UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FACULTAD 5

TÍTULO: "DISEÑO DE UN *MIDDLEWARE* BÁSICO PARA EL INTERCAMBIO DE INFORMACIÓN DE UN SISTEMA SUPERVISOR DE PROCESOS AUTOMATIZADOS"

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

AUTORA: **ANA SILVIA TELLERÍA MARTÍNEZ**

TUTOR: **ING. JOSÉ OMAR PADRÓN RAMOS**

CONSULTANTE: **M.SC. MOISÉS HERRERA VÁZQUEZ**

MAYO, 2007

DECLARACIÓN DE AUTORÍA

DECLARO QUE SOY EL ÚNICO AUTOR DE ESTE TRABAJO Y AUTORIZO AL <NOMBRE ÁREA> DE LA UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS A HACER USO DEL MISMO EN SU BENEFICIO.

PARA QUE ASÍ CONSTE FIRMO LA PRESENTE A LOS ____ DÍAS DEL MES DE _____ DEL AÑO _____

INSERTAR NOMBRE(S) DE AUTOR(ES) **INSERTAR NOMBRE(S) DE TUTOR(ES)**

_____ _____

DATOS DE CONTACTO

El tutor es profesor del departamento de automática de la facultad de Eléctrica de la Universidad Central de las Villas Marta Abreu, UCLV. Graduado de Ingeniería en Automática en el 2005. Posee la categoría docente de Instructor. Es doctorante aún no categorizado científicamente. Presenta dos años de graduado y 2 años de experiencia en el tema.

El consultante es Especialista Superior en Automática de la Empresa de Automatización Integral, CEDAI. Graduado de Ingeniería en Automática en el 1998, máster en automática y mención en robótica y control inteligente en el año 2003. Posee la categoría docente Instructor y es doctorante no categorizado, presenta 9 años de experiencia en el tema.

Le agradezco todo a mis padres, desde mi nacimiento hasta el día de hoy, todo su amor, sus esfuerzos y el apoyo en la realización de este trabajo.

A mi hermano entrañable por su ejemplo, por haber sido tantas veces inspiración, maestro, amigo, y hermano.

A mis abuelas por sus atenciones y cariño y a toda mi familia.

A mi tutor, al consultante y a Ariel, sin ellos no hubiera sido posible estas páginas.

Le agradezco a Emedia la vinculación con la producción, el ejemplo, enseñazas y amistad de Fung y René.

Le agradezco a mis amigos todos y a Magda, Iliana, Irina y Gretter que son también hermanitas.

A Amado, Maikel, Luis Ángel, Manuel y Maylín por su amistad y ayuda.

A Mildrey, Coca, Mayra y Yenieris por su preocupación.

A mi querida profe Mairela por el teatro.

A quienes me hayan impartido al menos una clase.

Le agradezco a Amauris cada segundo.

A mis padres, mi familia, mis amigos y a Amauris

A Edson y a Tati para que se empinen

Resumen

En el presente trabajo se realiza el diseño de un middleware que satisface las necesidades básicas de comunicación entre los distintos módulos de un Sistema Supervisor de Procesos Automatizados con requerimientos críticos de tiempo real y fiabilidad.

Se comienza con un estudio inicial de los principales conceptos incidentes en la problemática. En este estudio se analizan las características fundamentales, elementos históricos, de importancia y de actualidad de los sistemas distribuidos, así como de las aplicaciones para la supervisión y control de los procesos industriales, y del *middleware* como componente integrador de gran valor para dichas plataformas de software. También se investiga sobre las especificidades de la aplicación del tiempo real en tales ámbitos.

Seguidamente se lleva a cabo un proceso de selección tecnológica para determinar cuál modelo y tecnología utilizar en el diseño y futuro desarrollo del software facilitador de las comunicaciones entre módulos. Se definen los requisitos funcionales y no funcionales que guiarán el diseño del *middleware* básico. Se realiza dicho diseño, teniendo en cuenta la tecnología seleccionada y los requerimientos definidos. Finalmente se efectúa un análisis de los resultados determinando la relevancia de su aporte al desarrollo del Sistema Supervisor de Procesos Automatizados.

Palabras Claves

Sistema Distribuido, SCADA, *Middleware*, Selección Tecnológica, Diseño, Corba, TAO

CONTENIDO

CONTENIDO **1**

INTRODUCCIÓN **3**

CAPÍTULO I FUNDAMENTACIÓN TEÓRICA. SISTEMAS DISTRIBUIDOS, SISTEMAS SCADA, TIEMPO REAL Y MIDDLEWARE **8**

1.1 INTRODUCCIÓN 8

1.2 SISTEMAS DISTRIBUIDOS 8

1.2.1 Definición *8*

1.2.2 Características *9*

1.3 ¿QUÉ ES UN SCADA? 10

1.3.1 Definición *10*

1.3.2 Características *11*

1.3.3 El tiempo real y los sistemas SCADA *12*

1.3.4 Importancia de la tecnología. Ejemplos de aplicación. *14*

1.4 MIDDLEWARE 17

1.4.1 Sus inicios *17*

1.4.2 Concepto *19*

1.4.3 Importancia de su aplicación *20*

1.4.4 El software de comunicación en los sistemas SCADA *21*

CAPÍTULO II ANÁLISIS, SELECCIÓN Y JUSTIFICACIÓN DE LA TECNOLOGÍA MIDDLEWARE **24**

2.1 INTRODUCCIÓN 24

2.2 REQUERIMIENTOS TECNOLÓGICOS DEL MIDDLEWARE 24

2.3 REVISIÓN Y SELECCIÓN DE LA TECNOLOGÍA MÁS APROPIADA 25

2.3.1 Descripción de las tecnologías candidatas *26*

2.3.1.1 DCOM *26*

2.3.1.2 COM+ y .Net *27*

2.3.1.3 RMI *29*

2.3.1.4 CORBA *31*

2.3.1.5 DDS *33*

2.4 COMPARACIÓN DE TECNOLOGÍAS 34

2.4.1 Omniorb *35*

2.4.2 TAO *36*

2.4.3 MICO *36*

2.4.4 ORBit ... *37*
2.4.5 Al tomar el intercambio de datos como centro ... *37*
2.4.6 Selección y justificación de la tecnología más apropiada ... *38*
2.5 ANÁLISIS DEL *NOTIFICATION SERVICES* DE TAO. MODELO PROVEEDOR/CONSUMIDOR ... 39
2.5.1 Comunicación por eventos ... *40*
2.5.2 Eventos ... *40*
2.5.3 Filtros ... *41*
2.5.4 Políticas de calidad de servicio ... *41*
2.5.5 Fiabilidad y persistencia ... *43*

CAPÍTULO III DISEÑO DEL *MIDDLEWARE* BÁSICO CON ACE+TAO ... 45

3.1 INTRODUCCIÓN ... 45
3.2 REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES DEL *MIDDLEWARE* BÁSICO ... 45
3.3 MODELO DEL DISEÑO DEL *MIDDLEWARE* BÁSICO ... 48
3.3.1 Centrado en la arquitectura ... *48*
3.3.2 Dirigido por Casos de Uso ... *51*
3.3.2.1 Caso de Uso: Recibir datos ... *51*
3.3.2.2 Caso de Uso: Enviar datos ... *65*
3.4 ANÁLISIS DE LOS RESULTADOS ... 73

CONCLUSIONES ... 75

RECOMENDACIONES ... 76

BIBLIOGRAFÍA ... 77

ANEXOS ... 83

A1 IMPLEMENTACIONES FUNCIONALES DE CORBA ... 83
A2 DESACOPLAMIENTO: EJEMPLO DE IMPLEMENTACIÓN DEL DISEÑO. ... 84

GLOSARIO ... 90

INTRODUCCIÓN

Múltiples son las esferas donde se patentiza la integración entre las naciones de Cuba y Venezuela. Educación, cultura, salud e informática sólo son algunos de los numerosos ejemplos. La Universidad de la Ciencias Informáticas (UCI) ha tenido la oportunidad de tomar parte activa en estos lazos de amistad que fomentan el progreso para ambos países; primero con su participación en la misión Milagro y luego con el desarrollo de varios proyectos.

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. Entre sus fines también se encuentran "...motorizar el desarrollo armónico del país, afianzar el uso soberano de los recursos, potenciar el desarrollo endógeno..." (PDVSA, 2005)

PDVSA cuenta en sus instalaciones con Sistemas de Supervisión y Adquisición de Datos (SCADA) 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 acaecido 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. Estos hechos evidenciaron la necesidad de luchar por la independencia tecnológica, de PDVSA como empresa medular en la economía venezolana, y del país en general.

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, 2006)

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 Frías, 2004)

En el 2006 el estado cubano, y en específico la UCI, se incorporan 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, 2006). Inmediatamente comenzó la participación activa en el proyecto de los profesores y estudiantes de la UCI, junto a otros profesores con experiencia en el tema provenientes de otras universidades del país.

Los sistemas supervisores de procesos automatizados permiten controlar el estado de los procesos industriales. Dichas aplicaciones están estructuradas en módulos con diversas funcionalidades, entre las que pueden ser mencionadas: adquisición de datos de dispositivos de campo, tratamiento de alarmas, visualización de procesos, reportes de operación y persistencia de la información. La mayoría de estos sistemas a nivel mundial son privativos, existiendo en la actualidad gran escasez de soluciones libres.

El tráfico de información entre los diferentes componentes de un SCADA impone el uso de tecnologías de comunicación que garanticen un intercambio eficiente de datos entre los procesos, tales como estados de los dispositivos del proceso productivo, tratamiento de alarmas y comandos de operadores.

En la arquitectura de un sistema SCADA como en la de todo sistema distribuido, el *middleware* o software intermedio, es la capa de software situada por encima de los niveles físicos y de transporte, y por debajo de las capas de gestión y aplicación de usuario. Entre sus principales funcionalidades se encuentra, el establecimiento de métodos de comunicación entre las aplicaciones en ejecución desde cualquier punto del sistema, garantizando que la localización de cada una de ellas sea transparente a las demás. Esta cualidad caracteriza su complejidad, lo cual incide en el rendimiento, fiabilidad, eficiencia y sobrecarga del sistema.

Una de las líneas firmadas en contrato como responsabilidad cubana, tiene como objetivo desarrollar el *middleware* para el Sistema Supervisor de Procesos Automatizados de la compañía PDVSA, de forma tal que garantice el intercambio de información entre los diferentes módulos del sistema con eficiencia, seguridad y mantenibilidad. 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 tomada en compromiso. Su magnitud y su requerimiento de ejecución en tiempo real crítico. PDVSA es la cuarta compañía petrolera a nivel mundial y la empresa más grande de Latinoamérica (Prensa PDVSA, 2007). Varios de sus procesos industriales necesitan seguimiento en intervalos de valores en el orden de los milisegundos.

La elaboración de un *middleware* con requerimientos tan críticos sin poseer experiencia antecedente de su implementación con tecnología libre, como sucede en el caso del equipo de desarrollo cubano, representa un riesgo considerable.

La situación problémica expuesta con anterioridad deviene en necesidad de respuesta al subsiguiente **problema científico**:

¿Qué *middleware* es necesario diseñar como base al futuro software de comunicación del Sistema Supervisor de Procesos Automatizados de Petróleos de Venezuela S.A.?

El **objeto de estudio** de la presente investigación es el *middleware* para un Sistema Supervisor de Procesos Automatizados. La misma tiene como objetivo:

Diseñar un software de comunicación básico que tribute al futuro desarrollo del *middleware* para el Sistema Supervisor de Procesos Automatizados de Petróleos de Venezuela S.A.

El **campo de acción** lo constituyen las funcionalidades básicas del *middleware* del Sistema Supervisor de Procesos Automatizados de Petróleos de Venezuela S.A.

Las preguntas científicas que guiarán la investigación, de acuerdo con el problema científico y las necesidades detectadas, son:

1. ¿Cuál tecnología es la idónea para utilizar en la implementación del *middleware*?
2. ¿Qué funcionalidades básicas debe poseer el *middleware* del Sistema Supervisor de Procesos Automatizados de Petróleos de Venezuela S.A.?
3. ¿Qué diseño de *middleware* da respuesta a las necesidades de comunicación básicas entre los distintos componentes del Sistema Supervisor de Procesos Automatizados de Petróleos de Venezuela S.A., de forma eficiente, segura y brindado además predictibilidad y mantenibilidad?

Para el logro del objetivo de la investigación y teniendo en cuenta los elementos anteriores, se plantean las **tareas científicas** siguientes:

1. Desarrollar la fundamentación teórica del tema en interés donde se aborden los conceptos de sistema distribuido, sistema SCADA, tiempo real, y *Middleware*.
2. Realizar una revisión y justificación del modelo y la tecnología *middleware* a utilizar.
3. Definir los requerimientos funcionales y no funcionales de un *middleware* básico para el Sistema Supervisor de Procesos Automatizados de Petróleos de Venezuela S.A.

4. Diseñar un software de comunicación que garantice las funcionalidades fundamentales de intercambio de información, teniendo en cuenta la tecnología seleccionada y los requerimientos determinados.
5. Efectuar un análisis de los resultados y determinar la relevancia de su aporte en el Sistema Supervisor de Procesos Automatizados de Petróleos de Venezuela S.A.

Para el desarrollo de las tareas científicas se combinarán diferentes métodos y técnicas en la búsqueda y procesamiento de la información. Entre los que se prevén como fundamentales se encuentran:

A nivel Teórico

Analítico – Sintético: Durante la definición de las funcionalidades básicas necesarias en la capa de comunicación del SCADA PDVSA.

Inductivo – deductivo: En la revisión y justificación del modelo y la tecnología para desarrollar *middleware* que se utilizará. Mediante el análisis de casos aislados se arribarán a proposiciones generales que luego se emplearán en la inferencia de la tecnología y modelo más adecuados a emplear en la implementación.

Análisis sistémico: Durante el desarrollo del diseño mediante la elaboración de los diagramas de clases.

Método de modelación: Durante el desarrollo del diseño mediante la elaboración de los diagramas de secuencia.

A nivel empírico

Entrevista: Se aplica a los profesores Dr. Rafael Trujillo y M.Sc. Moisés Herrera Vázquez sobre diferentes temáticas. Se escoge dicha muestra por su reconocida labor a nivel nacional y años de experiencia en el trabajo con sistemas SCADA.

Capítulo I FUNDAMENTACIÓN TEÓRICA. SISTEMAS DISTRIBUIDOS, SISTEMAS SCADA, TIEMPO REAL Y MIDDLEWARE .

1.1 INTRODUCCIÓN

El *middleware* a desarrollar para el sistema SCADA PDVSA no es una aplicación para utilizar de forma independiente sino una biblioteca que se insertaría en un sistema distribuido garantizando una serie de funcionalidades necesarias de comunicación entre módulos.

En el presente capítulo se perfilan con más detalles, los conceptos de sistema distribuido, SCADA y *Middleware* como fundamentación teórica base para obtener una comprensión más cabal del problema a resolver y de su entorno.

1.2 SISTEMAS DISTRIBUIDOS

“Divide y vencerás”

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.

1.2.1 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" (Universidad Polítecnica de Catalunya, 2007)
- "..sistema de computación con un número de componentes que cooperan entre sí comunicándose a través de la red." (Universidad Polítecnica de 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.

1.2.2 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.

La primera tecnología de computación distribuida en adquirir un uso extendido fue la "Llamada a Procedimiento Remoto" o RPC por sus siglas en inglés (*Remote Procedure Call*), aunque pronto fue sustituida por otras que aportaban mayor funcionalidad y flexibilidad.

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. Douglas C. Schmidt**:

"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." (Schmidt, y otros, 2006)

1.3 ¿QUÉ ES UN SCADA?

"..see more, do more, and get more..."

1.3.1 DEFINICIÓN

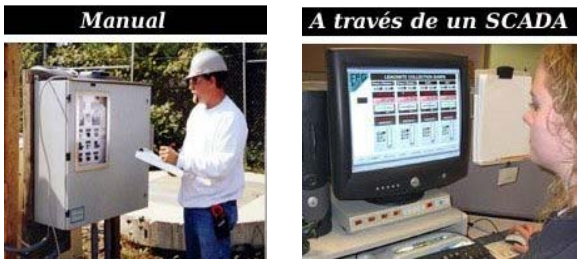

***Figura 1**: Diferencias entre el trabajo manual y automatizado.* (Dennison, 2004)

Tal como se ilustra, un Sistema Supervisor de Procesos Automatizados, **SCADA** por sus siglas en inglés (***S**upervisory **C**ontrol **A**nd **D**ata **A**cquisition*),

utiliza la computación además de tecnologías de medición, control y comunicación, para automatizar la supervisión y control de procesos industriales. En un SCADA se recolectan los datos provenientes de sensores y de **dispositivos de campo**, por ejemplo de distintos **PLC** (*Programmable Logic Controller*), probablemente ubicados a distancia. Le procede el procesamiento, trasmisión y despliegue de la información capturada, ya sea para propósitos de observación o de control. Usualmente estos datos se muestran en una o más terminales donde es posible efectuar análisis, automatizados o no, cuyos resultados proporcionan retroalimentación al personal interesado o al propio proceso en ejecución. La generación de alarmas y la toma de decisiones mediante operatoria humana constituyen ejemplos de esta retroalimentación.

La historia de los sistemas de supervisión comienza a inicios de los noventa con el advenimiento de la *telemetry;* tecnología que involucra la transmisión y recolección de datos obtenidos mediante detección de **condiciones** en **tiempo real**. La inspección de las condiciones remotas fue posible gracias a la convergencia de otros avances tecnológicos como la electricidad, el telégrafo, el teléfono, y la comunicación inalámbrica. (Dennison, 2004)

Los SCADA de hoy en día, deben enfrentarse a un nuevo reto dentro del control automatizado: provisión de interfaces a equipos obsoletos todavía lo suficientemente flexibles para adaptarse a los cambios de mañana.

En la actualidad se cuenta con la presencia de diversos SCADA como parte integral de la inmensa mayoría de los ambientes industriales complejos.

1.3.2 CARACTERÍSTICAS

Entre las razones que justifican el auge de estas valiosas aplicaciones para la automatización se encuentran, su capacidad de recolección y procesamiento de información de diversas fuentes de forma fiable, segura y a altas velocidades, y la presentación descriptiva de los datos del proceso tecnológico al personal

interesado, posibilitándole tomar en el momento adecuado, la decisión apropiada.

De forma general se consideran como funciones básicas de un sistema SCADA las enunciadas a continuación (Hernández Lugones, 1998):

1. Supervisión Remota de Instalaciones
2. Control Remoto de Instalaciones
3. Procesamiento de Información
4. Presentación de Gráficos Dinámicos
5. Generación de Reportes
6. Presentación de Alarmas
7. Almacenamiento de Información Histórica
8. Presentación de Gráficos de Tendencias
9. Programación de Eventos

1.3.3 EL TIEMPO REAL Y LOS SISTEMAS SCADA

Las tareas cuyas restricciones de tiempo en su realización sean de vital cumplimiento para el funcionamiento del sistema al que pertenezcan, se denominan "Tareas de Tiempo Real". A su vez, un sistema de tiempo real es una aplicación cuyo correcto funcionamiento depende de la cronología y predictibilidad de sus eventos.

Las aplicaciones de tiempo real están alcanzando en la actualidad una importancia creciente. Se les puede encontrar en ambientes variados, desde aplicaciones para la automatización hasta en equipamiento médico.

Dichas aplicaciones brindan o responden a un evento externo en un tiempo determinado y predecible. Independientemente de que muchas requieran altas velocidades de cómputo, las mismas cubren un amplio rango de tareas con diferentes dependencias temporales. Para cada una, la línea de tiempo posee una definición disímil. Lo que puede resultar rápido un sistema, puede resultar lento o "muy tarde" para otro. Por ejemplo, mientras un mecanismo de control necesita recolectar datos y dirigir un activador con una precisión de

microsegundos, un monitoreo científico de la presión del aire necesita recolectar los datos en intervalos de varios minutos. Sin embargo, el éxito de ambas aplicaciones depende de unos requerimientos de tiempo bien definidos.

El concepto de predictibilidad puede variar en dependencia del campo de aplicación, pero para las aplicaciones de tiempo real generalmente significa, la terminación de las tareas en todos los casos, necesariamente dentro de un tiempo determinado. En dependencia de la situación, las aplicaciones de tiempo real no predictivas pueden presentar pérdida de datos, incumplimiento del "tiempo límite", o pérdida de la producción de la industria. Ejemplos de aplicaciones de tiempo real incluyen control de procesos, fábricas de robots automáticos, simulación de vehículos, entre otros.

El *software* de tiempo real puede ser clasificado en tiempo real fuerte o tiempo real leve.

Las aplicaciones de tiempo real "fuerte" requieren responder a los eventos en un predeterminado intervalo de tiempo para su correcto funcionamiento. Una aplicación de tiempo real fallaría por completo si, no logra cumplir con los límites de tiempo especificados. Aunque muchas aplicaciones de tiempo real requieren brindar respuestas a altas velocidades, la brevedad del intervalo de respuesta no es un objetivo central en una aplicación de "fuerte" tiempo real.

Un ejemplo de aplicación de tiempo real fuerte, es un sistema de control de la trayectoria de misiles, donde una réplica tardía puede provocar un desastre.

Por otra parte, las aplicaciones de tiempo real leve no "fallan" por dejar de cumplir con un "tiempo límite". Algunas aplicaciones de tiempo real leve son capaces de procesar grandes cantidades de datos o replicar de forma muy veloz, pero la clave principal es que cumplir o no con la restricción de tiempo no es una condición primordial para la ejecución exitosa de dicho software. Un ejemplo de una aplicación en tiempo real es un sistema de reservación de aerolíneas, donde una tardanza ocasional es tolerable aunque no deseable.” (Nilsson)

En el SCADA PDVSA por desarrollar, se necesitarán ejecutar parte de las tareas y procesos con variados niveles de tiempo real. Dispositivos de la capa de más bajo nivel, como los PLC, lidiarán con el tiempo real "fuerte", por su parte el *middleware* trabajará con elementos cuyas categorías oscilan desde tiempo real leve hasta tiempo real medio. En la mayoría de las circunstancias si las réplicas sobrepasan determinados límites de tiempo podrían ocasionar trastornos entre simplemente "no deseables" y hasta "medios" al sistema. Por ejemplo, pudiera tener el operador una información desactualizada de los procesos que están ocurriendo sin mayores consecuencias aunque dicha situación no sea deseable, pero también pudiera producto de la información incorrecta poseída, tomar decisiones o dejar de tomar decisiones en casos críticos con afectaciones considerables para algunos de los procesos industriales o parte de ellos.

1.3.4 IMPORTANCIA DE LA TECNOLOGÍA. EJEMPLOS DE APLICACIÓN.

Los SCADA son sin lugar a dudas herramientas invaluables para el mejoramiento de las operaciones en los distintos sistemas que supervisan. Hoy en día prácticamente no se concibe un proceso industrial cuyo control no esté al menos parcialmente automatizado con estas utilísimas herramientas.

Los beneficios son múltiples. Primeramente, a través del monitoreo de los diferentes procesos haciendo uso extensivo de los avances tecnológicos es posible identificar los problemas y encontrarles solución antes del acaecimiento de un detenimiento no deseado y perjudicial de las actividades monitoreadas. (Johnson, 2005). La captura, procesamiento y presentación de extensas cantidades de datos a un operador pueden ser realizadas de forma más efectiva a través de un SCADA. También, como es posible operar remotamente con la mayoría de estos sistemas de control sobre los servicios o procesos productivos, la restauración tras una interrupción puede ser efectuada a distancia con mayor velocidad. Por ejemplo, en una situación crítica donde un técnico desee trabajar en un punto problemático localizado a millas de distancia y necesite cerrar una

válvula de flujo de petróleo puede realizarlo inmediatamente desde la subestación utilizando el sistema de control y continuar su trabajo sin tener que trasladarse, con exactitud y sin pérdidas de tiempo.

El SCADA también ayuda a determinar la localidad de donde proviene la interrupción, la severidad de la misma y el número de personas afectadas, informaciones importantes para garantizar la asignación apropiada de los recursos necesarios para la recuperación. Sus reportes permiten analizar las causas de las diversas problemáticas.

A través de la integración de una gran variedad de dispositivos electrónicos de control y red que facilitan el monitoreo y administración, los SCADA posibilitan la obtención de niveles de fiabilidad y robustez nunca antes pensados.

Maximizan la producción, la eficiencia de los procesos de control, la calidad de los productos, y posibilitan la integración con otros sistemas existentes. Mejoran la calidad del producto mediante la provisión de alarmas que permiten al personal efectuar una calibración predictiva y adecuada de parámetros del proceso previendo desviaciones de los límites antes de que las mismas ocurran. Conllevando incluso a un incremento significativo del tiempo de actividad del sistema. Mediante el despliegue de un SCADA centralizado es posible reducir los costos de operación y mantenimiento debido a que se requiere menos personal para monitorear equipamientos de localidades remotas, resultando en un incremento de la efectividad del operador y menor número de viajes de mantenimiento. La habilidad de monitorear y controlar los procesos de grandes plantas desde una única localidad, y al mismo tiempo recolectar y compartir en tiempo real los datos de los procesos, se ha convertido en una herramienta invalorable para el mejoramiento de las operaciones. Las alarmas remotas y avisos de mal funcionamiento brindan la capacidad de evitar las costosas reparaciones de los equipos. Permiten brindar respuestas frente a las situaciones de emergencia de una forma mucho más rápida y efectiva porque los datos son analizados antes, durante y después de cada evento. Se generan

alertas instantáneas debido a fallas de equipamiento en subestaciones incluso sin personal. Los operadores responden rápidamente a las alarmas y realizan cierres, apagados y nuevos enrutamientos sin la necesidad de enviar personal a la escena. Los puntos que presenten algún problema son capturados de forma instantánea. El diseño gráfico de la interfaz de usuario para cada campo, incrementa cualitativamente en varios grados el tiempo de respuesta. Automatizaciones de muchas otras funciones específicas brindan eficiencias adicionales. El sistema pudiera incluso ser reiniciado remotamente en momentos que sea necesario por situaciones críticas o de alarma. Reduce riesgos que puedan afectar al medio ambiente.

Algunos ejemplos de SCADA reconocidos a nivel mundial son:

- **CitectSCADA**
- **SIMATIC WinCC** de la compañía Siemens
- **Movicon X**
- **GENESIS 32** de IONICS
- **InTouch** de Wonderware

Todos son software privativo y únicamente se ejecutan sobre Windows. Constituyen aplicaciones configurables para su despliegue en variados ambientes industriales. Muchas otras compañías se dedican a desarrollar sistemas SCADA privativos en forma de soluciones a la medida para cada cliente. **Pvbrowser** es una de las pocas variantes de SCADA funcionales, capaz de ejecutarse también sobre Linux, es además, software libre. Otra variante similar pero en un estado inferior de madurez y mucho menos confiable es **Visual.**

En Cuba hasta el momento existe poca experiencia en el tema y ninguna incursión en software libre. Sobre Windows si se han desarrollado SCADA de mediana complejidad como "**Titán**" creado por el grupo Inel de Villa Clara y "**Eros**" desarrollado por el grupo cubano Eros de Nicaro en Holguín. Sobre este último sistema un integrante del grupo, el Dr. Rafael Trujillo Codorniú profesor de la Universidad **ISMM**, comenta que va por la versión 5.5 y aún sigue activo.

Corre sobre Windows únicamente aunque se prevé para versiones futuras, pase a ser multiplataforma. Está siendo utilizado en el 90 por ciento de los grupos electrógenos del país y en muchísimas otras empresas. (Trujillo Codorniú, 2007)

Es importante recordar que todo SCADA, como ejemplo representativo de sistema distribuido debe enfrentar diferentes retos para alcanzar un desarrollo exitoso y la coexistencia satisfactoria de sus funcionalidades.

El Dr. Douglas C. Schmidt, hace alusión a una crisis de la programación distribuida y seguidamente enuncia sus distintos "síntomas" alegando que mientras el hardware se torna más pequeño, más veloz y barato, en contraposición, el software aumenta de tamaño, se torna más lento y más caro. Pregunta cuáles serán las razones, y también se da la respuesta: "Las complejidades inherentes y accidentales de los esquemas de soluciones: componentes, ***frameworks***, **patrones** y arquitecturas". (Schmidt, 1998)

El *middleware* surge precisamente para facilitar el desarrollo de los sistemas distribuidos.

1.4 MIDDLEWARE

"Metadata is data about data; Middleware is software about software." ***Nicholas Gall***

1.4.1 SUS INICIOS

Algunas fuentes remontan la primera utilización del vocablo *middleware* a 1968, registrado en el reporte de la Conferencia de Ingeniería de Software NATO (NATO SOFTWARE COMITEE, 1969). Allí es empleado como software mediador entre la aplicación y el sistema operativo, con el objetivo de adaptar las funcionalidades genéricas del sistema de archivos a las necesidades específicas de las funcionalidades de la aplicación.

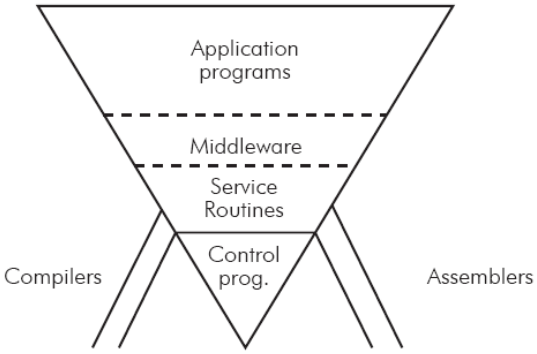

***Figura 2**: Pirámide Invertida de la Conferencia de Ingeniería de Software de NATO.*

También se han enunciado sendas definiciones más, la primera de ellas dos años después en 1970, donde el *middleware* fue definido como “software de fabricantes de computadoras, que ha sido confeccionado para las necesidades particulares de una instalación”. Y en 1972 se hace referencia a él como un término nuevo, introducido debido a que, algunos sistemas habían devenido en aplicaciones con “complejidades únicas”, provocando requerimientos de extensión o modificación del sistema operativo; “los programas realizadores de estos cambios necesarios fueron llamados *middleware* porque se situaban entre el sistema operativo y los programas de la aplicación”. (Gall, 2003)

Sin embargo la Enciclopedia de Computación Distribuida (“Encyclopedia of Distributed Computing”, Kluwer Academic Press, 2003) ficha la primera aparición del término a finales de la década del ochenta como descriptor de las aplicaciones de administración de conexiones de red, y fija su utilización ampliada a mediados de los noventa, cuando la tecnología de red hubo alcanzado suficiente fuerza y visibilidad. Plantea además que: “Para aquel entonces, el *middleware* había evolucionado en un conjunto de paradigmas y servicios más abundantes y maduros, ofrecidos para facilitar y brindar manejabilidad al desarrollo de sistemas distribuidos. El término fue mayormente asociado con las base de datos relacionales por muchos profesionales en el mundo de negocios durante los tempranos noventa, pero a mediados de década no siguió sucediendo. Conceptos similares a los de nuestros días fueron

primeramente denominados como sistemas operativos de red, sistemas operativos distribuidos, y entornos de computación distribuida." (David E., 2003)

En la actualidad, el éxito de las tecnologías *middleware* ha colocado a dicho término junto a otros tan familiares como sistema operativo, lenguaje de programación, red, y base de datos, legados de generaciones anteriores de desarrolladores de software. Mediante el desacoplamiento de las funcionalidades específicas de la aplicación y la lógica de las complejidades accidentales e inherentes en una infraestructura distribuida, el *middleware* posibilita a los desarrolladores de la aplicación concentrarse en la programación de las funcionalidades específicas del sistema, en lugar de enfrentarse continuamente a los desafíos de las estructuras de bajo nivel. (Schmidt, y otros, 2004)

1.4.2 CONCEPTO

"El *middleware* es un software de infraestructura que reside entre las aplicaciones y el sistema operativo, redes, y *hardware* subyacentes, específicamente intentando brindar una plataforma más apropiada para el desarrollo y ejecución de los sistemas distribuidos." (Schmidt, 2005)

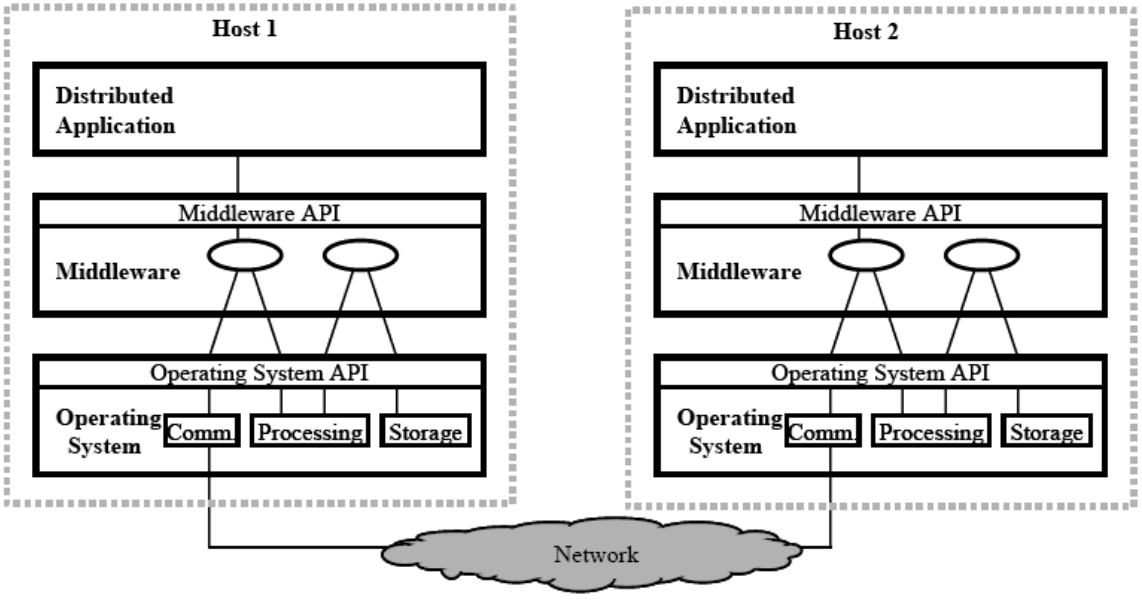

***Figura 3**: La capa del middleware ubicada en contexto* (David E., 2003)

Las funcionalidades fundamentales del *middleware* se resumen en: concertar cómo los módulos de una aplicación se interrelacionan y operan coordinadamente posibilitando, rebasar los obstáculos de comunicación existentes entre las distintas partes del sistema y tanto el hardware como el software de bajo nivel. Además de contribuir a la creación de los sistemas distribuidos imprimiendo velocidad y robustez al proceso al brindar, servicios configurables y posibilidad de integración con componentes desarrollados por distintos proveedores de tecnologías.

Al decir de Douglas C. Schmidt y Richard E. Schantz, el *middleware* representa la confluencia de dos áreas claves de las Ciencias de la Computación: los sistemas distribuidos y la ingeniería de *software* avanzada (Schmidt, 2005). Pues por un lado, como sucede con los recursos para la elaboración de aplicaciones distribuidas, procura la integración de dispositivos computacionales geográficamente independientes en función de obtener un sistema único. Y por otra parte, justo como las técnicas de ingeniería de *software* para desarrollar sistemas basados en componentes, utiliza programación orientada a objetos y la encapsulación de patrones de interacción exitosos, en frameworks reusables para obtener una reducción en las complejidades del software.

1.4.3 IMPORTANCIA DE SU APLICACIÓN

Entre los beneficios que aporta una correcta implementación del middleware se encuentran:

La abstracción. Libera a los desarrolladores de la programación de bajo nivel (como por ejemplo, la programación de **socket** a nivel de red) abstrayéndolos de detalles de la plataforma inherentes al desarrollo de aplicaciones distribuidas y en consecuencia, protegiéndolos contra el acometimiento de errores. Simplifica la implementación de los sistemas distribuidos al brindar un conjunto consistente de abstracciones de alto nivel

orientadas a la red mucho más cercanas a la aplicación y requerimientos del sistema.

La reutilización. Provee un amplio y probado rango de servicios orientados al desarrollador para los ambientes de red, como por ejemplo, los de autenticación y seguridad, y en general, reduce los costos del ciclo de vida del *software* a través de la reutilización de los conocimientos de desarrollos anteriores, capturando las implementaciones de patrones claves en *frameworks* reusables, y de esta forma evitar re-implementaciones innecesarias. (Schmidt, 2005)

1.4.4 EL SOFTWARE DE COMUNICACIÓN EN LOS SISTEMAS SCADA

Debido a las características, dimensiones y complejidades inherentes de la supervisión de los procesos industriales, la mayoría de los sistemas SCADA en funcionamiento constituyen sistemas distribuidos. Se dividen en diferentes módulos o entidades, por ejemplo, los módulos de software del despliegue y la base de datos en tiempo real se encuentran geográficamente dispersos. Dichas partes localmente distribuidas de la aplicación pueden y suelen presentar variadas heterogeneidades, por ejemplo, pueden estar ejecutándose sobre diferentes arquitecturas y sistemas operativos. Las mismas se mantienen en constante comunicación y cooperación funcionando como una entidad única cuyo objetivo fundamental es lograr el control y supervisión de los procesos industriales. En los sistemas SCADA como en todo sistema distribuido se comparten recursos, ejemplo de ellos son: elementos de configuración, datos capturados y procesados y lógica operacional. También, como en todo sistema distribuido, la lógica de los sistemas de control y supervisión se encuentra dispersa entre sus distintos componentes, como la base de datos histórica, los módulos de reporte y despliegue si solo se mencionan algunos. El fuerte carácter distribuido de estas aplicaciones de monitoreo y control condiciona su desarrollo, dificultándolo en gran medida. Entre las características más desafiantes del SCADA distribuido se pueden mencionar, la heterogeneidad y, el

carácter “abierto”, la escalabilidad, concurrencia y transparencia a obtener como valores indispensables e intrínsecos.

La comunicación desempeña un rol crucial entre las diversas entidades de los sistemas distribuidos. La abstracción de dicha comunicación posee distintos enfoques. Por un lado, un enfoque de bajo nivel requeriría que el desarrollador tuviera conocimiento de muchos detalles de la interacción entre los componentes del sistema, como el empaquetamiento de datos complejos en mensajes simples de transmisión. Por otra parte, un enfoque de alto nivel quedaría muy acoplado a un entorno de programación específico y a un único lenguaje de programación. El enfoque del middleware yace en el medio. El objetivo es introducir una capa de software intermedia entre la aplicación y la red que proporcione una abstracción de las comunicaciones y la transferencia de datos. El desarrollador no debería diferenciar entre las invocaciones de un procedimiento remoto y uno local. El middleware sería el “pegamento” conector de dos aplicaciones “independientes” que brinda, abstracción necesaria para la programación y enmascaramiento de la heterogeneidad de las redes subyacentes, hardware y sistema operativo y que se encuentra diseñado para resolver los desafíos mencionados con anterioridad presentes durante el desarrollo de la mayoría de los sistemas de supervisión y control.

Entre los mayores desafíos del software de comunicación en un sistema SCADA se encuentran la obtención de alta fiabilidad o tolerancia a fallas y la realización de operaciones en tiempo real. Dichas propiedades no dependen únicamente del middleware, sino también del sistema en su totalidad, tanto de las plataformas software como hardware.

Existen tecnologías variadas que facilitan el desarrollo del software de comunicación. Entre las más destacadas y utilizadas se pueden mencionar, OPC, DCOM, CORBA y Java/RMI.

Una vez estudiados con detenimiento los principales conceptos pertenecientes al entorno del desarrollo de software de comunicación para sistemas de procesos automatizados, se ha adquirido la base teórica necesaria

para proseguir con el siguiente paso, el “Análisis y selección de la tecnología Middleware” a utilizar.

Capítulo II ANÁLISIS, SELECCIÓN Y JUSTIFICACIÓN DE LA TECNOLOGÍA MIDDLEWARE

2.1 INTRODUCCIÓN

El *middleware* provee los lineamentos de implementación así como las herramientas de desarrollo para facilitar la construcción de grandes y heterogéneos sistemas distribuidos. “Elección de la arquitectura de aplicación, configuración o despliegue” son sólo tres de muchos de los típicos desafíos de la programación distribuida. Dado que no hay una arquitectura única que satisfaga a todos los sistemas de este tipo, la elección del *middleware* adecuado con su configuración más apropiada es un punto de diseño clave con un impacto dramático en el diseño y rendimiento de una aplicación.

2.2 REQUERIMIENTOS TECNOLÓGICOS DEL MIDDLEWARE

Para proceder con el análisis y obtener la tecnología más adecuada para el caso que ocupa el presente trabajo se hace necesario remitirse a la problemática planteada en la introducción de este documento y abundar en sus características.

El SCADA PDVSA será la primera población donde se insertará el *middleware* por desarrollar, pero a la vez, como dicho sistema de supervisión y control de procesos automatizados debe ser lo suficientemente flexible y configurable como para adaptarse a la inserción en otras industrias tanto venezolanas como cubanas, el *middleware* que gestionará la comunicación entre sus procesos distribuidos debe poseer también la flexibilidad necesaria mencionada con anterioridad.

Se debe tener en cuenta además otras importantísimas características de la situación problémica a resolver. Los procesos distribuidos a comunicar serán numerosos, con altas tasas de intercambio de volumen de información debido a la naturaleza propia del monitoreo de las actividades industriales y a la posible existencia de consolas para el despliegue de la información capturada en puntos geográficamente dispersos. Desde estos nodos supervisores en algunas situaciones críticas será

necesario enviar a través del middleware, comandos en tiempo real a la base de datos respectiva, quien a su vez se comunicará con las capas más bajas del SCADA y actuará en correspondencia, por ejemplo cerrando una válvula y evitando así desbordamientos cuyas consecuencias podrían variar en grados de nocividad.

Otro elemento a considerar, una interrupción de las comunicaciones puede resultar desastrosa y provocar serias afectaciones, tanto en la producción de la industria, como al medio ambiente e incluso a recursos humanos. La misma podría implicar desde planes trazados erróneamente hasta la pérdida del envío de alarmas críticas.

Los distintos nodos pueden tanto recibir como enviar información. Los nodos de monitoreo reciben los datos capturados por los PLC y enviados a través del *middleware* por la base de datos de tiempo real pero a su vez le envían comandos a la misma.

A continuación se enumeran las características que en resumen debe poseer la tecnología a escoger para la implementación del *middleware*.

1. Capacidad de comunicación totalmente distribuida.
2. Manejo de alta concurrencia de aplicaciones distribuidas.
3. Manejo de un elevado número de variables, entre puntos y alarmas (50000).
4. Implementación de mecanismos de prioridades para las alarmas.
5. Capacidad de transacción en Tiempo Real para procesos críticos.
6. Comunicación bidireccional.
7. Persistencia en las comunicaciones.
8. Tolerancia a fallas.
9. Implementación de mecanismos de seguridad.
10. Implementación del software de comunicación en lenguaje C++.
11. Permitir desarrollo sobre Linux.
12. Software Libre.

2.3 REVISIÓN Y SELECCIÓN DE LA TECNOLOGÍA MÁS APROPIADA

Existen variadas tecnologías *middleware* posibles a utilizar en la implementación de los 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 de la OMG. Se hace necesario decidir cuál es la idónea según las especificidades de la situación problémica a resolver y las características expuestas con anterioridad.

2.3.1 DESCRIPCIÓN DE LAS TECNOLOGÍAS CANDIDATAS

A continuación se presenta una descripción breve con los rasgos significativos de las filosofías más utilizadas actualmente en el desarrollo de software de comunicación. Serán mencionadas y explicadas de forma general primeramente las especificaciones para luego particularizar en las implementaciones representativas de las más adecuadas.

2.3.1.1 DCOM

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 la cual se le cambio 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 maquinas comunicadas en red, surgió DCOM. Esta es una tecnología que se ha ido desarrollando y actualmente forma parte esencial de ActiveX, Microsoft Transaction Server, COM+ y OPC.

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

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

Desventajas de DCOM

1. Aunque su especificación ha sido liberada las implementaciones más eficientes son cerradas.
2. Aunque posteriormente fue portado a otras plataformas su referencia por excelencia es Windows, las implementaciones de Linux son deficientes.
3. Programar su seguridad es complejo a pesar de poseer niveles bien definidos en la especificación, frecuentemente resulta poco seguro sobre todo en entornos abiertos.
4. 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 con todas las consecuencias que esto trae asociadas.
5. Microsoft sugiere la utilización de nuevas tecnologías para la sustitución de COM y DCOM dentro de las que se puede mencionar COM+ y .NET

2.3.1.2 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 todas complejas tareas internas que tradicionalmente están relacionadas con el desarrollo COM, incluyendo el conteo de referencias, descripción de interfaces y registración.

COM+ se refiere a la versión de COM que combina las características basadas en servicios de *Microsoft Transaction Server* (MTS), con COM distribuido (DCOM). COM+ proporciona 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*) proporciona acceso directo a ellos.

Se han encontrado en Internet únicamente dos implementaciones de .Net como software libre. Entre las cuales el mayor grado de madurez le corresponde Monodevelop. Pero ni Monodevelop ni Portable.Net brindan servicios para el intercambio de información en tiempo real. Además la implementación de Monodevelop está presentando conflictos de licencia. (Creating .Net Applications on Linux and Mac OS X, 2007) (Bollow, 2007)

Características

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

Ventajas

1. Abstracción del lenguaje de programación de los componentes.
2. Administración automática de memoria.
3. Avanzada arquitectura de acceso a datos.

4. Condiciona alta productividad del desarrollador.

Desventajas

1. Solo existen dos implementaciones libres disponibles. Un caso posee un nivel de madurez insuficiente y el segundo caso presenta actualmente conflictos de licencia. Ninguna de las implementaciones brinda servicios para facilitar la comunicación entre procesos en tiempo real.

2.3.1.3 RMI

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 la misma. Cuando un cliente adquiere la referencia de un objeto, invoca sus métodos con total transparencia a su ubicación, su espacio de direcciones puede estar en cualquier lugar de la red. La referencia de objetos es brindada a los clientes en forma de direcciones **URL** y ellos acceden a la invocación del objeto servidor de la misma forma que si accedieran a una dirección Web.

RMI forma parte de Java estándar y viene incluido en los **JDK** de Sun desde la versión1.1

Características propias de Java/RMI

- Transparencia
- Eficiencia
- Seguridad
- Funcionalidad

Ventajas de RMI

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

Desventajas de RMI

1. Su empleo más óptimo es utilizando lenguaje Java para la implementación. Existen mecanismos para adaptarlo a otros lenguajes utilizando CORBA pero el proceso se hace tedioso, propenso a fallas y sus resultados no son tan eficientes.
2. 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, condicionando varios aspectos de la solución.
3. Depende de una máquina virtual Java para su ejecución.
4. Al utilizar java, la ejecución es considerablemente lenta, comparada con implementaciones en lenguajes compilados como C o C++. (Herrera Vázquez, 2007)

2.3.1.4 CORBA

Siglas de *Common Object Request Broker*, 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) que define las diferentes API, el protocolo de comunicaciones y los mecanismos necesarios para permitir la interoperabilidad entre diferentes aplicaciones escritas en diferentes lenguajes y ejecutadas en diferentes plataformas, lo que es fundamental en computación distribuida.

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

CORBA utiliza un lenguaje de definición de interfaces (IDL) para especificar las interfaces con los servicios que los objetos ofrecerán. CORBA puede especificar a partir de este IDL la interfaz a un lenguaje determinado, describiendo cómo los tipos de dato CORBA deben ser utilizados en las implementaciones del cliente y del servidor. Implementaciones estándar existen para Ada, C, C++, Smalltalk, Java y Python. Hay también implementaciones para Perl y TCL. Al compilar una interfaz en IDL se genera código para el cliente y el servidor (el 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.

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

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 de OMG ha definido dieciséis de estos servicios.

- 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, 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

1. Interfaz independiente del lenguaje de programación
2. Integración de herencia
3. Infraestructura de objetos distribuidos
4. Transparencia en la localización
5. Transparencia en la red
6. Comunicación directamente con los objetos
7. Interfaz de invocación dinámica

Desventajas de CORBA

1. Complejidad
2. Incompatibilidad entre implementaciones

2.3.1.5 DDS

DDS es una especificación realizada por RTI (Real Time Investigations) y aprobada por 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.

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

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

Desventajas de DDS

1. No tiene un mecanismo de seguridad muy eficiente.

Está en constante desarrollo, las implementaciones existentes aun adolecen de varias características esenciales de los *middleware* para sistemas distribuidos.

2.4 COMPARACIÓN DE TECNOLOGÍAS

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 .Net.

Las aplicaciones elaboradas con el middleware Java/RMI deben ser desarrolladas con Java parcial o completamente. Por su parte DCOM y .Net limitan la plataforma de desarrollo a Windows debido a las desventajas identificadas en las implementaciones de entornos libres. Tales rasgos van en detrimento de los requerimientos tecnológicos

del middleware para el sistema SCADA PDVSA expuestos en análisis previos. Ha quedado planteada de forma expedita la necesidad de desarrollar utilizando únicamente software libre, estándares abiertos, y C++ como lenguaje de implementación.

Sendas especificaciones restantes (CORBA y DDS), tanto la “Arquitectura común de intermediarios en peticiones a objetos”, como el “Servicio de distribución de datos”, ofrecen soporte a todos los requerimientos tecnológicos básicos y generales identificados en epígrafes anteriores. 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.

La realización de una búsqueda en Internet arrojó la existencia de al menos 25 implementaciones funcionales de CORBA (ANEXO: **A1**). De ellas, sólo cinco satisfacían los requisitos 10, 11 y 12 enunciados en el epígrafe “Requerimientos tecnológicos del middleware”. En otras palabras, sólo las implementaciones Orbit, MICO, OmniORB y TAO cumplen a la vez con: clasificar como software libre, soportar la implementación para la plataforma linux y posibilitar el desarrollo utilizando el lenguaje C++.

A continuación se analizará cada una con más detalle.

2.4.1 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*. 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).

2.4.2 TAO

TAO ORB fue desarrollado por el *Distributed Object Computing Group* en la Universidad de Washington bajo la dirección del doctor Douglas Schmidt. La versión 1.0 de TAO es diseñada para compilar con CORBA 2.2 e incluye algunos aspectos de CORBA 2.3. Inicialmente fue pensado para tiempo real, pero ha llegado a ser un ORB de propósito general muy capaz, adaptable a cualquier tipo de sistema distribuido sea de tiempo real o no. TAO es ampliamente utilizado por los desarrolladores ofreciendo capacidades insustituibles en los servicios de CORBA que implementa. Actualmente TAO 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 de RT-CORBA lo cual le permite su funcionamiento en sistemas de tiempo real con una eficiencia muy alta.

2.4.3 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.

2.4.4 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. ORBit soporta POA, DII, DSI, *TypeCode*, *Any*, IR e IIOP. ORBit fue elaborado para la comunicación en los entornos gráficos del proyecto GNOME y 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 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.

2.4.5 AL TOMAR EL INTERCAMBIO DE DATOS COMO CENTRO

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 que garantizará el necesario intercambio de datos del SCADA. Pero 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 del sistema SCADA PDVSA.

La implementación de DDS realizada sobre el ORB de ACE, aunque brinda altas 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. La versión actual de dicho *framework* adolece aún de prestaciones necesarias dada la naturaleza crítica del proceso de intercambio de información que instrumentará el middleware a 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 las garantiza. 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. (Herrera Vázquez, 2007)

2.4.6 SELECCIÓN Y JUSTIFICACIÓN DE LA TECNOLOGÍA MÁS APROPIADA

Se decide no utilizar la implementación de DDS sobre el ORB de ACE dada la relevancia de los inconvenientes descritos con anterioridad. Adicionalmente 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” (Herrera 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 del *middleware* 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 de la posibilidad de utilizar, en versiones futuras, DDS, ya que aunque el desarrollo de TAO DDS, se presenta poca madurez podría perfilarse como una opción a tener en cuenta si superan sus actuales limitaciones.

2.5 ANÁLISIS DEL *NOTIFICATION SERVICES* DE TAO. MODELO PROVEEDOR/CONSUMIDOR

De forma estándar, CORBA solicita resultados en la ejecución sincrónica de una operación. Si la operación tiene definidos parámetros o valores de retorno, se comunican dichos datos entre el cliente y el servidor. El pedido es dirigido a un objeto específico. Tanto el cliente como el servidor deben estar disponibles para que la solicitud pueda efectuarse exitosamente. En otras palabras, por omisión el modelo utilizado en CORBA para las comunicaciones es el de cliente/servidor: desde un "objeto cliente" se realiza una llamada a un "objeto servidor". Generalmente dicha comunicación es de tipo "uno a uno" o "punto a punto". Pero en algunas circunstancias el esquema descrito no brinda suficiente desacoplamiento. En contraste en el modelo Publicador/Suscriptor, una aplicación envía mensajes a un tópico específico y todas las aplicaciones “suscritas" a dicho tópico reciben el mensaje. En este caso la comunicación es de tipo "uno a muchos", e intrínsecamente asíncrona porque la aplicación que publica un mensaje no espera respuesta de las aplicaciones receptoras.

El Servicio de Eventos de CORBA (*Event Service*) brinda una forma básica del modelo de Publicación/Suscripción. El Servicio de Notificación (*Notification Service*) extiende al Servicio de Eventos brindado una implementación superior del modelo, eliminando las limitaciones del Servicio de Eventos, e incorporándole funcionalidades adicionales.

Corba utiliza una terminología un tanto distinta a la expuesta con anterioridad. En lugar de Publicador y Suscriptor utiliza *Supplier* y *Consumer* respectivamente. En lugar de tópico utiliza el término: *event channel*.

2.5.1 COMUNICACIÓN POR EVENTOS

El Servicio de Notificación desacopla las comunicaciones entre los objetos. Define sendos roles para los objetos: Proveedor y Consumidor respectivamente. Los proveedores producen datos en forma de "Eventos" y los consumidores procesan dichos datos. Los datos eventos son transmitidos entre publicadores y suscriptores a través de solicitudes estándares de CORBA.

Existen dos enfoques de forma general para iniciar la comunicación de eventos entre proveedores y consumidores, "*Push*" (presión) y "*Pull*" (atracción). El modelo de presión permite al proveedor de eventos iniciar la transferencia de los datos eventos a los consumidores. El modelo de atracción permite a los consumidores, solicitar los datos eventos al proveedor. En el modelo *Push* el proveedor toma la iniciativa mientras que en el modelo *Pull* lo hace el consumidor.

Un canal de eventos es un objeto intermediario que posibilita la comunicación de múltiples proveedores con múltiples consumidores de forma asíncrona. Es además, tanto un consumidor como un proveedor. La interacción con dicho objeto estándar de CORBA se realiza por el mecanismo estándar de comunicación de la *Common Object Request Broker Architecture*.

2.5.2 EVENTOS

Cada aplicación que use el Servicio de Notificación, ya sea proveedora o consumidora, puede utilizar un tipo de dato entre tres posibles: *Any*, *StructuredEvent* y *EventBatch*, para la realización de sus actividades respectivas.

El tipo de dato *Any*, como su nombre lo indica es un tipo general que permite el almacenamiento de cualquier otro tipo de dato.

El *StructuredEvent* está compuesto a grandes rasgos por tres campos. En el primero, en el *EventHeader*, es posible enviar de forma embebida, políticas de calidad de servicio (**QoS**) con el objetivo de configurar la entrega, por ejemplo definiendo la

prioridad del mensaje o su tiempo límite. Su segundo campo está constituido por una secuencia de pares nombre - valor. Allí puede ser guardada la cantidad deseada de datos del evento. Este campo clasifica como "filtrable" porque puede ser accedido por los “filtros". El tercer y último campo de forma general es de tipo *any* y su objetivo es contener los datos que nos se desean "filtrar". Tal pudiera ser el caso de los ficheros debido a su tamaño.

El *EventBatch* es un tipo de dato constituido por una secuencia de *StructuredEvent*. Permite a las aplicaciones de elevadas tasas de transferencias enviar o recibir eventos en masa.

2.5.3 FILTROS

Un filtro es un objeto que encapsula una colección de restricciones o condiciones. Una restricción o condición es una expresión *Booleana* escrita en términos de pares nombre-valor dentro de un evento estructurado.

Los filtros pueden ser aplicados a los mensajes a su paso por el Servicio de Notificación. Sintácticamente existen dos tipos: el *Filter* y el *Mapping-Filter*.

El propósito del tipo *Filter* es borrar los mensajes antes de su transmisión a los consumidores. Contribuyendo con menor tráfico en la red, y a la vez posibilitando la llegada a cada consumidor, únicamente de los mensajes de su interés.

Los filtros pueden adjuntarse a todas las combinaciones de objetos "*proxy*", *push*/*pull*, y consumidor/proveedor. Aunque es más natural filtrar lo que recibe el consumidor.

El *Mapping-Filter* por su parte, se utiliza para sobrecargar la línea límite de tiempo y la prioridad asociadas a un mensaje. Está constituido por una encapsulación de pares restricción - valor.

En orden de administrar los filtros a niveles de grupo se utilizan los objetos *SupplierAdmin* and *ConsumerAdmin* respectivamente.

2.5.4 POLÍTICAS DE CALIDAD DE SERVICIO

El servicio de notificación permite a sus usuarios, escoger la política de calidad de servicio deseada para la transmisión de sus eventos. Un ejemplo de ello es que:

- Es posible crear varios canales, donde cada canal puede utilizarse para transmitir tipos de eventos diferentes. Utilizando las QoS cada canal creado puede ser configurado como *persistent* o *best effort* (que es la nominación brindada por el Servicio de Notificación para referirse a la "no persistencia") Que un canal sea *persistent* significa que toda su infraestructura y todos sus “eventos por entregar" serán almacenados en un fichero o en una base de datos para poder recurrir a ellos en caso de reinicio de los procesos, o falla del sistema.
- Un canal *best-effort* sólo puede ser utilizado para transmitir únicamente eventos *best-effort*, en contraste, un canal *persistent* puede enviar datos tanto de tipo *persistent* como *best-effort*.
- Existen variados valores de QoS posibles a utilizar en vista de controlar con cuánta firmeza el Servicio de Notificación intenta entregar un evento a un consumidor antes de renunciar y descartar el evento:
    - Los eventos aún no entregados pueden hacerse llegar a los consumidores en los órdenes: "FIFO" (el primero en llegar es el primero en salir), "por prioridad" (atendiendo al valor opcional de la prioridad como campo del mensaje), "por límite de vida" (basado en el valor opcional de límite de tiempo de vida del encabezado del evento) o sin importar el orden.
    - Es posible establecer un límite de eventos a poner en cola para ser enviados a al consumidor.
    - Si el servicio de notificación está próximo a los límites máximos de número de eventos por entregar a un consumidor, se comenzarán a descartar eventos. La política para la decantación es similar a la política de entrega de los datos a los consumidores.
    - En orden de despachar una entrega fallida, es posible configurar el tiempo a esperar por el Servicio de Notificación antes de intentar el reenvío del evento.

  - El número máximo de reintentos antes de renunciar y descartar el envío de un evento.
  - Un límite de tiempo de invocación de una operación en el consumidor para la entrega de un evento.
- Es posible establecer límites máximos a las cantidades de proveedores y consumidores que pueden permanecer conectados al servicio de notificación.

Estas políticas de calidad de servicio pueden ser sobrecargadas en otros contextos.

2.5.5 FIABILIDAD Y PERSISTENCIA

Una de las funcionalidades opcionales del Servicio de Notificación es la Fiabilidad. Por omisión el servicio de notificación y el servicio de eventos brindan un soporte *best-effort* en la entrega de eventos. Si las cosas van mal (ocurren faltas inesperadas, fallan las comunicaciones, etc.) los eventos se perderán sin el acaecimiento de notificación alguna.

Existen algunas circunstancias en las que la pérdida de eventos no es aceptable. El servicio de notificación debe utilizarse en tales situaciones si la operación de fiabilidad es configurada. La operación de fiabilidad no se encuentra disponible en el servicio de eventos. Operación fiable significa que la información será guardada de forma persistente (usualmente en un fichero en el disco) y utilizada para la recuperación de las variadas fallas que pueden de lo contrario conllevar a la pérdida de datos.

Existen dos puntos relacionados con la entrega de eventos: las persistencias de la topología y de los eventos.

En orden de brindar persistencia de la topología, conocida también como persistencia de las conexiones, el servicio de notificación debe conservar registros de qué clientes (ya sean Proveedores o Consumidores) se han conectado al servicio de notificación y qué opciones han especificado para el control de las entregas.

Para conservar persistencia de los eventos el servicio de notificación almacena cada evento en un medio persistente asegurando su entrega a cada consumidor que debe recibirlo.

Existen situaciones en que sólo es necesaria la persistencia de las conexiones (cuando puede ser aceptable la pérdida de eventos durante una falla siempre que el sistema sea restaurado a sus operaciones normales después de la misma). Por otro lado la persistencia de los eventos sólo puede obtenerse de forma conjunta a la persistencia de la topología. No cumple objetivo almacenar los eventos si el sistema no es capaz entregarlos luego a sus respectivos consumidores.

Para establecer la función de fiabilidad en las operaciones del servicio de notificación es necesario actuar en dos direcciones. A nivel de administración del sistema, el servicio de notificación debe ser configurado para la persistencia de las conexiones y si se desea, para la persistencia de los eventos. A nivel de aplicación, los programas que operan como consumidores y proveedores, deben utilizar los parámetros apropiados para habilitar la función de fiabilidad, y deben cooperar con el proceso de re-conexión que ocurre durante la recuperación de las conexiones.

Tras la realización de un análisis de los requerimientos tecnológicos del *middleware* para el SCADA PDVSA, se llevó a cabo una revisión de las principales tecnologías existentes en la actualidad para el desarrollo de aplicaciones de comunicación en sistemas distribuidos.

Se determinó al *framework* ACE+TAO como el ORB más adecuado entre las diversas opciones. Sus índices de eficiencia, madurez y la implementación del estándar de tiempo real de CORBA que realiza fueron las principales características que conllevaron a su elección.

Una vez vencida la primera etapa con las definiciones preliminares y necesarias, se puede proceder con el diseño del middleware básico teniendo en cuenta la tecnología escogida para su desarrollo y las características del software de comunicación hasta el momento analizadas.

Capítulo III DISEÑO DEL *MIDDLEWARE* BÁSICO CON ACE+TAO

3.1 INTRODUCCIÓN

Las preguntas generales a cuyas respuestas se pretende arribar durante el presente capítulo son: ¿Qué funcionalidades básicas debe poseer el *middleware* del Sistema Supervisor de Procesos Automatizados de Petróleos de Venezuela S.A.? ¿Qué diseño de *middleware* da respuesta a las necesidades de comunicación básicas de forma eficiente, segura y brindado además flexibilidad, predictibilidad y mantenibilidad?

3.2 REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES DEL *MIDDLEWARE* BÁSICO

A continuación se presentarán los requerimientos básicos del módulo de comunicación del SCADA. Los mismos están en correspondencia con la situación problémica y los conceptos expuestos. Servirán de punto de partida al diseño de una biblioteca que garantice las funcionalidades primarias del intercambio de información entre los diferentes módulos de las aplicaciones para el control y supervisión de los procesos automatizados. Seguidamente serán enumerados los requerimientos no funcionales tenidos en cuenta para la elaboración del diseño.

Glosario

Persistencia de las comunicaciones: Al ocurrir una interrupción abrupta, no planificada, o un reinicio forzoso de un nodo receptor o emisor, el sistema debe conservar el último estado normal de las comunicaciones (entre cuáles nodos se efectuaba), para lograr su restablecimiento tras la re-conexión del nodo al sistema.
Persistencia de los datos: Al detectar problemas en la comunicación, el sistema debe conservar los últimos datos emitidos para realizar su reenvío a los interesados en su recepción.

Nodo emisor: Módulo del sistema con necesidad de transmitir la información a su alcance, utilizando el *middleware,* a los restantes módulos del sistema que hallan demostrado interés en la misma.

Nodo receptor: Módulo con necesidad de recibir determinada información proveniente de otros módulos del sistema a través del middleware.

Datos: Los datos identificados en los SCADA incluyen los dos tipos básicos priorizados en el presente diseño: alarmas y puntos.

Requerimientos Funcionales:

RF1 Brindar la funcionalidad de propagar datos desde los nodos emisores hacia los nodos receptores.

RF1.1 Posibilitar el registro de los nodos según sus características receptoras, emisoras o duales.

RF1.2 Garantizar el envío de datos de los nodos registrados como emisores.

RF1.3 Permitir la especificación del tipo de dato a recibir.

RF1.4 Garantizar la recepción de los datos por los nodos registrados como sus receptores.

Requerimientos No Funcionales

- Proporcionar la posibilidad de que el envío y recepción se realicen en Tiempo Real.
- Garantizar la persistencia de los datos.
- Garantizar la persistencia de las comunicaciones

Los requerimientos funcionales vistos previamente quedarían representados como los siguientes casos de uso del sistema.

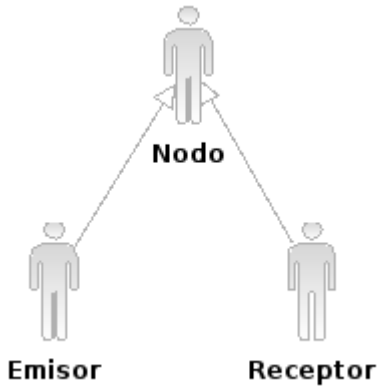

Figura 4 *Relación entre los actores del sistema.*

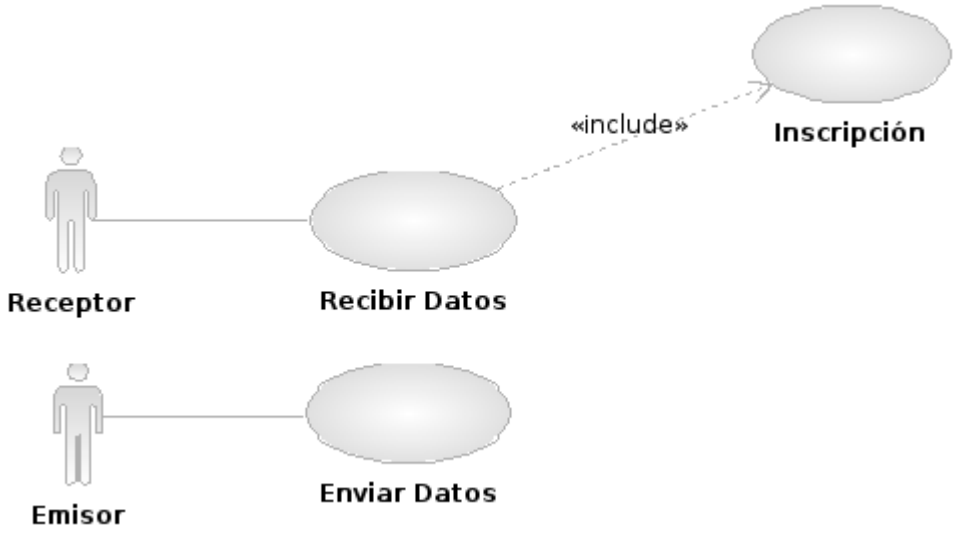

Figura 5 *Diagrama de Casos de Uso del sistema.*

Las descripciones de alto nivel de los Casos de Uso representados son las sucedidas a continuación.

Caso de Uso: Enviar datos.

Actor: Emisor

Tipo: Primario

Descripción: El nodo interesado en enviar datos comienza a emitir los datos utilizando el *middleware* y permanece realizando dicha acción mientras lo requiera.

Caso de Uso: Recibir datos

Actor: Receptor

Tipo: Primario

Descripción: El nodo interesado en recibir determinado tipo de datos se inscribe al servicio especificando qué tipo de datos desea recibir y proporcionando además, la información de configuración requerida por la aplicación para el establecimiento de la comunicación y entrega exitosa de los datos. Tras la culminación del proceso de inscripción el receptor puede informar al *middleware* de su aceptación con respecto al arribo de los datos. A partir de ese momento los datos emitidos por otros módulos y cuyos tipos hayan sido contemplados en el proceso de inscripción, llegarán certeramente al receptor. En todas las ocasiones después de cada inicio del flujo de datos, se brindará la opción de detener el flujo de los datos a exigencias de cada Receptor. Por esta vía se dejaría de recibir la información enviada por los módulos emisores. También ocurre a la inversa, cada vez que el flujo de datos haya sido detenido a demanda de los receptores, podrá también ser iniciado a demanda de ellos.

3.3 MODELO DEL DISEÑO DEL *MIDDLEWARE* BÁSICO

3.3.1 CENTRADO EN LA ARQUITECTURA

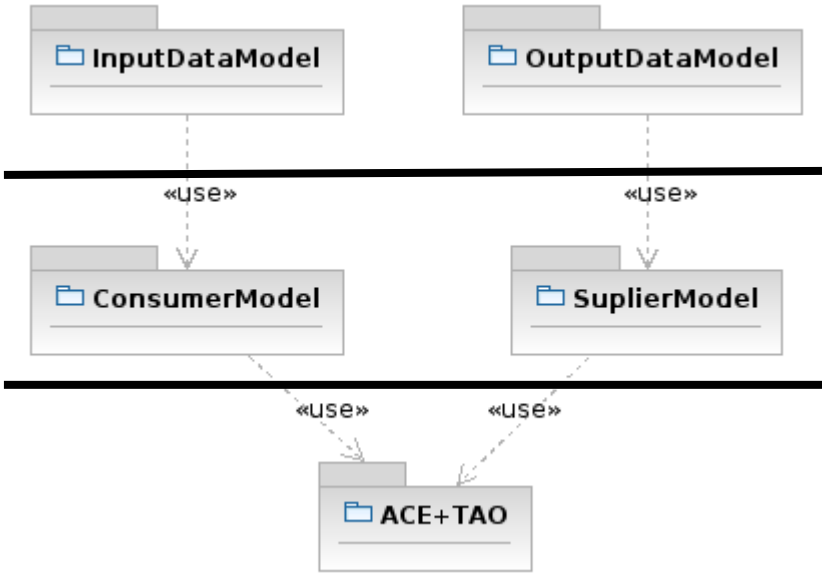

Figura 6 *Paquetes de clases del middleware diseñado por niveles*

El diseño efectuado se encuentra estructurado por cinco paquetes en tres niveles diferentes. Proporcionando la flexibilidad y mantenibilidad características de la aplicación del patrón N-Capas o *n-tier*. El nivel más bajo quedaría constituido por las clases y estructuras del *framework* seleccionado con anterioridad para el desarrollo del software de comunicación. Tal característica posibilita, a una nueva compilación del código sin mayores percances, la actualización de la tecnología *middleware* utilizada mientras sean conservados el contenido de los ficheros cabeceras y todas las interfaces. El nivel de ACE+TAO proporciona el primer conjunto de abstracciones y funcionalidades generales importantes para el desarrollo, como por ejemplo, interfaces más adecuadas para la utilización del tiempo real.

El nivel intermedio brinda un mayor grado de abstracción, posibilitando a la capa superior la interacción mediante el modelo productor/consumidor. Por lo tanto, en este punto es posible sustituir la tecnología *middleware* empleada por una homóloga en cuanto al paradigma, a través de la realización de muy pocos cambios en el código fuente y su posterior compilación. Tales cambios no tendrían repercusión alguna a más altos niveles.

Finalmente, la capa superior proporciona interfaces sencillas a los distintos módulos según su rol de receptor, emisor o dual.

De abajo hacia arriba, los niveles primero y segundo conformarían un componente del sistema, en específico, una biblioteca compartida. Tal recurso lo emplearía una segunda biblioteca conformada por las clases del tercer nivel (nivel superior).

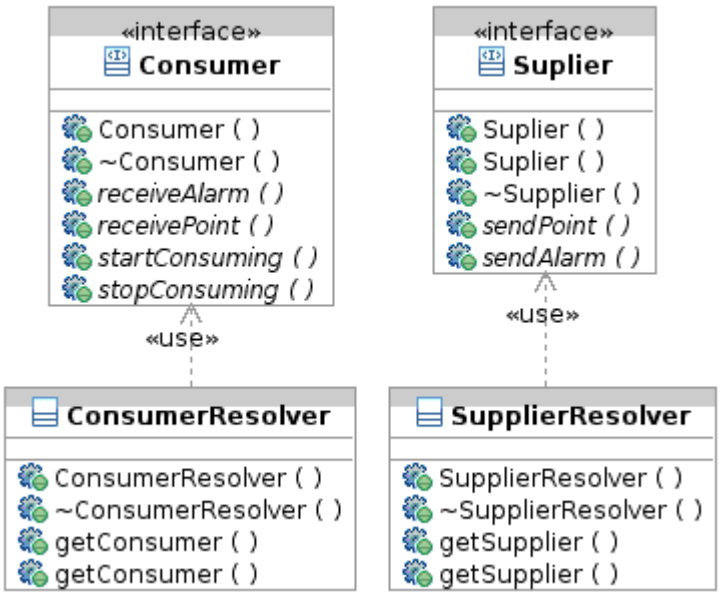

Figura 7 *Interfaces entre los niveles medio e inferior del middleware diseñado.*

La imagen vista con anterioridad ilustra las formas de interacción entre los niveles superior e intermedio. A través de métodos públicos de las clases *SupplierResolver* y *ConsumerResolver* se obtienen punteros a sendos objetos de tipo interfaz para el envío y recepción de datos respectivamente. El desacoplamiento se logró en este caso a través del polimorfismo. Al invocar a los métodos de las interfaces *Supplier* y *Consumer* se ejecutarían las implementaciones de los mismos realizadas por las clases *TAOSupplier* y *TAOConsumer* para cada caso. El último par de clases mencionado debe ser desconocido en su totalidad por el nivel superior, conocedor únicamente de las interfaces de la capa intermedia. (ANEXO: **A2**)

La subsiguiente figura, muestra las interfaces externas, dígase entre el nivel superior y los restantes módulos del sistema. Emplea un mecanismo similar al expuesto con anterioridad para la obtención del desacoplamiento.

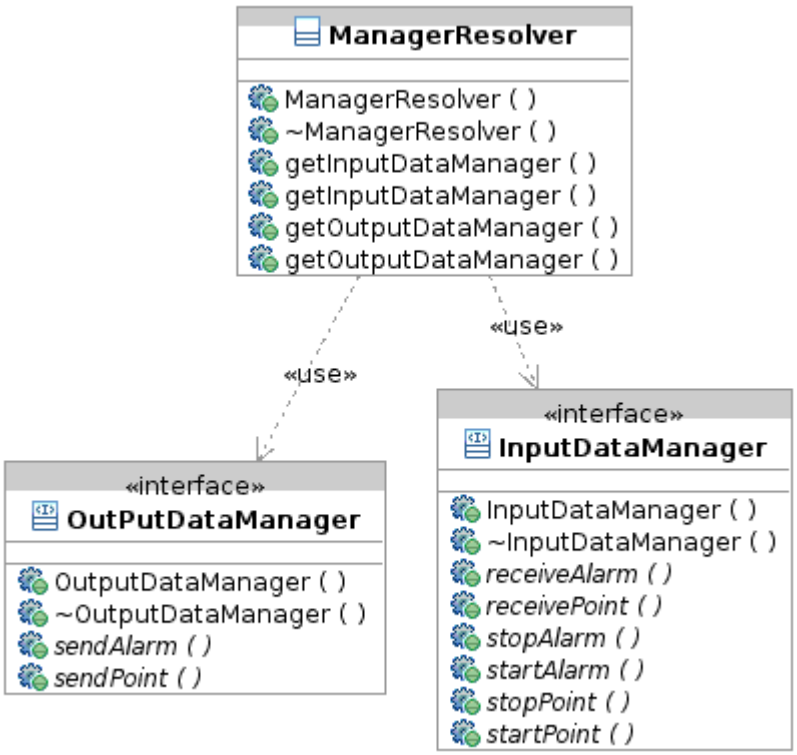

Figura 8 *Interfaces entre los niveles medio y superior del middleware diseñado*

3.3.2 DIRIGIDO POR CASOS DE USO

3.3.2.1 CASO DE USO: RECIBIR DATOS

Las clases del diseño del caso de uso: Recibir datos, quedarían relacionadas como se muestra a continuación.

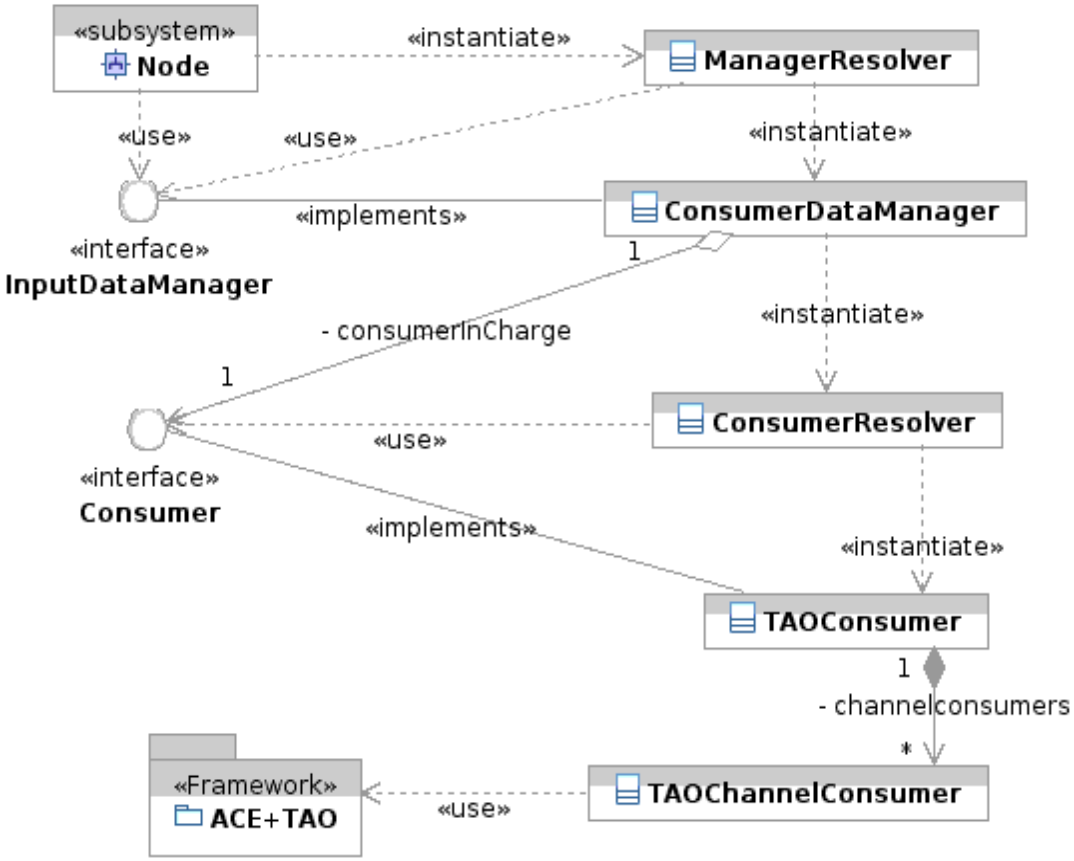

Figura 9 *Relación entre las clases del diseño del caso de uso Recibir datos.*

En la Figura 3.3 se encuentran ausentes dos clases pertenecientes a dicho diseño. No se muestran a las clases Punto y Alarma correspondientes a los tipos de datos que transmitirá el *middleware* en la imagen previa por problemas de legibilidad. Seguidamente sendos diagramas de clase reflejarán las relaciones de los tipos de datos Punto y Alarma con el resto de las clases pertinentes al desarrollo del caso de uso Recibir datos.

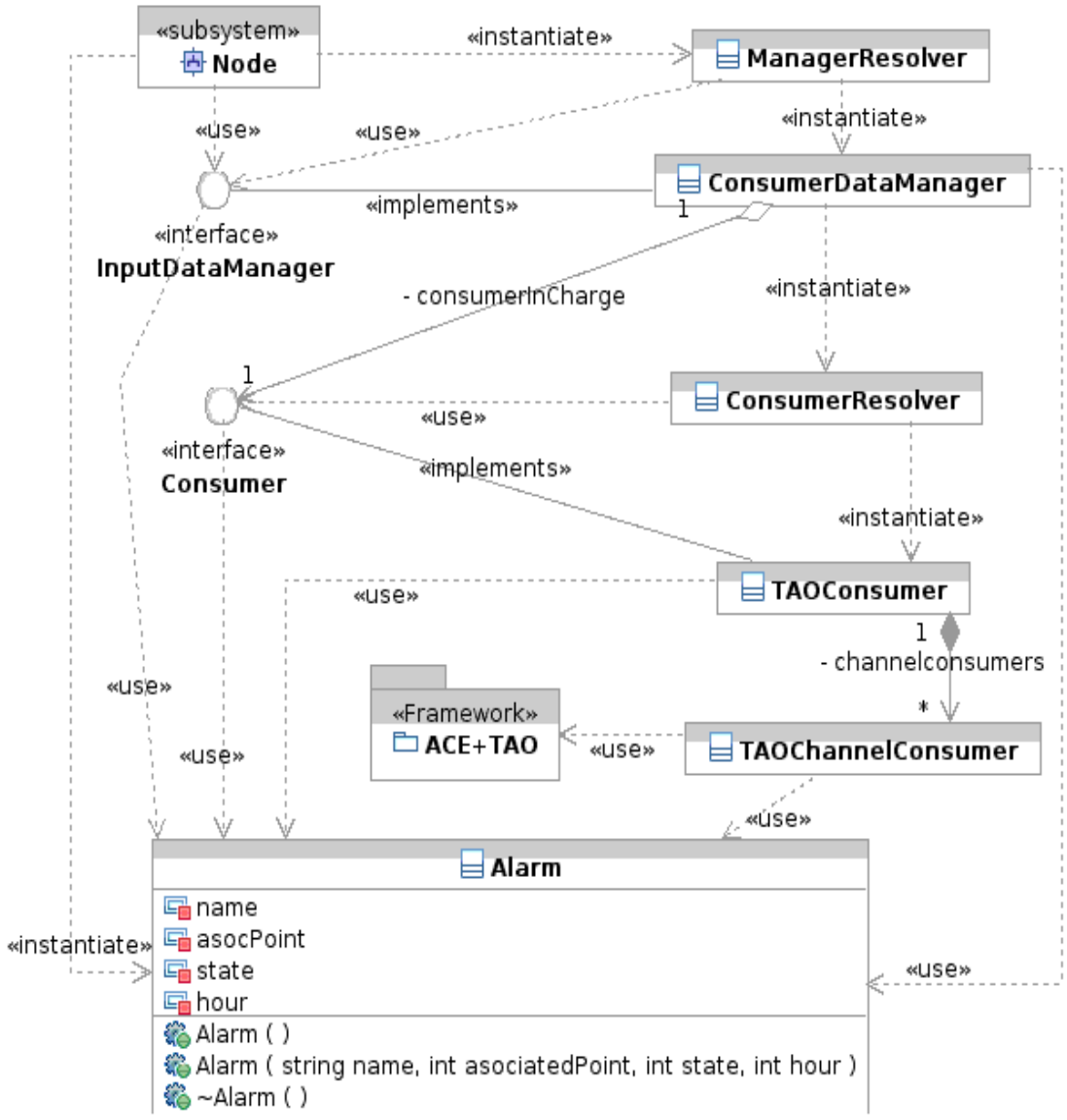

Figura 10 *Relación entre la clase "Alarm" y las clases del diseño del caso de uso Recibir datos.*

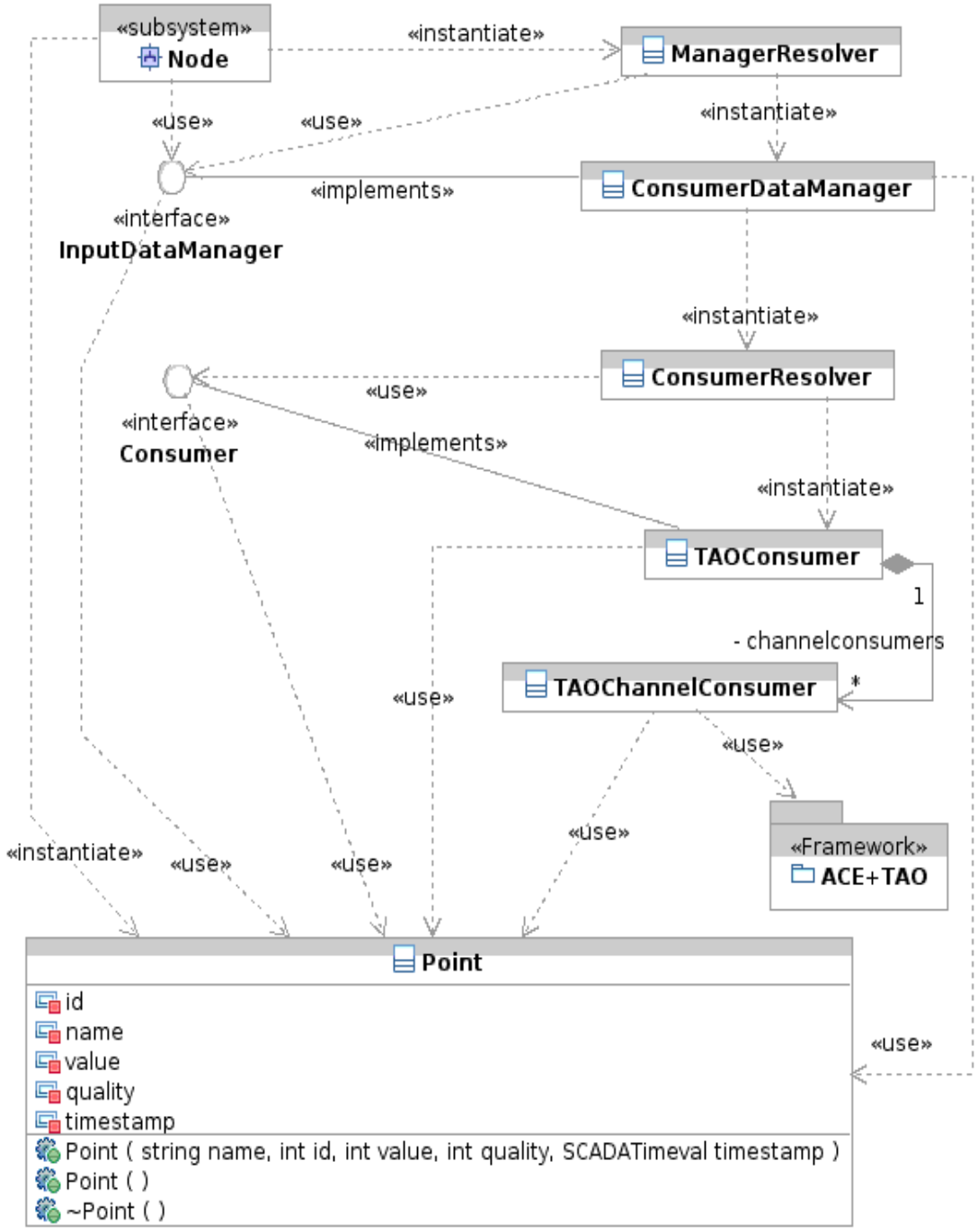

Figura 11 *Relación entre la clase "Point" y las clases del diseño del caso de uso Recibir datos.*

Diagramas de Clase

Seguidamente se irán presentando, cada una de las partes del diagrama del diseño correspondiente al caso de uso “Recibir datos” con un mayor grado en detalles. Los fragmentos se irán organizando por niveles, comenzando por la biblioteca de la capa más alta.

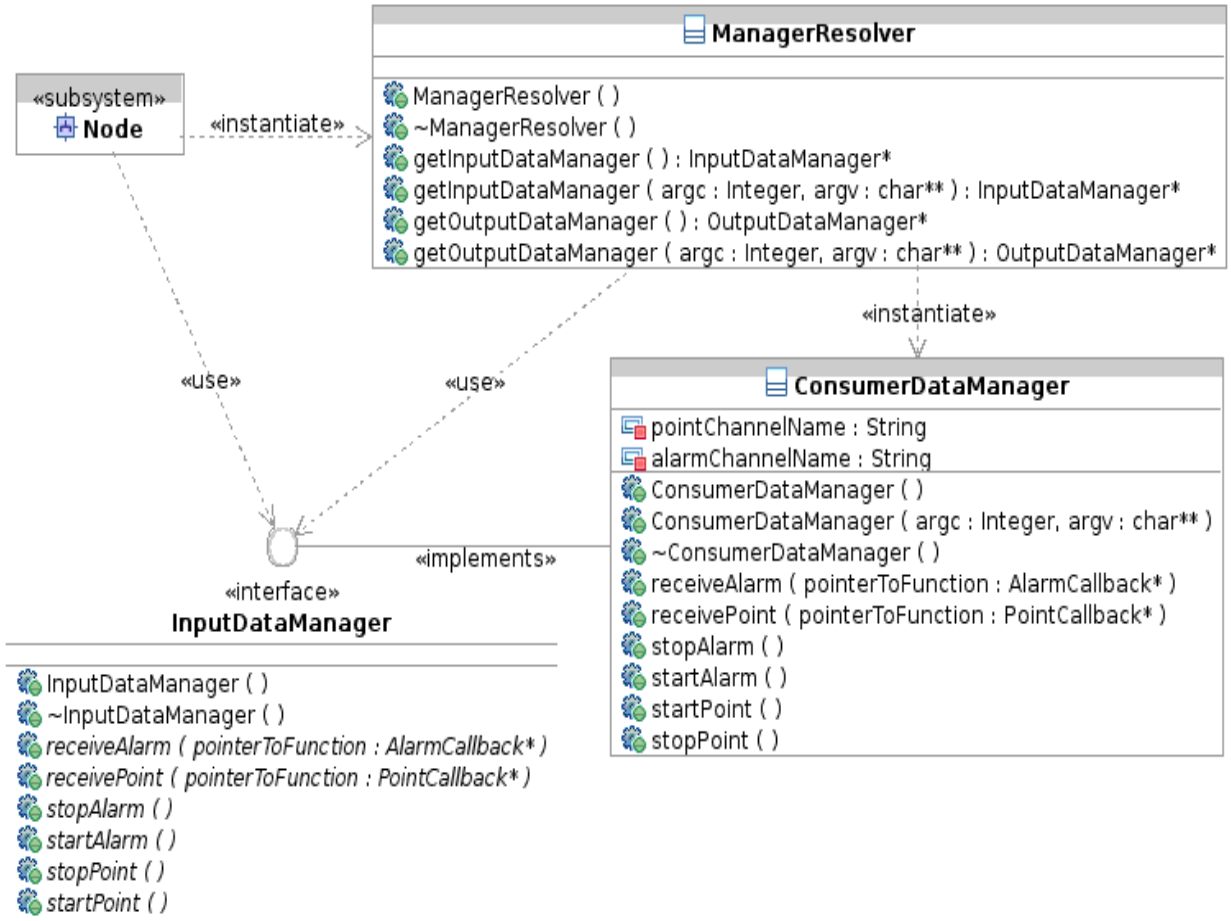

Figura 12 *Clases e interfaces del nivel superior del middleware del paquete InputDataModel en comunicación con otros módulos del SCADA.*

A través de la interfaz *InputDataManager*, se brinda un mecanismo de alto desacoplamiento entre los nodos receptores de datos y el *middleware*. Al invocar a las funciones *receiveAlarm* y *receivePoint* los módulos interesados en recibir datos iniciarán el proceso de “Inscripción”. A partir de este momento el *middleware* quedará informado sobre los tipos de datos que los nuevos receptores deben recibir, así como de las funciones que deben ser invocadas (pasadas como parámetro a los respectivos métodos) para la realización de cada entrega. El invocar una función, ya sea de tipo *AlarmCallback* o *PointCallback* perteneciente a los receptores, les

proporcionará a estos últimos la oportunidad de actuar según necesiten a la llegada de cada dato de su interés. El *middleware* no requiere conocimiento alguno sobre las interioridades de sus receptores ni de las operaciones a ejecutar por ellos a la llegada de cada dato. La responsabilidad del software de comunicación culminará con la invocación de los punteros a función proporcionados por los módulos interesados en la recepción.

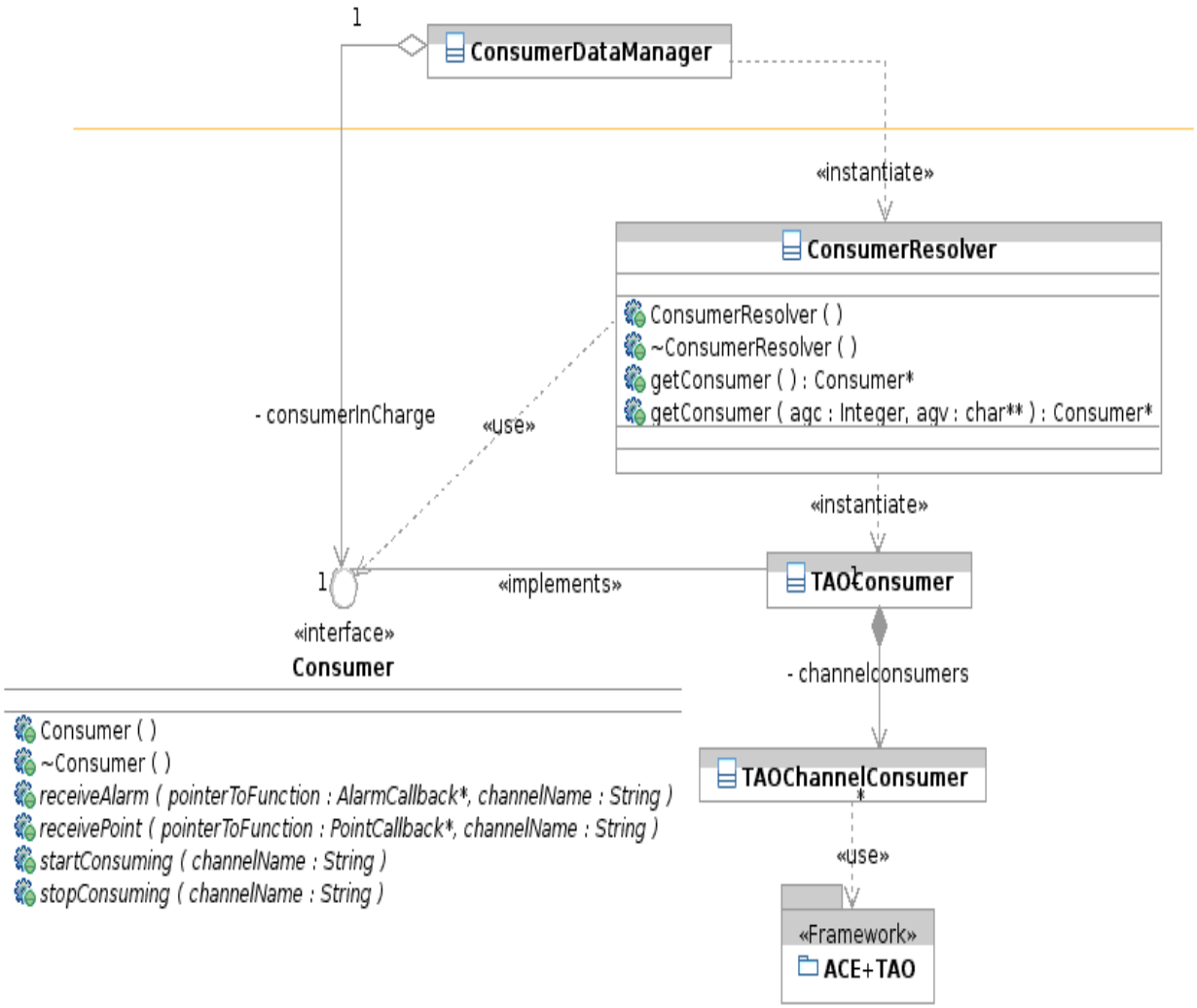

Figura 13 *Clases e interfaces del nivel más bajo del middleware pertenecientes al paquete ConsumerModel. Detallando en Consumer y ConsumerResolver*

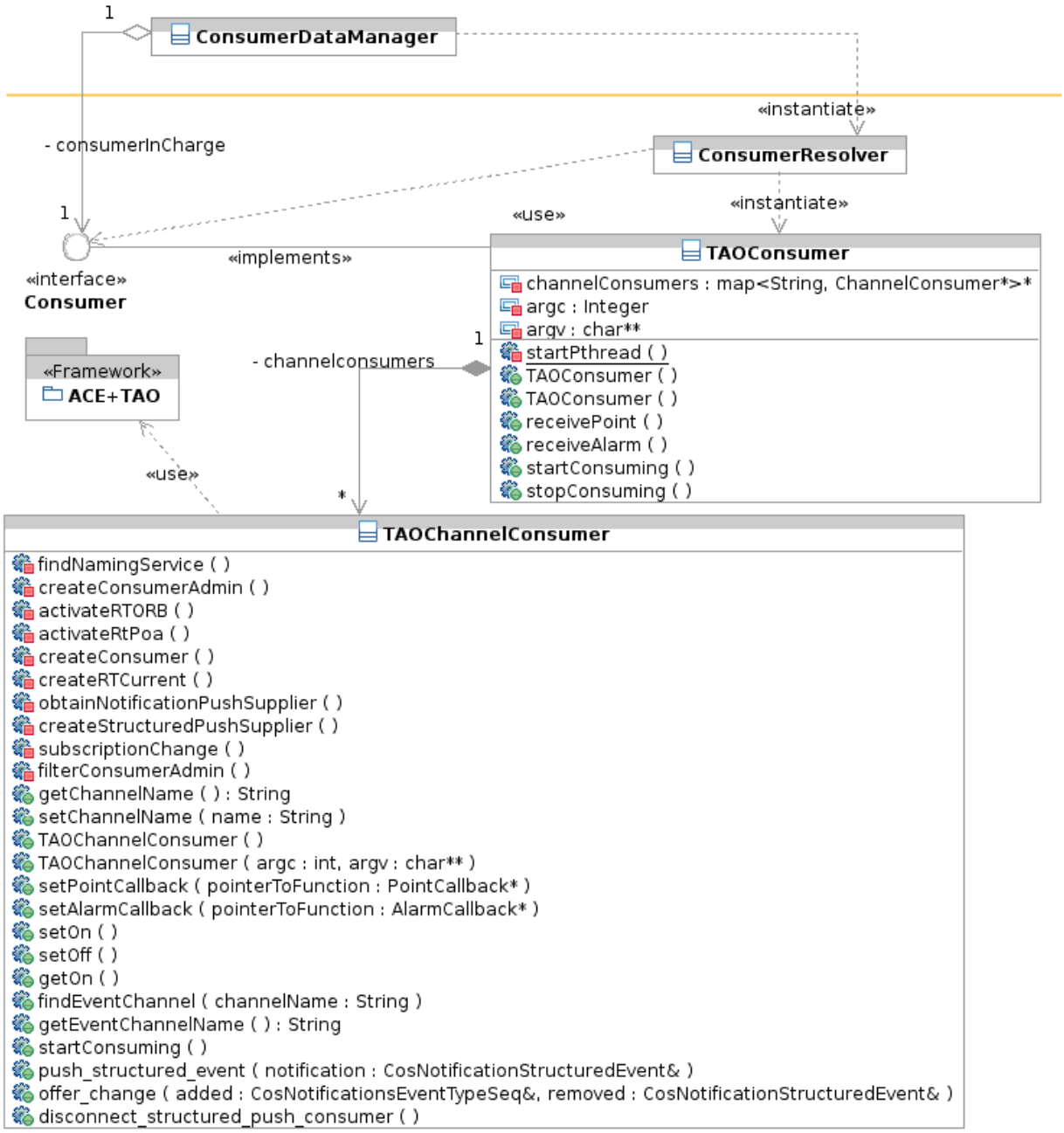

Figura 14 *Clases e interfaces del nivel más bajo del middleware pertenecientes al paquete ConsumerModel. Detallando en las clases TAOConsumer y TAOChannelConsumer.*

Diagramas de Secuencia

A continuación con el objetivo de alcanzar una mejor comprensión, se presentan los diagramas de secuencia correspondientes a las interrelaciones diseñadas para el caso de uso Recibir Datos.

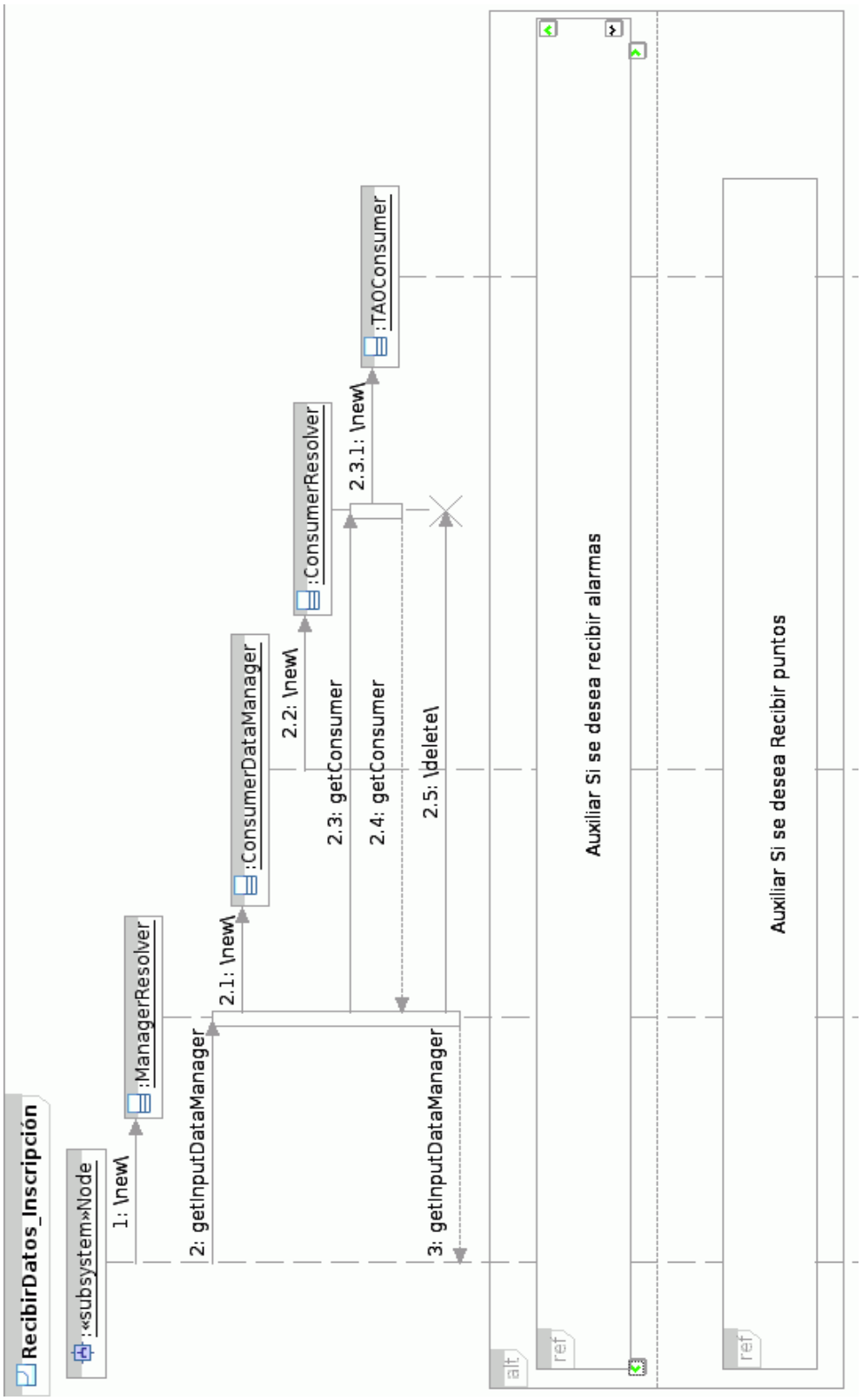

Figura 15 *Primera parte del diagrama de secuencia del caso de uso incluido Inscripción.*

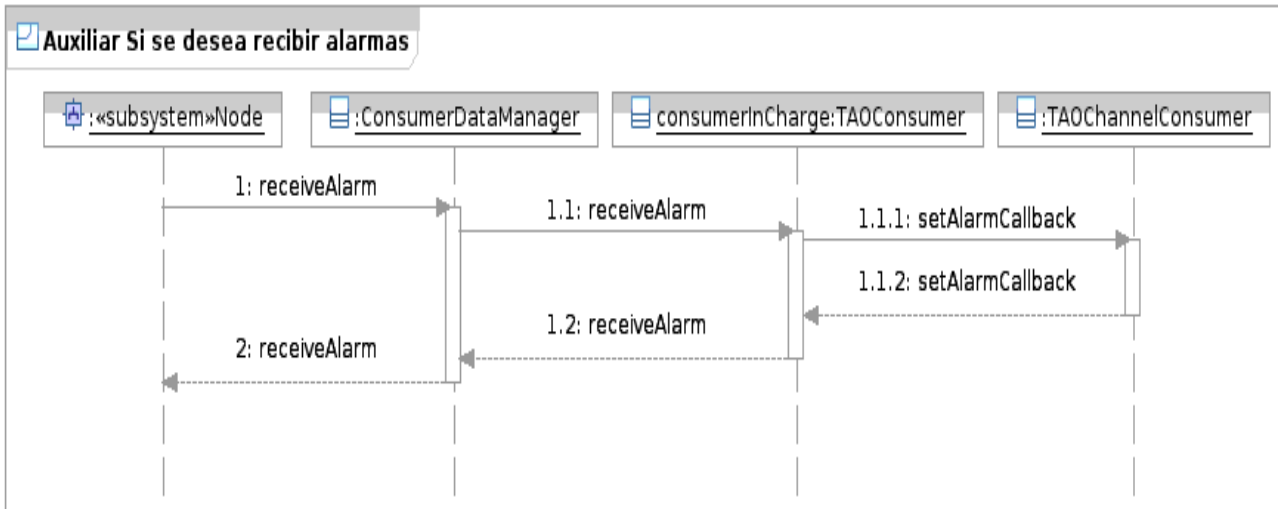

Figura 16 *Segunda parte del diagrama de secuencia del caso de uso incluido Inscripción.*

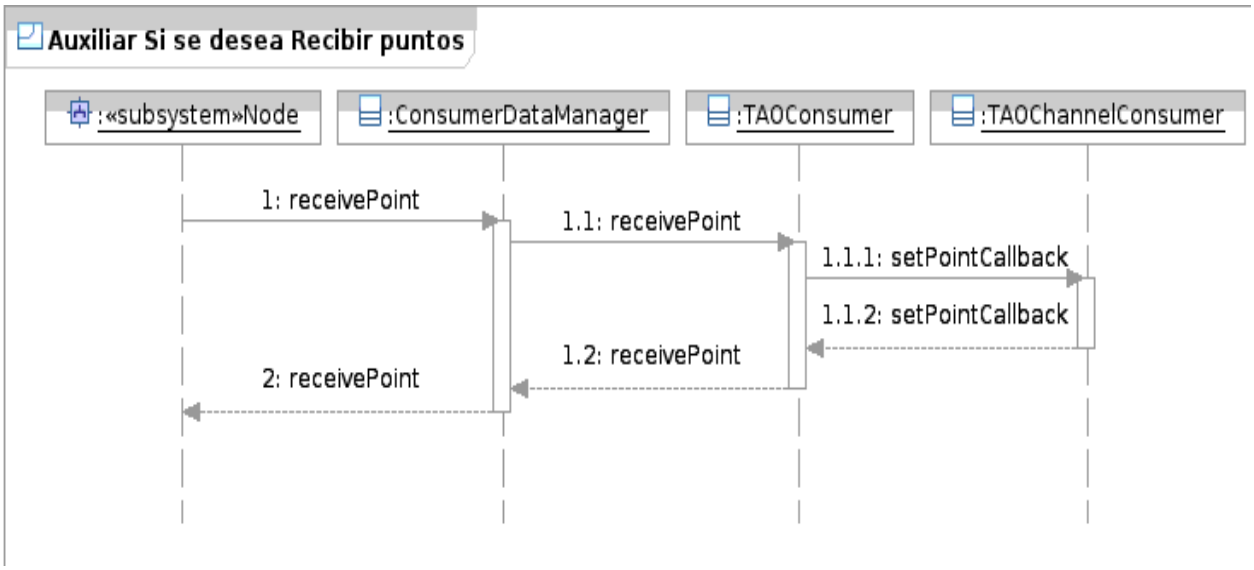

Figura 17 *Tercera parte del diagrama de secuencia del caso de uso incluido Inscripción.*

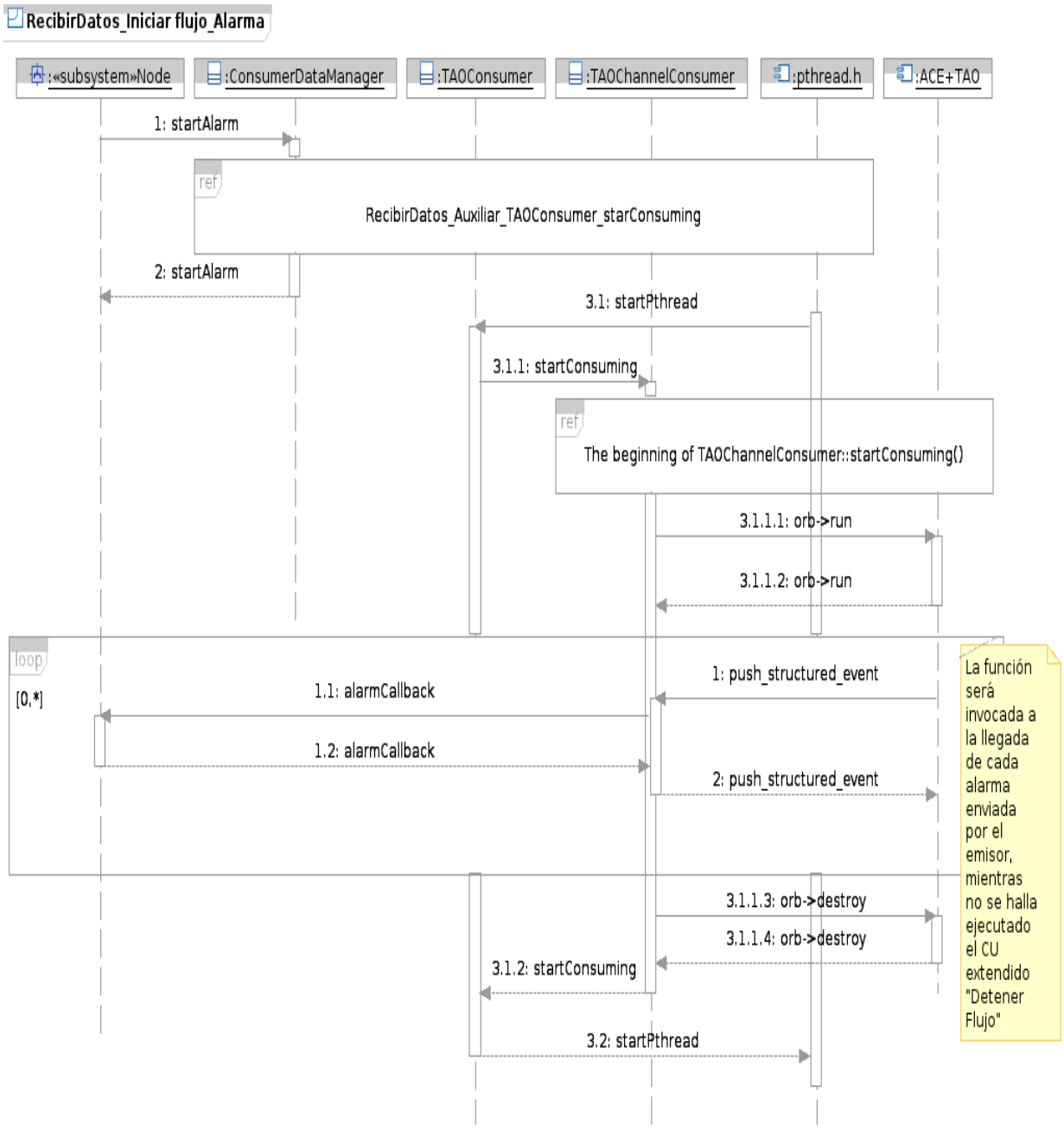

Figura 18 *Diagrama de secuencia del caso de uso extendido Iniciar Flujo primera variante*

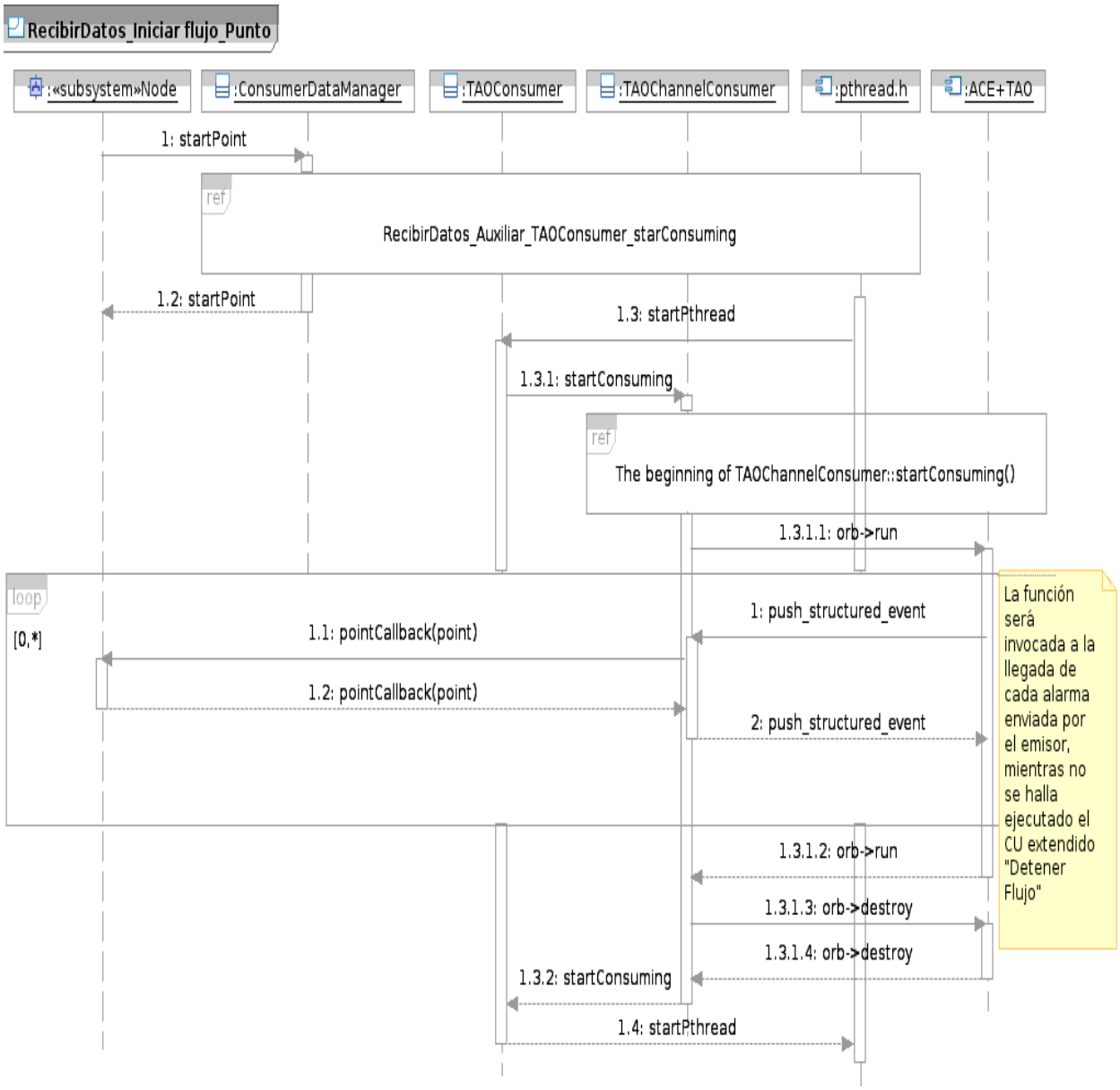

Figura 19 *Diagrama de secuencia del caso de uso extendido Iniciar Flujo segunda variante*

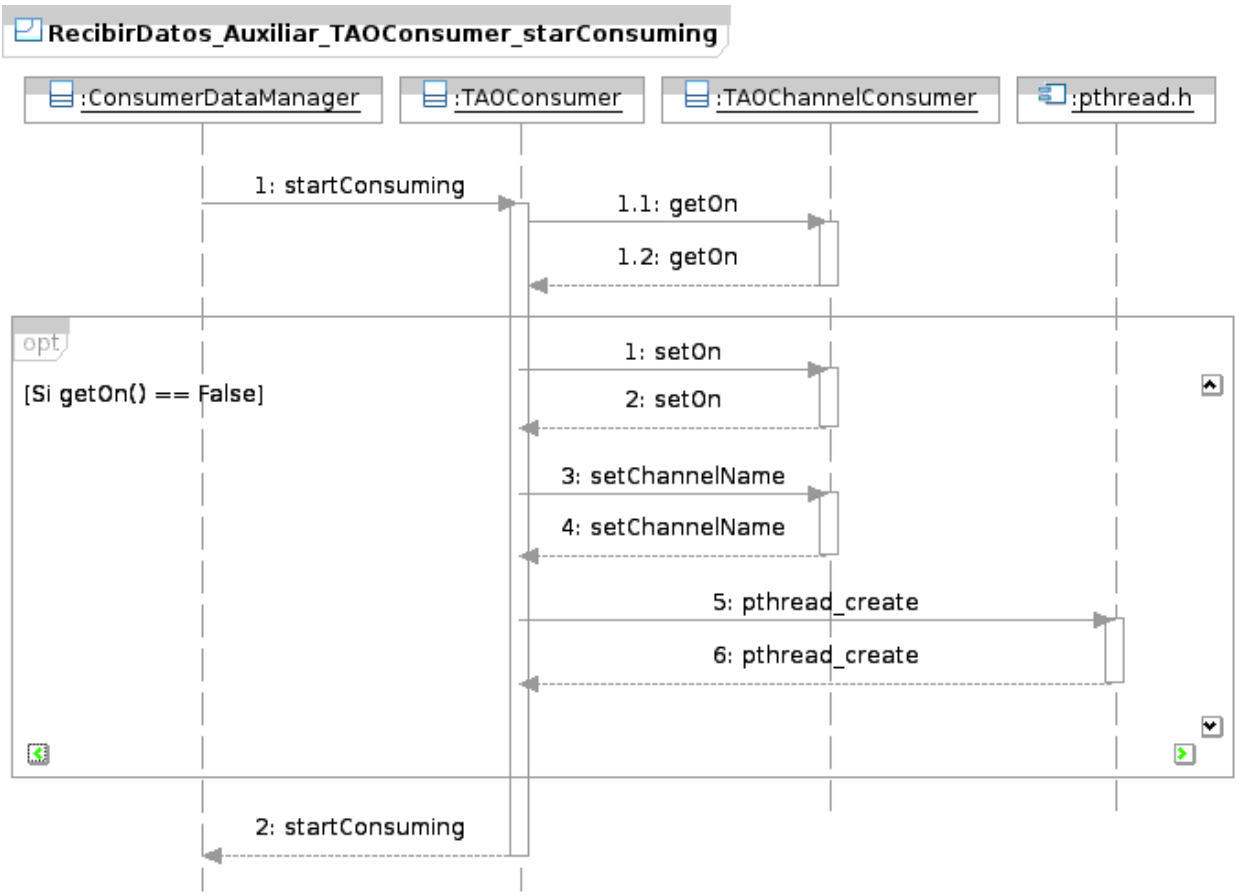

Figura 20 *Diagrama de secuencia referenciado en diagramas anteriores con el nombre "RecibirDatos_Auxiliar_TAOConsumer_startConsuming"*

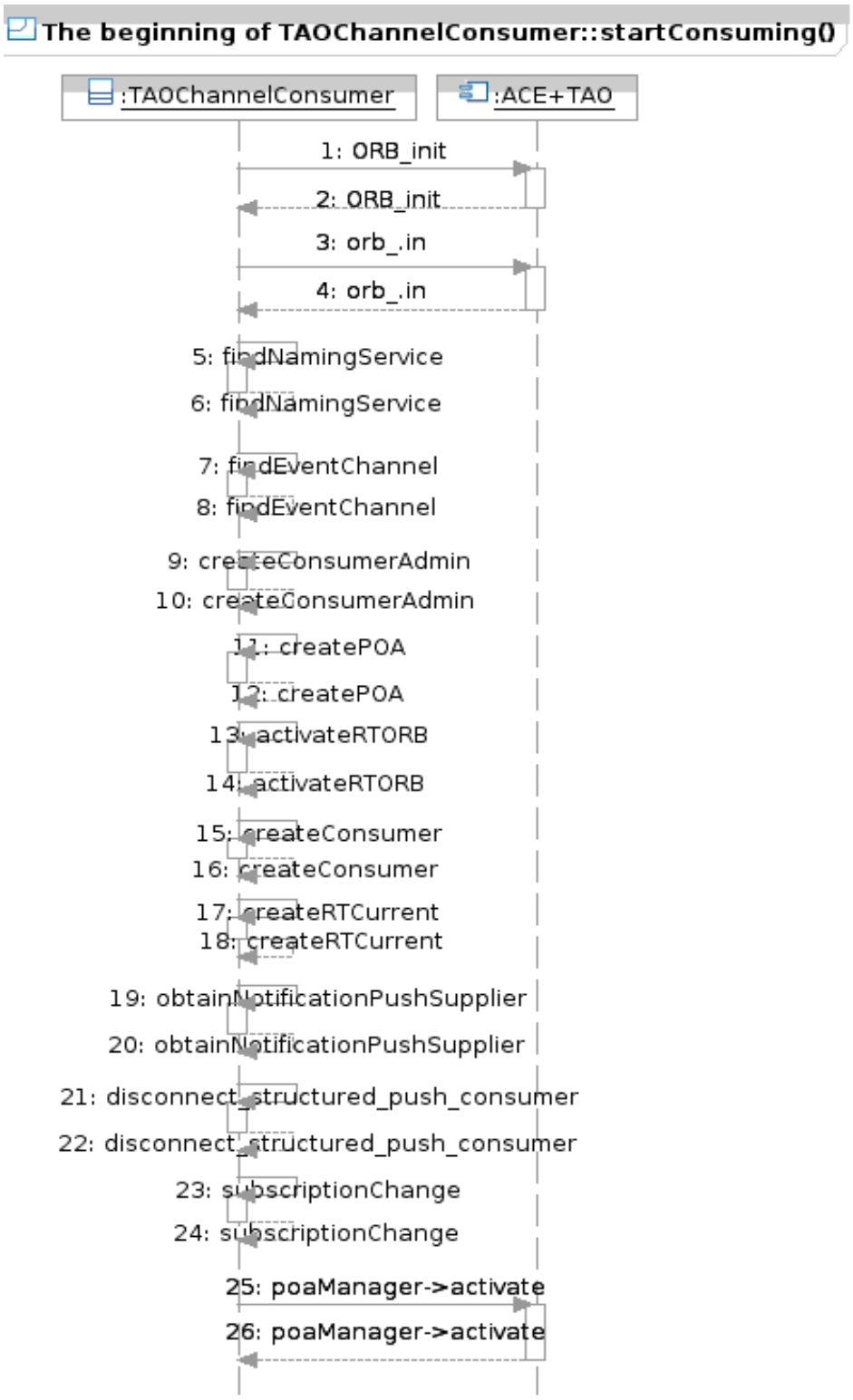

Figura 21 *Diagrama de secuencia referenciado en diagramas anteriores con el nombre "The beginning of TAOChannelConsumer::startConsuming()*

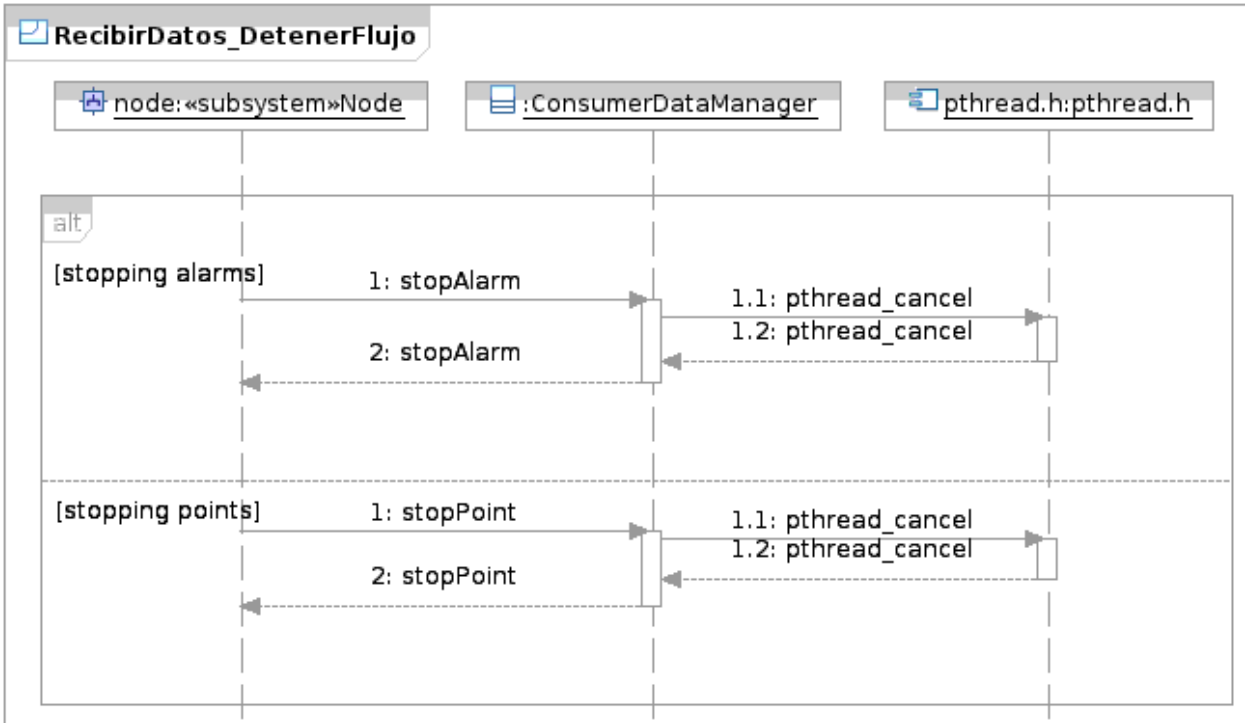

Figura 22 *Diagrama de secuencia del caso de uso extendido Detener Flujo*

3.3.2.2 CASO DE USO: ENVIAR DATOS

Las clases del diseño del caso de uso Enviar datos, quedarían relacionadas como se muestra a continuación en sendos diagramas de clase que reflejarán además las relaciones con los tipos de datos Punto y Alarma respectivamente. Los diagramas se separan de esta forma por alcanzar mayor legibilidad.

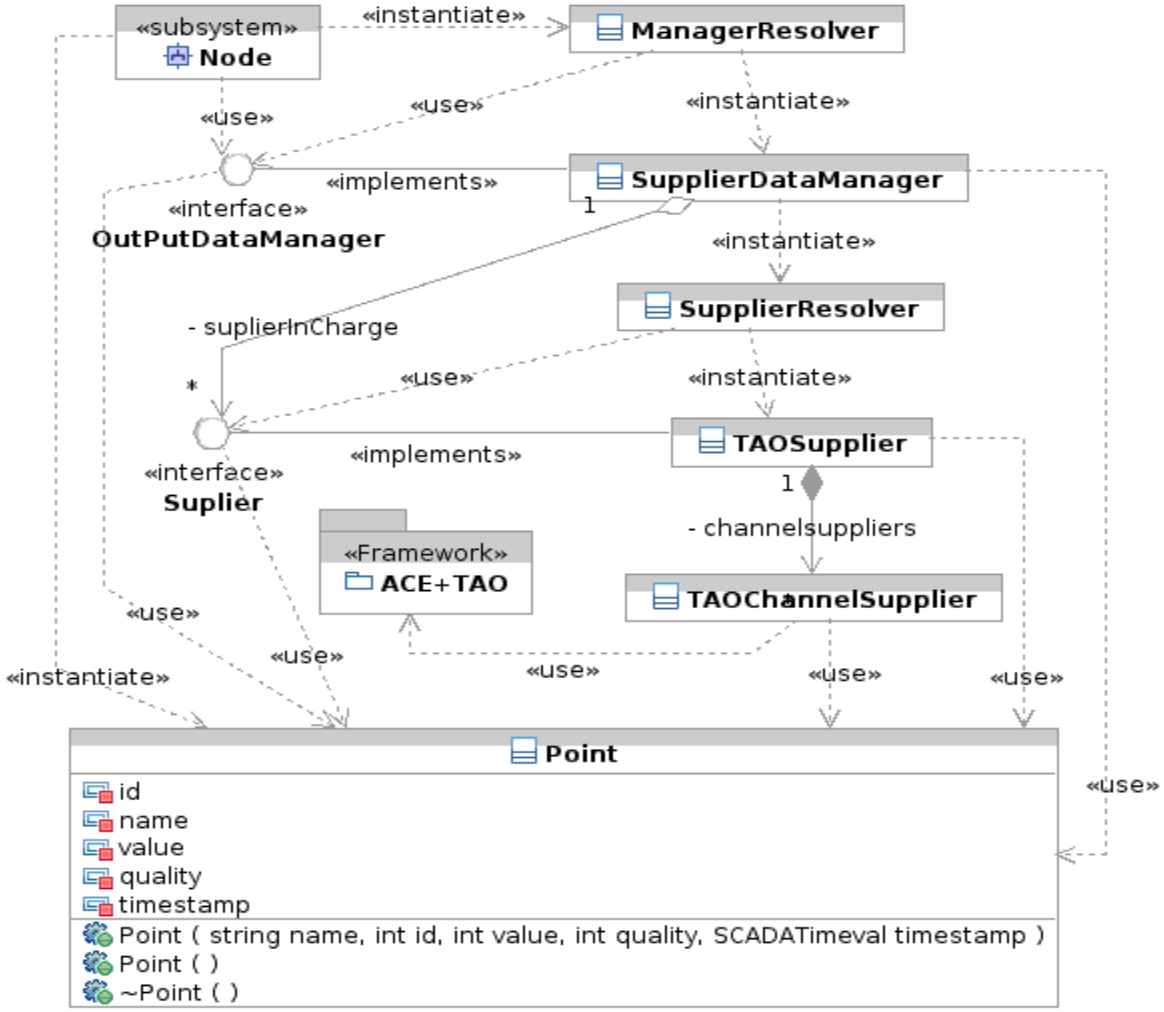

Figura 23 *Relación entre la clase "Point" y las clases del diseño del caso de uso Enviar datos*

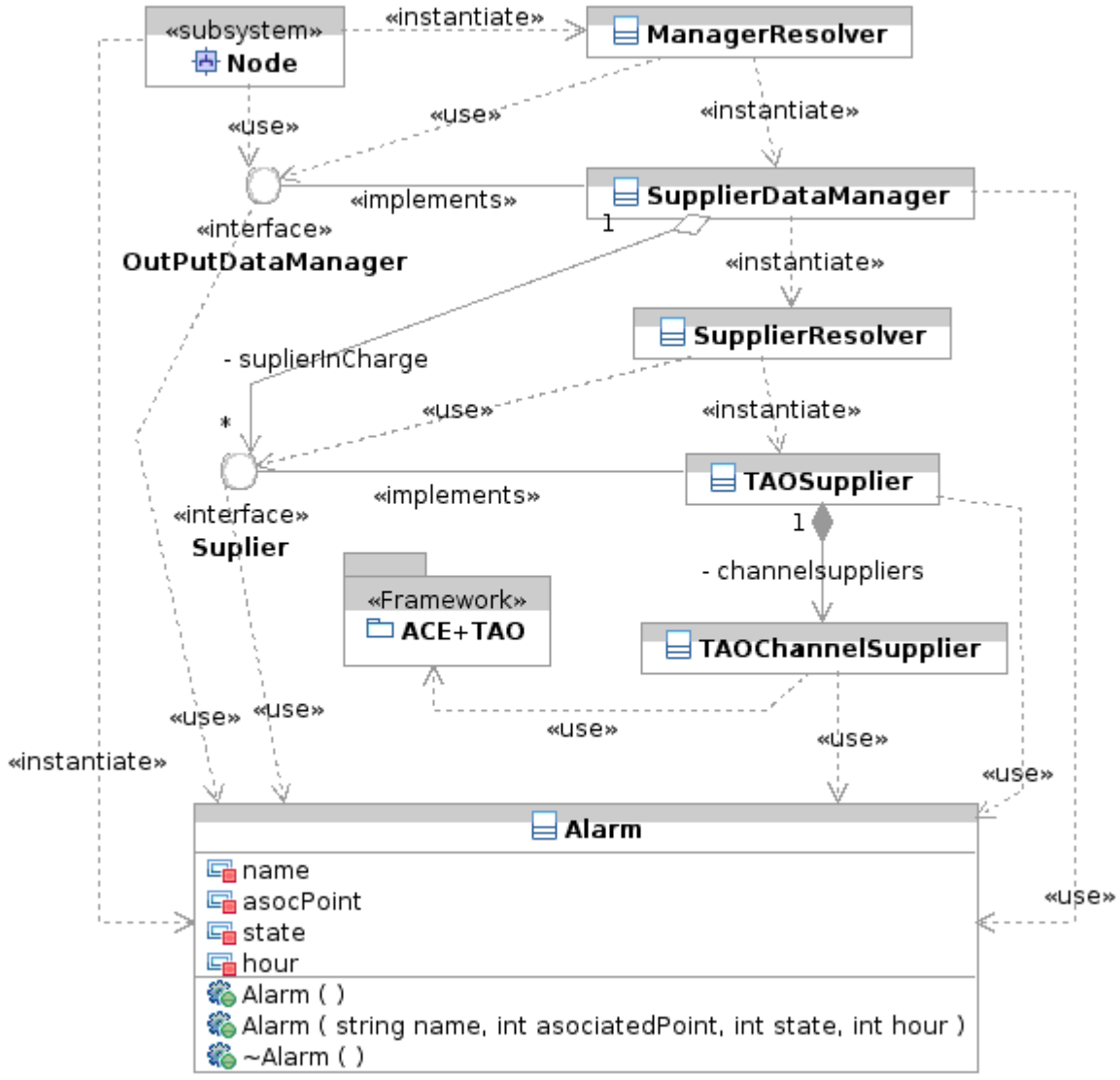

Figura 24 *Relación entre la clase "Alarm" y las clases del diseño del caso de uso Enviar datos*

Diagramas del sistema

Seguidamente se irán presentando, como se hizo con anterioridad, cada una de las partes del diagrama del diseño correspondiente al caso de uso "Enviar datos" con un mayor grado en detalles. Los fragmentos se irán organizando por niveles, comenzando por la biblioteca de la capa más alta.

.

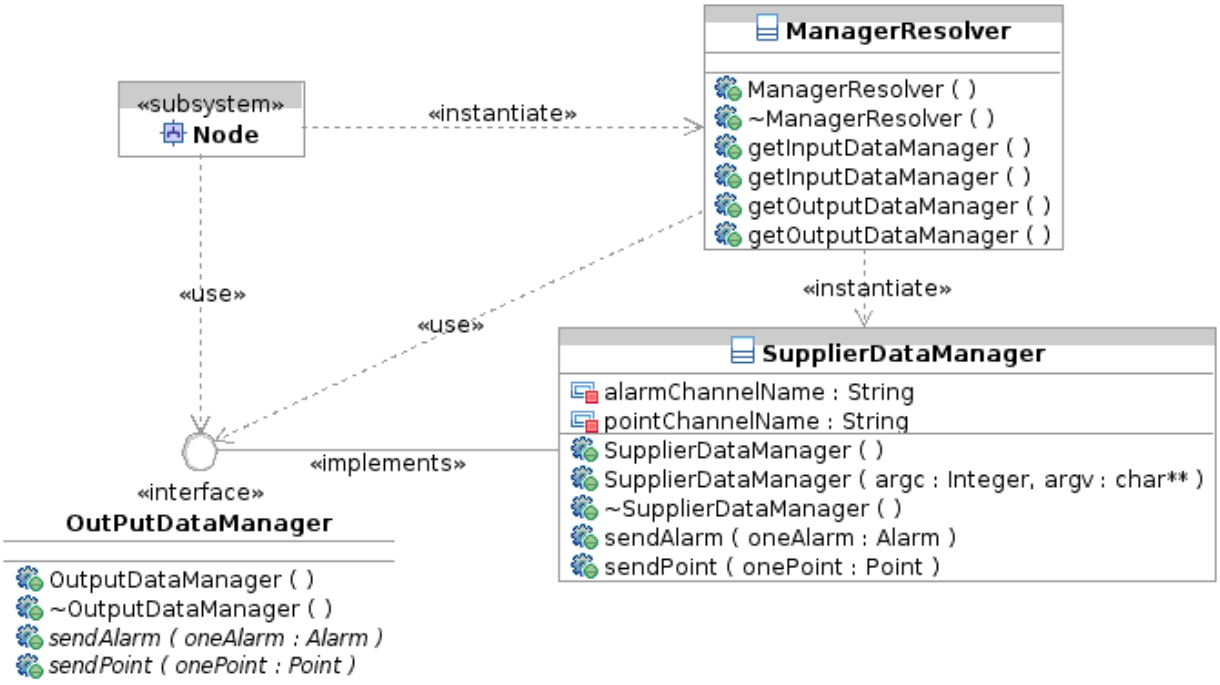

Figura 25 *Clases e interfaces del nivel superior del middleware del paquete OutPutDataModel en comunicación con otros módulos del SCADA*

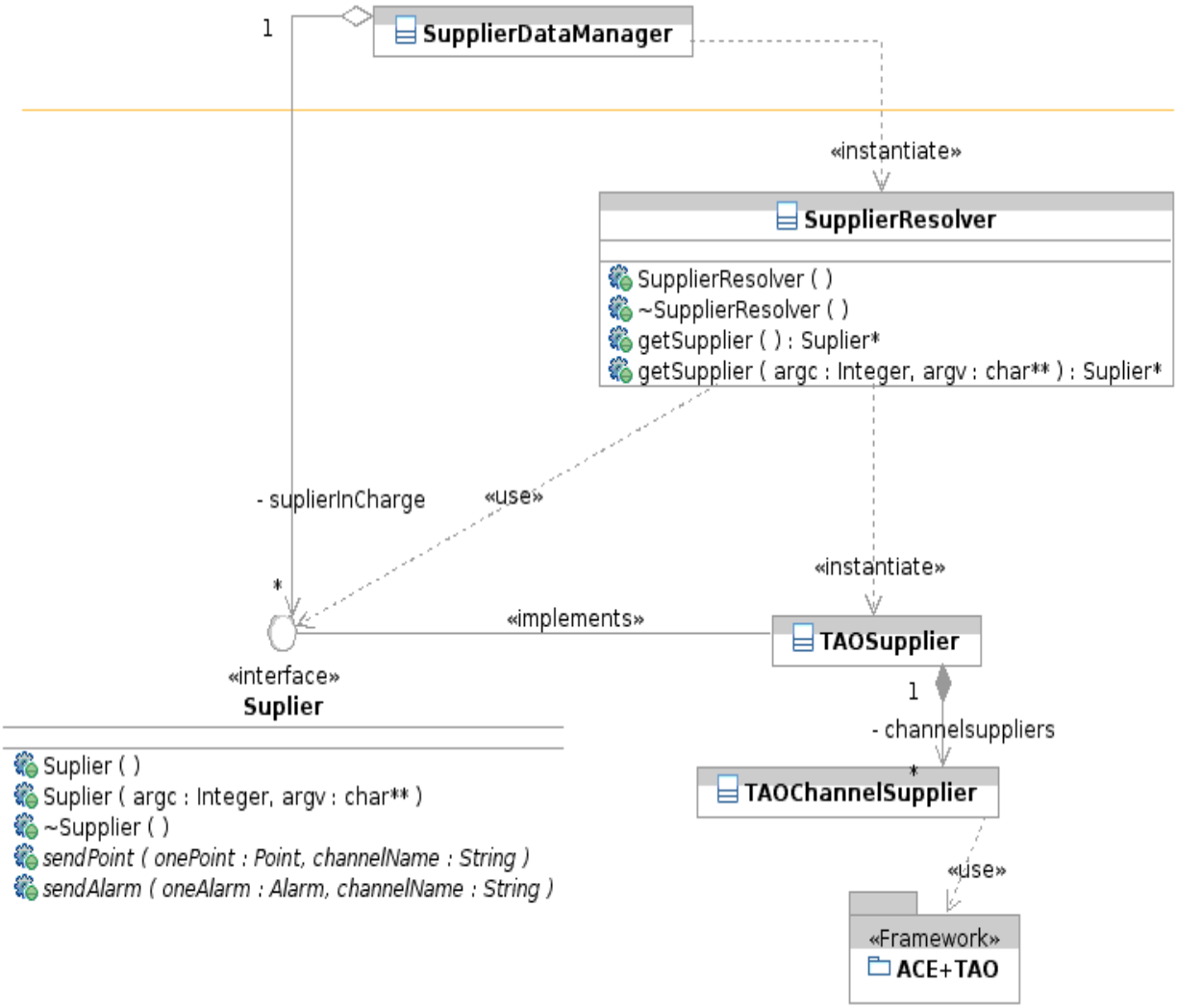

Figura 26 *Clases e interfaces del nivel más bajo del middleware pertenecientes al paquete SupplierModel. Detallando en Supplier y SupplierResolver*

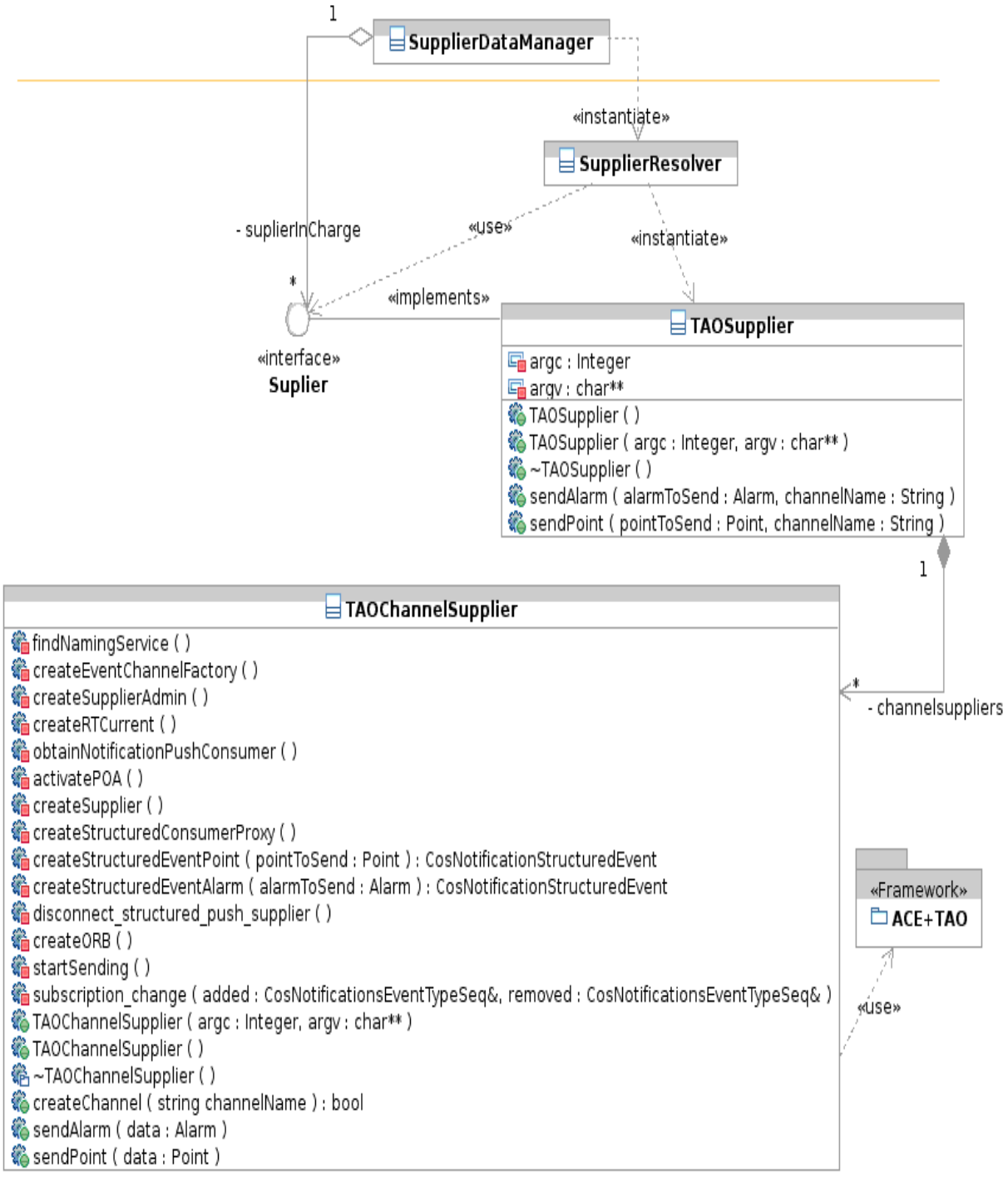

Figura 27 *Clases e interfaces del nivel más bajo del middleware pertenecientes al paquete SupplierModel. Detallando en la clase TAOChannelSupplier.*

Diagramas de Secuencia

A continuación, con el objetivo de alcanzar una mejor comprensión se presentan los diagramas de secuencia correspondientes a las interrelaciones diseñadas para el caso de uso Enviar Datos.

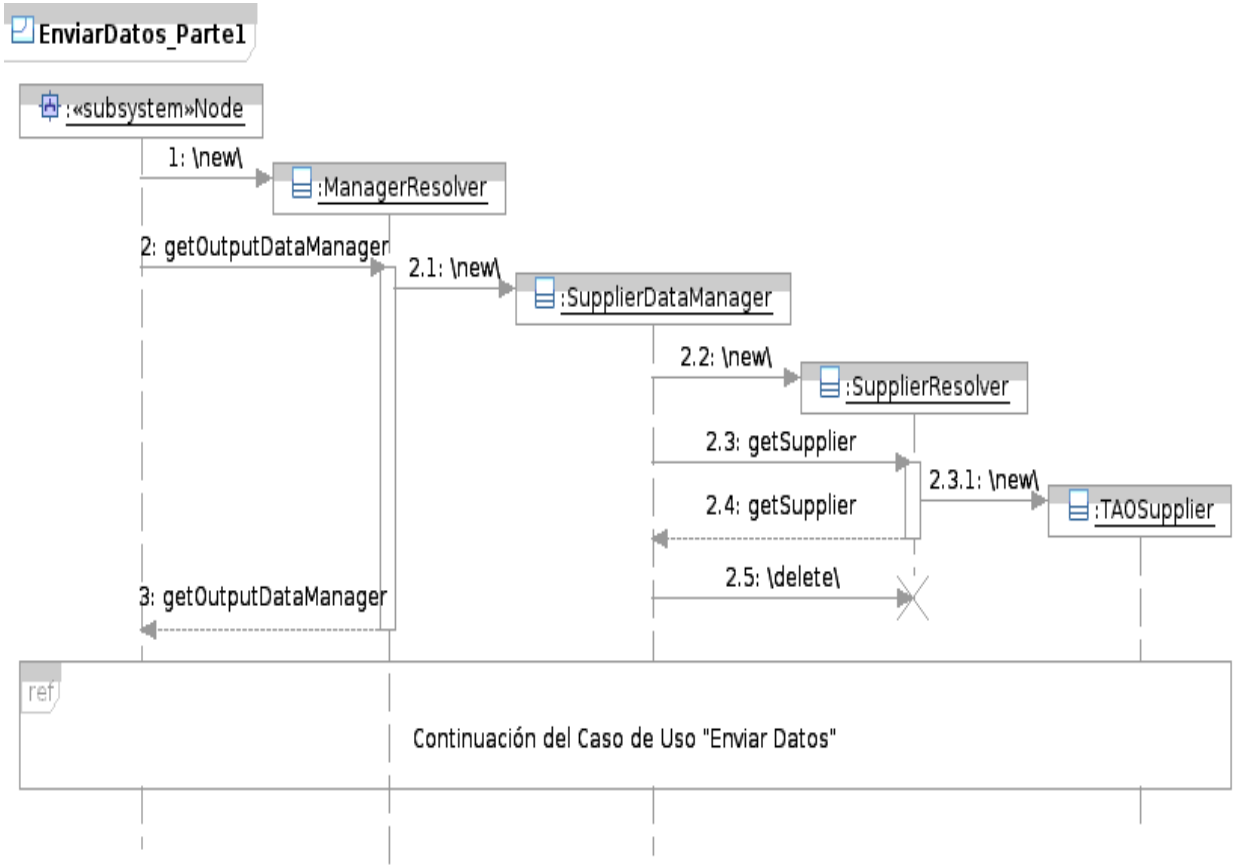

Figura 28 *Primera parte del diagrama de secuencia del caso de uso Enviar Datos.*

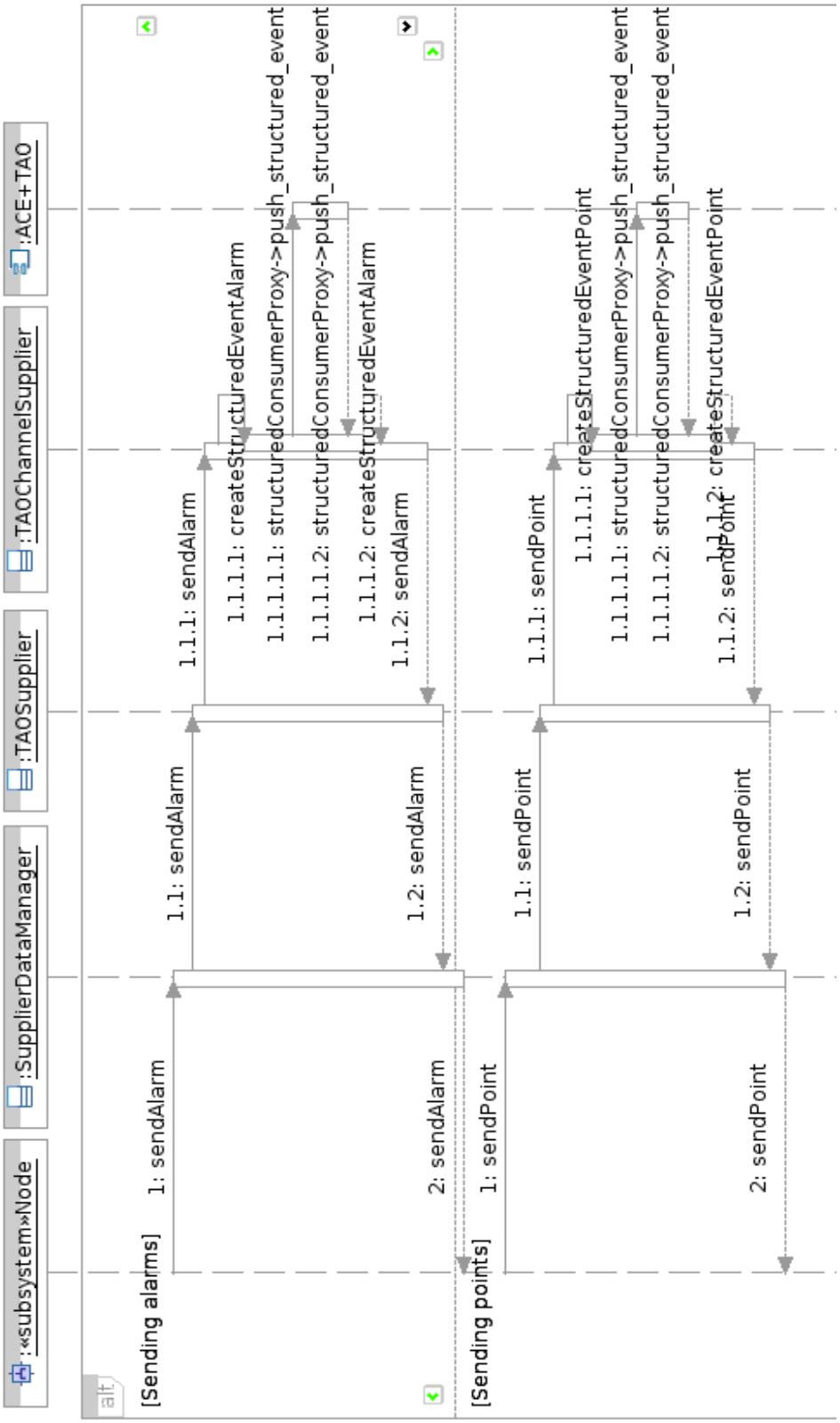

Figura 29 *Segunda parte del diagrama de secuencia del caso de uso Enviar Datos.*

3.4 ANÁLISIS DE LOS RESULTADOS

La selección realizada de la tecnología middleware a utilizar es un resultado con repercusión directa en el desarrollo del proyecto SCADA PDVSA. El *framework* seleccionado pudiera limitar o facilitar la implementación del sistema crítico en tiempo real, así como la fiabilidad para la supervisión de procesos automatizados. El carecimiento de servicios de carácter fundamental y complejo desarrollo, podrían imposibilitar la evolución exitosa del proyecto, pues el mismo se encuentra supeditado a estrictos márgenes de tiempo. Una plataforma con las abstracciones adecuadas evitaría la codificación de bajo nivel, innecesaria, propensa a errores y en muchas ocasiones, imposible de llevar a cabo dada las particularidades de la situación. Un conjunto certero de servicios de alguna tecnología probada, podrían brindar al desarrollo del software de comunicación del SCADA PDVSA los niveles de fiabilidad requeridos.

El saldo de la investigación llevada a cabo en este sentido es positivo. La tecnología middleware escogida ha sido ampliamente probada en variadas aplicaciones (Schmidt's, 2006), y posee las funcionalidades primarias de interés ineludible al desarrollo del software de comunicación del proyecto SCADA PDVSA. Se ha determinado la utilización de una plataforma proveedora de las abstracciones propicias, tanto para la simplificación necesaria de las tareas a realizar como para la obtención de un nivel de fiabilidad adecuado (Epígrafes: 2.4.2, 2.4.6 y 2.5).

El diseño efectuado del *middleware* básico constituye una primera e importante aproximación al *middleware* del sistema supervisor.

1. **Posibilita la implementación de un software de comunicación que garantice las funcionalidades básicas de intercambio de información y de esta forma, proporcione un precedente en el desarrollo de middleware con software libre.** Dada la inexistencia de antecedentes a nivel nacional de soluciones *middleware* desarrolladas con aplicaciones no privativas, se comprende como necesario la elaboración de un prototipo en calidad de ejemplo base al futuro sistema. Aunque el equipo de desarrollo cubano posee entre sus líderes profesores con experiencias en la elaboración de sistemas SCADA, ellos no habían incursionado hasta el momento

en este campo utilizando herramientas libres. Por su parte, el resto de los integrantes clasificaban como pioneros, también en la esfera de aplicaciones para la automatización de la supervisión y control de los procesos.

2. **Posibilita la adquisición de conocimiento, experiencias y habilidades sin dejar de aportarle incrementos a la solución final del sistema, dadas la flexibilidad y mantenibilidad tenidas en cuenta durante su elaboración.** Al establecerse los límites del diseño realizado junto a las funcionalidades básicas de intercambio de información necesarias al sistema, se obtuvo un diseño más comprensible y con pocos obstáculos para su implementación. Lo cual condiciona una alta curva en el aprendizaje y rápida apropiación de experiencias por parte del equipo de desarrollo menos familiarizado con el tema. Además, la división del problema posibilitó la obtención de soluciones de diseño eficientes, menos probables de alcanzar si se hubiese intentado resolver todos los desafíos a la vez. La realización del diseño para el middleware básico se efectuó teniendo en cuenta imprimirle flexibilidad, mantenibilidad y extensibilidad. Tal hecho posibilita la asimilación de cambios y extensiones del diseño resultante sin implicar necesariamente su re-elaboración.
3. **Permite la detección de los posibles problemas sin graves consecuencias.** Pues acondiciona la realización de un desarrollo menor que implicaría en el peor de los casos, si fuese fallido, un saldo menor de pérdidas de recursos como el tiempo invertido en su implementación que si se hubiese llevado a cabo un desarrollo mayor.

CONCLUSIONES

Las funcionalidades fundamentales del *middleware* se resumen en: concertar cómo los módulos de una aplicación se interrelacionan y operan coordinadamente posibilitando, rebasar los obstáculos de comunicación existentes entre las distintas partes del sistema y tanto el hardware como el software de bajo nivel. Además de contribuir a la creación de los sistemas distribuidos imprimiendo velocidad y robustez al proceso al brindar, servicios configurables y posibilidad de integración con componentes desarrollados por distintos proveedores de tecnologías.

Tanto CORBA como DDS constituyen estándares propicios para el desarrollo de software de comunicación de un SCADA, pues ofrecen soporte a todos los requerimientos tecnológicos básicos y generales identificados.

No obstante en la actualidad, para su desarrollo con software libre, en sistemas distribuidos críticos en el tiempo real y fiabilidad, el *framework* más adecuado es ACE+TAO. DDS, independientemente de asemejarse al estándar ideal en estos casos, no posee aún implementaciones con la suficiente madurez en cuanto a su no disponibilidad de certeros mecanismos de tolerancia a fallas.

Con respecto al diseño obtenido se concluye que da respuesta a las necesidades de comunicación básicas entre los distintos componentes del Sistema Supervisor de Procesos Automatizados de Petróleos de Venezuela S.A., de forma eficiente, y brindado además predictibilidad y mantenibilidad, alto nivel de desacoplamiento e interfaces sencillas.

RECOMENDACIONES

Debido a las características del estándar que implementa y a su constante evolución, se recomienda efectuar estrecho seguimiento de las actualizaciones de la tecnología *middleware,* TAODDS de la OCI. Cuyas versiones futuras pudieran valorarse nuevamente para su utilización.

También se recomienda implementar el diseño realizado y de esta forma obtener un precedente flexible y escalable que permita, asentar las bases necesarias al desarrollo de la aplicación de comunicación del SCADA PDVSA.

BIBLIOGRAFÍA

Biznestehnologia Ltd. 2007. CORBA. *Millennium ERP: a competitive ERP solution for the new millennium / CORBA (eng).* [En línea] UralWES, 2007. [Citado el: 14 de Mayo de 2007.] http://www.millennium-group.ru/index/lang/eng/parent_id/30/level/1.

Bollow, Norbert. 2007. DotGNU Project. *DotGNU Project.* [En línea] DotGNU Project , 20 de Marzo de 2007. [Citado el: 14 de Mayo de 2007.] http://www.gnu.org/projects/dotgnu/.

Borland. 2007. CORBA Environment for Distributed Processing & Computing Applications – from Borland. *Borland The Open Alm Company.* [En línea] Borland, 2007. [Citado el: 14 de Mayo de 2007.] http://www.borland.com/us/products/visibroker/index.html.

Cetus Team. 2002. Cetus Links: 16604 Links on Objects and Components / Distributed Objects & Components: General Information. *Cetus Links:18,193 Links on Objects and Components.* [En línea] Cetus Team, 5 de Julio de 2002. [Citado el: 6 de Mayo de 2007.] http://www.cetus-links.org/oo_distributed_objects.html.

Chávez Frías, Hugo. 2004. Decreto N° 3.390. [En línea] 23 de Diciembre de 2004. [Citado el: 18 de Febrero de 2007.] http://www.gobiernoenlinea.gob.ve/docMgr/sharedfiles/Decreto3390.pdf.

Citect Pty. Ltd. 2006. SCADA Benefits. *Citect.* [En línea] Citect, 2006. [Citado el: 12 de abril de 2007.] http://www3.citect.com/products/citectscada/benefits.

Creating .Net Applications on Linux and Mac OS X. **Avery, James y Holmes, Jim. 2007.** s.l. : O'Reilly Media Inc., 2007.

Curtis, David, Stone, Christopher y Bradley, Mike. IIOP: OMG's Internet Inter-ORB Protocol. *Object Management Group.* [En línea] [Citado el: 17 de Mayo de 2007.] http://www.omg.org/library/iiop4.html.

David E., Bakken. 2003. MIDDLEWARE. *Washington State University.* [En línea] School of Electrical Engineering and Computer Science, 2003. [Citado el: 7 de Mayo de 2007.] http://www.eecs.wsu.edu/~bakken/middleware.htm.

Dennison, Randy. 2004. *A Pre-SCADA System Assessment.* [PDF] s.l. : EPG Companies Inc., 2004.

Department of Energy. Importance of Energy Control Systems to Protecting Critical National Infrastructures. *Department of Energy.* [En línea] Department of Energy. [Citado el: 12 de abril de 2007.] http://www.oe.energy.gov/439.htm.

Fermitools. 2005. Abstract - ROBIN. *Fermitools.* [En línea] Fermitools, 7 de Julio de 2005. [Citado el: 14 de Mayo de 2007.] http://fermitools.fnal.gov/abstracts/robin/abstract.html.

Gall, Nicholas. 2003. The Origin (Coining) of the Term "Middleware". [En línea] 2 de Noviembre de 2003. [Citado el: 7 de Mayo de 2007.] http://radio.weblogs.com/0126951/2003/11/02.html#a72.

Grisby, Duncan. 2006. Free High Performance ORB. *OmniORB.* [En línea] 28 de Noviembre de 2006. [Citado el: 14 de Mayo de 2007.] http://omniorb.sourceforge.net/.

Hernández Lugones, Luis Alberto. 1998. *Sistema Automatizado de Diagnóstico, Supervisión y/o Control del Consumo de Energía Eléctrica en Instalaciones Industriales y de Servicio.* [Word Document] 1998.

Herrera Vázquez, Moisés. 2007. Sistemas Distribuidos, sistemas SCADA y Middleware. La Habana : Ana Silvia Tellería Martínez, 3 de Mayo de 2007.

IEEE. 1998. *Guía para el desarrollo de Especificaciones de Requerimientos de Sistemas.* [PDF] 1998.

Institute for System Progamming of RAS . 1996. http://www.ispras.ru/~microrb/download/COPYRIGHT. *Distributed Systems.* [En línea] Institute for System Progamming of RAS , 1996. [Citado el: 14 de Mayo de 2007.] http://www.ispras.ru/~microrb/download/COPYRIGHT.

IONA Technologies . 2007. Orbacus: Your CORBA Source. *Orbacus: Your CORBA Source.* [En línea] IONA Technologies , 2007. [Citado el: 14 de Mayo de 2007.] http://www.orbacus.com/.

IONA Technologies. 2007. IONA Orbix: CORBA for the Enterprise. *IONA: Making Software Work Together.* [En línea] IONA Technologies, 2007. [Citado el: 14 de Mayo de 2007.] http://www.iona.com/products/orbix/.

Jamia Millia Islamia. SCADA - AN INTRODUCTION. *Jamia Millia Islamia.* [En línea] Jamia Millia Islamia. [Citado el: 12 de abril de 2007.] http://www.jmi.nic.in/Fengg/SCADA/Aboutscada.htm.

Johnson, Doug. 2005. Technology Improve Reliability. *Blue Ridge Electric.* [En línea] Blue Ridge Electric Membership Corporation, 2005. [Citado el: 12 de abril de 2007.] http://www.blueridgeemc.com/EN_2007/Enlightener_0207.htm.

Lee, Elliot. 1999. ORBit. *CORBA Applications In GNOME.* [En línea] Red Hat, Inc., 1999. [Citado el: 24 de abril de 2007.] http://developer.gnome.org/doc/whitepapers/ORBit/about-orbit.html.

2001. LuaOrb Online. *LuaOrb.* [En línea] Tecgraf / PUC-Rio, 15 de Mayo de 2001. [Citado el: 14 de Mayo de 2007.] http://www.tecgraf.puc-rio.br/~rcerq/luaorb/.

MICO Project Team. 2006. MICO CORBA. *MICO CORBA.* [En línea] ObjectSecurity Ltd., 2006. [Citado el: 14 de Mayo de 2007.] http://www.mico.org/index.html.

NATO SOFTWARE COMITEE. 1969. The NATO Software Engineering Conferences. *Staff, Computing Science, Newcastle University.* [En línea] Enero de 1969. [Citado el: 7 de Mayo de 2007.] http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF.

Nilsson, Staffan. An introduction to real time concepts. *The embedded developers pages.* [En línea] [Citado el: 12 de Abril de 2007.] http://hem.bredband.net/stafni/developer/realtimeintro.htm.

Object Computing, Inc. 2007. Home - The ACE ORB. *Object Computing, Inc. - An OO Software Engineering Company.* [En línea] Object Computing, Inc., 2007. [Citado el: 14 de Mayo de 2007.] http://www.theaceorb.com/.

Objective Interface. OIS - Product and Services - ORBexpress Family Overview. *Objective Interface.* [En línea] Objective Interface. [Citado el: 14 de Mayo de 2007.] http://www.ois.com/products/prod-1.asp.

OMG. 2004. *Common Object Request Broker Architecture: Core Specification.* [PDF] s.l. : OMG, March de 2004.

—. 2007. Data Distribution Service for Real-time Systems. *Object Management Group.* [En línea] 1 de Enero de 2007. [Citado el: 14 de Mayo de 2007.] http://www.omg.org/cgi-bin/apps/doc?formal/07-01-01.pdf.

OpenORB Community Project. 2007. SourceForge.net Project News. *SourceForge.net.* [En línea] OpenORB Community Project, 2007. [Citado el: 14 de Mayo de 2007.] http://sourceforge.net/projects/openorb/.

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

PDVSA. 2006. *Anexo 5 Diagnóstico del Proyecto SCADA Nacional del Convenio Marco PDVSA – ALBET, S.A.* [Word Document] Caracas, Venezuela : Petróleos de Venezuela SA, 2006.

—. 2006. *Desarrollo de SCADA Nacional.* [Word Document] Caracas, Venezuela : PDVSA, 17 de Abril de 2006.

—. 2005. PDVSA - Petróleos de Venezuela S.A. *PDVSA - Petróleos de Venezuela S.A.* [En línea] PDVSA, 2005. [Citado el: 6 de mayo de 2007.] http://www.pdvsa.com/.

Preferred Utilities Manufacturing Corporation. SCADA syste to monitor and control plant-wide processes froma a single location, while at the same time collecting and sharing real-time plant data-Preferred Instruments. *Preferred Instruments.* [En línea] Preferred Utilities Manufacturing Corporation. [Citado el: 12 de abril de 2007.] http://www.preferredinstruments.com/scada.html.

Prensa PDVSA . [En línea] http://www.aporrea.org/energia/n90215.html.

Prensa PDVSA. 2007. Banca internacional ratifica su confianza en PDVSA al otorgarle una línea de crédito por $1.000 millones. *Aporrea.org.* [En línea] Prensa PDVSA, 5 de Febrero de 2007. [Citado el: 6 de Mayo de 2007.] http://www.aporrea.org/energia/n90215.html .

PrismTech. Mission Critical Networked Systems. *PrismTech Productivity Tools & Middleware.* [En línea] PrismTech. [Citado el: 14 de Mayo de 2007.] http://www.prismtechnologies.com/section-item.asp?id=175&sid=18&sid2=10.

Rosenberger, Jeremy L. 1998. *Sams Teach Yourself CORBA in 14 Days.* s.l. : Sams, 1998. 0672312085.

Schmidt, Douglas C. 2005. *Department of Computer Science & Engineering - Home - Department of Computer Science & Engineering - Washington University.* [En línea] 2005 de Febrero de 2005. [Citado el: 7 de Mayo de 2007.] http://www.cs.wustl.edu/~schmidt/PDF/middleware-encyclopedia.pdf.

—. 1998. *Department of Computer Science & Engineering - Home - Department of Computer Science & Engineering - Washington University.* [En línea] 19 de Noviembre de 1998. [Citado el: 7 de Mayo de 2007.] www.cs.wustl.edu/~schmidt/PDF/corba4.pdf.

—. 2007. The ADAPTIVE Communication Environment (ACE). *Douglas C. Schmidt.* [En línea] 9 de Mayo de 2007. [Citado el: 17 de Mayo de 2007.] http://www.cs.wustl.edu/~schmidt/ACE.html.

Schmidt, Douglas C. y Fayad, Mohamed. 2006. Lessons Learned Building OO Telecommunication Software. *Department of Computer Science & Engineering - Home - Department of Computer Science & Engineering - Washington University.* [En línea] 28 de Septiembre de 2006. [Citado el: 7 de Mayo de 2007.] http://www.cs.wustl.edu/~schmidt/CACM-lessons.html.

Schmidt, Douglas C. y Vinoski, Steve. 2004. Dr. Dobb's | The CORBA Component Model: Part 1, Evolving Towards Component Middleware | January 27, 2004. *Dr. Dobb's.* [En línea] CMP Technology, 27 de Enero de 2004. [Citado el: 7 de Mayo de 2007.] http://www.ddj.com/dept/cpp/184403884.

Schmidt's, Douglas C. 2006. ACE and TAO Success Stories. *Douglas C. Schmidt.* [En línea] 19 de Octubre de 2006. [Citado el: 14 de Mayo de 2007.] http://www.cs.wustl.edu/~schmidt/TAO-users.html.

—. 2006. Real-time CORBA with TAO (The ACE ORB). *Douglas C. Schmidt.* [En línea] DOC, 12 de Diciembre de 2006. [Citado el: 14 de Mayo de 2007.] http://www.cs.wustl.edu/~schmidt/TAO.html.

Spiegel, Andre. 2007. JacORB. *JacORB.* [En línea] "Freie Universität" de Berlín, 17 de Febrero de 2007. [Citado el: 14 de Mayo de 2007.] http://www.jacorb.org/index.html.

SURVALENT technology; Hometown connections. 2006. *SURVALENT Technology.* [En línea] 26 de Julio de 2006. [Citado el: 2007 de Abril de 2007.] http://www.survalent.com/solutions/Washington%20Survalent%20Case%20Study%2026Jul06.pdf.

2005. TAO 1.4a Downloads - The ACE ORB. *Home - The ACE ORB.* [En línea] 2005. [Citado el: 15 de Mayo de 2007.] http://download.ociweb.com/TAO-1.4a/TAO1.4aGettingStarted.pdf.

The Community OpenORB Project. 2007. The Community OpenORB Project. *The Community OpenORB Project.* [En línea] The Community OpenORB Project, 14 de Mayo de 2007. [Citado el: 14 de Mayo de 2007.] http://openorb.sourceforge.net/.

Trujillo Codorniú, Rafael. 2007. EROS. La Habana : Ana Silvia Tellería Martínez, 3 de Mayo de 2007.

Universidad Polítecnica de Catalunya. 2007. EPSC - DAC. *Department of Computer Architecture.* [En línea] 20 de Febrero de 2007. [Citado el: 6 de Mayo de 2007.] http://studies.ac.upc.edu/EPSC/FSD/FSD-ConceptosGenerales.pdf.

Vinoski, Steve y Henning, Michi. 1999. *Advanced CORBA® Programming with C++.* [PDF] s.l. : Addison Wesley, 12 de February de 1999. 0-201-37927-9.

Vinoski, Steve y Schmidt, Doug. 2003. Dr. Dobb's | Object Interconnections: Dynamic CORBA, Part 3 - The Dynamic Skeleton Interface | April 15, 2003. *Dr. Dobbs's Portal.* [En línea] 15 de Abril de 2003. [Citado el: 17 de Mayo de 2007.] http://www.ddj.com/dept/cpp/184403847.

ANEXOS

A1 IMPLEMENTACIONES FUNCIONALES DE CORBA

1. PrismTech: OpenFusion e*ORB SDR C++ Edition (no brinda el código fuente)
2. PrismTech: JacORB ME(Lenguaje: Java)
3. PrismTech: OpenFusion TAO ORB (no completamente libre)
4. PrismTech: OpenFusion JacORB ORB (Lenguaje: Java)
5. PrismTech: OpenFusion RTOrb Java Edition (Lenguaje: Java)
6. PrismTech: OpenFusion RTOrb Ada Edition (OrbRiver Ada) (Lenguaje: ADA)
7. Borland: VisiBroker.(Software privativo)
8. The Community OpenORB Project: OpenORB (Lenguaje: Java)
9. MTDORB (Lenguajes: “Delphi” y “Kylix”)
10. LuaORB (solo para “Lua”)
11. Robin (no completamente libre)
12. IONA: Orbix (software privativo)
13. Orbacus
14. FU Berlin ("Freie Universität" de Berlín): JacORB (Lenguaje: Java)
15. Objective Interface Systems: ORBexpress RT (software privativo)
16. Objective Interface Systems: ORBexpress ST (software privativo)
17. ObjectWeb: OpenCCM (Lenguaje: Java)
18. Mico, implementación de la que la compañía "Object Security" brinda soporte.
19. Mico/E (Lenguaje: Eiffel, no tiene tiempo real, ni notification service, proyecto inmaduro)
20. Gibraltar (Todavía inmaduro, versión 0.2XX, no tiene iiop, ni tiempo real)
21. ISP C++ (solo es (CORBA 2.0)
22. ORBit
23. OmniORB

24. DOC: TAO
25. OCI: TAO

A2 DESACOPLAMIENTO: EJEMPLO DE IMPLEMENTACIÓN DEL DISEÑO.

SupplierResolver.h	*#ifndef SUPPLIERRESOLVER_H_* *#define SUPPLIERRESOLVER_H_* *#include "Supplier.h"* *namespace SCADA* *{* *namespace Middleware* *{* *class SupplierResolver* *{* *public:* *SupplierResolver();* *virtual ~SupplierResolver();* *Supplier* getSupplier();* *Supplier* getSupplier(int argc, char** argv);* *};* *}* *}* *#endif /*SUPPLIERRESOLVER_H_*/*
Supplier.h	*#ifndef SUPPLIER_H_* *#define SUPPLIER_H_* *#include "Alarm.h"* *#include "Point.h"* *#include <iostream>* *namespace SCADA*

	{ *namespace Middleware* *{* *using namespace std;* *class Supplier* *{* *public:* *Supplier();* *virtual ~Supplier();* *virtual void sendAlarm(Alarm alarm, string channelName)=0;* *virtual void sendPoint(Point point, string channelName)=0;* *};* *}* *}* *#endif /*SUPPLIER_H_*/

Supplier.cpp	#include "Supplier.h"
	namespace SCADA
	{
	namespace Middleware
	{
	Supplier::Supplier()
	{
	}
	
	Supplier::~Supplier()
	{
	}
	}
	}

TAOSupplier.h	#ifndef TAOSUPPLIER_H_ #define TAOSUPPLIER_H_

	#include "Supplier.h"
	#include "TAOChannelSupplier.h"
	#include <map>
	#include "Point.h"
	#include "Alarm.h"
	//typedef map<string,TAOChannelSupplier*> channelSupplier;
	namespace SCADA
	{
	namespace Middleware
	{
	class TAOSupplier :public Supplier
	{
	private:
	map<string,TAOChannelSupplier*>* channelSuppliers;
	int argc;
	char ** argv;
	public:
	TAOSupplier();
	TAOSupplier(int argc, char* argv[]);
	~TAOSupplier();
	void sendAlarm(Alarm alarm, string channelName);
	void sendPoint(Point point, string channelName);
	};
	}
	}
	#endif

TAOSupplier.cpp	#include "TAOSupplier.h" #include <iostream> #include "Point.h" #include "Alarm.h" namespace SCADA {

```
namespace Middleware
{
  using namespace std;
    TAOSupplier::TAOSupplier()
    {
    }

    TAOSupplier::~TAOSupplier()
    {
      delete channelSuppliers;
      delete[] argv;
    }



    TAOSupplier::TAOSupplier(int argc, char* argv[])
    {
      this->argc = argc;
      this->argv = new char*[argc];
      for(int i = 0; i < argc;i++)
      {
          this->argv[i] = argv[i];
      }
      channelSuppliers = new map<string,TAOChannelSupplier*>();
    }

    void TAOSupplier::sendPoint(Point point, string channelName)
    {
          int cant = 0;
      cant = channelSuppliers->count(channelName);
      if(cant != 0)
      {
        (*channelSuppliers)[channelName]->sendPoint(point);
      }
```

```
        else
        {
          int pargc = argc;
          char** pargv = new char*[pargc];

          for(int i = 0; i < pargc;i++)
          {
            pargv[i] = argv[i];
          }

          TAOChannelSupplier* channelSupplier = new
TAOChannelSupplier(pargc,pargv);
          channelSupplier->createChannel(channelName);
          (*channelSuppliers)[channelName] = channelSupplier;
          channelSupplier->sendPoint(point);
        }
      }

      void TAOSupplier::sendAlarm(Alarm alarm, string channelName)
      {
         int cant = 0;
        cant = channelSuppliers->count(channelName);
        if(cant != 0)
        {
          (*channelSuppliers)[channelName]->sendAlarm(alarm);
        }
        else
        {
          int pargc = argc;
          char** pargv = new char*[pargc];

          for(int i = 0; i < pargc;i++)
          {
```

```
                pargv[i] = argv[i];
            }
            TAOChannelSupplier* channelSupplier = new
TAOChannelSupplier(pargc,pargv);
            channelSupplier->createChannel(channelName);
            (*channelSuppliers)[channelName] = channelSupplier;
            channelSupplier->sendAlarm(alarm);
          }
        }
    }
}
```

GLOSARIO

ACE (Adaptive Communications Environment)

Traducido al español como entorno de comunicación adaptativa, es un framework software libre y orientado a objetos. Implementa varios núcleos de patrones para programación concurrente proporcionando, un valioso conjunto de interfaces y componentes reutilizables en C++ que efectúan tareas comunes para la comunicación entre aplicaciones sobre distintas plataformas de sistemas operativos. Las tareas de comunicación brindadas por ACE incluyen, entrega de administración de eventos, manejo de señales, inicialización de servicios, comunicación interprocesos, administración de la memoria compartida, enrutamiento de mensajes, reconfiguración dinámica de los servicios distribuidos, ejecución y sincronización concurrente. (Schmidt, 2007)

ACTIVEX

Entorno de aplicaciones desarrollado por Microsoft. Incluye un gran número de componentes que inter-operan y enlazan diferentes aplicaciones en una computadora o red de computadoras.

API (Application Program Interface)

En español, Interfaz de Programación de Aplicaciones, es un conjunto de especificaciones de comunicación entre componentes software.

DII (*Dynamic Invocation Interface*)

En español, la interfaz de invocación dinámica es una API que permite la construcción de invocaciones CORBA dinámicamente. Es utilizada en tiempo de compilación cuando los clientes no tienen conocimiento sobre el objeto que desean invocar.

DSI (*Dynamic Skeleton Interface*)

Es la API homóloga a la DII, pero del lado del servidor. Algunas aplicaciones servidoras no tienen conocimiento de antemano sobre los tipos o entidades objetos que brindarán. La API, DSI, permite a los desarrolladores construir este tipo de aplicaciones con la portabilidad entre sus características. (Vinoski, y otros, 2003)

framework

En los sistemas orientados a objeto un framework es un conjunto de clases que encapsulan diseños abstractos de soluciones a un determinado números de problemas en relación. Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

hilos

Los hilos son las distintas partes en las que un programa se puede dividir para ejecutarse en varias tareas simultáneas o casi simultáneas.

iiop(Internet Inter-Orb Protocol)

Protocolo que posibilita la interoperabilidad de las aplicaciones de ambientes heterogéneos en Internet. Adoptada como parte de CORBA por la OMG a partir de 1994. Es un mecanismo subyacente a CORBA y es administrado de forma transparente a los usuarios finales por el ORB para la comunicación con sus homólogos implementados con otras tecnologías. Constituye la individualización para tcp/ip de la especificación más general de GIOP (*General Inter-ORB Protocol*).

GNU/Linux

Aunque se ha difundido el término Linux como siglas de la frase: ***L****inux* ***I****s* ***N****ot* ***Uni*****X**, linux es un núcleo de sistema operativo tipo Unix. El núcleo Linux se complementa con una serie de aplicaciones desarrolladas por el grupo GNU para conformar el sistema operativo software libre GNU/Linux. Linux/GNU es además Multiusuario, Mutitarea, Multiprocesador, Multiplataforma, Multilingüe, nacido en la red de redes Internet.

socket (enchufe)

Es un método para la comunicación entre un programa del cliente y un programa del servidor en una red. Un socket se define como el punto final en una conexión. Los sockets se crean y se utilizan con un sistema de peticiones o de *llamadas de función* a veces llamados interfaz de programación de aplicación de sockets (API, *Application Programming Interface*).

software libre

Las cuatro reglas esenciales que deben cumplir las aplicaciones para ser consideradas como software libre son:

- Libertad 0: Libertad de ejecutar el programa como quieras.
- Libertad 1: Libertad de estudiar el código fuente y cambiarlo para realizar lo que desees.
- Libertad 2: Libertad de realizar copias y distribuirlas cuando quieras.
- Libertad 3: Libertad de distribuir o publicar versiones modificadas cuando desees.

software privativo

Es todo software que no cumple con alguna de las cuatro reglas del software libre.