UNIVERSIDAD CENTRAL "MARTA ABREU" DE LAS VILLAS
VERITATE SOLA NOBIS IMPONETUR VIRILIS TOGA. 1948

Facultad de Ingeniería Eléctrica
Departamento de Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

Interfaz OPC para SCADA Nacional de PDVSA

Autores: **Sándor Otero Rodríguez**
Roberto Carlos Rodríguez López

Tutor: **M. Sc. René González Rodríguez**

Santa Clara
2007
"Año 49 de la Revolución"

Universidad Central "Marta Abreu" de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones y Electrónica

TRABAJO DE DIPLOMA

Interfaz OPC para SCADA Nacional de PDVSA

Autores: **Sándor Otero Rodríguez**
e-mail: sorlovergg@gmail.com

Roberto Carlos Rodríguez López
e-mail: rcrguez@gmail.com

Tutor: **M. Sc. René González Rodríguez**
Especialista en Automática
Empresa de Automatización Integra CEDAI
Sucursal Villa Clara
e-mail: voltus@cedaivc.co.cu

Santa Clara

2007

"Año 49 de la Revolución"

Hacemos constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Telecomunicaciones y Electrónica, autorizando a que el mismo sea utilizado por la Institución para los fines que estime conveniente, tanto de forma parcial como total, y que además, no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

_____ _____

Firmas de los Autores

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firmas de los Autores

_____ _____

Firma del Jefe de Departamento donde se defiende el trabajo

Firma del Responsable de Información Científico-Técnica

TAREA TÉCNICA

- Estudio de los requerimientos del proyecto SCADA Nacional.
- Estudio del estado del arte de los estándares de OPC.
- Estudio de la especificación de acceso a datos, OPC-DA.
- Búsqueda bibliográfica y estudio de trabajos relacionados con los túneles de OPC Windows-Linux.
- Estudio de las tecnologías de comunicación entre computadoras.
- Análisis y diseño del sistema.
- Estudio sobre herramientas de programación en Linux y Windows.
- Implementación y análisis de los resultados.

_____ _____

Firmas de los Autores

_____

Firma del Tutor

RESUMEN

El presente trabajo tiene como propósito dotar, al SCADA Nacional de PDVSA, de la posibilidad de interacción con servidores externos que soporten el estándar OPC-DA. Consiste en el diseño e implementación de un sistema para la comunicación entre el SCADA hecho sobre Linux y los servidores de OPC que se ejecutan sobre plataforma Windows. La inclusión de una interfaz OPC en el desarrollo del SCADA es de gran importancia, dada la existencia de una gran variedad de dispositivos de campo que utilizan este estándar para su comunicación con software de supervisión.

El sistema implementado está compuesto por dos módulos: un *Gateway* o Pasarela en Windows y un *Driver* o Manejador sobre plataforma Linux. El primero tiene como función interactuar con los servidores de OPC para dar respuesta a las peticiones provenientes de los clientes del SCADA. El segundo es la vía que poseen dichos clientes para interactuar con el *Gateway*, y de esa forma acceder a las variables disponibles en los servidores de OPC.

La comunicación se realiza a través del *framework* multiplataforma CORBA, específicamente la implementación ACE-TAO, garantizando una excelente estabilidad y cumpliendo los requerimientos del proyecto.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Petróleos de Venezuela Sociedad Anónima (PDVSA), es la corporación estatal de la República Bolivariana de Venezuela dedicada a la exploración, producción, manufactura, transporte y mercadeo de los hidrocarburos. Es la cuarta compañía petrolera a nivel mundial y la empresa más grande de Latinoamérica (Prensa PDVSA 2007). Entre sus fines también se encuentran "...motorizar el desarrollo armónico del país, afianzar el uso soberano de los recursos, potenciar el desarrollo endógeno..." (PDVSA 2005a).

PDVSA cuenta en sus instalaciones con Sistemas de Supervisión y Adquisición de Datos (Kane 2007), suministrados por compañías extranjeras, que son imprescindibles para la confiabilidad y eficiencia de sus procesos tecnológicos; pero el costo de mantenerlos en funcionamiento es elevado e imponen una total dependencia de los proveedores.

Algunas de estas compañías se plegaron y apoyaron con la inadecuada operación de estas tecnologías, el sabotaje petrolero ocurrido durante diciembre del 2002 y enero del 2003. PDVSA sufrió pérdidas de aproximadamente 14 430 millones de dólares en calidad de las ventas no efectuadas (PDVSA 2005b). Estos hechos evidenciaron la necesidad de luchar por la independencia tecnológica de PDVSA como empresa medular en la economía venezolana.

Debido a la alta subordinación técnica inherente a los productos existentes en el mercado, no era posible contar con ellos como solución a la problemática. Luego, como parte de las premisas y los objetivos analizados en el plan de negocios de la corporación del período 2004 – 2009, y considerados con anterioridad en los planes del 2001 al 2007 de desarrollo nacional de Venezuela, surge el proyecto SCADA Nacional (PSN), con el propósito de "Desarrollar un Sistema que supervise, controle, optimice y gestione los procesos industriales de PDVSA sirviendo de base para la implantación en otras Industrias y Organismos del país" (PDVSA 2006a).

Paralelamente, en el 2004, se ponen en práctica leyes que promueven la utilización de software libre y estándares abiertos en los sistemas, proyectos y servicios informáticos de la

Administración Pública Nacional de Venezuela, además de orientar la migración en el caso del software privativo (Chávez 2004).

En el 2006 el estado cubano se incorpora al proyecto conocido desde entonces en Cuba como SCADA PDVSA. Se firmó un convenio de trabajo entre ambas partes para la obtención del apoyo cubano en diferentes líneas, y quedó planteado de manera expedita que todos los componentes de las tareas propuestas debían ser desarrollados con software libre y con la utilización de estándares abiertos (PDVSA 2006b). Inmediatamente comenzó la participación activa en el proyecto de profesores y estudiantes provenientes de la UCI, empresas de automática y otras universidades del país.

Una de las líneas firmadas en contrato como responsabilidad cubana, tiene entre otros objetivos, el de desarrollar una Interfaz OPC para el Sistema Supervisor de Procesos Automatizados de la compañía PDVSA. De esta forma se garantizaría el flujo de información en ambos sentidos entre el SCADA y los servidores (encargados de intercambiar datos de forma directa con la red de dispositivos de campo) que soportan el estándar OPC *(OLE for Process Control)*. La industria petrolera venezolana, primera población donde se insertaría el SCADA por desarrollar, posee dos características importantes a tener en consideración durante la producción del software; su magnitud y su requerimiento de ejecución en tiempo real. Varios de sus procesos industriales necesitan seguimiento en intervalos de valores en el orden de los milisegundos.

Proceder con la elaboración de un software intermedio de tal magnitud y complejidad, sin poseer experiencia antecedente de su implementación con tecnología libre, como sucede en nuestro caso, representa un gran riesgo.

Para solucionar este problema se propone hacer el diseño y la implementación de un sistema de comunicación que le brinde al SCADA Nacional la posibilidad de interactuar con servidores que soportan el estándar OPC. Específicamente se pretende:

- Realizar un estudio de las especificaciones del estándar OPC.
- Realizar un sistema de comunicación para el enlace de un cliente en Linux y un servidor en Windows.
- Implementar el cliente IP sobre Linux y un *Gateway* o pasarela en Windows para permitir la integración del SCADA al estándar OPC.

Para lograr los objetivos se plantean las siguientes tareas de investigación:

- Estudio de los requerimientos del proyecto SCADA Nacional.
- Estudio del estado del arte de los estándares de OPC.
- Estudio de la especificación de acceso a datos, OPC-DA.
- Búsqueda bibliográfica y estudio de trabajos relacionados con los túneles de OPC Windows-Linux.
- Estudio de las tecnologías de comunicación entre computadoras.
- Análisis y diseño del sistema.
- Estudio sobre herramientas de programación en Linux y Windows.
- Implementación y análisis de los resultados.

Para su confección, el trabajo, se dividió en tres capítulos que abarcan los elementos necesarios para la comprensión e implementación del sistema.

El capítulo 1 contempla generalidades sobre los sistemas SCADA. Parámetros relacionados con el proyecto SCADA Nacional de PDVSA. Una breve introducción al conjunto de estándares OPC, profundizando en el estándar de acceso a datos y detallando las características de un servidor de OPC-DA, sus objetos principales e interfaces.

En el capítulo 2 se describen los sistemas distribuidos, evaluando los modelos correspondientes. Se hace un exhaustivo estudio de las tecnologías de comunicación, profundizando en la más apropiada según sus potencialidades y los requerimientos del proyecto.

El capítulo 3 concluye con el análisis y diseño del sistema a implementar. Se realiza un breve análisis de las herramientas para el desarrollo de clientes que soportan el estándar OPC, así como de las herramientas de programación para Linux y Windows. Además se comprueba el correcto funcionamiento a través de pruebas reales de comportamiento.

Con la ejecución del proyecto se dan soluciones a problemáticas modernas vinculadas con la adquisición de datos, su transmisión hacia los clientes y administradores del sistema, y el logro de la independencia tecnológica, lo cual tiene un impacto directo sobre la economía, dada la posibilidad de prevención a tiempo de desastres, además de poder controlar en tiempo real los dispositivos de campo.

Los resultados del proyecto fueron expuestos en el Forum Estudiantil a nivel universitario en la Comisión de Telecomunicaciones y Electrónica, obteniendo el premio Relevante Especial, además fue seleccionado para participar en la Convención de Ingeniería Eléctrica CIE 2007.

Como constancia del desarrollo de este proyecto se cuenta con un Aval emitido por la Empresa de Automatización Integral CEDAI Villa Clara, la cual está vinculada directamente con varias de las líneas que conforman el proyecto SCADA Nacional (Ver Anexo 1).

CAPÍTULO 1. SCADA NACIONAL DE PDVSA. ESTÁNDAR DE ACCESO A DATOS OPC-DA

1.1. Introducción a los Sistemas SCADA

SCADA viene de las siglas de *Supervisory Control And Data Acquisition*, es decir: Adquisición de Datos y Supervisión de Control. Se trata de una aplicación software especialmente diseñada para funcionar sobre ordenadores de control de producción, proporcionando comunicación con los dispositivos de campo (controladores programables, instrumentos de medición, actuadotes, etc.) y controlando y supervisando el proceso de forma automática desde la pantalla del ordenador. Además, provee de toda la información que se genera en el proceso productivo a diversos usuarios, tanto del mismo nivel como de otros supervisores dentro de la empresa: control de calidad, supervisión, mantenimiento, etc. (NeutralBit 2006).

1.1.1. Funciones

Un SCADA debe cumplir tres funciones principales según Balcells en (Balcells 2005):

- Adquisición de datos para recoger, procesar y almacenar la información recibida.
- Supervisión, para observar desde el monitor la evolución de las variables del proceso.
- Control para modificar la evolución del proceso, actuando bien sobre los reguladores autónomos básicos (consignas, alarmas, menús, etc.), o directamente sobre el proceso mediante las salidas conectadas.

1.1.2. Requisitos

Un SCADA debe cumplir varios objetivos para que su instalación sea perfectamente aprovechada (Balcells 2005):

- Deben ser sistemas de arquitectura abierta, capaces de crecer o adaptarse según las necesidades cambiantes de la empresa.
- Deben comunicarse con total facilidad y de forma transparente al usuario, con el equipo de planta y con el resto de la empresa (redes locales y de gestión).
- Deben ser programas sencillos de instalar, y fáciles de utilizar, con interfaces amigables con el usuario.

1.1.3. Estructura y componentes

Los módulos o bloques software que permiten las actividades de adquisición, supervisión y control son los siguientes (Balcells 2005):

- **Configuración:** permite al usuario definir el entorno de trabajo de su SCADA, adaptándolo a la aplicación particular que se desea desarrollar.
- **Interfaz gráfica del operador:** proporciona al operador las funciones de control y supervisión de la planta. El proceso se representa mediante sinópticos gráficos.
- **Módulo de proceso:** ejecuta las acciones de mando preprogramadas a partir de los valores actuales de variables leídas. La programación se realiza por medio de bloques de programa en lenguaje de *scripts* de alto nivel (como C, *Visual Basic for Application* VBA, entre otros).
- **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 ellos.
- **Comunicaciones:** transfiere información entre la planta y la arquitectura hardware que soporta el SCADA, y entre esta y el resto de los elementos informáticos de gestión.

1.2. SCADA Nacional de PDVSA

En las líneas de PDVSA se señala una dependencia existente de sistemas SCADA propietarios, los cuales en muchos casos no cubren las expectativas de diseño del usuario final. Algunos de estos sistemas están obsoletos y sobredimensionados, otros sin soporte del fabricante y poca disponibilidad de personal especializado en las regiones. Dicha problemática involucra igualmente el incurrir en riesgos de fallas operacionales. Esto afecta a los responsables de las operaciones de los diferentes negocios de PDVSA y de los

sistemas automatizados de monitoreo y control, así como al personal involucrado en los procesos y tecnología de soporte (equipos críticos y otros sistemas de apoyo).

Lo antes expuesto provoca:

- Baja capacidad de respuesta a eventos no programados.
- Disminución de los niveles de producción.
- Riesgos a la integridad de personas y equipos.
- Desconocimiento de acciones de control tomadas por personas o sistemas automatizados.
- Costos incrementados de operación.
- Riesgos a la continuidad operacional indispensable para la empresa.

Una solución exitosa sería un Sistema Integral de Supervisión, Control y Adquisición de Datos adaptable a todas las operaciones y negocios de PDVSA, que permita sin dependencias externas, supervisar y controlar los procesos operacionales, minimizar los riesgos, auditar las acciones ejecutadas y manejar eficientemente el uso de los recursos.

Hoy formamos parte de esta solución que va dirigida a los diferentes negocios de PDVSA.

Este producto a elaborar permitirá:

- Supervisar y controlar las operaciones actuales y futuras de PDVSA.
- Minimizar riesgos a la integridad de las personas, ambiente y de los activos monitoreados.
- Mejorar la eficiencia en el uso de los recursos y aportar datos a las distintas aplicaciones de PDVSA que los requieran en sus operaciones.
- Establecer responsabilidades sobre las acciones ejecutadas en los distintos procesos operacionales.
- El control por parte de PDVSA de la tecnología utilizada.

Esta solución enmienda los sistemas SCADA usados actualmente, cuya tecnología, crítica para el negocio, es controlada por terceros o se encuentra ya en estado de obsolescencia.

El SCADA Nacional estará basado en tecnologías y herramientas que permitan manejar y representar el proceso operacional del negocio al cual atiende; de forma escalable,

distribuido y extensible para la configuración y programación de sus componentes, con un enfoque a la soberanía tecnológica a través de su mantenimiento por las empresas nacionales, las PyMES y las universidades, bajo la coordinación de PDVSA.

1.2.1. Parámetros de estricto cumplimiento

El Proyecto SCADA Nacional para su elaboración está dividido en varias líneas de trabajo las cuales deben cumplir restricciones, rangos de calidad y requerimientos.

1.2.1.1. Restricciones

- El sistema debe cumplir, como producto final, con los esquemas y estándares abiertos y de software libre, de tal manera que la puesta en marcha del mismo no implique gastos en licencias de ningún tipo. Esto no limita el uso de herramientas propietarias para el desarrollo del producto (Free Software Foundation 2006) (Stallman 2006).
- El sistema debe ser distribuido, es decir permitir la distribución de datos a lo largo de su arquitectura, permitiendo la fácil adaptación y escalabilidad a los distintos contextos de las áreas operativas.
- El sistema debe contar con mecanismos de tolerancia a fallas, ya que no se debe permitir fluctuaciones en el funcionamiento normal del sistema. Una alternativa a implementar es la redundancia de los servidores y la opcional implementación de esquemas de distribución de carga.
- El lenguaje de programación a ser utilizado para la recolección de datos y la arquitectura basada en servicios será C y C++ respectivamente, se descarta el uso de otros lenguajes en estos módulos.
- Las estructuras y tipos de datos empleados por el sistema SCADA para la integración de este con otros sistemas o entre sus módulos internos, deberán emplear mecanismos como XML en donde los datos viajan en un esquema con formato de texto.

1.2.1.2. Rangos de Calidad

El sistema SCADA debe ser capaz de manejar los puntos que requieran las operaciones. En las primeras versiones a ser instaladas en el Patio de Tanques San Silvestre (PTS) y en Distrito Norte, estas se estiman en 10.000 puntos.

Adicionalmente, el SCADA debe manejar redundancia y recuperación automática a fin de que la aplicación SCADA tenga una disponibilidad de 99,99%. Esto a su vez requerirá implementaciones de sistemas de respaldo de los datos en línea, a fin de que durante el respaldo de los mismos, el sistema pueda continuar monitoreando y controlando los procesos. Lo mismo aplica para cualquier elemento de edición (despliegues, bases de datos, etc.).

En cuanto a tiempos mínimos de recolección de datos y tiempos de respuesta, la aplicación debe ser capaz de manejar los valores de tiempos establecidos en la documentación de desarrollo del SCADA (Kane 2007) (sujeto a validación y medición del impacto en el SCADA).

1.2.1.3. Requerimientos

Requerimientos del Sistema

- Servidores y Estaciones de Trabajo: Debian con *kernel* Linux 2.6.x o superior.

Requerimientos de Rendimiento

- El módulo de datos se implementará con PostgresSQL (Sitio Oficial: http://www.postgresql.org) como base de datos histórica. Esta base de datos tendrá la capacidad de almacenar los datos adquiridos hasta 3 meses de historia en línea y mínimo de 1 año fuera de línea.
- De igual forma, esta base de datos será constantemente accesada por otras aplicaciones a fin de obtener los datos que el sistema ha adquirido y almacenado, y cuyos tiempos de respuesta deben ser menores de 10 segundos para una consulta. Esta capacidad no debe degradar el tiempo de respuesta de la adquisición de los datos de campo, los cuales deben estar dentro de los 5 milisegundos en el *cache*, mecanismo para garantizar la alta respuesta en la interrogación a los PLC, y

específicamente evitar retrasos que puedan repercutir en las actividades de estos dispositivos.

1.2.2. Necesidad de una interfaz OPC

La necesidad de implementar sistemas donde el proceso automatizado y la gestión de la información se cohesionen, para formar un todo único, ha sido uno de los principales problemas a resolver por parte de las firmas especializadas en automatización como Siemens (Sitio Oficial: http://www.siemens.com/index.jsp), Progea (Sitio Oficial: http://www.progea.com), Allan Bradley (Sitio Oficial: http://www.ab.com) y OPC Foundation (Sitio Oficial: http://www.opcfoundation.org). En muchas ocasiones los fabricantes crean una barrera al desarrollo de sistemas de control de procesos debido a la no-compatibilidad de sus dispositivos e interfaces de comunicación.

Esto crea un problema a los desarrolladores de sistemas automatizados que se limitan a utilizar el hardware y software de un determinado suministrador, al no poder comunicar los sistemas de una firma con los de otra. OPC brinda una solución para el intercambio de información entre los dispositivos y las aplicaciones de producción y gestión de la información, sin importar el fabricante y bajo los preceptos del *plug and play* (Kondor 2003).

En el mundo industrial, la integración de componentes de diversos fabricantes ha resultado frecuentemente una tarea difícil, e incluso en la actualidad en ciertos casos lo sigue siendo. Las aplicaciones, como operaciones de control y monitorización o procesamiento de datos de gestión, requieren de un *driver* para cada componente de automatización cuyos datos necesiten ser leídos.

Dotar al SCADA Nacional de la interacción de clientes y servidores externos que soporten el estándar OPC, es una funcionalidad requerida dada la existencia de una variedad de dispositivos de campo propietarios que utilizan este estándar para su comunicación con software de supervisión.

1.3. Introducción al OLE *for Process Control* (OPC)

El primer borrador de las especificaciones OPC, surge en mayo de 1995 cuando Microsoft, unida a una fuerza constituida por cinco empresas (Intellution, Opto-22, Fisher-Rosemount, Rockwell Software e Intuitiv Software), creó un comité con el objetivo de definir completamente las interfaces basadas en el estándar OLE/COM, tarea esta que fue completada a finales de este propio año. Con la colaboración de unas 90 compañías a lo largo del mundo, fueron comprobadas estas especificaciones, para finalmente dar el primer conjunto de especificaciones completado en agosto de 1996. El objetivo del comité fue proporcionar un interfaz de programación de aplicaciones estándar para el intercambio de datos que puede simplificar el desarrollo de *drivers* de entrada / salida y mejorar el rendimiento de los sistemas (Kamen 1999).

Seguidamente en septiembre de 1996 se crea la Fundación OPC, a la cual se le asignó la tarea de establecer varios comités técnicos para extender el espectro y la función de las especificaciones de OPC. El desarrollo y mantenimiento de estos estándares fueron asumidos por dicha fundación.

La prioridad principal de OPC está en proporcionar un acceso flexible, poderoso y simple a los datos sin importar qué fabricante construya el dispositivo de origen de los mismos.

La Fundación OPC es una organización global, independiente y sin fines de lucro. Entre los miembros de la misma se cuentan varias compañías industriales líderes en su campo, que comparten la misma visión de una comunicación estandarizada entre hardware y software basada en la tecnología COM. Fabricantes de hardware y software, usuarios, integradores de sistemas, así como empresas e instituciones ajenas al mundo de la automatización, pueden formar parte de la Fundación OPC.

Las tecnologías COM/DCOM definidas desde un inicio como base para la comunicación en el estándar OPC, crean una gran dependencia de este con la plataforma Windows, pues aunque existen implementaciones de las tecnologías COM/DCOM para Linux, son ineficientes y de muy bajo rendimiento. Esta dependencia de los estándares de OPC con la plataforma Windows, genera un problema debido a que el SCADA será implementado sobre plataforma Linux. Una búsqueda realizada en Internet arrojó resultados poco alentadores en cuanto a soluciones a problemas de este tipo. Existe una compañía nombrada

Cogent Real-Time Systems que ha desarrollado túneles de OPC para la comunicación Linux Windows (Cogent Real-Time Systems 2007), pero el código es cerrado y por tanto no puede adecuarse a características específicas de quien pretendiera usarlo .

1.3.1. Estándares de OPC

La Fundación OPC ha dividido sus estudios y actualizaciones en dependencia de las necesidades que se han creado con el desarrollo de los sistemas automatizados. En respuesta a estos requerimientos se han creado varios estándares que especifican la forma de realizar tareas comunes en el campo del control de procesos.

El desarrollo de las especificaciones ha evolucionado a la par de las tecnologías. A continuación se enumeran las principales que rigen los estándares de OPC existentes hasta la actualidad (OPC Foundation 2007).

- **OPC *Data Access.*** Estándar para el intercambio de datos en tiempo real entre los clientes y servidores de OPC, es el más utilizado y el que más variaciones ha sufrido, existen versiones desde OPC-DA 1.0 hasta la 3.0.
- **OPC *Alarms & Events.*** Estándar de monitoreo de alarmas y eventos, hasta hoy existen dos versiones de este estándar, la OPC-AE 1.0 y 1.10.
- **OPC *Historical Data Access.*** Estándar de almacenamiento de datos históricos, evolucionó desde OPC-HDA 1.0 a 1.10.
- **OPC *Security*.** Estándar de requerimientos de seguridad de la información, este estándar posee solamente una versión, OPC *Security* 1.0.
- **OPC *Data eXchange*.** Estándar para el intercambio de información entre servidores de OPC, existe la versión OPC-DX 1.0.
- **OPC *eXtensible Markup Language.*** Estándar de intercambio de datos entre clientes y servidores de OPC con compatibilidad para sistemas multi-plataforma, existe la versión OPC-XML-DA 1.0.
- **OPC *Unified Architecture*.** OPC-UA la nueva visión de la Fundación OPC, se ha convertido en una de las tecnologías más relevantes de comunicación de datos de procesos, logrando unificar en una sola concepción los servicios especificados en las versiones anteriores.

1.3.2. Justificación del estándar

Según encuesta hecha a técnicos de PDVSA, la mayoría de los clientes y servidores de OPC utilizados en sus sistemas implementan OPC-DA 2.05, por lo que será este el estándar a comunicar con el SCADA, cumpliendo los requerimientos de mantener en total funcionamiento la tecnología que existe actualmente.

La evolución de OPC-DA ha estado muy ligada al desarrollo de las tecnologías de programación. Desde la aparición de DCOM, *Distributed Component Object Model*, hasta las más recientes orientadas a servicios WEB. OPC-DA, surgió basada en DCOM y evolucionó desde OPC-DA 1.0 hasta OPC-DA 3.0. A partir de la versión OPC-DA 2.05 se introdujo el soporte de datos complejos, esta versión del estándar es una de las más difundidas en la actualidad.

1.4. Descripción del estándar OPC *Data Access*

Las especificaciones de Acceso a Datos de OPC, se especializan en la transmisión, planificación e intercambio de datos entre el servidor y sus clientes orientados a interfaces COM/DCOM.

La captura de datos en tiempo real desde los dispositivos de campo, supervisión del proceso, estado actual de los dispositivos y subsistemas, son algunas de las tareas que en él se perfilan. La Figura 1.1 muestra un sistema de control y supervisión de flujo utilizando el estándar OPC-DA. El sistema está formado por una bomba que se encuentra controlada por un PLC y un flujómetro que realimenta al PLC.

El servidor de OPC-DA, debe ser capaz de captar y modificar, por medio del *driver* de comunicación con el PLC, el estado del sistema de control (valor del flujo, valores de los parámetros del controlador, y otros.).

Los clientes de OPC-DA podrán acceder a los datos captados por medio del servidor de OPC-DA que se actualiza a través del *driver* de comunicación con el PLC. En este caso existe un cliente en la misma PC del servidor como cliente de HMI (*Human Machine Interface*) y dos clientes en red con aplicaciones que requieren datos del proceso. Estas aplicaciones pueden o no ser del mismo fabricante, siempre y cuando cumplan con el estándar de OPC-DA (Suchit 2005).

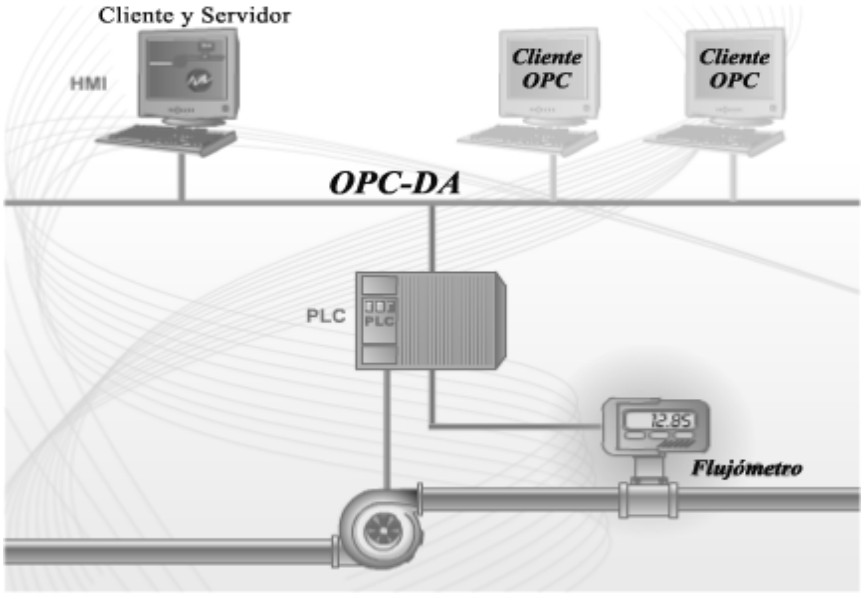

Figura 1.1. Ejemplo de sistema de control de flujo utilizando el estándar OPC-DA, en aplicaciones de supervisión con varios clientes.

1.4.1. Modelo de objetos de los servidores de OPC-DA

Los servidores de acceso de datos OPC-DA contienen los siguientes objetos: Grupos (*Group*) e Ítems (*Item*), mostrados en la Figura 1.2.

El servidor mantiene la información y es usado como depósito de los grupos, los cuales contendrán los objetos que se comparten a los clientes, los ítems. Ambos, el servidor de OPC y los diferentes grupos compartidos por el servidor implementan varias interfaces para la manipulación de datos.

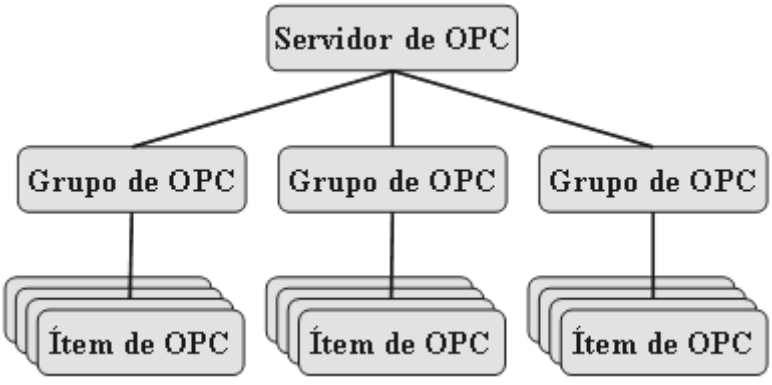

Figura 1.2. Arquitectura de un servidor de OPC-DA.

1.4.1.1. Grupo de OPC

Cada grupo de OPC-DA, tiene un nombre único. El cliente de OPC puede especificar si un grupo determinado está en estado activo o no. Los grupos de OPC-DA pueden ser creados dinámicamente acorde a las aplicaciones que lo requieran (clientes). Un cliente puede agrupar varios ítems en un grupo para organizar los accesos al servidor de OPC. Por ejemplo, un sistema supervisor que requiera muestrear varias variables con el mismo tiempo de muestreo, deben ser organizadas en un mismo grupo, así también pueden realizarse grupos lógicos que representen zonas del proceso (Suchit 2005).

Un grupo presenta las siguientes propiedades:

***Name*:** identificador único, en el mismo deben respetarse las mayúsculas y minúsculas.

Active: define si un grupo está activo, en caso contrario, todos los ítems que contiene no serán refrescados.

***UpdateRate*:** esta propiedad define el tiempo de actualización del dato desde el servidor. Todos los ítems que están incluidos en el grupo serán refrescados por el servidor en el tiempo descrito, en caso contrario se genera una excepción.

***TimeZone*:** en algunos casos se muestrean los datos en un lugar con diferencias de horario, esta propiedad ayuda a que se corrijan este tipo de errores.

***PercentDeadBand*:** el rango de la zona muerta es desde 0 hasta 100. La zona muerta es aplicada solamente a valores del tipo analógico, esta propiedad se utiliza para generar excepciones en caso de que se superen los límites permitidos. Puede colocarse cuando se crea el nuevo grupo, de esta forma todos los ítems internos presentarán el mismo valor, aunque la versión 3.0 de la especificación OPC-DA, ya permite que cada ítem presente esta propiedad. Su principal aplicación es hacer un filtrado en caso de que los valores estén defectuosos por causa del ruido.

***ClientHandle*:** es un manipulador devuelto en cada evento. Esto permite al cliente identificar el grupo al cual pertenecen los datos. Es esperado que el cliente asigne un valor único a cada manipulador.

1.4.1.2. Ítem de OPC

A diferencia de los grupos y servidores de OPC, los ítems no implementan ninguna interfaz de OPC y por lo tanto no es un objeto COM. Es un objeto interno que el servidor de OPC alberga, contiene la información que debe ser compartida al cliente, por ejemplo, tipo de datos, valor, si está activo, si está protegido contra escritura, etc.

Desde la perspectiva de un cliente de OPC, un ítem representa la conexión lógica a la fuente de datos que comparte dicho objeto.

Usando el identificador del ítem, ItemID, el cliente puede realizar las solicitudes o escrituras de los datos compartidos, teóricamente, no existe límite para la cantidad de ítems que publica un servidor, incluso en un solo grupo del servidor (Suchit 2005).

Un ítem provee un punto de conexión entre el objeto físico o lógico que ofrece el valor, por ejemplo una consigna de control.

Las propiedades de un ítem son (Ver Figura 1.3):

***CurentValue*:** Valor actual del ítem, entregado por el servidor de OPC-DA.

***Quality*:** Calidad del valor leído, puede ser buena o mala en caso de errores en el servidor.

***TimeStamp*:** Tiempo en que fue muestreado el dato.

***Data Type*:** Tipo de dato en que se entrega el valor del ítem al cliente.

***CanonicalDataType*:** Los valores siempre son devueltos en tipo VARIANT, en el servidor cada ítem puede tener un tipo de dato definido por defecto, a este tipo es el que se le denomina CanonicalDataType.

***Access Rights*:** Si el ítem presenta permiso de escritura o no.

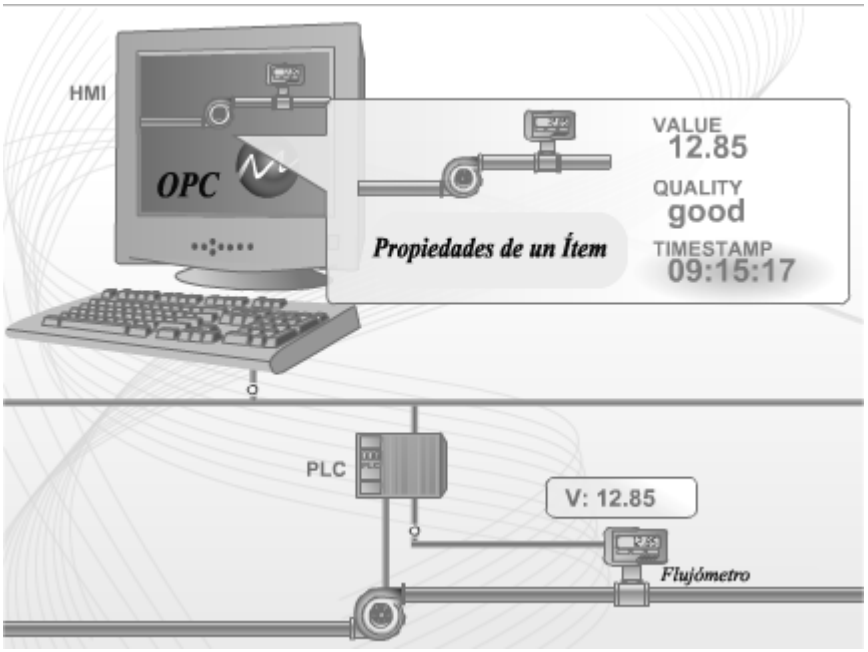

Figura 1.3. Propiedades de un ítem de OPC-DA, en un sistema de control de flujo.

1.4.1.2.1. Identificador del ítem, ItemID

El identificador del ítem, es usado por los clientes para establecer la conexión al servidor de OPC, no es más que una cadena compuesta por el camino de todos los niveles que presenta el ítem.

Ejemplo de ello es *ServidorOPC1.Grupo2.Item3*

Cuando un cliente utiliza la interfaz *IOPCBrowseServerAddressSpace*, puede obtener la lista de las colecciones de variables compartidas por el servidor para crear su propia configuración. Esta interfaz es utilizada en versiones de OPC-DA menores de 3.0, a partir de esta se elimina esta interfaz y se implementa la *IOPCBrowse*.

1.4.2. Interfaces del modelo de objetos de los servidores de OPC-DA

El objeto servidor de OPC brinda la posibilidad al cliente de crear y manipular los objetos grupo de OPC a través de una serie de interfaces. Estos grupos les permiten a los clientes organizar los datos a los que ellos quieren acceder y pueden activarse y desactivarse como una unidad. Además un grupo proporciona al cliente la forma de subscribirse a la lista de ítems para poder notificarse cuando ellos cambien (Brochure 2006).

Esta sección incluye una breve referencia de la interfaces existentes en los objetos Servidor y Grupo, así como a sus métodos y comportamiento.

1.4.2.1. Interfaces del Objeto Servidor

La Figura 1.4 describe las interfaces que implementa un objeto servidor.

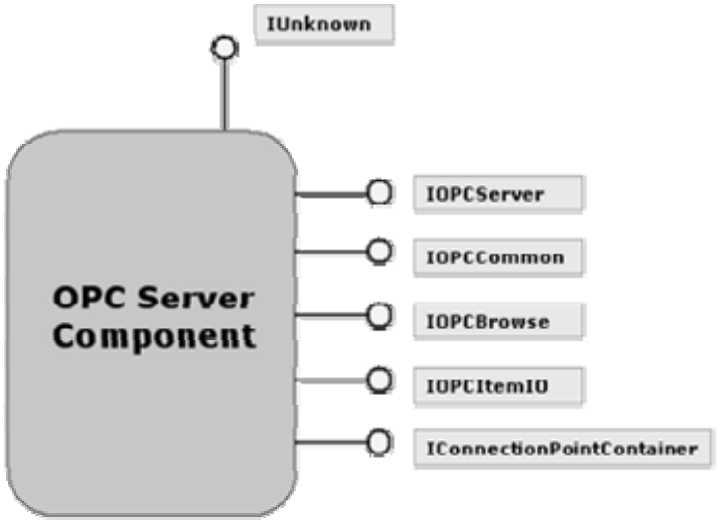

Figura 1.4. Objeto servidor y sus interfaces.

IOPCServer: Es la interfaz principal del servidor de OPC, con ella se pueden manipular los objetos grupos y los errores. Implementa las siguientes funciones:

- **AddGroup:** Adiciona un objeto grupo a la lista del servidor.
- **GetErrorString:** Retorna una cadena para un código de error específico.
- **GetGroupByName:** Devuelve un puntero a una interfaz del objeto grupo identificado por su nombre.
- **GetStatus:** Retorna la información del estado actual del servidor.
- **RemoveGroup:** Borra un objeto grupo de la lista del servidor.
- **CreateGroupEnumerator:** Crea un objeto encargado de enumerar los grupos según su clase. Este método puede ser utilizado por los clientes para hacer un listado de las conexiones a los grupos que existen en el servidor.

IOPCBrowse: Proporciona métodos para la búsqueda del espacio de dirección del servidor y para obtener las propiedades de los ítems. Sus métodos son:

- **Browse:** Explora una rama del espacio de direcciones del servidor a partir de una rama especificada o total. Con él se puede obtener información de los ítems, sus propiedades y estado.
- **GetProperties:** Devuelve un arreglo de las propiedades de los ítems, esta información la contienen los datos del tipo OPCITEMPROPERTIES definidos en el estándar y se devuelve uno para cada ítem especificado.

IOPCCommon: Esta es la interfaz básica que contiene todas las definiciones de tipo y de manejo de enumerativos y errores, otros tipos de servidores OPC como los de alarmas y eventos comparten esta interfaz. Para más información ver la especificación OPC *Common* publicada en (OPC Foundation 2007).

IOPCItemIO: El propósito de esta interfaz es proporcionar una manera fácil para que las aplicaciones simples adquieran los datos de los dispositivos. Proporciona un método para leer y escribir los ítems en el servidor. Es válido comentar que el método de intercambio de datos por medio de esta interfaz, es menos eficiente que utilizar los objetos grupos, sin embargo es extremadamente más fácil de implementar. Los métodos que comparte son:

- **Read:** Lee uno o varios ítems retornando las propiedades *Value*, *Quality* y *Timestamp* de cada uno de ellos.
- **WriteVQT:** Escribe hacia uno o varios ítems las propiedades *Value*, *Quality* y *Timestamp*.

IConnectionPointContainer: Esta interfaz es utilizada en el estándar de COM por objetos que requieran recibir eventos, manipulando las llamadas *Callbacks*, de esta forma el servidor puede notificar a los clientes la ocurrencia de un evento. En este caso es utilizada para manipular los puntos de conexión para la interfaz *IOPCShutDown*. Las funciones que comparte son:

- **EnumConnectionPoints:** Crea un enumerador para los puntos de conexión soportados entre el servidor a través de la interfaz *IOPCShutDown* y el cliente.
- **FindConnectionPoint:** Encuentra un punto de conexión particular entre el servidor y el cliente.

1.4.2.2. Interfaces del Objeto Grupo

La Figura 1.5 muestra el grupo como un objeto COM con sus interfaces.

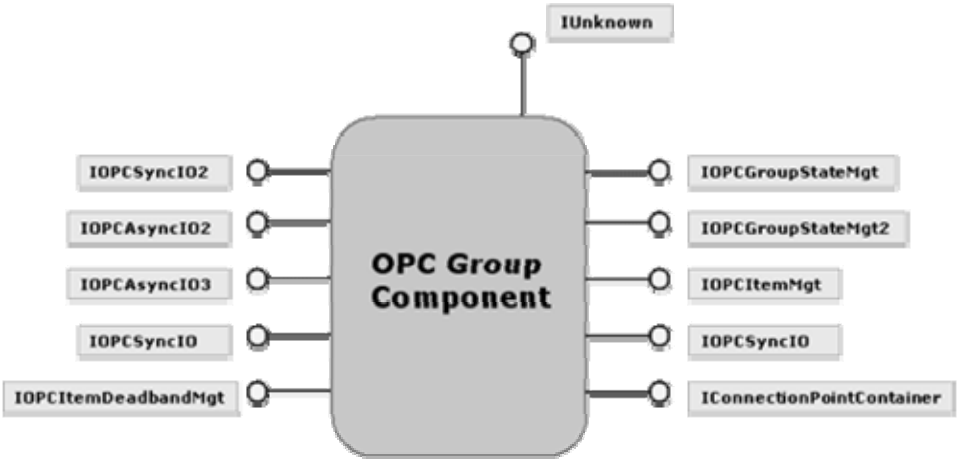

Figura 1.5. Objeto Grupo y sus interfaces.

IOPCGroupStateMgt: Permite al cliente administrar el estado del grupo. Fundamentalmente se usa para activar o desactivar los grupos y cambiarles su tiempo de muestreo.

- **GetState:** Toma el estado actual del grupo, si está activo, tiempo de actualización, etc.
- **SetState:** El cliente puede cambiar el estado de las propiedades de un grupo. Llámense activarlo y tiempo de actualización.
- **SetName**: Cambia el nombre del grupo.
- **CloneGroup:** Crea una copia del grupo. Todas las propiedades del grupo y los ítems son duplicadas. Se pueden adicionar y borrar ítems del nuevo grupo sin afectar el otro.

IOPCGroupStateMgt2: Esta interfaz fue adicionada para enriquecer la *IOPCGroupStateMgt*. La *IOPCGroupStateMgt2* hereda de *IOPCGroupStateMgt*, por lo que todos los métodos definidos en esta última son parte de la nueva. Esta interfaz debe ser implementada por los servidores de OPC–DA 3.0 en lugar de IOPCGroupStateMgt y su propósito es editar el tiempo de validez de una suscripción. Cuando una suscripción tiene un tiempo de validez diferente de cero, el servidor se asegurará de que el cliente reciba la notificación aun cuando no pueda ofrecerle el dato al cliente. El servidor comunicará al

cliente por dos razones, para entregarle el dato, o en caso de que se cumpla el tiempo de validez y el servidor aun no posea el dato. Los métodos que comparte esta interfaz son:

- **SetKeepAlive:** Los clientes pueden cambiar el valor del tiempo de validez de la suscripción para que el servidor provea al cliente de las notificaciones aun en caso de que el servidor no posea datos.
- **GetKeepAlive:** Retorna el tiempo de validez actual para la suscripción.

IOPCItemMgt: Le permite a un cliente agregar, quitar y controlar el comportamiento de los ítems en un grupo.

- **AddItems:** Agrega uno o más ítems a un grupo, inicializándoles sus propiedades, como son ItemID y otras.
- **ValidateItems:** Determina si un ítem es válido, esto es si se pudo adicionar sin errores. También retorna la información de los ítems como el tipo de dato por defecto que tiene el servidor.
- **RemoveItems:** Borra un ítem específico de un grupo.
- **SetActiveState:** Pone uno o más ítems en un grupo a activo o inactivo. Esto controla si los datos válidos pueden ser obtenidos o no desde la *cache* de lectura para esos ítems, y si son incluidos o no en la suscripción de cambios definida por *OnDataChange* del grupo.
- **SetClientHandles:** Cambia el identificador del cliente para uno o más ítems en el grupo.
- **SetDatatypes:** Cambia el tipo de dato encuestado por el cliente para uno o más ítems en un grupo.
- **CreateEnumerator:** Crea un enumerador para un ítem en un grupo.

IOPCSyncIO: Permite a los clientes ejecutar operaciones de lecturas y escrituras sincrónicas al servidor.

- **Read:** Esta función lee las propiedades *Value*, *Quality* y *Timestamp* de uno o más ítems en un grupo. Los datos pueden ser leídos desde la *cache* del servidor o pueden realizarse las lecturas y escrituras directamente al dispositivo.

- **Write:** Escribe el valor de uno o más ítems en un grupo. La función no debe retornar hasta que verifique si el dispositivo realmente ha aceptado o rechazado las escrituras.

IOPCSyncIO2: Esta interfaz fue agregada para reforzar la interfaz *IOPCSyncIO* existente. La interfaz *IOPCSyncIO2* hereda de *IOPCSyncIO* y por consiguiente todos los métodos definidos en la anterior también son parte de ella y no se documentarán aquí. El propósito de esta interfaz es proveer métodos para la escritura de las propiedades *Value*, *TimeStamp* y *Quality*. Esta interfaz se diferencia de la *OPCItemIO*, en que es orientada a los objetos grupos y no al objeto servidor. Además provee la posibilidad de especificar si las lecturas y escrituras se realizan desde la *cache* o del dispositivo.

- **ReadMaxAge:** Lee las propiedades *Value, Quality* y *Timestamp*, de los ítems especificados. Esto es funcionalmente similar al método *Read* de la interfaz *OPCSyncIO*, excepto que se especifica la fuente de los datos (dispositivo o *cache*). La decisión se basará en el parámetro *MaxAge*.
- **WriteVQT:** Escribe las propiedades *Value, Quality* y *Timestamp*, a los ítems especificados. Esto es funcionalmente similar al método *Write* de la interfaz *IOPCSyncIO*, sólo que aquí también pueden escribirse las propiedades *Quality* y *Timestamp*.

IOPCASyncIO2: Permite a un cliente operaciones de lectura y escritura asíncronas a un servidor. Cada operación se trata como una transacción y es asociada con un identificador de transacción. Cuando las operaciones se completan, una notificación, *Callback*, se hará a la *IOPCDataCallback* en el cliente. La información en el *Callback* indicará la identificación de la transacción y los resultados de la operación. También lo esperado es que para cualquier transacción a *Read*, *Write* y *Refresh*, todos los resultados pueden ser devueltos en una sola llamada a la función apropiada en *IOPCDataCallback*.

- **Read:** Lee uno o más ítems en un grupo. Los resultados son retornados por medio de la conexión de la interfaz *IOPCDataCallback* del cliente establecida a través del *IConnectionPointContainer* del servidor. Las lecturas son desde el dispositivo y no son afectadas por el estado activo del grupo o ítem.

- **Write:** Escribe uno o más ítems en un grupo. Los resultados son retornados por medio de la conexión de la interfaz *IOPCDataCallback* del cliente establecida a través del *IConnectionPointContainer* del servidor.
- **Refresh2:** Fuerza una notificación de cambio por medio de un *Callback* al *OnDataChange* de la interfaz *IOPCDataCallback*, para todos los ítems activos en el grupo. Los ítems inactivos no son incluidos en la respuesta del *Callback*.
- **Cancel2:** Demanda la cancelación por parte del servidor de una transacción.
- **SetEnable:** Controla la operación de notificación de cambios *OnDataChange*, habilitándola o deshabilitándola.
- **GetEnable:** Devuelve el estado de las notificaciones, Callback, anteriormente cambiadas por *SetEnable*.

IOPCASyncIO3: Esta interfaz fue agregada para reforzar la interfaz *IOPCAsyncIO2* existente y hereda todos los métodos de esta última. Los servidores de OPC-DA 3.0 deben implementar esta interfaz en lugar de la *IOPCAsyncIO2*. Además los métodos proveen la escritura asincrónica de las propiedades *TimeStamp* y *Quality* teniendo en cuenta si se interactúa con la *cache* o con el dispositivo directamente.

- **ReadMaxAge:** Lee las propiedades *Value, Quality* y *Timestamp*, de los ítems especificados. Esto es funcionalmente similar al método *Read* de la interfaz *OPCSyncIO*, excepto que es asincrónica y se especifica si la fuente de datos es el dispositivo o la *cache* del servidor, esta decisión se basará en el parámetro *MaxAge*.
- **WriteVQT:** Escribe las propiedades *Value*, *Quality* y *TimeStamp*, a los ítems especificados. Los resultados son retornados por medio de la conexión de la interfaz *IOPCDataCallback* del cliente establecida a través del *IConnectionPointContainer* del servidor.
- **RefreshMaxAge:** Fuerza una notificación de cambios, *Callback*, a *OnDataChange* de la interfaz *IOPCDataCallback*, para todos los ítems activos en el grupo. Los ítems inactivos no son incluidos en el *Callback*. El valor de *MaxAge* determinará de donde se obtuvieron los datos. Esto significa que algunos de los valores pueden obtenerse de la *cache* mientras otros pudieran obtenerse del dispositivo.

- **IOPCItemDeadBandMgt:** Esta interfaz permite manipular la propiedad *PercentDeadband* para cada ítem dentro de un grupo. Una vez que el *PercentDeadband* del ítem es fijado, este sobrescribe el *PercentDeadband* del grupo entero. Esto proporciona un mecanismo para poner esta propiedad a un ítem "ruidoso" que puede residir en un grupo que no tiene el *PercentDeadband* del grupo fijado.
- **SetItemDeadband:** Sobrescribe la banda muerta especificada en el grupo para cada ítem.
- **GetItemDeadband:** Toma el valor de la banda muerta para cada ítem encuestado.
- **ClearItemDeadband:** Limpia la banda muerta de un ítem individual y le vuelve a colocar el valor de la zona muerta del grupo.

IOPCItemSamplingMgt: Interfaz opcional que permite manipular la velocidad a la que se obtienen ítems individuales desde el dispositivo dentro de un grupo. Este no afecta el *UpdateRate* del grupo en un *Callback*.

- **SetItemSamplingRate:** Pone el tiempo de muestreo en ítems individuales. Esto sobrescribe el *UpdateRate* del grupo hasta lo que concierne a la colección del dispositivo subyacente. El *UpdateRate* asociado con ítems individuales no afecta el período del *Callback*.
- **GetItemSamplingRate:** Consigue el tiempo de muestreo de ítems individuales que fue previamente fijado con *SetItemSamplingRate*.
- **ClearItemSamplingRate:** Limpia el tiempo de muestreo de ítems individuales, previamente fijado con *SetItemSamplingRate*. El ítem tomará de nuevo el del grupo.
- **SetItemBufferEnable:** Solicita que el servidor active o no el buffer, el buffer de datos es para los ítems que tienen una velocidad de muestreo más rápida que la del grupo a la que pertenecen.
- **GetItemBufferEnable:** Retorna el estado actual del buffer de los servidores para los ítems pedidos.

1.4.2.3. Interfaces para aplicaciones del cliente de OPC-DA

Las aplicaciones clientes de OPC-DA no requieren implementar todas las interfaces expuestas en los epígrafes anteriores, todo lo que requieran del servidor deben solicitarlo a la interfaz correspondiente. Sin embargo requieren implementar una serie de interfaces que son la base del mecanismo de suscripción y notificación de eventos que sucedan en el servidor y que son de gran importancia para los clientes.

Un ejemplo de lo anterior son las actualizaciones de los valores de los ítems, estas operaciones deben ser notificadas por el servidor hacia todos lo clientes interesados en el ítem modificado.

Las aplicaciones clientes de OPC-DA, básicamente deben implementar dos interfaces de eventos. La *IOPCShutDown*, especializada en las notificaciones de desconexión del servidor y la interfaz *IOPCDataCallback*, especializada en las notificaciones de operaciones de lectura y escritura de ítems (Suchit 2005).

A continuación se describen los métodos de estas interfaces.

IOPCDataCallback: Una aplicación cliente debe soportar esta interfaz para que el servidor de OPC-DA notifique los cambios que existen en los ítems que el cliente esté interesado. El mecanismo para soportar estas notificaciones está basado en los puntos de conexión que existen en el servidor, los cuales se obtienen a través de la interfaz *IConnectionPointContainer* por medio de los métodos *FindConnectionPoint* o *EnumConnectionPoints*. El servidor obtendrá el puntero a la interfaz *IOPCDataCallback* del cliente que a su vez habrá implementado los métodos que se invocarán cuando ocurra el evento. Los eventos que debe implementar el cliente son:

- **OnDataChange:** Este método es proporcionado por el cliente para ocuparse de notificaciones de un grupo al ocurrir cambios en los datos y refrescamientos.
- **OnReadComplete:** Este método es proporcionado por el cliente para ocuparse de notificaciones de un grupo cuando se efectúan lecturas asincrónicas.
- **OnWriteComplete:** Este método es proporcionado por el cliente para ocuparse de notificaciones de un grupo cuando se completan escrituras usando la interfaz *IAsyncIO2*.

- **OnCancelComplete:** Este método es proporcionado por el cliente para ocuparse de notificaciones de un grupo cuando se completan cancelaciones asincrónicas.

IOPCShutdown: Una aplicación cliente debe soportar esta interfaz para que el servidor de OPC-DA notifique las desconexiones. El mecanismo para soportar estas notificaciones está basado en los puntos de conexión que existen en el servidor, los que se obtienen a través de la interfaz *IConnectionPointContainer* por medio de los métodos *FindConnectionPoint* o *EnumConnectionPoints*. El servidor obtendrá el puntero a la interfaz *IOPCShutDown* del cliente que a su vez habrá implementado un método que se invocará cuando ocurra el evento. Un cliente que posea conexiones a varios servidores debe poseer una instancia de los métodos separada para cada servidor.

Los clientes deben implementar el siguiente método.

- **ShutdownRequest:** Este método es proporcionado por el cliente para que el servidor le notifique una desconexión. El cliente debe deshabilitar todas las conexiones, quitar todos los grupos y liberar todas las interfaces.

1.4.3. Informaciones sobre el desarrollo de las especificaciones de OPC-DA

La especificación OPC-DA, es la más utilizada en los sistemas automatizados actuales por su versatilidad, permitiendo muestreos en tiempo real, base de todos los controles de procesos industriales.

Desde su creación en 1996, los estándares de OPC han evolucionado en gran medida y esta especificación ha sido la más actualizada de todas. La descripción del estándar OPC-DA tratada en este capítulo se realizó basándose en la versión 3.0. Esta versión tiene grandes cambios con respecto a su antecesora la versión 2.05, a continuación resumiremos esta evolución.

Se eliminaron interfaces para el manejo de grupos, ítems y para la exploración de los datos publicados por el servidor, dentro de ellas se pueden mencionar.

- **IOPCServerPublicGroups:** Dedicado al manejo de grupos públicos.
- **IOPCBrowseServerAddressSpace:** Para la exploración de los datos que comparte el servidor, esta interfaz fue sustituida por *IOPCBrowse*.
- **IOPCPublicGroupStateMgt:** Manejo de los estados de los grupos de ítems.

- **IOPCAsyncIO:** Utilizada para las escrituras y lecturas asincrónicas, esta interfaz fue sustituida por *IOPCAsyncIO3*.
- **IOPCItemProperties:** Edición de las propiedades de los ítem publicados en el servidor.

Fueron adicionadas las siguientes interfaces, descritas en los epígrafes anteriores

- **IOPCBrowse.**
- **IOPCItemDeadbandMgt.**
- **IOPCItemSamplingMgt.**
- **IOPCItemIO.**
- **IOPCSyncIO2.**
- **IOPCAsyncIO3.**
- **IOPCGroupStateMgt2.**

1.4.4. Estructura de un Servidor de OPC-DA

La Figura 1.6 muestra los componentes más comunes de un servidor de OPC-DA. En el nivel más bajo se encuentran dos capas que pueden formar parte del *driver* que intercambiará información con el dispositivo de campo, típicamente un PLC.

En el nivel medio del servidor se encuentran los bloques de manipulación y optimización de la colección de datos. Estas tareas se logran manteniendo una correcta latencia de la comunicación con el dispositivo y una correcta manipulación de las encuestas de los clientes.

Utilizando las interfaces de OPC-DA, se pueden compartir los datos del dispositivo hacia los clientes OPC. Estas interfaces y su interrelación con los clientes de OPC forman parte de las capas más altas del servidor.

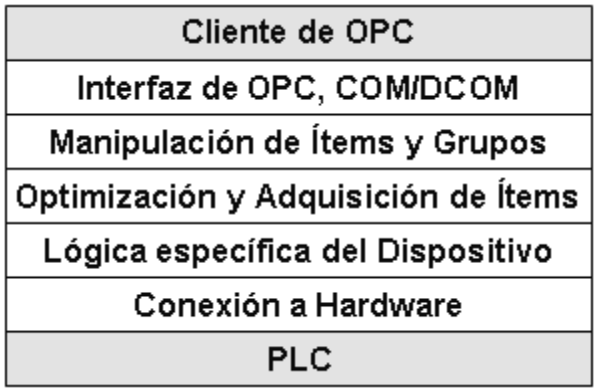

Figura 1.6. Estructura de un Servidor OPC-DA.

1.5. El futuro de OPC

OPC UA permite integrar los servicios que anteriormente se encontraban separados, además provee numerosos avances sobre las características de los estándares de OPC que le precedieron (Burke 2007). Dentro de ellas podemos mencionar:

- **Manejo de Datos Complejos.** OPC UA permite manejo de datos sencillos y complejos de forma nativa, datos OPC estándares y personalizados.
- **Espacios de nombres avanzados.** OPC-UA permite tipos de nodos y relaciones ilimitados. Cada nodo puede participar en un número ilimitado de relaciones con otros nodos. Esto permite que no exista en el futuro un sistema que sea demasiado complicado para ser modelado vía OPC-UA.
- **Conjuntos de Servicios Básicos Avanzado.** La especificación base UA contiene los servicios genéricos necesarios para realizar navegación y búsqueda en los espacios de nombres, escritura y lectura de datos y publicación suscripción de eventos y cambios de datos.
- **Especificaciones UA derivadas.** La especificación base de UA, es genérica y no implica una semántica avanzada. Esto permite que especificaciones funcionales avanzadas sean derivadas de la especificación base.
- **Escalabilidad.** Comparada con otras especificaciones de OPC, UA podrá ser escalable desde dispositivos empotrados hasta grandes modelos de negocios.
- **Fiabilidad y Redundancia.** OPC UA permite el diagnóstico y características informativas que permitan realizar aplicaciones fiables. Esto incluye la redundancia

de los sistemas, resincronía, *keep-alives*. Se permiten tanto clientes como servidores redundantes.

- **Seguridad.** OPC UA incluye la especificación de seguridad integrada, superando a su predecesor OPC XML DA.
- **Comandos.** OPC UA incluye la especificación de manejo de comandos.

Dada la veloz evolución y diversidad que han mostrado las tecnologías de comunicación, la Fundación OPC previó que la especificación OPC UA fuese escrita para ser independiente de la plataforma, con vistas a que pueda ser migrada a cualquiera deseable. Esta especificación aun no ha sido del todo liberada.

1.6. Conclusiones parciales

Para resolver la incompatibilidad entre OPC y Linux debido a la alta dependencia de este estándar con la plataforma Windows, se propone la creación de un *Gateway* sobre plataformas o emuladores de Windows, capaz de comunicarse con los servidores de OPC, y que a su vez le brinde los datos al SCADA empleando una técnica de comunicación multiplataforma que cumpla los requerimientos del proyecto. Se propone además la creación de un *Driver* que le permita al SCADA el intercambio de datos con dicho *Gateway* (Ver Figura 1.7).

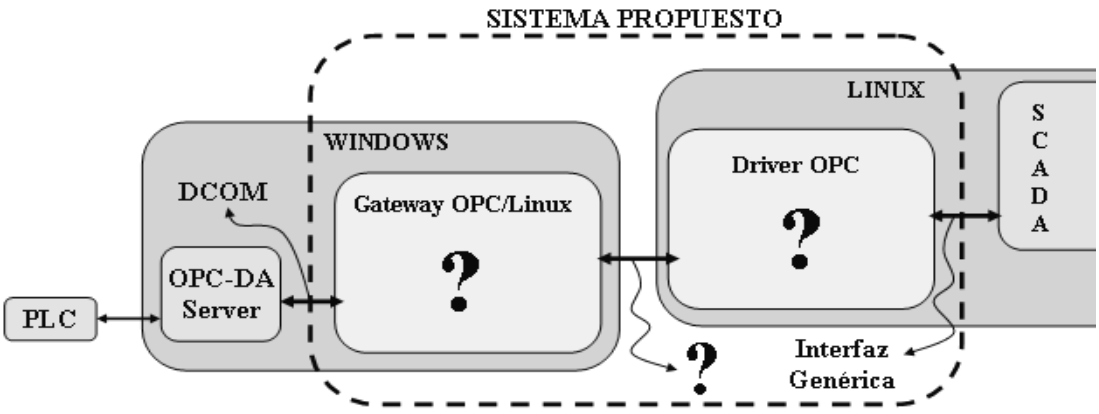

Figura 1.7. Sistema propuesto para la integración de OPC al SCADA.

En el siguiente capítulo se hará un análisis detallado de las tecnologías de comunicación con el fin de elegir la más apropiada para realizar el intercambio de información entre *Gateway* y *Driver*.

CAPÍTULO 2. SISTEMAS DISTRIBUIDOS. EVALUACIÓN Y SELECCIÓN DE TECNOLOGÍAS

2.1. Sistemas Distribuidos

El elevado desarrollo de la electrónica digital, las estaciones personales y las redes de computadoras a mediados y finales de los años ochenta, impulsó el vertiginoso ascenso de las técnicas computacionales. Las necesidades de intercambio y reparto del volumen de información, así como de compartir los recursos tanto de software como de hardware a través de la red, y de reutilización en sentido general, constituyeron algunos de los elementos propiciadores del nacimiento de los sistemas distribuidos.

2.1.1. Definición

Por sistema distribuido se ha entendido:

- "Un sistema en el que los componentes hardware y/o software ubicados en computadores en red, se comunican y coordinan sus acciones intercambiando mensajes."
- "Colección de ordenadores autónomos enlazados por una red y soportados por aplicaciones que hacen que la colección actúe como un servicio integrado" (Catalunya 2007).
- "..sistema de computación con un número de componentes que cooperan entre sí comunicándose a través de la red" (Catalunya 2007).

En resumen, los sistemas distribuidos pueden definirse como aplicaciones con una fuerte componente geográfica, pues su ejecución no ocurre en una única estación de trabajo como era el caso de los antiguos sistemas monolíticos y de otras aplicaciones autónomas. Sus distintas partes o componentes pueden encontrarse bien distantes ejecutándose en diferentes computadoras interconectadas en red. Sus procesos tienen lugar en más de un ordenador,

junto a la ocurrencia de un constante intercambio de información como requisito indispensable para su funcionamiento satisfactorio mediante el paso de mensajes.

2.1.2. Características

Hay autores que clasifican a los sistemas distribuidos como la máxima expresión de las aplicaciones multicapas (N-*tier*), o que repiten un *slogan* difundido sobre la computación distribuida: "la red es la computadora" (Cetus Team 2002).

La ventaja de las aplicaciones N-*tier* radica en la posibilidad de dividir la lógica del sistema en componentes modulares y reusables en lugar de obtener un único código monolítico. Al distribuir el procesamiento cada nodo del sistema incluirá un menor número de complejidades y los requerimientos de *hardware* podrán ser complacidos con mayor facilidad. Las distintas partes en comunicación tendrán la posibilidad de colaborar a pesar de la distancia física y responder como si se tratara de una misma entidad.

Entre los grandes exponentes de los sistemas distribuidos actuales se encuentra la "*World Wide Web*" como el mayor y el más popular.

Una importante característica a tener en cuenta durante la construcción de la mayoría de las aplicaciones distribuidas es la heterogeneidad del software con el que interactúan (plataformas). Dicha característica es además, uno de los factores condicionantes del siguiente planteamiento realizado por el Dr. Schmidt en (Schmidt 2006a):

"El desarrollo de software de alta calidad para sistemas distribuidos posee una elevada dificultad; el logro de un desarrollo exitoso y de la correcta aplicación de los principios para obtener una alta calidad y reusabilidad, posee incluso dificultades mayores."

Para la concepción de un sistema distribuido, existen dos modelos que serán analizados a continuación.

2.1.3. Modelo de igual a igual *(peer-to-peer)*

Este modelo basa su funcionamiento en un sistema sin jerarquías donde todas las máquinas son a la vez clientes y servidores, por ello se le llama intercambio entre iguales (Ver Figura 2.1). Su implementación masiva se ha dado sobre redes IP, constituyéndose en redes virtuales que funcionan sobre la infraestructura técnica de internet, pero singularizadas

porque la búsqueda de la información no se realiza en función de la ubicación del recurso sino de su descripción (Subías 2003).

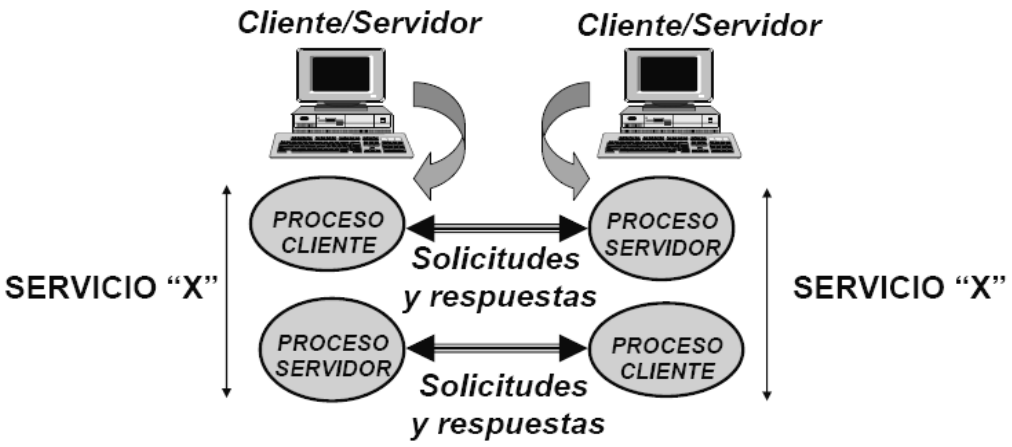

Figura 2.1. Representación del modelo *peer-to-peer*.

La conexión masiva de estas redes virtuales en internet puede permitir el cumplimiento de la promesa del "Gran Almacén Universal Virtual", en el que se encuentra todo el catálogo de productos mediáticos dispuestos para la libre elección del público. El conocimiento de los usuarios de esta red electrónica se retroalimenta gracias a la circulación de la información entre los diferentes nodos del sistema (Prado 1997).

A través de los sistemas *peer-to-peer*, un significativo grupo de usuarios que hacían un uso de la red orientado casi exclusivamente hacia el consumo de los recursos disponibles, se convierte en creador, o como mínimo, distribuidor de contenidos, lo que provoca un aumento del valor de uso de las redes.

En resumen, las principales características de este modelo son:

- Comunicación entre sistemas finales iguales, ningún sistema final tiene mayor prioridad que otro.
- No existen clientes y servidores fijos, cada proceso participante es cliente (solicitudes) y servidor (respuestas), y a menudo, simultáneamente.

2.1.4. Modelo cliente-servidor

Esta arquitectura consiste básicamente en que un programa, el cliente, realiza peticiones a otro programa, el servidor, que les da respuesta (Ver Figura 2.2). Aunque esta idea se puede

aplicar a programas que se ejecutan sobre una sola computadora, es más ventajosa en un sistema multiusuario distribuido a través de una red.

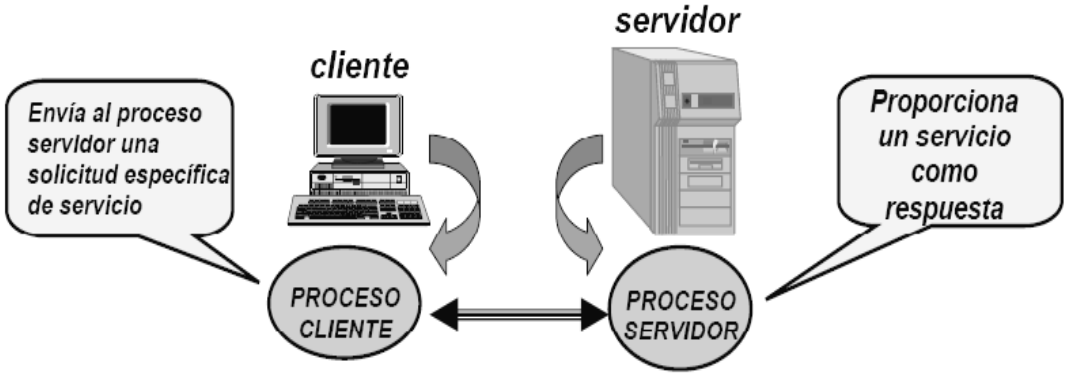

Figura 2.2. Representación del modelo cliente-servidor.

En esta arquitectura la capacidad de proceso está repartida entre clientes y servidores, aunque son más importantes las ventajas de tipo organizativo, debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

La separación entre cliente y servidor es de tipo lógica, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un solo programa. Una disposición muy común son los sistemas multicapa, en los que el servidor se descompone en diferentes programas que pueden ser ejecutados por diferentes computadoras, aumentando así el grado de distribución del sistema (Wikipedia 2007).

Esta arquitectura presenta dos ventajas fundamentales:

- ***Centralización del control*:** los accesos, recursos y la integridad de los datos son controlados por el servidor, de forma que un programa cliente defectuoso, o no autorizado, no pueda dañar el sistema.
- ***Escalabilidad*:** se puede aumentar la capacidad de clientes y servidores por separado.

En la actualidad este es el modelo predominante, cuya aplicación se puede ver a diario. Visitar un sitio en internet es un buen ejemplo de ello, en el cual un *Web Server* sirve las páginas a un *Web Browser*. La mayoría de los servicios de internet son de este tipo, algunos ejemplos son servidores de archivo, de impresora, de DNS, etc.

2.1.5. Selección del modelo apropiado

En nuestro caso, no es posible la utilización del modelo *peer-to-peer*, puesto que existe diferencia entre los sistemas terminales. El SCADA necesita acceder a los datos disponibles en el servidor de OPC, y en ningún caso este se servirá del SCADA, ya que la fuente de sus datos es solamente la red de dispositivos de campo.

2.2. Tecnologías de comunicación en Sistemas Distribuidos

Existen variadas tecnologías para la implementación de un software de comunicación. Entre las principales en el presente, por sus potencialidades y nivel de utilización global se encuentran: DCOM y .Net de Microsoft, Java/RMI de Sun Microsystem y CORBA y DDS del OMG. Se hace necesario decidir cuál es la idónea según los requerimientos del SCADA de PDVSA, y a continuación se presenta una descripción con los rasgos significativos de estas filosofías.

2.2.1. DCOM (*Distributed Component Object Model*)

DCOM surgió a partir de la tecnología OLE (*Object Linking and Embedding*) de Microsoft, la cual vio la luz en el año 1990 con el fin de hacer la funcionalidad de "cortar y pegar" en los entornos Windows. Después se extendió a OLE2, a la cual se le cambió el 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 máquinas comunicadas en red, surgió DCOM. Esta es una tecnología que se ha ido desarrollando y actualmente forma parte esencial de *ActiveX*, *Microsoft Transaction Server*, COM+ y OPC (Horstmann 1997).

Características propias de DCOM

- Transparencia en la localización y la arquitectura.
- Modelo de hilos libres.
- Múltiples niveles de seguridad.
- Referencia de presencia.
- Administración.
- DCOM no sólo brinda todos los componentes básicos necesarios para el diseño y el desarrollo de sistemas sofisticados, también brinda escalabilidad y robustez.

Ventajas de DCOM

- Amplia utilización por los usuarios de componentes registrados.
- Integración de software binario.
- Reutilización del software a gran escala.
- Reutilización del software a través de los lenguajes.
- Actualización de software en línea.
- Permite actualizar un componente en una aplicación sin recompilación.
- Múltiples interfaces por objeto.
- Selección de las herramientas de programación habilitadas, muchas de las cuales brindan automatización del código estándar.

Desventajas de DCOM

- Aunque su especificación ha sido liberada las implementaciones más eficientes son cerradas.
- Aunque posteriormente fue portado a otras plataformas su referencia por excelencia es Windows, las implementaciones de Linux son deficientes.
- Programar su seguridad es complejo a pesar de poseer niveles bien definidos en la especificación, frecuentemente resulta poco seguro sobre todo en entornos abiertos.
- COM no fue concebido para sistemas distribuidos y por tanto las tecnologías basadas en él siempre se montan sobre una base no dispuesta para estos fines.
- Microsoft sugiere la utilización de nuevas tecnologías para la sustitución de COM y DCOM dentro de las que se puede mencionar COM+ y .NET.

2.2.2. COM+ y .NET

El estándar .NET proporciona abstracciones de los detalles de conexión entre los componentes, posibilitando al desarrollador centrarse en la elaboración utilizando cualquier lenguaje de la lógica de su negocio, en lugar de invertir tiempo y esfuerzo con las especificidades de la infraestructura COM.

Entre sus fines principales está el facilitar el desarrollo de componentes y con tal objetivo automatiza las complejas tareas internas que tradicionalmente están relacionadas con el

desarrollo COM, incluyendo el conteo de referencias, descripción de interfaces y registro (Microsoft Corp. 2005).

COM+ se refiere a la versión de COM que combina las características basadas en servicios de *Microsoft Transaction Server* (MTS), con COM distribuido (DCOM), proporcionando un conjunto de servicios orientados a la capa intermedia. En particular, COM+ proporciona administración de procesos, servicios de sincronización, un modelo de transacción declarativo y agrupación de conexiones a objetos y bases de datos. De forma general se enfoca en brindar confiabilidad y escalabilidad para aplicaciones distribuidas a gran escala. Estos servicios son complementarios a los servicios de programación proporcionados por el Entorno .NET; y la BCL *(Base Class Library)* (Microsoft Corp. 2004).

Se han encontrado en Internet únicamente dos implementaciones de .Net como software libre: MonoDevelop y Portable.Net, pero ninguna de las dos brinda servicios para el intercambio de información en tiempo real. Ambas implementaciones están presentando, además, conflictos con sus respectivas licencias (Avery 2007) (Bollow 2007).

Características de .NET

- Utiliza a COM+ como modelo de componentes distribuidos.
- Posibilita programar componentes en múltiples lenguajes.
- Brinda transacciones, puesta en cola *(queuing)* y agrupación *(pooling)* de objetos.
- Posee un entorno de programación sencillo.

Ventajas

- Abstracción del lenguaje de programación de los componentes.
- Administración automática de memoria.
- Avanzada arquitectura de acceso a datos.
- Condiciona alta productividad del desarrollador.

Desventajas

- Las dos implementaciones libres disponibles poseen actualmente conflictos de licencia. No brindan servicios para facilitar la comunicación entre procesos en tiempo real.

2.2.3. RMI *(Remote Method Invocation)*

RMI es el mecanismo ofrecido en Java que permite a un procedimiento (método, clase, aplicación) poder ser invocado remotamente, o sea que este puede existir en cualquier espacio de direcciones de la red incluso en la misma computadora en la que se hace la invocación. RMI es básicamente un mecanismo RPC *(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 *stream* (conjunto de datos). Desde que *Java Object Serialization* es específico del entorno, tanto el objeto servidor como el objeto cliente tienen que ser escritos en Java. Cada objeto servidor define una interfaz, la cual puede ser usada para acceder a este objeto desde la misma máquina virtual o desde otra en cualquier lugar de la red. La interfaz brinda un grupo de métodos, los cuales indican los servicios ofrecidos por el objeto servidor. Para que un cliente pueda localizar un objeto servidor, RMI depende de un servicio nombrado RMI-*Registry*, que corre en el servidor y da información acerca de los objetos que se encuentran en el mismo. 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 (Sun Microsystems 2004).

Características propias de Java/RMI

- Transparencia.
- Eficiencia.
- Seguridad.
- Funcionalidad.

Ventajas de RMI

- Soporta invocaciones remotas de objetos en diferentes máquinas virtuales.
- Soporta llamadas desde los servidores a los *applets*.

- Integra el modelo de objetos distribuidos dentro del lenguaje Java en un modo natural.
- Preserva la seguridad brindada por el ambiente de trabajo de Java en tiempo de ejecución.
- Hace escrituras confiables de aplicaciones distribuidas tan sencillas como es posible.
- Lenguaje simple.
- Libre.

Desventajas de RMI

- Su empleo más óptimo es utilizando lenguaje Java para la implementación. Existen mecanismos para adaptarlo a otros lenguajes, pero el proceso se hace tedioso, propenso a fallas y sus resultados no son tan eficientes.
- Los objetos a ser exportados tienen que heredar obligatoriamente de un objeto base dado por la API. En general la implementación de RMI tiende a ser bastante invasiva.
- Depende de una máquina virtual Java para su ejecución.
- Al utilizar Java, la ejecución es considerablemente lenta, comparada con implementaciones en lenguajes compilados como C o C++.

2.2.4. CORBA *(Common Object Request Broker Architecture)*

La Arquitectura Común para la Mediación de Solicitudes entre Objetos, es actualmente una de las opciones tecnológicas más importantes a la hora del desarrollo de sistemas software distribuidos. CORBA es un estándar que establece una plataforma de desarrollo de sistemas distribuidos, facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos. El ORB *(Object Request Broker)* constituye el núcleo de la tecnología.

CORBA fue definido y está controlado por el *Object Management Group* (OMG) (Sitio Oficial: http://www.omg.org) que define las APIs, el protocolo de comunicaciones y los mecanismos necesarios para permitir la interoperabilidad entre diferentes aplicaciones escritas en diferentes lenguajes y ejecutadas en diferentes plataformas, lo que es fundamental en computación distribuida.

En un sentido general CORBA "envuelve" el código escrito en un lenguaje en un paquete que contiene información adicional sobre las capacidades del código, y sobre cómo llamar a sus métodos. Los objetos que resultan pueden entonces ser invocados desde otro programa (u objeto CORBA) en la red. En este sentido CORBA se puede considerar como un formato de documentación legible por la máquina, similar a un archivo de cabeceras pero con más información.

CORBA utiliza un lenguaje de definición de interfaces (IDL) para especificar las interfaces con los servicios que los objetos ofrecerán (Ver Anexo 2). CORBA puede especificar a partir de este IDL la interfaz a un lenguaje determinado, describiendo cómo los tipos de dato deben ser utilizados en las implementaciones del cliente y del servidor. Implementaciones estándar existen para Ada, C, C++, Smalltalk, Java, Python, Perl y TCL. Al compilar una interfaz en IDL se genera código para el cliente y el servidor (el implementador del objeto). El código del cliente sirve para poder realizar las llamadas a métodos remotos. Es el conocido como *stub* (cabo), el cual incluye un *proxy* (representante) del objeto remoto en el lado del cliente. El código generado para el servidor consiste en unos *skeletons* (esqueletos) que el desarrollador tiene que rellenar para implementar los métodos del objeto. CORBA es más que una especificación multiplataforma, también define servicios habitualmente necesarios como seguridad y transacciones (Castillo 2006).

CORBA es una tecnología basada en la clásica arquitectura encuesta/respuesta. Existe una implementación de objetos en el servidor, al cual el cliente encuesta para ejecutar. Las implementaciones de los objetos cliente y servidor no tienen ninguna restricción de espacio de direcciones, el cliente y el servidor pueden o no existir en un espacio común.

Servicios CORBA

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

Los servicios de CORBA representan un conjunto de funcionalidades que complementan el desarrollo funcional básico de los objetos que dan lugar a una aplicación. El número de servicios se amplía continuamente para añadir nuevas facilidades a los sistemas

desarrollados con CORBA. Esto no quiere decir que se encuentren implementados en los ORBs comerciales.

El estándar CORBA 2.0 del OMG ha definido dieciséis de estos servicios (OMG 2007a):

- **Ciclo de Vida:** define operaciones para crear, copiar, mover y eliminar objetos.
- **Eventos:** permite registrarse para recibir eventos que pueden ser producidos por otros objetos bajo un modelo *consumer/supplier*.
- **Nombre:** permite localizar objetos por un nombre.
- **Registro:** permite localizar objetos por sus propiedades.
- **Persistencia:** ofrece una interfaz para almacenar objetos.
- **Transacciones:** proporciona coordinación transaccional.
- **Concurrencia:** proporciona un gestor de bloqueos.
- **Externalización:** permite obtener y producir datos como *streams* (flujos).
- **Seguridad:** da soporte a la autenticación, listas de control de acceso y confidencialidad.
- **Tiempo:** permite definir y gestionar eventos temporizados.
- **Propiedades:** permite asociar propiedades a un objeto.
- **Encuesta:** ofrece una interfaz SQL para realizar operaciones en objetos.
- **Licencia:** ayuda a medir el uso de los objetos.
- **Relaciones:** permite crear asociaciones dinámicas entre objetos.
- **Colección:** proporciona las interfaces para crear y manipular colecciones de objetos (listas, conjuntos...).

Ventajas de CORBA

- Interfaz independiente del lenguaje de programación.
- Integración de herencia.
- Infraestructura de objetos distribuidos.
- Transparencia en la localización.
- Transparencia en la red.
- Comunicación directamente con los objetos.
- Interfaz de invocación dinámica.

Desventajas de CORBA

- Complejidad.
- Incompatibilidad entre implementaciones.

2.2.5. DDS (*Data Distribution Service*)

DDS es una especificación realizada por RTI (*Real Time Investigations*) y aprobada por el OMG, bajo el paradigma publicación/suscripción. La especificación de DDS estandariza la API, mediante la cual una aplicación distribuida puede utilizar el *Data Centric Public Suscribe* (DCPS) como un mecanismo de comunicación. Desde que DDS está especificado como una solución de infraestructura, este puede ser utilizado como la interfaz de comunicación para cualquier tipo de software.

DCPS es la capa que permite el mecanismo de publicación/suscripción con un alto nivel de interoperabilidad y facilidades a los programadores que hacen uso de este servicio, brindando además gran cantidad de parámetros de calidad del servicio (RTI 2007).

DDS permite implementar soluciones que facilitan la programación distribuida, su modelo publicación/suscripción está especificado para el envío de datos, eventos y comandos a través de los tópicos, desde publicadores a suscriptores.

Ventajas de DDS

- Basado en un simple paradigma de comunicación Publicar-Suscribir.
- Arquitectura flexible y adaptable.
- Baja sobrecarga, por lo cual puede ser usado en sistemas de alto funcionamiento.
- Entrega de datos determinística.
- Escalable dinámicamente.
- Uso eficiente del ancho de banda para el transporte.
- Soporta comunicaciones uno a uno, uno a muchos, muchos a uno y muchos a muchos.
- Gran número de parámetros de configuración.

Desventajas de DDS

- No tiene un mecanismo de seguridad muy eficiente.

- Está en constante desarrollo, las implementaciones existentes aun adolecen de varias características esenciales para el desarrollo de sistemas distribuidos.

2.2.6. Selección de la tecnología apropiada

Hasta este punto se han analizado los rasgos fundamentales de los cinco estándares principales en el desarrollo de software de comunicación.

Debido a características no convenientes, con afectación directa en las especificidades del proyecto SCADA PDVSA, se descarta la utilización de Java/RMI, DCOM y COM+.

Las aplicaciones elaboradas con el middleware Java/RMI deben ser desarrolladas con Java parcial o completamente. Por su parte DCOM y COM+ limitan la plataforma de desarrollo y despliegue al sistema operativo Windows, ya que las implementaciones existentes del *Distributed Object Model* en Linux no poseen la suficiencia necesaria. Tales rasgos van en detrimento de los requerimientos tecnológicos del SCADA expuestos en análisis previos (software libre, estándares abiertos, y C++ como lenguaje de implementación).

Las especificaciones restantes, CORBA y DDS, ofrecen soporte para todos los requerimientos tecnológicos básicos y generales, por lo tanto, se proseguirá con el estudio de las implementaciones más adecuadas de ambos estándares para finalmente conseguir la solución más correcta a utilizar en el desarrollo del software de comunicación del proyecto en interés.

DDS, especificación cuyo diseño se centra en obtener un intercambio eficiente de los datos, pareciera por todas sus características el estándar ideal para la elaboración del software de comunicación. Al efectuar una búsqueda de sus implementaciones libres que posibiliten el desarrollo en C++, solo quedaría por analizar el middleware ACE+TAO+DDS, desarrollado por el grupo DOC de la universidad de California, y mejorado por la compañía OCI (Object Computer, Inc). TAO DDS, como también es dada a conocer la implementación, no posee en la actualidad un grado de madurez suficiente para ser utilizada en la elaboración del software de comunicación para el sistema SCADA PDVSA.

Esta implementación de DDS, aunque brinda alta eficiencia y predictibilidad, permite el control de los servicios a través de un amplio rango de políticas de QoS, entre otros apropiados beneficios, presenta inconvenientes ineludibles. Este *framework* adolece aun de

prestaciones necesarias dada la naturaleza crítica del proceso de intercambio de información que se desea desarrollar. Para la obtención de la aplicación de comunicación son absolutamente necesarias las persistencias de las conexiones y de los datos. La versión actual de ACE+TAO+DDS no propicia el restablecimiento de las conexiones previamente en funcionamiento tras la caída de las comunicaciones, ni el reenvío de los datos tras el conocimiento de su pérdida. Tales características van en total detrimento de su utilización, por lo que no cabe duda que CORBA es el *framework* ideal para la implementación de las aplicaciones del SCADA de PDVSA (Vázquez 2007).

2.3. Implementaciones de las especificaciones de CORBA

La realización de una búsqueda en Internet arrojó la existencia de al menos 24 implementaciones funcionales de CORBA (Ver Anexo 3). De ellas, sólo cuatro satisfacen los requisitos tecnológicos del SCADA. En otras palabras, sólo las implementaciones Orbit, MICO, OmniORB y TAO cumplen a la vez con: clasificar como software libre, soportar la implementación para las plataformas Linux-Windows y posibilitar el desarrollo utilizando el lenguaje C++. A continuación se analizarán con más detalle.

2.3.1. OmniORB

Omniorb fue desarrollado por *Olivetti Research Ltd* (que en 1999 pasó a ser parte de *AT&T Laboratories*) para ser utilizado en pequeños equipos de entorno empotrado que necesitaban de comunicación con ORBs comerciales en ordenadores de sobremesa y servidores. Ha evolucionado con el tiempo a través de 11 versiones públicas para la comunidad CORBA, varias versiones beta y pequeños desarrollos, en la actualidad lo utilizan numerosos desarrolladores de programas en múltiples aplicaciones. Omniorb es un *broker* que implementa la especificación 2.6 de CORBA, es robusto y tiene grandes prestaciones, permite la programación en C++ y *Python* y es libre bajo los términos GNU (*General Public License*). Es uno de los tres ORB al que se le ha concedido el *Open Group's Brand* de CORBA, lo que significa que ha sido probado y certificado para CORBA 2.1 (le faltan todavía características para ajustarse a la norma 2.6 como por ejemplo no se soportan las interfaces locales ni los objetos por valor) (Grisby 2006).

2.3.2. TAO

TAO ORB fue desarrollado por el *Distribuited Object Computing Group* en la Universidad de Washington bajo la dirección del Dr. Douglas Schmidt. La versión 1.0 de TAO es diseñada para compilar con CORBA 2.2 e incluye algunos aspectos de la 2.3. Inicialmente fue pensado para tiempo real, pero ha llegado a ser un ORB de propósito general muy capaz, 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 que implementa. Actualmente TAO compila 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 algunos sentidos. TAO tiene un mecanismo de prioridades basado en la especificación RT-CORBA (*Real Time* CORBA), lo cual le permite su funcionamiento en sistemas de tiempo real con una eficiencia muy alta (Schmidt 2006b).

2.3.3. MICO

Su nombre está conformado por las siglas de la frase en inglés *Mico Is Corba*. 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 ORBs, 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 (MICO Project Team 2006).

2.3.4. ORBit

ORBit es una implementación en C de la versión 2.2 de la especificación CORBA. Tiene enlaces habilitados también para C++, Lisp, Pascal, Python, Ruby, TLC y fue elaborado

para la comunicación en los entornos gráficos del proyecto GNOME. Tiene características inigualables por otros ORBs, como la gran velocidad en el intercambio de datos y el bajo uso de memoria, o sea, cumple a cabalidad la función para la que fue hecho. El núcleo está escrito en C y corre en Linux, Unix y Windows.

Después de ORBit, el proyecto GNOME continuó su desarrollo y salió a la luz una versión aun más acabada cuyo código fue implementado en C++ y compila para este lenguaje, lo cual explaya sus capacidades en el tratamiento de los objetos. Esta nueva versión es conocida como ORBit2 (Lee 1999).

2.3.5. Selección de la implementación apropiada

En primera instancia, es posible prescindir del elevado uso de la memoria y la sobrecarga de MICO por la existencia de otros ORB más eficientes como OmniORB, TAO y Orbit.

Las variadas fuentes consultadas revelan prestaciones similares y convenientes para su utilización en un SCADA en el caso de dichos “brokers” (Vázquez 2007), sobresaliendo Orbit como el más ligero de los tres por sus cortos tiempos de respuesta y baja utilización de memoria. No obstante, TAO, ORB concebido para el desarrollo con requerimientos de tiempo real, demuestra ser el más adecuado debido a la eficiente implementación que a diferencia de sus homólogos realiza del estándar CORBA-RT.

Algunos de los servicios de TAO posibles a utilizar en la solución de los desafíos más importantes para el SCADA PDVSA son: el Servicio de Notificación en Tiempo Real (*Real-Time Notification Service*), el Servicio de Nombres (*Naming Service*) y el Repositorio de implementación (*Implementation Repository*).

El repositorio de implementaciones automáticamente ejecuta a los servidores en respuesta a pedidos de los clientes. Permite al ORB brindar referencias persistentes sin requerir que los servidores permanezcan en una localización fija ni en constante ejecución. Puede utilizarse para iniciar servicios tras su caída por fallas, y si se desea en conjunto con el servicio de nombres.

El servicio de nombres brinda un mecanismo simple para el anuncio de objetos servidores por su nombre y la localización de los mismos por parte de los clientes que provean el

nombre correcto. Proporciona un repositorio lógico único para el almacenamiento de las referencias a objetos.

La comunicación desacoplada entre múltiples proveedores y consumidores se puede alcanzar en TAO a través del servicio de notificación en tiempo real. Quien además aporta posibilidades de filtrado de la información, uso de prioridades y utilización de políticas de calidad de servicio para realizar las configuraciones necesarias con el objetivo de alcanzar la flexibilidad, escalabilidad y predictibilidad necesarias al sistema.

Además, elegir TAO, nos provee la posibilidad de utilizar, en versiones futuras, DDS, ya que aunque el desarrollo de TAO DDS presenta poca madurez, podría perfilarse como una opción a tener en cuenta si supera sus actuales limitaciones.

2.3.6. Servicio de Nombres de TAO *(Naming Service)*

Como se analizó anteriormente, el ORB TAO brinda un amplio número de servicios para el desarrollo de la comunicación cliente-servidor. El Servicio de Nombres es uno de los principales, el cual es empleado en nuestro sistema.

Este servicio proporciona una equivalencia entre nombres y referencias a objetos, es decir, dado un nombre el servicio devuelve una referencia a objeto asociada a ese nombre (OMG 2007a). Es similar al servicio de dominio de nombres de Internet (DNS – *Domain Name Service*) que traslada los dominios de internet a direcciones IP.

El Servicio de Nombres ofrece una serie de ventajas. Una de ellas es que los clients pueden utilizar nombres significativos para referirse a objetos en vez de tener que tartar con referencias a objeto en forma de cadenas. Además, resuelve el problema que tienen los clientes para acceder a los componentes al iniciar una aplicación.

La misma referencia a objeto puede ser almacenada varias veces con diferentes nombres, pero cada nombre identifica únicamente a una referencia. Un contexto de nombres (*naming context*) es un objeto que almacena relaciones entre referencias y nombres. Es decir, dicho objeto implementa una tabla que traslada nombres a referencias a objeto. Este servicio se puede estructurar como un sistema de ficheros jerárquico en el que los contextos de nombres se comportarían como directorios.

Las definiciones para este servicio se proporcionan en un fichero llamado *CosNaming*, que debe existir en los ORBs que implementen el mismo. Este fichero contiene varias definiciones de tipo y dos interfaces: *NamingContext* y *BindingIterator*, las cuales proporcionan todos los recursos necesarios para utilizar este servicio

A continuación se muestra cómo se puede obtener una referencia a un contexto de nombres utilizando el método resolve_initial_references() que sirve para iniciar varios servicios de CORBA (incluido POA) (Peñalvo 2002).

Iniciar ORB

```
CORBA::ORB_var orb = CORBA::ORB_init(argc,argv);
```

Conseguir una referencia a un contexto de nombres

```
CORBA::Object_var obj = orb->resolve_initial_references("NameService");

CosNaming::NamingContext_var inc = CosNaming::NamingContext::_narrow(obj);
```

Con esto ya se tendría una referencia a objeto para el contexto de nombres inicial. El siguiente paso sería unir un nombre a una referencia a objeto. Esto se realiza con el método bind(). Por ejemplo, si se tiene un objeto de la clase Caja y se quiere exportar utilizando el servicio de nombres.

Se supone que ya se tiene una referencia al contexto inicial, inc. Se crea un objeto de tipo Caja

```
Caja_impl *registradora = new Caja_impl();

Caja_var rg = registradora->_this();
```

Se crea el nombre Super

```
CosNaming::Name nombre;

nombre.length(1);

nombre[0].id = CORBA::string_dup("Super");
```

Se establece la relación entre nombre y objeto

```
inc->bind(nombre,rg)
```

En este momento ya se tendría el objeto de tipo Caja dado de alta en el Servicio de Nombres, listo para ser utilizado por un cliente. La parte del cliente es muy sencilla, para resolver un nombre en una referencia a objeto utiliza el método resolve(). Los pasos a seguir son similares a los realizados por el servidor. También debe obtener una referencia a un objeto contexto de nombres.

Se supone que ya se tiene una referencia a un contexto de nombres llamada inc

```
CosNaming::Name nombre;

nombre.length(1);

nombre[0].id = CORBA::string_dup("Super");
```

Se resuelve el objeto

```
CORBA::Object_var obj = inc->resolve(name);
```

Se hace un *casting*

```
Caja_var cliente = Caja::_narrow(obj);
```

El cliente ya tendría una referencia a un objeto de tipo Caja y podría acceder a los distintos servicios ofrecidos por el objeto. El Servicio de Nombres ofrece más posibilidades que aquí no se detallan, y que se pueden encontrar en la especificación de CORBA correspondiente disponible en (OMG 2007b). Como se ha visto, proporciona un mecanismo muy sencillo para que los clientes puedan localizar los objetos ofrecidos por un servidor.

2.4. Conclusiones parciales

Se han analizado de forma detallada las tecnologías de comunicación actuales, de ellas se ha escogido la más apropiada, profundizando en las implementaciones de la misma, eligiendo también la más conveniente para el intercambio de datos entre el *Gateway* y el *Driver*, como partes del sistema a desarrollar. Las condiciones están creadas para proceder con el diseño y la implementación de cada uno de los elementos que conforman el sistema.

CAPÍTULO 3. GATEWAY OPC/LINUX Y DRIVER OPC. DISEÑO E IMPLEMENTACIÓN

Para el diseño e implementación del sistema se hace necesario analizar en detalle los requerimientos y funcionalidades que debe cumplir, paralelamente a los ya especificados para el proyecto de forma general.

3.1. Requerimientos del sistema

- Proporcionar la posibilidad de que la comunicación se realice en tiempo real con un mínimo de latencia.
- Los datos se deben intercambiar con el SCADA cumpliendo con la interfaz genérica desarrollada por la Línea de Manejadores de Dispositivos (Trujillo 2007).
- El sistema debe garantizar la persistencia de los datos, o sea, conservar los últimos emitidos para realizar su reenvío en caso de que por algún problema no hallan podido llegar a los interesados en su recepción.
- Se debe garantizar la persistencia de las comunicaciones, es decir, al ocurrir una interrupción abrupta no planificada, o un reinicio forzoso del *Gateway* o el *Driver*, debe conservarse el último estado normal para lograr su restablecimiento tras la reconexión.

3.2. Funcionalidades necesarias del sistema

- Adquirir la lista de servidores de OPC locales disponibles.
- Conectar/Desconectar un servidor local. Aunque los servidores de OPC se encuentran permanentemente instalados, estos no necesariamente tienen que estar activos (conectados). Para realizar cualquier acción sobre un servidor este debe ser activado previamente.
- Explorar la información de un servidor. Se obtiene la lista de ítems disponibles en el mismo así como otras informaciones propias del servidor.

- Lectura/Escritura de ítems. Permiten acceder a los valores de las variables así como modificar estos valores en caso de tener privilegios de escritura.
- Creación de grupos de muestreo. Brinda la posibilidad de agrupar las variables de acuerdo a una frecuencia de muestro deseada.

3.3. Sistema propuesto

En la Figura 3.1 se puede ver el sistema propuesto. La PC que contiene Linux respeta la especificación de la interfaz genérica (Trujillo 2007) y usa una capa de transporte para enlazarse con el *Gateway*. A su vez la PC de Windows es la encargada de atender todas las peticiones hechas por los clientes (Linux) y enviar las solicitudes a una capa inferior que gestiona todas las operaciones OPC. Por problemas de seguridad no se aconseja usar servidores de OPC remotos, por lo que todos los servidores de OPC a los que se quiera encuestar deben estar en la misma PC o de lo contrario poner varios *Gateways* en el SCADA.

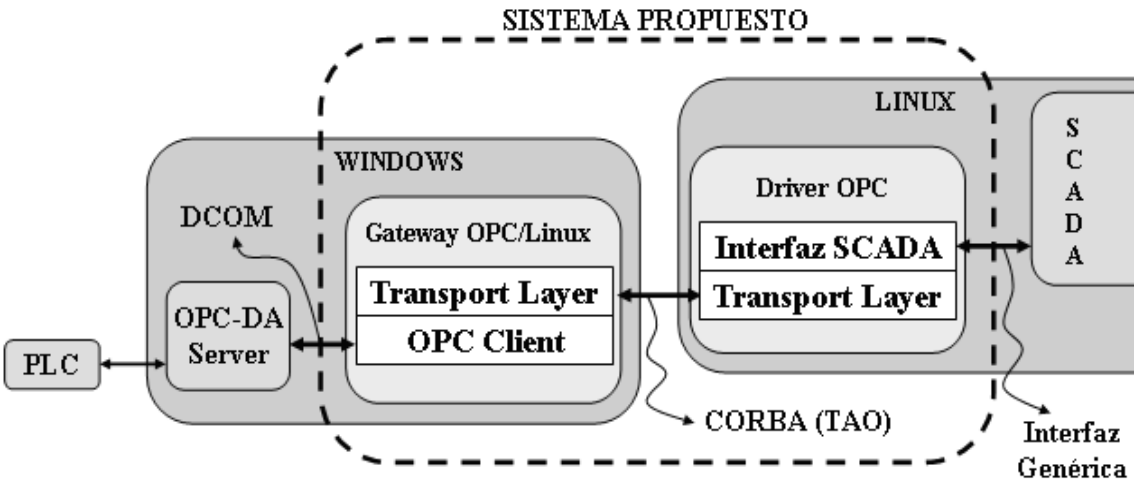

Figura 3.1. Diagrama general del sistema propuesto.

3.4. Diseño del *Gateway* OPC/Linux

Una vez vistos los requerimientos del SCADA y las funcionalidades necesarias, estamos en condiciones de definir la estructura del *Gateway*. Se propone la división en dos capas con el objetivo de delimitar las responsabilidades (Ver Figura 3.2), las cuales serán detalladas a continuación.

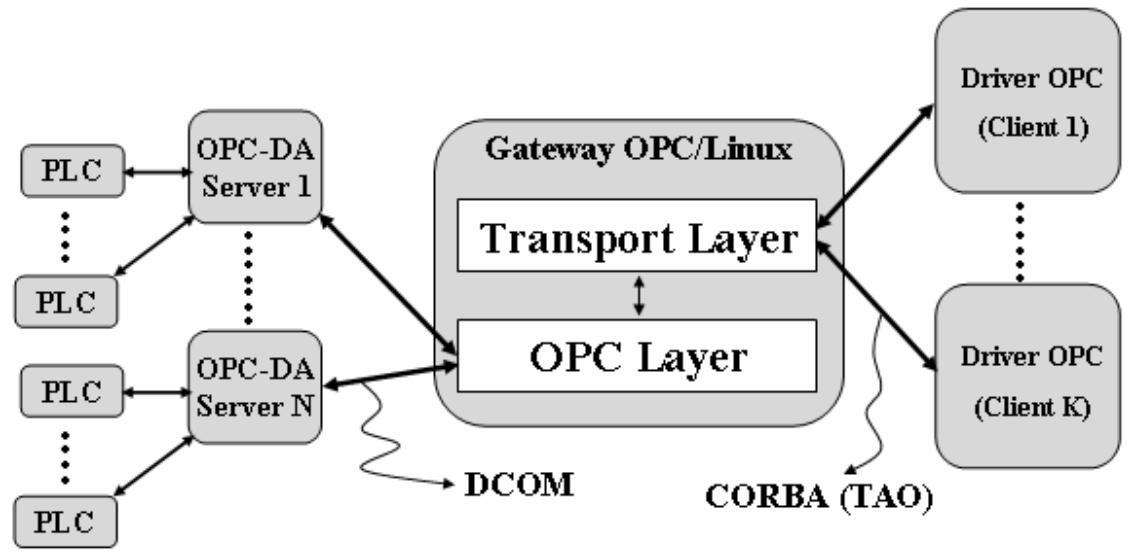

Figura.3.2. Estructura del *Gateway*.

3.4.1. *Transport Layer*

Esta es la capa encargada de la comunicación con los clientes usando CORBA, sus funciones son las siguientes:

- Recibir las peticiones realizadas por uno o varios clientes a través del *Driver* OPC.
- Transmitir el pedido a la capa de OPC.
- Recibir el resultado de la operación devuelto por la capa de OPC.
- Transmitir dicho resultado al cliente que realizó la petición.
- Garantizar que la comunicación se realice en tiempo real con un mínimo de latencia.

3.4.2. OPC *Layer*

Esta es la capa encargada de interactuar con los servidores de OPC por medio de las interfaces DCOM disponibles en los mismos, sus funciones son las siguientes:

- Recibir los pedidos tramitados por la capa de transporte.
- Transmitir el pedido al servidor de OPC indicado.
- Recibir el resultado de la operación devuelto por el servidor de OPC.
- Transmitir dicho resultado a la capa de transporte.
- Garantizar la validación del pedido, en caso de errores devolver un mensaje de error a la capa de transporte.

3.4.2.1. Selección del *toolkit* para la implementación

Para facilitar el desarrollo tanto de clientes como servidores de OPC, varias firmas han desarrollado utilitarios dentro de las que se pueden mencionar: Advasol, eehoo Technology, Kassl, Softing, Technosoftware, entre otras. La mayoría de los *toolkits* elaborados por estas empresas permiten el desarrollo de clientes y servidores sin necesidad de adquirir un amplio conocimiento de la tecnología DCOM; pero no pueden ser utilizados en nuestra implementación, ya sea porque no emplean C o C++ como lenguaje de desarrollo, no son libres o no brindan el código fuente. Entre los pocos *toolkits* disponibles que cumplen con los requerimientos del proyecto, el más completo es el OPC-DA Client SDK, desarrollado por Mark Beharrell (OPC Connect Team 2007) y publicado en el sitio de la Fundación OPC bajo licencia GPL.

3.4.2.2. OPC-DA Client SDK

OPC-DA Client SDK es un *toolkit* orientado a objetos desarrollado en C++ que provee un gran nivel de abstracción para los desarrolladores de clientes de OPC, ocultando detalles tales como la comunicación a través de DCOM. Este *toolkit* brinda la posibilidad de implementar todas las funcionalidades requeridas por nuestro sistema y puede ser descargado de forma gratuita en (Sourceforge 2007). Las Figuras 3.3 y 3.4 muestran los diagramas de clases de este utilitario, ofreciéndose además una breve descripción de dichas clases.

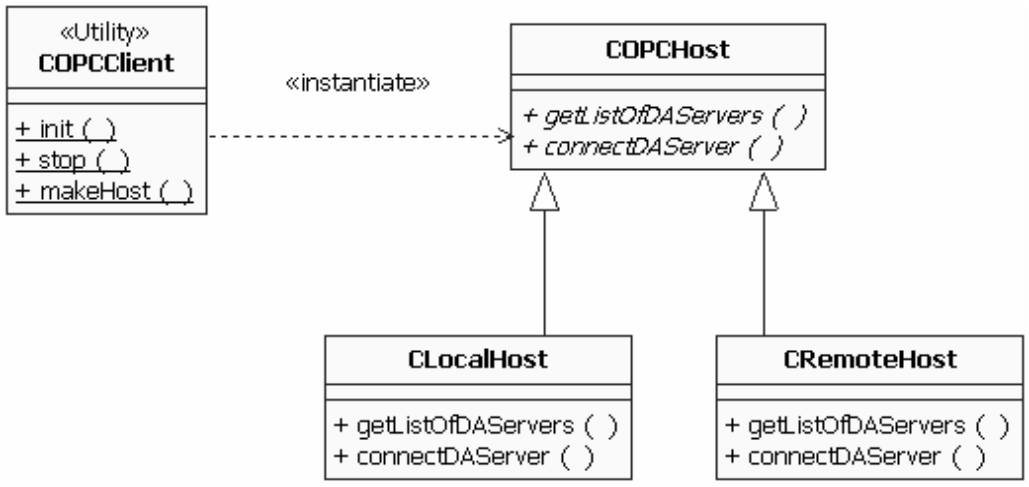

Figura 3.3. Clases principales del OPC-DA Client SDK.

COPCClient: Es el punto de comienzo, encargada de la manipulación de los objetos COM.

COPCHost: Clase abstracta que representa una PC que puede acoger uno o más servidores de OPC.

CLocalHost: Clase usada para acceder a los servidores de OPC locales.

CRemoteHost: Clase usada para acceder a los servidores de OPC remotos.

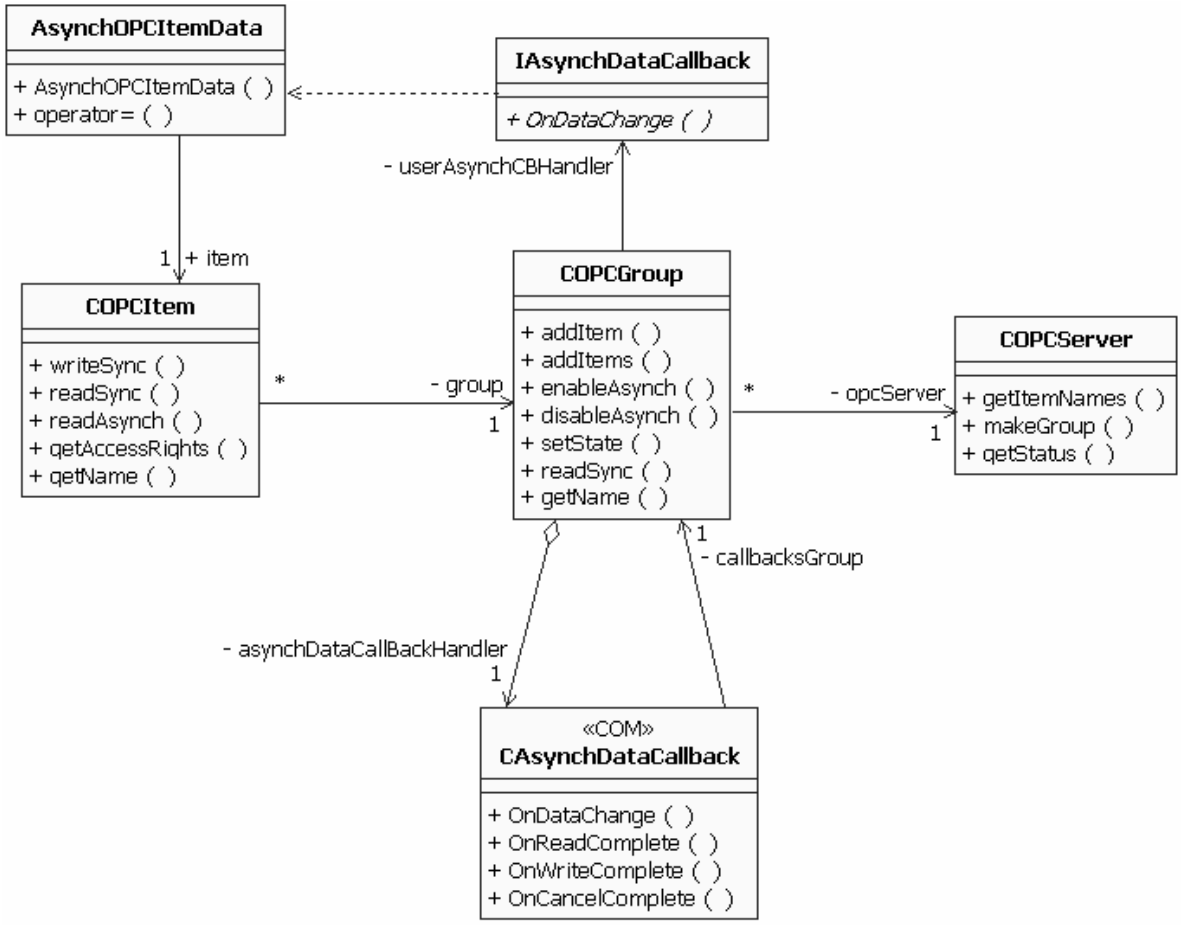

Figura 3.4. Clases utilitarias del OPC-DA Client SDK.

COPCServer: Representación local de un servidor de OPC, ya sea local o remoto. Encierra las interfaces COM para el acceso a dicho servidor.

COPCGroup: Abstracción para el cliente de un grupo de OPC. Contiene las interfaces COM para la manipulación de dicho grupo dentro del servidor.

COPCItem: Provee las interfaces para las operaciones que se desarrollan típicamente a nivel de grupo, como es el caso de lectura y escritura. Se provees a nivel de ítem para facilitar su uso.

CAsynchDataCallback / IAsynchDataCallback: Clases encargadas de manipular la lectura asincrónica a nivel de grupo.

AsynchOPCItemData: Usada para representar los datos de OPC intercambiados de forma asincrónica.

3.4.3. El *Gateway* como cliente de OPC

Como se ha visto hasta el momento, la función principal del *Gateway* es el intercambio de datos con los servidores de OPC respondiendo a las solicitudes de los clientes del SCADA. Una funcionalidad extra y de gran utilidad incorporada en nuestro diseño, es la posibilidad de utilizar el propio *Gateway* como cliente, permitiendo a los administradores del módulo instalado en *Windows* hacer uso de las funcionalidades del sistema a través de una interfaz gráfica (Ver Figura 3.5).

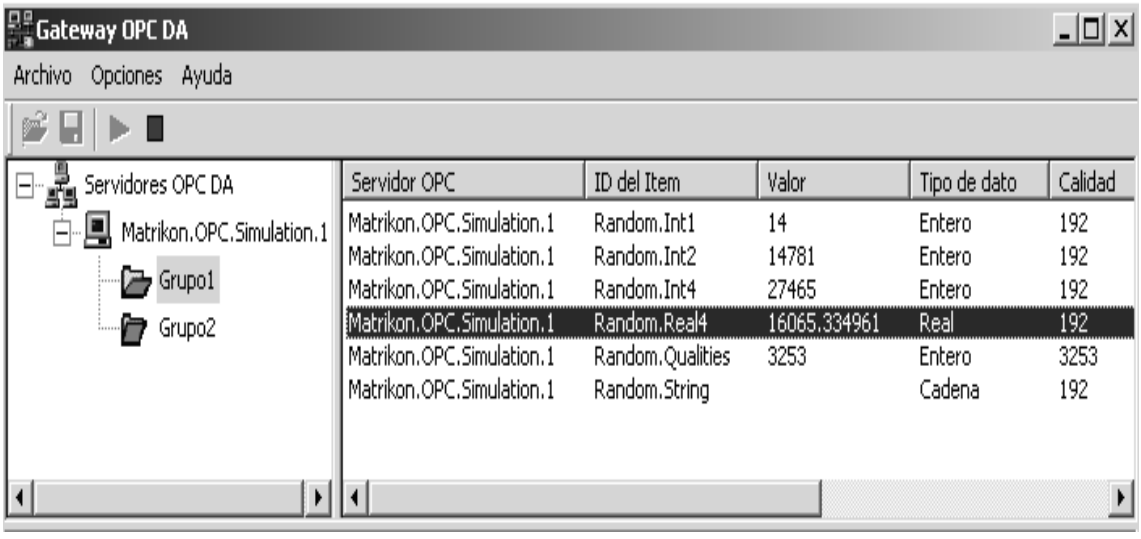

Figura 3.5. Interfaz gráfica del *Gateway*.

3.4.4. Diagrama de clases del *Gateway*

En la Figura 3.6 se muestra el diagrama de clases realizado para el *Gateway*, dichas clases se describen a continuación:

CTransportLayer: Es la encargada de la configuración del ORB y los servicios de CORBA utilizados, como el *Naming Service*.

CTransportManager: Encargada de la transmisión de las peticiones recibidas a través de las interfaces de IDL a la capa de OPC.

COPCLayer: Manipula la lista de servidores disponibles, tramitando las solicitudes recibidas hacia el servidor indicado. Esta clase implementa la funcionalidad de adquirir la lista de servidores locales disponibles (Ver Anexo 4).

COPCManager: Esta es la clase que representa un servidor de OPC, brindando la implementación real de las interfaces descritas en el IDL (Ver Anexos del 5 al 8). Es la clase que emplea directamente los objetos definidos en el *toolkit*.

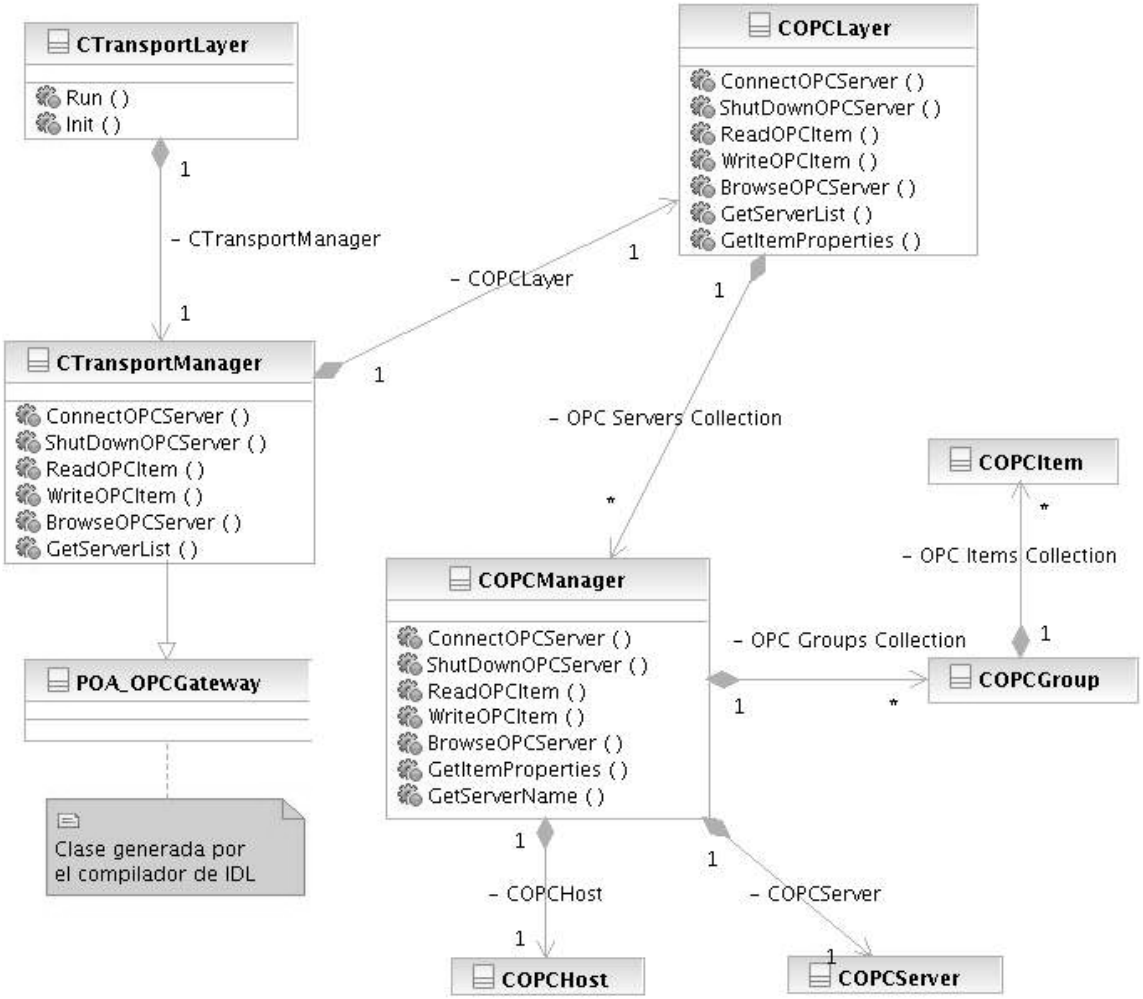

Figura 3.6. Diagrama de clases de *Gateway*.

3.5. Diseño del *Driver* OPC

Al igual que en el caso del *Gateway*, el diseño del *Driver* se realizó basado en dos capas (Ver Figura 3.7), las cuales explicaremos en detalles.

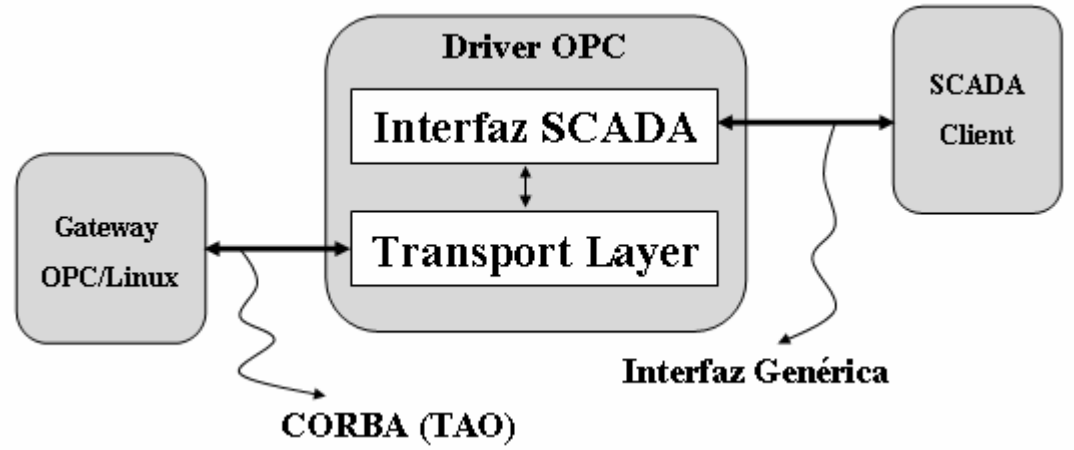

Figura 3.7. Estructura del *Driver*.

3.5.1. *Transport Layer*

Esta capa es la que complementa la comunicación *Gateway-Driver*, sus funciones son:

- Recibir las peticiones tramitadas por la capa interfaz SCADA.
- Transmitir las peticiones a la capa de transporte del *Gateway*.
- Recibir el resultado de la operación devuelto por el *Gateway*.
- Transmitir dicho resultado a la capa interfaz SCADA.
- Garantizar junto a la capa de transporte del *Gateway* la transmisión en tiempo real.

3.5.2. Interfaz SCADA

Esta es la capa encargada de interactuar (cumpliendo con la interfaz genérica) con los clientes del SCADA que deseen acceder, a través del *Gateway*, a las variables disponibles en los servidores de OPC, sus funciones principales son:

- Recibir las peticiones hechas por el cliente.
- Transmitir las peticiones a la capa de transporte.
- Recibir la respuesta de la capa de transporte.
- Transmitir el resultado de la petición al cliente.

3.5.2.1. Interfaz Genérica

En la cadena de tratamiento de la información en un SCADA el primer eslabón es la adquisición de datos, los cuales provienen de disímiles equipos.

Es difícil que un SCADA contemple, en su código, todos los protocolos posibles para esta gran variedad de dispositivos. Por el contrario, lo común es crear un protocolo genérico y la tarea de traducir este protocolo genérico a los protocolos específicos se le encarga a los *drivers*, que, como regla, se programan en módulos independientes al SCADA.

3.5.3. Diagrama de Clases del *Driver* OPC

Las clases principales que aparecen en el *Driver* se muestran en la Figura 3.8. Este diseño, como se describió anteriormente, fue desarrollado por la Línea de Manejadores de Dispositivos (Trujillo 2007) para estandarizar la comunicación del SCADA con todos los *drivers* existentes.

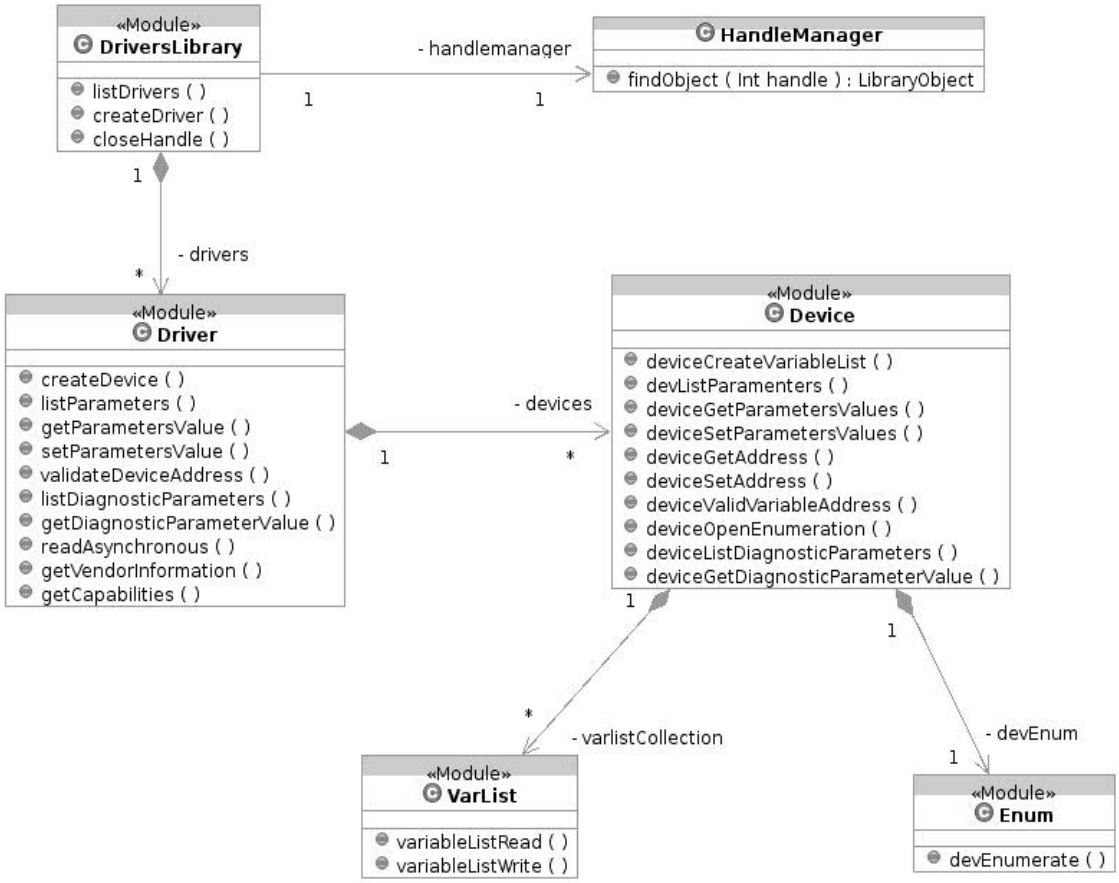

Figura 3.8. Diagrama de clases del *Driver* OPC.

En la siguiente figura se muestra la interfaz gráfica desarrollada en el *Driver* con el objetivo de facilitar la interacción con el mismo.

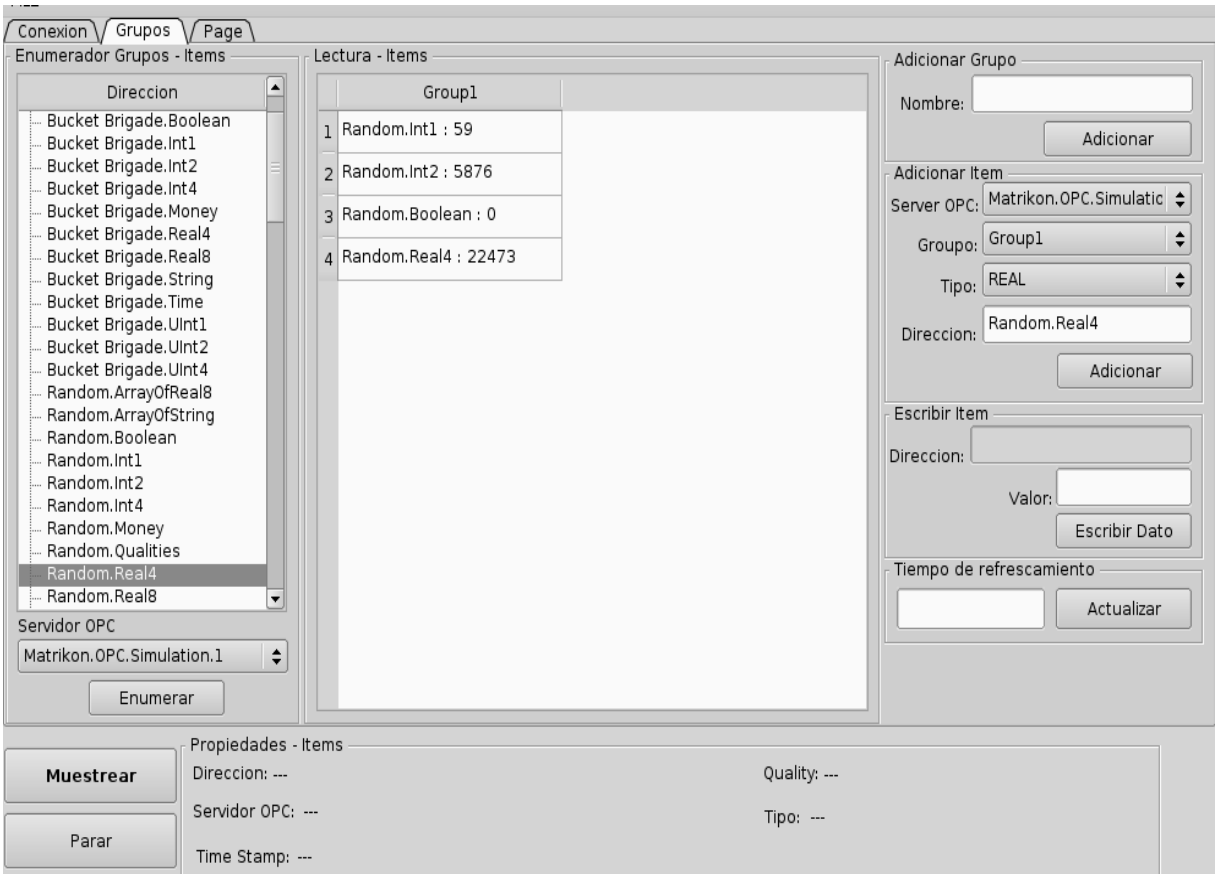

Figura 3.9. Interfaz gráfica del *Driver*.

3.6. Implementación del sistema. Herramientas utilizadas

Para la implementación del *Gateway* sobre plataforma Windows se utilizó Visual C++ .NET, el cual es un completo conjunto de herramientas para la creación de aplicaciones basadas en Microsoft Windows y Microsoft .NET, aplicaciones Web dinámicas y servicios Web XML utilizando el lenguaje de programación C++. Este sólido entorno de desarrollo incluye las bibliotecas estándar del sector ATL (*Active Template Library*) y MFC (*Microsoft Foundation Class*), extensiones avanzadas del lenguaje y eficaces características del entorno de desarrollo integrado (IDE) que permiten a los programadores editar y depurar código fuente de un modo eficaz (Microsoft Corp. 2007).

Para la implementación del *Driver* se utilizo Eclipse, conjuntamente con el plugin que permite adaptar un entorno de desarrollo para C/C++ denominado CDT (C/C++

Development Tooling). Eclipse es una plataforma de software de código abierto, típicamente ha sido usada para desarrollar entornos integrados de desarrollo (del Inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) (Eclipse Foundation 2007).

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para *VisualAge* y actualmente es desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios.

3.7. Pruebas de prototipo

Estas pruebas tienen como propósito asegurar el correcto funcionamiento del sistema.

Las pruebas son realizadas con:

Tipo de dispositivo: PLC S7 200 de Siemens.

Variable censada: Temperatura (variable analógica).

3.7.1. Persistencia de la comunicación

Para comprobar la persistencia de la comunicación se realizaron tres acciones estando el sistema funcionando establemente:

1. Desconexión del *switch* encargado de enrutar los paquetes en la red.
2. Reinicio abrupto del *Driver*.
3. Reinicio abrupto del *Gateway*.

Tras el restablecimiento en cada uno de los casos se comprobó que el sistema mantuvo el último estado de las comunicaciones.

3.7.2. Prueba de latencia para un cliente

En la comunicación siempre es importante conocer, para la buena interpretación de los resultados de una implementación, el tiempo que tarda un paquete de datos en salir de un punto de la red y llegar a otro. En nuestro caso queremos valorar el lapso de tiempo que

demora el *Gateway* en responder a 50000 peticiones hechas por el *Driver*, se hace la prueba de dos formas:

1. Se prueba solamente la comunicación a través de TAO, el *Driver* solicita las variables y el *Gateway* responde con un número aleatorio generado, sin solicitar nada al Servidor de OPC: **la demora fue de 26 segundos (0.52 milisegundos por petición).** Queda demostrado que con el ORB seleccionado se puede cumplir con el tiempo mínimo establecido, dejando un amplio margen para la capa encargada de acceder al Servidor de OPC.
2. El *Driver* solicita las variables y el *Gateway* realiza las peticiones al Servidor de OPC de forma real, o sea, leyendo directamente desde el PLC: **la demora fue de 110 minutos (132 milisegundos por petición).** Se concluye que para un cliente realizando peticiones, el sistema responde adecuadamente según los parámetros establecidos.

3.7.3. Prueba de latencia para varios clientes

La prueba de latencia para un cliente no justificaría totalmente el correcto funcionamiento del sistema, en este coexisten muchos clientes puesto que el SCADA comprende una gran cantidad de usuarios que necesitan frecuentemente acceder a los datos del campo, y en ocasiones ocurren de forma simultánea. Es por ello necesario analizar el comportamiento de nuestro diseño ante tales circunstancias. Para esta prueba 200 clientes realizarán 50000 peticiones cada uno de forma simultánea.

Al comprobar sólo la capa de comunicación similar a la prueba anterior, se obtuvo como resultado que se atiende a un cliente en aproximadamente **89 minutos (112 milisegundos por petición)**.

Esto ocurre debido a que hay un solo hilo atendiendo a los clientes, provocando que se cree una cola de peticiones, donde cada cliente tiene que esperar por el resto para ser atendido. Para optimizar esta respuesta se propone un diseño basado en multihilo, donde la capa de transporte atienda simultáneamente más de una petición.

3.7.3.1. Diseño basado en multihilo

Un hilo es un flujo de control secuencial dentro de un programa. Un hilo es, al igual que un proceso, un flujo de control que puede gozar de cierta autonomía (puede tener sus propias estructuras de datos), pero a diferencia de un proceso, diversos hilos de una aplicación pueden compartir los mismos datos.

Un hilo no puede correr por sí mismo, siempre se ejecuta dentro de un programa y se pueden programar múltiples hilos de ejecución para que corran simultáneamente. La utilidad de la programación multihilo resulta evidente, por ejemplo, un navegador Web puede descargar un archivo de un sitio, y acceder a otro sitio al mismo tiempo. Si el navegador puede realizar simultáneamente dos tareas, no tendrá que esperar hasta que el archivo haya terminado de descargarse para poder navegar a otro sitio.

Algo importante a tener en cuenta en el diseño de multihilos es el acceso compartido a los recursos, varios hilos pueden estar accediendo a la vez a un mismo recurso solamente si este no es crítico. En el diseño basado en multihilos nos encontramos con este problema, las variables a intercambiar que se encuentran en los dispositivos de campo son recursos de hardware (críticos) por lo que no se puede acceder a ellas simultáneamente. Para resolver esta situación se hace uso de la *cache* del servidor de OPC, la cual es un recurso de software (no crítico) y contiene una imagen de los datos del campo.

Una ventaja adicional en el uso de esta memoria es la reducción del tiempo de respuesta a los clientes, puesto que los datos que estos solicitan ya no se buscan directamente en el campo sino que se toman de la *cache*, la cual se actualiza a la frecuencia deseada por el cliente.

3.7.4. Prueba de latencia para varios clientes atendidos con multihilo

Se realizo la prueba anterior con el nuevo diseño y se obtuvo el siguiente resultado:

Tiempo total de atención a un cliente: **277 segundos (5.54 milisegundos por petición).**

Esto reafirma el beneficio del uso de multihilo acompañado de la memoria *cache*, por lo que es nuestra propuesta definitiva.

3.8. Conclusiones parciales

Para resolver el problema planteado, se ha diseñado he implementado un sistema compuesto por dos elementos que se comunican a través de TAO, donde se definen dentro de ellos diferentes capas con funcionalidades bien especificadas. Se han escogido detalladamente las herramientas para la confección general del sistema, por ejemplo, el paquete para la implementación del cliente de OPC y las herramientas para la programación del *Gateway* en Windows y el *Driver* en Linux. Con la realización de las pruebas se ha garantizado el correcto funcionamiento del sistema, sirviendo estas como justificación para la proyección hacia mejores soluciones de forma inmediata, como es el diseño basado en multihilo.

CONCLUSIONES

1. El amplio estudio realizado sobre la especificación OPC-DA, sus objetos e interfaces, ha permitido desarrollar correctamente la aplicación, además sirve de base para futuros trabajos relacionados con el tema de creación de clientes y servidores OPC-DA, dada la escasa bibliografía en español existente en el tema.

2. La búsqueda bibliográfica realizada solo arrojó la existencia de unos pocos trabajos relacionados con el tema de túneles de OPC Windows-Linux, pero en ningún caso basados en los paradigmas de software libre y código abierto.

3. El estudio comparativo realizado de las tecnologías de comunicación cliente-servidor, resulta de gran interés práctico en el tema de las telecomunicaciones.

4. La eficacia del sistema fue demostrada a través de las pruebas realizadas al prototipo.

5. El prototipo desarrollado fue recientemente entregado a las instancias que dirigen el proyecto SCADA Nacional de PDVSA y finalmente certificado para su puesta a prueba en las instalaciones de esta gigante empresa.

RECOMENDACIONES

1. Migrar la tecnología existente en el SCADA a OPC-UA. Proponemos el empleo de esta tecnología dadas sus características robustas y ampliadas con respecto a las anteriormente existentes, resolviendo de esta forma problemas tales como el bajo rendimiento de OPC en Linux.
2. DDS es una tecnología joven recientemente aprobada por el OMG que aun se encuentra en constante cambio. Brinda un gran número de posibilidades entre ellas robustez en aplicaciones de tiempo real, y uso muy eficiente del canal de comunicación, pero su actual inestabilidad atenta contra su utilización. Se propone observar el desarrollo de esta tecnología para su futura aplicación.
3. Potenciar el estudio teórico de las tecnologías de comunicación entre computadoras en la carrera de Ingeniería en Telecomunicaciones y Electrónica, dado el gran auge que tienen actualmente a nivel mundial los sistemas distribuidos.
4. Enfocar las herramientas de programación tratadas en la carrera de Ingeniería en Telecomunicaciones y Electrónica, al desarrollo práctico de sistemas de comunicación basados en las tecnologías actuales.

REFERENCIAS BIBLIOGRÁFICAS

Avery, J. (2007). "Creating .Net Applications on Linux and Mac OS X." Retrieved 15/05/2007, from http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9016832.

Balcells, J. (2005). "Introducción a Sistemas SCADA." Retrieved 06/05/2007, from www.isa.uniovi.es/~felipe/files/infindII/documentos/scadas.pdf.

Bollow, N. (2007). "DotGNU Portable.NET." Retrieved 15/05/2007, from http://www.gnu.org/projects/dotgnu/pnet.html.

Brochure. (2006). "SIMATIC WinCC - Visualización de procesos." Retrieved 13/05/2007, from www.automation.siemens.com/salesmaterial-as/brochure/es/brochure_simatic-wincc_es.pdf.

Burke, T. J. (2007). "The Passion and Mystique of OPC UA." Retrieved 15/05/2007, from http://www.automationworld.com/view-1315.

Castillo, A. D. (2006). "Curso práctico de Corba." Retrieved 17/05/2007, from http://www.programacion.com/tutorial/acscorba/.

Catalunya, U. P. d. (2007). "Conceptos generales de sistemas distribuidos." Retrieved 10/05/2007, from http://studies.ac.upc.edu/EPSC/FSD/FSD-ConceptosGenerales.pdf.

Cetus Team. (2002). "Distributed Objects & Components: General Information." Retrieved 10/05/2007, from http://www.cetus-links.org/oo_distributed_objects.html.

Chávez, H. (2004). "Decreto N° 3.390." Retrieved 04/05/2007, from http://www.gobiernoenlinea.gob.ve/docMgr/sharedfiles/Decreto3390.pdf.

Cogent Real-Time Systems. (2007). "OPC to Linux systems." Retrieved 08/05/2007, from http://www.opcdatahub.com/Features/OPC_to_Linux.html.

Eclipse Foundation. (2007). "Eclipse - an open development platform." Retrieved 29/05/2007, from http://www.eclipse.org/.

Free Software Foundation. (2006). "Licencias GNU GPL, GNU LGPL, GNU FDL." Retrieved 07/05/2007, from http://www.gnu.org/licenses/licenses.es.html.

Grisby, D. (2006). "Free High Performance ORB." Retrieved 20/05/2007, from http://omniorb.sourceforge.net/.

Horstmann, M. (1997). "DCOM Architecture." Retrieved 13/05/2007, from http://msdn2.microsoft.com/en-us/library/ms809311.aspx.

Kamen, E. (1999). Introduction to Industrial Controls and Manufacturing, Academic Press.

Kane, V. (2007). "Desarrollo SCADA Nacional - Documento Visión."

Kondor, R. (2003). "SCADA Deployment using OPC." Retrieved 08/05/2007, from http://www.automationmedia.com/ARDetail.asp?ID=%2033.

Lee, E. (1999). "CORBA Applications In GNOME." Retrieved 21/05/2007, from http://developer.gnome.org/doc/whitepapers/ORBit/about-orbit.html.

MICO Project Team. (2006). "MICO is CORBA." Retrieved 21/05/2007, from http://www.mico.org/index.html.

Microsoft Corp. (2004). "Summary of Available COM+ Services." Retrieved 13/05/2007, from http://msdn2.microsoft.com/en-us/library/k702b5fx(VS.80).aspx.

Microsoft Corp. (2005). "What Is .NET?" Retrieved 12/05/2007, from http://www.microsoft.com/net/basics.mspx.

Microsoft Corp. (2007). "Resumen del producto Visual C++ .NET." Retrieved 29/05/2007, from http://www.microsoft.com/latam/visualc/producto/resumen.asp.

NeutralBit. (2006). "Introducción a la Seguridad en Sistemas SCADA." Retrieved 05/05/2007, from http://www.neutralbit.com/downloads/NB-NB-0001-EXT-SCADA-Introduccion%20a%20la%20seguridad%20en%20sistemas%20SCADA%20ES.pdf.

OMG. (2007a). "Catalog of OMG CORBA Services Specifications." from http://www.omg.org/technology/documents/corbaservices_spec_catalog.htm.

OMG. (2007b). "Catalog of OMG CORBA Specifications." Retrieved 23/05/2007, from http://www.omg.org/technology/documents/corba_spec_catalog.htm.

OPC Connect Team. (2007). "OPC Toolkits and Free Source Code." Retrieved 26/05/2007, from http://www.opcconnect.com/source.php.

OPC Foundation. (2007). "Currently Released Specifications." Retrieved 09/05/2007, from http://www.opcfoundation.org/Products/Specifications.aspx?CM=-1.

PDVSA. (2005a). "Petróleos de Venezuela S.A. PDVSA." Retrieved 03/05/2007, from http://www.pdvsa.com/.

PDVSA. (2005b). "El sabotaje contra la industria petrolera nacional." Retrieved 03/05/2007, from http://www.pdvsa.com/index.php?tpl=interface.sp/design/readmenuhist.tpl.html&newsid_obj_id=119&newsid_temas=13.

PDVSA (2006a). Desarrollo de SCADA Nacional.

PDVSA (2006b). Anexo 5 Diagnóstico del Proyecto SCADA Nacional del Convenio Marco PDVSA – ALBET, S.A.

Peñalvo, F. G. (2002). "Fundamentos para el desarrollo de aplicaciones distribuidas basadas en CORBA." Retrieved 22/05/2007, from http://tejo.usal.es/inftec/2002/DPTOIA-IT-2002-001.pdf.

Prado, E. (1997). "Nuevas tecnologías e interactividad: Gran Almacén Universal Virtual." Retrieved 12/05/2007, from www.cibersociedad.net/textos/articulo.php?art=77.

Prensa PDVSA. (2007). "Energía y Petróleo para el Pueblo." Retrieved 03/05/2007, from http://www.aporrea.org/energia/n90215.html.

RTI. (2007). "RTI Data Distribution Service." Retrieved 17/05/2007, from http://www.rti.com/products/data_distribution/RTIDDS.html.

Schmidt, D. C. (2006a). "Lessons Learned Building Reusable OO Frameworks for Distributed Software." Retrieved 11/05/2007, from http://www.cs.wustl.edu/~schmidt/CACM-lessons.html.

Schmidt, D. C. (2006b). "Real-time CORBA with TAO." Retrieved 20/05/2007, from http://www.cs.wustl.edu/~schmidt/TAO.html.

Sourceforge. (2007). "OPC DA Client SDK 0.4 Download." Retrieved 26/05/2007, from https://sourceforge.net/project/showfiles.php?group_id=136222.

Stallman, R. M. (2006). "Artículos sobre Software Libre." Retrieved 07/05/2007, from http://www.smaldone.com.ar/documentos/stallman.shtml.

Subías, M. P. (2003). "Redes P2P una nueva forma de almacenar y acceder a la información." Retrieved 12/05/2007, from www.coit.es/publicac/publbit/bit141/especial/2.pdf.

Suchit. (2005). "OPC Technology." Retrieved 12/05/2007, from http://www.codeproject.com/useritems/opctechnology.asp?df=100&forumid=246673&exp=0&select=1861420.

Sun Microsystems. (2004). "Getting Started Using Java RMI." Retrieved 16/05/2007, from http://java.sun.com/j2se/1.5.0/docs/guide/rmi/hello/hello-world.html.

Trujillo, R. (2007). "Especificación de la interfaz genérica con el SCADA - Línea de Manejadores de Dispositivos." Retrieved 25/05/2007.

Vázquez, M. H. (2007). "Sistemas Distribuidos, sistemas SCADA y Middleware." Retrieved 19/05/2007.

Wikipedia. (2007). "Cliente-servidor." Retrieved 13/05/2007, from http://es.wikipedia.org/wiki/Cliente-servidor.

ANEXOS

Anexo 1. Aval de aplicación

A quien pueda interesar:

Por medio de la presente se les informa del resultado de los estudiantes de 5to de Telecomunicaciones Roberto Carlos Rodríguez López y Sándor Otero Rodríguez en el proyecto SCADA Nacional de PDVSA.

Durante su estancia de producción en nuestra empresa, CEDAI, y en la Universidad de las Ciencias Informáticas, los compañeros se incorporaron a la línea de OPC, para posibilitar la integración de la tecnología OPC al sistema SCADA de PDVSA.

Por la línea en cuestión, el país logró ingresar un monto de **97 882.00 USD**, donde los estudiantes tienen un peso importante en la realización de la tarea. Como reconocimiento al trabajo realizado y para garantizar el cumplimiento en tiempo de los entregables contratados se ha invitado a uno de los estudiantes a realizar una estancia de trabajo en las instalaciones de PDVSA en Mérida, Venezuela. En dicha estancia se deben exponer los resultados y discutir el entregable con toda la documentación pactada.

Además, el diseño propuesto, se puede aplicar como integración de los sistemas SCADA que se ejecutan en Linux u otras plataformas, con la tecnología OPC dependiente de Windows, dotando a los SCADA del estándar OPC.

Sin otro asunto,

M.Sc. René González Rodríguez
Especialista en Automática CEDAI VC
Responsable de la línea OPC del proyecto
ALBET-PDVSA SCADA Nacional

Anexo 2. Definición de las interfaces disponibles para la comunicación en el lenguaje IDL

```
interface OPCGateway
  {
      enum varType {intType, floatType, strType};
      typedef unsigned long TDateTime;
      typedef sequence<string> strSequence;

      union UnionValue switch (varType){
            case intType:
                  long long iValue;
            case floatType:
                  double fValue;
            case strType:
                  string strValue;
      };

      struct VarLinkInfo{
            string  address;
            TDateTime timeStamp;
            UnionValue value;
            long quality;
            varType valueVarType;
      };

      long ConnectOPCServer (in string serverNamne);
      long ShutDownOPCServer (in string serverNamne);
      long ReadOPCItem (in string serverName, inout VarLinkInfo item);
      long WriteOPCItem (in string serverName, in VarLinkInfo item);
      long BrowseOPCServer (in string serverName, inout strSequence workSpace);
      long GetServerList (inout strSequence serverList);
  };
```

Anexo 3. Implementaciones funcionales de CORBA

- PrismTech: OpenFusion e*ORB SDR C++ Edition (No brinda el código fuente)
- PrismTech: JacORB ME(Lenguaje: Java)
- PrismTech: OpenFusion TAO ORB (No completamente libre)
- PrismTech: OpenFusion JacORB ORB (Lenguaje: Java)
- PrismTech: OpenFusion RTOrb Java Edition (Lenguaje: Java)
- PrismTech: OpenFusion RTOrb Ada Edition (OrbRiver Ada) (Lenguaje: ADA)
- Borland: VisiBroker.(Software privativo)
- The Community OpenORB Project: OpenORB (Lenguaje: Java)
- MTDORB (Lenguajes: “Delphi” y “Kylix”)
- LuaORB (Solo para “Lua”)
- Robin (No completamente libre)
- IONA: Orbix (Software privativo)
- Orbacus (Software privativo)
- FU Berlin ("Freie Universität" de Berlín): JacORB (Lenguaje: Java)
- Objective Interface Systems: ORBexpress RT (Software privativo)
- Objective Interface Systems: ORBexpress ST (Software privativo)
- ObjectWeb: OpenCCM (Lenguaje: Java)
- Mico/E (Lenguaje: Eiffel, no tiene tiempo real, proyecto inmaduro)
- Gibraltar (Todavía inmaduro, versión 0.2xx, no tiene iiop, ni tiempo real)
- ISP C++ (Solo es CORBA 2.0)
- ORBit
- OmniORB
- DOC: TAO
- OCI: TAO

Anexo 4. Implementación de la interfaz encargada de adquirir la lista de servidores de OPC

```
int COPCLayer::GetServerList(OPCGateway::strSequence & serverList)
        throw(CORBA::SystemException)
{
        int result;
        try
        {
                char** tmpServerList = NULL;
                COPCHost* tmpHost;
                int count;

                tmpHost = COPCClient::makeHost("");
                tmpHost->getListOfDAServers(IID_CATID_OPCDAServer20,
                                                        tmpServerList, &count);

                serverList = OPCGateway::strSequence(count, count, tmpServerList);

                for (unsigned int i=0; i<count; i++)
                        delete tmpServerList[i];

                delete[] tmpServerList;
                delete tmpHost;

                result = GET_SERVER_LIST_OK;
        }
        catch (...)
        {
                result = GET_SERVER_LIST_ERROR;
        }
        return result;
}
```

Anexo 5. Implementación de la interfaz encargada de activar un servidor de OPC

```
int COPCManager::ConnectOPCServer (const char* newServerName)
	throw(CORBA::SystemException)
{
	int result;
	try
	{
		serverName = new char [strlen(newServerName)+1];
		strcpy(serverName, newServerName);
		host = COPCClient::makeHost("");
		opcServer = host->connectDAServer(serverName);
		refreshRate = 100;
		group = opcServer->makeGroup("Group", true, 100, refreshRate, 0.0);
		result = CONNECT_OPC_SERVER_OK;
	}
	catch (...) {
		result = CONNECT_OPC_SERVER_ERROR;
	}

	return result;
}
```

Anexo 6. Implementación de la interfaz encargada de adquirir la lista de variables disponibles en un servidor de OPC

```
int COPCManager::BrowseOPCServer (OPCGateway::strSequence & workSpace)
  throw(CORBA::SystemException)
{
        int result;
        try
        {
                char** opcItemNames = NULL;
                int count;
                opcServer->getItemNames(opcItemNames, &count);

                workSpace = OPCGateway::strSequence(count, count, opcItemNames);

                for (unsigned int i=0; i<count; i++)
                        delete opcItemNames[i];
                delete[] opcItemNames;

                result = BROWSE_OPC_SERVER_OK;
        }
        catch (...) {
                result = BROWSE_OPC_SERVER_ERROR;
        }
        return result;
}
```

Anexo 7. Implementación de la interfaz encargada de la lectura de un ítem

```
int COPCManager::ReadOPCItem (OPCGateway::VarLinkInfo& item)
	throw(CORBA::SystemException)
{
	int result;
	COPCItem* readWritableItemLocal;

	string strPrig = item.address;
	try
	{
		OPCItemData data;
		readWritableItemLocal = group->addItem(strPrig.c_str(), true);

		readWritableItemLocal->readSync(data, OPC_DS_DEVICE);
		item.quality = data.wQuality;
		item.timeStamp = data.ftTimeStamp;

		if (item.valueVarType == OPCGateway::intType)
			item.value.iValue(data.vDataValue.iVal);
		if (item.valueVarType == OPCGateway::floatType)
			item.value.fValue(data.vDataValue.fltVal);
		if (item.valueVarType == OPCGateway::strType)
			item.value.strValue(data.vDataValue.strVal);

		result = READ_OPC_ITEM_OK;
	}
	catch (...) {
		result = READ_OPC_ITEM_ERROR;
	}
	return result;
}
```

Anexo 8. Implementación de la interfaz encargada de la escritura de un ítem

```
int COPCManager::WriteOPCItem (const OPCGateway::VarLinkInfo& item)
	throw(CORBA::SystemException)
{
	int result;
	const char* tmpAddress = item.address;
	try
	{
		VARIANT var;
		readWritableItem = group->addItem(tmpAddress, true);

		if (item.valueVarType==OPCGateway::intType)
		{
			var.vt = VT_I2;
			var.iVal = item.value.iValue();
			readWritableItem->writeSync(var);
		}

		if (item.valueVarType==OPCGateway::floatType)
		{
			var.vt = VT_R4;
			var.fltVal = item.value.fValue();
			readWritableItem->writeSync(var);
		}

		if (item.valueVarType==OPCGateway::strType)
		{
			var.vt = VT_LPSTR;
			var.strVal = item.value.strValue();
			readWritableItem->writeSync(var);
		}

		result = WRITE_OPC_ITEM_OK;
	}
	catch (...)
	{
		result = WRITE_OPC_ITEM_ERROR;
	}
	return result;
}
```