaciones utilizadas por los ESB y la competencia que existe entre las diferentes normas, ya sean avaladas por alguna organización u otras que se encuentran en su primera versión, se dificulta la interoperabilidad entre buses de distintos proveedores y la posibilidad de adquirir conectores proporcionados por terceros. En los próximos años cabe esperar que el panorama mejore y que esto redunde en un abaratamiento de los productos y en una mayor independencia de los proveedores. [Signes Andreu, 2005]

1.2.3 – Tipos de Middleware.

Es muy común ver que los middleware se agrupen en diferentes clasificaciones en dependencia de para que tipo de sistemas se desarrolle el mismo. Además pueden clasificarse a partir de su escalabilidad y su tolerancia a fallos aunque pueden existir muchos tipos de clasificaciones no relacionadas con la que aquí se muestra.

1. Distributed Tuples (DT): Es el Middleware para acceso a base de datos que permite desarrollar sistemas independientes del manejador de base de datos que lo soporta.

2. Remote Procedure Call (RPC): Es el Middleware diseñado como servicio síncrono para permitir la gestión remota de redes. Esconde las operaciones de envío y recepción bajo el aspecto de una llamada convencional a una rutina o procedimiento. Los RPC tienen la misma semántica que las llamadas a procedimientos ordinarios; es decir, se realiza la llamada y se pasa el control al procedimiento servidor; cuando éste devuelve el resultado, el cliente recupera el control. El software que soporta RPC debe ocuparse de tres tareas importantes: la interfaz del servicio, la búsqueda del servidor, y la gestión de comunicación.

3. Messaging Oriented Middleware (MOM): Es el Middleware orientado a mensajes; está diseñado para el servicio de mensajes con tecnología asíncrona. Permite el envío de mensajes entre aplicaciones, las aplicaciones sólo ponen y sacan mensajes de las colas, no se conectan. El cliente y el servidor pueden ejecutarse en diferentes tiempos (mensajes asíncronos), por lo que no necesariamente se requiere respuesta.

4. Distributed Object Middleware (DOM): Es el Middleware para tecnologías orientadas a objetos; los objetos piden servicio a otros objetos que se encuentran en la red. Se encarga de establecer comunicación entre los clientes y los objetos de forma transparente respecto a la distribución. Permite localizar a un objeto remoto dada una referencia a ese objeto. El núcleo de estos Middleware es el Object Request Broker (ORB).

4.1. Common Object Request Broker Architecture (CORBA): Es un modelo de soporte para la programación distribuida orientada a objetos. Hace posible que los objetos interactúen a través de lenguajes de programación, protocolos de comunicación y plataformas heterogéneas. Este modelo no especifica cómo hacer el soporte, sino qué debe hacer, basado en cinco aspectos de los sistemas distribuidos:

- Interface Definition Lenguaje (IDL): Permite la descripción de la interfaz que ofrece un objeto.

- Corba Services (Servicios CORBA): Complementan a los objetos que sirven para la construcción de aplicaciones.
- Corba Facilities (Facilidades CORBA): Cubren servicios de alto nivel, como interfaces, administración de sistemas y redes.
- Corba Domains (Interfaces de dominio CORBA): Proveen funcionalidad a usuarios finales en áreas de interés particular.
- General Inter-ORB Protocol (GIOP). Define los mensajes y el empaquetado de datos que se transmiten entre objetos. Además, define su implementación sobre otros protocolos.

4.2. Remote Method Invocation (RMI): Posee la misma finalidad que el RPC: invocar de la manera más transparente posible un servicio en una máquina virtual distinta a la que reside el cliente (en la misma máquina física pueden existir varias máquinas virtuales de Java). La diferencia entre estas dos tecnologías radica en que RPC se utiliza en diseños no orientados a objetos, mientras que RMI está soportado por el lenguaje orientado a objetos Java. RMI es un Middleware específico que permite a clientes invocar métodos de objetos como si estuviesen en la misma máquina virtual.

4.3. Distributed Component Object Model (DCOM): Al igual que CORBA y RMI, permiten la comunicación de objetos, pero únicamente para las diferentes versiones del sistema operativo Windows. DCOM surge cómo evolución de Object Linking and Embedding (OLE) y Component Object Model (COM).

5. Transaction Processing Monitors (TP Monitors): El Middleware para Procesamiento de Transacciones ya que facilita la conectividad y el acceso a un gran número de usuarios con servicios de back-end limitados. Este tipo de Middleware requiere del soporte de un monitor; es decir, un programa que supervise las transacciones entre procesos, con el propósito de asegurar el éxito de la transacción, o en caso de ocurrir un error, tomar acciones apropiadas. Su principal uso es coordinar el flujo de solicitudes entre los dispositivos y las aplicaciones que procesan esas solicitudes.

6. Database Access Technology (DBAT): Son las Application Programming Interface (API) creando una capa transparente para el acceso a base de datos, ocultando la complejidad dada por el manejador de base de datos.

6.1. Java Database Connectivity (JDBC): Es una API que facilita programar el acceso a bases de datos para Java sin que se tenga en cuenta a que Servidor se dirige (SQL Server, Oracle, Sybase, Informix, etc.). JDBC hace tres cosas: establece la conexión con una BD, envía sentencias SQL (Structure Query Language) y procesa los resultados.

6.2. Open Database Connectivity (ODBC): Es una API abierta para acceder a bases de datos. Especifica un conjunto de funciones para manejar conexiones a bases de datos, ejecutar declaraciones SQL y consultar las capacidades del sistema de base de datos.

7. Component Oriented Framework (COF): Este Middleware está soportado en el modelo de desarrollo de aplicaciones basado en componentes, permite la creación de una aplicación como conjunto de componentes reusables. Los componentes son una evolución del software orientado a objetos para dar respuesta a la reusabilidad. Los Middleware basados en componentes permiten a éstos "pegarse" con otros componentes para lograr la integración.

8. Directory Services (DS): Es el Middleware que se basa en directorios que permiten reducir costos administrativos, simplifican y/o distribuyen tareas administrativas, reducen el número de passwords para un usuario mediante una única combinación de login-password, aumentan la seguridad y proporcionan una única localización para la información de los usuarios. Son similares a las bases de datos, pero contienen información más descriptiva; la frecuencia de lectura sobre ellos es bastante más alta que la de escritura y en consecuencia su manejo es más simple que el de una base de datos ya que no hace actualizaciones complejas.

9. Application Servers (AS): Es el Middleware que se enfoca en la parte de la aplicación o lógica del negocio. Conceptualmente un sistema puede tener muchas capas; sin embargo, la arquitectura más popular es de tres capas: interfaz, aplicación y base de datos.

1.3.4 – Concluyendo sobre Middleware.

A modo de conclusión se puede echar un vistazo al mercado de forma tal que a partir de los productos middleware disponibles en el mismo se extraigan algunas consideraciones interesantes que permitan darle la importancia requerida a los proyectos relacionados con esta área de conocimientos de la informática y que puedan ayudar para un trabajo futuro tanto como consumidores de estos productos como de desarrolladores.

- Se puede comenzar diciendo que no existe un acuerdo que nos hable sobre el uso de los términos para describir las características de los productos por tanto puede darse la situación de que hayan tantas interpretaciones de lo que es un ESB, por poner un ejemplo, como proveedores. Esta situación es aprovechada por los departamentos de marketing para colocar sus productos en un segmento del mercado bastante exclusivo y prometedor obligando de esta forma al consumidor a solicitar una lista detallada de funcionalidades y comprobar por si mismo si el producto ofrece o no todo lo que le interesa pues los folletos no suelen ser suficientes.

- Por la misma razón anterior resulta muy complicado realizar comparaciones entre distintos productos. Las comparativas publicadas por algunas consultoras resultan engañosas.

- Algunos proveedores también lo son de otros tipos de software y los comercializan conjuntamente dentro de paquetes por lo que es importante estar advertido de las dependencias y posibles costos ocultos.

- Los productos disponibles incorporan gran número de funcionalidades y servicios y es probable que sólo parte de ellos sean interesantes para un proyecto determinado lo cual hace innecesario comprar todos los servicios sino sólo lo que se vaya a usar.

Estos elementos corroboran la necesidad de contar con un subsistema de comunicación que pueda ser utilizado en nuestros propios productos y cuya arquitectura nos permita: Primero, cambiar la capa de la tecnología sin incurrir en grandes cambios y sin afectar la interfaz con el cliente y segundo; agregar, modificar y eliminar servicios dependiendo del ambiente o las necesidades del cliente.

CAPÍTULO 2: ANÁLISIS DE TECNOLOGÍAS.

Introducción.

Como se ha podido apreciar en el capítulo anterior los middleware constituyen una de las partes más importantes en cualquier proyecto de software distribuido, es por eso que mientras más conozcamos de ellos y de su papel, mejor será todo el proceso de concepción y desarrollo de los mismos. Esta razón es la que motiva a profundizar en un tema tan amplio y complejo como es el de las tecnologías que se utilizan a nivel mundial para su desarrollo. Mientras más conocimientos tengamos de las mismas y de las tendencias actuales mejor será el proceso de estudio, evaluación y selección de la tecnología más adecuada para implementarlos independientemente del área en el que se pretenda utilizar o cuál de ellas es la que mejor puede satisfacer las características que se les definan al middleware.

Así mismo es muy importante el análisis de las distintas arquitecturas de software que pueden proveer mayor facilidad a la hora de estructurar, asignar responsabilidades y organizar los diferentes elementos que conformarán el sistema de comunicación.

Esto es lo que ha motivado la estructura de este capítulo, que en una primera parte se ha querido dedicar a la caracterización de las principales tecnologías que se utilizan en el desarrollo de los middleware. Y en una segunda parte del capítulo se hace referencia a dos estilos de arquitectura muy relacionados con la distribución de las aplicaciones que pueden ser de mucha utilidad para la investigación.

2.2 – Tendencias y Tecnologías.

Resulta muy difícil seleccionar una tecnología para desarrollar un middleware genéricos 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 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, algunos ejemplos son: la de cliente/servidor o publicación/subscripción; también están los compiladores que utilizan o las diferentes calidades de servicios, incluso muchos otros elementos que no son objetivo mencionarlos aquí pero todos tienen un objetivo común, el de garantizar la comunicación.

2.2.1 – CORBA.

Common Object Request Broker Architecture (CORBA) es un estándar que establece una plataforma de desarrollo de sistemas distribuidos facilitando la invocación de métodos remotos bajo el paradigma orientado a objetos.

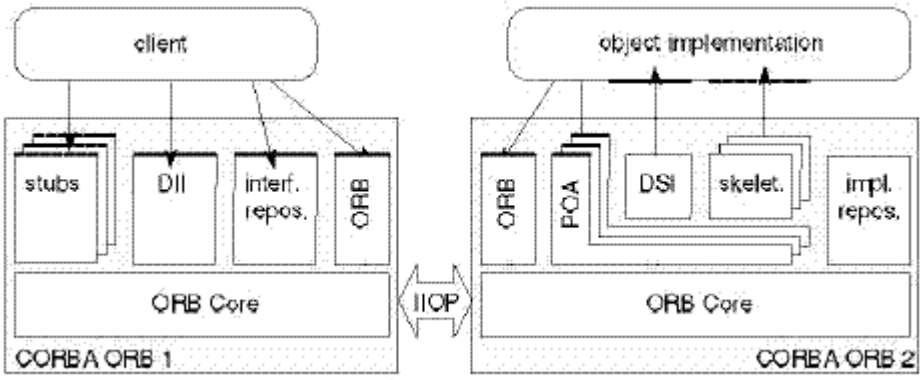

Figura # 2.1: Esquema Estructural de CORBA.

CORBA constituye la respuesta del Object Management Group (OMG) a la necesidad de inter-operatividad entre la rápida proliferación de numeroso hardware y software disponible hoy en día ya que permite a las aplicaciones comunicarse unas con otras independientemente de donde estén localizadas o quien las haya diseñado. Es el propio OMG quién define las APIs, el protocolo de comunicaciones, y los mecanismos necesarios para permitir dicha inter-operatividad entre las diferentes aplicaciones escritas en diferentes lenguajes y ejecutadas en diferentes plataformas, lo que es fundamental en computación distribuida.

En sentido general 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.

CORBA utiliza un lenguaje de definición de interfaces (IDL) para especificar las interfaces con los servicios que los objetos ofrecerán. CORBA puede especificar a partir de este IDL la interfaz a un lenguaje determinado, describiendo cómo los tipos de dato CORBA deben ser utilizados en las implementaciones del cliente y del servidor. Existen implementaciones estándar para Ada, C, C++, Smalltalk, Java y Python. Hay también implementaciones para Perl y TCL. Se dice que actualmente existen disponibles alrededor de 29 implementaciones CORBA. [Anexo 1]

Al compilar una interfaz en IDL se genera código para el cliente y el servidor. El código del cliente sirve para poder realizar las llamadas a métodos remotos. Es el conocido como "*stub"*, 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.

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 una 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 inter-operatividad entre aplicaciones en diferentes máquinas en entornos distribuidos heterogéneos y conecta múltiples sistemas de objetos.

En el terreno de las aplicaciones típicas cliente/servidor, los desarrolladores usan sus diseños propios o estándares reconocidos para definir el protocolo que se va a usar entre los dispositivos. Las definiciones del protocolo dependen del lenguaje de implementación, el transporte en red y una docena de factores. El ORB simplifica este proceso, con él, el protocolo se define a través de interfaces de aplicación por medio de una especificación independiente del lenguaje de implementación, el IDL. Los ORBs proporcionan flexibilidad. Dejan a los programadores elegir el sistema operativo más apropiado, entorno de ejecución e incluso el lenguaje de programación a usar para cada componente de un sistema en construcción. Más importante, permiten la integración de componentes existentes. En una solución basada en ORB, los desarrolladores simplemente modelan la herencia del componente usando el mismo IDL que usan para crear nuevos objetos, después escriben el código "wrapper" que traduce entre el bus estandarizado y las interfaces heredadas.

2.1.1.1 – Arquitectura de CORBA.

Después de dar una vista panorámica de lo que es CORBA se explicará más detalladamente su funcionamiento. Se puede decir que CORBA define una arquitectura para objetos distribuidos cuyo paradigma básico consiste en realizar solicitudes para servicios de objetos distribuidos, todo el resto de las cosas definidas por el OMG es en términos de este paradigma. Existen dos aspectos de la arquitectura de CORBA que sobresalen y los cuales se muestran a continuación:

1. Las implementaciones tanto del cliente como del objeto están aisladas del ORB por una interfaz IDL, esto es para garantizar la sustitución de las implementaciones para cada interfaz.
2. Las solicitudes no pasan directamente del cliente al objeto, siempre son administradas por un ORB. Lo que hace que los detalles de la distribución sean transparentes para los clientes y para los objetos.

Además podemos destacar otras características como:

1. Los servicios que un objeto provee son dados por su interfaz. Las interfaces son definidas en el Lenguaje de Definición de Interfaz (IDL) del OMG.
2. Los objetos distribuidos son identificados por referencias a objetos, las cuales son definidas por las interfaces IDL.
3. Un cliente tiene una referencia a un objeto distribuido. La referencia al objeto es dada por una interfaz.
4. El ORB envía la solicitud al objeto y regresa cualquiera de los resultados al cliente.

La arquitectura de administración de objetos propuesta por OMG define una arquitectura común que hace posible que exista un sistema unificado de cómputo basado en componentes heterogéneos que están inter-operando entre si. Los servicios proporcionados por CORBA proporcionan servicios necesarios para casi todo sistema basado en objetos, mientras que las facilidades de CORBA permiten un acceso estándar a tipos de datos comunes y la funcionalidad necesaria para grupos de aplicaciones corporativas y específicas para las aplicaciones industriales.

2.1.1.2 – Componentes de CORBA.

A veces resulta muy engorroso entender el funcionamiento de CORBA por lo que la definición de algunos de sus componentes pudiera resultar de gran ayuda en el proceso de aprendizaje de esta tecnología.

1. Object Request Broker (Agente de Petición de Objetos) - (u ORB): Un ORB es una pieza de middleware, se sitúa en medio de los clientes y servidores y hace posible la fácil comunicación entre ellos. El ORB es un ente conceptual, que algunas veces toma la forma de una librería compartida y otras veces toma la forma de un programa externo. El ORB es el responsable de establecer y destruir las sesiones entre clientes y servidores y dirigir (o no), y transportar mensajes entre ellos durante una sesión.

2. Interface Definition Language (Lenguaje de Definición de Interfaz) – (o IDL): La definición de la

interfaz especifica los métodos que el objeto esta preparado para realizar, sus parámetros de entrada, su resultado y cualquier excepción que pueda generarse durante la ejecución. En el momento de construir un objeto CORBA el primer paso es definir cual va a ser la funcionalidad que va a proporcionar, para de esta forma poder escribir la interfaz en IDL. Toda la información necesaria para construir un cliente del objeto es proporcionada por la interfaz. Se debe escoger un lenguaje de programación que facilite la implementación de la interfaz de cada objeto.

3. Dynamic Invocation Interface (Interfaz de Invocación Dinámica) – (o DII): La DII puede ser utilizada cuando no se tiene acceso a las interfaces del servidor en tiempo de compilación. En tiempo de ejecución los detalles de la descripción de la interfaz se obtienen del Repositorio de Interfaces.

4. Dynamic Skeleton Interface (Interfaz de Esqueleto Dinámico) – (o DSI): Permite a los servidores implementarse sin skeletons compilados estáticamente. DSI tiene muchos usos pero uno de los más importantes es definir objetos con comportamiento dinámico.

5. Object Adapter (Adaptador de Objetos) - (u OA): Un adaptador de objetos proporciona el canal por el cual un objeto servidor se comunica con el Object Request Broker.

6. Clientes y Servidores: En CORBA la terminología cliente/servidor no es muy estricta: servidor es la aplicación que contiene objetos, cliente es quien realiza las peticiones sobre dichos objetos. Una aplicación CORBA puede jugar ambos roles, incluso al mismo tiempo.

2.1.1.3 – Servicios de CORBA.

Una parte muy importante del estándar CORBA es la definición de un conjunto de servicios distribuidos que soportan la integración y la interoperabilidad de objetos distribuidos. Los servicios de CORBA (COS) están definidos en la parte de arriba del ORB lo cual quiere decir que están definidos como objetos estándares de CORBA con interfaces IDL, algunas veces referidas a ellas como Servicios de Objetos. Estos servicios son unos “built on top” de la infraestructura básica de CORBA que facilitan la programación de objetos distribuidos. Algunos de los servicios son:

- Object Life Cycle: Define cómo los objetos de CORBA son creados, removidos, movidos y copiados.
- Naming: Define cómo los objetos de CORBA pueden tener nombres simbólicos amigables.
- Events: La comunicación entre objetos distribuidos.
- Relationships: Provee relaciones arbitrarias n-areas entre tipos de objetos CORBA.
- Externalization: Coordina la transformación de objetos CORBA hacia y desde medios externos.
- Transactions: Coordina accesos atómicos a objetos CORBA.
- Concurrency Control: Provee un servicio de bloqueo para objetos de CORBA en orden de asegurar acceso concurrente.
- Property: Soporta la asociación de pares nombre-valor con objetos CORBA.
- Trader: Soporta el encuentro de objetos CORBA basado en propiedades, describiendo el servicio ofrecido por el objeto.
- Query: Soporta colas en objetos.

2.1.2 – ORBit.

El surgimiento de ORBit está relacionada 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. [Fundación GNOME, 2002]

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 ganchos que permiten una fácil integración con los programas de GNOME.

ORBit cumple con los dos requisitos del proyecto GNOME, es libre y es uno de los ORB más rápido que existen 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.

2.1.3 – ACE+TAO.

ACE+TAO ha sido desarrollado por el grupo DOC (Distributed Object Computing), 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 ACE ORB (TAO) 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. TAO se ha diseñado utilizando las mejores prácticas de software y los patrones que descubrió el grupo de desarrollo en 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 entra salida para proporcionar sistemas ORBs 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 ORBs. Proporcionar una plataforma middleware basada en CORBA con una alta calidad, disponible gratuitamente, de código abierto para que sea usado por investigadores y desarrolladores.

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. [Schmidt, 2007]

2.1.3.1 – ACE.

El Entorno de Comunicación Adaptativa (ACE por sus siglas en inglés) es un framework disponible libremente, de código abierto y orientado a objeto 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 señal, inicialización de servicios, comunicación entre procesos, gestión de memoria compartida, enrutamiento de mensajes, reconfiguración dinámica de servicios distribuidos, ejecución concurrente y sincronización. [Schmidt, 2007]

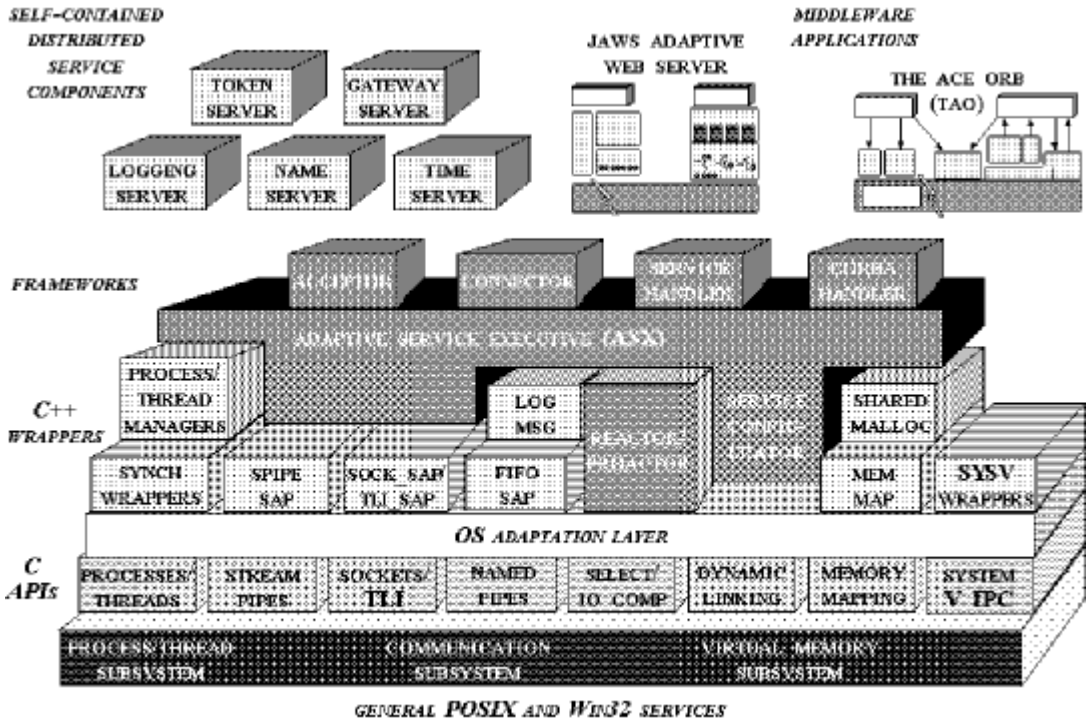

Figura # 2.2: La Estructura en capas de los componentes en el framework de ACE.

2.1.3.2 – Beneficios de la utilización de ACE.

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 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 ACE ORB (TAO) que es un estándar de código abierto y compatible con la implementación 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.

2.1.3.3 – Estructura y funcionalidad de ACE.

- Capa Adaptadora del Sistema Operativo de ACE:

La capa adaptadora de sistema operativo de ACE se encuentra directamente arriba de las APIs nativas del SO y que están escritas en C. Proporciona una capa de adaptación del SO "POSIX-like" que protege las otras capas y componentes de ACE en la plataforma específica de las dependencias relacionadas con las siguientes API del SO: concurrencia y sincronización, comunicación entre procesos y memoria compartida, mecanismos para demultiplexar eventos, enlace dinámico explícito y mecanismos de sistemas de archivo. La portabilidad de la Capa de Adaptación del SO de ACE le permite ejecutarse en muchos sistemas operativos y debido a la abstracción proporcionada por esta capa, se utiliza un único árbol de código fuente para todas estas plataformas, este diseño simplifica grandemente la portabilidad y la facilidad de mantenimiento de ACE.

- Envoltura de fachadas C++ para Interfaces SO:

Es posible programar aplicaciones en C++ altamente portables directamente encima de la Capa de Adaptación del SO de ACE. Este componente simplifica el desarrollo de aplicaciones mediante el suministro de interfaces C++ "type safe" que encapsulan y mejoran la concurrencia del sistema operativo nativo, la comunicación, administración de memoria, demultiplexar eventos, enlace dinámico y las APIs del sistema de archivos. Las aplicaciones pueden combinar y componer estos envoltorios por heredar

selectivamente, agregar y/o instanciar los siguientes componentes: concurrencia y sincronización de procesos, IPC y componentes de sistemas de ficheros y componentes de gestión de memoria.

La envoltura de fachadas C++ proporciona muchas de las características de la capa de adaptación pero estas están estructuradas en términos de C++: clases y objetos. Este paquete orientado a objeto ayuda a reducir el esfuerzo necesario para aprender y usar correctamente ACE. El uso del C++ mejora la robustez de la aplicación ayudando al compilador a detectar tipos de violaciones del sistema en tiempo de compilación en lugar de en tiempo de ejecución. ACE emplea una serie de técnicas para minimizar o eliminar el rendimiento sobre la cabeza, además evita el uso de métodos virtuales para el rendimiento de envolturas críticas.

- Marcos:

ACE también contiene un framework de programación de red de alto nivel que integra y mejora el nivel inferior de la envoltura de fachadas C++. Este framework apoya la configuración dinámica de servicios distribuidos concurrentes en las aplicaciones. Este marco contiene los siguientes elementos: componente para demultiplexar eventos, componente de inicialización de servicios, componente de configuración de servicios, componente de flujo de capas jerárquicas y componente adaptador ORB. Los componentes del framework de ACE facilita el desarrollo de software de comunicación que puede ser actualizado y ampliado sin necesidad de modificar, recompilar, relinkear o reiniciar aplicaciones en ejecución. Esta flexibilidad se logra en ACE por la combinación de las características del lenguaje C++, tales como los templates, la herencia, la vinculación dinámica, los patrones de diseño, configuración de servicios y los mecanismos del SO.

- Servicios distribuidos y componentes:

ACE además de los elementos vistos anteriormente ofrece una librería estándar de servicios distribuidos que están envasados como componentes autónomos. A pesar de que estos componentes de servicios no forma parte estrictamente de la librería de ACE desempeñan dos funciones: llevar a cabo bloques de construcción reusables en aplicaciones distribuidas y demostrar los casos de uso comunes de los componentes de ACE.

- Nivel superior en los componentes middleware para computación distribuida:

Lograr un desarrollo robusto, extensible y una comunicación eficiente entre las aplicaciones es un desafío pues los desarrolladores deben dominar una serie de complejidades y conceptos tales como: direccionamiento de red, servicio de identificación, presentación de conversaciones como la codificación y compresión, proceso de creación de hilos y sincronización, llamadas al sistema y rutinas de interfaces de librerías para mecanismos de comunicación entre procesos locales y remotos. Es posible aliviar este tipo de complejidades en el desarrollo de aplicaciones de comunicación empleando el más alto nivel de middleware para la informática distribuida. Este nivel reside entre clientes y servidores, además en aspectos tediosos de desarrollo de aplicaciones distribuidas, y propensos a errores entre los que se incluyen las siguientes: autenticación, autorización y seguridad de los datos, servicio de localización, servicio de registro y activación, demultiplexar y envío de respuestas a eventos, implementación de elaboración de mensajes orientados a protocolos de comunicación.

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: el ACE ORB (TAO).

2.1.3.4 – 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 alto rendimiento y en tiempo real de servicios de calidad para aplicaciones distribuidas.

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 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 inter-funcionar con otros ORBs por lo que se tiene mucha confianza en que su aplicación es robusta e interoperable. [Schmidt, 2007]

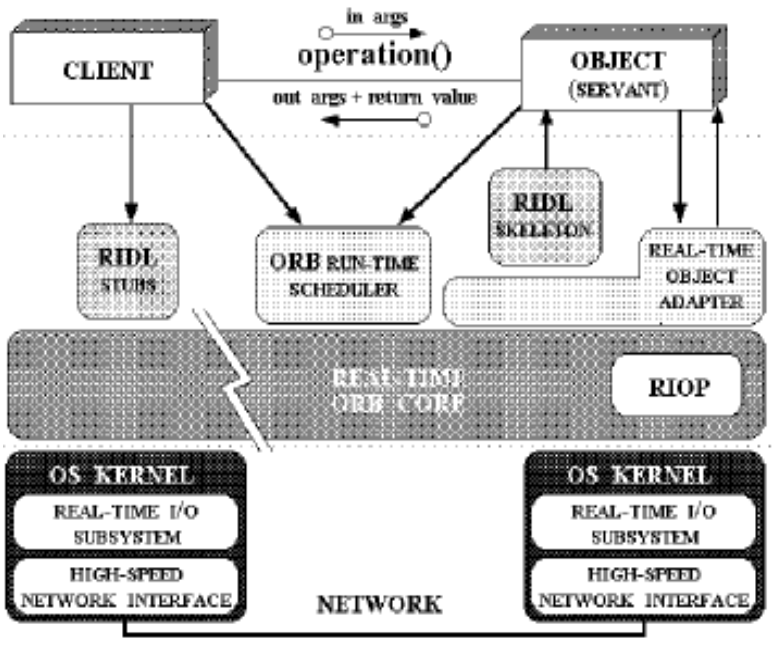

Figura # 2.3: Componentes en el ORB de tiempo real TAO.

2.1.3.5 – Componentes y servicios de TAO.

Para realizar su trabajo TAO cuenta con los siguientes componentes:

- Compilador IDL.
- Motor de protocolos Inter-ORB.
- Núcleo ORB.
- Adaptador de objetos portables.
- Repositorio de implementaciones.
- Repositorio de interfaces.

TAO brinda además 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.

Además, TAO presta otros servicios adicionales para diversos tipos de dominios de aplicaciones DRE. Algunos de ellos son:

- Servicio de Equilibrio de Carga: Este servicio aplica los algoritmos “round robin” y mínima dispersión para dispersar las cargas en un grupo de máquinas.

- Servicio de Eventos en Tiempo Real: Este servicio aumenta el modelo del servicio de eventos del estándar de CORBA mediante el suministro de la fuente y filtrado basado en tipos, la correlación de eventos, el envío en tiempo real y la comunicación multicast UDP/IP.

- Servicio de planificación: Soporta la planificación monotónica del índice estático y la planificación primera de urgencia máxima dinámica para asignar prioridades y validar la planificación. Actualmente está integrado con el servicio anterior.

Podemos concluir diciendo que varias pruebas y ejemplos de aplicaciones sobre TAO han mostrado como programar el ORB para aplicaciones en tiempo real y que no son de tiempo real. El código fuente en C++ de todos los códigos abiertos de los componentes, servicios, ejemplos y pruebas del ORB TAO está disponible en la liberación de TAO.

2.1.3.6 – Optimizaciones de rendimiento.

Durante mucho tiempo el equipo de desarrollo de TAO conjuntamente con otros desarrolladores contribuyentes ya sean estudiantes o profesionales del área han trabajado en la optimización de esta tecnología diseñándola cuidadosamente, utilizando arquitecturas, diseños, patrones de optimización que mejoren sustancialmente la eficiencia, previsibilidad y la escalabilidad en los sistemas DRE. Las optimizaciones relacionadas con la investigación del proyecto TAO se mencionan a continuación.

- Un núcleo ORB que apoya la concurrencia en tiempo real determinista y estrategias de envío.
- Optimización de demultiplexado activo y lineado perfecto.
- Un motor de protocolo de IIOP altamente optimizado de CORBA y un compilador de IDL.
- Puede configurarse para usar un modelo de conexión no multiplexada.
- El protocolo de conexión permite el soporte para subsistemas E/S en tiempo real.
- Los servicios de eventos en tiempo real y planificación de servicios, estática y dinámica, integran las capacidades del ORB TAO que se han mencionado anteriormente.

2.1.3.7 – La orientación futura de ACE+TAO.

Es importante señalar que el grupo de desarrollo tiene un trabajo de mantenimiento, ampliación y mejoras de ACE y TAO. Todo parece indicar que las orientaciones futuras van destinas a lograr una mayor solidez sobre todo en el trabajo con la tolerancia a fallos, incorporar nuevas funciones además de la mencionada anteriormente, lograr nuevas optimizaciones, incorporar el trabajo en nuevas plataformas especialmente relacionadas con el tiempo real y sistemas embebidos, mejorar la documentación existente y alcanzar una mayor integración para que otros ORBs puedan utilizar servicios públicos de ACE y TAO.

2.1.4 – DDS.

El Servicio de Distribución de Datos (DDS) para sistemas de tiempo real es una especificación de middleware que cumple con el modelo Publicación/Subscripción para Sistemas Distribuidos creado en respuesta a la necesidad de aumentar CORBA con una especificación Data Centric Public Suscribe (DCPS). Unas pocas soluciones propietarias de DDS han estado disponibles durante varios años. En el año 2004 los dos principales proveedores de DDS la americana Real Time Innovations (RTI) y la francesa Thales Group se unieron para crear las especificaciones DDS que han sido aprobadas por el OMG.

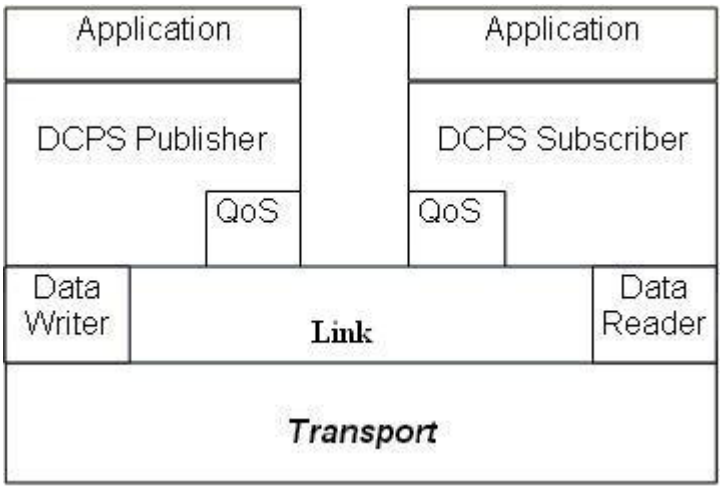

Figura # 2.4: Integración de la capa DCPS en DDS.

Las especificaciones DDS describen dos niveles de interfaces:

- Uno menor DCPS (Data-Centric Publicar-Suscribir): Este nivel está orientado a la prestación eficiente de la información adecuada para los beneficiarios adecuados.
- Uno superior opcional DLRL (Data Local Reconstruction Layer): Este nivel permite una sencilla integración de DDS en la capa de aplicación.

2.1.4.1 – El Modelo DDS.

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

Data Distribution Service 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 un 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.
- Donde están localizados los destinatarios.
- Que 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 es recuperado. DDS está disponible con APIs para C, C++ y Java, y en multitud de plataformas (Linux, Solaris, Windows, Integrity, LynxOs, VxWorks…).

2.1.4.2 – DDS y el Envío de Eventos.

DDS se puede utilizar para enviar cualquier tipo de datos, incluidos datos de eventos. Cuando las personas hablan acerca de los eventos, por lo general se refieren a dos cosas:

- Un esquema de los datos por ejemplo, los datos que llevan una cabecera con sectores específicos que identifican a cosas como la fuente del evento, la fecha y hora, la prioridad, una o más categorías de clasificación, palabras claves, etc.

- Una infraestructura que apoya una variedad de filtrado y mecanismos de envío basados en los campos de cabecera, la prioridad, el contenido, etc.

En DDS los desarrolladores de aplicación son los encargados de definir el esquema de los datos. DDS no define un tipo de evento “built-in”, pero la aplicación puede definir cualquier tipo de evento e incluir los

campos que quiere. Además el uso de filtros a base de tiempo y tema filtrado de contenidos permite la solicitud para realizar el filtrado flexible de los acontecimientos.
DDS contiene una QoS que es más rica que la mayoría de los sistemas de envío de eventos que pueden utilizarse para propagar no sólo eventos sino también información de estado y los valores de los datos incluidos en la muestra. Cuando se usa para propagar los eventos la QoS debe estar configurada para que coincida con la semántica típica de los eventos. Además los dominios y la partición QoS pueden ser aprovechados para crear sobreimpresiones de distribución de eventos flexibles y escalables.

2.1.4.3 - Entidades de DDS.

- Domain Participant Factory: Una fábrica individual que es el principal punto de entrada a DDS.

- Domain Participant: Punto de partida para la comunicación en un dominio específico, sino que representa la participación de una solicitud en un dominio de DDS. Por otra parte, actúa como una fábrica para la creación de publicadores, suscriptores, temas, multitemas y filtrado de contenido de temas.

- Topic Description: Clase base abstracta por Tema, Filtrado de Contenido de Temas y Multitemas.

- Topic Tema: Una especialización de Topic Description que es la más básica descripción de los datos que se publicarán y suscribirán.

- Content Filtered Topic: Una especialización de Topic Description como el tema que adicionalmente permite suscripciones basadas en contenidos.

- Multi Topic: Una especialización de Topic Description como el tema que además permite combinar las suscripciones, filtro y reordenar los datos procedentes de varios temas.

- Publisher Editorial: Un editor es el objeto responsable de la diseminación real de publicaciones.

- Data Writer: Permite la aplicación para ajustar el valor de los datos que deben publicarse en virtud de un determinado tema.

- Subscriber El suscriptor: Un suscriptor es el objeto responsable de la recepción real de los datos resultantes de sus suscripciones.

- Data Reader: Un Data Reader permite la aplicación de declarar los datos que desea recibir y el acceso a los datos recibidos por el adjunto del suscriptor.

2.1.5 – OpenDDS.

OpenDDS es una implementación C++ de código abierto de la especificación del Data Distribution Service para Sistemas de Tiempo Real del Grupo de Administración de Objetos (OMG). Implementa la mayor parte de los perfiles mínimos de la capa Data-Centric Public-Subscriber (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 como la base del Repositorio de Información del DCPS de OpenDDS (DCPSInfoRepo). Adicionalmente aprovecha MPC para aliviar la carga de mantenimiento para apoyar la construcción de múltiples entornos y plataformas. OpenDDS cuenta con el apoyo del Object Computing, Inc. (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.

2.1.5.1 – Descripción y características de OpenDDS.

OpenDDS está basado en la especificación 1.0 de DDS. 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 cuenta.

OpenDDS soporta algunas de las capacidades definidas en la versión 1.0 de DDS. Podemos decir, por ejemplo, que no todas las políticas QoS están plenamente soportadas. Las condiciones, esperas establecidas, tema filtrado por contenido y multitemas, además de los métodos que soportan temas relacionados con estas clases no son compatibles. La funcionalidad de los Temas "Built-In" están activadas y disponibles por defecto pero estas tienen algunos problemas. Los desarrolladores pueden definir una estructura en IDL que se utilizará como tipo de datos DDS, la estructura básica puede incluir tipos escalares, cadenas, secuencias, arreglos, enumeraciones y unión; puede que no contenga interfaces o tipos de valores y el cero o más claves pueden ser especificadas para un tipo de datos. OpenDDS proporciona un framework de transporte que facilita añadir un nuevo transporte. [Object Computing Inc., 2007]

2.1.5.2 – Relación con CORBA.

OpenDDS es un híbrido de la comunicación CORBA/GIOP para tareas administrativas y transporte personalizado de tipo seguro para la transferencia de datos. Permite a un desarrollador beneficiarse del paradigma de computación de objetos distribuidos, interoperabilidad y la transparencia de CORBA en la conexión publicador/subscriptor al tiempo que satisface las necesidades de rendimiento del transporte en aplicaciones de tiempo real. El uso de transporte personalizado permite a los desarrolladores de sistemas obtener las ventajas de eficiencia y baja latencia en protocolos de transmisión de datos, manteniendo al mismo tiempo los beneficios de interfaces estándares de alto nivel.

OpenDDS fue desarrollado para asegurar su existente funcionalidad CORBA y fue compartido con DDS y la duplicación de funcionalidad debe evitarse, por ejemplo, OpenDDS utiliza el compilador IDL de TAO. TAO se utiliza para registrarse en un tema y se requieren los servicios de calidad. El ORB se requiere y se ejecuta con DDS. OpenDDS permite a los diseñadores mezclar fácilmente estilos DDS y CORBA en una forma única.

OpenDDS 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. Con cuidado, será posible en muchos casos mejorar sin que esto afecte a los demás.

En las zonas en las que no hay utilización de ningún núcleo ORB, OpenDDS utiliza ACE como capa de abstracción o portabilidad. De esta forma DDS apoya a la misma amplia variedad de plataformas como TAO. También permite DDS para obtener socket (como ACE) para el nivel de rendimiento en su transferencia de datos. Si se compara OpenDDS con el canal de eventos TAO RT, que tiene un servicio publicar/subscribir, parece indicar una mejora tres veces mayor en el rendimiento, en una mensajería de punto a punto.

2.1.5.3 – Framework de Transporte.

A diferencia de CORBA las especificaciones OMG DDS no han definido todavía un protocolo de comunicación común como es el caso del IIOP para CORBA. Aunque se está trabajando en la culminación de una especificación que está en fase final llamada RTPS. Es por esto que las aplicaciones DDS pueden en la actualidad soportar cualquiera que se estime apropiado. OpenDDS soporta los estilos de publicación uno a uno (punto a punto) y uno a muchos (multicast). En el caso de los transportes que actualmente soportan TCP/IP y UDP; ambos, multicast y multicast fiable son ofrecidos por UDP.

Las características distintivas de muchos sistemas con el modelo publicar/subscribir en tiempo real es un transporte que es muy particular para el dominio del problema. Diferentes industrias ya tienen los transportes tales como la automatización industrial, sistemas de negociación financiera, sistemas de sensores de defensa entre otros. Muchos de estos dominios han combinado transportes con protocolos de nivel mayor.

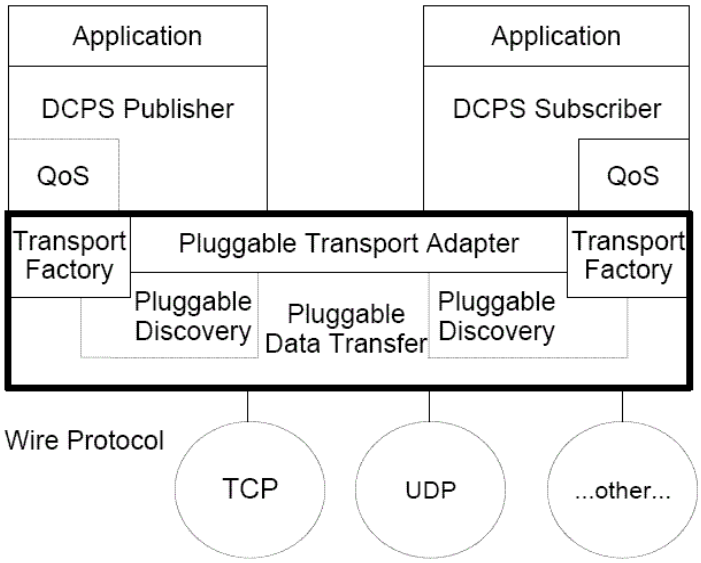

Figura # 2.5: Framework de Transporte en OpenDDS.

OpenDDS separa el transporte de los protocolos de nivel superior por medio de un framework de transporte extensible (ETF). Este ETF no es estándar porque actualmente no hay ninguna norma. Esto tiene un impacto mínimo en los desarrolladores DDS pues las cuestiones de transporte son transparentes para ellos. El ETF solo ha de soportar un transporte una vez y todos los desarrolladores pueden aprovecharlo. Un enfoque similar se utiliza dentro de TAO y se llama “Pluggable Protocols”.

2.1.5.4 – Políticas de uso de los QoS.

A continuación se muestran algunas políticas de QoS importantes para el rendimiento del sistema, estos aspectos descritos se verán en cuatro direcciones: entrega en tiempo real, ancho de banda, redundancia y persistencia. No todas las políticas de QoS se aplican por lo que se recomienda siempre revisar el listado de políticas QoS para que se pueda conocer cual es la política en la que se está interesado antes de su aplicación.

Entrega en Tiempo Real:

- Plazo: Exigir que el publicador siempre envíe al menos un valor dentro de dicho período.
- Fiabilidad: El uso de una configuración fiable puede resultar en reenvíos de datos. La configuración fiable puede bloquear potencialmente al intentar enviar. El tiempo de bloqueo máximo es importante para minimizar el tiempo que el envío puede bloquearse.
- Durabilidad: Al establecer un tiempo de caducidad de las muestras se asegurará que la aplicación receptora no recibirá ningún valor demasiado viejo.
- Límites de Recursos: Debido a muy pocas muestras disponibles se van a producir retrasos en la adquisición de muestras para enviar.
- Presentación: No permite la agrupación y reordenación de los valores que pueden retrasar la entrega de la solicitud.

Ancho de Banda:

Es importante aclarar que ninguna política de QoS es para el control directo del ancho de banda aunque algunas tienen efectos sobre esta.

- Animación: Una baja duración de la animación tendrá como resultado mensajes de animación extras para los publicadores que no son frecuentemente valores publicados.
- Plazo: Establecer un plazo inferior se traducirá en más tráfico. (Menor plazo aumenta el ancho de banda)
- Filtro Basado en Tiempo: Establecer una mayor separación mínima se traducirá en menos muestras de las que están establecidas. (Mayor separación reduce el ancho de banda)
- Fiabilidad: El uso de una configuración fiable puede resultar en reenvíos de datos. (Configuración fiable aumenta el ancho de banda)

Redundancia:

- Titularidad: Una configuración exclusiva permite que varios publicadores puedan publicar por la

misma instancia pero solo uno es reconocido como el propietario (fuente autorizada). Un error o la muerte del actual propietario permite a los suscriptores a continuar recibiendo los valores del nuevo propietario.

- Titularidad Resistente: Se establece el orden de la propiedad cuando la propiedad tiene una configuración exclusiva.
- Animación: La duración del contrato es importante para determinar que una entidad ha muerto. Un valor más bajo significa que una muerte se detectó antes.
- Orden de Destino: By-Reception-Timestamp puede resultar en dos suscriptores que tengan datos en órdenes diferentes. By-Source-Timestamp fuerza a todos los suscriptores para tener los datos en el mismo orden.
- Historia: Las configuraciones de buffer entre los cambios de los valores y las entregas a las suscripciones. La última configuración guardada y una pequeña profundidad podría causar el bloqueo en función de otras políticas QoS.

Persistencia:

- Durabilidad: Control de como los datos serán almacenados. Valores de transición y persistencia de los datos obtenidos del Data Writer.
- Servicio de Durabilidad: Esta política no se aplica todavía pues no está implementada por haberse añadido después de la versión 1.0. Esta política configura el volumen de datos que se almacena después de que se hay publicado. La política Límite de Recursos define la cantidad de datos que se almacene en OpenDDS.
- Historia: Controla el almacenamiento de valores antes de que sean entregados.
- Límite de Recursos: Especifica que número de muestras están disponibles para que puedan ser manipuladas.
- Ciclo de Vida del Data Writer: Si las muestras serán automáticamente dispuestas.
- Ciclo de Vida del Data Reader: Si las muestras serán automáticamente eliminadas.

2.1.6 – ICE.

El Internet Communication Engine (ICE) es una plataforma ligera desarrollada por la empresa ZeroC, creado por algunos de los creadores de CORBA e intenta resolver algunos de los problemas de esta última. Actualmente ICE es gratuito. Este Middleware es demasiado complejo pues no asume nada lo cual hace que en la actualidad ningún otro ORB implemente todas las funciones que ICE contempla.

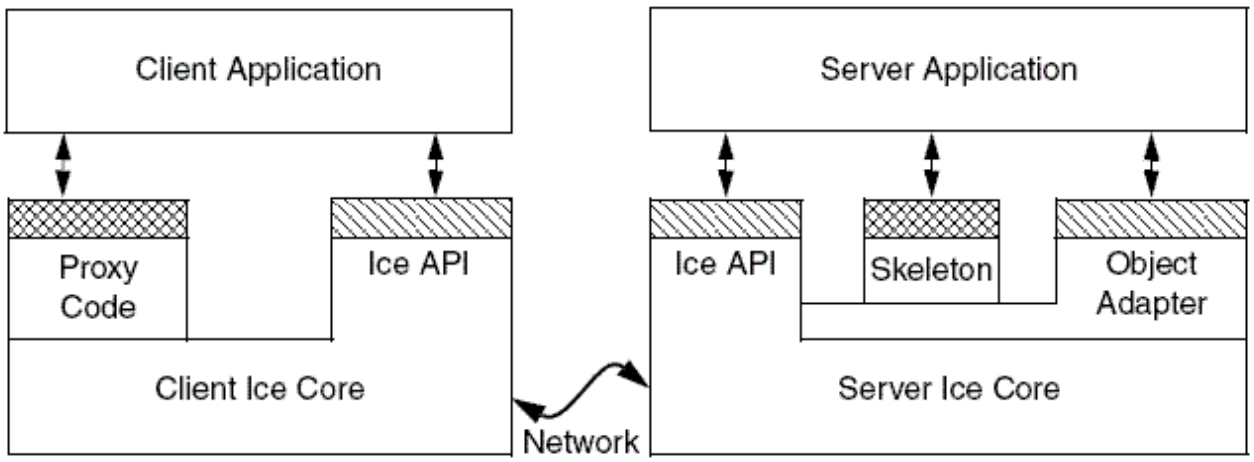

Figura # 2.6: Estructura cliente-servidor en ICE.

Provee una plataforma abierta adecuada para sistemas heterogéneos de cualquier fabricante y proporciona un soporte completo para el desarrollo de aplicaciones distribuidas realistas de cualquier tipo de dominio. Rehúye de la complejidad innecesaria y optimiza la implementación lo cual lo hace mucho más rápido y ligero por lo que garantiza un uso eficiente del ancho de banda de la red, uso de memoria y utilización del CPU. Provee medidas de seguridad que garantiza la construcción de aplicaciones seguras lo que permite que estas puedan ser utilizadas en redes inseguras incluyendo redes públicas. [ZeroC Inc., 2008]

2.1.6.1 – Características generales de ICE.

Como se pudo ver ICE no es más que un middleware para el programador práctico, una plataforma de comunicaciones por Internet de alto rendimiento que incluye varias capas de servicios y plugin, y se compone de los siguientes paquetes:

- 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 una piscina de hilos flexibles para servidores multiprocesadores y una funcionalidad adicional que apoya la extrema escalabilidad con millones de potencialidades de un objeto ICE.

- Ice Util: Una colección de funciones de utilidad tales como la manipulación de un estándar de codificación e hilos de programación. (Solamente C++)

- Ice Box: 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.

- Ice Grid: Un sofisticado servidor de activación y despliegue de herramientas avanzadas para computación grid.

- 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.

- Freeze Script: 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 apoyan un XML basado en su capacidad de scripting que es a la vez poderoso y fácil de usar.

- Ice SSL: Un SSL dinámico transporta plugin para el núcleo de ICE. Ofrece autenticación, cifrado e integridad en los mensajes utilizando el protocolo estándar de la industria SSL.

- Glacier: Uno de los retos más difíciles para los sistemas middleware es la seguridad y los cortafuegos. Glacier es la solución de cortafuegos para ICE, lo 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.

- Ice Storm: Es un servicio de mensajería con el apoyo de la federación. En contraste con la mayoría de las mensajerías o servicios de eventos, Ice Storm soporta tipos de eventos, en el sentido de que la radiodifusión de un mensaje a través de una federación es tan fácil como invocar un método en una interfaz.

- 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 directorio. Sólo los archivos que han cambiado son descargados a la máquina cliente, utilizando algoritmos de compresión eficiente.

2.1.6.2 – Arquitectura de ICE.

Como cualquier otra tecnología vista anteriormente ICE cuenta con diferentes elementos que le permiten cumplir con sus objetivos de forma tal que una vez vistas las características generales de esta tecnología podemos ver un poco mejor como es su funcionamiento.

En ICE los clientes son entidades activas que requieren servicios y los servidores son entidades pasivas que proveen servicios en respuesta a las peticiones que le llegan. Los objetos ICE son entidades en la máquina local o remota que responden a peticiones de clientes además pueden ser instanciados en uno o varios servidores de forma redundante pero sigue siendo un único objeto y cada uno de estos objetos cuenta con una o más interfaces.

Una interfaz es una colección de operaciones soportadas por el objeto y cada operación tiene cero o más parámetros y un valor de retorno tipado. Un objeto tiene una interfaz principal y puede tener varias secundarias denominadas “facets” y los clientes pueden seleccionar la interfaz con la que desean trabajar. Cada objeto además posee una identidad (id) y estas son globalmente únicas.

Otro de los elementos a tener en cuenta en la arquitectura de ICE son los proxys que juegan el papel de intermediarios entre los clientes y los servidores. Cuando el cliente invoca un método este localiza el objeto, activa el servidor del objeto si es necesario, activa el objeto en el servidor, transmite los parámetros de entrada, espera a que termine la operación y devuelve los parámetros de salida y el valor de retorno.

Existen varios métodos de invocación que ICE utiliza, estos son:

- Síncrona: El cliente llama al método y se queda suspendido hasta que se completa.

- Asíncrona: El cliente pasa los parámetros y también un objeto callback. Al hacer la llamada continúa su ejecución y cuando acaba la operación se llama a este objeto.

- One Way: El cliente manda una petición asíncrona pero no espera una respuesta y en realidad esta nunca se da por lo que es poco confiable.

- Batched One Way: Acumula varias peticiones One Way en un buffer y las manda de golpe.

Otro de los elementos de la arquitectura de ICE es el SLICE que se vio en el epígrafe anterior y el cual no es más que el lenguaje de especificación de ICE, es decir, juega el mismo papel que el IDL en CORBA. El SLICE es pre-procesado como C++ y admite #include y macros, es por esto que se recomienda protegerse de una doble inclusión si utilizamos el #include. Los comentarios podemos hacerlos como en C/C++ y las cosas se organizan con módulos por lo que todas las definiciones deben estar dentro de uno de ellos. SLICE tiene varios tipos básicos que son definidos por el usuario.

Estos tipos pueden ser:

- Enumerados.
- Estructuras: No se puede definir una dentro de la otra pero sí referenciarse.
- Secuencias: Son vectores de elementos de longitud variable y pueden contener otras secuencias.
- Diccionarios (mapeos key-valor): La llave tiene que ser tipo básico, secuencia o estructura que contenga tipo básico y el valor puede ser definido por el usuario.
- Constantes: Sólo pueden ser tipos básicos o enumerados.

Ahora bien, el punto fuerte del SLICE es la definición de interfaces. Dentro de una definición de interfaz solo pueden declararse operaciones. No se pueden definir tipos, excepciones o estados dentro de la interfaz pero la implementación de la interfaz si puede contener un estado. Un objeto tiene una interfaz y ICE provee un mecanismo para asociar múltiples interfaces. La interfaz define el nivel mas bajo posible ya que para comunicarse hay que invocar a las operaciones de un objeto. Permiten definir si los tipos son de entrada o salida pero si no se específica se pone por defecto de entrada y si hay de varios tipos primero se ponen los de entrada y luego los de salida. SLICE no soporta sobrecarga de ningún tipo, todas las operaciones dentro de una misma interfaz deben tener distinto nombre, incluso si tienen distintos parámetros. Una operación es ídem potent si dos invocaciones sucesivas de la misma tienen el mismo efecto que una sola.

2.2 – Arquitecturas de Software.

El diseño y especificación de la estructura global de los sistemas se ha convertido en un nuevo tipo de problema al cual hay que enfrentarse siempre que se comienza un nuevo proyecto de desarrollo de software, con el tiempo se han ido desarrollando metodologías y técnicas que intentan conseguir esos propósitos y a todas estas se les ha llamado arquitectura de software. Esta área de la ingeniería de software consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software. Establece los fundamentos para que analistas, diseñadores, programadores entre otros, trabajen en una línea común que permita alcanzar los objetivos del sistema y cubrir todas sus necesidades. Define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación ente ellos. Tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales.

En esta investigación se muestran dos arquitecturas de software que pueden ser útiles en la construcción del middleware. La primera de ellas es SOA (Service Oriented Architecture), una tecnología que últimamente se ha mencionado mucho debido a las potencialidades que ofrece al orientar los sistemas hacia un nuevo paradigma que es la utilización de servicios. La otra tecnología se denomina REST (Representational State Transfer) que ha sido muy utilizada en aplicaciones hipermedias como la World Wide Web. A continuación se ofrecen una serie de elementos sobre estas dos tecnologías con le objetivo de que podamos conocer más sobre ellas y su análisis sirva de guía a la hora de seleccionar la más adecuada para el sistema de comunicación.

2.2.1 – Arquitectura Orientada a Servicios.

La Arquitectura Orientada a Servicios se ha transformado probablemente en uno de los conceptos más nombrado en el área de las tecnologías. Según los expertos, las principales empresas del mercado han comenzado a adoptar este tipo de arquitectura y se espera que en la próxima década que la mayoría de las grandes compañías la hayan adoptado. Pero: ¿Qué es una Arquitectura Orientada a Servicios

realmente? La Arquitectura Orientada a Servicios es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requerimientos de software del usuario.

SOA es una arquitectura de software que permite la creación y/o cambios de los procesos de negocio desde la perspectiva de las TI de forma ágil, a través de la composición de nuevos procesos utilizando las funcionalidades de negocio que están contenidas en la infraestructura de aplicaciones actuales o futuras. Proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las actividades de integración y consolidación.

En un ambiente SOA, los nodos de la red hacen disponibles sus recursos a otros participantes en la red como servicios independientes a los que tienen acceso de un modo estandarizado. La mayoría de las definiciones identifican la utilización de Servicios Web en su implementación, no obstante se puede implementar una SOA utilizando cualquier tecnología basada en servicios. Las SOAs están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma subyacente y del lenguaje de programación. La definición de la interfaz encapsula las particularidades de una implementación, lo que la hace independiente del fabricante, del lenguaje de programación o de la tecnología de desarrollo. Con esta arquitectura, se pretende que los componentes software desarrollados sean muy reusables, ya que la interfaz se define siguiendo un estándar.

SOA está constituida por tres partes: un proveedor, un intermediario y un cliente, que no presentan ningún acoplamiento entre ellos. El proveedor ofrece un servicio determinado que el cliente no conoce. El cliente entonces aprende a utilizar el servicio a partir de la información que le ofrece el intermediario, que normalmente simplifica el uso de dicho servicio. El cliente normalmente sólo sabe cómo utilizar el servicio, es decir, como enviar y recibir datos, pero no conoce ningún detalle de su implementación interna.

2.2.1.1 – Conceptos Básicos.

En las Arquitecturas Orientadas a Servicios, el elemento básico es el servicio. Pero únicamente con este concepto, no podríamos diseñar una arquitectura SOA. Cuatro son los elementos esenciales necesarios para la construcción de esta Arquitectura:

- Operación: Es la unidad de trabajo o procesamiento en una arquitectura SOA.
- Servicio: Es un contenedor de lógica. Estará compuesto por un conjunto de operaciones, las cuales las ofrecerá a sus usuarios.
- Mensaje: Para poder ejecutar una determinada operación, es necesario un conjunto de datos de entrada. A su vez, una vez ejecutada la operación, esta devolverá un resultado. Los mensajes son los encargados de encapsular esos datos de entrada y de salida.
- Proceso de Negocio: Son un conjunto de operaciones ejecutadas en una determinada secuencia (intercambiando mensajes entre ellas) con el objetivo de realizar una determinada tarea.

Por lo tanto, una aplicación SOA estará formada por un conjunto de procesos de negocio. A su vez esos procesos de negocio estarán compuestos por aquellos servicios que proporcionan las operaciones que se necesitan ejecutar para que el proceso de negocio llegue a buen término. Por último para ejecutar esas operaciones es necesario el envío de los datos necesarios mediante los correspondientes mensajes.

Además de estos elementos que acabamos de conceptualizar encontramos otros como son:

- Sin Estado: No mantiene ni depende de condición preexistente alguna. En una SOA los servicios no son dependientes de la condición de ningún otro servicio. Reciben en la llamada toda la información que necesitan para dar una respuesta. Debido a que los servicios son "sin estado", pueden ser secuenciados (orquestados) en numerosas secuencias (algunas veces llamadas tuberías o pipelines) para realizar la lógica del negocio.
- Orquestación: Secuenciar los servicios y proveer la lógica adicional para procesar datos. No incluye la presentación de los datos. Coordinación.
- Proveedor: La función que brinda un servicio en respuesta a una llamada o petición desde un consumidor.
- Consumidor: La función que consume el resultado del servicio provisto por un proveedor.

2.2.1.2 – Las Capas de Software definidas por SOA.

- Aplicativa Básica: Sistemas desarrollados bajo cualquier arquitectura o tecnología, geográficamente dispersos y bajo cualquier figura de propiedad.
- De exposición de funcionalidades: Donde las funcionalidades de la capa aplicativa son expuestas en forma de servicios (web services).
- De integración de servicios: Facilitan el intercambio de datos entre elementos de la capa aplicativa orientada a procesos empresariales internos o en colaboración.
- De composición de procesos: Que define el proceso en términos del negocio y sus necesidades, y que varía en función del negocio.
- De entrega: Donde los servicios son desplegados a los usuarios finales.

2.2.1.3 – Facilidades que ofrece.

Primero que todo es importante destacar que SOA permite evolucionar desde una tecnología de software basada en una estructura monolítica hacia una flexible compuesta por módulos. Estos bloques que se unen para armar una solución responden a las necesidades específicas de un cliente lo cual permite una respuesta mucho mejor desde el punto de vista del negocio. Los resultados que se obtienen son superiores, porque es algo que sirve específicamente para un problema de negocio. No es algo genérico, sino algo hecho específicamente a la medida.

Otra ventaja con respecto a antiguas estructuras de software es que estas últimas eran funcionales pero estáticas. Las aplicaciones basadas en SOA tienen un nuevo tipo de software basado en módulos flexibles que incluso pueden apartarse y volver a unir lo que realmente interesa porque no tienen esa estructura monolítica antigua. Por tanto se pueden ofrecer nuevas soluciones que pueden estar sustentadas en un nuevo requerimiento, un nuevo negocio, un nuevo mercado o una nueva legislación.

Incluso SOA permite tomar componentes de software viejos y convertirlos en módulos para que trabajen eficientemente sobre una nueva arquitectura para que sirvan a nuevas necesidades que antes por sí mismos no eran capaces de satisfacer. Estos elementos antes mencionados la proveen de una flexibilidad

que hasta hace poco no existía. Por eso uno de sus mayores logros es que se pueda implementar usando estas aplicaciones antiguas inflexibles, transformando pedazos de esos programas y expresándolos luego como un servicio que es parte de un todo superior. SOA le da un nuevo sentido al antiguo software e incluso le da nuevos usos para que se mantengan vigentes.

Por tanto se puede decir que SOA brinda cuatro ventajas: es a la medida del cliente, es absolutamente modular, más flexible y facilita la extensión de las estructuras antiguas.

2.2.1.4 – Componentes de SOA.

Enterprise Service BUS (ESB):

El ESB es el componente tecnológico central en una arquitectura SOA. Un ESB es un middleware que se preocupa de todos los aspectos de la conectividad e integración entre las aplicaciones clientes, los servicios y datos en la capa de backend. En él reside la responsabilidad de ejecutar el flujo, las orquestaciones y en general, todas las labores que guardan relación con la ejecución de los servicios.

Un ESB debe cumplir con las normas definidas en el patrón de integración VETRO, otro acrónimo, que indica y define las labores del ESB:

- (V)alidation: Se encarga de realizar la validación de la mensajería involucrada, generalmente basada en estándares, como por ejemplo, validar el XML de entrada contra su respectivo XSD u otras validaciones de estructuras del mismo estilo.
- (E)nrichment: Corresponde al enriquecimiento del mensaje inicial del cliente con la información necesaria para la invocación del servicio final. En la mayoría de las ocasiones, este enriquecimiento es generado en llamadas a servicios de negocio, adicionando información al mensaje en tiempo de ejecución.
- (T)ransformation: Es la transformación del mensaje original a algún otro formato que sea requerido por el servicio final. Por ejemplo, utilizar “Xquery” para trasformaciones entre archivos XML, de pipe a XML, etc.

- (R)outing: Consiste en el ruteo del mensaje a diferentes servicios, de acuerdo al contenido de este, a la operación, tablas de configuración, etc.
- (O)perate: Comunicación con el servicio final, es decir, con los servicios de negocio. Para estos efectos se vale de una serie de conectores que cumplen una norma en particular y que solucionan la problemática de conectividad puntual.

Adicionalmente, sobre los ESB podemos agregar:

- Un ESB es distribuido, no corresponde a la arquitectura clásica de los middleware de mensajería, por lo que se puede distribuir en toda la red física.
- Está basado en un sistema de mensajería o Message Oriented Middleware, lo cual permite manejar muchos escenarios de manera asíncrona.
- Permite el monitoreo detallado de niveles de servicio.

Los estándares VETRO trabajando en conjunto, implementan lo que se conoce como **"**Orquestación**"**. En resumen, la orquestación es la implementación de la interoperabilidad, la transformación y control de un flujo de datos a través de los servicios involucrados en el proceso. Generalmente se implementa en el ESB vía BPEL, un lenguaje parecido a un "XML potenciado".

Service Data Object (SDO):

SDO provee una interfaz uniforme para el manejo, de diferentes formas, de datos, incluyendo los representados en estándar XML. Proveen un mecanismo para darle seguimientos a los cambios en los datos que manejan los servicios en la red.

SOA crea un ambiente donde conviven diferentes tipo de datos, lo que implica unas alta complejidad de las aplicaciones. SDO, provee un modelo de programación que simplifica la resolución de este tipo de aplicaciones.

Ventajas:

- Acceso de manera uniforme a los datos de fuentes heterogéneas, dentro de las cuales pueden mencionarse XML, RBD, POJO, SOAP, etc.
- Provee los modelos de programación estático y dinámico.
- Provee metadatos para fácil manejo de tipos.
- Provee un modelo desconectado, donde los datos pueden ser devueltos desde la fuente, vía una interfaz estándar llamada Data Access Service. Los datos pueden ser modificados por lo clientes controlados por un seguimiento de los cambios.
- El modelo de programación es neutral.

Service Component Architecture (SCA):

Una de las características de los sistemas basados en arquitectura SOA, es la habilidad de ensamblar servicios nuevos y existentes para crear una nueva aplicación, que pueden estar implementados en diferentes tecnologías. SCA (Arquitectura Orientada a Componentes), define un modelo basado en servicio para la construcción de ensamblado y despliegue de servicios existentes y nuevos.

SCA presenta un modelo de programación altamente extensible y de lenguaje neutral, puede ser extendido para trabajar con varias variantes como son:

- Múltiples tipos de implementación, incluyendo, Java, C++, BPEL, PHP, Spring, etc.
- Múltiples integraciones incluyendo, web service, JMS, EJB, JSON RPC, etc.
- Múltiples plataformas incluyendo, Tomcat, Jetty, Geronimo, OSGI, etc.

SCA separa la infraestructura de la lógica del negocio y permite a los desarrolladores enfocarse en el negocio. Esto es posible definiendo calidades del servicio como son las fiabilidad, seguridad, manejo de transacciones, etc.

Business Process Management (BPM):

Un conjunto de servicios conformará lo que se denomina un proceso de negocio. Para entender correctamente la relación entre BPM y SOA, hay que tener en cuenta que todo proceso de negocio tiene muchos aspectos que han de ser gestionados, como por ejemplo capacidades de orquestación en los

procesos, modelización del proceso, monitorización del proceso y de su estado, motores de reglas de negocio más amigables para el usuario, análisis de eficiencia de los procesos a nivel granular o agregado. SOA no especifica la gestión de cada aspecto, pero toda infraestructura de servicios requiere herramientas para gestionarlos y así garantizar la eficiencia en la arquitectura planteada.

Para poder comprender la diferencia y la relación existente entre Servicios SOA y Procesos de BPM, es importante primero distinguir Servicio SOA del Servicio que el negocio ofrece al consumidor o cliente. A veces, en las organizaciones, se denomina Servicio a lo que realmente es un proceso de negocio. Este proceso de negocio estaría compuesto de servicios SOA e invocaría a las operaciones de estos servicios. Además los procesos de negocio pueden tener un estado y sin embargo los Servicios SOA no tienen nunca estado.

Service Registry:

Antes de implementar un servicio, hay que diseñar las definiciones de las interfaces y las políticas para la invocación de servicios entre los proveedores de servicios SOA y los consumidores de servicios SOA. El "registro de servicios" permite tal funcionalidad, además de proporcionar un mecanismo de publicación de registros de servicios. Con este mecanismo todos los servicios conocerán de la existencia de nuevos servicios y además se permite verificar que todos ellos utilizan las políticas establecidas, asegurándose la consistencia de los servicios y su calidad.

2.2.1.5 – Barreras a vencer.

La principal limitación que presenta SOA es que constituye un nuevo horizonte para las tecnologías de la información. Como cualquier gran cambio, las principales barreras son organizacionales, no técnicas. Seguidamente podemos ver dos de las más importantes:

Administración: Los servicios compartidos es lo principal para utilizar SOA. La habilidad para ensamblar rápidamente aplicaciones o procesos está basada en la disponibilidad de algunos servicios que pueden ser compartidos. Hacer esto, por definición, requiere administración.

Desarrollo Cultural: Al utilizar SOA se requiere un cambio significativo en el estilo de programar. Muchos desarrolladores utilizan equipos diferentes para resolver problemas de manera independiente para cada aplicación. En SOA necesitarán escribir aplicaciones para ser reutilizadas en mente, usando códigos existentes, a los cuales se podrá tener acceso constantemente.

2.2.2 – Transferencia de Estado Representacional.

La Transferencia de Estado Representacional (REST por sus siglas en Ingles, de Representational State Transfer) es una técnica de arquitectura de software para sistemas hipermedia distribuidos. El término se originó en el año 2000, en una tesis doctoral sobre la web escrita por Roy Fielding, uno de los principales autores de la especificación del protocolo HTTP, desde entonces el término ha sido ampliamente utilizado por las comunidades de desarrollo de software. Originalmente el término se refería a un conjunto de principios de arquitectura pero en la actualidad se usa en el sentido más amplio para describir cualquier interfaz web simple que utilice XML y HTTP, sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes como el protocolo de servicios web SOAP.

Es un estilo arquitectónico que puede utilizarse para guiar la construcción de servicios web ligeros, así como protocolos de aplicación con propiedades “Web-like”. Queda definido por cuatro restricciones de interfaz: identificación de recursos, manipulación de recursos mediante representaciones de estos, mensajes auto-descriptivos, e hipermedia como motor del estado de la aplicación. La aplicación cliente cambia de estado con cada representación de recurso manipulada. Como sucede con otros estilos arquitectónicos como cliente/servidor, REST no representa ningún estándar ni supone ninguna especificación, tan sólo se puede entender su motivación, asimilar su filosofía y sus principios y diseñar las aplicaciones en ese estilo. No es por tanto un estándar, pero sí se basa en estándares como HTTP, URL, XML, XHTML, GIF, JPEG, y tipos MIME. Tampoco implica ningún detalle de implementación. (Uso de Servlets, OO, CGI, Perl, etc.)

Emplea el protocolo HTTP con sus 4 métodos bien definidos Get, Post, Put y Delete, y mensajes auto-descriptivos en algún formato “Web-like”. La mayoría de los mensajes están en XML, estructurado por un esquema escrito en un lenguaje de esquemas como XML Schema del W3C o RELAX NG. Los mensajes

simples pueden codificarse con URL Encoding. Los servicios y los proveedores de servicios deben ser recursos, mientras que un consumidor puede ser un recurso. Hoy en día sólo existe un protocolo que soporte la semántica apropiada de manipulación de recursos (HTTP). Sin embargo, los sistemas basados en REST son realmente independientes del protocolo. Si en un futuro aparece otro, será sencillo conservar el mismo diseño y simplemente soportar la nueva interfaz del protocolo.

2.2.2.1 – Objetivos de la Arquitectura REST.

El estilo de la arquitectura subyacente a la web es el modelo REST. Los objetivos de este estilo de arquitectura se listan a continuación:

- Escalabilidad en las interacciones entre componentes: La red ha ido creciendo de manera exponencial y se ha ido comportado de manera satisfactoria.
- Generalidad en las interfaces: Se proporciona acceso con distintos clientes y con distintos mecanismos de acceso.
- Desarrollo independiente de componentes: Implementaciones cliente y servidor pueden ser desarrolladas en distintos momentos.
- Existencia de componentes intermediarios: Con los cuales existe compatibilidad (como los proxy) que permiten encapsulación e integración de sistemas no web dentro de la misma.

Es el cumplimiento de estas propiedades el que ha hecho exitosa la web y deben por tanto guiar su evolución, pueden diseñarse servicios que funcionen óptimamente en el contexto de la web. Requieren además escaso soporte de infraestructura, aparte de las tecnologías estándar de procesado HTTP y XML.

2.2.2.2 – Principios de Diseño.

Todas las propiedades citadas anteriormente van a ser conseguidas imponiendo una serie de restricciones:

- El estado y la funcionalidad de las aplicaciones se representan por forma de recursos.
- La identificación de recursos se realiza de forma única global mediante Uniform Resource Identifiers (URI). Los recursos identificados con URIs son los objetos lógicos a los cuales se les mandan los mensajes.
- Manipulación de recursos a través de representaciones, luego no se manejan directamente los recursos, sino sus representaciones.
- Todos los recursos comparten un interfaz uniforme formado por:
  - Un conjunto de operaciones limitadas para transferir el estado. Se van a aprovechar de las operaciones que HTTP define.
  - Un conjunto limitado de tipos de contenidos, identificados mediante tipos MIME.
- Uso de un protocolo cliente/servidor, sin estado y basado en capas. Cada mensaje HTTP contendrá la información necesaria para comprender la petición, no es necesario que ésta sea entendida tanto en el cliente como en el servidor.
- Uso de hipermedios para representar el estado de una aplicación, permitiendo que el estado de una aplicación web particular esté almacenado en uno o más documentos de hipertexto que residen bien en el servidor o en el cliente. Esto permite al servidor saber el estado de sus recursos sin necesidad de almacenar el estado de los clientes concretos.

Las ventajas prácticas de seguir estos principios son, al menos dos:

- Los encargados de poner la aplicación en explotación saben con certeza cuándo y dónde pueden guardar en una memoria caché las respuestas a ciertas peticiones, lo que incrementa el rendimiento y disminuye notablemente la latencia.

- Las interfaces para acceder a los recursos que maneja la aplicación están bien definidas, lo que facilita la escritura y el mantenimiento de aplicaciones.

2.2.2.3 – Los Recursos en la Arquitectura REST.

Un concepto muy importante en esta arquitectura es la existencia de los recursos que no son más que elementos de información, que pueden ser accedidos utilizando un identificador global (Identificador Uniforme de Recurso). Para manipular estos recursos, lo componentes de la red: clientes y servidores, se comunican a través de una interfaz estándar (HTTP) e intercambian representaciones de estos recursos que no son otra cosa que los ficheros que se descargan y se envían.

La petición puede ser tramitada por cualquier número de conectores, por ejemplo clientes, servidores, cachés, túneles etc.; pero cada uno lo hace sin ver más allá de su propia petición, esto se conoce como separación en capas, otra restricción de REST, que es un principio común con muchas otras partes de la arquitectura de redes y de la información. Esto posibilita que una aplicación pueda interactuar con un recurso conociendo el identificador del recurso y la acción requerida, no necesitando saber si existen cachés, proxys, cortafuegos, túneles, o cualquier otra cosa entre ella y el servidor que guarda la información. La aplicación debe comprender el formato de la información devuelta, es decir de la representación, que es por lo general un documento HTML o XML, aunque también puede ser una imagen o cualquier otro contenido.

2.2.2.4 – Ventajas y aplicabilidad de REST.

Todo lo visto en los apartados anteriores nos hace llegar a una conclusión final sobre este tipo de arquitectura, y esta no es más que las ventajas que esta arquitectura brinda y por tanto en que situaciones es aconsejable utilizarla. Las principales ventajas son:

- Mejores tiempo de respuesta y disminución de carga en servidor.
- Mayor escalabilidad al no requerir mantenimiento de estado.
- Facilita desarrollo de clientes.
- Mayor estabilidad frente a futuros cambios.
- Permite la creación independiente de los tipos de documentos, sin afectar a los anteriores.

Dichas ventajas aconsejan su uso en una serie de escenarios localizados, estos son:

- Cuando los recursos van a estar disponibles para un grupo de usuarios desconocidos y potencialmente grande.
- Cuando el propósito de la aplicación es mandar información.
- Cuando está previsto que la aplicación crezca continuamente.
- Cuando el uso de recursos hardware no es determinante. (CPU, memoria, ancho de banda, etc.)
- Cuando la gestión del estado de la aplicación es simple.

Existen sin embargo algunas situaciones en las que no será útil:

- Situación donde la seguridad de los datos es determinante.
- Cuando el propósito principal de la aplicación es recibir y procesar grandes cantidades de información.
- Cuando el uso de los recursos hardware es determinante.
- Cuando la gestión del estado de la aplicación no es trivial.

CAPÍTULO 3: PROPUESTA DEL SISTEMA DE COMUNICACIÓN (MIDDLEWARE).

Introducción.

Ha llegado el momento de determinar las características que no deben faltar en un middleware para que pueda ser insertado en diferentes entornos. Antes, se verá de forma más explícita cuál es el papel de los middleware de forma tal que podamos entender su función de una manera más simple. Primero que todo debemos aclarar que los middleware cumplen doble papel, primeramente como infraestructura y en segundo lugar como abstracción de la programación. A continuación se muestran algunas de las características que nos van a ayudar a conocerlo.

Infraestructura:

1. Recoge todo lo necesario para desarrollar y ejecutar sistemas distribuidos complejos.
2. La tendencia es hacia arquitecturas orientadas a servicios a una escala global y a la estandarización de las interfaces.
3. Otra tendencia importante es hacia una única infraestructura para minimizar la complejidad y las iteraciones.
4. La evolución es hacia la integración de plataformas y la flexibilidad en la configuración.

Abstracciones de Programación:

1. Pretenden ocultar los detalles de bajo nivel del hardware, redes y distribución.
2. La evolución es hacia primitivas más potentes que se basan en el concepto básico de RPC, añadiendo propiedades adicionales o permitiendo un uso más flexible del concepto.
3. Su aspecto viene dictado por la evolución de los lenguajes de programación. (RPC y C, CORBA y C++, RMI y Java, SOAP'XML y Servicios Web)

Por todo lo visto anteriormente podemos decir que para lograr un correcto funcionamiento de un producto middleware, sobre todo si queremos que este sea genérico, se deben definir una serie de requisitos que permitan al middleware desarrollar sus funcionalidades básicas. Pero no sólo definiendo sus

características se logra moldear un sistema de esta envergadura sino que es necesario también definir cuál de todas las tecnologías existentes puede satisfacer de una forma mejor las características definidas y por último que arquitectura se utilizará que le permita al sistema de comunicación la genericidad que se necesita. A partir de ahora estará abordando todo lo que el Sistema de Comunicación Distribuido (middleware) debe hacer.

3.1 – Características del Sistema de Comunicación Genérico.

Es importante remitirnos a la situación problémica planteada en la introducción de esta investigación antes de continuar con el análisis para ganar en claridad sobre que es lo que tienen que garantizar las características que estamos buscando. Como bien hemos visto los proyectos de desarrollo del Polo de Automática de la Facultad 5 cada día se van incrementando lo cuál requiere que gran número de recursos sean destinados para esa función. Si en el momento en que se obtiene un contrato con un usuario de define que se necesita un middleware para que gestione las comunicaciones en el sistema a desarrollar, nuestra Facultad ganaría mucho si contara con un middleware genérico de forma tal que sólo deba adaptarse a las características del ambiente distribuido en el que va a ser insertado, así sea en la industria eléctrica, azucarera, petrolera, bancos, servicios web entre otros. Por tanto las características que se definan deben garantizar que el middleware sea flexible y configurable para que pueda adaptarse a la inserción en otras industrias tanto nacionales como extranjeras.

A continuación se señalan las características que el middleware debe poseer y que deben tenerse en cuenta a la hora de su implementación.

3.1.1 – Desarrollo en C++.

C++ es un lenguaje para la programación que se utiliza ampliamente en la industria del software. En el caso de esta investigación se cuenta con varias razones para proponerlo como lenguaje de programación del sistema de comunicación las cuales están muy relacionadas con las posibilidades que brinda y en las cuales hay que detenerse para conocer un poco de él. Ahora bien, hay otro elemento a tomar en

consideración y es que la concepción del proceso docente en la UCI está relacionado con la vinculación de los estudiantes a proyectos productivos. La docencia y la producción deben cooperar en un objetivo común, el de formar al estudiante como futuro profesional y en este sentido es que en la Facultad 5 se ha adoptado que la asignatura de programación se imparta apoyada en el lenguaje C++. Los recursos que tiene este lenguaje son muy útiles en la rama de la automatización que es uno de los polos productivos de dicha Facultad.

Ahora sí hablando de C++, este es un lenguaje imperativo orientado a objetos derivado del lenguaje C pero que se mantiene muy ligado al hardware subyacente manteniendo una considerable potencia para la programación a bajo nivel pero que añade otros elementos con respecto a su predecesor que le permiten también un estilo de programación con alto nivel de abstracción. La definición oficial del lenguaje 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 inline, 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.

C++ no es un lenguaje orientado a objeto puro sino que está desarrollado por programadores de alto nivel para otros programadores lo cual se define como un diseño pragmático al que se le han añadido todos los elementos que la práctica aconsejaba como necesarios. 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.

3.1.2 – Implementado por Herramientas Libres.

Con el propósito de explotar al máximo las herramientas necesarias para el desarrollo del sistema de comunicación es muy importante la utilización del software libre. El concepto de software libre tal y como lo ha definido la Free Software Foundation y su fundador Richard Stallman se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software.

De modo más preciso, se refiere a cuatro libertades que tienen los usuarios del software:

- Libertad 0: La libertad de usar el programa, con cualquier propósito.
- Libertad 1: La libertad de estudiar cómo funciona el programa, y adaptarlo a tus necesidades. (El acceso al código fuente es una condición previa para esto.)
- Libertad 2: La libertad de realizar y distribuir copias.
- Libertad 3: La libertad de mejorar el programa y hacer públicas las mejoras a los demás, de modo que toda la comunidad se beneficie. (El acceso al código fuente es un requisito previo para esto.)

Todas estas libertades del software libre permiten la libertad de hacer modificaciones en dependencia de las necesidades que se tengan y utilizarlas de manera privada en el desarrollo del sistema, sin ni siquiera tener que anunciar que dichas modificaciones existen. Así mismo si se publican los cambios, no hay por qué avisar a nadie en particular, ni de ninguna manera en particular. La libertad para usar un programa significa la libertad para cualquier persona u organización de usarlo en cualquier tipo de sistema informático, para cualquier clase de trabajo, y sin tener obligación de comunicárselo al desarrollador o a alguna otra entidad específica. Es por todo esto que se decide utilizar este tipo de software y no otro que puede resultar mucho más complicada su adquisición y utilización, de esta forma estamos contribuyendo a alcanzar una soberanía tecnológica.

3.1.3 – Comunicación Distribuida.

La comunicación distribuida tiene como base estructural una topología de red distribuida que se caracteriza por la ausencia de un centro individual o colectivo, es decir, los nodos se conectan entre sí sin que tengan que pasar necesariamente por uno o varios centros. Ninguno de estos nodos, vinculados unos a otros, tiene poder de filtro de la información que se transmite en la red. Este tipo de red se caracteriza por su robustez ante caídas de nodos pues ninguno al ser extraído genera la desconexión de otro.

Esto facilita el proceso de comunicaciones en ambientes distribuidos pues la responsabilidad de estas deja de formar parte del tipo de estructura y recae en el sistema que se utilice para gestionar las mismas.

Estos sistemas deben permitir la comunicación entre aplicaciones, sin importar dónde estén localizadas o quién las diseñó. Funciona a través de programas o llamadas a objetos, que se comunican, independientemente del lenguaje de programación en que se escribieron o en que sistema operativo corren.

3.1.4 – Comunicación Bidireccional.

La comunicación, como hemos visto a través de este trabajo, es la función principal de todo middleware y este debe estar preparado en todo momento para dar respuesta a las necesidades de comunicación de cualquiera de los componentes que conforman el sistema. Para comprender la comunicación bidireccional supongamos que contamos con dos entes, uno de ellos juega el papel de emisor que con un determinado código desea enviar un mensaje al otro que cumple el papel de receptor y utiliza para esto un canal de comunicación.

Puede decirse que una comunicación es bidireccional cuando el receptor devuelve al emisor algún tipo de respuesta después de que este último emita un mensaje y espera a que el receptor lo recoja y lo entienda. En dependencia del motivo de la comunicación o del contenido del mensaje este producirá un efecto en el receptor que en dependencia de cual sea lo motivará a responder, o no, el mensaje del emisor.

3.1.5 – Capacidad de Tiempo Real.

Se abordará el tema del tiempo real de forma tal que se pueda entender que es lo que caracteriza a estos sistemas pero primeramente debe conocer cual es el significado de las palabras que dan nombre a esta característica. La palabra *tiempo* significa que el correcto funcionamiento de un sistema no sólo depende del resultado lógico que devuelve la aplicación sino que también depende del tiempo en que se produce ese resultado. La palabra *real* quiere decir que la reacción de un sistema a eventos externos debe ocurrir durante su evolución, como una consecuencia el tiempo del sistema (tiempo interno) debe ser medido usando la misma escala con que se mide el tiempo del ambiente controlado (tiempo externo).

Es por eso que se entiende por Sistema en Tiempo Real (STR) a aquel que interactúa activamente con un entorno con dinámica conocida en relación con sus entradas, salidas y restricciones temporales para darle un correcto funcionamiento de acuerdo con los conceptos de estabilidad, controlabilidad y alcanzabilidad. En otras palabras un STR se define como sistema informático que tiene la capacidad de interactuar rápidamente con su entorno físico, además puede realizar funciones de supervisión o control para su propio beneficio, todos tienen la facultad de ejecutar actividades o tareas en intervalos de tiempo bien definidos y estas tareas son ejecutadas inmediatamente en una forma concurrente para lograr la sincronización del funcionamiento del sistema con la simultaneidad de acciones que se presentan en el mundo físico. Los intervalos de tiempo en que se ejecutan las tareas se definen por un esquema de activación y por un plazo de ejecución.

La mayoría de los STR son utilizados cuando existen requerimientos de tiempo muy rígidos en las operaciones o en el flujo de datos, generalmente son requeridos como sistemas de control en una aplicación dedicada. Teniendo en cuenta que el middleware puede insertarse en una de estas aplicaciones es necesario que este cuente con esta característica también. La predictibilidad es su característica principal. Además se caracteriza por tener que producir una salida, como respuesta a una entrada, en un tiempo determinado el cual debe ser pequeño para que la respuesta temporal sea aceptable.

3.1.6 – Manejo de Alta Concurrencia.

Primero que todo debemos definir que se entiende por concurrencia. En computación la concurrencia no es más que la propiedad de los sistemas que permite que múltiples procesos sean ejecutados al mismo tiempo y que potencialmente puedan interactuar entre si. Generalmente la concurrencia es simulada cuando solo existe un procesador encargado de ejecutar los procesos concurrentes, esto lo logra ocupándose de forma alternada de uno u otro proceso a pequeñísimos intervalos de tiempo y de esta forma simula que se están ejecutando a la vez. Todo esto nos lleva a que los procesos concurrentes pueden ser ejecutados realmente de forma simultánea sólo cuando cada uno es ejecutado en diferentes procesadores.

Todo lo mencionado anteriormente es lo que da la medida de lo importante que es en un sistema distribuido el manejo de la concurrencia debido a que mientras más elementos compongan al sistema más alta será la concurrencia en él lo cual deriva en una mayor interacción entre procesos que se ejecutan concurrentemente, aumentando de esta forma el número de caminos de ejecución lo cual resulta bastante complejo para un sistema. Por tanto el manejo de la alta concurrencia debe prever algunos problemas como pueden ser que no se cumpla la exclusión mutua, el bloqueo mutuo o dead lock, el retraso indefinido o starvation, la espera ocupada, las condiciones de carrera o competencia, la postergación o aplazamiento indefinido, la condición de espera circular y la condición de no apropiación por solo mencionar algunos. Todo esto puede resumirse en que se debe garantizar que dos procesos que tienen una misma sección crítica no se encuentren en ella a la misma vez.

3.1.7 – Prioridades.

Teniendo en cuenta que una de las características propuestas anteriormente es el funcionamiento en tiempo real es muy importante establecer las prioridades dentro del sistema por su importancia en la planificación de los diferentes procesos del sistema para que este sea más eficiente en el uso de los recursos.

Teniendo en cuenta que el middleware debe estar capacitado para funcionar en cualquier entorno las prioridades son muy importantes. Para lograr esto supongamos que se implementa en un sistema de control de una industria y ocurre una falla eléctrica, el middleware debe establecer como prioridad el lanzamiento de una alarma para que se tomen las medidas necesarias. Todo esto es posible gracias a la prioridad ya que esta se encargó de asignarle el procesador al proceso que tenía como función enviar el mensaje de alerta.

Muchos sistemas implementan mecanismos de planificación utilizando prioridades incluyendo algunos sistemas operativos. Tenemos dos formas de prioridades en los procesos, la primera es fija y esta se le asigna al comienzo de una tarea y no se puede cambiar; la otra forma es la prioridad variable que permite cambiarla durante el ciclo de vida del proceso. En los sistemas de tiempo real las prioridades las asignan los usuarios a partir de las especificaciones temporales de los procesos. A la hora de asignar una

prioridad fija existen dos formas de hacerlo, estas puedan ser por frecuencia de ejecución o por urgencia. Ahora bien, existe otro elemento importante y es si el planificador de procesos es expulsivo o no. En el primer caso si un proceso pasa a estar listo para ejecutarse y hay uno de menor prioridad ejecutándose este último es detenido para que se ejecute el de mayor prioridad. En caso de que el planificador sea no expulsivo no se detiene el que se está ejecutando pero si se tiene en cuenta la prioridad del nuevo para organizar la cola de los procesos.

3.1.8 – Manejo de un alto número de Variables.

Esencialmente podemos comenzar diciendo que una variable no es más que una estructura de programación que contiene datos, esta puede contener números o caracteres alfanuméricos y el programador le asigna un nombre único. Mantiene los datos hasta que un nuevo valor se le asigna o hasta que el programa termine. Específicamente las variables son estructuras de datos que como su nombre indica pueden cambiar de contenido durante la ejecución del programa. Cada variable posee un área reservada en la memoria del procesador que puede ser de longitud fija si se define que su tamaño no variará en la ejecución del programa o también pueden ser de longitud variable que es lo contrario de lo explicado anteriormente. Se representan con identificadores que hacen referencia a un lugar de la memoria del programa en donde se almacena un dato, además cada una está asociada a un tipo de dato el cual determina en función del tamaño del mismo la cantidad de bytes que serán necesarios para almacenarla.

Todo lo visto hasta aquí es muy importante pues refuerza los argumentos de por qué esta característica es tan importante. Como se ha dicho en otras partes de este trabajo los sistemas distribuidos generalmente son muy grandes por lo que el intercambio de variables entre un proceso y otro resulta imprescindible. Muchas veces estas variables son adquiridas del campo en dónde está implantado el sistema o puede provenir de una base de dato y además pueden estar disponibles para varios procesos, o sea, ser una variable global y de esta forma ser muy utilizada. Otro elemento es que mientras más elementos conformen nuestra red mayor será el número de variables que deben ser manipuladas y en dependencia del ambiente esta cifra puede incrementarse exponencialmente. Estas requieren un seguimiento especial pues en determinados sistemas una pequeña variable que no se atienda como lo

requiere puede ocasionar grandes daños. Es por eso que el middleware debe estar preparado para el manejo de un gran número de variables por si se necesita en alguno de los sistemas en que se desee incorporar.

3.1.9 – Persistencia en las Conexiones.

Esta es una característica muy importante para el sistema teniendo en cuenta que su principal objetivo es gestionar la comunicación entre los entes distribuidos de cualquier sistema. Debido a esto es muy importante tener en cuenta el estado de las conexiones entre estos elementos. Una conexión persistente es una conexión continua entre un cliente y un servidor. Para que se entienda mejor, en el caso de las conexiones no persistentes se conecta el cliente con el servidor, se realizan una serie de transacciones y seguidamente se finaliza la conexión. En las persistentes esto último no ocurre, manteniéndose la conexión en espera hasta que se realicen nuevas transacciones o hasta que se indique explícitamente que se finalice la misma.

Las conexiones persistentes garantizan que, siempre que haya una interrupción, la conexión entre los elementos sea automáticamente establecida tan pronto como se recupere el sistema. Esto evita que se tengan que reiniciar los componentes para reconocerse nuevamente, debido a que la persistencia asegura que el enlace esté siempre disponible relanzando automáticamente la conexión. Su ventaja radica en esta facilidad precisamente, ya que estas son más óptimas en ciertos aspectos, al no tener que conectar y desconectar cada vez que se hace una transacción.

3.1.10 – Tolerancia a Fallas.

Para comenzar a hablar sobre la característica de tolerancia a fallas se debe comenzar diciendo que un sistema falla cuando deja de proveer un servicio que debe prestar. Existen varios conceptos relacionados con este tema, primero tenemos que una falla es un defecto dentro de un componente de hardware o software, además relacionado a este está el de error que no es más que la manifestación de una falla y por último tenemos una avería que es la no realización de una acción esperada. Las fallas pueden

clasificarse en tres clases, las fallas permanentes que se refiere a la falla total de un componente, tenemos además las fallas transitorias que se corresponden con una falla temporal de un componente y también están las fallas intermitentes que no son más que una falla temporal que se repite con frecuencia.

Las fallas pueden estar relacionadas con diferentes causas, algunas de ellas pueden ser de hardware producidas por la falta de soporte o mantenimiento. Otro tipo de fallas pueden ser de software como la introducción de un virus en el sistema. Podemos encontrar fallas por averías en la red definidas por la modificación del diseño o la topología así como la modificación de protocolos de comunicación. Además de los antes mencionados, podemos agregar los fallos operativos que se refieren en gran medida a deficiencias de funcionamiento. Por último podemos señalar fallas producidas por el medio ambiente tales como los cortes de electricidad debido a determinadas catástrofes naturales.

Por tanto de define como tolerancia a fallas a la propiedad de algunos ordenadores o sistemas de funcionar aún cuando se haya producido una falla en alguno de sus componentes y es algo propio de sistemas que precisan de una alta disponibilidad en función de la importancia estratégica de las tareas que realizan, o del servicio que han de dar a un gran número de usuarios. Estos sistemas o componentes están diseñados de tal forma que en caso de que algún elemento falle, un equipo o procedimiento de respaldo pueda respaldar inmediatamente sin pérdida de servicios. Muchas veces se ha dicho que ningún sistema puede evitar por completo las fallas es por esto que generalmente los sistemas tolerantes a fallas no las prevean sino que las controlan.

Todo esto ha hecho que se empleen diferentes técnicas para evitar este tipo de pérdidas, una de las más conocidas es la configuración de mecanismos de redundancia cuya función principal es duplicar los recursos críticos. Existen varios tipos de redundancia entre ellos encontramos la de información que duplica datos y códigos de corrección de errores, está la redundancia de recursos que agrega equipo adicional para tolerar la pérdida o el mal funcionamiento de componentes, además está la redundancia de tiempo que realiza una acción y si es necesario vuelve a realizarla. Sobre esta característica se hablará más adelante. Este no es el único mecanismo existente sino que existen muchos otros que pueden ser utilizados en dependencia de la situación.

3.1.11 – Seguridad.

Los datos y los recursos de software y hardware son de vital importancia en los sistemas de información por lo que su seguridad constituye un reto para lograr proteger la integridad de los sistemas. Esto indica que durante el desarrollo de todo proyecto de software es necesario tener en cuenta los mecanismos de seguridad que van a ayudar a que los materiales y los recursos sólo sean usados para los propósitos que fueron creados y dentro del marco que se prevea.

Es importante destacar que la seguridad informática puede resumirse por lo general en cinco objetivos.

- Integridad: Consiste en la verificación de que los datos no hayan sido alterados durante la transmisión así sea accidental o intencionalmente.
- Confidencialidad: Asegura que solo los individuos autorizados tengan acceso a los recursos que se intercambian de forma tal que la información sea ininteligible para el resto.
- Disponibilidad: Se encarga del correcto funcionamiento de los sistemas de información garantizando el acceso a un servicio o un recurso.
- No repudio: Constituye una garantía de que en el futuro ninguno de los involucrados pueda negar una operación realizada.
- Autenticación: Asegura que solo los individuos autorizados tengan acceso a los recursos mediante la confirmación de la identidad de un usuario utilizando un control de acceso.

Teniendo en cuenta que en cualquier sistema pueden existir amenazas que nos son más que las acciones que tienden a ser dañinas y que estas últimas se aprovechan de las vulnerabilidades para poner las aplicaciones en riesgo, es necesario que en consecuencia a esto se tomen medidas. A la hora de prever la seguridad del sistema se tiene que tener en cuenta dos aspectos fundamentales. El primero de ellos es la seguridad lógica que es la que se encuentra a nivel de los datos. El segundo aspecto a tener en cuenta es la seguridad en las telecomunicaciones incluyendo las tecnologías de red, servidores del sistema y las redes de acceso entre otras.

Para gestionar todo lo relacionado con la seguridad se pueden emplear diferentes mecanismos dentro de los que pueden mencionarse el cambio de autenticación, el cifrado, la firma digital, el control de acceso, el

tráfico de relleno, el control de encaminamiento, la unicidad entre otros. Lo que si queda bien claro es que la gestión de la seguridad en el sistema comprende dos campos bien grandes, uno es la seguridad en la generación, localización y distribución de la información de forma tal que solo pueda ser accedida por las entidades autorizadas. El otro campo importante es el de la política de los servicios y mecanismos de seguridad para detectar infracciones y emprender acciones correctivas.

3.1.12 – Redundancia.

Siguiendo con el tema de los requerimientos del sistema se pretende incorporar una característica que nos permita la disponibilidad del mismo a pesar de que sea un sistema crítico que tenga que estar funcionando las 24 horas del día y los 365 días del año de forma tal que se minimicen los fallos que puedan afectar el funcionamiento del sistema. Esta característica se hace más necesaria en este sistema debido a la gran cantidad de componentes que pueden conformar el sistema en que el middleware se acople y por tanto más posibilidades de fallos.

Primero que todo se tiene que redundancia en un sistema de comunicación no es más que el factor de dicha comunicación que consiste en intensificar y repetir la información contenida a fin de que el factor de la comunicación ruido no provoque una pérdida fundamental de información. Esta es la razón fundamental por la que los sistemas utilicen determinadas técnicas para garantizar que la información nunca se pierda. Los mecanismos pueden variar en dependencia de la parte del sistema que utilice esta característica, una de estas partes puede ser servidor, donde normalmente se duplican los discos, las tarjetas de red o las fuentes de alimentación. Otro elemento es el suministro eléctrico utilizando algunos componentes que garanticen la estabilidad eléctrica como pueden ser los UPS, generadores eléctricos o una línea independiente de suministro. Pero sin duda alguna dónde más importante es proteger la información es en los componentes de la red, es decir, enrutadores, conmutadores, tarjetas de red, cables de red y líneas de conexión por lo que se utilizan técnicas para configurar la red de forma tal que existan 2 caminos diferentes entre dos componentes para garantizar la redundancia, lo mismo puede realizarse para crear 3 o 4 caminos y de esta forma garantizar redundancia triple o cuádruple.

3.1.13 – Orientado a Servicios.

Teniendo en cuenta que el sistema de comunicación necesita estar preparado para aplicarse en diferentes entornos es necesario que sus funcionalidades se vean desde un punto de vista que facilite esta genericidad a partir de la gestión de las mismas. Es debido a esto que organizar el sistema en servicios puede resultar muy provechoso.

Un servicio es una pieza de software que conforma una serie de estándares de intercambio de información. Estos estándares permiten el intercambio de operaciones entre diferentes tipos de computadoras, apartándose del problema del hardware que utilicen, como así también de los sistemas operativos que estén corriendo en dichos equipos, o de los lenguajes de programación en los que estén escritos. Para mantener su independencia, los servicios, encapsulan la lógica dentro de un contexto. Este contexto puede ser una tarea de negocio, una entidad de negocio o alguna otra agrupación lógica. Cada servicio puede encapsular una tarea realizada por una rutina individual, o un subproceso compuesto de varias rutinas, puede además encapsular la lógica completa del proceso.

El tamaño y alcance de la lógica representada por los servicios puede variar. Además, la lógica de un servicio puede requerir de la lógica provista por otros servicios. En este caso el primero se valdrá de otros para resolver su problema. Para poder interactuar, los servicios deben estar al tanto de la existencia de cada uno. Esto se logra a través del uso de descripciones de servicios. Una descripción de servicio establece, en su forma más básica, el nombre del servicio y los datos esperados y retornados por el servicio. Estas descripciones pueden ser almacenadas en un directorio que se encarga de publicar esta información para el conocimiento del resto de los servicios.

A todo lo anteriormente descrito podemos agregar que en un sistema orientado a servicios, estos pueden gestionarse de forma tal que puedan modificarse en dependencia de las necesidades que se tengan, así como agregarse o eliminarse. Esto permite que estos elementos, que son independientes, puedan reutilizarse. Todo esto garantiza que el sistema pueda ser útil en diferentes ambientes siguiendo este principio de diseño, pues permite que se hagan los cambios necesarios en dependencia de las exigencias del negocio.

3.2 – Selección de la Tecnología.

Teniendo en cuenta toda la información relacionada con las diferentes tecnologías que se utilizan para la implementación de middlewares presentada en este documento y analizando a la vez las características que se han presentado anteriormente y que el sistema debe cumplir, llegamos al punto de hacer la propuesta de la tecnología más adecuada que puede ser utilizada a la hora de desarrollar el sistema de comunicación genérico.

Para comenzar con este proceso es importante que en todo momento se tengan presentes las características que deben cumplirse. Es por ese motivo que como primer elemento se debe decir que teniendo en cuenta que el middleware debe ser implementado con herramientas libres, un propósito de esta investigación fue estudiar tecnologías que cumplieran con esta condición. Además para el análisis de las mismas se tuvo en cuenta que tuvieran especificaciones para el lenguaje C++ y de esta forma se garantizaba otra de las características que debe distinguir al middleware.

A partir de los elementos que se investigaron sobre estas tecnologías también se sometió a análisis las potencialidades que estas ofrecen para establecer la comunicación respondiendo al resto de las características que fueron planteadas. En este proceso 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 entre muchas otras, esta presenta una gran desventaja y es su complejidad. La complejidad se debe a que permite la interoperabilidad de distintos lenguajes, arquitecturas y sistemas operativos además de que hay que utilizar un compilador propio (IDL) que traduce algunos tipos de datos estándares a los tipos del lenguaje de programación que se va a utilizar pero otras veces es necesario utilizar otros tipos de datos que no son los que proporciona habitualmente el lenguaje y es necesario utilizar tipos de datos adaptados de IDL. Otro aspecto 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 de estos sufren restricciones en cuanto a sus capacidades que los objetos normales no tienen.

Tiene además una desventaja más, que no sólo la afecta a ella sino a las diferentes implementaciones que esta tiene, incluyendo las dos que se analizaron en esta investigaciones: ORBit y TAO. Esta desventaja es la incompatibilidad que existe entre las diferentes implementaciones debido, principalmente, a que estas son ofrecidas por diferentes empresas. Aunque las diferencias entre los diversos ORBs 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. Cuestiones como la colocación de librerías o las diferentes formas de implementar la gestión de la concurrencia, hacen difícil la portabilidad del código, además donde el estándar no concreta, las implementaciones pueden variar entre sí, lo que da lugar a molestas incompatibilidades que complican la vida al usuario. Además que es necesario que se sepa que CORBA no es una tecnología en sí sino una especificación que cuenta con muchas implementaciones.

Ahora bien, continuando con el análisis del resto de las tecnologías puede verse que ORBit y TAO poseen prestaciones similares y convenientes para su utilización en un sistema como el que se necesita debido a que ambas son implementaciones de CORBA. La primera de estas se destaca por ser más ligera, por sus cortos tiempos de respuesta y baja utilización de memoria, sin embargo TAO parece estar concebido para el desarrollo de sistemas en tiempo real debido a la eficiente implementación que realiza del estándar CORBA-RT. No obstante debe aclararse que estos dos ORB mantienen algunas de las desventajas de CORBA como es la incompatibilidad con otros ORB. Además se puede decir que ninguno implementa todos los servicios de los que dispone CORBA, 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. 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. Se hace necesario aclarar que ninguna de las implementaciones de CORBA, existentes hasta el momento, implementan todos los servicios, lo cuál afectaría mucho la genericidad del middleware que se necesita ya que esta debe poseer la mayor cantidad de servicios posibles que puedan dar solución a cualquier problema en cualquier ambiente en que se utilice. En el caso de TAO podemos agregar que su equipo de desarrollo está trabajando en lograr mayor solidez con la tolerancia a fallos y agregarle nuevas funcionalidades que le permitan el trabajo en nuevas plataformas y lograr mayor integración con otros ORB. Todo esto

determina que tampoco sea muy conveniente su utilización 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, además es escalable dinámicamente, tiene una entrega de datos determinística entre otras características. 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 aun carecen de varias características esenciales de los middleware para sistemas distribuidos.

Otra de las tecnologías que ha alcanzado un cierto reconocimiento en los últimos tiempos es OpenDDS, esta tecnología como bien se vio en el Capítulo 2 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 especificación en C++ de DDS y de código abierto que implementa varios perfiles del DCPS también se basa en la capa de abstracción de ACE para facilitar la portabilidad e 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 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 genérico.

Por último nos queda analizar la tecnología ICE que también ha cogido mucha fuerza en la actualidad, ha sido desarrollada por varios de los desarrolladores de CORBA con el objetivo de resolver muchos de los problemas de esta última. [Anexo 2] 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, 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 tiene disponible el código fuente. Además una ventaja que tiene ICE sobre TAO, una de las tecnologías antes analizadas, es el rendimiento en cuanto a la velocidad de transmisión de los eventos. 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.

Teniendo en cuenta todos los análisis hechos anteriormente, todo parece indicar que ICE es la tecnología que más facilidades ofrece para dar respuesta a las características definidas en esta investigación y que debe soportar el middleware genérico. Por tanto se define que ICE puede ser la tecnología que se utilice para su desarrollo.

3.3 – Selección de la Arquitectura.

Después de hacer un estudio profundo de las características que deben distinguir al sistema de comunicación así como la tecnología con que se va a dar respuesta a dichas características es hora de tratar un tema tan importante como la arquitectura que va a soportar el desarrollo de dicho sistema a partir de su concepción para la genericidad. Como bien se pudo apreciar, en el Capítulo 2 se describieron dos tipos de arquitectura de software diferentes pero que a nivel mundial se han comenzado a utilizar en muchos sistemas distribuidos por las propiedades que poseen y que pueden utilizarse para la comunicación. No es objetivo de este epígrafe describir nuevamente ambas arquitecturas pero si sería bueno brindar alguna información que pueda servir de guía durante la selección.

Una de las características más resaltante de SOA, es que está basada en contratos, donde el proveedor establece las reglas de comunicación, el transporte, y los datos de entrada y salida que serán intercambiados por ambas partes. Los beneficios son grandes, se incentiva la reusabilidad, la independencia de la tecnología de implementación, el manejo de contratos, entre muchas otras. Por otra parte tenemos que REST es un estilo de arquitectura que ofrece un buen desempeño, escalabilidad y

abstracción de recursos, donde cada petición HTTP contiene toda la información necesaria para responder a la petición, sin necesidad de que el cliente ni el servidor tengan que recordar el estado de su comunicación.

A partir del análisis que se puede hacer teniendo en cuenta lo mencionado anteriormente y la bibliografía consultada sobre ambos temas puede decirse que REST es una buena opción si se necesita que el criterio de escalabilidad tenga mucho peso dentro de la arquitectura porque permite que todos los recursos presenten la misma interfaz a los clientes. Por otro lado se puede ver que SOA está apoyado sobre estándares y especificaciones de amplia madurez por ejemplo WS-Security sin embargo REST no cuenta con una amplia gama de estándares. Si nos referimos a los protocolos de transporte, elemento esencial para los sistemas de comunicación, podemos decir que SOA soporta más protocolos de transportes tales como JMS, SOAP, etc.; REST sólo conoce HTTP.

Teniendo en cuenta las necesidades que debe tener el sistema que se está proponiendo y las valoraciones que se han realizado sobre las dos arquitecturas de software analizadas, se ha decidido utilizar SOA. Esta selección está sustentada en las potencialidades que esta arquitectura ofrece para la creación de software que pueden utilizarse en diferentes entornos debido a las facilidades que da para su construcción y modificación y además da cumplimiento a una de las características propuestas. A continuación mencionamos algunas de las ventajas que SOA nos ofrece.

Desde el punto de vista de las tecnologías SOA supone un marco conceptual mediante el cual se puede simplificar la creación y mantenimiento de sistemas y aplicaciones integradas además propone una fórmula para alinear los recursos tecnológicos con el modelo del negocio y las necesidades y dinámicas de cambio que la afectan. Con SOA obtenemos:

1. Aplicaciones más productivas y flexibles: La estrategia orientada a servicios permite conseguir una mayor productividad de los recursos existentes, como pueden ser las aplicaciones y sistemas ya instalados e incluso más antiguos y obtener mayor valor de ellos sin necesidad de aplicar soluciones de integración desarrolladas para este fin. Permite además el desarrollo de nuevas aplicaciones compuestas que ofrecen capacidades avanzadas y multinacionales con

independencia de las plataformas y lenguajes de programación que soportan los procesos de base. Cómo los servicios son entidades independientes de la infraestructura subyacente una de sus características más importantes es su flexibilidad a la hora del diseño de cualquier solución.

2. Desarrollo más rápido y económico de aplicaciones: El diseño de servicios basado en estándares facilita la creación de un repositorio de servicios reutilizables que se pueden combinar en servicios de mayor nivel y aplicaciones compuestas en respuesta a nuevas necesidades de la empresa. Esto hace posible que se reduzca el costo del desarrollo de soluciones y de los ciclos de pruebas, además se consigue finalizarlas en menor tiempo. El uso de un entorno y modelo de desarrollo unificados simplifica y homogeneíza la creación de aplicaciones desde su diseño y prueba hasta su despliegue y mantenimiento.

3. Aplicaciones más seguras y manejables: Las soluciones orientadas a servicios proporcionan una infraestructura común para desarrollar servicios seguros, predecibles y gestionables. Según van evolucionando las necesidades de negocio, SOA facilita la posibilidad de añadir nuevos servicios y funcionalidades para gestionar los procesos críticos. Se accede a los servicios y no a las aplicaciones y gracias a esto se optimizan las inversiones realizadas en tecnologías potenciando la introducción de nuevas capacidades y mejoras. Puesto que se utilizan mecanismos de autenticación robustos en todos los servicios y estos existen de forma independientes unos de otros y no se interfieren entre ellos, la estrategia SOA permite dotarse de un nivel superior de seguridad.

3.4 – Vista de la Arquitectura.

Desde el punto de vista tecnológico, SOA propone varias capas de servicios que exponen funcionalidad, fundamentalmente de negocio.

En el nivel más bajo, se tendrá una capa de servicios de acceso a información y datos, cuya función es exponer las diferentes funcionalidades y facilidades provistas como son el ESB, GSOAP, UDDI, WSDL si

es para servicios web entre otras. Muchas veces, estos servicios básicos no representan realmente servicios de negocio que aporten real valor al resto de la organización, al menos no por sí solos. En ella sólo se encuentran los servicios de procesos de negocio, más conocidos por servicios controladores o de comunicación. Aquí se gestiona en detalles la interacción de las peticiones que se realizan sobre un servicio, se comprueban los requerimientos de las mismas, asegurándose con las operaciones que el servicio ejecuta.

Es por esto que sobre esta capa, se encuentra una segunda capa de servicios que se denomina de servicios de negocio que componen y enriquecen la funcionalidad expuesta en la capa precedente, con el objeto de agregar valor al sistema desde el punto de vista del negocio. Los servicios contenidos estarán dirigidos a cumplir una tarea del negocio. Con base en estos servicios básicos y de negocio, es posible comenzar el desarrollo del sistema.

En este desarrollo es que se aprovechan los beneficios de un esquema SOA en cuanto a reusabilidad de servicios existentes y adaptabilidad al negocio. Sin embargo, es necesario que en la capa inferior, los servicios también expongan la funcionalidad que resuelven, para que sean aprovechadas por otros servicios.

Además en esta arquitectura se encuentran servicios que se exponen incluyendo su propia lógica de presentación. Este tipo de servicios se publican en la capa de servicios de presentación o aplicación. En ella se coleccionarán los servicios que sus funciones tengan un intercambio directo con la capa de aplicaciones. A continuación se muestra un modelo dónde puede apreciarse de una mejor forma la estructura de las capas antes mencionadas y como se relacionan entre ellas para lograr que el sistema funcione como un todo.

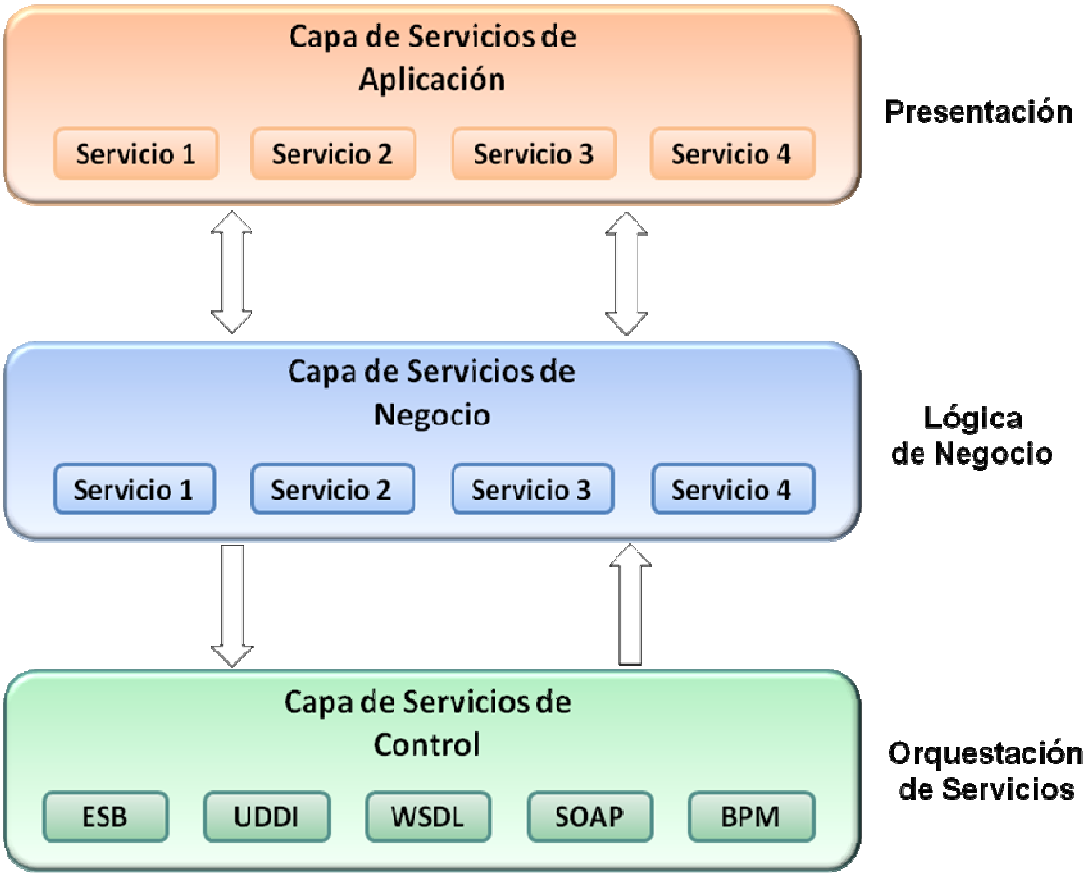

Figura # 3.1: Vista general de la Arquitectura SOA en capas del Sistema de Comunicación.

Es muy importante señalar que los servicios deben poder comunicarse entre ellos debido a que algunos necesitan de otros para poder realizar su función. Además cada servicio debe ser una aplicación completamente autónoma e independiente. Ellos exponen una interfaz de llamado basada en mensajes, capaz de ser accedida a través de la red. Los servicios incluyen tanto lógica de negocio como manejo de estado o datos relevantes a la solución del problema para el cual fueron diseñados. La manipulación del estado es gobernada por las reglas de negocio. La comunicación hacia y desde el servicio, se realiza utilizando mensajes y no llamadas a métodos. Estos mensajes deben contener o referenciar toda la información necesaria para entenderlo.

Uno de los objetivos de la arquitectura del sistema basado en SOA es proveer de la transparencia en la localización de los servicios, es decir, dar la posibilidad de utilizar un determinado servicio que se encuentre en cualquier lugar, sin la necesidad de tener que modificar el código existente. Es por este motivo que en el nivel más alto de esta arquitectura podemos encontrar tres componentes:

- El servicio: Pueden participar cualquier número de servicios. Cada servicio tiene una funcionalidad a la que pueden acceder el resto de servicios y clientes.

- El directorio: El directorio tiene información sobre los servicios y la funcionalidad de estos. Y también tiene la información de cómo se puede acceder a cada servicio.

- El cliente: Usa el directorio para localizar servicios y poder usar su funcionalidad.

De esta forma podemos encontrar tres colaboraciones entre los componentes que pueden llegar a ser una de las partes más importantes de la arquitectura:

- Localización de servicios: Clientes potenciales de los servicios localizan los servicios por medio del directorio. El directorio aporta a los clientes la información sobre como encontrar un servicio.

- Publicación de servicios: Un componente publica un servicio, haciéndolo disponible a los clientes a través del directorio.

- La comunicación entre los servicios y el cliente: El cliente hace peticiones al servicio a través del protocolo de red especificado en la información del servicio que tiene el directorio. El servicio recoge la petición del cliente y le retorna la información pedida.

Todo lo anteriormente expuesto podemos verlo en los diagramas que se muestran a continuación.

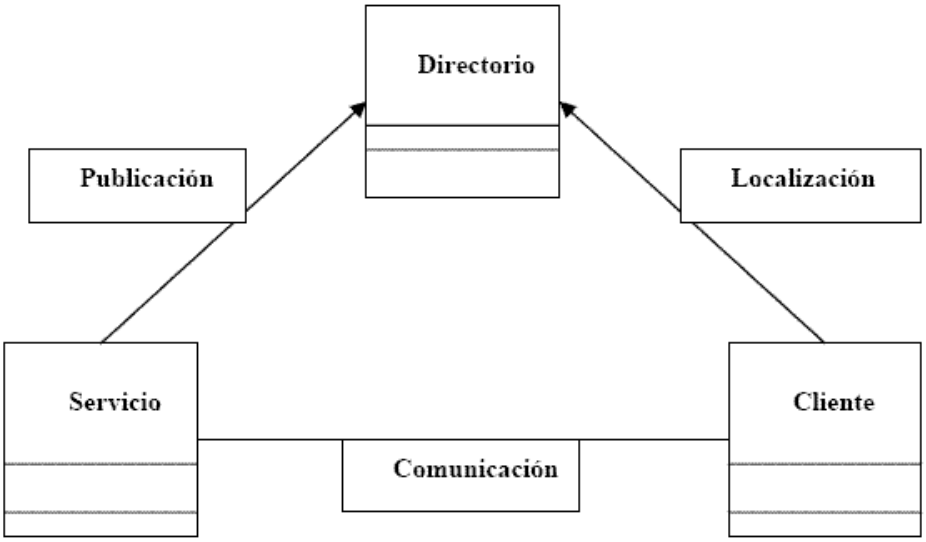

Figura # 3.2: Componentes y colaboraciones en la arquitectura SOA

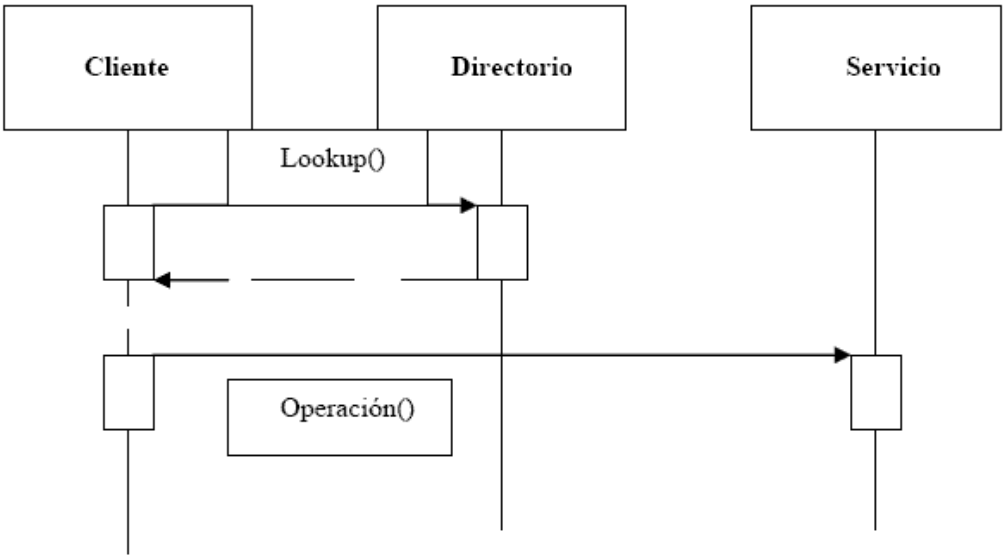

Figura # 3.3: Diagrama de secuencia de las colaboraciones entre los componentes.

A partir de lo visto hasta aquí es que se puede definir en el siguiente modelo la interacción entre los servicios en el sistema de comunicación (middleware).

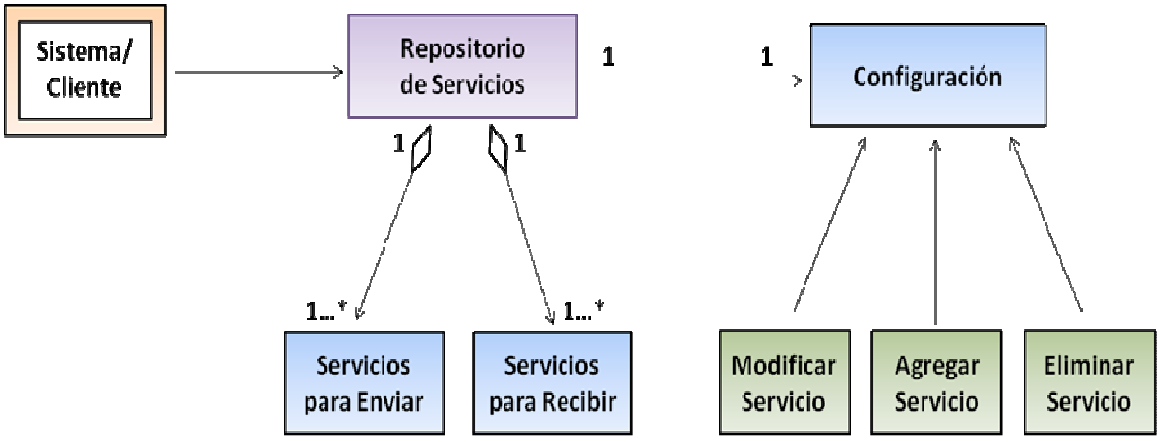

Figura # 3.4: Interacción entre los Servicios del Sistema de Comunicación.

En la figura anterior se puede ver como el cliente hará la solicitud de un servicio al repositorio de servicios del sistema de comunicación. En este repositorio van a estar almacenados tanto los servicios necesarios para el envío como para recibir información necesaria a través del middleware. Además en este repositorio también se encontrarán los servicios de configuración tales como modificar servicios, agregar servicios y eliminar servicios que facilitarán la función genérica de la propuesta realizada.

CONCLUSIONES

Podemos concluir diciendo que durante el desarrollo de este trabajo se transitó por varias etapas que permitieron profundizar en la definición de conceptos fundamentales para la investigación así como caracterizar diferentes tecnologías y arquitecturas a partir de la consulta de una amplia y actualizada bibliografía. Todo esto permitió que se diera cumplimiento a los objetivos y las tareas investigativas planteadas que permiten solucionar la situación problémica presentada.

Por tanto podemos decir que:

- Se definieron las características fundamentales con las que debe contar el sistema de comunicación.

- Se propuso una tecnología y una arquitectura que por sus facilidades pueden ser utilizadas en el desarrollo del middleware.

- Se mostró una vista de la arquitectura del middleware a partir de la utilización de SOA.

Además se puede decir que la presente investigación cuenta con varios resultados, ellos son:

- El presente trabajo representa parte de un entregable pactado por el Proyecto SCADA con la parte Venezolana relacionado con la especificación de una nueva tecnología para implementar la versión 2.0 del Middleware del SCADA.

- La arquitectura planteada está avalada por los desarrolladores del Middleware del SCADA para ser implantada en el momento de desarrollar la versión 2.0 de este sistema.

RECOMENDACIONES

Teniendo en cuenta que la propuesta realizada puede ser muy útil para el desarrollo de los proyectos del Polo de Automatización de la Facultad 5, debido a que la tecnología definida da respuesta a las necesidades del sistema y la arquitectura lo provee de la genericidad necesaria para que este pueda ser utilizado en diferentes ambientes, se recomienda:

- Comenzar el desarrollo del sistema para su posterior utilización en diferentes proyectos.

- Que el trabajo de diploma se utilice como material de consulta por los miembros del Polo de Automatización para su estudio y futuras investigaciones por los temas que en él se abordan.

BIBLIOGRAFÍA

Arosteguy, Corina y Calabró, Carlos. 2005. *SOA: Soluciones para el Futuro.* [En línea] 2005. [Citado el: 10 de Junio de 2008.] http://www.worktec.com.ar/agsi2006/encuentro/SOA_Vision_de_Accenture.ppt.

Astorga Campos, Julián y Quirós Venegas, Juan Luis. *Un enfoque teórico e intuitivo a la arquitectura orientada a servicios (SOA).* [En línea] [Citado el: 11 de Junio de 2008.] http://www.di-mare.com/adolfo/cursos/2007-2/pp-SOA.pdf.

Barco, Antonio. 2006. Blog Arquitectura Orientada a Servicios (SOA). *Elementos esenciales de una Arquitectura Orientada a Servicios.* [En línea] 9 de Mayo de 2006. [Citado el: 3 de Marzo de 2008.] http://arquitecturaorientadaaservicios.blogspot.com/2006/05/elementos-esenciales-de-una.html.

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.

2006. Blog Día a Día. *REST y SOAP.* [En línea] 1 de Febrero de 2006. [Citado el: 5 de Junio de 2008.] http://dia10.blogspot.com/2006/02/rest-y-soap.html.

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.

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.

Cejas, Julio. 2007. Mijao Blog. Incentivando la Innovación. *SOA vs REST.* [En línea] 30 de Junio de 2007. [Citado el: 5 de Junio de 2008.] http://mijao.blogspot.com/2007/06/soa-vs-rest.html.

Cubo Velázquez, Alberto. *Representational State Transfer (REST). Un estilo de arquitectura para Servicios Web. Panorámica y estado del arte.* [En línea] [Citado el: 5 de Junio de 2008.] http://trajano.us.es/~antonio/proyecto-REST.doc.

Deloitte Touche Tohmatsu. 2007. Deloitte. *La arquitectura orientada a servicios.* [En línea] 23 de Octubre de 2007. [Citado el: 3 de Marzo de 2008.] http://www.deloitte.com/dtt/press_release/0,1014,sid%253D17009%2526cid%253D176670,00.html.

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.

Duré, Octavio. *SOA: Un Modelo de Dominios que Ataca Todos los Aspecto de una Organización.* [En línea] [Citado el: 11 de Junio de 2008.] http://www.willydev.net/InsiteCreation/v1.0/descargas/soa/willydev_nota_revista_code.pdf.

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.

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.

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.

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.

Gutiérrez Gómez, Isaac y Otón Tortosa, Salvador. *Arquitecturas Orientadas a Servicios.* [En línea] [Citado el: 11 de Junio de 2008.] http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-132/paper09.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.

—. 2007. 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.

Schmidt, 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.

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.

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. Wikipedia. La enciclopedia libre. *Arquitectura orientada a servicios.* [En línea] [Citado el: 3 de Marzo de 2008.] http://es.wikipedia.org/wiki/Arquitectura_orientada_a_servicios.

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

—. 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 1: Visión General de las Implementaciones Disponibles de CORBA.

1. BEA Tuxedo: Un ORB comercial compatible con CORBA 2.5 para Java y C++ de BEA System.
2. Borland Enterprise Server, VisiBroker Ed: Un ORB comercial compatible con CORBA 2.6 para Java y C++ de Borland.
3. Combat: Un ORB Tcl y una capa Tcl para ORBs C++.
4. Fnorb: Un ORB CORBA 2.0 para Python.
5. ILU: Un sistema de interfaz de objetos abierto de Xerox PARC.
6. GNU Classpath: Contiene Software Libre (GPL + Linking Exception) Implementación de Java.
7. IIOP.NET: Un ORB libre (LGPL) de Microsoft .NET.
8. JacORB: Un ORB libre (LGPL) implementado en Java.
9. J-Integra-Espresso: Un ORB en Microsoft .NET comercial por Intrinsyc J-Integra.
10. MICO: Un ORB libre (LGPL) implementado en C++.
11. omniORB: Un ORB libre (LGPL) para C++ y Python.
12. PrismTech's OpenFusion CORBA: Soluciones CORBA a nivel de empresa en C, C++, Java, Java Real-Time y ADA para sistemas embebidos en tiempo real; incluyendo servicios CORBA, capacitación, consultoría y grado de apoyo industrial.
13. OpenORB: Un ORB libre (BSD) para Java.
14. Orbacus: ORB comercial en C++ y Java de IONA Technologies.
15. ORB express: ORBs estándar en tiempo real comercial en ADA, C++ y Java de Objetive Interface System.
16. ORBit2: Un ORB libre (LGPL) para C, C++ y Python.
17. Orbix: ORB comercial de IONA Technologies.
18. OpalORB: Una implementación CORBA escrita completamente en Perl.
19. Perl ORB: Un ORB de código abierto implementado en Perl.
20. PolyORB: Un ORB libre (MGPL) implementado en ADA.
21. Python ORB: Un ORB libre (Python License) implementado en Python.

22. SANKHYA Varadhi: Un ORB comercial para C++.
23. TAO: EL ORB de ACE, un ORB de código abierto para C++.
24. VBOrb: Un ORB libre (LGPL) para Visual Basic.
25. ORBLink: Un ORB comercial para Allegro Common LISP.
26. Clorb: Para Common LISP.
27. R2CORBA: Un ORB de código abierto para Ruby.
28. OiL: Un ORB libre en LUA, actualmente soporta sólo una parte de la especificación CORBA.
29. TIDorbfor Java: Un ORB de código abierto basado en la implementación 2.6 de CORBA para Java de la Comunidad Morfeo.

Anexo 2: ICE vs CORBA.

A continuación se muestra una comparación por aspectos entre el estándar CORBA y la tecnología ICE que puede apoyar la selección de la tecnología que se realizó en el Capítulo 3. Esta comparación ha sido tomada de: [ZeroC Inc., 2008]

Exhaustividad:

Cuando decimos integridad, nos referimos a la exhaustividad de los productos en el mundo real, no a la exhaustividad de especificaciones que nunca se aplicaron. Creemos que ICE es más completo que cualquier producto CORBA único en el mercado. Échale un vistazo usted mismo: ¿Cuál producto CORBA ofrece realmente un conjunto de características comparables a las de ICE?

Rendimiento:

Debido a las ventajas de la arquitectura CORBA que no puede igualar, ICE tiene un desempeño destacado. El protocolo de alta eficiencia de ICE, solicitud en lotes, y la eficiente transmisión de eventos (entre otras características) significa que funciona más rápido y consume menos ancho de banda que un ORB de CORBA.

No “Comité de Diseño”:

ICE fue diseñado por un pequeño grupo de dedicadas y altamente experimentadas personas. ICE no intenta ser todo para todos. Las especificaciones de CORBA, por otro lado, están llenas de “material de fantasía”, empujadas dentro de las especificaciones de los intereses creados, sin realmente hacer seguro que las especificaciones puedan ser realmente implementadas.

A menudo, la única forma de llegar a un acuerdo durante el proceso de envío de una especificación CORBA, es para tomar la gran unión de las características fijas de todas las implementaciones propietarias preexistentes y de alguna forma calzarse dentro de un estándar. Esto se traduce en especificaciones que son mucho mayores y mucho más complejas de lo necesario. Esto significa que la plataforma es más grande y más lenta, y que el fichero es mucho más difícil de utilizar en las complejas APIs resultantes. ICE proporciona APIs mucho más pequeñas, más eficientes y más fácil de aprender que las APIs equivalentes de CORBA. Sin comprometer la funcionalidad.

Slice:

Slice, el lenguaje de especificación de ICE, es más pequeño, más limpio y más potente que el IDL de CORBA. Slice tiene menos lenguajes pero construye una mayor flexibilidad general. Por ejemplo, un diccionario incorporado proporciona apoyo directo para el acceso rápido a las estructuras de datos, excepciones y la herencia que permite asignaciones limpias para lenguajes con una función de manejo de excepciones. Al mismo tiempo, Slice acaba con muchas de las complejidades innecesarias del IDL de CORBA, como atributos, parámetros “in-out”, contextos, y las complejidades de objetos por valor.

Asignación de Lenguajes:

ICE apoya C++, Java, Python, Ruby, PHP y .NET. No hay conocimiento de ninguna empresa CORBA que ofrezca tantas elecciones. De hecho, la mayoría de los vendedores de CORBA sólo ofrecen C++ y Java. Si se desea utilizar otros lenguajes, hay que cambiar a diferentes proveedores, o estudiar la posibilidad de uso de implementaciones experimentales de CORBA.

Persistencia:

Slice no es sólo un lenguaje de definición de interfaz. También se puede utilizar para describir el estado de la persistencia de objetos ICE, por lo que es fácil escribir los servidores que almacenan automáticamente estados de objetos en una base de datos.

Metadatos:

Slice apoya un servicio de metadatos extensible, que permite la marcación de construcciones Slice para aplicaciones con propósitos específicos. Por ejemplo, los metadatos se pueden utilizar para personalizar el mapeo del lenguaje Java para satisfacer las necesidades de una aplicación particular.

Núcleo ICE:

El núcleo CORBA ha llegado a ser muy complicado a lo largo del tiempo. Un buen ejemplo es el adaptador de objetos portátiles que requiere conocimientos especializados para utilizarse correctamente, aunque sólo hay un puñado de técnicas de implementación recurrentes que deben ser soportadas. El adaptador de objetos de ICE por otra parte, es simple y sencillo, y sólo tan potente como el POA: unas APIs bien diseñadas hacen un corto trabajo de lo que, con el POA, puede convertirse en un largo proyecto de programación.

Protocolo de ICE:

IIOP es uno de los puntos más débiles de CORBA, con demasiados defectos de diseño para hablar de todos ellos. Para nombrar algunos: la falta de encapsulamiento impide solicitar los servicios de transmisión, como el servicio de eventos, de trabajo sin tener un conocimiento profundo de todos los tipos que participan; ineficiente alineación de las normas vigentes innecesarias en la copia de datos, las reglas de codificación de datos son complejas y sin ningún tipo de tratamiento concomitante para ganar en rendimiento, la codificación de referencia de objetos es muy compleja, la prevención de cálculo eficiente y las implementaciones de memoria de cálculo, la negociación de código es bajo especificaciones y sufre de condiciones de competencias, toda esta complejidad significa que IIOP es difícil de implementar, resultando en la interoperabilidad y los problemas de rendimiento. El protocolo de ICE es simple y más

eficientes, y ofrece características como la compresión de datos y solicitudes batched. (Con IIOP no se puede soportar)

Seguridad:

La seguridad ha sido siempre uno de los mayores problemas de CORBA. El OMG ha pasado varias iteraciones de los controles sobre las especificaciones en papel que todavía no tienen implementaciones ampliamente disponibles, y clientes CORBA todavía no tienen ORBs viables y seguros. En el diseño de ICE, por otra parte, la seguridad siempre ha sido considerada esencial. Es por eso que ICE ofrece SSL fuera de la caja y ofrece una flexible y no intrusiva solución cortafuegos que funciona de verdad.

C++ Mapping:

El trabajo con CORBA y C++ es muy difícil, incluso para los experimentados desarrolladores de C++. Existen innumerables trampas y escollos en lo que respecta a la gestión de memoria y seguridad en las excepciones. En contraste, el C++ mapping de ICE es muy sencillo y directo. Es prácticamente imposible tener pérdidas de memoria a causa de errores. El número de normas a tener en cuenta es infinitamente menor que las reglas del C++ mapping de CORBA, y el C++ mapping de ICE se basa en el estándar de la industria STL.

Escalabilidad:

CORBA es una tecnología muy escalable a condición de que usted sea un experto. Con ICE cualquiera puede escribir una aplicación altamente escalable. Por ejemplo, ICE implementa un patrón “evictor” persistente, que le permite manejar fácilmente millones de objetos. Todo lo que tienes que hacer es especificar los datos del objeto persistente en Slice, y dejar que ICE haga el resto: el tiempo de ejecución de ICE carga automáticamente y guarda objetos utilizando una base de datos de alta velocidad.

Versionado:

CORBA no tiene ningún mecanismo para el soporte de versiones de estados de objetos. Freeze, el servicio de persistencia para ICE, permite una fácil migración de bases de datos cuando la descripción Slice de los cambios de datos es persistentes.

Actualizaciones de Software:

Ice Patch es una herramienta que le permite mantener el software del cliente actualizado, utilizando la compresión para una eficiente transmisión de datos y la comprobación para garantizar la coherencia. CORBA no ofrece ningún mecanismo para distribuir actualizaciones de software en torno a un sistema distribuido.

Servicio de Eventos Typed:

CORBA tiene una especificación para un servicio de eventos typed, pero hay pocas implementaciones. El servicio de eventos typed tiene muchos problemas conocidos, lo que prácticamente no utilizados en despliegues en el mundo real. ICE fue diseñado para soportar los servicios de eventos typed desde el primer día. Ice Storm es una eficiente implementación de servicios de eventos typed que apoya también la federación.

Facetas:

CORBA apoya la herencia, DCOM apoya la agregación. En el pasado hubo muchos debates acerca de cuál es el mejor enfoque. ICE soporta ambos: herencia de interfaces mas agregación en forma de facetas. Las facetas le permiten ampliar los tipos en tiempo de ejecución usando la agregación dinámica en lugar de la herencia estática.

Mensajes Asíncronos:

CORBA soporta la Invocación Asíncrona de Mensajes (AMI), pero muy pocos productos CORBA aplican AMI. ICE soporta AMI desde el primer día, de forma sencilla y eficiente. ICE también apoya el Envío Asíncrono de Mensajes (AMD), que no tiene ningún equivalente en CORBA. AMD es del lado del servidor equivalente a AMI en los clientes. Con AMI puede enviar una solicitud, y posteriormente, obtener una devolución de llamada cuando se devuelve el resultado del servidor. Con AMD, puede devolver el hilo enviado de ICE, y retornar la llamada cuando el resultado está listo para ser entregado al cliente. AMI y AMD se pueden encadenar, lo que le permite construir routers muy eficientes con un mínimo consumo de recursos.

GLOSARIO DE TÉRMINOS

- **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.
- **Firewall:** Cortafuego. Herramienta del sistema operativo que mantiene la seguridad de acceso a través de la red a las computadoras.
- **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.
- **Free Software Foundation (FSF):** Entidad que busca eliminar las restricciones de uso, copia, modificación y distribución del software.
- **General Public License (GPL):** Esta licencia regula los derechos de autor de los programas de software libre (free software) promovido por la FSF en el marco de la iniciativa GNU.
- **Hilos:** Los hilos son las distintas partes en las que un programa se puede dividir para ejecutarse en varias tareas simultáneas o casi simultáneas.
- **Mensajes:** Mecanismo de comunicación entre los servicios. Los datos que componen el mismo están contenidos en formato XML Schema.
- **Plug-in:** Funcionalidad que se le puede agregar a un programa.
- **Simple Object Access Protocol (SOAP):** Protocolo basado en mensajería XML definido por W3C. Usado para codificar la información tanto de solicitud como de respuesta en los servicios web.
- **Tecnología de Información (TI):** Término muy general que se refiere al campo entero de la tecnología informática - que incluye hardware de computadoras y programación hasta administración de redes.
- **Universal Description, Discovery and Integration (UDDI):** Protocolo que permite descubrir servicios web existentes en Internet y conocer su descripción para saber qué tarea realizan.
- **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.
- **Web Services Description Language (WSDL):** Describe la manera adecuada de utilizar un servicio web desde otros programas.
- **XML Schema:** Lenguaje para definición de los tipos de datos contenidos en un documento XML. Permite luego validar que un documento XML está bien construido. Permite utilizar las mismas tecnologías para su uso.