Universidad de las Ciencias Informáticas
Facultad 5

Título: Subsistema de Comunicaciones para el SCADA Guardián del ALBA

Memoria de Proyecto para optar por el título de

Ingeniero en Ciencias Informáticas

Autor(es): Ariel Chávez Lorenzo

Tutor(es): Ing. Maikel Pérez Javier

Co-tutor: Ing. Amado Espinosa Hidalgo

Ciudad de la Habana, abril de 2009.

DATOS DE CONTACTO

Ing. Maikel Pérez Javier
Es Ingeniero Informático, profesor con categoría de Instructor, 2 años de experiencia en la docencia formando parte del Departamento de Ingeniería y Gestión de Software de la Facultad 5, de la Universidad de las Ciencias Informáticas. Amplios resultados en la producción como parte de la participación en el proyecto SCADA durante 2 años y lo que va del curso 2008-2009.
Correo electrónico: mperezj@uci.cu

Ing. Amado Espinosa Hidalgo
Es Ingeniero Informático y profesor asistente del Departamento de Ingeniería y Gestión de Software de la Facultad 5. Posee 6 años de experiencia en la actividad docente y productiva.
Correo electrónico: aespinosa@uci.cu

AGRADECIMIENTOS

A mi mamá, por su amor, firmeza, integridad y apoyo incondicional, que me ha permitido alcanzar mis metas. Le agradezco a mi papá por su ejemplo implícito a seguir en todo momento, que me convirtió en mejor persona.

A mi hermana, la doctora, por su apoyo, cariño y complicidad de tantos años. A mi abuela Rosa por sus atenciones y amor a lo largo de toda mi vida. A mis abuelos Jesús y Caridad, por su cariño, ejemplo y sencillez, a mis tíos Víctor y Osvaldo, a Mary, a toda mi familia.

Agradezco al proyecto Emedia la vinculación de la docencia con la producción; la oportunidad de contar con el ejemplo, las enseñanzas y la amistad de Fung, René y Sancho.

A mis compañeros del SCADA, en especial a mi tutor, Maikel Pérez, por sus consejos oportunos y por su amistad.

Le agradezco a Moisés y Amado, por sus revisiones, consejos técnicos y metodológicos en la realización de la tesis y la mía como profesional, a Maikel Arcia, por sus enseñanzas, visión y motivación.

A mis amigas de ayer y de hoy, a Irina, Iliana, Anita, Dunia. A Rita, por tanto amor y apoyo. A mis amigos, a Eduardo, Manuel, Luis Enrique, Antonio y José Antonio, que son mis hermanos entrañables de siempre y que me apoyaron en todo momento.

DEDICATORIA

A mi mamá,

a mi familia y amigos.

SÍNTESIS

La realización de un sistema de control, supervisión y adquisición de datos, SCADA en inglés, sobre software libre es una realidad hoy en día. Uno de sus componentes claves para brindar flexibilidad en la distribución geográfica del mismo, así como en el ámbito de aplicabilidad es el **subsistema de comunicaciones**, llamado usualmente *middleware*, que interconecta todos los módulos.

El presente trabajo expone el desarrollo de un componente *middleware* para el SCADA Guardián del ALBA, con el objetivo de satisfacer las necesidades de comunicación entre los diferentes módulos del sistema. Estas necesidades son catalogadas en dos áreas de servicios: sincrónicos, a los que se asocian los de Seguridad, Configuración e Históricos, y asíncronos, orientados a la distribución de datos, tales como: puntos, eventos, alarmas y comandos.

A partir de la realización de tareas concretas, se realiza un análisis crítico de la solución y se exponen las ideas que enriquecieron el trabajo, resaltando los aportes individuales a la misma. Una panorámica de los contenidos abordados y los principales resultados obtenidos es brindada al lector, además de realizar una propuesta que complemente la solución actual, lo cual permite consolidar un producto para la gestión de la comunicación a alto nivel con gran calidad y eficiencia.

PALABRAS CLAVE

Comunicación distribuida, *Middleware*, SCADA, TAO, Software Libre, Guardián del ALBA

ÍNDICE

INTRODUCCIÓN

La Universidad de las Ciencias Informáticas (UCI) es una universidad productiva, cuya misión es *formar profesionales, comprometidos con su Patria, calificados en la rama de la Informática, a partir de un modelo pedagógico flexible, que vincula dinámica y coherentemente el estudio con la producción y la investigación, acorde con las necesidades sociales del país y de otros pueblos hermanos*. (Universidad de la Ciencias Informáticas, 2007)

La Producción de Software y Servicios Informáticos se basa en la integración de los procesos de formación, investigación y producción en torno a una temática para convertirla en una rama productiva, siendo los polos el espacio natural para ejecutar proyectos temáticos. Dentro de estos centros se encuentra el de *Hardware* y Automática, perteneciente a la Facultad 5; uno de los proyectos que se desarrollan actualmente es el "SCADA Guardián del ALBA".

La UCI en conjunto con la Gerencia AIT de la empresa Petróleos de Venezuela S.A. (PDVSA), desarrollan desde el año 2006 un proyecto de software con el objetivo de crear un producto SCADA, conocido como SCADA Nacional o SCADA PDVSA en sus inicios, entre 2006 y 2007, y SCADA Guardián del ALBA luego de ser presentado en la "Cumbre del ALBA" en enero de 2008, proyectando una futura instalación del sistema en los países integrantes de esta organización. El software de la solución se realiza de manera cooperada entre equipos de la UCI, DST-AIT PDVSA, CEDAI, UCLV, ISMMM, ULA, DBAccess, IntelCom, Ingesi e Isca.

Por la magnitud del proyecto este fue dividido en varios módulos de desarrollo, entre los cuales se encuentra el Subsistema de Comunicaciones (*Middleware*), con la responsabilidad de satisfacer los requerimientos de comunicación inter-procesos de los restantes componentes del SCADA. Se estructura a través de un conjunto global de servicios, que a su vez fue dividido en dos partes fundamentales (Pérez Javier, 2007a):

1. Gestor de comunicación sincrónico para el acceso a Históricos, Configuración y Seguridad.
2. Gestor de comunicación asíncrono para el envío de datos, tales como puntos, comandos, alarmas, y eventos, entre otros.

Los estudios para el desarrollo del módulo *Middleware* del SCADA comenzaron a mediados del año 2006, etapa fundamentalmente de conceptualización del sistema SCADA como un todo y que posteriormente devendría en el sistema concreto, obteniéndose el primer prototipo funcional a principios de 2007.
A continuación se presenta el siguiente trabajo, ya concluidos e incorporados ambos desarrollos, con un grado de madurez aceptable para los requerimientos y versión actuales del producto.

Análisis Científico Metodológico

En el SCADA Guardián del ALBA no existía un módulo gestor de las comunicaciones entre los diferentes componentes de medio y alto nivel de la pirámide de control, que muestra las diferentes jerarquías de control desde los niveles de sensores y actuadores hasta planta y gestión. Los diferentes módulos requieren solicitar credenciales de seguridad, configuración, puntos, alarmas, comandos, eventos, entre otros elementos de información, sin los cuales no pueden garantizar su funcionamiento de manera cooperada para controlar y supervisar un proceso determinado.

Para dar respuesta a la situación problemática planteada anteriormente se define como el **problema científico**: ¿Cómo gestionar el intercambio de información entre los diferentes subsistemas del SCADA Guardián del ALBA en un ambiente distribuido?

Nos formulamos como **objetivo general**: desarrollar un conjunto de servicios de comunicación que brinde mecanismos sincrónicos y asíncronos de envío y recepción de información en el sistema SCADA Guardián del ALBA.

A partir del objetivo general se definieron las siguientes **tareas específicas**:

- Probar el sistema GNU/Linux para comprobar cuán adecuado es para ser usado como base del SCADA Guardián del ALBA.
- Probar tecnologías de comunicación para identificar la más adecuada.
- Diseñar componente sincrónico.
- Diseñar componente asíncrono.
- Implementar componente sincrónico.
- Implementar componente asíncrono.

- Integrar y probar el subsistema de comunicaciones como un todo con vistas a valorar el grado de aceptación y acople con el resto del sistema.

Impacto de la solución

La abstracción de la capa de comunicaciones en un conjunto de servicios comunes es una técnica empleada comúnmente en los sistemas distribuidos. Entre los beneficios que provee se encuentran el encapsulamiento de topologías de red, desacople entre la ubicación física de los datos y los nodos que los requieren, confiabilidad, fiabilidad y autenticidad. El **Sistema de comunicaciones** del SCADA está diseñado teniendo la eficiencia como un eslabón fundamental, con el propósito de satisfacer las restricciones de tiempo real en cualquiera de los escenarios de configuración del sistema, siempre y cuando este cumpla con los requerimientos mínimos establecidos en la documentación para una operatividad adecuada. Estos servicios de comunicación de alto nivel basados en **ACE+TAO**, que ofrecen envíos de datos a altas velocidades, constituyen una alternativa software libre a tecnologías privativas como **DCOM** de Windows, y a otras menos eficientes como **SOAP**, que proveen funcionalidades similares basados en objetos remotos, pero que no están disponibles para otras plataformas, GNU/Linux entre ellas, o que no cumplen con las restricciones de tiempo real establecidas en el entorno industrial.

Como bibliografía fueron consultados artículos disponibles en Internet mayormente, la base bibliográfica del proyecto, *TAO Developers Guide*, así como documentos generados por asesores y jefes de línea. El trabajo fue realizado usando completamente software libre y con un grado medio de portabilidad a otros entornos y plataformas, tales como Windows y Mac OS/X.

Para la elaboración de esta primera etapa se emplearon los medios disponibles en la UCI y en el Distrito AIT-PDVSA de Mérida para el desarrollo del proyecto. Este presupuesto incluye además el costo de desarrollo del resto de los subsistemas que están siendo desarrollados por equipos en paralelo.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En los últimos años la informática ha alcanzado un auge tal incomparable al de ningún otro campo, se han acortado distancias, eliminado barreras. Sin embargo, las capacidades de cómputo existentes siguen siendo limitadas para ciertos tipos de problemas, en los cuales un solo computador es insuficiente, que involucran la necesidad de compartir recursos. Con esta meta surgen los sistemas distribuidos.

En el presente capítulo se analizarán los conceptos de sistema distribuido y *middleware,* como base teórica para obtener una mejor comprensión del entorno en el cual se desarrollan las actividades; así como las tecnologías de comunicación de mayor auge en el momento de inicio de la investigación.

1.1 Sistemas Distribuidos

En general, los sistemas distribuidos surgen por la necesidad de compartir recursos, tanto de *software* como *hardware*. Entre las definiciones planteadas por varios autores están:

- "*Un sistema en el que los componentes de hardware o software ubicados en computadores en red, comunican y coordinan sus acciones únicamente mediante el paso de mensajes.*" (Coulouris, y otros, 2001)
- "*Colección de ordenadores autónomos enlazados por una red y soportados por aplicaciones que hacen que la colección actúe como un servicio integrado*". (Universidad Politécnica de Cataluña 2005)

En resumen, los sistemas distribuidos pueden definirse como aplicaciones con una fuerte componente geográfica, donde la ejecución no ocurre en una única estación de trabajo. Entre sus componentes se encuentra el encargado de gestionar la comunicación entre los diferentes nodos con vistas a garantizar el paso de mensajes y que el sistema responda como un servicio integrado único.

Coulouris en 2001 identifica varios retos asociados a los sistemas distribuidos que deben tomarse en consideración a la hora de desarrollar sistemas más ambiciosos y robustos

1. Heterogeneidad: Se refiere a la variedad existente entre los elementos que componen la red y que participan en el sistema. Esta variedad aplica a redes, *hardware* de computadoras,

sistemas operativos, lenguajes de programación e implementaciones por desarrolladores diferentes. Puede que a pequeña escala las redes sean homogéneas, pero mientras mayor es la red, mayor heterogénea se convierte.

2. Extensibilidad: La extensibilidad se refiere al grado con que el sistema puede ser extendido y re-implementado en diferentes formas. En el caso de los sistemas distribuidos se refiere al grado con que nuevos servicios de acceso a recursos pueden ser adicionados. *Sistema abierto, estándares públicos.* (Universidad Politécnica de Cataluña, 2005)

3. Seguridad: Muchos de los recursos de información que se hacen disponibles y mantenidos en los sistemas distribuidos tienen un alto valor para sus usuarios, por lo que la seguridad es de considerable importancia. La seguridad tiene tres componentes: la confidencialidad se refiere a la protección ante la apertura a individuos no autorizados; la integridad, que se refiere a la protección contra alteración o corrupción de la información y la disponibilidad, encargada de la protección contra la interferencia en los medios de acceder a los recursos. En este último caso se encuentran los famosos ataques de negación de servicio, que consisten en bombardear el servicio con un gran número de pedidos sin sentido, de esta manera los usuarios verdaderamente interesados en los recursos compartidos no pueden usarlos.

4. Escalabilidad: La escalabilidad se refiere al grado con que el sistema distribuido puede operar a diferentes escalas, desde una pequeña intranet hasta Internet. Un sistema se dice escalable si permanece efectivo cuando aumenta significativamente el número de recursos y de usuarios. En este sentido tiene vital importancia controlar el costo de los recursos físicos y determinar sus límites, la pérdida de rendimiento y la prevención del agotamiento de los recursos. En general, es una buena práctica descentralizar los algoritmos en vistas de evitar los conocidos **cuellos de botella**.

5. Manejo de fallas: Cualquier proceso, computadora o red puede fallar independientemente de los demás. Por este motivo cada componente debe estar al tanto de las posibles razones por las cuales los componentes de los que él depende pueden fallar, y ser diseñado para lidiar con estos fallos apropiadamente.

6. Concurrencia: La presencia de múltiples usuarios en un sistema distribuido es la fuente de pedidos concurrentes a sus recursos. Cada recurso debe ser diseñado de tal manera que esté a salvo en un ambiente concurrente.

7. Transparencia: Tiene el propósito de hacer invisible para el programador ciertos aspectos de la distribución, de tal manera que solo sea necesario preocuparse del diseño de la aplicación particular. Entre estos elementos se encuentran la ubicación física y los detalles de cómo sus operaciones son accedidas por otros componentes.

1.2 Subsistema de comunicaciones

Con el devenir de los sistemas distribuidos surge el subsistema de comunicaciones, usualmente llamado *middleware,* que se refiere a *la capa de software que provee una abstracción de programación a la vez que enmascara la heterogeneidad de las redes, hardware, sistemas operativos y lenguajes de programación* (Coulouris, y otros, 2001). Esta capa de *software* provee un modelo computacional uniforme para ser usado por los programadores de servidores y aplicaciones distribuidas.

Con el objetivo de resolver los problemas de heterogeneidad implícita en las redes de computadoras, un conjunto de técnicas y patrones de diseño se han extendido hasta constituir completas plataformas de trabajo y especificaciones. En la construcción del sistema de comunicaciones del sistema SCADA Guardián del ALBA, se analizaron las tecnologías de mayor impacto en ese entonces (año 2006), entre las que se encuentran diferentes implementaciones de CORBA (TAO, ORBit, MicoORB, entre otras), DCOM, RMI y SOAP. A continuación se veremos en mayor profundidad algunas de ellas.

CORBA

Se refiere a *Common Object Request Broker* por sus siglas en inglés, constituye una de las opciones tecnológicas más importantes a la hora del desarrollo de sistemas distribuidos. "*CORBA provee interfaces de programación y modelos independientes de la plataforma para aplicaciones distribuidas orientadas a objetos"* (Henning, y otros, 1999). CORBA es un estándar que establece una plataforma de desarrollo, especificando la invocación de objetos remotos, esto permite a un programa corriendo en una computadora invocar métodos de un objeto que se encuentra en otro programa y en otra computadora. El ORB, siglas de *Object Request Broker*, constituye el núcleo de la tecnología.

CORBA fue definido y está controlado por el Object Management Group (OMG), que define las diferentes API, el protocolo de comunicaciones y los mecanismos necesarios para permitir la interoperabilidad entre

diferentes aplicaciones escritas en diferentes lenguajes y ejecutadas en diferentes plataformas, tema fundamental en computación distribuida.
CORBA utiliza un lenguaje de definición de interfaces, IDL por sus siglas en inglés, para especificar las interfaces con los servicios que los objetos ofrecerán. CORBA puede especificar a partir de este IDL la interfaz a un lenguaje determinado, describiendo cómo los tipos de dato CORBA deben ser utilizados en las implementaciones del cliente y del servidor. Adicionalmente, CORBA define un conjunto de servicios genéricos que pueden ser usados en las aplicaciones distribuidas, tales como *Name Service*, *Event Service* y *Notification Service*, servicios de nombre, eventos y notificación respectivamente.

Implementaciones del estándar existen para Ada, C, C++, Smalltalk, Java y Python, así como para Perl y TCL. Al compilar una interfaz escrita usando el lenguaje IDL se genera código para el cliente y el servidor. El código generado para el cliente es conocido como *stub* (cabo), el cual incluye un *proxy* (representante) del objeto remoto en el lado del cliente, mientras que el código generado para el servidor consiste en *skeletons* (esqueletos) donde el desarrollador debe escribir el código del comportamiento asociado a los métodos del objeto. CORBA es más que una especificación multiplataforma y también define servicios habitualmente necesarios como seguridad, transacciones y concurrencia.

Entre las implementaciones existentes del estándar CORBA se encuentran:

1. **TAO**

   TAO ORB (por sus siglas en inglés ***T**he **A**CE **O**RB*) fue desarrollado por DOC, siglas de Distributed Object Computing Group, en la Universidad de Washington, bajo la dirección de Douglas Schmidt. La versión 1.0 de TAO es compatible con CORBA 2.2 e incluye algunos aspectos de CORBA 2.3. Inicialmente fue pensado para tiempo real, pero ha llegado a ser un ORB de propósito general muy capaz y adaptable a cualquier tipo de sistema distribuido sea de tiempo real o no. TAO es ampliamente utilizado por los desarrolladores ofreciendo capacidades insustituibles en los servicios de CORBA que implementa. Actualmente TAO es compatible con la versión 3.0 de CORBA. Es una plataforma robusta para la facilitación de las comunicaciones en sistemas distribuidos, ofrece capacidades muy superiores a otros *middleware* en varios sentidos. TAO tiene un mecanismo de prioridades basado en la especificación de RT-CORBA lo cual le permite su funcionamiento en sistemas de tiempo real con una eficiencia muy alta.
2. **MICO**

Su nombre está conformado por las siglas de la frase en inglés ***Mi**co Is **Co**rba*. Fue el resultado de la colaboración de cientos de programadores independientes, trabajando juntos para lograr una de las primeras implementaciones de la especificación de CORBA. Inicialmente adoptado por el proyecto GNOME para la comunicación en sus entornos gráficos. La intención de este proyecto fue brindar una implementación de CORBA con la mayor cantidad de servicios como fuera posible. MICO por ser una de las primeras implementaciones del estándar ganó gran popularidad y versatilidad como proyecto de código libre. Con el paso del tiempo se desarrollaron otros ORB, más funcionales y más capaces a la vez que salían a la luz ineficiencias en la ejecución de MICO, como la elevada sobrecarga y el alto consumo de memoria que necesitaba. Debido a ello, el principal cliente de MICO, el proyecto GNOME, comenzó a desarrollar su propio ORB, y de esta forma nació ORBit. Actualmente MICO continúa siendo activamente desarrollado. ObjectSecurity Ltd. ofrece soporte comercial y es uno de los que más activamente aporta al proyecto desde 2000. (MICO Project Team, 2006)

3. **ORBit2**

   "ORBit2 es un ORB compatible con la versión 2.4 de CORBA. Ofrece una API madura para los lenguajes C, C++ y Python, y con diferentes grados de completitud para Perl, Lisp, Pascal, Ruby y TCL.ORBit2 admite POA, DII, DSI, TypeCode, Any, IR e IIOP" (GNOME Foundation, 2002). ORBit2 fue elaborado a partir de ORBit por el proyecto GNOME, y tiene características únicas en comparación a ORB, tales como la gran velocidad en el intercambio de datos y el bajo consumo de memoria, diseñado principalmente para el entorno de escritorio. El núcleo está escrito en C y corre en Linux, Unix y Windows.

A pesar del gran auge con que cuenta CORBA, otros equipos han creado tecnologías diferentes con vistas a solucionar el mismo problema de comunicación, entre ellas:

1. **DCOM**

   DCOM (*Distributed Component Object Model)*, en español, Modelo de Objetos de Componentes Distribuidos) surgió a partir de COM, que a su vez estuvo condicionado por la tecnología OLE (*Object Linking and Embedding*) de Microsoft. OLE surgió en el año 1990 con el fin de hacer la funcionalidad de "cortar y pegar" en los entornos Windows. Después se extendió a OLE2, que cambió su nombre por COM (*Component Object Model*) y constituyó una infraestructura genérica para la comunicación entre componentes. Al extenderse esta infraestructura con la funcionalidad

de aplicarla a varias maquinas comunicadas en red, surgió DCOM. Esta es una tecnología que se ha ido desarrollando y actualmente forma parte esencial de ActiveX, *Microsoft Transaction Server* y COM+.

DCOM brinda componentes básicos necesarios para el diseño y el desarrollo de sistemas sofisticados, con escalabilidad y robustez, sin embargo presenta problemas de seguridad y Microsoft sugiere que se usen tecnologías más modernas como COM+ y .Net.

2. Java RMI

"El paquete RMI es un esquema centrado en Java para objetos distribuidos que ahora es parte de la API de Java". (Farley, 1998)

La invocación a método remoto de java, en inglés *Java Remote Method Invocation* o RMI, es el mecanismo ofrecido en Java que permite a un procedimiento poder ser invocado remotamente, es decir, este puede existir en cualquier espacio de direcciones de la red incluso en la misma computadora en la que se hace la invocación.

RMI es básicamente un mecanismo RPC (*Remote Procedure Call*) orientado a objetos. Java/RMI está basado en un protocolo llamado *Java Remote Method Protocol* (JRMP). Una de las características fundamentales de la arquitectura Java es *Java Object Serialization*, la cual permite que los objetos sean traducidos o transmitidos como un flujo de datos. Desde que *Java Object Serialization* es específico del entorno, tanto el objeto servidor como el objeto cliente tienen que ser escritos en Java. Cada objeto servidor define una interfaz, la cual puede ser usada para acceder a este objeto desde la misma máquina virtual o desde otra en cualquier lugar de la red. La interfaz brinda un grupo de métodos, los cuales indican los servicios ofrecidos por el objeto servidor. Para que un cliente pueda localizar un objeto servidor, RMI depende de un servicio nombrado *RMI-Registry* que corre en el servidor y da información acerca de los objetos que se encuentran en la misma. Cuando un cliente adquiere la referencia de un objeto, invoca sus métodos con total transparencia a su ubicación, su espacio de direcciones puede estar en cualquier lugar de la red. La referencia de objetos es brindada a los clientes en forma de direcciones URL y ellos acceden a la invocación del objeto servidor de la misma forma que si accedieran a una dirección Web.

1.3 Middleware en el sistema SCADA

En el sistema SCADA, el software de comunicación tiene la responsabilidad de garantizar la interacción con los servicios de Seguridad, Configuración e Históricos, así como la distribución de información asociada a puntos, alarmas, comandos, eventos, estadísticas de dispositivos y bitácora. Por la complejidad que el proceso a controlar puede tener, *Middleware* debe garantizar que la información esté en el momento necesario en los módulos, pues de lo contrario las consecuencias podrían ser fatales.

La función del subsistema de comunicaciones en los sistemas SCADA, desde el punto de vista de un sistema distribuido, es garantizar el intercambio de información de manera transparente sin que la ubicación geográfica sea un problema a tomar en consideración, encapsulando tecnologías, protocolos de red, entre otros.

CAPÍTULO 2: APORTE A LA SOLUCIÓN

El subsistema de comunicaciones, desarrollado para el sistema SCADA PDVSA, consiste en un conjunto de servicios a los que se accede a través de bibliotecas de funciones, las que se usan en los módulos del sistema, garantizando las funcionalidades de comunicación necesarias entre módulos.

Como integrante del equipo de desarrollo del SCADA Guardián del ALBA, desde mediados de 2006, el autor participó en diversas tareas que han aportado a varios entregables, pactados entre las partes cubana y venezolana. Entre los que se encuentran “Documento de pruebas del núcleo Linux” (González Vázquez, 2006), “Documento de diseño del *Middleware*” (Pérez Javier, 2007b), “Documento de implementación del *Middleware*” (Pérez Javier, 2007c), “Informe de prueba del *Middleware*” (Pérez Javier, 2008a), “Especificación de las nuevas funcionalidades a incorporar, plan de pruebas y resultados a obtener” (Pérez Javier, 2008b) y en la confección de la “Especificación de comandos del SCADA Guardián del ALBA” (Herrera, y otros, 2007)

A continuación detallaremos los principales retos enfrentados así como los resultados obtenidos en las diferentes esferas de trabajo.

2.1 Pruebas al sistema GNU/Linux

La tarea de realizar pruebas al sistema operativo, más específicamente al núcleo Linux, fue asignada a la línea Sabores de Linux, a cargo de Ms.C. Fidel González Vázquez, que funcionó en los primeros meses del proyecto y arrojó como resultados un conjunto de sugerencias, modificaciones y configuraciones al sistema.

Como integrante de esta línea al autor se le asignaron un conjunto de tareas, entre las que figuran:

- Compilación del núcleo de tiempo real RTAI
  RTAI, del inglés *RealTime Application Interface*, es un núcleo de tiempo real basado en Linux y muy similar a RTLinux. Esta tarea fue una de las de mayor envergadura, en adición a la complejidad de compilar un núcleo, tarea para la cual el equipo no se encontraba preparado, estaba el número de parches que se le debían aplicar al código fuente para compilarlo. El problema fundamental fue lograr una configuración del núcleo que reconociera todo el *hardware* de

la computadora y fuera capaz de arrancar. Tras una serie de investigaciones el equipo fue logró configurar e instalar el núcleo compilado, para posteriormente realizar las pruebas pertinentes.

- Pruebas de carga del sistema: *I/O* (entrada/salida), *CPU* (espacio usuario)
  En el sistema operativo podemos identificar dos espacios de ejecución: modo usuario y modo núcleo, siendo el segundo el que más privilegios otorga a los procesos. La *API* del núcleo es restringida por políticas de seguridad, siendo reservado mayormente para manejadores de hardware, rutinas de respuesta a interrupciones, núcleo del sistema operativo, entre otros; por esta razón fue necesario probar el comportamiento del sistema bajo condiciones anormales de carga. Al autor se le asignó la responsabilidad de preparar las pruebas de lectura / escritura intensa (*I/O*) y sobrecarga del *CPU* o procesamiento intenso, siendo desarrolladas de manera exitosa y arrojando resultados de peso en las valoraciones correspondientes al sistema GNU/Linux.
- Pruebas de latencia de interrupciones (núcleo), temporizadores (espacio usuario) y re-planificación
  Medir la latencia de interrupciones es de suma importancia en sistemas de tiempo real, por esta razón se asignó la responsabilidad de desarrollar esta herramienta de prueba. El problema fundamental es que esta prueba, a pesar de ser muy importante, no había sido realizada previamente en el intento de usar Linux como núcleo de aplicaciones de tiempo real.
  Luego de un proceso investigativo con el objetivo de entender el mecanismo de interrupciones de Linux, se pudo llegar a diseñar tres componentes: dos módulos del núcleo, uno basado en el código fuente del manejador del puerto paralelo ***parport*** para registrar el instante de propagación de la interrupciones y un generador de interrupciones por software; el tercero, en espacio usuario, tenía el objetivo de recolectar los datos almacenados en el núcleo. En relación con los temporizadores fue necesario investigar cómo usar las señales del sistema para implementar temporizadores. La tarea investigativa estuvo centrada en el uso de las 32 señales auxiliares para tiempo real que brinda el núcleo. Para la re-planificación se dio la tarea de medir la latencia de la re-planificación de un proceso, con el uso de *sleep*, esto fue implementado de manera exitosa y se registraron los resultados en el entregable correspondiente.

Estas tareas, a pesar del alto grado de complejidad, brindaron la oportunidad de adentrar en el funcionamiento del núcleo y el sistema operativo en general, así como en el uso de la API de desarrollo más primitiva que ofrece el sistema operativo, la mayoría en lenguaje C.

Un primer acercamiento al subsistema de comunicaciones del SCADA nos permite observar que este tiene dos propósitos fundamentales. El primero de ellos consiste en brindar interfaces a servicios globales, tales como Configuración, Seguridad e Históricos. El segundo proporciona un mecanismo de entrega de datos desacoplado, estos pueden ser alarmas, puntos, comandos, entre otros. Uno de los medios de enfrentar la complejidad del *middleware* fue separar en dos componentes estas responsabilidades.

2.2 Componente sincrónico

El componente sincrónico, como se ha dicho anteriormente, dota de interfaces a los servicios de Seguridad, Configuración e Históricos. Se dice que es sincrónico, pues el modelo de comunicación es cliente - servidor, es decir, todas las peticiones son realizadas de manera directa y bloqueante a los servicios, una vez invocado el método, este solo retorna cuando el servidor da respuesta.

- Seguridad y Configuración

  Durante este desarrollo surge el problema de interfaces inestables, cambiando varias veces cómo comunicar a los demás módulos con estos servicios. Fue necesario diseñar una solución que homogenizara el uso de la tecnología. El principal problema consistió en crear una meta-función, a través de la cual se pudiera traducir cada elemento de la interfaz, y de esta forma mantener la reusabilidad. La solución consistió en crear una función genérica siguiendo el siguiente principio: <Colección Retorno> función (<Colección Parámetros>), la complejidad adicional consistió en linealizar los parámetros originales, en caso de ser complejos, a elementos simples, tales como: enteros, cadenas y flotantes, por tal motivo, se restringe la extensibilidad al factor linealizable de los atributos y a que el subsistema de comunicaciones necesito conocer a priori las interfaces concretas de estos elementos parámetros – retornos. Esta solución se recopiló en la biblioteca *SingleObject,* y en ella se basaron *Security* y *Configuration.*

  En configuración hubo un problema adicional, pues la comunicación podía ser cliente – servidor (solicitud de configuración en frío o inicial) o desde el servidor hasta el cliente (configuración en caliente, no es necesario reiniciar el sistema para que este actualice un cambio de configuración), desde esta perspectiva, con vistas a no sobrecargar el cliente con una lógica eterna de encuesta por cambio de configuración, se decidió realizar un mecanismo bidireccional de comunicación, que pudiera ser iniciada tanto por el cliente como por el servidor. Fue asignada la tarea de implementar dicho mecanismo usando la tecnología TAO, mecanismo incorporado en las versiones actuales del

sistema desde agosto de 2007 para la modelación de la configuración, y permite que los clientes persistan incluso ante fallas del servidor.

- Históricos

  La necesidad de comunicar a la Base de Datos de Históricos con el resto de los módulos por medio del subsistema de comunicaciones constituye un requerimiento desde el surgimiento del SCADA[1], siendo los requerimientos muy similares a los expuestos para configuración, con las particularidades de su especificación. El objetivo fundamental era permitir la realización de consultas usando el lenguaje estándar SQL y la obtención de los resultados a dichas consultas, de manera desacoplada y sin espera ocupada durante se realizaba la consulta a la base de datos.

  Hubo de diseñar e implementar estas funcionalidades. Para resolver este problema se efectuaron reuniones con los asesores de los módulos involucrados (Históricos y Reportes fundamentalmente), se definieron las interfaces y diseñó el componente, este fue aprobado por los implicados y se pasó a la fase de implementación. Acorde a lo planificado se obtuvo la primera versión funcional en diciembre de 2007, que luego se iteró y transfirió al resto del equipo de desarrollo.

2.3 Componente asíncrono

El subsistema asíncrono tiene la meta objetiva de garantizar el envío masivo de información desde una o varias fuentes hacia uno o varios destinos, en dependencia del tipo de información. Se dice que es asíncrono pues el mecanismo de envíos además de ser publicador – consumidor, es no bloqueante, tanto para el que envía como para el que recibe.

El primer prototipo funcional realizado del subsistema de comunicaciones estuvo enfocado hacia estas funcionalidades, por pertenecer a los casos de uso priorizados que tenían una componente de comunicación y que involucraban a varios módulos. Fueron asignadas las siguiente tareas dentro de este contexto:

- Diseñar la arquitectura

---

1 La descripción de la arquitectura del SCADA Guardián del ALBA establece que ningún módulo debe comunicarse con otro por un medio de comunicación distinto de *Middleware*.

Fue dada la responsabilidad de crear un mecanismo que desacoplara el cliente final de la tecnología. El resultado fue un sistema compuesto por tres componentes bibliotecas de software con responsabilidades bien definidas que cooperaban para intercambiar información y exportaban una interfaz sencilla para ser usada por aplicaciones del SCADA. Dentro de los problemas en el diseño estuvo el uso de patrones con vistas a ocultar la tecnología, que demandó investigación en el tema. Otro inconveniente consistió en la división de la aplicación en componentes que a su vez fueran bibliotecas, ningún miembro del equipo conocía como crearlas, cargarlas, exportar correctamente los símbolos, resolver dependencias indefinidas; para lo que fue necesario investigar la abundante documentación existente en internet. Esta tarea contribuyó al entregable "Documento de Diseño del *Middleware*, 2007", además afirmó la confianza del equipo en la creación del subsistema.

- Garantizar envíos asíncronos

  El problema consistió en permitir el envío de información de manera asíncrona, la solución fue la incorporación del patrón estanque de hilos (*threads pool* en inglés) entre las capas medio y superior, de manera tal que el grueso de esfuerzo en el envío de información se realizaba como una tarea de fondo de manera no bloqueante para la aplicación que compartía el proceso con el *middleware*. Contribución al entregable "Documento de implementación del *Middleware*, 2007".

- Diseñar excepciones

  Al comenzar a tratar el comportamiento ante situaciones excepcionales, tales como pérdida de conexión de la red o caída de los servicios, surgió el problema del tratamiento de situaciones excepcionales que ocurrían en un contexto de ejecución en paralelo, esto difiere de los escenarios usuales de propagación de errores, pues al estar en contextos diferentes se comportan casi como procesos, la solución brindada fue mediante un diseño de llamadas asíncronas para la notificación de excepciones, intentando mantener en todo momento la estabilidad del sistema y contribuyendo en la definición de la jerarquía de excepciones del módulo. En la fecha actual parte de este mecanismo ha sido insertado en la solución actual, el resto espera por la próxima iteración de construcción para formar parte del *build* del sistema. Contribución al entregable "Documento de implementación del *Middleware*, 2007".

- Optimizar mecanismo de envío en la capa de la tecnología

  Un punto débil de subsistema de comunicaciones era la baja velocidad de transmisión respecto a implementaciones en otros SCADA, en parte debido al uso de tecnología *middleware* de propósito

general (TAO) para implementar los servicios requeridos. Se dio la tarea de analizar el mecanismo actual, con vistas a identificar mejoras. El autor identificó una oportunidad en el hecho de que *middleware* contaba con mecanismos de envío de lotes de puntos (técnicamente demostrado ser más rápido) sin embargo este no era usado por los clientes. Se dio a la tarea de convertir el envío de estos datos por lotes a la hora de usar la tecnología. El principal problema consiste en que el lote debe ser enviado cuando este se llena o cuando ha transcurrido un tiempo máximo entre la llegada del primer punto y la hora actual. Este problema fue resuelto usando el patrón *Observer* de manera satisfactoria. Contribución al entregable "Documento de implementación del *Middleware*, 2007".

- Optimizar estanque de hilos.

  El estanque de hilos, original del subsistema de comunicaciones tenía el problema de crear hilos exclusivos para un tipo de dato determinado, no permitiendo reutilizar este contexto de ejecución para el envío de otros datos cuando los hilos asignados a estos últimos estuvieran saturados. Para resolver este problema, se diseñó un estanque de hilos genéricos, con un número máximo de hilos controlado a nivel de aplicación, a diferencia del anterior que era a nivel de tipo de dato. Para esto fue necesario crear un desacople total entre el contexto de ejecución y el dato que se procesa, creando los conceptos de trabajo, trabajador y tarea, relacionados mediante el principio de que un trabajo consiste en un trabajador realizando una tarea. Con esta solución el subsistema funciona más rápido, consume menos recursos y aumenta la legibilidad del código. Con vistas a satisfacer dos reglas diferentes: alta velocidad de transmisión y mantención del orden de envío de comandos y respuestas, se implementaron dos modelos en el estanque de hilos. El estanque tradicional comparte una cola de trabajos entre los diferentes contextos, garantizando que cada los elementos se envíen en el orden de la cola, el cual puede estar sujeto a prioridades, el otro, maximiza la eficiencia minimizando el número de contextos que comparten recursos, consiste en mantener una cola de trabajos individual por hilo, y un planificador que decide hacia qué hilo enviar un trabajo determinado. Este último mecanismo basa su funcionamiento en el hecho de que los trabajos ocupan poco procesamiento y se realizan con gran frecuencia, mirando con una lupa, podríamos ver los microsegundos perdidos por concepto de cambios de contexto, bloqueos y desbloqueos de los hilos de trabajo. Contribución al entregable "Documento de implementación del *Middleware*, 2007".

2.4 Integración y pruebas

En adición a las restricciones impuestas por la gestión de la comunicación, el grado de interacción con el resto de los subsistemas, constituye uno de los principales pilares de complejidad en el desarrollo, pues se debe responder a intereses diversos, e implica gran esfuerzo en integración, a la hora de incorporar los cambios en el sistema. En el entregable “Informe de pruebas del *Middleware*, 2008” se recogen los casos de prueba y los resultados obtenidos.

Fue asignada la responsabilidad de realizar las entregas formales de las versiones piloto 7.02, 7.03 y 7.04 del SCADA Guardián del ALBA, así como entregas de código y documentación de las iteraciones de la fase Construcción de la metodología Proceso Unificado de Desarrollo: 1, 2.1, 2.2 y 2.3, implicando la participación en equipos de integración para la incorporación de cambios, con vistas a aplicar el componente de comunicaciones realizado en los módulos de ejecución reales.

Con vistas a validar en cierto grado las características funcionales del resultado, se asignaron las siguientes tareas:

- Pruebas funcionales al subsistema de comunicación
  Las pruebas realizadas estuvieron encaminadas a verificar las funcionalidades y el rendimiento fundamentalmente.
- Creación de la plataforma genérica de pruebas
  Debido a que *Middleware* está desarrollado en general como un conjunto de servicios independientes entre sí, para probar cada funcionalidad prácticamente es necesario escribir una herramienta de prueba desde cero, con vistas a probar un servicio en específicos. Por esta razón, surge la tarea de crear una aplicación parametrizable a través de un archivo de descripción de prueba, que creara las instancias particulares. El trabajo principal fue diseñar un formato de archivo genérico que permitiera crear cualquier tipo de prueba, teniendo en cuenta que cada una de ellas requería parámetros completamente diferentes y que la plataforma pudiera comportarse de manera estable y permitiera la recolección de estadísticas. La tarea se cumplió con éxito, la plataforma además de permitir realizar disímiles casos de prueba es extensible, lo que fue validado con la rápida incorporación de nuevas funcionalidades, tales como conexión a múltiples fuentes de puntos.

2.5 Publicaciones y eventos

Entre los eventos en que el *Middleware* ha sido presentado se encuentra UCIENCIA 2007, por Ing. José Omar Padrón, con el título "Componente *Middleware* para la comunicación intermodular del proyecto SCADA Nacional de PDVSA", en el cual se participó como co-autor y fue publicado en las memorias del mismo. En la Jornada Científica Estudiantil (JCE) del año 2007, fue premiado de Relevante en la Facultad 5 y en la Universidad con el título "*Middleware* para el SCADA PDVSA". En el XVI Fórum UCI de Ciencia y Técnica premio relevante, en el XVI Fórum Municipal de Ciencia y Técnica premio relevante y en el XVI Fórum Provincial de Ciencia y Técnica mención con el trabajo "*Middleware* para el SCADA PDVSA". Igualmente fue seleccionado para participar en el XVI Concurso de Computación y preseleccionado para participar en FIE'07.

Como parte del producto SCADA Guardián del ALBA, ha sido implantado en varias instalaciones de la hermana República Bolivariana de Venezuela, en fase de prueba, brindando resultados satisfactorios y meritando diferentes reconocimientos, tales como "Solución más Integral", en las Primera y Segunda Feria Expositiva de Productos de la UCI en el año 2008, en la primera de las cuales se participó como parte del equipo representante del *stan*. Obtiene el proyecto Guardián del ALBA del Polo de Automática y Hardware dos premios en el balance de producción 2008 efectuado el viernes 20 de Febrero 2009, otorgado por el Consejo de Dirección de la Universidad: "Premio al Mejor Proceso de Software" y "Premio al proyecto de Mayor Ingreso al País". Igualmente obtuvo premio en el XVII Concurso Nacional de Computación para Estudiantes, presentado por el autor, en representación del colectivo de proyecto.

2.6 Ingresos

Hasta la actualidad, por conceptos de ingresos directos, el proyecto ha aportado alrededor de 6 millones de dólares al país, con un reporte de utilidades netas del 90%. El SCADA Guardián del ALBA constituye un antecedente de la futura creación de la empresa mixta de software cubano – venezolana. Esta empresa en el futuro brindará grandes beneficios en la informatización de Nuestra América.

CONCLUSIONES

Con la exposición de tareas se logra dar una vista global de la participación activa del autor en el sistema de comunicaciones del SCADA Guardián del ALBA. Mostrando variantes que optimizaron la propuesta.

La realización del *Middleware* demuestra la factibilidad de reutilizar componentes en *software* libre de manera eficiente, extensible y escalable para la comunicación interprocesos de aplicaciones de control. El mismo garantiza el desacople entre los diferentes módulos, brindándoles un contexto homogéneo de desarrollo.

A través de un mecanismo general se logra cubrir las necesidades específicas del subsistema de Configuración, tanto en frío como en caliente, y del subsistema Históricos, usando un modelo de comunicación bidireccional.

La plataforma de pruebas creadas para el subsistema de comunicación asíncrono minimiza el costo de probar e integrar nuevas funcionalidades, además, permite observar el comportamiento del *Middleware* en diferentes ambientes que de otra forma serían difíciles de reproducir.

RECOMENDACIONES

Las tecnologías de la informática evolucionan constantemente, se recomienda realizar un estudio evaluativo de las nuevas tecnologías con vistas a incorporar al *Middleware* las mejoras existentes, así como a mantener estrecho seguimiento de las actualizaciones de la tecnología usada y de los errores reparados.

Se recomienda continuar el desarrollo con vistas a optimizar los mecanismos de comunicaciones, minimizar el número de canales, aumentar la tolerancia ante fallas y garantizar la seguridad de la información transmitida.

Incorporar las técnicas de redundancia disponibles en la tecnología usada en el sistema de comunicaciones y tener en cuenta este aspecto para futuras selecciones tecnológicas de *Middleware.*

REFERENCIAS

Anderson, Ross. 2001. Distributed Systems. *Security Engineering: A Guide to Building Dependable Distributed Systems.* 2001.

Coulouris, George, Dollimore, Jean y Kindberg, Tim. 2001. *Distributed Systems: Concepts and Design.* s.l. : China Machine Press, 2001. ISBN 7-111-11749-2.

Farley, Jim. 1998. *Java Distributed Computing.* s.l. : O'Reilly, 1998. ISBN-10: 1-56592-206-9.

Gamma, Erich, y otros. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Addison-Wesley, 1994. ISBN 0-201-63361-2.

GNOME Foundation. 2002. Orbit2. [En línea] GNOME Foundation, 2002. http://projects.gnome.org/ORBit2.

González Vázquez, Fidel, 2006. *Documento de pruebas del núcleo Linux.* Mérida, Mérida, Venezuela : s.n., 2006.

Henning, Michi y Vinoski, Steve. 1999. *Advanced CORBA® Programming with C++.* s.l. : Addison Wesley, 1999. ISBN 0-201-37927-9.

Herrera, Moisés, y otros. 2007. *Especificación de Comandos del SCADA.* Mérida, Venezuela : s.n., 2007.

MICO Project Team. 2006. MICO CORBA. [En línea] ObjectSecurity Ltd., 2006. http://www.mico.org.

ORGANIZACIÓN INTERNACIONAL DE NORMALIZACIÓN. 1987. *ISO 690.* 1987.

PDVSA. PDVSA - Petróleos de Venezuela S.A. [En línea] http://www.pdvsa.com.

Pérez Javier, Maikel. 2007. *Especificación de requerimientos del Middleware del SCADA Nacional.* Mérida, Venezuela : s.n., 2007.

Pérez Javier, Maikel y Chávez Lorenzo, Ariel. 2007. *Documento de diseno del Middleware*. Mérida, Mérida, Venezuela : s.n., 7 de mayo de 2007a.

—. 2007. *Implementación del Middleware*. Mérida, Mérida, Venezuela : s.n., 13 de diciembre de 2007b.

—.2008. *Especificación de las nuevas funcionalidades a incorporar, plan de pruebas y resultados a obtener*. *Proyecto de Comunicación con Terceros de la versión 2.0 del SCADA Guardián del ALBA.* Mérida, Mérida, Venezuela : s.n., 15 de octubre de 2008a.

—. 2008. *Informe de pruebas del Middleware*. Mérida, Mérida, Venezuela : s.n., 21 de febrero de 2008b.

Rojo, J. Oscar. 2003. Introducción a los sistemas distribuidos. [En línea] 2003.

Schmidt, Douglas C. y Schantz, Richard E. 2005. Middleware for Distributed Systems. [En línea] 2005. http://www.cs.wustl.edu/~schmidt/PDF/middleware-encyclopedia.pdf.

Stroustrup, Bjarne. 1986. *The C++ Programming Language.* Tercera Edición. 1986. pág. 328. ISBN 9780201120783.

Tellería Martínez, Ana Silvia. 2007. *Diseño de un Middleware Básico para el Intercambio de Información en un sistema Supervisor de Procesos Automatizados.* Ciudad de la Habana, Cuba : sn., Universidad de las Ciencias Informáticas 2007. Registro MIS-005861.

Universidad de las Ciencias Informáticas. 2007. Carta del Rector - Portal UCI | Universidad de las Ciencias Informáticas. [En línea] 2007. http://www.uci.cu/?q=node/47.

Universidad Politécnica de Cataluña. 2005. Conceptos generales de sistemas distribuidos. [En línea] septiembre de 2005. http://studies.ac.upc.edu/EPSC/FSD/FSD-ConceptosGenerales.pdf.

ANEXOS

Anexo A Comunicación a través del Middleware

Módulos que se comunican de forma asíncrona(Modelo Proveedor/Consumidor).

1. BDTR con el Módulo de históricos.
2. BDTR con otras BDTR.
3. BDTR con Visualización y/o aplicaciones que requieran puntos y alarmas.
4. Reportes con Visualización.

Módulos que se comunican de forma sincrónica (Modelo Cliente/Servidor).

1. Configuración con los restantes subsistemas.
2. Monitoreo con los restantes subsistemas.
3. Seguridad con los restantes subsistemas
4. Tolerancia a fallas con los restantes subsistemas.
5. Reportes con módulos de Históricos.

GLOSARIO

ACE (Adaptive Communications Environment): Traducido al español como entorno de comunicación adaptativa, es un framework software libre y orientado a objetos. Implementa varios núcleos de patrones para programación concurrente. Incluye: administración de eventos, manejo de señales, inicialización de servicios, comunicación interprocesos, administración de la memoria compartida, enrutamiento de mensajes, reconfiguración dinámica de los servicios distribuidos, ejecución y sincronización concurrente. (Schmidt, 2007)

API (Application Programming Interface): Conjunto de rutinas y definiciones de funciones que abstraen los detalles de la implementación y hace más fácil el desarrollo de aplicaciones.

Cortafuego: Herramienta del sistema operativo que mantiene la seguridad de acceso a través de la red a las computadoras.

FPU: Unidad de Punto Flotante o Unidad de Coma Flotante (*Floating Point Unit* en inglés)

Free Software Foundation (FSF): Entidad que busca eliminar las restricciones de uso, copia, modificación y distribución del software.

General Public License (GPL): Esta licencia regula los derechos de autor de los programas de software libre (free software) promovido por la FSF en el marco de la iniciativa GNU.

Hilos: Distintas partes en las que un programa se puede dividir para ejecutar varias tareas paralelamente o pseudo-paralelas, dependiendo del número de unidades de procesamiento CPU.

Latencia de interrupción: Tiempo que transcurre entre que el sistema operativo recibe una señal de interrupción hasta que pasa el control efectivamente al proceso que maneja la petición de interrupción.

ORBIT: ORB compatible con CORBA 2.4. Con soporte para los lenguajes C, C++ y Python, distribuido mayormente sobre licencia LGPL, aunque algunos componentes son GPL. Originalmente fue desarrollado como *middleware* para el proyecto GNOME, pero ha tenido otros usos.

RTAI: Permite escribir aplicaciones con estrictas restricciones de tiempo, basado en el núcleo de Linux. Es mantenido por la comunidad y cuenta con soporte para las arquitecturas x86 (con y sin FPU y TSC) , x86_64, PowerPC y ARM (StrongARM; ARM7: clps711x-family, Cirrus Logic EP7xxx, CS89712, PXA25x) También incluye RTAI-Lab, herramienta de programación visual.

TAO (The ACE ORB): ORB de segunda generación, compatible con la especificación CORBA V3.0, para el lenguaje C++, desarrollado con una arquitectura altamente extensible basado en la biblioteca ACE.

Tecnología de Información (TI): Término muy general que se refiere al campo entero de la tecnología informática, que incluye desde hardware de computadoras y programación hasta administración de redes.
TSC: Contador de Marca de Tiempo (*Time Stamp Counter* en inglés) es un registro de 64-*bit* presente en todos los procesadores x86 desde Pentium.