Universidad de las Ciencias Informáticas
Facultad 5

INTERFACES DE COMUNICACIÓN DISTRIBUIDA ENTRE COMPONENTES DE UN SISTEMA DE SUPERVISIÓN Y CONTROL

Trabajo final presentado en opción al título de
Máster en Informática Aplicada

Autor: Ing. Maikel Pérez Javier

Tutores: Dr.C. Carlos Eulalio Novo Soto
Ms.C. Moisés Herrera Vázquez

La Habana, Julio 2015

Agradecimientos

A todos los que de alguna manera me apoyaron en este empeño.

A mi familia, esposa, tutores, amigos, miembros del equipo de proyecto del SCADA y de la línea de Comunicaciones del propio proyecto.

A mis compañeros del Centro de Informática Industrial en general.

Declaración Jurada de Autoría

Declaro por este medio que yo Maikel Pérez Javier, con carné de identidad 81080901441, soy el autor principal del trabajo final de maestría “Interfaces de comunicación distribuida entre componentes de un sistema de supervisión y control”, desarrollada como parte de la Maestría en Informática Aplicada y que autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Y para que así conste, firmo la presente declaración jurada de autoría en Ciudad de La Habana a los _____ días del mes de _____ del año _____.

_____

Firma del maestrante

Resumen

Con el objetivo de disminuir los tiempos de transferencia de los datos entre los componentes distribuidos de un sistema SCADA basado en código abierto, se ha desarrollado un conjunto de interfaces de comunicación que posibilitan el funcionamiento de las aplicaciones en plataformas heterogéneas, tomando como base la tecnología Middleware ICE de ZeroC. La solución propuesta brinda una arquitectura flexible, que abstrae en todo momento a los clientes de los mecanismos de comunicación y la tecnología utilizada, que unido a la tecnología ICE garantizan la escalabilidad de la solución y una baja latencia en la transmisión de los datos.

La inclusión de la regla que define “que los tipos de datos de todo el sistema son definidos por el Middleware”, elimina la necesidad de conversión de los datos a cada lado de las comunicaciones, por lo que también contribuye a la reducción del tiempo en el proceso de transferencia de la información.

Entre las principales funcionalidades que brindan las interfaces se encuentra el envío y recepción en forma desacoplada de variables, alarmas, comandos y eventos en tiempo real. También permite realizar llamadas a procedimientos remotos implementados por servidores como el de Seguridad y Configuración. Se incorporan además, mecanismos de tolerancia ante fallas basados en la persistencia de las conexiones y redundancia de los servicios.

Finalmente se obtiene un subsistema de comunicación multiplataforma que puede ser instalado en cualquier sistema SCADA basado en código abierto, sin realizar grandes modificaciones, y como otro aspecto relevante, la arquitectura permite ser implementada sobre las base de otras tecnologías similares a ICE.

Palabras Claves: comunicación, latencia, middleware, SCADA.

Índice de contenidos

Índice de figuras

Índice de tablas

Introducción

En los inicios de la automatización, los sistemas que se utilizaban para el control de procesos eran tecnológicamente simples. Con el tiempo fue aumentando la complejidad de estos sistemas de manera exponencial. Para la industria, la productividad es un factor crítico, minutos de paro, se pueden traducir en miles o hasta en millones de pesos de pérdidas o de ventas no realizadas, así como pérdidas humanas. Es por ello que desde hace unos años tienen gran popularidad los sistemas SCADA, acrónimo de *Supervisory Control and Data Acquisition* (en español, Supervisión, Control y Adquisición de Datos), los cuales tienen como función principal, el monitoreo y control de los procesos industriales. Los sistemas de este tipo pueden aplicarse en diversas ramas de la industria, desde el control de tráfico, hasta el monitoreo, supervisión y control de plataformas petroleras. Las aplicaciones SCADA se utilizan entonces en todo proceso donde se exige un control de calidad, producción y optimización del servicio.

Petróleos de Venezuela Sociedad Anónima (PDVSA), es la corporación estatal de la República Bolivariana de Venezuela a cargo de la exploración, producción, transporte y comercialización de los hidrocarburos. Esta corporación empleaba para el control de sus instalaciones diversos sistemas SCADA suministrados por compañías privadas encargadas de brindar la seguridad y eficiencia operacional de los sistemas automatizados. Dichas compañías tuvieron un papel protagónico en el sabotaje petrolero acaecido durante diciembre del 2002 y enero del 2003 en la que se intervinieron y descontrolaron sistemas automatizados que garantizaban la distribución del crudo y sus derivados, y el bloqueo de diversos servicios tecnológicos esenciales (PDVSA, 2009).

En el contexto de las relaciones entre Cuba y Venezuela y debido a la necesidad en este hermano país de independizarse tecnológicamente se comenzó el desarrollo conjunto de un sistema SCADA utilizando herramientas de código abierto, con el objetivo de sustituir los ya existentes por uno de elaboración nacional, que eliminaría la excesiva dependencia de sistemas propietarios, los cuáles en muchos casos no cubren las expectativas de diseño del usuario final.

En la Universidad de las Ciencias Informáticas se comenzó a trabajar en el SCADA a inicios del 2007 y para el 2008 ya se habían implementado un grupo de componentes para satisfacer la arquitectura del sistema.

La arquitectura propuesta para el SCADA fue la de un sistema completamente distribuido, *"Sistema en el cual componentes de hardware y software, localizados en computadores en red, se comunican y coordinan sus acciones sólo por el paso de mensajes"(Coulouris, 2009)*. Se puede resumir que un sistema distribuido no es más que un conjunto de elementos, tanto de procesamiento como de almacenamiento ejecutándose en diferentes nodos físicos de la red y que para un usuario se comportan como un único computador. En este punto, todos los componentes de la arquitectura debían ser desarrollados sobre herramientas de código abierto logrando así el principal objetivo del proyecto, obtener una solución propiamente Venezolana, de código abierto y establecerla así como el SCADA Nacional.

Los sistemas SCADA representan sistemas complejos y su potencialidad, en gran medida, depende de la solución de comunicación distribuida que se implemente. Los sistemas de comunicación distribuida (Middleware), en el caso de los sistemas de supervisión y control, se encargan de gestionar las comunicaciones entre los diferentes procesos distribuidos de mediano y alto nivel que forman parte del sistema (Vázquez, 2008).

El proceso de intercambio de información y/o peticiones remotas entre los componentes de un SCADA, está dirigido a cumplir con la distribución de los datos en tiempo real, es por eso el impacto que tiene el Middleware en el rendimiento del sistema.

Las soluciones SCADA desarrolladas en el Centro de Informática Industrial (CEDIN) en su mayoría son dedicas a la supervisión y control del proceso de producción de petróleo aunque pueden utilizarse sin problemas en otros procesos industriales. En el caso del petróleo siempre será un proceso crítico donde se maneja información en tiempo real y donde los datos deben mostrársele al operador según las restricciones de tiempo establecidas, tiempos de muestreo pequeños en el orden de los milisegundos. Donde se establece por requisitos el envío 50 mil variables por segundo hacia un único cliente y hacia varios, al menos cinco mil variables por segundo.

El Middleware para este SCADA fue implementado en su primera versión sobre la tecnología TAO (*The Ace ORB)*, una de las principales implementaciones del estándar CORBA (*Common Object Request Broker Architecture*). Después de un período en explotación con la implantación de varios pilotos del SCADA Nacional, desde el 2008 bautizado como SCADA Guardián del ALBA, fueron detectados algunos problemas de rendimiento durante la transferencia de los datos entre los componentes del sistema, que llevaron al equipo a realizar a revisión exhaustiva de la solución de comunicación, identificándose una serie de **problemáticas:**

1. A partir de realizar un grupo de pruebas a la tecnología TAO, en función de la transferencia de grandes cantidades de datos y en un ambiente ideal, en correspondencia con el ambiente donde debía funcionar el SCADA Guardián del ALBA, se pudo determinar que la tecnología no garantizaba el envío de cinco mil variables por segundo hacia varios clientes como se puede apreciar en el Anexo 1, lo que provocaba un aumento del tiempo de transferencia de los datos y de esta manera que se viera afectado el rendimiento del sistema.
2. Esta solución de Middleware se soportaba por una arquitectura en tres capas sobre la tecnología TAO, con un alto grado de desacople, que garantizaba la abstracción de los mecanismos de comunicación y la tecnología utilizada a los diferentes subsistemas, sin embargo, el tiempo de envío y recepción de mensajes era lento producto al esquema de conversión de los tipos de datos. La capa de comunicación debía recibir la información de un cliente, convertirla al tipo de dato que maneja la tecnología y posteriormente, antes de ser entregada al servidor, volver a convertirla al lenguaje que entendía este último. Si al proceso anterior se le suman las demoras causadas por el intercambio de mensajes entre las capas de la arquitectura, que aunque en menor medida, también influyeron, unido a las deficiencias propiamente de la tecnología TAO, entonces se originó un incremento en los tiempos de transferencias de los datos y por consiguiente una disminución del rendimiento en el sistema.
3. Como respuesta a las problemáticas anteriores fue desarrollado por parte de un equipo venezolano un nuevo Middleware, siguiendo una arquitectura similar a la anterior pero basado puramente en socket. Aunque esta solución se ha mantenido hasta la fecha, aun presenta problemas de rendimiento y su diseño poco escalable impide agregar nuevas funcionalidades, siendo hoy una petición del cliente y del

equipo de proyecto, el desarrollo de una nueva capa de comunicaciones que además incluya políticas de calidad de servicios (*QoS*) que hagan del Middleware una solución de comunicación más robusta y fiable para aplicaciones SCADA basadas en código abierto.

Atendiendo a los inconvenientes descritos, surge el siguiente **problema científico:** ¿Cómo disminuir los tiempos de transferencia en el proceso de intercambio de datos entre los componentes distribuidos de un sistema SCADA basado en código abierto?

Dicho problema se enmarca dentro de los Mecanismos de Comunicación Distribuida **como objeto de estudio.**

Para resolver este problema **se planteó como objetivo general:** Desarrollar interfaces de comunicación distribuida, que contribuyan a disminuir los tiempos de transferencia de datos en el proceso de intercambio de información entre los componentes de un sistema SCADA basado en código abierto.

El campo de acción sería las Interfaces de Comunicación Distribuida entre los componentes de un SCADA.

Como hipótesis científica se precisa que, si se introducen interfaces de comunicación distribuida, en el proceso de intercambio de información entre los componentes de un SCADA basado en código abierto, entonces se podrá disminuir el tiempo de transferencia de los datos.

Para dar cumplimiento al objetivo se plantearon las siguientes **tareas de investigación:**

- Establecimiento de los fundamentos teórico-metodológicos para el desarrollo de sistemas Middleware en aplicaciones SCADA.
- Caracterización de los mecanismos de comunicación en sistemas SCADA.
- Selección de la tecnología adecuada que servirá como base para desarrollar las interfaces de comunicación distribuida entre los componentes del SCADA.
- Diseño de las interfaces de comunicación distribuida entre los módulos del SCADA.
- Implementación de las interfaces de comunicación distribuida para el SCADA.
- Validación de la solución a partir del diseño de pruebas experimentales y método estadístico.

Aporte práctico

Interfaces de comunicación distribuida capaces de brindar mecanismos de comunicación que garantizan, cumplir con las restricciones de tiempo durante el proceso de intercambio de información entre los componentes de un SCADA basado en código abierto.

Con el objetivo de dar cumplimiento a las tareas de investigación se emplearon un grupo de **métodos y técnicas.**

A nivel Teórico:

- Analítico – Sintético: con el empleo de este método en la caracterización de los sistemas SCADA y los mecanismos de comunicación utilizados, se logró identificar los aspectos más relevantes relacionados con la comunicación en estos ambientes. Permitió caracterizar las tecnologías y tendencias Middleware de mayor impacto, y conocer su comportamiento frente a varias características relevantes.
- Inductivo – Deductivo: el empleo de este método permitió descartar un grupo de tecnologías de comunicación que por sus características no aportan elementos significativos para el desarrollo exitoso de la investigación. Además hacer generalizaciones acerca de las tecnologías candidatas y seleccionar la tecnología adecuada para implementar la propuesta de interfaces de comunicación para el SCADA, utilizando como apoyo una matriz de decisión.
- Modelación: el empleo de este método garantizó, mediante modelos conceptuales y diagramas de la ingeniería de software, representar elementos importantes relacionados con el objeto de estudio, necesarios para comprender el problema existente y la solución que se propone.

A nivel Empírico:

- Entrevista: la utilización de este método permitió, a partir del conocimiento de especialistas en sistemas SCADA, miembros de la Empresa CEDAI, Villa Clara, Cuba, entender mejor el fenómeno y arribar a conclusiones respecto al camino a seguir y los elementos o funcionalidades que no pueden faltar en una solución de este tipo.
- Experimento: con la realización de pruebas de software se logró comparar las tecnologías Middleware candidatas respecto al tiempo de envío de los datos, siendo esta una de las características de mayor impacto en la decisión final. También se

utilizó este método como parte de la validación de la propuesta, después de integrarla al SCADA y comparar los resultados obtenidos con versiones anteriores del Middleware. Finalmente se usa el método estadístico de la Prueba t para la comparación de medias independientes.

El presente trabajo consta de Introducción, tres capítulos, Conclusiones, Recomendaciones, Referencias Bibliográficas y Anexos. Los capítulos se estructuran de la siguiente manera:

Capítulo 1. Fundamentos teóricos-metodológicos.
Describe los principales conceptos relacionados con sistemas SCADA haciendo énfasis en la forma en que éstos implementan sus mecanismos de comunicación. Se realiza una caracterización de los sistemas Middleware y un análisis y evaluación de las tecnologías y tendencias que permiten desarrollar estas aplicaciones, con un enfoque principalmente dirigido a las que son de código abierto. Concluyendo con la determinación de las tecnologías candidatas.

Capítulo 2. Propuesta de comunicación distribuida entre los componentes de un sistema SCADA.
Expone las funcionalidades del sistema que son objeto de interés de acuerdo a las necesidades. Muestra una vista de alto nivel de la propuesta de solución, sus principios de comunicación y los subsistemas. Se selecciona la tecnología Middleware que servirá como base para desarrollar las interfaces de comunicación distribuida para el SCADA con el apoyo de un matriz de decisión, esta incluye un análisis crítico, tanto cualitativo como cuantitativo de cada una de las alternativas evaluadas.

Capítulo 3. Implementación de interfaces de comunicación distribuida entre los componentes de un sistema SCADA.
Describe las características del ambiente de desarrollo, los modelos de diseño e implementación con la lógica de las diferentes interfaces de comunicación que utilizarán los subsistemas para comunicarse. Se exponen los principales resultados asociados a los procesos de verificación y validación de la propuesta.

Por último se arriba a una serie de conclusiones y recomendaciones válidas para desarrollos futuros.

Capítulo 1. Fundamentos teóricos-metodológicos

En el presente capítulo se realiza una descripción de los principales conceptos y características relacionadas con sistemas SCADA y Middleware, así como las tendencias Middleware en sistemas SCADA y aquellas tecnologías que permiten el desarrollo de aplicaciones de comunicación distribuida.

1.1. Sistemas SCADA como aplicaciones distribuidas

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

"Damos el nombre de SCADA (Supervisory Control and Data Acquisition) a cualquier software que permita el acceso a datos remotos de un proceso y permita, utilizando las herramientas de comunicación necesarias en cada caso, el control del mismo. No se trata de un sistema de control, sino de una utilidad de software de monitorización o supervisión, que realiza la tarea de interface entre los niveles de control y los de gestión a un nivel superior" (Penin, 2012).

A partir del análisis de los diferentes conceptos estudiados, el autor de la presente investigación considera que un SCADA es *"Una solución de software que permite la captura de información de un proceso o planta industrial de forma remota, a veces ejerciendo control sobre ésta y obteniendo información relevante para la toma de decisiones".*

1.1.1 Descripción general de un SCADA

El SCADA es una tecnología que permite obtener y procesar información de procesos industriales dispersos o lugares remotos inaccesibles, transmitiéndola a un lugar para su supervisión, control y procesamiento, normalmente una sala o centro de control.

La incorporación de un SCADA en un proceso permite al usuario conocer el estado de las instalaciones bajo su responsabilidad y coordinar eficazmente las labores de producción y mantenimiento en el campo, supervisando y controlando operaciones críticas y

proporcionando los recursos para recibir la información en forma dinámica y en tiempo real, y proceder a su procesamiento ulterior (Briceño, 2007).

En la actualidad los sistemas SCADA son usados por un sin número de aplicaciones dentro de las que se pueden mencionar: el control de oleoductos, sistemas de transmisión de energía eléctrica, yacimientos de gas y petróleo y control de todos los procesos de esta rama, redes de distribución de gas natural, subterráneos, generación energética**,** sistemas de distribución de agua, entre otros. Estos sistemas han ido evolucionando y ya su explotación no se circunscribe solamente a procesos industriales, sino que se extiende a otras esferas de la tecnología moderna, como es el caso de los edificios inteligentes.

Los sistemas SCADA, como sistemas de control, proporcionan una nueva característica de automatización que realmente pocos sistemas ofrecen, y es lo que los identifica, la **supervisión**. Esta función es llevada a cabo por los supervisores u operadores del sistema. La labor del supervisor representa una tarea delicada y esencial desde el punto de vista normativo y operativo; de esta acción depende en gran medida garantizar la calidad y eficiencia del proceso que se desarrolla. En el supervisor descansa la responsabilidad de orientar o corregir las acciones que se desarrollan (SALAH & LAKHOUA, 2012). La Figura 1 muestra la vista de un SCADA y un operador interactuando con este.

Un SCADA no debe confundirse con un DCS (*Distributed Control System)* aunque los principios y tecnologías que ambos utilizan son similares. La diferencia principal es que los DCS normalmente se utilizan para controlar procesos industriales complejos dentro de un área pequeña, por ejemplo, una planta industrial y las restricciones en tiempo son muy diferentes. En cambio, el SCADA se emplea para el control y supervisión de áreas geográficas muy grandes.

Figura 1. Vistas de un sistema SCADA

1.1.2. Componentes de Hardware

Dentro de los componentes que integran un sistema SCADA, se destacan cuatro fundamentales (Penin, 2012):

- **Instrumentación de campo:** incluye los sensores y actuadores que están directamente conectados a la planta o el equipo que está siendo controlado y supervisado por el sistema SCADA.
- **Dispositivos de control:** son dispositivos basados en microprocesadores que están conectados físicamente al instrumento de campo, permitiendo obtener señales independientes de los procesos, procesarlas y enviar la información a un sitio remoto que generalmente es una sala de control donde se encuentra un sistema central SCADA, el que permite visualizar las variables enviadas. Dentro de los dispositivos más utilizados se encuentran las RTU *(Remote Terminal Units*) y los Controladores Lógicos programables (PLC).
- **Ordenador central (*MTU, Master Terminal Unit*):** es la estación maestra que actúa como el controlador central para el sistema SCADA. La MTU es la responsable de recoger los datos de los dispositivos de campo y procesar la información para generar las medidas de control necesarias, todos estos datos son visualizados a través del HMI *(Human Machine Interface).*
- **Comunicaciones:** es la espina dorsal de todo sistema SCADA. Se utiliza para la transferencia de datos entre el ordenador central y los dispositivos de campo, o con otros equipos que se encuentran dentro de la red corporativa.

1.1.3. Componentes de Software

Los módulos o subsistemas de software que permiten las actividades de adquisición, supervisión y control son los siguientes:

- **Configuración:** permite al usuario definir el entorno de trabajo de su aplicación SCADA.
- **Interfaz hombre máquina:** proporciona al operador la interfaz de usuario para la ejecución de las funciones de control y supervisión de la planta.
- **Módulo de proceso:** ejecuta las acciones de mando pre-programadas a partir de los estados de variables y alarmas existentes.
- **Gestión y archivo de datos:** se encarga del almacenamiento y procesado ordenado de los datos, de forma que otra aplicación o dispositivo pueda tener acceso a éstos para su posterior análisis.

- **Comunicaciones:** garantiza la transferencia de información entre la planta y la arquitectura de hardware que soporta el sistema SCADA, y entre ésta y el resto de elementos informáticos de gestión.

1.1.4. Comunicaciones en los sistemas SCADA

Los sistemas SCADA necesitan comunicarse de varias maneras, con los dispositivos de campo, entre sus propios subsistemas y con aplicaciones externas incluyendo otros SCADA. Un aspecto importante que define la calidad es la capacidad de garantizar la confiabilidad en la transferencia de los datos entre todos sus componentes.

Para realizar el intercambio de información entre los dispositivos de campo y las estaciones del SCADA es necesario un medio de comunicación. Cada fabricante de equipos para sistemas SCADA emplea diferentes protocolos de comunicación y no existe una única estructura de los mensajes. Existen algunos estándares internacionales que regulan el diseño de las interfaces de comunicación entre los equipos del sistema SCADA y equipos de transmisión de datos, dentro de ellos se puede mencionar OPC UA (*OPC Unified Architecture*) (Stopper & Katalinic, 2009). Los sistemas SCADA hacen uso de los protocolos de las redes industriales. Normalmente se usan manejadores (*drivers*) que hablan el idioma de los dispositivos de campo para transmitir la información desde los niveles inferiores hacia los superiores.

La selección de redes de comunicación en sistemas SCADA siempre estará ligada a la topografía del lugar donde se despliega la solución, al presupuesto con que cuenta la empresa para invertir, entre otros aspectos. Más complejo se puede tornar la definición de una tecnología para la gestión de las comunicaciones entre las aplicaciones de un sistema SCADA teniendo en cuenta los niveles de persistencia, seguridad, velocidad y precisión de los datos que necesitan éstos para su operación. Por lo general se utiliza un "*Middleware*" o software de comunicación entre aplicaciones (Dr.S.Bhargavi† & Dr.B.R.Ramamurthy††, 2011; Iacob, Bejan, & Andreescu, 2010; Mohamed Najeh & Mohamed Kamel, 2012).

Los sistemas de automatización pueden ser divididos en distintos niveles, conformando la pirámide de automatización de cinco niveles como describe la Figura 2. La mayoría de los autores hacen referencia a cuatro niveles de automatización y de alguna manera, la propuesta de cinco niveles, no difiere, simplemente se divide el nivel cuatro o de gestión en dos niveles de gestión de alto nivel, los sistemas MES (*Manufacturing Execution Systems*)

y ERP (*Enterprise Resource Planning*) respectivamente. El sistema SCADA se ubica en el nivel tres o de Supervisión, haciendo uso el Middleware para la comunicación entre sus componentes, este último también se enlaza con las aplicaciones de los niveles superiores, de gestión o gerencial.

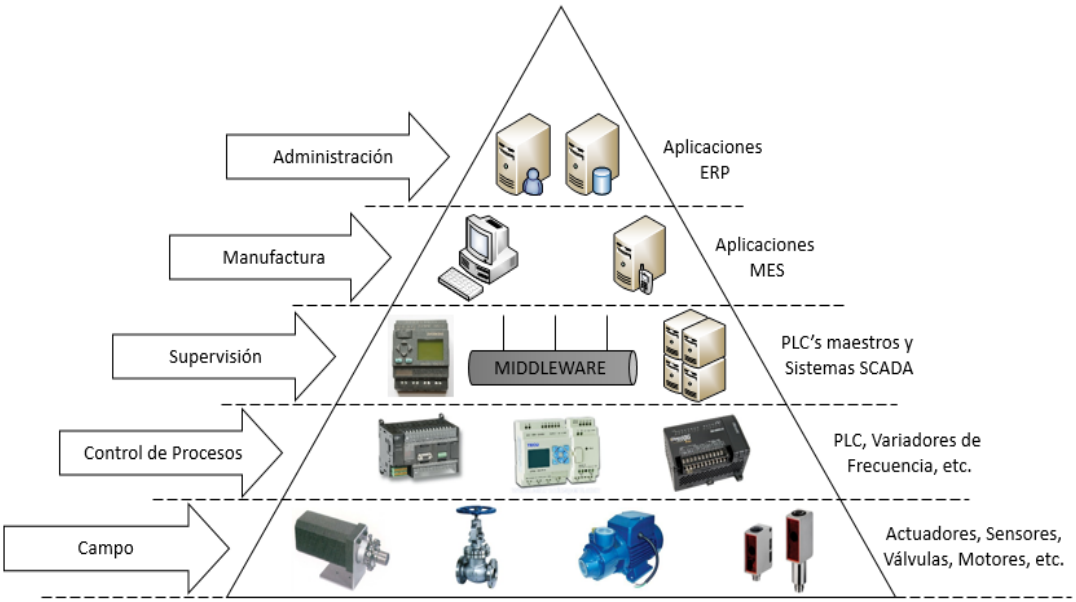

Figura 2. Pirámide de automatización

La evolución de la programación desde el enfoque estructurado a enfoques orientados a objetos, arquitecturas orientadas a servicios, unido a las tecnologías desarrolladas como Microsoft DCOM (*Distributed Component Object Model*), *.NET Remoting*, CORBA, entre otras, han permitido que los sistemas SCADA se expandan más allá de las redes LAN (*Local Area Network*) y lleguen a ofrecer sus servicios por interfaces Web que pueden ser accedidas por los clientes en Internet.

1.2. Sistemas Middleware

Como bien se pudo apreciar en el epígrafe anterior una parte importante de los SCADA es su subsistema de comunicación. Éste es el encargado de gestionar los mensajes y el traslado de información de un componente del sistema a otro.

1.2.1. Definiciones sobre Middleware

"El Middleware es una pieza de software que se encuentra ubicada entre las capas del sistema operativo y la de aplicación. Es, esencialmente, una herramienta que va a hacer la vida más fácil a los desarrolladores de sistemas distribuidos. Logra esto ocultando varias

de las dificultades que involucra el desarrollo de estos sistemas. Desde que los sistemas distribuidos consisten en componentes escritos en lenguajes diferentes y funcionan en sistemas operativos diferentes, es a menudo de ayuda tener un solo desarrollo común y un ambiente de ejecución. Esto es una de las ventajas más significativas de los sistemas Middleware y que facilitará el desarrollo de los sistemas distribuidos" (Nunn, 2002).

"El Middleware es un software que se utiliza para integrar sistemas heterogéneos" (Völter, Kircher, & Zdun, 2005).

"Un Middleware puede ser visto como un conjunto de servicios y funciones reutilizables, expandibles, que son comúnmente utilizadas por muchas aplicaciones para funcionar bien dentro de un ambiente interconectado. Es un software que media entre un programa de aplicación y una red" (Ortega, 2012).

El autor concluye que un Middleware es *"una capa de software que garantiza que aplicaciones diferentes, ejecutándose en diferentes nodos físicos de la red, en diferentes sistemas operativos e implementadas sobre diferentes lenguajes de programación, puedan comunicarse e inter operar entre sí abstrayendo a éstas de las complejidades de la red."*

La Figura 3 tomada en su forma original, describe el funcionamiento de un Middleware.

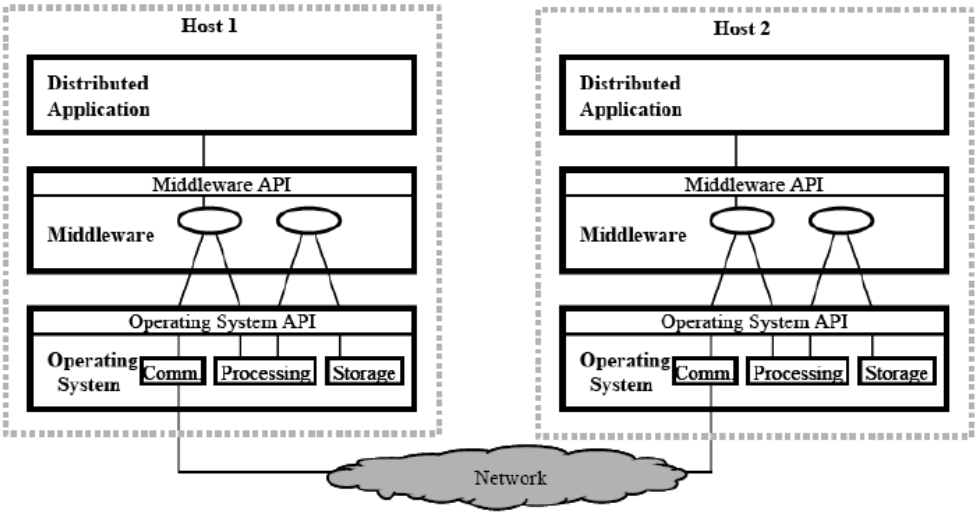

Figura 3. La capa del Middleware ubicada en contexto (Bakken, 2003)

1.2.2. Tipos de Middleware

Es muy común ver que los Middleware se agrupen en diferentes clasificaciones, principalmente en dependencia de para qué tipo de procesos se desarrollen. A continuación

se muestran las clasificaciones que mayor relación tienen con el objeto de la investigación (Bakken, 2003; Pinus, 2004):

1. *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.

2. *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.

3. *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 ORB *(Object Request Broker).*

1.2.3. Funcionamiento tradicional de un Middleware

La Figura 4 describe el proceso de intercambio de información entre aplicaciones implementadas en diferentes lenguajes de programación y ejecutándose sobre diferentes sistemas operativos, a partir de interfaces y servicios que brinda la API (*Application Programming Interface*) del Middleware, se puede garantizar el funcionamiento de estas aplicaciones en plataformas heterogéneas.

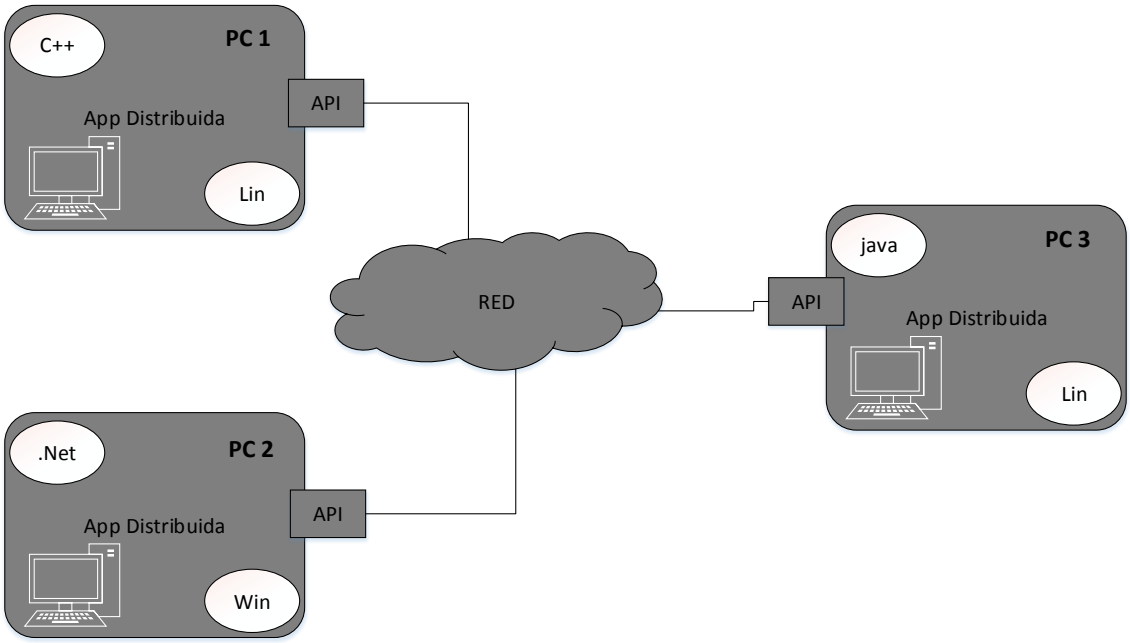

Figura 4. Modelo tradicional de un Middleware

1.2.4. El Middleware en el SCADA desarrollado en el CEDIN

Las soluciones SCADAs desarrolladas en el CEDIN, son sistemas distribuidos donde cada uno de sus componentes se encuentra ejecutándose en diferentes nodos físicos de la red, enlazados mediante un Middleware o aplicación intermediaria (Capa de comunicaciones) como se observa en la Figura 5.

Su principal objetivo es garantizar las comunicaciones en tiempo real y las peticiones a objetos remotos entre los componentes distribuidos del sistema, ocultando los mecanismos de comunicación y las complejidades de los sistemas distribuidos, donde las operaciones remotas suelen parecer hacerse de manera local.

De forma general, el Middleware debe ser uno de los puntos más fiables del sistema, por ello la necesidad de brindar comunicaciones seguras y mecanismos de tolerancia a fallas durante las comunicaciones, mediante la persistencia de las conexiones y/o mecanismos de redundancia.

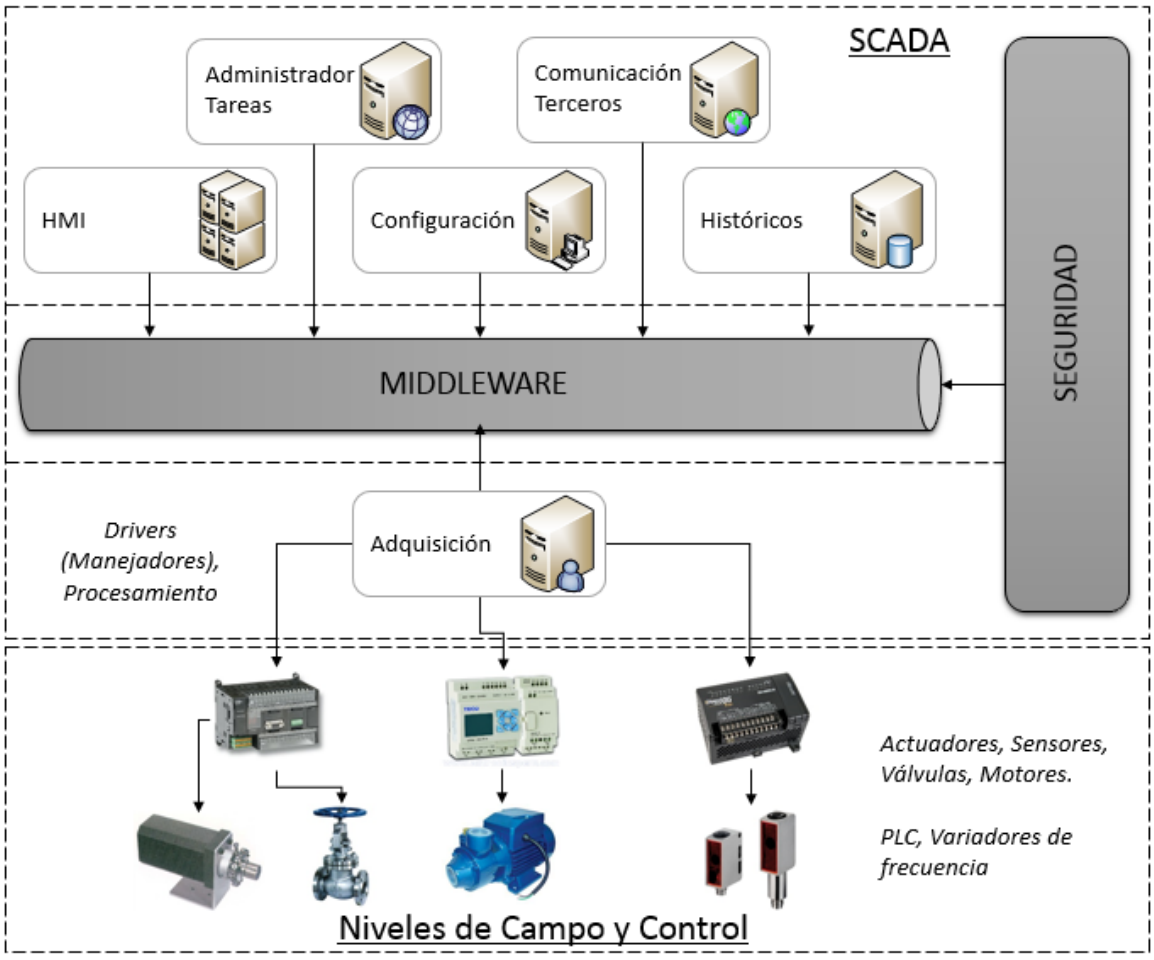

Figura 5. El Middleware en el contexto de la arquitectura distribuida del SCADA

Se involucra con las siguientes funciones y procesos:

- A partir de recolectar los datos del proceso desde los diferentes dispositivos de campo con el uso del módulo de Adquisición y Procesamiento, la información debe enviarse en tiempo real o cumpliendo ciertas restricciones de tiempo, desde éste hacia el más alto nivel en los subsistemas de: Comunicación con Terceros donde los datos se usan para brindar información a aplicaciones externas al SCADA; BD Históricos donde se almacena el historial del proceso y el sistema y que permitirá realizar análisis estadísticos y gráficos de tendencias en tiempo real e históricos; por último en el Visualizador o HMI donde el operador realiza las acciones de supervisión.

- En segundo lugar y a partir de acciones propias del operador y algunas automáticas, enviar hacia el subsistema de Adquisición y Procesamiento, comandos de control con el objetivo de modificar el estado del sistema y/o el proceso.

- Por último, realizar invocaciones a métodos remotos implementados por los servidores de Seguridad y Configuración cuando se requiere la autenticación de un usuario o subsistema, desde la interfaz gráfica del operador y leer o guardar la configuración respectivamente, entre otro grupo de funciones.

1.3. Tendencias y tecnologías actuales

El presente epígrafe detalla las tendencias actuales respecto a las soluciones Middleware adoptadas por los sistemas SCADA de código abierto y propietarios de los principales fabricantes. Se realiza una caracterización de las tecnologías Middleware de propósito general y aquellas bibliotecas que son la base para implementar una solución de comunicación con las condiciones que requiere un sistema de este tipo. Se realiza una breve comparación entre las más usadas en la actualidad y se expone el nivel de adopción de cada una de ellas en diferentes áreas de aplicación y empresas.

1.3.1. Tendencias Middleware en sistemas SCADA

El Middleware es un componente clave para un sistema SCADA, encargado de enlazar a todos los subsistemas distribuidos mediante el paso de mensajes, por lo que un fallo en este punto, por acciones internas o externas, significaría el fallo de todo un sistema casi siempre encargado de las supervisión de procesos críticos, donde minutos de paro, equivalen a detener la producción, grandes pérdidas monetarias y dependiendo de la magnitud del problema, hasta pérdidas de vidas humanas. Esto hace a los grandes desarrolladores de sistemas SCADA implementar sus propias interfaces de comunicación basadas en socket u otras tecnologías y no hacer público los diseños de éstas, evitando accesos no autorizados a su sistema, algo que se ha visto en varias ocasiones en los últimos tiempos, en relación a ataques a los sistemas SCADA mediante el uso de troyanos (Eusgeld, Nan, & Dietz, 2011; Qiang, Barria, & Green, 2011; Terada, Onishi, & Tsuchiya, 2012; Zapata, 2013).

La Tabla 1 ofrece información sobre las comunicaciones en algunas de las principales soluciones de supervisión y control a nivel mundial.

Tabla 1. Tendencias Middleware en sistemas SCADA

Producto	Fabricante	Licencia	Tecnología Middleware
OpenSCADA (Chemnitz, Kruger, Patzlaff, & Tuguldur, 2010; Kumar & Paulus, 2014; "Open SCADA Home project," 2015)	IBH Systems GmbH	GNU/GPL	API basada en socket, protocolo NGP (*Next Generation Protocol*) con conexión vía TCP (*Transmission Control Protocol*) y OPC.
oSCADA (Savochenko, 2015)	Roman Savochenko	GNU/GPL v.2	API basada en socket y OPC UA.
Proview ("Proview Home project," 2014)	Proview team	GNU/GPL	TCP y UDP (*User Datagram Protocol*) socket y Siemens 3964R.
WinCC	Siemens AG	Comercial	Todas las potencialidades de la Plataforma .Net y OPC UA.
InTouch	Wonderware	Comercial	Todas las potencialidades de la Plataforma .Net
FAST/TOOLS (Yokogawa, 2012)	Yokogawa	Comercial	Todas las potencialidades de la Plataforma .Net, OPC UA.

Existen muchos desarrolladores de SCADA que basan sus soluciones de comunicación en tecnologías Middleware de propósito general, no siendo el caso de las aplicaciones abordadas en la Tabla 1. También se estila el uso de un Middleware para garantizar la interoperabilidad entre los propios sistemas SCADA, utilizando para ello la combinación de varias de estas tecnologías (Dayal, Tbaileh, Yi, & Shukla, 2015; Gaitan, 2014).

1.3.2. Estándares, tecnologías y bibliotecas para comunicación distribuida

Antes de volcar todo el esfuerzo a pensar en una solución de comunicación desde cero, es preciso hacer primero un estudio del estado del arte de las tecnologías que hoy permiten desarrollar aplicaciones tipo Middleware o simplemente ya representan Middleware de propósito general que pueden adecuarse a variados entornos distribuidos y dominios de aplicación. De igual forma aquellos sistemas de serialización de datos que unidos a estas tecnologías, pueden mejorar sus prestaciones.

A continuación se muestra una descripción de las principales tecnologías y sus características más relevantes. En todos los casos se hace en base a estándares y herramientas de código abierto.

Tabla 2. Breve caracterización de las tecnologías y bibliotecas Middleware

Alternativas de comunicación interprocesos
CORBA
• OMG ha definido la arquitectura OMA *(Object Management Architecture*) como su estándar más importante, que es CORBA basado en el mecanismo de intercambio de objetos definido en el software como ORB. • Define un estándar de comunicación entre diferentes ORB-s que es el protocolo IIOP (*ORB Protocol Inter*), basado en TCP / IP y aplicado en el estándar CORBA desde la versión 2.0 (Onderka, 2011). • Se ha introducido el lenguaje IDL (*Interface Description Language*) que se utiliza para la especificación de la interfaz de forma independiente del lenguaje. • A través del IDL actualmente existe *mapping* para C, C ++, Smalltalk, Java, Ada y COBOL que ya están estandarizados (Juric, Rozman, & Hericko, 2000). • La especificación del estándar es libre de descarga.
TAO (Calkins, 2009; Schmidt, Gokhale, Harrison, Levine, & Cleeland, 2011)
• TAO es una implementación avanzada de CORBA en C++, en tiempo real y de código abierto. • Soporta IIOP 1.2 que permite un alto grado de interoperabilidad con otros ORB. • Es un ORB de segunda generación desarrollado con una arquitectura altamente extensible, como resultado del uso del *framework* ACE. • ACE es un acrónimo de *ADAPTIVE Communication Environment*, es un *framework* orientado a objetos que implementa varios patrones básicos para software de comunicación concurrente. • TAO implementa muchos de los servicios definidos por CORBA, entre los más relevantes se encuentran: *Notification Service, Event Service, Naming Service, Security Service y Load Balancing Service.* • Se encuentra disponible bajo licencia GPL (*GNU Public Licence*).
Data Distribution Service **(DDS)**(OMG, 2007, 2011)
• DDS es un estándar definido por la OMG (*Object Manager Group*), es un middleware orientado a datos que implementa las comunicaciones bajo el paradigma publicación/suscripción con capacidad para tiempo real. • DDS está disponible con API para C, C++ y Java, y en multitud de plataformas (Linux, Solaris, Windows, Integrity, LynxOs, VxWorks…). • Independencia de la plataforma por el uso de estándares como el IDL. • Descarga libre.
Open Data Distribution Service **(OpenDDS)** (Busch, 2010, 2013; Martinez, 2010)
• OpenDDS es una implementación en C++ y de código abierto del estándar DDS de la OMG. • Soporta actualmente transporte TCP y UDP punto a punto. • Además implementa el paradigma publicación/suscripción como principal mecanismo de transporte. • Tiene un solo proceso por separado, el *DCPSInfoRepo* actúa como un distribuidor de datos central, asociando publicadores con suscriptores. • Exhibe una arquitectura multihilos escalable. • Licencia GPL. • Soporte comercial a través de OCI (*Object Computing Inc.*).

Alternativas de comunicación interprocesos
***Internet Communication Engine* (ICE**) (Henning & Spruiell, 2013; Krizsán, 2010)
• ICE es una plataforma ligera desarrollada por la empresa ZeroC, algunos de sus miembros son los creadores de CORBA e intentan resolver varios de los problemas de esta última. • Es un Middleware orientado a objetos, proporciona herramientas, API, y soporte de bibliotecas para construir aplicaciones cliente/servidor y publicación/suscripción orientadas a objetos. • Tiene un grupo de características que garantizan la calidad de la comunicaciones: proporciona un manejo síncrono y asíncrono de mensajes (servicio *IceStorm*); es independiente de la máquina, lenguaje de programación con el servicio Slice (*Specification Language for ICE*), implementación y del sistema operativo; soporta el manejo de hilos; es independiente del protocolo de transporte empleado; mantiene transparencia en cuanto a localización de servidores y servicios (*IceGrid*); es seguro (SSL y *Glacier2*); y permite hacer persistentes las aplicaciones (*Freeze*). • Licencia GPL y comercial por ZeroC.
ZeroMQ (Akgul, 2013; Y. Le Goc* et al., 2013)
• Es una biblioteca Middleware orientada a mensajes. Provee cuatro tipos de transporte: en-proceso (*INPROC*), inter-procesos (*IPC*), multicast vía PGM y UDP y por último TCP. • Brinda varios paradigmas o patrones de comunicación: encuesta/respuesta, publicación/suscripción y *pipeline*. • Escrito en C y con enlaces a más de 30 lenguajes entre los que se encuentran: Ada, C, C + +, Common Lisp, Erlang, Go, Haskell, Java, Lua, NET, OOC, Perl, PHP, Python y Ruby, etc. • Licencia GPL v3 con soporte comercial por iMatix.
OpenAMQ (iMatix, 2009)
• OpenAMQ es una implementación del estándar AMQP, un producto de mensajería distribuida y de código abierto. • Está diseñado para ser extremadamente rápido y robusto. Mensajería asíncrona a través del modelo publicación/ suscripción. • Soporta alta disponibilidad y federación de servidores. Presenta un mecanismo de seguridad, extensible y configurable, ya sea por definición del usuario o autenticación plana SASL (*Simple Authentication and Security Layer*). • Construido sobre un *framework* de alto rendimiento, portable para Linux, Windows, Solaris y otros sistemas Unix. • Licencia GPL y soporte comercial para negocios por iMatix.

1.3.3. Tendencias Middleware por áreas de aplicación

En la actualidad existe poca información sobre las tendencias Middleware en sistemas SCADA como fue abordado en el epígrafe 1.3.1, sin embargo, diferentes artículos explican el uso que tienen hoy las diferentes tecnologías en importantes áreas de aplicación y de la industria. A continuación los proyectos y sectores que adoptan estas herramientas con mayor fuerza:

- Implementaciones comerciales de CORBA tienen su mayor aplicación en soluciones embebidas con un alto nivel de adopción en el área Militar/Aeroespacial de un 44 %, muy superior a otras áreas como (Czerny, 2000):

    - Instrumentación Científica, 6%
    - Consumo, 12%
    - Medicina, 12%
    - Telecomunicaciones, 26%
- Implementaciones comerciales de DDS alcanzan su mayor nivel de aplicación en soluciones militares de EE.UU y todo el mundo, principalmente en sistemas de combate naval (OMG, 2011).
    - También es utilizado comercialmente en una serie de industrias, destacándose los sistemas SCADA y automatización industrial en general.
    - Además hay varias universidades importantes que usan DDS en proyectos de investigación activos, entre ellas: MIT, Universidad Carnegie Mellon y Vanderbilt en Estado Unidos y ETH Zurich, Universidad Técnica de Munich, e Instituto Avanzado de Ciencia y Tecnología de Corea.
- A pesar de lo novedoso ya sobrepasan las 52 compañías y organizaciones que usan ICE como la plataforma de comunicación en sus aplicaciones, destacándose el área de supervisión y control de procesos (ZeroC, 2013).
    - La aplicación de ICE se sale además del espacio del software para aplicaciones tipo escritorio, tiene variantes para desarrollar aplicaciones embebidas.
- Tecnologías como ZeroMQ tienen un nivel de aplicación más general en diferentes áreas, destacándose su uso en Plataformas de negociación, Multimedias, Computación Grid, entre otras (iMatix, 2013).

La decisión sobre qué tecnología utilizar siempre va a depender del entorno y las características que requiera la aplicación final. Todas pueden utilizarse para el desarrollo de soluciones en tiempo real o con limitantes en el tiempo de transferencia de los datos.

1.3.4. Evaluación cualitativa de las tecnologías Middleware

La Tabla 3 expone en forma sintetizada la evaluación de las características que en primera instancia hay que analizar cuando se está seleccionando una tecnología para desarrollar un Middleware para un sistema crítico como es caso del SCADA. Antes se describen los principales atributos de calidad a tener en cuenta en la identificación de las tecnologías candidatas:

1. **Sistema de mensajes (síncronos y asíncronos):** esta propiedad va a determinar el tipo de comunicación que provee la tecnología, siendo necesario contar con otros mecanismos de comunicación además de RPC, por ejemplo el de publicación/suscripción.
2. **Rendimiento:** en este caso no se refiere al tiempo real duro, sino a la posibilidad de transmitir grandes cantidades de datos (por ej.: 50 mil) por segundo, o lo que es lo mismo, baja latencia en la transmisión de los datos.
3. **Fiabilidad:** la tecnología de comunicación debe brindar políticas de QoS como la redundancia y persistencia de las conexiones (tolerancia a fallas) que garanticen la disponibilidad y fiabilidad del servicio.
4. **Seguridad:** con soporte de mecanismos de seguridad durante las comunicaciones. Entre más seguro sea el sistema, más fiable será.
5. **Soporte:** es necesario contar con bibliotecas soportadas por comunidades activas, con proveedores desarrollando nuevas versiones y/o soportando las anteriores de manera gratis, con abundante documentación.

Tabla 3. Comparación cualitativa entre tecnologías y bibliotecas Middleware

Tecnología o biblioteca	Mensajes síncronos y asíncronos	Rendimiento (Tiempo real)	Fiabilidad	Seguridad	Soporte	Eval.
ACE+TAO	x	x	x	x		4
OpenDDS		x	x			2
ZeroC ICE	x	x	x	x	x	5
ZeroMQ	x	x	x		x	4
OpenAMQ	x	x	x			3

Después de realizar un análisis crítico del estado del arte y las tendencias y tecnologías actuales se puede determinar cuáles serían las tecnologías candidatas:

1. Todas, a excepción de OpenDDS brindan mecanismos de comunicación cliente/servidor y publicación/suscripción.
2. En todos los casos, según sus propios desarrolladores y el resto de la bibliografía, ofrecen baja latencia durante las comunicaciones asíncronas.
3. Todas ofrecen mecanismos de tolerancia a fallas con *QoS* para garantizar soluciones fiables, lo que no significa que sean robustos en todas las alternativas.

4. Son ICE y ACE+TAO las únicas tecnologías que brindan mecanismos de seguridad durante las comunicaciones.
5. A diferencia del resto, existe una comunidad muy activa y que brinda abundante documentación actualizada para ICE y ZeroMQ.

Como parte del proyecto Sistema de Control y Aceleración (CERN) se realizó un estudio de las principales tendencias y marcas de Middleware para utilizarlo en sustitución del Middleware que tenía implementado dicho proyecto, donde también ICE y ZeroMQ resultaron calificar como las alternativas más adecuadas (Dworak, Charrue, Ehm, Sliwinski, & Sobczak, 2011).

1.3.5. Evaluación cualitativa de los sistemas de serialización

En muchos casos las tecnologías analizadas en el epígrafe anterior ven afectado su rendimiento producto al tiempo que tardan en llevar los datos al lenguaje o estado que conocen las aplicaciones a cada lado del Middleware, por eso no es para nada descabellado pensar en una solución híbrida entre las tecnologías de comunicación distribuidas y las bibliotecas de serialización como comúnmente se les conoce.

La serialización es el proceso de traducción de las estructuras de datos o el estado del objeto en un formato que se puede almacenar y llevar al estado inicial más tarde en el mismo o en otro entorno informático. En la mayoría de los casos los formatos de serialización binaria no son legibles, pero se pueden comprimir con eficacia los datos, lo cual es muy útil para cachés, comunicación entre procesos, los agentes de mensajes, etc. En muchos casos de desarrollo es importante seleccionar un buen formato de serialización binaria, más si serán utilizados en sistemas distribuidos para la inter comunicación y el almacenamiento. Echemos un vistazo a lo más interesante de estos formatos.

Tabla 4. Evaluación cualitativa de los sistemas de serialización

Sistemas de serialización de datos
Protocol Buffer **(Protobuf)** (Buffers, 2013; Maeda, 2012)
• Es una forma de codificar datos estructurados de manera eficiente en formato aún extensible. Google usa *Protocol Buffers* para casi todos sus protocolos RPC internos y formatos de archivo. • Por defecto, Protobuf solo soporta C++, Java y Python como lenguajes de destino. Hay generadores de código para otros lenguajes, pero no son oficiales.

Sistemas de serialización de datos
• En tiempo de ejecución sólo ofrece la serialización y deserialización. Puede generar algunos *stub* RPC genéricos pero el tiempo de ejecución no termina en una implementación RPC. • Licenciado bajo BSD.
Apache Thrift ("Apache Thrift Wiki," ; Felden, 2014; Grigorik, 2011; Kleppmann's, 2013; Thusoo et al., 2009)
• Thrift es un lenguaje de definición de interfaz que se utiliza para definir y crear servicios para numerosos lenguajes. • Se utiliza como un *framework* RPC y se desarrolló por Facebook para el desarrollo de servicios escalables. • Combina una pila de software con un motor de generación de código para construir servicios que funcionan de manera eficiente y sin problemas entre C#, C++, Cappuccino, Cocoa, Erlang, Go, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, Node.js y Smalltalk.
MessagePack (Grigorik, 2011; Kago & Yamashita, 2013; "The MessagePack Project," ; Ohta, 2011)
• MessagePack es un nuevo protocolo de serialización de objetos binario y una biblioteca construida pensando en la eficiencia y velocidad. • Su desarrollador, Sadayuki Furuhashi lo presenta como una alternativa más rápida que JSON, que tiene igualmente amplio soporte a través de varios lenguajes. • Se vincula con diferentes lenguajes como: C, C++, Ruby, Python, Perl, PHP, Java, Haskell, D, Erlang, Lua. • Está licenciado bajo Licencia Apache en su Versión 2.0.

1.4. Conclusiones del capítulo

Los grandes desarrolladores de sistemas SCADA implementan sus propios mecanismos de comunicación distribuida, y no hacen público el diseño de los mismos.

Las tecnologías de comunicación interprocesos, ICE, ZeroMQ y TAO son las que más se adecuan a las necesidades, teniendo en cuenta las características deseables para la comunicación en un sistema SCADA.

El desarrollo de una solución híbrida entre una tecnología como ICE, TAO o ZeroMQ y un sistema de serialización de datos como ProtoBuf, Apache Thrift o MessagePack podría mejorar el rendimiento de las tecnologías en sí mismas.

Capítulo 2. Propuesta de comunicación distribuida entre los componentes de un sistema SCADA

El presente capítulo viene a exponer una vista de alto nivel de la propuesta de solución, donde se modelan las interfaces de comunicación distribuida para el SCADA sobre la base de la tecnología que también se seleccionará acá. Tiene como fin brindar una alternativa de comunicación más sencilla y eficiente para mejorar los tiempos de transferencia de los datos entre los componentes del sistema.

2.1. Principios de la solución de comunicación distribuida

Tomando como base el modelo tradicional de desarrollo de aplicaciones Middleware expuesto en el epígrafe 1.2.3 así como los estándares y tecnologías de comunicación distribuida, y las limitantes expresadas en la introducción de la presente investigación, se tienen en cuenta los siguientes principios que van a contribuir a la mejora de los rendimientos durante el intercambio de información entre los componentes distribuidos del SCADA.

La propuesta debe centrarse en dos aspectos fundamentales:

1. Arquitectura flexible y extensible que garantice la abstracción de los mecanismos de comunicación y la tecnología, y a su vez no afecte el rendimiento del sistema.
2. Tecnología base de comunicación que como principal característica, garantice la distribución de los datos con muy baja latencia y de esta forma permita disminuir los tiempos de transferencia de los mismos.

2.1.1. Vista general de la propuesta de solución

La propuesta de solución (véase Figura 6) está enfocada en brindar un grupo de interfaces de comunicación distribuida que van a permitir alcanzar el objetivo propuesto. Se caracteriza por otorgar a la solución dos características muy específicas (principios) y otras más generales que también van a impactar en la calidad de la propuesta al momento de su completa implementación y puesta a punto.

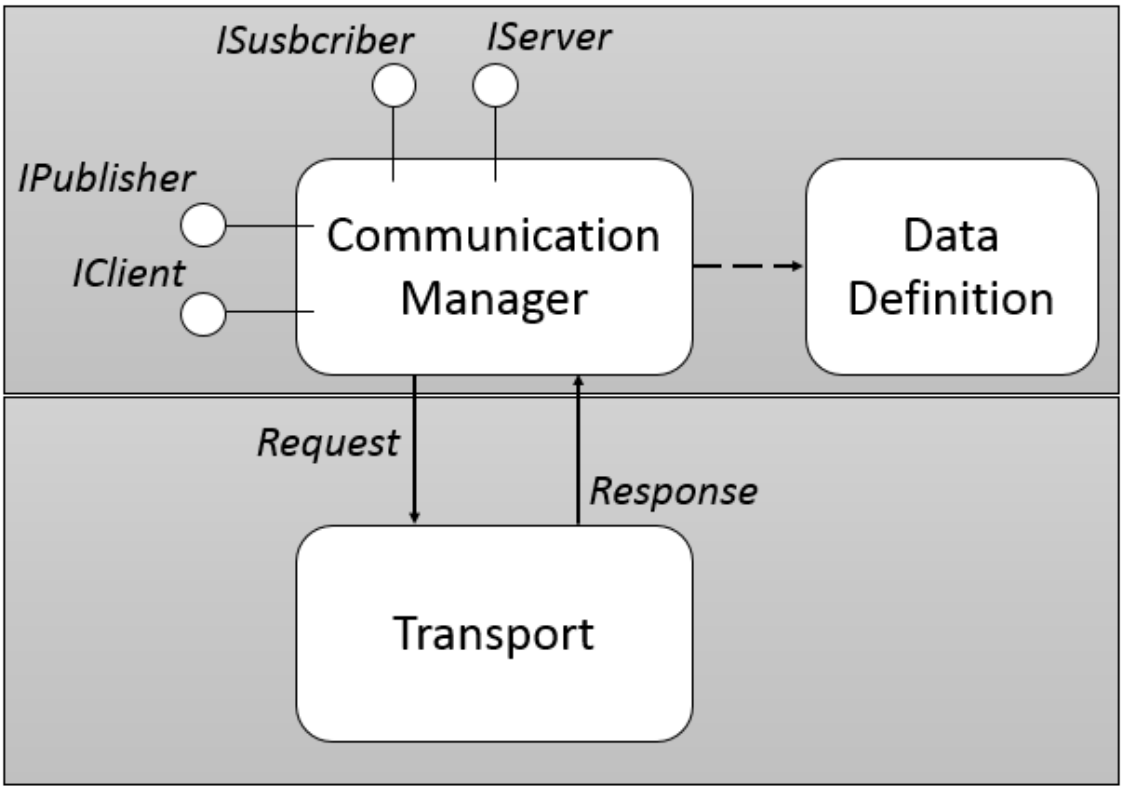

Figura 6. Esquema de alto nivel de la propuesta

Los elementos fundamentales que se tuvieron en cuenta en la propuesta, fueron los siguientes:

1. ***Communication Manager*** (Gestor de la Comunicación): subsistema independiente de la tecnología Middleware de propósito general que se seleccione, garantiza la abstracción de ésta a los componentes del SCADA y la posibilidad de cambiar de tecnología sin incurrir en grandes modificaciones.
    - Al mismo tiempo esta característica no debe afectar el rendimiento del sistema durante las comunicaciones, un elemento de vital importancia en la supervisión y control de procesos industriales.
    - Este subsistema va a exponer a los componentes del SCADA un conjunto de interfaces de comunicación bien definidas *(IPublisher, ISubscriber, IClient y IServer)*, las cuales encapsulan el conjunto de funcionalidades para el envío y recepción de datos de tipo complejo y la invocación de métodos remotos que van a formar parte de otro grupo de subsistemas internos más específicos.

2. ***Data Definition*** (Definición de Datos): va agrupar en su ámbito el conjunto de las definiciones de los datos del SCADA y las relaciones entre los mismos, permitiendo de esta manera eliminar el proceso de conversión de datos a cada lado de las comunicaciones, unas de las debilidades que presenta la actual solución de comunicación. Sin duda esta característica será un aporte al rendimiento del sistema.
3. ***Transport*** (Transporte): en este caso hace referencia a la tecnología Middleware de propósito general que provea un *Broker* (enrutador o intermediario), capaz de establecer un contrato de comunicación entre las partes mediante el enrutamiento de mensajes de manera fiable.
    - Esta debe garantizar un buen rendimiento durante las comunicaciones que requieren tiempo real o con restricciones de tiempo, además de aportar políticas de *QoS.*

Con estos principios se pretende garantizar una solución que cumpla con los principales requisitos de comunicación que exigen los sistemas SCADA, con interfaces de comunicación que provean simplicidad y fiabilidad de cara a los desarrolladores de aplicaciones distribuidas para sistemas de supervisión y control en general. Teniendo en cuenta los principios definidos y la propuesta general de solución, se presentan más adelante las interfaces y subsistemas específicos que se proponen como parte de dos mecanismos o paradigmas de comunicación: el paradigma cliente/servidor (sincrónico) y el paradigma publicación/suscripción (asincrónico), este último como el de mayor relevancia por ser el que interviene en la distribución de datos en tiempo real.

2.1.2. Selección de la tecnología Middleware (Transporte)

En el capítulo 1 se hace una caracterización de las tecnologías Middleware y de los sistemas de serialización de datos con mejores prestaciones en la actualidad, de igual forma se realiza una comparación cualitativa a partir de aspectos extraídos de los diferentes artículos estudiados y los criterios de sus propios desarrolladores o proveedores. En este epígrafe se pretende, a partir de una matriz de decisión, realizar una comparación cuantitativa con pruebas de concepto y rendimiento incluidas, entre las tecnologías candidatas que se dieron a conocer en las conclusiones del capítulo anterior.

Cuando se trata de la evaluación de tecnologías Middleware, la primera reacción a menudo es idear una serie de pruebas de rendimiento de las diferentes implementaciones. La primera idea es decir, el Middleware “que va más rápido" se considera que es el "mejor".

Desafortunadamente, este proceso puede así defraudarnos. Irónicamente, una de las razones es que el rendimiento podría ser irrelevante para la aplicación. Mientras que el rendimiento es realmente importante y puede tener un gran impacto en el costo de operación, utilizarlo como principal criterio de evaluación sólo tiene sentido si la aplicación de hecho requiere un alto rendimiento. Sin embargo, para muchas aplicaciones, el rendimiento del Middleware importa poco, ya sea porque la carga general nunca es lo suficientemente alta como para que el rendimiento sea un problema, o porque los cuellos de botella son inherentes al trabajo que la aplicación tiene que hacer, y ninguna mejora del rendimiento del Middleware va a mejorar significativamente el sistema en general.

Son los criterios de evaluación por mucho lo más importante. A continuación se realiza una pequeña selección (Henning & Spruiell, 2011):

- ¿Cómo es la curva de aprendizaje del Middleware? ¿Es posible que los desarrolladores puedan empezar a escribir código de calidad sin tener que pasar por un largo proceso de aprendizaje?
- ¿Cuál es la calidad de las API? ¿Son las API consistentes y fáciles de aprender? ¿Tiene la API un manejo seguro de hilos y excepciones? ¿Es la gestión de memoria automática o debe ser atendido por el programador?
- ¿Cuál es la calidad de la documentación? ¿Es fácil encontrar material de referencia que es claro y comprensivo? ¿La documentación proporciona suficientes materiales estilo tutorial y ejemplos de código fuente para que los programadores sean productivos rápidamente?
- ¿Tiene el Middleware una vibrante comunidad de desarrolladores que intercambia activamente experiencias y proporciona soporte?
- ¿Es de alta calidad el entrenamiento profesional disponible?
- ¿Está el código fuente disponible?
- ¿Cuántos sistemas operativos y lenguajes de programación son soportados por el Middleware? ¿Son estos relevantes para la aplicación? ¿Es fácil portar el Middleware a una nueva plataforma?

- ¿Qué nivel de soporte es proporcionado por el proveedor? ¿Es posible obtener un soporte prioritario y arreglos en caliente si hay un problema?
- ¿Qué tan confiable es el Middleware? ¿Son los defectos que se encuentran en el Middleware rectificados con prontitud?
- ¿Cuán completo es el conjunto de características del Middleware? ¿Tiene este las características que son irrelevantes para la aplicación, pero llevan una penalización en términos de rendimiento o complejidad de las API?
- ¿El Middleware proporciona sólo un mecanismo básico de RPC, o incluye los servicios esenciales, tales como un servicio de eventos, herramientas administrativas, un servicio de persistencia, seguridad, etc.? ¿Cuánto esfuerzo se requiere para construir los servicios que faltan, pero que necesita la aplicación?
- ¿Tiene el Middleware un uso eficiente de los recursos como la memoria, descriptores de fichero, ciclos de CPU (*Central Processing Unit*) y ancho de banda de la red?
- ¿En qué medida el Middleware es escalable para aplicaciones de gran tamaño? ¿Es posible escalar una aplicación mediante la adición de más servidores sin influir negativamente en un sistema ya implementado?

Estos son sólo algunos de los aspectos que se pueden tener en cuenta cuando se quiere comparar varias tecnologías de comunicación inter procesos, sin embargo, todo va a estar determinado por las características y alcance de la solución.

2.1.2.1. Ambiente físico del escenario de pruebas

Se usaron tres computadores con las mismas características, sin embargo, el ambiente utilizado a penas se acerca al ambiente ideal donde realmente corren los sistemas SCADA, ya sea por procesamiento como por la velocidad de la red.

Ambiente de pruebas

- Procesador Intel(R) Core (TM) i3-2120 CPU @ 3.30GHz
- Memoria 8GB de RAM
- Tarjeta de red a 100Mbit/s

La red utilizada para nada es una red dedicada, las pruebas se desarrollaron en el espacio de un laboratorio de desarrollo en una red de datos con el ruido natural que imprimen las operaciones que realiza cada usuario, desde el envío de mensajería instantánea, el uso del

correo electrónico como de internet. Adicionar que hasta cierto punto el tipo de pruebas realizada permitió comprobar el rendimiento, pero por la forma en que se aplicaron, también se pueden asumir como pruebas de estrés.

Normalmente se publica cierta cantidad de datos, evento a evento y se calcula el tiempo al momento de haber recibido el total de la información. En la presente investigación se realizaron 23 corridas para cada valor de muestra de datos, donde para cada caso se envió un lote (*batch*) de alarmas, se esperó un segundo y se procedió al próximo envío y así sucesivamente hasta llegar a 23 iteraciones. De esta forma, se puede ver afectada la memoria, donde un publicador estará enviando datos sin importar si ya los suscriptores recibieron los publicados con anterioridad, aun así el sistema no se vio afectado y tampoco se evidenció perdida de información.

2.1.2.2. Evaluación cuantitativa de las tecnologías ICE, TAO y ZeroMQ

En el caso de las pruebas de rendimiento para TAO y ZeroMQ se utilizó la combinación de éstas con el sistema de serialización MessagePack, obteniendo variantes híbridas de comunicación y donde este último se presentó más eficiente frente a las variantes de Google y Facebook respectivamente (Bunch et al., 2010; Yamashita & Kago, 2013).

En la ejecución de estas pruebas (ver Anexo 2) se pudo observar que ZeroMQ es superior a TAO en cuanto a los tiempos de respuesta en el paradigma cliente/servidor así como en manejo de eventos/segundos en el paradigma publicación/suscripción. Sin embargo, la diferencia entre una y otra no es significativa si se tiene en cuenta que TAO es un *framework* que implementa muchos servicios de objetos y tiene un manejo de la concurrencia bien optimizado y ZeroMQ se limita a una biblioteca de mensajes (Viltre, Lalcebo, González, & Cáceres, 2014).

Para el caso de ICE se realizaron pruebas de rendimiento similares, consistieron en medir el tiempo de transferencia de los datos, para valores potencia de 10 como: 1, 10, 100, 1000, 10000 y 100000 estructuras de datos del tipo alarma. Además se evaluó el rendimiento de ICE a partir de los siguientes aspectos:

- Cantidad de eventos por segundos.
- Latencia en la transferencia de los datos.
- Tasa de transferencia (MB/segundos).

- Consumo de recursos durante la ejecución de las pruebas.

Las Figuras de la 7 a la 10 muestran el rendimiento obtenido con ICE y el consumo de recursos en función del uso de la memoria RAM. Las gráficas asociadas con el tiempo presentan los valores de media obtenidos para cada una de las muestras, pudiendo detallar el resto de los resultados en el Anexo 3.

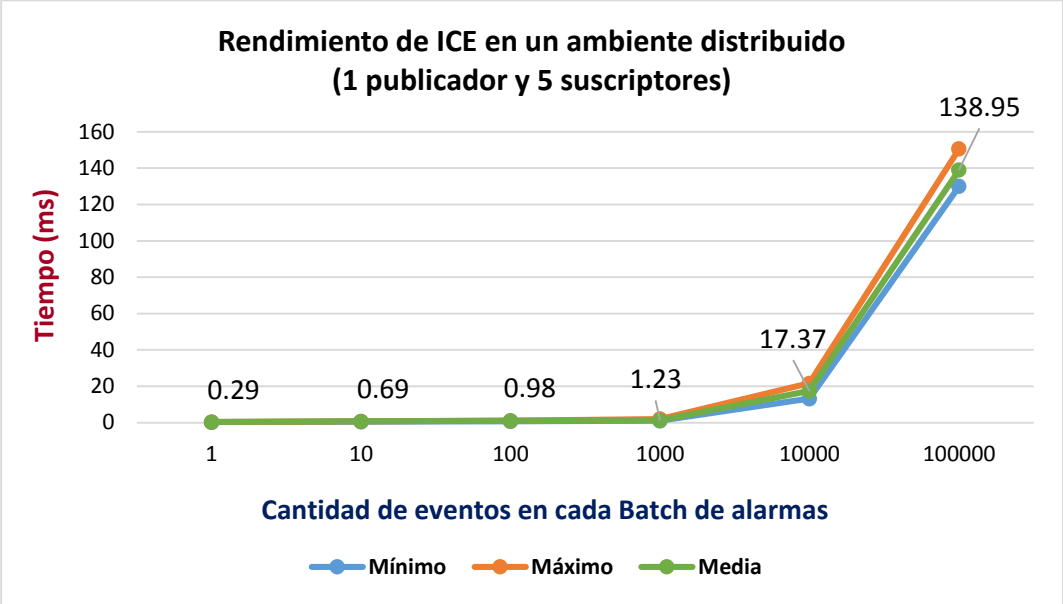

Figura 7. Rendimiento de ICE en el escenario de un publicador y cinco suscriptores

Esta prueba consistió en ejecutar el servicio *IceBox* (*Broker* de ICE) en una computadora, en esta misma un publicador que envía hasta 23 iteraciones de un lote de una hasta cien mil alarmas hacia varios suscriptores que se encuentran ubicados en una segunda computadora, lo que permitió obtener los valores de mínimo, máximo y media asociados a los tiempos de transferencia como se muestra en la Figura 7, siempre en milisegundos (ms). Se pudo apreciar que la tecnología necesita menos de un ms para transferir una, diez y hasta 100 alarmas y para las cantidades más representativas como diez mil y 100 mil, se comprobó que esta variante de Middleware es siete veces más rápida que ZeroMQ y TAO en pruebas similares.

Tomando como referencia el escenario de 100 mil alarmas, se alcanzó el mejor tiempo de 130 ms con (véase Anexo 3):

- Una tasa de transferencia de 74.07 MB/segundos,
- Un rendimiento 720823.4814 Eventos/segundos y

- Una latencia promedio de 1.38957E-06 segundos.

La Figura 8 da a conocer el comportamiento de la memoria, señalando principalmente el uso de la misma por parte del servicio *IceBox*, donde alcanza el mayor consumo para la prueba de 100 mil alarmas, que es muy bueno si se tienen en cuenta las características de la prueba realizada. Aún con este consumo el funcionamiento de la computadora fue estable y el uso del CPU no sobrepasó el 1 %.

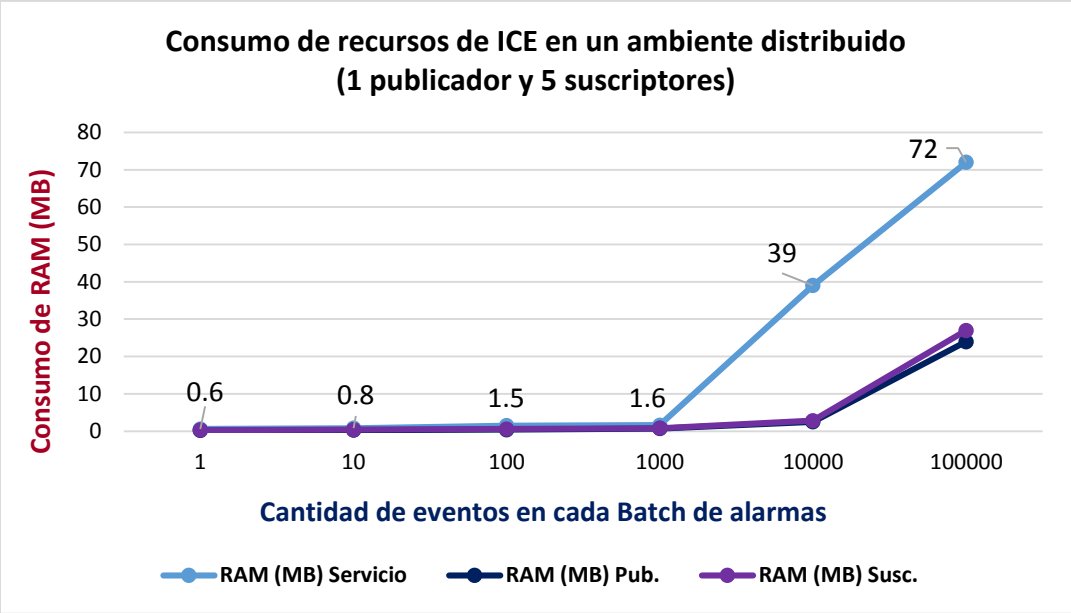

Figura 8. Consumo de recursos de ICE en el escenario de un publicador y cinco suscriptores

Muy similar a la prueba anterior se realizó otra prueba pero en este caso se utilizaron diez suscriptores para recibir datos, con la implicación de tres computadoras, una con el servicio *IceBox* y el publicador y otras dos con cinco receptores cada una.

También aquí se obtuvo como resultado que la tecnología necesita menos de un ms para transferir una, diez y hasta 100 alarmas y para las cantidades más representativas como diez mil y 100 mil, se comprobó que esta variante de Middleware sigue siendo siete veces más rápida que ZeroMQ y TAO en pruebas similares como se aprecia en la siguiente gráfica.

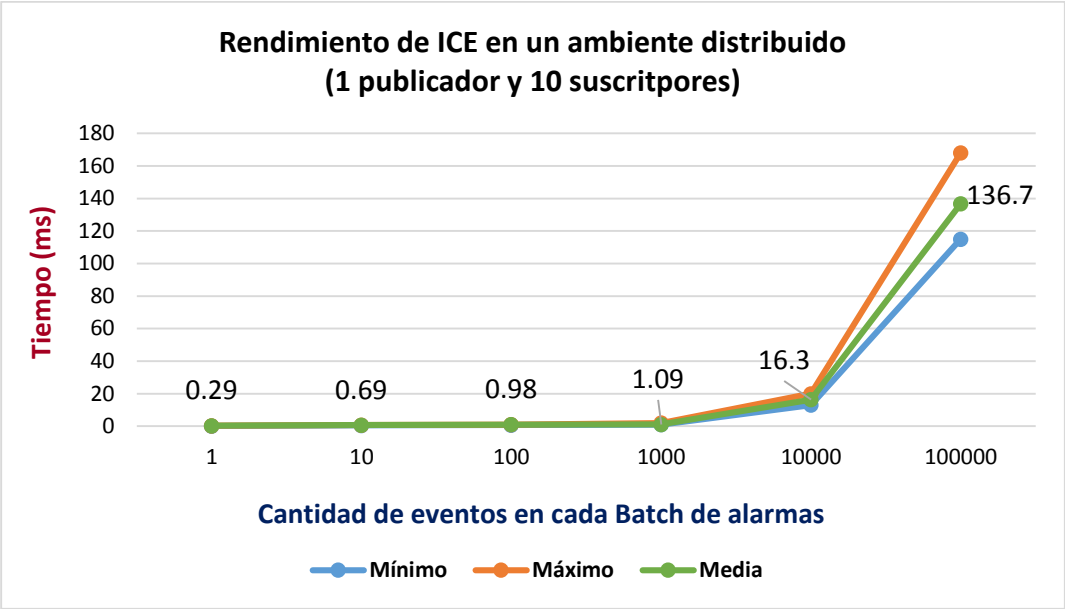

Figura 9. Rendimiento de ICE en el escenario de un publicador y diez suscriptores

La Figura 10 describe el comportamiento de la memoria para este escenario, siguiendo un patrón similar a la prueba anterior pero con un consumo diferente como resultado de haber duplicado el número de clientes.

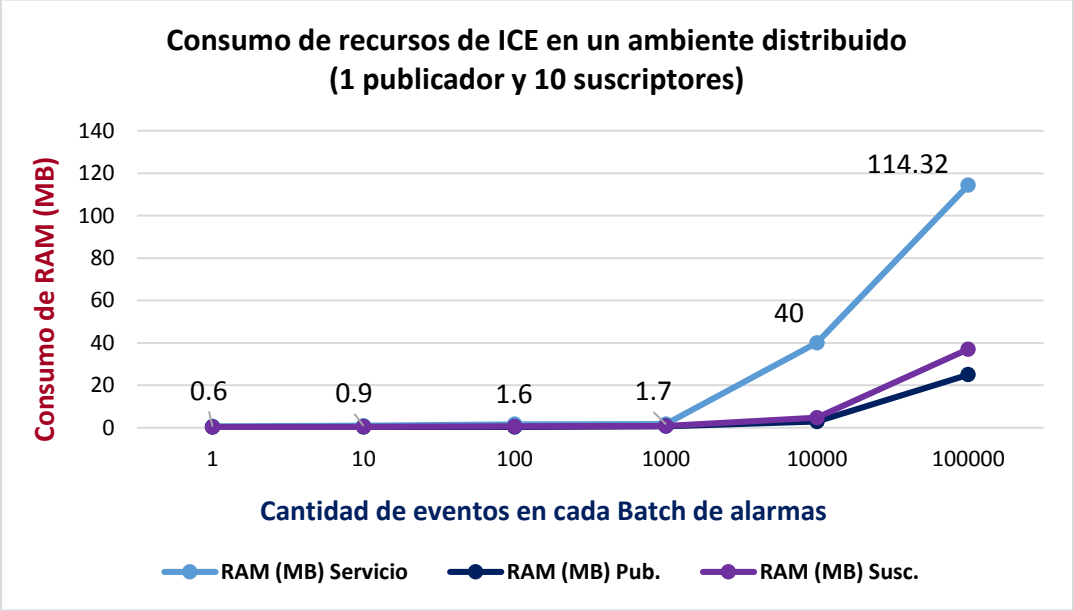

Figura 10. Consumo de recursos de ICE en el escenario de un publicador y diez suscriptores

Tomando como referencia el escenario de 100 mil alarmas, se alcanzó el mejor tiempo de 115 ms con (véase Anexo 3):

- Una tasa de transferencia de 83.74 MB/segundos,
- Un rendimiento promedio de 737664.4705 Eventos/segundos y
- Una latencia promedio 1.36696E-06 segundos.

2.1.2.3. Desarrollo de la matriz de decisión

A partir del análisis realizado en los epígrafes 1.3.4 y 2.1.2, los criterios de especialistas pertenecientes a la Empresa CEDAI Villa Clara, Cuba, y los propios criterios de especialistas del SCADA en función de los requisitos que exige este tipo de soluciones, se decide entonces evaluar cualitativa y cuantitativamente el comportamiento de cada tecnología frente a los siguientes aspectos:

1. Facilidad de aprendizaje
2. Soporte de parte de sus desarrolladores o proveedores
3. Calidad de la documentación
4. Código fuente disponible
5. Soporte de lenguajes y sistemas operativos
6. Fiabilidad (otras políticas de QoS)
7. Uso eficiente de los recursos (Consumo de memoria y CPU)
8. Tolerancia a fallas (Redundancia y/o persistencia de las conexiones)
9. Otros mecanismos de comunicación además de RPC
10. Rendimiento y latencia.

El objetivo de la matriz de decisión que se presenta en la Tabla 5 es seleccionar entre varias alternativas, la adecuada para alcanzar el objetivo esperado, a partir de comparar los resultados después de aplicar un peso (importancia) a cada criterio de evaluación y multiplicarlo por la calificación asignada a cada tecnología según su correspondencia con cada criterio de selección. Dónde: Importancia será un valor en un rango entre uno y 100; Puntaje será igual a Valor de Calificación (Calif.) * Importancia; por último, la Calificación de cada criterio será: “0” No Aceptable, “1” Poco Aceptable, “2” Aceptable, “3” Sobresaliente, y “4”, Excelente.

Se debe destacar que los valores de Importancia y Calificación otorgados se decidieron como parte de un análisis en equipo, que incluyó los propios expertos anteriormente

mencionados, los especialistas de la línea de Comunicaciones del SCADA y el autor de la presente investigación. Se tuvieron en cuenta el impacto de cada criterio en la propuesta de solución que se requiere.

Tabla 5. Matriz de decisión (Evaluación de alternativas tecnológicas)

Modelo de decisión		ALTERNATIVAS					
		TAO + MessagePack		ZeroMQ + MessagePack		ICE	
Criterios	**Importancia**	**Calif.**	**Puntaje**	**Calif.**	**Puntaje**	**Calif.**	**Puntaje**
Facilidad de aprendizaje	5	1	5	4	20	3	15
Soporte de parte de sus desarrolladores o proveedores	5	1	5	3	15	4	20
Calidad de la documentación	5	1	5	2	10	4	20
Código fuente disponible	10	4	40	4	40	4	40
Soporte de lenguajes y sistemas operativos	10	4	40	3	30	4	40
Fiabilidad (otras políticas de *QoS*)	10	3	30	2	20	4	40
Uso eficiente de los recursos (Consumo de memoria y CPU)	10	4	40	4	40	4	40
Tolerancia a fallas (Redundancia y/o persistencia de las conexiones)	10	3	30	0	0	4	40
Otros mecanismos de comunicación además de RPC	15	4	60	3	45	4	60
Rendimiento y latencia	20	2	40	3	60	4	80
TOTAL	100	28	**295**	29	**280**	39	**395**

Los atributos que se evaluaron cuantitativamente a partir de pruebas experimentales y pruebas de conceptos fueron:

- Uso eficiente de los recursos (Consumo de memoria y CPU): las tres tecnologías ofrecen resultados similares, sin impactar en la decisión final.
- Tolerancia a fallas (Redundancia y/o persistencia de las conexiones): ICE y TAO ofrecen estas características, no así en el caso de ZeroMQ. En el caso de ICE se garantiza de manera fácil y configurable, sin embargo TAO exige incorporar nuevas líneas de código a la solución.
- Otros mecanismos de comunicación además de RPC: las tres tecnologías soportan varios paradigmas de comunicación, siendo más eficientes en ICE y TAO.

- Rendimiento y latencia: en cuanto a la cantidad de eventos por segundo y el tiempo que transcurre entre el envío de un evento y otro, se obtuvieron resultados alentadores, sin embargo con ICE se logra mejorar siete veces el rendimiento del resto de las alternativas, siendo ZeroMQ ligeramente superior a TAO.

Los resultados expresados en la matriz demuestran la superioridad de ICE sobre TAO y ZeroMQ, no solo por garantizar un mejor rendimiento durante las comunicaciones, sino por proveer a la solución de mejores políticas de *QoS*.

2.2. Descripción de alto nivel de la solución

A continuación se relacionan las funciones y responsabilidades de cada uno de los subsistemas que forman parte de la propuesta de solución abordada en el Epígrafe 2.1.1.

2.2.1. Subsistema *Data Definition*

Este va encontrarse en el ámbito de los componentes del SCADA que utilizarán el Middleware y brindará a éstos las definiciones de los tipos de datos a transmitir por la red. Define la lógica para alcanzar una uniformidad en los datos que se manejan en todos los escenarios del sistema distribuido, lo que garantizará además un mejor rendimiento durante las comunicaciones pues se elimina del ámbito del Middleware la responsabilidad de serializar/deserializar los datos, será este tipo de dato el que utilizará tanto el Middleware como el resto de los módulos involucrados en el proceso.

Entre sus principales responsabilidades se encuentran:

- Encapsular las definiciones de los datos, incluyendo tanto definiciones realizadas en C++ como definiciones SLICE.
    - *Slice* es el mecanismo de abstracción principal para la separación de interfaces de objetos de sus implementaciones. Establece un contrato entre el cliente y el servidor que describe los tipos y las interfaces de objetos utilizados por una aplicación. Esta descripción es independiente del lenguaje de programación, por lo que no importa si el cliente está escrito en el mismo lenguaje que el servidor (Henning & Spruiell, 2013).
- Proveer los diferentes tipos de datos que soporta el Middleware, en este caso todos los que deben manejarse en los procesos que involucra el SCADA.
- Permitir la creación de instancias de los tipos de datos que se requiera transmitir por la red en un momento dado.

- No solo se utilizarán los datos para ser enviados por la red, también se podrán instanciar los datos para subscribirse a ellos.

2.2.2. Subsistema *Communication Manager*

Va a residir en el espacio de los módulos del SCADA que utilizarán el Middleware y proveerá las interfaces de comunicación distribuida para las interacciones síncronas y asíncronas. Define la lógica para realizar invocaciones remotas como si estuviera en una computadora de manera local, suscribirse y publicar datos en tiempo real. Permitirá ocultar al resto de los componentes del SCADA la lógica del Middleware y la forma en que la tecnología resuelve las peticiones.

Entre sus principales responsabilidades se encuentran:

- Exponer las interfaces, *stub (*código del lado del cliente*)* y *skeleton* (código del lado del servidor) generados a partir de la definición de interfaces en el *Slice* ej.: "*send.ice*". Los mismos son utilizados para comunicar clientes con servidores y publicadores con suscriptores basados en ICE y lograr un entendimiento entre ellos.
- Instanciar los diferentes tipos de objetos, según la postura que va asumir el módulo en el SCADA, dígase: cliente, servidor, publicador o suscriptor.
- Realizar los tipos de solicitudes siguientes:
    - Envío y recepción de variables, alarmas, eventos, bitácoras, estados de la comunicación, comandos y respuestas a comandos en tiempo real.
    - Creación dinámica de canales de comunicación que permitan la suscripción a varios tipos de datos y por diferentes canales en una única sentencia.
    - Invocación de métodos remotos implementados por el servidor de Seguridad del SCADA, por ejemplo para autenticación y gestión del control de acceso.
- Encapsular y abstraer la implementación de un cliente basado en ICE para hacer peticiones a procedimientos remotos asociados al módulo de Seguridad del SCADA.
- Establecer mediante una interacción punto a punto basada en ICE una comunicación con el servidor, para invocar los métodos remotos implementados por el servidor de Seguridad del SCADA, entre ellos:
    - Autenticación al sistema desde la interfaz HMI.
    - Determinación del tipo de autenticación.
    - *LogOut* del sistema.
    - Accesibilidad al sistema y los recursos.

    - Determinación del tipo de acceso.
    - Monitoreo de usuarios conectados.
    - Etc.
- Encapsular y abstraer la implementación de un servidor basado en ICE para resolver las peticiones a procedimientos remotos asociados al módulo de Seguridad del SCADA.
- Establecer mediante una interacción punto a punto basada en ICE una comunicación con el cliente, para dar respuesta a los métodos remotos implementados por el servidor de Seguridad del SCADA, entre ellos:
    - Registrar una instancia de un objeto con los métodos que puede invocar el cliente.
    - El servidor de Middleware será quien recibirá la solicitud pero en sí se estará invocando y ejecutando un método remoto implementado en el contexto del servidor de Seguridad del SCADA, el cual brinda al Middleware una interfaz para poder acceder a estos procedimientos.
- Iniciar cada instancia de servidor a la vez que esté listo para recibir peticiones remotas.
- Encapsular y abstraer la implementación de un suscriptor basado en ICE para suscribirse a uno o varios datos de tipo complejo.
    - Recepción de datos en tiempo real del tipo puntos, alarmas, eventos, bitácoras de usuario, estado de la comunicación y respuesta de comandos.
    - Gestión y control de la lista de suscriptores asociados a cada tópico o canal de comunicación.
    - Inicio y detención del proceso de recepción por tipos de datos y tópicos.
- Encapsular y abstraer la implementación de un publicador basado en ICE para publicar uno o varios datos de tipo complejo.
    - Publicación en tiempo real de datos complejos del tipo punto, alarmas, eventos, bitácoras de usuario, estado de la comunicación y comandos.
    - Manejo de los hilos a usar por cada publicador.

2.2.3. Subsistema *Transport*

Este define la lógica que garantiza las respuestas a las peticiones realizadas por los componentes del SCADA. Encapsula las funcionalidades de los objetos de ICE del tipo cliente, servidor, publicador y suscriptor. En este caso el acceso a las funcionalidades que

él expone, será a través de interfaces virtuales puras bien definidas, como se establece en el epígrafe anterior.

Debe proveer un *Broker* para establecer el contrato de comunicación entre los entes a cada lado de la red, mediante un canal seguro de comunicación y sustentado con políticas de *QoS*.

Los servicios de ICE que realizan la función de puente entre las partes a cada lado de la red, son implementados como servidores y las aplicaciones que los usan actúan como clientes, entre ellos se encuentran (Henning & Spruiell, 2013):

- Servidor *IceBox:* es un servidor de aplicaciones simple que puede orquestar el arranque y la parada de un número de componentes de aplicación. Los componentes de la aplicación se pueden implementar como una librería dinámica en lugar de un proceso.

- *IceStorm*: es un servicio de publicación/suscripción que desacopla los clientes y servidores. Fundamentalmente, actúa como un distribuidor de eventos. Los publicadores envían eventos al servicio, que, a su vez, pasa los eventos a los suscriptores. De esta manera, un único evento publicado puede ser enviado a varios suscriptores.

2.3. Flujo de comunicación entre los componentes que forman parte de la propuesta

En la Figura 11 se muestra el flujo de comunicación entre los subsistemas sin tener en cuenta las *QoS*. En este flujo se representan a través de mensajes direccionales, las actividades y responsabilidades fundamentales de los componentes en el proceso de ejecución de operaciones síncronas o asíncronas.

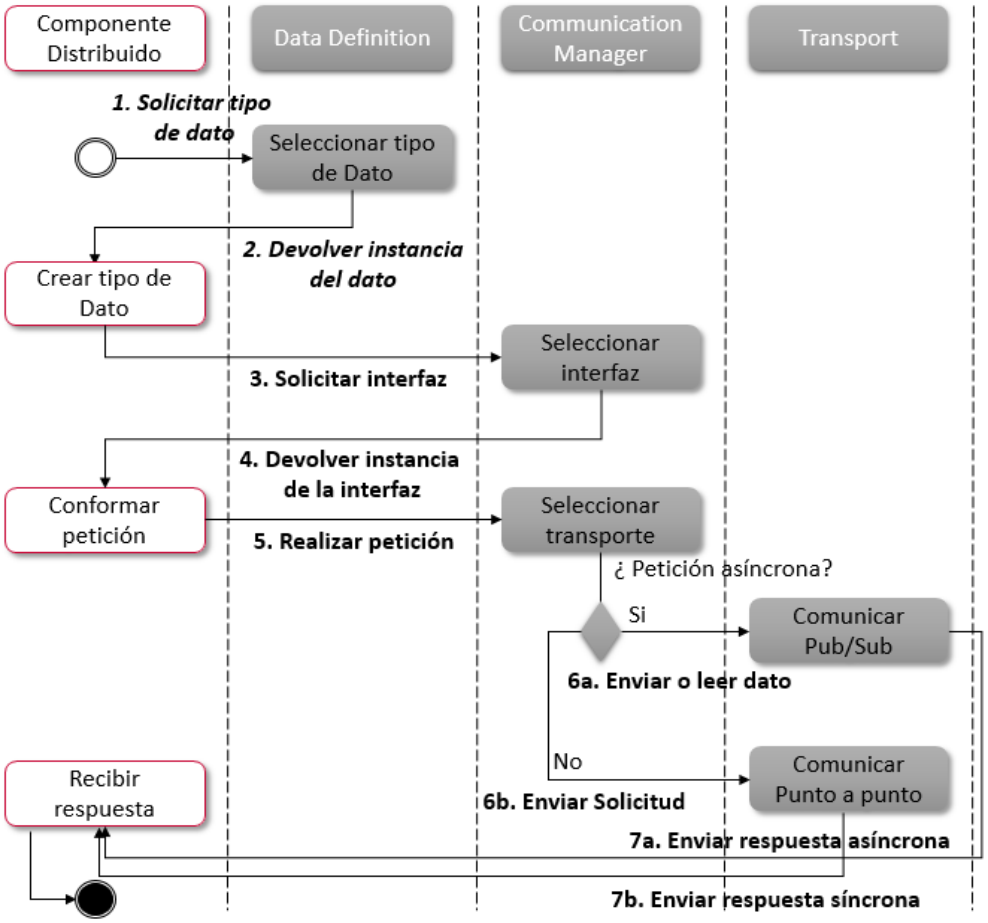

Figura 11. Flujo de comunicación entre los subsistemas del Middleware

2.4. Conclusiones del capítulo

La puesta en práctica de una solución híbrida de las tecnologías TAO y ZeroMQ con el sistema de serialización MessagesPack permitió mejorar el rendimiento de las tecnologías en sí mismas.

ICE resutó ser la tecnología de comunicación interprocesos adecuada para ejercer como soporte a las interfaces de comunicacion, siendo siete veces más rápida que el resto de las alternativas analizadas.

La propuesta garantiza la abstracción del transporte manteniendo la concurrencia en los mensajes y elimina el proceso de conversión de los datos a cada lado de las comunicaciones lo que permitirá disminuir el número de peticiones.

Capítulo 3. Implementación de interfaces de comunicación distribuida entre los componentes de un sistema SCADA

Formando parte de la implementación y validación se describen en este capítulo los requisitos no funcionales que debe soportar la propuesta, una vista de la arquitectura así como los elementos asociados al diseño, implementación y despliegue de las interfaces. Se enuncian las principales herramientas y tecnologías que formarán parte del ambiente de desarrollo y finalmente se presentan los resultados de la verificación y validación de la propuesta de solución.

3.1. Características generales de la solución

Este apartado engloba las metas y objetivos generales del sistema de forma clara y completamente definidos a través de los requisitos no funcionales.

Las funcionalidades que debe cubrir cada uno de los componentes a implementar como parte de la solución se corresponden con lo expresado en el epígrafe 2.2, sin embargo, atendiendo a la importancia del Middleware dentro del sistema SCADA se precisa detallar los requisitos no funcionales con los que debe cumplir la solución propuesta.

3.1.1. Requisitos no funcionales del Middleware

Por la propia naturaleza de los sistemas SCADA, donde se deben garantizar altos rendimientos durante las comunicaciones, los requisitos no funcionales en su mayoría giran en torno al rendimiento y disponibilidad del sistema propuesto, tanto que son considerados por algunos autores de igual relevancia que los requisitos funcionales. A continuación se describen los más significativos para el sistema.

Requerimientos de software:

- Se requiere que estén instalados los componentes y servicios de ICE a partir de su versión 3.4.1.

Rendimiento:

- Transferencia cómo mínimo de 50 mil variables en menos de un segundo entre dos clientes del Middleware.

Tolerancia a fallas:

- Persistencia de las conexiones entre los módulos del SCADA que se registran en los servicios *IceBox* y *IceStorm*, en el caso de ICE se garantiza a través del servicio *Freeze*.
- Mecanismos de redundancia de los servicios, *IceGrid*.

Portabilidad:

- La solución debe ser multiplataforma para garantizar su portabilidad en los sistemas operativos Linux y Windows.

Usabilidad:

- La cantidad de operaciones en función su integración con los componentes del SCADA debe ser mínima, de manera que se minimice el tiempo de desarrollo y la complejidad de la solución.

3.2. Arquitectura de software de la solución de comunicación

El diseño de la arquitectura siempre es un reto a la hora de desarrollar sistemas distribuidos, se debe modelar una arquitectura que proporcione interfaces de comunicación que oculten a los subsistemas los mecanismos de comunicación y la tecnología. Al mismo tiempo la arquitectura no debe afectar el rendimiento del sistema durante las comunicaciones, un elemento de vital importancia en la supervisión y control de procesos industriales.

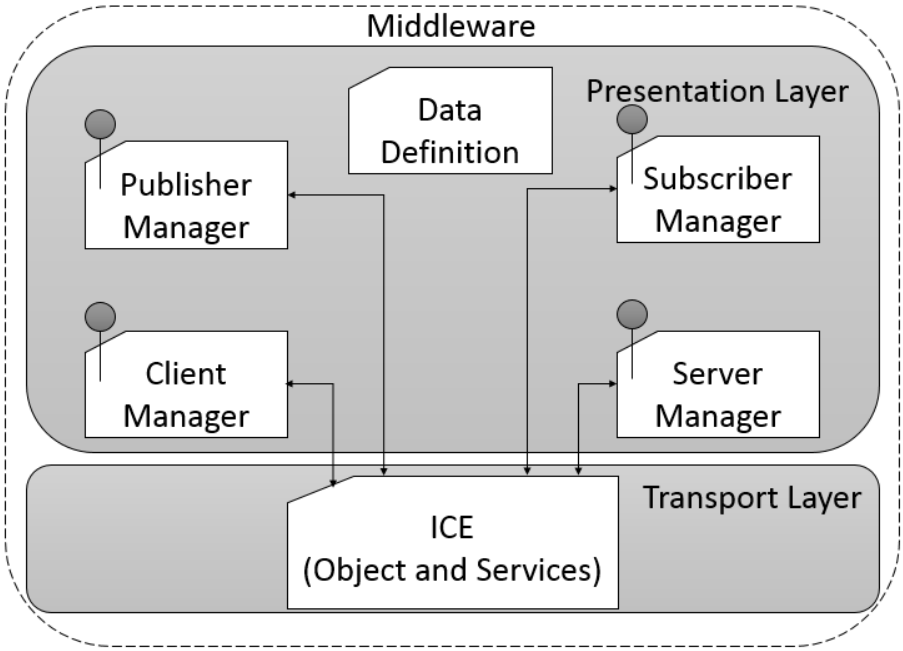

Figura 12. Arquitectura de software en dos capas

La Figura 12 muestra una representación de la arquitectura donde se puede apreciar que la **Capa Presentación** constituye la capa superior, se encarga de exponer a los subsistemas un conjunto de interfaces y clases en C++, haciendo posible el acceso a las funcionalidades para el envío y recepción de datos e invocación de métodos remotos. También se encarga de encapsular la implementación de los mecanismos de comunicación, cliente/servidor y publicación/suscripción basados en ICE.

De forma general en la primera capa se encuentra la descomposición del subsistema ***Communication Manager*** descrito en el capítulo anterior en cuatro nuevos subsistemas que responden a funciones bien específicas de un cliente, servidor, publicador o suscriptor y bridan a su vez una interfaz que será el punto de entrada a los mismos.

Cada uno de ellos puede hacer uso del subsistema ***Data Definition*** con el objetivo de instanciar y acceder a todas y cada una de las definiciones de datos existentes.

Las peticiones de los componentes del SCADA realizadas desde la capa superior se resuelven mediante un enlace con la **Capa de Transporte**. Esta última representa el transporte utilizado para gestionar las comunicaciones, debiendo proveer a la capa superior los objetos y servicios necesarios para lograr la finalidad del subsistema que se desarrolla. Además de brindar mecanismos de seguridad, persistencia de las conexiones y redundancia, atributos de calidad de servicios que garantizan la fiabilidad de las comunicaciones.

3.2.1. Atributos de calidad que ofrece la arquitectura

De forma general la arquitectura propuesta incorpora un grupo de atributos que vienen a imprimir mayor calidad a la solución:

- Sencillez: la vista en dos capas con sus responsabilidades bien definidas hace posible el mejor entendimiento de la propuesta para programadores y usuarios.
- Extensibilidad. Es posible incorporar y/o modificar componentes, funcionalidades y tecnologías sin necesidad de realizar grandes modificaciones.
- Escalabilidad. La solución está preparada para crecer sin perder la calidad del rendimiento y los servicios.

3.3. Ambiente de desarrollo

Para definir las herramientas y tecnologías a utilizar durante la construcción de las interfaces de comunicación se tuvieron en cuenta los dos mecanismos de comunicación

que se proponen y el ambiente de desarrollo donde se soporta el SCADA, por tanto en aras de lograr una homogeneidad entre los productos que se desarrollan en el CEDIN, se hizo la siguiente selección:

A pesar de estar frente a una solución pequeña, se seleccionó la metodología **RUP** (*Rational Unifed Process*) que da la posibilidad de adecuar los procesos a proyectos grandes, medianos y pequeños, además en ella se basan los principales proyectos desarrollados en el centro, específicamente el SCADA.
El enfoque iterativo e incremental de RUP permite la retroalimentación entre las fases y garantiza ir teniendo una idea y un resultado de mayor calidad al finalizar cada iteración (Bashir & Qureshi, 2012).

Como tecnología base de comunicación o transporte fue seleccionada la tecnología **ICE** fundamentada en el epígrafe 2.1.2, siendo en la actualidad la tecnología Middleware más completa y robusta.
ICE es una herramienta orientada a objetos moderna que permite la construcción de aplicaciones distribuidas con el mínimo de esfuerzo. Permite enfocar el esfuerzo en la lógica de la aplicación, y se encarga de todas las interacciones con las interfaces de red de bajo nivel. Con ICE, no es necesario preocuparse por los detalles de cómo abrir las conexiones de red, serializar/deserializar los datos para su transmisión por la red o de molestarse por atender las fallas de conexión (Henning & Spruiell, 2013).

C++ es el lenguaje principal sobre el cual de desarrollan la mayoría de las aplicaciones en el CEDIN, también al igual que la metodología, este es el lenguaje sobre el que implementan todos los componentes del SCADA, por consiguiente la capa de comunicación debe ser compatible con el resto de las partes del sistema.

Son varios los IDE (*Integrated Development Environment*) los que se pueden usar para desarrollar aplicaciones en C++, mucho más si no se hace necesario el trabajo con interfaces gráficas, regularmente se usan en el desarrollo del SCADA *QtCreator* y *Eclipse*. Realmente en la presente investigación la plataforma de desarrollo es irrelevante, por tanto se decide por ***Eclipse*** porque involucra menos dependencias y paquetes y se integra perfectamente con C++ y las bibliotecas de ICE.

3.4. Diseño de las interfaces de comunicación distribuida

Siendo una de las principales responsabilidades del Middleware la exposición de interfaces de comunicación, se muestra un diseño de alto nivel a través de las vistas más significativas para la arquitectura, compuestas por subsistemas de diseño encargados de definir y exponer las interfaces necesarias para garantizar la comunicación publicación/suscripción y cliente/servidor, como se evidencia en las Figuras de la 13 a la 16.

Para cada escenario se utilizan patrones de diseño con el fin de robustecer la solución, el patrón fachada (*facade*) está representado en todas las interfaces que exponer un conjunto de funcionalidades. Se utilizan clases "*Resolver"* que no son más que fábricas (*factory*) que permiten crear instancias de las implementaciones para cada una de las interfaces.

Cada subsistema involucra al menos una interfaz virtual pura (sin implementación) como *IClient, IServer, IPublisher y ISubscriber* según corresponda, cada una de ellas haciendo uso del patrón *facade* para exponer el conjunto de funcionalidades que encapsulan el funcionamiento de todo un subsistema.

Existe un objeto intermedio que va a garantizar la implementación adecuada para cada interfaz pura, este hace la función del patrón *factory.* No es más que un *Resolver* encargado de enlazar cada interfaz con su implementación, según corresponda.

La combinación de las interfaces con los *Resolver* hace posible el ocultamiento de los mecanismos de comunicación y sus complejidades.

3.4.1. Subsistemas de la capa de presentación

Subsistema *Client Manager*: define la lógica de comunicación para las llamadas remotas del lado del cliente. Representa un cliente de Middleware basado en la tecnología ICE, accesible mediante la interfaz *IClient* como se presenta en la Figura 13.

A través de la clase *ClientResolver* se obtiene una instancia de la interfaz cliente, que llevará en su interior un objeto del tipo *IceClient*, quien implementa la interfaz, ejecutando sus métodos por polimorfismo en el momento que son invocados desde la interfaz que hace de fachada, *IClient*.

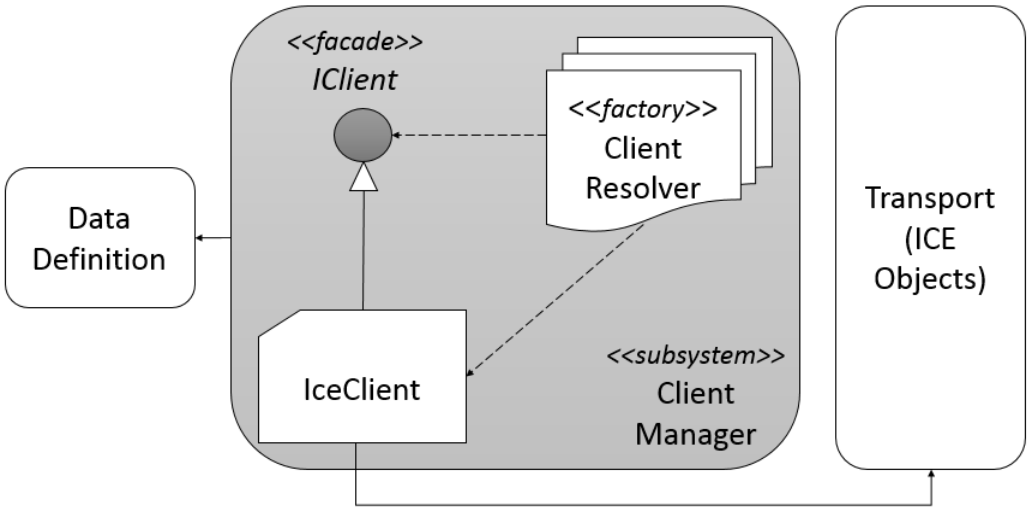

Figura 13. Vista Lógica del subsistema *Client Manager*

La clase *IceClient* tiene la responsabilidad de enviar la solicitud del cliente mediante un enlace con los objetos y servicios de ICE.

El resto de los subsistemas tendrá un comportamiento similar pero utilizando otras clases y objetos.

Subsistema *Server Manager*: define la lógica de comunicación para las llamadas remotas del lado del servidor. Representa un servidor de Middleware basado en la tecnología ICE, accesible mediante la interfaz *IServer* como se presenta en la siguiente figura.

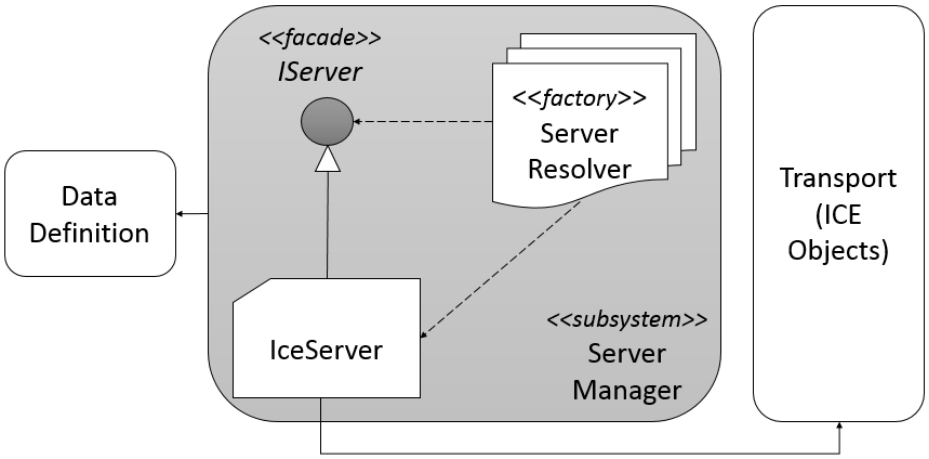

Figura 14. Vista lógica del subsistema *Server Manager*

A través de la clase *ServerResolver* se obtiene una instancia de la interfaz del servidor, que llevará en su interior un objeto del tipo *IceServer*, quien implementa la interfaz, ejecutando sus métodos por polimorfismo en el momento que son invocados desde la interfaz que hace de fachada, *IServer*.

En este caso particular el módulo del SCADA debe entregar al Middleware la referencia al objeto remoto que implementa las funciones que podrán ser invocadas desde el cliente. De esta forma se garantiza dar una respuesta adecuada al cliente haciendo parecer que el proceso de solicitud/respuesta se realiza de manera local y no remota.

La clase *IceServer* tiene la responsabilidad de enviar la respuesta del servidor mediante un enlace con el servicio *IceBox* de ICE.

Subsistema *Subscriber Manager*: define la lógica de suscripción a uno o varios tipos de datos complejos, las operaciones del lado cliente pero bajo el paradigma publicación/suscripción. Representa un suscriptor del Middleware basado en la tecnología ICE, accesible mediante la interfaz *ISubscriber* como se presenta en la siguiente figura.

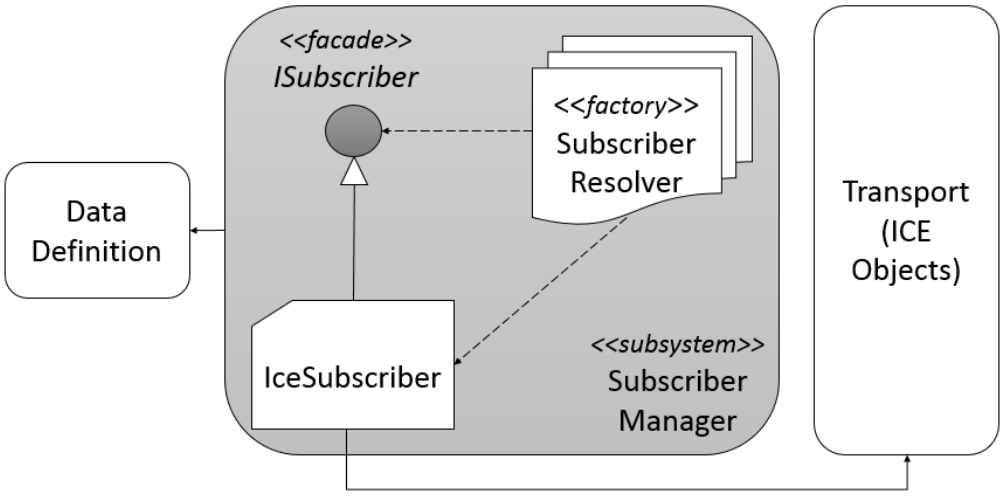

Figura 15. Vista lógica del subsistema *Subscriber Manager*

A través de la clase *SubscriberResolver* se obtiene una instancia de la interfaz del suscriptor, que llevará en su interior un objeto del tipo *IceSubscriber*, quien implementa la interfaz, ejecutando sus métodos por polimorfismo en el momento que son invocados desde la interfaz que hace de fachada, *ISubscriber* en este caso.

La clase *IceSubscriber* tiene la responsabilidad de enviar la solicitud de suscripción mediante un enlace con los objetos y servicios de ICE, el *IceStorm* específicamente.

Subsistema *Publisher Manager*: define la lógica de publicación de uno o varios tipos de datos complejos, las operaciones del lado de un servidor pero bajo el paradigma publicación/suscripción. Representa un publicador del Middleware basado en la tecnología ICE, accesible mediante la interfaz *IPublisher* como se presenta en la Figura 16.

A través de la clase *PublisherResolver* se obtiene una instancia de la interfaz del publicador, que llevará en su interior un objeto del tipo *IcePublisher*, quien implementa la interfaz, ejecutando sus métodos por polimorfismo en el momento que son invocados desde la interfaz que hace de fachada, *IPublisher* en este caso.

La clase *IcePublisher* tiene la responsabilidad de enviar la solicitud de publicación mediante un enlace con el servicio *IceStorm* específicamente.

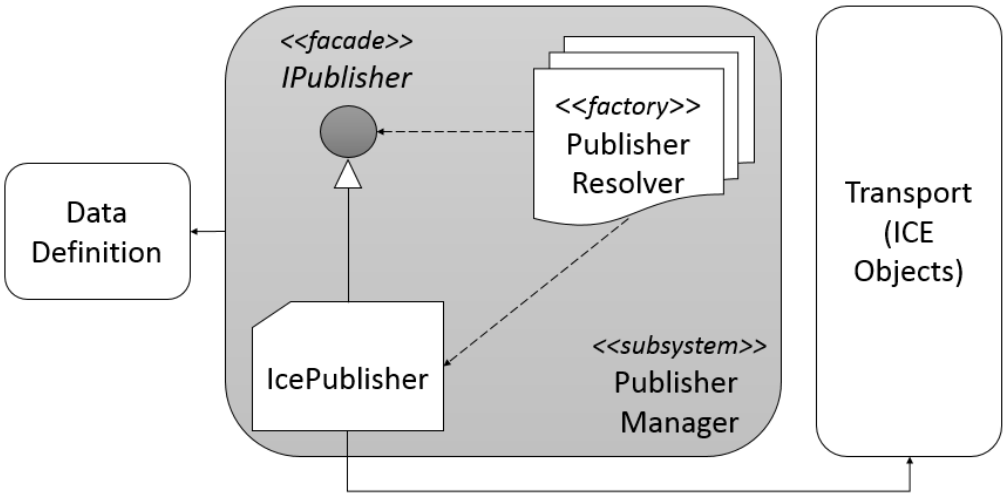

Figura 16. Vista lógica del subsistema *Publisher Manager*

3.5. Implementación de las interfaces de comunicación distribuida

Los componentes representan la implementación de las clases del diseño con las que tienen trazas. Cada uno de los componentes que se implementan se construye como biblioteca de datos y clases agrupando uno o más subsistemas del diseño en cada caso. ***DataTypes*** encapsula las definiciones de datos (*Data Definition*), ***SyncComm*** implementa las clases del modelo síncrono (*Client y Server Manager*) y ***AsyncComm*** las clases para

la comunicación asíncrona (*Publisher y Subscriber Manager*), incluyendo en cada modelo las interfaces asociadas.

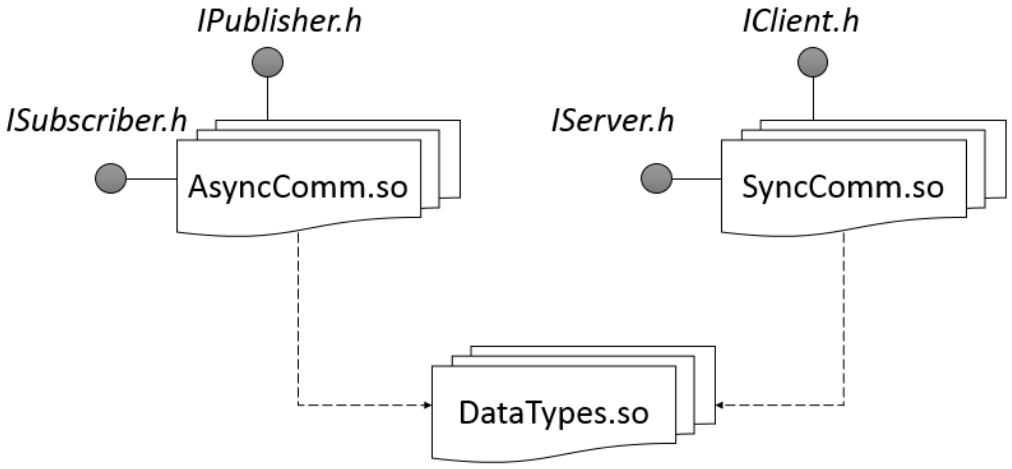

Figura 17. Diagrama de componentes del Middleware

Los tres componentes se construyen como parte de una biblioteca de extensión .so producto a la compilación de la solución desde el sistema operativo Linux, en su distribución *Debian* 6. La solución propuesta puede ejecutarse en Windows, solo debe compilarse el código fuente en este sistema operativo obteniéndose un fichero con extensión .dll como producto final en lugar de un .so.

En el caso de los componentes *AsyncComm* y *SyncComm* no deben verse las interfaces como elementos aislados de las bibliotecas, ellas se incluyen también como parte de la solución, simplemente se pretende acercar aún más al lector a los elementos de comunicación explicados en epígrafes anteriores sobre el funcionamiento del Middleware y las interfaces que se brindan a los subsistemas del SCADA.

Las tres bibliotecas utilizan objetos y servicios de ICE, encapsulando elementos de ambas capas de la arquitectura.

3.5.1. Descripción de las políticas de calidad de servicio

A pesar de ser el tiempo de transferencia de los datos en las interacciones publicación/suscripción la principal variable en la presente investigación y el principal objetivo la disminución de éste, la solución incorpora el mecanismo de comunicación cliente/servidor y un grupo de políticas de *QoS* para lograr una solución de comunicación completa, robusta y fiable en ambientes distribuidos con requerimientos críticos. Entre las

principales políticas que se tienen en cuenta en la implementación y además se incluyen en la misma, se encuentran: la persistencia de las conexiones y la redundancia del servicio *IceStorm*.

3.5.1.1. Persistencia de las conexiones

La persistencia de las conexiones permitirá conocer el estado de los objetos conectados después de la caída de uno o varios servicios, haciendo más fácil la disponibilidad y la redundancia. Para lograr persistir las conexiones la propuesta se basa en las propias herramientas que ofrece ICE sin necesidad de generar código fuente adicional.

ICE tiene un servicio de persistencia de objetos, conocido como *Freeze*. Hace fácil el almacenamiento de los estados de los objetos en una base de datos, estados que se definirán desde un inicio en el fichero *Slice* y el servicio generará el código compilado para almacenar o devolver el estado de los objetos desde y hasta la base de datos. *Freeze* usa *Berkeley DB* como base de datos y solo es necesario incluir una línea como parte del fichero de configuración del *IceStorm*.

3.5.1.2. Redundancia de los servicios

Esta característica siempre será necesaria en sistemas que deban estar operativos las 24 horas, y con la capacidad de reponerse a las fallas que puedan surgir. Entonces será preciso contar con una réplica idéntica de los principales servicios y servidores, donde los que se encuentren en estado pasivo sean capaces de activarse después de un fallo en el nodo activo, de forma transparente para el usuario y el resto de los componentes del sistema.

El SCADA soporta mecanismos de redundancia en sus principales subsistemas pero en muchos casos son irrelevantes sino es posible persistir las conexiones y mucho menos si los servicios que garantizan la transferencia de datos se apagan y no se reinician de forma automática y transparente después de una falla.

Es por ello que la solución propuesta también incorpora como principal mecanismo de tolerancia a fallas, la redundancia del servicio *IceStorm*, garantizando de esta manera que después de un fallo, los subsistemas continúen comunicándose de manera normal y manteniendo su estado inicial. Todo este proceso es trasparente para el usuario y se garantiza también sin costo de desarrollo, mediante configuración del servicio *IceGrid*.

Este servicio va a garantizar la localización y activación de servicios homogéneos que mantienen el mismo estado usando una posible configuración *Grid*, donde cada computadora es esencialmente un clon de las otras y todas son capaces de manejar las tareas de igual manera. Este servicio además permite:

- Activación de servidores por demanda
- Replicación y balanceo de carga
- Consultas dinámicas
- Monitoreo del estado de los servicios

3.6. Despliegue de las interfaces de comunicación distribuida

Con el objetivo de representar la distribución de los componentes e interfaces en los diferentes nodos físicos de la red, se propone el diagrama de despliegue de la manera en que se observa en la Figura 18.

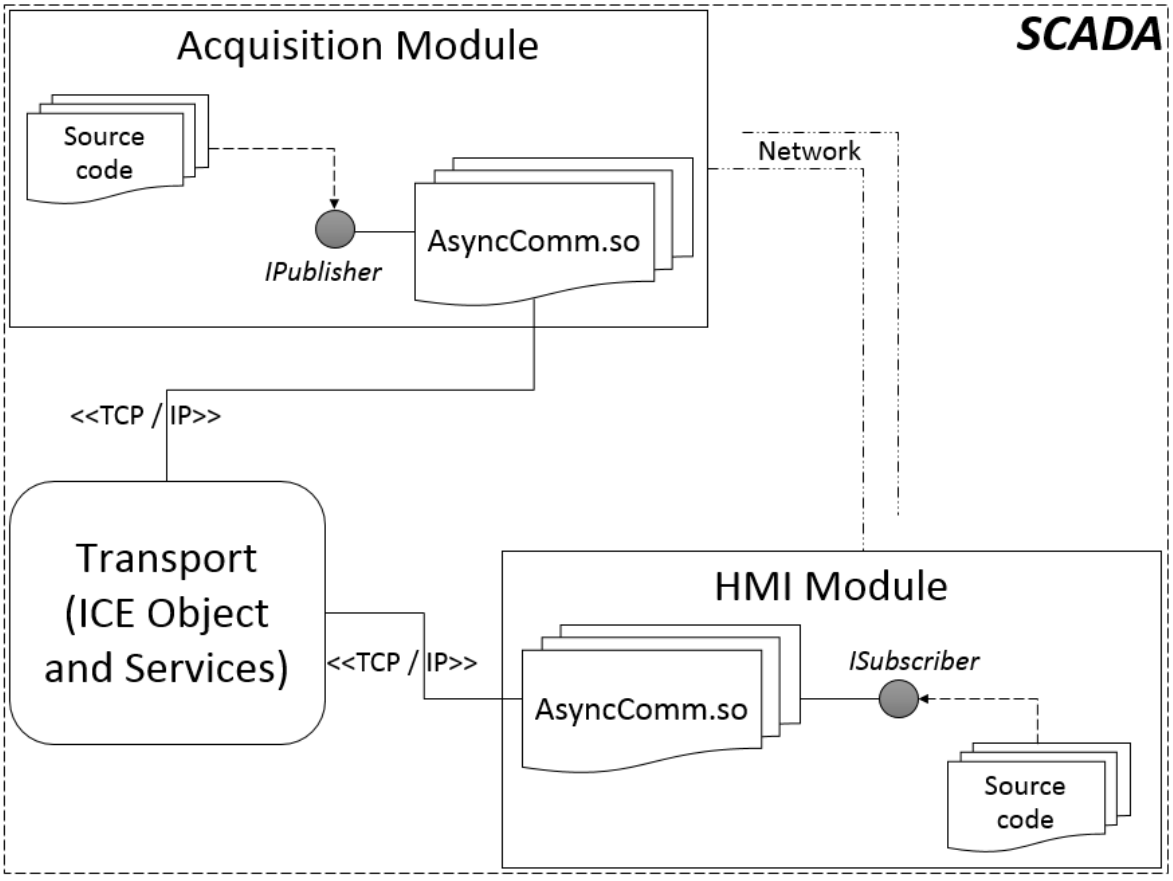

Figura 18. Despliegue de las interfaces de comunicación

El despliegue simula dos nodos de ejecución, el primero de ellos con la participación del módulo de Adquisición del SCADA que juega el rol de publicador de datos y el segundo con la participación del módulo HMI del SCADA en el rol de suscriptor.

EL SCADA es un sistema por naturaleza distribuido pero puede ejecutarse y trabajar correctamente si se ejecutan todos sus servidores, servicios y módulos de manera local o monolítica.

En el ambiente distribuido que se ha presentado se están comunicando dos de los principales componentes del SCADA, el subsistema de Adquisición, donde llegan los datos desde el campo a través de los manejadores y se publican en tiempo real utilizando la interfaz *IPublisher*. Este último publica los datos en los correspondientes tópicos o temas registrados en el servicio *IceStorm* dedicado a las transmisiones publicación/suscripción. De manera similar sucede si se desea realizar una invocación remota, por ej.: una llamada al subsistema de Seguridad del SCADA, pero usando la biblioteca *SyncComm* y a través del *IceBox*.

Al otro lado de la comunicación se encuentra el HMI, el que se suscribe a un grupo de datos que desea monitorear, en este caso también usando la biblioteca asíncrona *AsyncComm* y solicitando finalmente los datos al servicio *IceStorm*. Puede hacer función tanto de cliente como de servidor a través de las interfaces síncronas del Middleware y con el uso del *IceBox*.

En ambos nodos deben configurarse los datos asociados a la dirección Ip, Puerto y Canal para el caso de una comunicación publicación/suscripción, diferente a la comunicación cliente/servidor que solamente requiere los dos primeros parámetros.

El diagrama de despliegue representa solo dos nodos, el funcionamiento real del SCADA involucra otro grupo de componentes los cuales van a tener similar comportamiento en dependencia de su rol en el sistema.

Las bibliotecas se van a encargar de hacer llegar las peticiones al enrutador de los mensajes, en este caso los servicios de ICE mediante una conexión TCP/IP y de manera transparente tanto para el usuario como para los módulos del SCADA.

En el caso de la comunicación sincrónica existe un orden en las interacciones, primeramente el cliente debe realizar una solicitud, quedando bloqueado hasta tanto exista una falla o reciba una respuesta del servidor. En los escenarios asincrónicos no interesa el orden, una parte puede publicar información sin necesidad de conocer si del otro lado ya fueron solicitados estos datos y de manera no bloqueante para ambas partes.

Si se desea tener acceso a todas las funcionalidades del Middleware, es necesario instanciar las tres bibliotecas, en caso contrario simplemente la que encapsula el paradigma de comunicación de su interés.

Como principio, es importante que los servidores del SCADA y los servicios del Middleware se ejecuten en el nodo principal (Adquisición en este caso), éste será siempre el más seguro, con posibilidades de respaldo eléctrico, mejores prestaciones en disco, memoria y procesamiento, además de contar con mecanismos de redundancia. De esta forma se garantizará la disponibilidad del sistema que captura los datos del campo y del propio Middleware que permitirán quien se encarga de publicar dicha información.

3.7. Pruebas de software. Verificación y validación

Para verificar el correcto funcionamiento de las interfaces de comunicación implementadas sobre la tecnología Middleware ICE y validar el cumplimiento del objetivo propuesto, se realizaron un conjunto de pruebas de software.

Entre los niveles de pruebas se encuentran las de unidad, integración, validación y sistema. El objetivo fundamental de las pruebas de unidad es centrar el proceso de verificación en la menor unidad de diseño de software: el componente de software o modulo, es decir, probar cada parte del software por separado. Las pruebas de integración no son más que tomar los módulos probados en las pruebas de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño. Tras terminar las pruebas de integración, el software está completamente ensamblado como un paquete, se han encontrado y corregido los errores y se puede comenzar una serie final de pruebas de software: las pruebas de validación. Finalmente, el software es incorporado a otros elementos del sistema, por ejemplo, nuevo hardware, información, bases de datos, etc. y se realizan una serie de pruebas de integración del sistema y de validación (Ruiz Tenorio, 2010).

Al sistema propuesto se le realizaron un conjunto de pruebas de caja negra con el objetivo de medir la funcionalidad operativa del mismo, a cada uno de los módulos por separado, a la comunicación e integración de los mismos y al Middleware de forma integral, para determinar si cumple con todas las funcionalidades especificadas.
Las pruebas de unidad permitieron comprobar que cada una de las funcionalidades previstas para cada componente tiene correspondencia con los requisitos especificados y funcionan correctamente, pudiéndose comprobar que:

- La biblioteca *DataTypes* permite instanciar los diferentes tipos de datos manejados en el SCADA.
- La biblioteca *AsyncComm* permite desde las interfaces que provee invocar los métodos para enviar y recibir datos de tipo punto, alarma, evento, etc.
- La biblioteca *SyncComm* permite desde sus interfaces, la invocación de métodos remotos y el registro de los objetos.

Las pruebas de integración arrojaron que es posible integrar los componentes según corresponde, tanto *AsyncComm* como *SyncComm* permiten desde sus espacios utilizar los datos encapsulados en *DataTypes*, crear una lista de un determinado tipo e invocar la publicación o suscripción del mismo. Además, también se comprobó el correcto registro de estos componentes en los servicios de ICE, elemento fundamental para garantizar la interoperabilidad entre las aplicaciones distribuidas.

Mediante las pruebas de validación se logró probar el funcionamiento de las interfaces distribuidas. Tomando como base *stub* de pruebas desarrolladas en C++ que simularon los procesos de publicadores y suscriptores basados en las interfaces propuestas, se comprobó la integración correcta entre los componentes, y se pudo apreciar el resultado más importante, los mensajes llegaron a su destino sin pérdida de información y bajo los tiempos establecidos.

La validación de la solución (diferente a pruebas de validación), se correspondió con las pruebas de sistema, donde la propuesta de solución se integró SCADA para comprobar si realmente era más eficiente que la capa de comunicación actual en cuanto a rendimiento se refiere. El próximo epígrafe describe en detalles los resultados obtenidos.

3.8. Validación de la solución mediante su integración en el SCADA

3.8.1. Ambiente de prueba

Este apartado describe los detalles sobre el ambiente en que desarrollaron los experimentos para validar la solución, especificando los elementos físicos y lógicos.

Recursos físicos

Se utilizaron dos computadores con las siguientes características:

- Computadora 1:
    - Procesador Intel(R) Core (TM) i3-2120 CPU @ 3.30GHz
    - Memoria 4GB de RAM
    - Tarjeta de red a 100Mbit/s

- Computadora 2:
    - Procesador Intel(R) Core (TM) i3-3120 CPU @ 2.50 GHz
    - Memoria de 4GB de RAM
    - Tarjeta de red a 100Mbit/s

Recursos lógicos

Se utilizaron los siguientes elementos de software para realizar los experimentos:

1. Bibliotecas de código fuente del Middleware y sus interfaces como son *DataTypes, AsyncComm y SyncComm.*
2. Ejecutables del SCADA Guardián del ALBA.
3. Códigos de pruebas implementados en C++ que simulan componentes tanto del SCADA como del propio Middleware.

3.8.2. Diseño de las pruebas de sistema

Para comprobar la efectividad de la Propuesta se diseñaron y aplicaron varias pruebas experimentales que permitieron realizar una valoración acerca de la inmediatez con la que se manejan y transmiten grandes cantidades de datos entre los componentes distribuidos. Como parte del SCADA intervinieron los módulos de Adquisición y HMI, comunicándose primero a través de la solución Actual y posteriormente a través de la Propuesta. En ambos casos las características de las pruebas fueron idénticas, donde el módulo de Adquisición jugó el rol de publicador y cinco instancias del HMI el rol de suscriptores.

Para ambas variantes la prueba consistió en medir el tiempo en milisegundos que demoró el proceso de envío y recepción desde un publicador hacia cinco suscriptores ejecutándose remotamente, y realizando diez iteraciones para las cantidades de 6800, 13600, 27200 y 54400 variables configuradas en el SCADA. Además se obtuvo la tasa de transferencia en MB/segundos, la cantidad de Eventos/segundos y la latencia en la mensajería.

La prueba se realizó en dos computadoras con las características expresadas en el epígrafe anterior, donde se obtuvieron resultados muy favorables a pesar de no haber utilizado un ambiente ideal para sistemas SCADA, en referencia tanto a las características físicas de las PC como de la red.

La siguiente tabla describe la media de los tiempos de respuesta obtenidos en cada una de las iteraciones, para cada una de las variantes de solución y en el escenario de 54400 variables considerado éste el más significativo.

Tabla 6. Medias de tiempo obtenidas por cada variante con una muestra de 54400 variables

Publicacion y Suscripción a 54400 variables Tiempo (ms)		
Tamaño (bytes): 217600 - 0.207536 Mb		
No	**Media Actual (ms)**	**Media Propuesta (ms)**
1	662.2	70.4
2	698.2	62.6
3	682.8	68.6
4	639.2	65.6
5	662.2	65
6	668.6	81
7	661.4	59.2
8	652.8	63
9	681.2	66.8
10	661.2	62

La Figura 19 evidencia la superioridad de la Propuesta sobre la solución Actual. Independientemente de los resultados obtenidos, donde son evidentes las diferencias entre los tiempos para cada solución, se procede a demostrar que la diferencia entre las medias es estadísticamente significativa, lo que permitirá defender la propuesta con elementos de más peso.

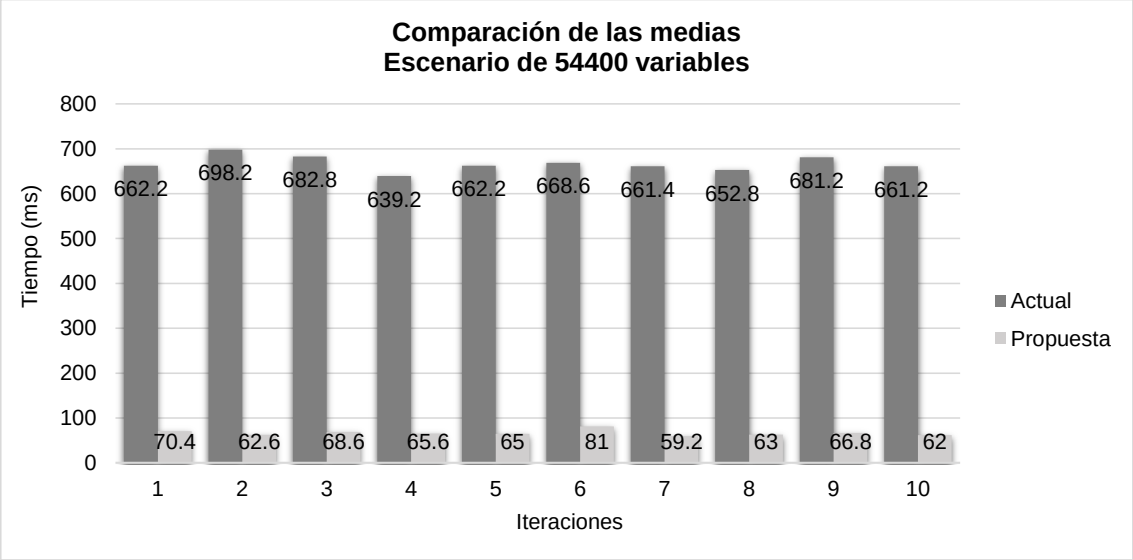

Figura 19. Medias de las respuestas utilizando muestras de 54400 variables

Para obtener mayor información acerca del total de los resultados para esta prueba vea el Anexo 4.

3.8.3. Discusión de los resultados mediante un método estadístico

Para demostrar que realmente la Propuesta ofrece un mejor rendimiento durante las comunicaciones se aplicó un método estadístico para comparar las medias de cada muestra, en este caso la **Prueba t-student para medias independientes**. Se consideran independientes producto a que se realiza la prueba sobre el mismo sistema SCADA, bajo las mismas condiciones y porque no existe interrelación entre las variantes de comunicación.

El método aplicado consistió en:

1. Decidir los valores de medias a comparar, para ello se seleccionaron los descritos en la Tabla 6.
2. Se utilizó el software estadístico *STATGRAPHICS* para automatizar el proceso, donde se aplicó la prueba t a las muestras seleccionadas con el fin de decidir si la hipótesis nula (H_0) se acepta o se rechaza.
3. Se definieron entonces como hipótesis nula y alternativa las siguientes:
    a. H_0: Media-Actual = Media-Propuesta
    b. H_1: Media-Actual > Media-Propuesta

4. Se supuso un riesgo del 5% (o un nivel de confianza del 95%), α=0.05, y grados de libertad igual a 11 definido por la propia herramienta. Se utilizó α ya que dejamos el espacio correspondiente a la región de rechazo para un solo lado.

De la comparación de las medias se obtuvieron diferentes resultados destacándose los siguientes:

Resumen Estadístico

Tabla 7. Resumen estadístico para cada media

	Actual	Propuesta
Recuento	10	10
Promedio	666.98	66.42
Desviación Estándar	16.7094	6.09149
Coeficiente de Variación	2.50523%	9.17116%
Mínimo	639.2	59.2
Máximo	698.2	81.0
Rango	59.0	21.8
Sesgo Estandarizado	0.47633	2.07119
Curtosis Estandarizada	0.258052	2.20407

De particular interés son el sesgo estandarizado y la curtosis estandarizada que pueden usarse para comparar si las muestras provienen de distribuciones normales. La Propuesta tiene un valor de sesgo estandarizado y curtosis estandarizada fuera del rango normal (-2 a +2). Sin embargo, al realizar un análisis más crítico desde el propio software, se recibe como salida que estos valores se encuentran cerca del límite por lo tanto se sigue considerando a la Propuesta como una distribución normal.

Comparación de Medias

- Intervalos de confianza del 95.0% para la media de Actual: 666.98 +/- 11.9532 [655.027, 678.933].
- Intervalos de confianza del 95.0% para la media de Propuesta: 66.42 +/- 4.3576 [62.0624, 70.7776].
- Intervalos de confianza del 95.0% para la diferencia de medias suponiendo varianzas diferentes: 600.56 +/- 12.3322 [588.228, 612.892].
- Valor estadístico de t = 106.782.

De lo anterior se interpreta:

1. El valor de t > 1.7958 (valor crítico de t para una cola), por tanto se rechaza la hipótesis nula lo que demuestra que hay diferencias entre las medias.
    a. El valor crítico se obtiene en tabla t-student de la intersección del nivel de confianza seleccionado con los grados de libertad, obteniendo 1.7958 como valor crítico para rechazar la hipótesis alternativa.
2. Los intervalos de confianza para cada una de las variantes son completamente disjuntos por tanto las diferencia entre las medias se mantiene estadísticamente significativa.
3. El intervalo de confianza para la diferencia de las medias no contiene el valor cero, lo que demuestra que en ningún momento se igualan las medias, por tanto este es otro elemento que prueba que existe una diferencia estadísticamente significativa entre las medias de las dos muestras, con un nivel de confianza del 95.0%.

La Figura 20 describe el comportamiento de las medias de cada variante respecto a la diferencia entre las medias.

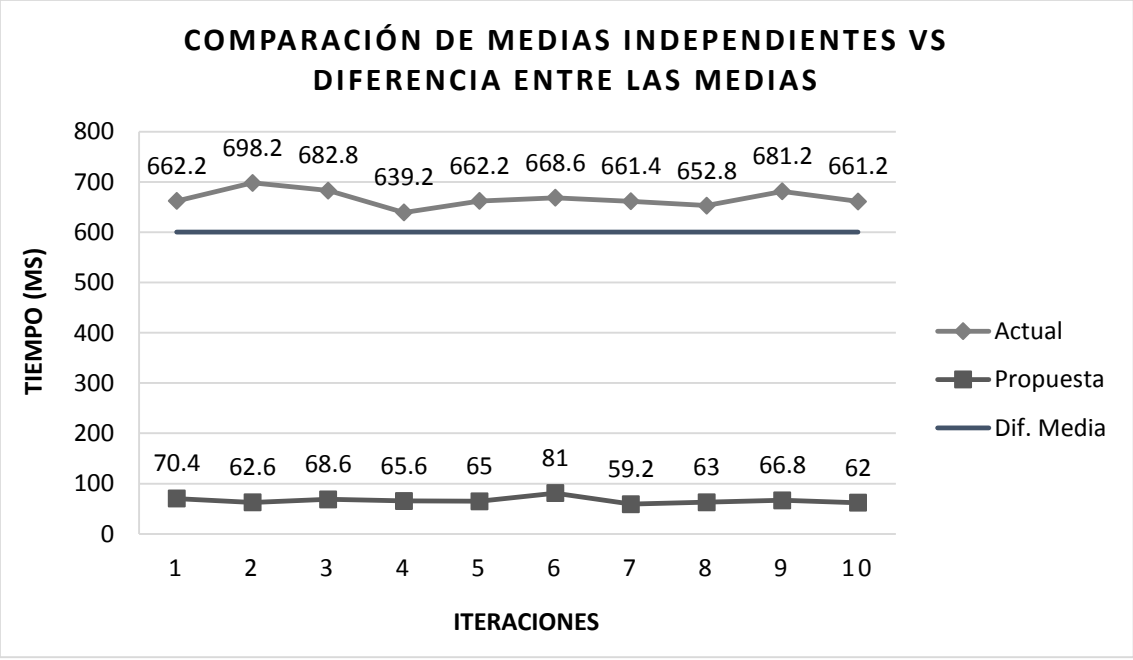

Figura 20. Medias independientes VS Diferencia entre las medias

Después de demostrar que las medias son diferentes, es posible comparar los valores de las medias uno a uno, en cuanto a diferencia de tiempo y diferencia puntual, resultando que la Propuesta es aproximadamente 600 ms o diez veces más rápida que la solución actual.

3.9. Conclusiones del capítulo

La arquitectura en dos capas disminuye sin dudas la complejidad y el número de peticiones, contribuyendo de esta manera a un mejor rendimiento durante las comunicaciones.

La solución actual se basa en socket como mecanismo nativo de comunicación en los sistemas operativos, que por defecto garantiza altos rendimientos y es por ello que se logran tiempos por debajo del segundo, sin embargo, esta solución se comporta inestable a la vez que aumentan la cantidad de datos y de clientes.

Existe una diferencia estadísticamente significativa entre las medias de los tiempos de respuesta obtenidos a favor de la Propuesta. Esta última también ofrece un rendimiento de hasta diez veces superior en relación a la cantidad de MB/segundos y de Eventos/segundos respectivamente.

A pesar de que no se puede garantizar que en un ambiente real, con una red industrial dedicada los resultados sean superiores, existe una alta probabilidad de que la Propuesta presente un comportamiento similar al mostrado en las pruebas realizadas.

Conclusiones

En el presente trabajo se realizó un análisis de las tendencias y tecnologías actuales en referencia a la comunicación interprocesos, principalmente su utilización en entornos distribuidos como los sistemas SCADA. Se desarrollaron un grupo de interfaces de comunicación basadas en la tecnología ICE con el enfoque de abstraer a los clientes de los mecanismos de comunicación y fundamentalmente, solventar las principales limitaciones que presenta la solución actual del Middleware del SCADA. Finalmente, la investigación realizada permitió arribar a las siguientes conclusiones:

- Las interfaces de comunicación distribuida garantizan la abstracción del transporte y mantienen la concurrencia en los mensajes.

- La definición de los datos del SCADA en el ámbito del Middleware, elimina el proceso de conversión a cada lado de la comunicación, disminuyendo significativamente el número de peticiones, que sin duda, unido a la baja latencia de la tecnología ICE, posibilitó mejorar el tiempo de transferencia de los datos.

- La propuesta disminuye hasta diez veces el tiempo de transferencia de los datos entre los componentes distribuidos de SCADA.

- La solución permite además robustez y fiabilidad, con características como la persistencia de las conexiones y redundancia de los servicios con las que no cuenta la solución actual.

Recomendaciones

1. Incorporar la propuesta de solución como el Middleware en las soluciones SCADAs desarrolladas por el CEDIN.

2. Implementar en la primera capa del software un sistema de dato único, con el fin de obtener un metadato genérico que permita escalar el sistema a otras áreas de aplicación, que exigen definiciones de datos diferentes a las utilizadas en un SCADA.

Referencias Bibliográficas

Akgul, F. (2013). ZeroMQ. *Use ZeroMQ and learn how to apply different message patterns* (pp. 7-21, 82-83). 35 Livery Street Birmingham B3 2PB, UK.: Packt Publishing Ltd.

Apache Thrift Wiki. Retrieved December, 2013, from http://wiki.apache.org/thrift/FrontPage

Bakken, D. E. (2003). MIDDLEWARE. http://www.eecs.wsu.edu/~bakken/middleware.htm

Bashir, M. S., & Qureshi, M. R. J. (2012). Hybrid Software Development Approach For Small To Medium Scale Projects: Rup, Xp & Scrum. *Cell, 966*, 536474921.

Briceño, J. (2007). Transmisión de datos. *Editorial: ULA Facultad de Ingeniería Publicaciones*.

Buffers, P. (2013). Google's data interchange format http://code.google.com/p/protobuf/

Bunch, C., Chohan, N., Krintz, C., Chohan, J., Kupferman, J., Lakhina, P., . . . Nomura, Y. (2010). *An evaluation of distributed datastores using the AppScale cloud platform.* Paper presented at the Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on.

Busch, D. (2010). Comparing OpenDDS and ZeroMQ Usage and Performance. *Middleware News Brief*.

Busch, D. (2013). Introduction to OpenDDS. *OpenDDS Articles, 2014*.

Calkins, C. (2009). Multi-Language CORBA Development with C++ (TAO), Java (JacORB), Perl (opalORB), and C# (IIOP.NET). *Middleware News Brief*(04).

Chemnitz, M., Kruger, J., Patzlaff, M., & Tuguldur, E. O. (2010, 13-16 Sept. 2010). *SOPRO - Advancements in the self-organising production.* Paper presented at the Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on.

Coulouris, G. F. (2009). *Distributed Systems: Concepts and Design* (4e ed.): Pearson Education.

Czerny, P. (2000). *Current uses of middleware in embedded/real-time applications.* Paper presented at the First OMG workshop on real-time and embedded distributed object computing.

Dayal, A., Tbaileh, A., Yi, D., & Shukla, S. (2015, 13-13 April 2015). *Distributed VSCADA: An integrated heterogeneous framework for power system utility security modeling and simulation.* Paper presented at the Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2015 Workshop on.

Dr.S.Bhargavi†, & Dr.B.R.Ramamurthy††. (2011). Remote Sensing and Control of Various Electrical, Atmospheric Parameters in large Scale Industries. *International Journal of Computer Trends and Technology, 1*(2), 22-29.

Dworak, A., Charrue, P., Ehm, F., Sliwinski, W., & Sobczak, M. (2011). *Middleware trends and market leaders 2011*.

Eusgeld, I., Nan, C., & Dietz, S. (2011). "System-of-systems" approach for interdependent critical infrastructures. *Reliability Engineering & System Safety, 96*(6), 679-686. doi: http://dx.doi.org/10.1016/j.ress.2010.12.010

Felden, T. (2014). Efficient and Change-Tolerant Serialization for Program Analysis Tool-Chains.

Gaitan, N. C. G., V.G. (2014). Transparent interaction of SCADA systems developed over different technologies. *2014 18th International Conference on System Theory, Control and Computing (ICSTCC)*, 476 - 481 doi: 10.1109/ICSTCC.2014.6982462

Grigorik, I. (2011). Protocol Buffers, Avro, Thrift & MessagePack. http://www.igvita.com/2011/08/01/protocol-buffers-avro-thrift-messagepack/

Henning, M., & Spruiell, M. (2011). Choosing Middleware: Why Performance and Scalability do (and do not) Matter. Retrieved from http://doc.zeroc.com/pages/viewpage.action?pageId=2523226

Henning, M., & Spruiell, M. (2013). *Distributed Programming with Ice* Vol. 3.5.1. *Ice Manual* (pp. 2706). Retrieved from http://doc.zeroc.com/display/Ice/Ice+Manual

Iacob, M., Bejan, C. A., & Andreescu, G. D. (2010). Supervisory Control and Data Acquisition Laboratory. *Telfor Journal, 2*(1), 49-54.

iMatix. (2009). Introduction to OpenAMQ. *What is OpenAMQ?* , 2013, from http://www.openamq.org/doc:user-1-introduction

iMatix. (2013). Market Analysis. *Zeromq Home page.* from http://www.zeromq.org/whitepapers:market-analysis

Juric, M. B., Rozman, I., & Hericko, M. (2000). Performance comparison of CORBA and RMI. *Information and Software Technology, 42*, 915 - 933.

Kago, M., & Yamashita, A. (2013). Development of a Scalable and Flexible Data Logging System Using NoSQL Databases. *TUPPC012, proceedings of ICALEPCS*.

Kleppmann's, M. (2013). Schema evolution in Avro, Protocol Buffers and Thrift. http://martin.kleppmann.com/2012/12/05/schema-evolution-in-avro-protocol-buffers-thrift.html

Krizsán, Z. (2010). *ICE extension of RT-Middleware framework.* Paper presented at the Proceedings of the 10th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics.

Kumar, C., & Paulus, R. (2014). A prospective towards M2M Communication. *Journal Convergence Information Technology (JCIT), 9*(2).

Maeda, K. (2012). Comparative Survey of Object Serialization Techniques and the Programming Supports. *Journal of Communication and Computer, 9*(8), 920-928.

Martinez, M. (2010). Interpreting OpenDDS Performance Testing Results. *Middleware News Brief*.

The MessagePack Project. Retrieved January, 2014, from http://msgpack.org/

Modesti, M. R. (2008). Sistemas de supervisión y monitoreo. *Introducción a los sistemas SCADA*.

Mohamed Najeh, L., & Mohamed Kamel, J. (2012). Project Management Phases of a SCADA System for Automation of Electrical Distribution Networks. *International Journal of Computer Science Issues, 9*(2), 157-162.

Nunn, R. (2002). Distributed software architectures using middleware. *3C05 Coursework, 2*, 1.

Ohta, K. (2011). Overview - MessagePack - Confluence. http://wiki.msgpack.org/display/MSGPACK/Overview

OMG. (2007). Data Distribution Service for Real-time Systems Version 1.2 *OMG Available Specification* (pp. 260).

OMG. (2011). Data Distribution Service (DDS). *Middleware Brief*.

Onderka, Z. (2011). *The Efficiency Analysis of the Object Oriented Realization of the Client-Server Systems Based on the CORBA Standard.* Paper presented at the Schedae Informaticae, Kraków.

Open SCADA Home project. (2015). *openSCADA is the companion project to Eclipse SCADA.* Retrieved February 10, 2015, from http://openscada.org/

Ortega, E. (2012). BREVE EXPLICACIÓN DE UN MIDDLEWARE.

PDVSA. (2009). Petróleos de Venezuela. *Sitio oficial Ministerio del Poder Popular de Petróleo y Minería*.

Penin, A. R. (2012). *Sistemas SCADA* (G. E. Alfaomega Ed. 3 ed.): Marcombo, Ediciones Técnicas.

Pinus, H. (2004). Middleware: Past and Present a Comparison.

Proview Home project. (2014). *Proview - Open Source Process Control* Retrieved January 22, 2014, from http://www.proview.se/v3/

Qiang, Y., Barria, J. A., & Green, T. C. (2011). Communication Infrastructures for Distributed Control of Power Distribution Networks. *IEEE Transactions on Industrial Informatics, 7*(2), 316-327. doi: 10.1109/tii.2011.2123903

Ruiz Tenorio, R. (2010). *Las pruebas de software y su importancia en las organizaciones.* Universidad Veracruzana.

SALAH, Z., & LAKHOUA, M. N. (2012). STUDY OF THE PROJECT MANAGEMENT PHASES AND THE ARCHITECTURE OF A SCADA SYSTEM. *JEE8 SCADA projects*.

Savochenko, R. (2015). oSCADA Home project. *Open SCADA. Open supervisory control and data acquisition project.* Retrieved January 20, 2015, from http://oscada.org/main/documentation/

Schmidt, D. C., Gokhale, A., Harrison, T. H., Levine, D., & Cleeland, C. (2011). *TAO: a High-performance Endsystem Architecture for Real-time CORBA*. Department of Computer Science,Washington University. St. Louis, MO 63130, USA.

Stopper, M., & Katalinic, B. (2009, March 18 - 20). *Service-oriented Architecture Design Aspects of OPC UA for Industrial Applications.* Paper presented at the International MultiConference of Engineers and Computer Scientists (IMECS 2009), Hong Kong.

Terada, H., Onishi, T., & Tsuchiya, T. (2012, 10-14 Sept. 2012). *Proposal of environmental adaptation for the next-generation distribution SCADA system.* Paper presented at the Electricity Distribution (CICED), 2012 China International Conference on.

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., . . . Murthy, R. (2009). Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment, 2*(2), 1626-1629.

Vázquez, M. H. (2008). Introducción a la arquitectura del Guardián del ALBA.

Viltre, J. M. B., Lalcebo, Y. d. R., González, Y. M., & Cáceres, J. A. A. (2014). *TECNOLOGÍAS MIDDLEWARE PARA LA CAPA DE COMUNICACIONES DE SAINUX*. Paper presented at the 17 Convención Científica de Ingeniería y Arquitectura, CUJAE, Cuba.

Völter, M., Kircher, M., & Zdun, U. (2005). *Remoting Patterns. Foundations of Enterprise, Internet and Realtime Distributed Object Middleware.* (1 ed.): John Wiley & Sons.

Y. Le Goc*, F. Cecillon, C. C., A. Elaazzouzi, J. Locatelli, P. Mutti, H. Ortiz, & Ratel, J. (2013). *PROTOTYPE OF A SIMPLE ZEROMQ-BASED RPC IN REPLACEMENT OF CORBA IN NOMAD.* Paper presented at the Proceedings of ICALEPCS2013, San Francisco, CA, USA.

Yamashita, A., & Kago, M. (2013). *A NEW MESSAGE-BASED DATA ACQUISITION SYSTEM FOR ACCELERATOR CONTROL.* Paper presented at the CALEPCS2013, San Francisco, CA, USA.

Yokogawa. (2012). FAST/TOOLS *General Specifications*.

Zapata, D. A. (2013). *DESARROLLO DE UN SISTEMA SCADA PARA USO EN PEQUEÑAS Y MEDIANAS EMPRESAS.* Universidad de Piura.

ZeroC, I. (2013). Our Customers. *ZeroC Home page.* from http://www.zeroc.com/customers.html

Anexos

Anexo 1. Pruebas de rendimiento realizadas a la tecnología TAO durante el pilotaje del sistema SCADA Guardián del ALBA.

Tabla 8. Pruebas de rendimiento a la tecnología TAO

Puntos (Variables)	Tiempos (ms)		
	Mínimo	Máximo	Media
1000	1272	1505	1370
5000	**6202**	**6716**	**6399.6**
10000	12137	12600	12276
15000	17139	18733	17898.8
20000	21715	24794	23781.2
25000	25300	30353	28207.4
30000	31055	45192	37084.6
35000	49954	54227	52492.6
40000	49725	65866	59555.8
45000	74728	76360	75430.6
50000	82285	85380	84295.6

Anexo 2. Rendimiento de las tecnologías TAO y ZeroMQ.

Tabla 9. Serialización-deserialización con Protocol Buffers y MessagePack

Cantidad Alarmas		1	10	100	1000	10000	100000	1000000
Protobuf	Tamaño en bytes	416	3776	37344	372128	3745568	37324320	372194336
	Tiempo en msec	0.04	0.13	1.3	2.04	22.17	205.91	6811 (mitad)
Msgpack	Tamaño en bytes	102	1011	10103	101003	1010003	10100005	101000005
	Tiempo en msec	0	0	0	1.47	16.04	159.82	6551

Tabla 10. Rendimiento de TAO (1 servidor y 1 cliente remoto)

Cantidad de estructuras (alarma)	Serialización (ms)	Deserialización (ms)	Respuesta (ms)	Mb/Seg
1: 102 bytes-0.000097Mb	0.04	0.04	1.43	0.07
10: 1011 bytes-0.00096 Mb	0.04	0	1.78	0.56
100: 10103 bytes-0.0096 Mb	0.17	0.34	3.17	3.75
1000: 101003 bytes-0.096 Mb	1.56	3.34	14.47	10.07
10000: 1010003 bytes-0.96 Mb	16.47	23.39	125.34	11.23
100000: 10100005 bytes-9.63 Mb	158.6	180.39	1195.82	11.23
1000000: 101000005 bytes-96.3 Mb	1775.78	1650.73	12075.34	11.13

Tabla 11. Rendimiento de TAO (1servidor y varios clientes remotos)

Tamaño de la petición: 10000 alarmas	5 clientes	10 clientes	30 clientes
Consumo Memoria	3.6 Mb	5.4Mb	5.6 Mb
Uso del CPU	0.00%	4.00%	16.00%
Tiempo respuesta (mseg)	119.9	122.32	1019

Tabla 12. Rendimiento de TAO (1 publicador y varios suscriptores)

Evento: 1 alarma Envío 10000 alarmas	5 suscriptores	10 suscriptores	15 suscriptores
Latencia (mseg)	0.1	0.21	0.31
Rendimiento (eventos/seg)	10272.2	4682.56	3277

Tabla 13. Rendimiento de ZeroMQ (1 servidor y 1 cliente remoto)

Cantidad de estructuras (alarma)	Serialización (ms)	Deserialización (ms)	Respuesta (ms)	Mb/Seg
1: 102 bytes-0.000097Mb	0.04	0.04	1.13	0.09
10: 1011 bytes-0.00096 Mb	0.04	0	1.3	0.81
100: 10103 bytes-0.0096 Mb	0.04	0.08	2.08	4.8
1000: 101003 bytes-0.096 Mb	0.39	1.78	11.78	9.6
10000: 1010003 bytes-0.96 Mb	3.17	17	105	10.9
100000: 10100005 bytes-9.63 Mb	21.43	169.34	1036.95	11.06
1000000: 101000005 bytes-96.3 Mb	516.73	1654.26	10366.95	11.05

Tabla 14. Rendimiento de ZeroMQ (1 servidor y varios clientes remotos)

Tamaño de la petición: 10000 alarmas	5 clientes	10 clientes	30 clientes
Consumo Memoria	3.5 Mb	3.5Mb	19 Mb
Uso del CPU	0.00%	0.00%	0.00%
Tiempo respuesta (mseg)	108.8	117.68	1511.7

Tabla 15. Rendimiento de ZeroMQ (1 publicador y varios suscriptores)

Evento: 1 alarma Envío 10000 alarmas	5 suscriptores	10 suscriptores	15 suscriptores
Latencia (mseg)	0.08	0.12	0.13
Rendimiento (eventos/seg)	13518.8	10925.44	7916.72

Anexo 3. Rendimiento de la tecnología ICE.

Tabla 16. Rendimiento de ICE (1 publicador y 5 suscriptores remotos)

Cantidad de estructuras (alarma)	Mínimo (ms)	Máximo (ms)	Promedio (ms)	Mb/Seg
1: 102 bytes-0.000097Mb	0	0	0	0.097
10: 1011 bytes-0.00096 Mb	0	0	0	0.96
100: 10103 bytes-0.0096 Mb	0	0	0	9.6
1000: 101003 bytes-0.096 Mb	0	2	1.17	77.76397516
10000: 1010003 bytes-0.96 Mb	13.2	21.6	17.37	56.30047632
100000: 10100005 bytes-9.63 Mb	130	150.6	138.95	69.41530126

Tabla 17.Rendimiento de ICE (1 publicador y 10 suscriptores)

Cantidad de estructuras (alarma)	Mínimo (ms)	Máximo (ms)	Promedio (ms)	Mb/Seg
1: 102 bytes-0.000097Mb	0	0	0	0.097
10: 1011 bytes-0.00096 Mb	0	0	0	0.96
100: 10103 bytes-0.0096 Mb	0	0	0	9.6
1000: 101003 bytes-0.096 Mb	1	2	1.09	91.82608696
10000: 1010003 bytes-0.96 Mb	13	20	16.3	60.18988845
100000: 10100005 bytes-9.63 Mb	115	168	136.7	71.03708851

Tabla 18. Consumo de Recursos de ICE

Tamaño de la petición	5 suscriptores				10 suscriptores			
	CPU (%)	RAM (MB)			CPU (%)	RAM (MB)		
		Icebox	Pub.	Susc.		Icebox	Pub.	Susc.
1	0%	0.6	0.2	0.3	0%	0.6	0.2	0.3
10	0%	0.8	0.3	0.4	0%	0.9	0.3	0.4
100	0%	1.5	0.4	0.6	0%	1.6	0.3	0.6
1000	0%	1.6	0.7	0.8	0%	1.7	0.7	0.8
10000	0%	39	2.5	2.8	0%	40	2.9	4.7
100000	1%	72	24	27	1%	114.32	25	37

Tabla 19. Cantidad de eventos/segundos y latencia de la comunicación con ICE

Evento: 1 alarma Envío 100000 alarmas	5 suscriptores	10 suscriptores
Latencia (mseg)	1.38957E-06	1.36696E-06
Rendimiento (eventos/seg)	720823.4814	737664.4705

Anexo 4. Resultados asociados a la validación de la propuesta de solución.

Tabla 20. Propuesta de Solución - Rendimiento obtenido para una muestra de 54400 variables

Propuesta de Solución										
Publicacion y Suscripción a 54400 variables Tiempo (ms)								Tamaño (bytes): 217600 - 0.207536 Mb Rendimiento		
No	Pub.	S1	S2	S3	S4	S5	Tiempos de Respuesta Media	MB/Seg	Eventos/Seg.	Latencia
1	13	63	41	66	79	38	70.4	2.947954545	772727.2727	1.29412E-06
2	9	47	61	35	64	61	62.6	3.315271565	869009.5847	1.15074E-06
3	10	64	64	62	57	46	68.6	3.025306122	793002.9155	1.26103E-06
4	8	54	57	48	63	66	65.6	3.163658537	829268.2927	1.20588E-06
5	8	53	80	54	42	56	65	3.192861538	836923.0769	1.19485E-06
6	14	61	56	78	66	74	81	2.56217284	671604.9383	1.48897E-06
7	8	59	48	62	39	48	59.2	3.505675676	918918.9189	1.08824E-06
8	8	78	41	45	62	49	63	3.294222222	863492.0635	1.15809E-06
9	11	34	68	56	56	65	66.8	3.106826347	814371.2575	1.22794E-06
10	11	66	38	50	57	44	62	3.347354839	877419.3548	1.13971E-06
Media	10	57.9	55.4	55.6	58.5	54.7	66.42	3.146130423	824673.7675	1.22096E-06
Mínimo	8	34	38	35	39	38	42	2.56217284	671604.9383	1.08824E-06
Máximo	14	78	80	78	79	74	94	3.505675676	918918.9189	1.48897E-06

Tabla 21. Solución Actual - Rendimiento obtenido para una muestra de 54400 variables

Solución Actual										
Publicacion y Suscripción a 54400 variables Tiempo (ms)								Tamaño (bytes): 217600 - 0.207536 Mb Rendimiento		
No	Pub.	S1	S2	S3	S4	S5	Tiempos de Respuesta Media	MB/Seg	Eventos/Seg.	Latencia
1	18	659	632	644	620	666	**662.2**	0.313403805	82150.40773	1.21728E-05
2	20	677	657	721	641	695	**698.2**	0.297244343	77914.63764	1.28346E-05
3	20	665	688	551	670	740	**682.8**	0.303948448	79671.93907	1.25515E-05
4	18	582	615	617	650	642	**639.2**	0.324680851	85106.38298	0.00001175
5	20	688	599	648	665	611	**662.2**	0.313403805	82150.40773	1.21728E-05
6	20	604	675	668	647	649	**668.6**	0.310403829	81364.04427	1.22904E-05
7	18	625	707	608	664	613	**661.4**	0.313782885	82249.77321	1.21581E-05
8	20	619	639	619	664	623	**652.8**	0.317916667	83333.33333	0.000012
9	18	706	676	607	638	689	**681.2**	0.304662361	79859.07223	1.25221E-05
10	20	699	634	611	577	685	**661.2**	0.313877798	82274.65215	1.21544E-05
Media	19.2	652.4	652.2	629.4	643.6	661.3	666.98	0.311332479	81607.46503	1.22607E-05
Mínimo	18	582	599	551	577	611	569	0.297244343	77914.63764	0.00001175
Máximo	20	706	707	721	670	740	760	0.324680851	85106.38298	1.28346E-05