Universidad de las Ciencias Informáticas
"Facultad 5"

Título: "Capa de acceso a datos para dispositivos Bristol"

Trabajo de Diploma para optar por el título de

Ingeniero en ciencias Informáticas

Autor(es): "Javier Lorié Guerra"

Tutor(es): "Ing. Rubén Gómez Johnson"

Co-tutor: "Ing. Adrián Carlos Moreno Borges"

Ciudad de La Habana

Junio 2011

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Javier Lorié Guerra	**Rubén Gómez Johnson**	**Adrián Carlos Moreno Borges**
_____	_____	_____
Firma del Autor	Firma del Tutor	Firma del Tutor

DATOS DE CONTACTO

Ing. Rubén Gómez Johnson
Graduado con el título de Ingeniero en Ciencias Informáticas en el 2008 en la Universidad de las Ciencias Informáticas.
Email: rjohnson@uci.cu

Ing. Adrián Carlos Moreno Borges
Graduado con el título de Ingeniero en Ciencias Informáticas en el 2008 en la Universidad de las Ciencias Informáticas.
Email: acmoreno@uci.cu

AGRADECIMIENTOS

Agradezco:

A mis padres Beatriz y Orlando por siempre darme su cariño y apoyarme en todo.

A mi familia y a mis hermanas Yanela y Leydis por ser tan especiales.

A mi novia Loraima por su apoyo y comprensión, por hacerme cada día una mejor persona.

Agradezco a mis tutores Rubén y Adrián Carlos por toda su ayuda en este trabajo.

A mis amigos de toda la vida por siempre estar ahí para mí y nunca defraudarme.

A mis compañeros de grupo.

A mis profesores.

A la Revolución y a Fidel por darme la oportunidad de estudiar en esta universidad maravillosa.

A todos los que de una forma u otra influyeron en mi formación como profesional.

A mis padres Beatriz y Orlando

A mis hermanas Yanela y Leydis

A mi familia

RESUMEN

En todo proceso de automatización es necesario captar las magnitudes de la planta, para poder así, saber el estado y evolución del proceso que estamos controlando. Para ello se emplean dispositivos como autómatas programables, PLC y sensores capaces de adquirir los valores anteriormente mencionados. Estos dispositivos poseen uno o varios protocolos de comunicación mediante el cual se puede acceder a los datos recopilados del campo y los manejadores de dispositivos son los que gestionan esta comunicación.

En las instalaciones de PDVSA se cuenta con una amplia variedad de dispositivos para el control de la producción, entre ellos los llamados Bristol de redes 3000 y el SCADA Guardián del Alba presenta dificultades en la adquisición de datos con dichos dispositivos. Pues el manejador correspondiente demora en actualizar las variables asociadas a este tipo de dispositivos. Situación que limita la posibilidad para los operadores de monitorear los procesos que se llevan a cabo en la industria. Teniendo en cuenta esta situación se toma como objetivo para este trabajo, desarrollar un nuevo manejador que permita disminuir la demora en la adquisición de datos para los dispositivos Bristol.

Con tal objetivo se realizó un análisis del manejador que se encuentra instalado actualmente y un estudio del protocolo BSAP, el cual rige la comunicación con este tipo de dispositivos. Esto permitió desarrollar una solución que reduce los tiempos de actualización de las variables en un 88%. Permitiendo un mejor control de la producción en las industrias de PDVSA.

Palabras clave: Bristol, Guardián del Alba, manejador de dispositivos, SCADA.

Tabla de Contenidos

INTRODUCCIÓN

En el año 2002 la industria petrolera, Petróleos de Venezuela Sociedad Anónima (PDVSA) es víctima de un sabotaje tecnológico que estuvo marcado por el bloqueo de los sistemas de automatización existentes para la supervisión y el control de sus industrias. Esto trajo consigo la caída total de la producción de crudo y gas del occidente del país, siendo las compañías proveedoras causantes de esta catástrofe. A raíz de esta situación nace el proyecto Guardián del Alba, el mismo consiste en el desarrollo de un sistema de supervisión y control sobre plataformas libres. Este proyecto tiene como principal objetivo lograr independencia tecnológica, ahorrar millones de dólares que se pagan por licencias de productos propietarios y obtener además personal calificado para dar soporte a este proyecto.

La mayoría de las plantas industriales necesitan un sistema de supervisión y control para asegurar una operación segura y económica de los procesos que se llevan a cabo. El sistema debe ser capaz de traducir los comandos del operador en las acciones necesarias, así como mostrar el estado de la planta. A estos sistemas se les conoce como SCADA, Supervisory Control And Data Acquisition (en español Control Supervisor y Adquisición de Datos). “Un sistema SCADA es un software diseñado para el control de la producción, permite obtener y procesar información de procesos industriales dispersos o en lugares remotos inaccesibles, transmitiéndola a un lugar para supervisión, control y procesamiento, normalmente una sala o centro de control” [1].

En un nivel más simple, la planta podría ser un motor eléctrico que acciona un ventilador de refrigeración. Aquí el sistema de control sería un iniciador del motor eléctrico con protección contra la sobrecarga del motor y fallas en los cables. Los controles del operador serían los botones de inicio / parada y las pantallas de estado de la instalación. Si una luz de alarma se enciende diciendo "nivel bajo de aceite” se espera agregar más aceite para mantener un buen funcionamiento en la planta.

La adquisición y escritura de información de y hacia el campo de producción, son funcionalidades básicas de cualquier sistema SCADA, estas funcionalidades permiten recoger, procesar y almacenar los datos provenientes del campo. Esto se realiza a través de los manejadores de dispositivos.

Un manejador de dispositivo (en inglés, driver), no es más que un programa informático que permite al sistema operativo interactuar con los periféricos, haciendo una abstracción del hardware y permitiendo, mediante una interfaz bien definida, el acceso a los mismos. Los manejadores de dispositivos constituyen

piezas esenciales de los sistemas computarizados, pues sin ellos no se podría usar el hardware [2]. Su función principal en los sistemas de supervisión de industrias es establecer una comunicación entre el SCADA y los diferentes dispositivos del campo. Durante esta comunicación se obtiene información estadística del estado de los dispositivos y canales asociados a la red de campo. La comunicación se rige por un lenguaje denominado protocolo de comunicación.

Un protocolo de comunicación es un conjunto de reglas y procedimientos que proporcionan una técnica uniforme para gestionar un intercambio de información [1]. En la adquisición de datos para los sistemas de supervisión el protocolo define la semántica y estructura de los mensajes que envía el manejador a los dispositivos. Estos mensajes comúnmente contienen solicitudes para las variables configuradas en el SCADA, que representan el comportamiento de los procesos industriales.

Cada variable configurada en el sistema está asociada a un dispositivo y tiene un periodo de muestreo determinado, que no es más que el tiempo establecido para que se actualice su valor de forma periódica. Este tiempo es configurado en dependencia del nivel de criticidad del proceso en cuestión y es de vital importancia para el SCADA que las variables se actualicen en el tiempo establecido. Pues de este tiempo depende la posibilidad para los operadores de monitorear los procesos que se llevan a cabo y poder tomar las medidas necesarias ante cualquier situación extraordinaria.

En las instalaciones de PDVSA se cuenta con una amplia variedad de dispositivos para el control de la producción, entre ellos los llamados Bristol de redes 3000. El SCADA Guardián del Alba presenta dificultades en la adquisición de datos con dichos dispositivos, ya que las variables correspondientes no se actualizan en el periodo establecido. Esto trae consigo que el sistema trabaje con información desactualizada provocando que los operadores desconozcan lo que sucede en la planta con exactitud.

Haciendo un análisis del manejador desarrollado para la comunicación con los dispositivos Bristol su pudo evidenciar que este tiene la responsabilidad de agrupar en bloques las variables que tengan el mismo periodo de muestreo, para de esta manera poder recuperar sus valores en una misma solicitud. Cada vez que se realiza una solicitud de datos se necesita acceder a un canal, que por las características de la comunicación con estos dispositivos es de forma exclusiva. Esto significa que ninguna solicitud puede acceder al canal de comunicación, mientras sea usado por otra, por lo que las solicitudes se realizan de manera secuencial. Cada bloque de variables que se solicita, por alguna razón ocupa el canal un tiempo mayor que el configurado y tiende a retrasar el momento en que se debe solicitar los próximos bloques,

afectando así el proceso de adquisición de los datos.

Para ilustrar la situación antes planteada, es válido mencionar que el manejador actual con una configuración de 3000 variables, el periodo de muestreo real de las variables en un bloque es de más de 30 minutos, situación que limita en gran medida la supervisión y control de los procesos de la planta. No siendo así en el SCADA propietario que se encuentra instalado en la planta y que se pretende sustituir.

Dada la **situación problémica** antes planteada, surge el siguiente **problema científico**:

¿Cómo disminuir la demora en la adquisición de datos para los dispositivos Bristol en el SCADA Guardián del Alba?

Para darle solución a dicho problema se abordó como **objeto de estudio**, el proceso de adquisición de datos en los sistemas SCADA.

Como **objetivo general**: Desarrollar una capa de acceso a datos para dispositivos Bristol que permita disminuir la demora en la adquisición de datos para el SCADA Guardián del Alba.

Como **campo de acción**, el intercambio de mensajes BSAP con los dispositivos Bristol.

Para dar cumplimiento al objetivo general se definieron las siguientes **Tareas de Investigación**:

1. Estudio del protocolo BSAP para comprender su funcionamiento.
2. Estudio del estado del arte de los manejadores para dispositivos Bristol, en los sistemas SCADA.
3. Selección de la tecnología adecuada para la transmisión de datos.
4. Estudio de la arquitectura de los manejadores para el SCADA Guardián del ALBA.
5. Diseño e implementación del manejador para los dispositivos Bristol.
6. Diseño y ejecución de casos de prueba para detectar posibles errores en el manejador.

Por lo que se plantea la siguiente **idea a defender:** El desarrollo de una capa de acceso a datos para dispositivos Bristol que realice un aprovechamiento máximo del protocolo de comunicación BSAP, podría reducir la demora en la adquisición de datos para los dispositivos Bristol en el SCADA Guardián del Alba.

En el cumplimiento de las tareas de investigación se utilizarán varios métodos científicos de investigación como:

Analítico-Sintético: Permitirá conocer los principales fundamentos y teorías relacionadas con la comunicación con dispositivos Bristol, para ello se analizarán varios documentos para la extracción de los elementos más importantes sobre el tema en cuestión.

Modelación: Se realizarán diagramas y modelos que permitan estudiar nuevas relaciones y cualidades del objeto de estudio, y así estructurar teóricamente el sistema.

Histórico lógico: Para conocer la evolución de las teorías y tendencias en el proceso de intercambio de mensajes con dispositivos Bristol.

Capítulo1: Fundamentación teórica

1.1. SCADA Guardia del ALBA.

Guardián del ALBA, está dirigido a cubrir inicialmente las necesidades de la industria petrolera de Venezuela en cuanto a sistemas de supervisión, control y adquisición de datos distribuidos. Fuera del contexto de la industria petrolera de Venezuela, el sistema se visualiza como la solución a ser implantada por cualquier industria de los pueblos del ALBA, tales como industrias de alimentos, mineras, eléctricas, de manufacturas, entre otras.

El SCADA Guardián del ALBA integra las funcionalidades de alto nivel que permiten la solución de aplicaciones de supervisión y control de procesos, utilizando para ello una arquitectura distribuida de módulos que permite escalar a aplicaciones de gran envergadura [3].

Consta de una serie de módulos que trabajan cooperadamente para el funcionamiento del sistema, como por ejemplo: el recolector de la información de campo, la Base de Datos de Tiempo Real (BDTR), encargada del tratamiento de la información recolectada y el módulo IHM (Interfaz Hombre-Máquina). Dichos módulos se encuentran interconectados a través de un software para la distribución de los servicios en la red, conocido como “middleware” o software de comunicación entre aplicaciones.

1.2. Controladores lógicos Programables (PLC)

En los sistemas de control modernos la mayor parte de las operaciones desarrolladas por el subsistema de instrumentación local, con excepción de la captación de variables y la manipulación de dispositivos, son efectuadas por un autómata programable, el denominado Controlador Lógico Programable.

Un PLC es una máquina electrónica diseñada para controlar en tiempo real procesos secuenciales en un medio industrial. Su manejo y programación puede ser realizada por personal electricista, electrónico o de instrumentación sin conocimientos de informática. El PLC realiza funciones lógicas: conversión serie/paralelo, temporizaciones, conteos y otras funciones más potentes como cálculos, regulación, emisión de comandos, etc. El PLC dispone también de facilidades de comunicación para acceder a un subsistema de comunicaciones.[1]

Su uso es extensivo en el control de una gran variedad de procesos industriales de distinta magnitud y su aplicación va desde la automatización de máquinas de fabricación y líneas de ensamblaje en un proceso

aislado, hasta aplicaciones que requieran comunicación en red de PLCs, ordenadores y otros dispositivos de control, permitiendo una integración y manejo total de la información en planta.

El potencial de los PLC en el mejoramiento de los procesos industriales se basa fundamentalmente en las siguientes cualidades:

- Alta confiabilidad
- Alta integración
- Simplificación del cableado
- Mayor flexibilidad y funcionalidad en los procesos controlados
- Alta velocidad de respuesta del sistema
- Comunicación en red
- Bajo costo

1.3. Dispositivos Bristol

Los dispositivos Bristol que en su mayoría son PLC cumplen con los requerimientos para cualquier aplicación de control industrial. Pueden ser instalados fácilmente en lugares remotos para aplicaciones de control. Normalmente presentan procesadores ARM de alta velocidad y su consumo de energía es bajo. Pueden trabajar en un amplio rango de temperatura (-40 a +70ºC) y utilizan como protocolo de comunicación BSAP (Bristol Babcock Synchronous/Asynchronous Communication Protocol) y Ethernet IP. Para la comunicación soporta las interfaces RS232, RS485 y Ethernet.

Existe gran variedad de dispositivos Bristol, entre ellos Bristol ControlWave Express, DPC 3330, DPC 3335, GFC 3308-x, RTU 3305, RTU 3310.

Figura 1 Dispositivo Bristol ControlWave Express

Todos los dispositivos de redes 3000 utilizan ACCOL, (Advance Communication and Control Oriented Language de Bristol Babcock). ACCOL es un lenguaje multitarea que permite a un dispositivo Bristol realizar control continuo y discreto por lotes, así como medición, cálculos de flujo, gestión de alarmas y almacenamiento de datos.

ACCOL incluye un conjunto de más de 90 módulos pre-programados que se pueden combinar para formar una medición completa y estrategia de control para aplicaciones de SCADA y automatización.

1.4. Protocolo BSAP.

El protocolo BSAP brinda soluciones para la comunicación con los dispositivos Bristol Babcock de redes 3000. Está orientado a la comunicación por encuestas y ofrece una seguridad alta para sus mensajes, ha sido diseñando de acuerdo al modelo OSI, donde cada capa tiene una función lógica propia y es operable bajo la modalidad síncrona y asíncrona.

BSAP es un protocolo propietario de red, el cual es aplicado para topologías de árbol donde cada nodo tiene una dirección única basada en su posición en la red y puede ser maestra de los niveles inferiores o esclava de los niveles superiores. Este protocolo puede soportar diversas configuraciones y más de un nodo por cada uno de los seis niveles que son permisibles en el árbol. Los mensajes pueden ser enviados

entre nodos del mismo nivel hasta nodos de otros niveles, cada mensaje contiene una identificación única y agrega un sistema de corrección de redundancia cíclica CRC.

1.4.1 Estructura jerárquica de red.

El protocolo BSAP soporta una estructura de árbol donde el nodo raíz juega el papel del nodo maestro y está ubicado en el primer nivel del protocolo. Los nodos que se posicionan en el segundo nivel del árbol forman parte de una red de nodos esclavos del nodo maestro que está en el primer nivel. De esta manera se va formando la red, donde el nodo del nivel enésimo es el maestro de los nodos hijos del nivel enésimo +1 como se muestra en la Figura 2.

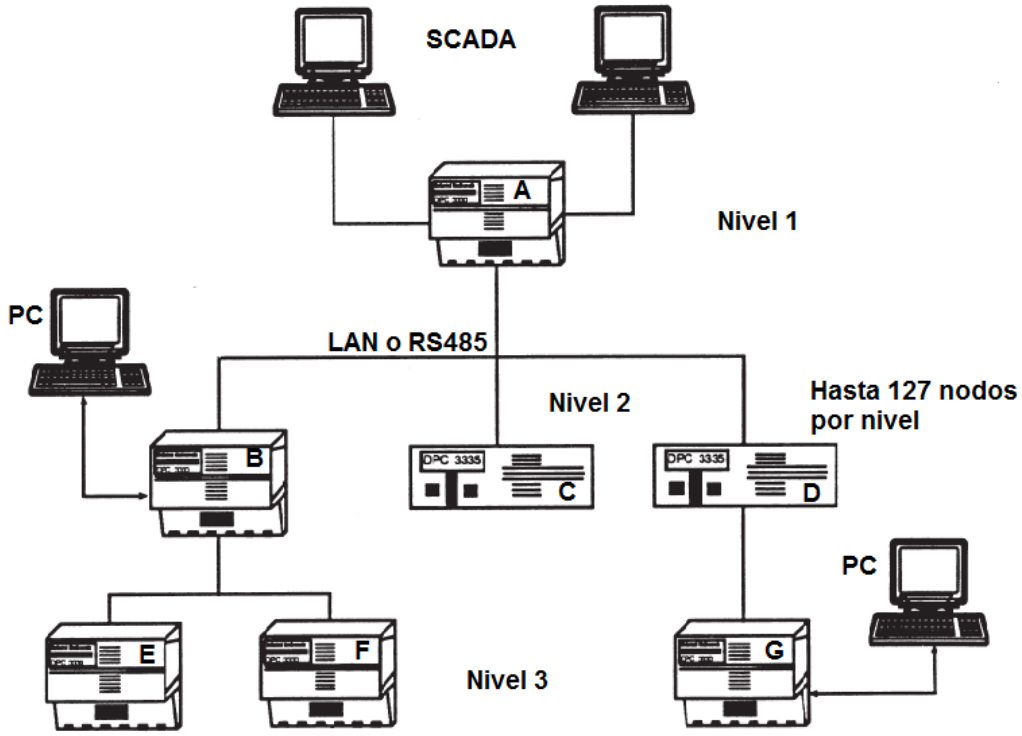

Figura 2 Estructura jerárquica del protocolo BSAP.

El número de nodos esclavos que se pueden colocar en un nivel está condicionado por dos factores: El primer factor se debe a que la cantidad de nodos existentes deben permitir un tiempo de respuesta óptimo para los mensajes críticos.

El número de nodos permisibles se condiciona al direccionamiento físico de la dirección nodal (7 bits= 127 nodos). Por ser el protocolo BSAP un protocolo orientado a interrogaciones, el tiempo que tarde en llegar un mensaje de un nodo a otro, está condicionado con el tiempo de interrogación, por lo que se le da mayor prioridad a mensajes relacionados con las alarmas. Además de la limitación con el proceso de interrogación está la configuración del direccionamiento de los nodos. Cada nodo tiene una dirección única la cual es basada en una secuencia dependiendo de su posición en la red como su posición en un nivel dado. Para cada nodo hay una secuencia en las posiciones que está dentro del rango de 1 a 127 y estas posiciones se definen como direcciones locales. Las direcciones locales no tienen razón para ser consecutivas, se puede configurar la red con ciertos huecos y luego colocar en ese espacio otro dispositivo sin necesidad de volver a cambiar el direccionamiento. Hay que tener en cuenta que el número más alto utilizado determina la cantidad de espacio requerido para definir la próxima dirección del nodo interno. Los niveles de las direcciones locales se concatenan para dejar una única dirección para cada nodo en la red. Esta dirección única es conocida como Dirección Global. El valor de esta dirección no puede exceder 32767 [4].

1.4.2 Relación de los nodos en la red.

Cualquier nodo dentro de la red (excepto los extremos: nivel 0 y nivel último) tiene un doble papel: puede ser maestro de sus nodos inferiores o puede ser esclavo del nodo inmediatamente superior. Esta doble relación se define como una “relación local”, pues los nodos en cuestión son verticalmente adyacentes entre sí. Se denomina entonces “mensajes locales” al intercambio de información entre un nodo maestro y un esclavo o nodo sin pasar por ningún otro nodo; en este caso se aplica las direcciones locales. Los mensajes que pasan por uno o más nodos hasta alcanzar su destino, se denominan “mensajes globales” en donde se aplica las direcciones globales [1]. Por ejemplo, en la Figura. 2, un mensaje de B a E, o de B a F, son mensajes locales; mientras que mensajes desde A hasta G son mensajes globales. En el epígrafe 1.4.5.1 se describe la estructura de estos mensajes.

1.4.3 Formatos generales de comunicación.

El protocolo BSAP hace referencia a las variables como señales y estas pueden ser accedidas mediante su nombre o dirección MSD (en inglés Master Signal Database). El nombre para cada señal tiene el formato Base.Extensión.Atributo, en este caso hay que tener en cuenta que los tres campos pueden ser

de longitud variable terminado en el carácter nulo. La base tiene como longitud máxima 9 caracteres ASCII, 7 caracteres ASCII máximo en caso de la extensión y 5 en caso del atributo. Sin embargo las direcciones MSD se representan con un número entero asociado a la dirección física de la señal en el dispositivo, ocupando solamente 2 bytes en el buffer de transferencia.

Cada dispositivo Bristol posee una base de datos donde se guarda la información para todas sus señales. Cada señal guarda su información en distintos campos, entre ellos su valor, su dirección MSD y el tipo de la señal que puede ser analógica o digital. Además el protocolo brinda la posibilidad de especificar los campos deseados en cada petición. De esta manera se controla la cantidad de datos transmitidos por la red, ya que una vez conocida la dirección MSD para una señal no es necesario volver a solicitarla. Esto permite minimizar la cantidad de datos a transmitir y por tanto un mejor aprovechamiento de la comunicación.

1.4.4 Capas del protocolo.

BSAP se ha diseñado e implementado de acuerdo con las normas del modelo OSI. Este modelo define claramente las interfaces entre cada capa permitiendo que diferentes sistemas operativos y protocolos de red puedan trabajar juntos.

El protocolo BSAP utiliza las cuatro capas inferiores del modelo OSI, estas se describen a continuación:

Capa de transporte: es responsable de la correcta transmisión del mensaje a nivel funcional. Cuando la capa de transporte determina que está listo para transmitir, el control se pasa a la siguiente capa.

Capa de red: Tiene la responsabilidad de determinar la ruta del mensaje a través de la red y las direcciones que debe utilizar.

Capa de enlace de datos: Es responsable de mejorar el mensaje para incluir la comprobación de errores y mecanismos de corrección. También controla el acceso al canal físico sobre el cual se envía el mensaje.

Capa física: Esta capa se compone principalmente del hardware y el software necesario para su control. Esta capa es totalmente independiente del formato final del mensaje que se transmite.

La ventaja de esta arquitectura es que, al aislar las funciones de comunicación de la red en capas, se minimiza el impacto de cambios tecnológicos en el protocolo, es decir, podemos añadir nuevas

aplicaciones sin cambios en la red física y también podemos añadir nuevo hardware a la red sin tener que reescribir el software de aplicación.

1.4.5 Formato general de las tramas.

Fundamentalmente, este protocolo tiene dos clases de tramas: las tramas de información y las tramas de supervisión y control.

1.4.5.1. Tramas de información:

Las tramas o mensajes de información se dividen en Mensajes de Datos Globales y Mensajes de Datos Locales. El formato del mensaje global tiene la siguiente estructura:

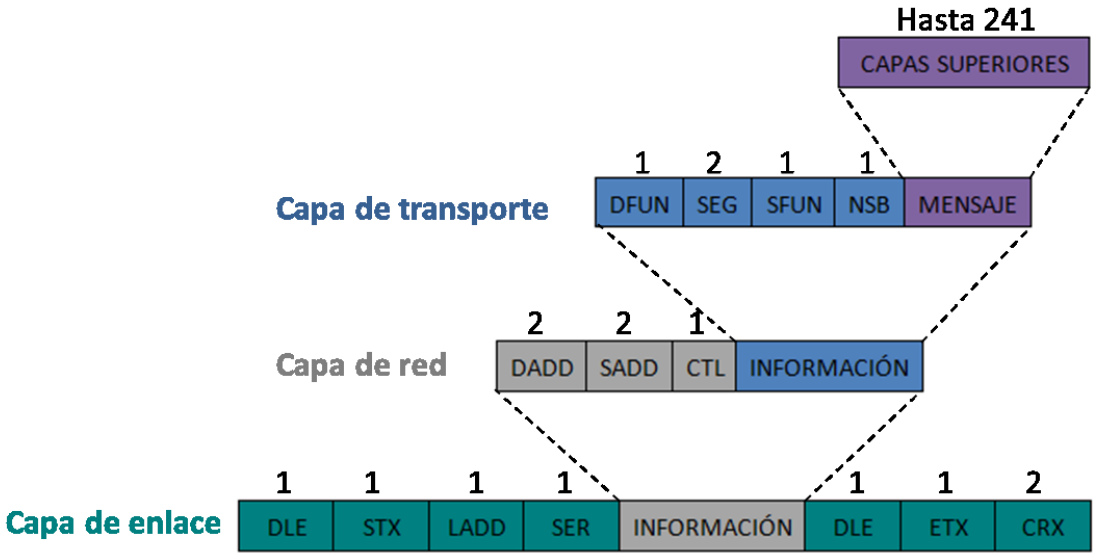

Figura 3 Formato de los mensajes globales

El mensaje global presenta varias cabeceras que son distribuidas por las capas del modelo OSI. Cuando se ensambla un paquete global la capa superior contiene el mensaje a transmitir y la capa del protocolo contiene la información para que el mensaje sea enrutado por la red dispositivos con éxito.

La capa de transporte contiene los siguientes campos:

Código	Descripción
DFUN	Código de función del destino

SEQ	Número de secuencia del mensaje. Se requiere para evitar la duplicación accidental de mensajes en condiciones de ruido extremo.
SFUN	Código de función de la fuente. El número de funciones puede llegar hasta 20.
NSB	Octeto de Estado del Nodo. Si hay una falla en la comunicación, el dígito de mayor peso del octeto se pone a UNO y los otros 7 dígitos indican el tipo de falla.
MENSAJE	Depende de la aplicación y puede contener hasta 241 octetos

La capa de Red aporta la cabecera siguiente:

Código	Descripción
DADD	Dirección de destino global.
SADD	Dirección de fuente global. Nótese que DADD y SADD son las direcciones de las maestras destino y origen, respectivamente.
CTL	Octeto de control. Contiene el tipo de mensaje (petición o respuesta) y el estado de la respuesta o número de nivel si hubo fallas.

A nivel de Capa Enlace:

Código	Descripción
DLE	Caracter ASCII10H
STX	Caracter ASCII 02H
ETX	Caracter ASCII 03H
LADD	Dirección local. El dígito de mayor peso es una bandera que cuando está en UNO, indica que el mensaje contiene una estructura a nivel de red. LADD es la dirección del esclavo hacia dónde va dirigido el mensaje.
SER	Número de serie del mensaje. Este número es asignado por el maestro; el esclavo debe retornarlo en la respuesta. Se utiliza para evitar la confusión que se presenta cuando un mensaje o reconocimiento se pierde debido al ruido, etc. El número de serie cero no se utiliza porque está reservado para uso interno en el nodo.
CRC	Campo para la verificación de error. Se calcula desde STX a ETX y no toma en cuenta los

DLE intermedios. Utiliza el algoritmo CRC UIT-T V.41.

El formato para los mensajes locales es el siguiente:

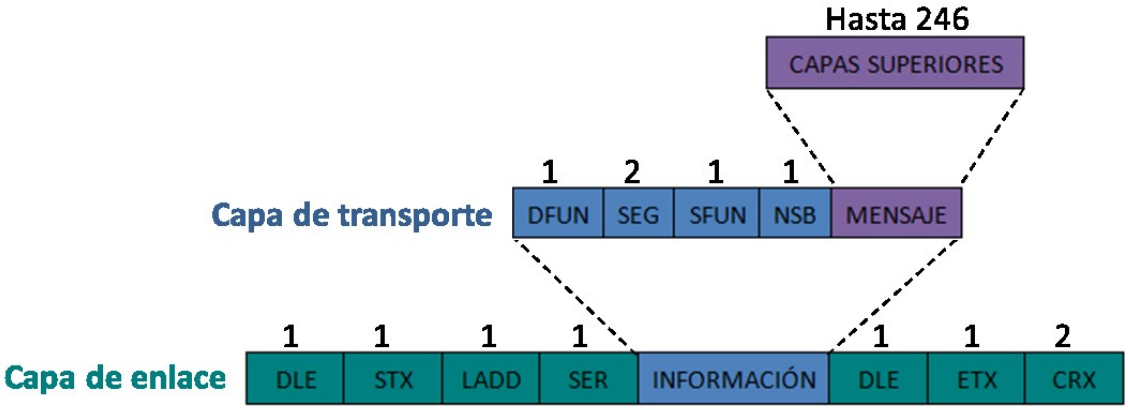

Figura 4 Formato para mensajes locales.

La trama de datos para mensajes locales a diferencia de la trama global no contiene capa de red, para este caso el digito de mayor peso en la dirección local LADD es puesto en cero; los otros siete dígitos indican la dirección local de 1 a 127. La dirección del maestro por lo general es cero.

1.4.5.2. Mensajes de Supervisión y Control

Las tramas de supervisión y control más utilizadas son las siguientes:

1. Mensaje de Interrogación (Poll), Código 85H.
2. Mensaje de Reconocimiento (ACK o DOWN-ACK), Código 86H.
3. Mensaje de Reconocimiento/“Nada para Transmitir” (ACK-NO DATA), Código 87H.
4. Mensaje de Reconocimiento Negativo (NAK), Código 95H.
5. Mensaje de Reconocimiento desde arriba (UP-ACK), Código 8BH.
6. Último Mensaje Descartado (DIS), Código 83H.

Mensaje de Interrogación (Poll), Código 85H.

Este mensaje es utilizado por el nodo maestro para interrogar a sus nodos esclavos y determinar si están activos. De estar activos, se les solicitará información con el mensaje respectivo. El formato del mensaje es el siguiente:

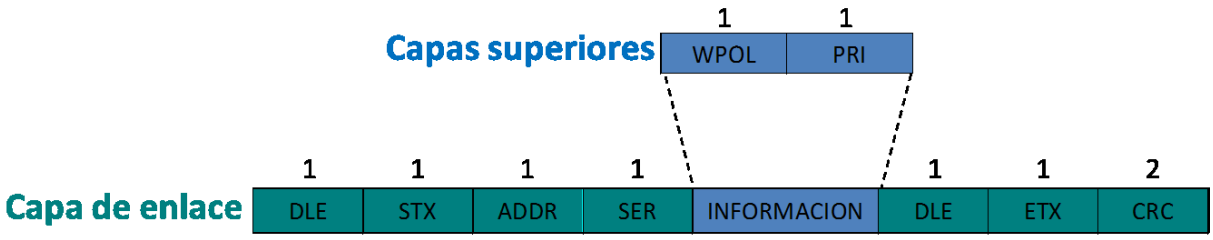

Figura 5 Formato de mensaje de interrogación.

Código	Descripción
ADDR	Dirección de la esclava o nodo interrogado
SER	Número de serie del mensaje
WPOLL	Código de Función 85H
PRI	Prioridad de los datos requeridos. PRI = 00H indica que se puede aceptar alarmas o mensajes de datos. PRI = 10H indica que no se puede aceptar alarmas

Reconocimiento (ACK o DOWN-ACK), Código 86H.

Este mensaje es una respuesta; la utiliza el nodo esclavo para reconocer a su maestro la recepción de un mensaje, excepto cuando se trata de una Interrogación (Poll). El formato correspondiente tiene la siguiente forma:

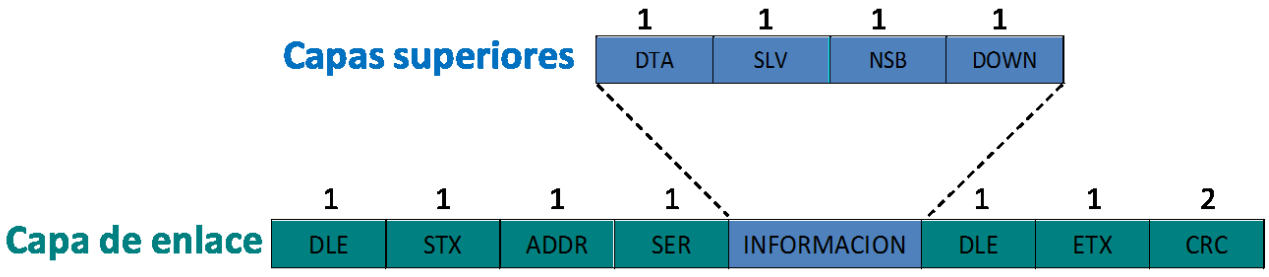

Figura 6 Formato de mensajes de reconocimiento (ACK)

Código	Descripción
ADDR	Dirección de la maestra (siempre a CERO).
SER	Número de serie del mensaje reconocido.
DTA	Código de Función 86H.
SLV	Dirección local del nodo esclavo o nodo que responde.
NSB	Octeto de Status del nodo esclavo. Con este octeto se le notifica al maestro ciertas condiciones existentes dentro del nodo esclavo.
DOWN	Número de buffers en uso.

Reconocimiento/Nada para Transmitir (ACK - NO DATA), Código 87H.

Este mensaje es utilizado por un nodo esclavo para reconocer la recepción de una Interrogación (Poll) indicando que no tiene mensajes de datos para transmitir.

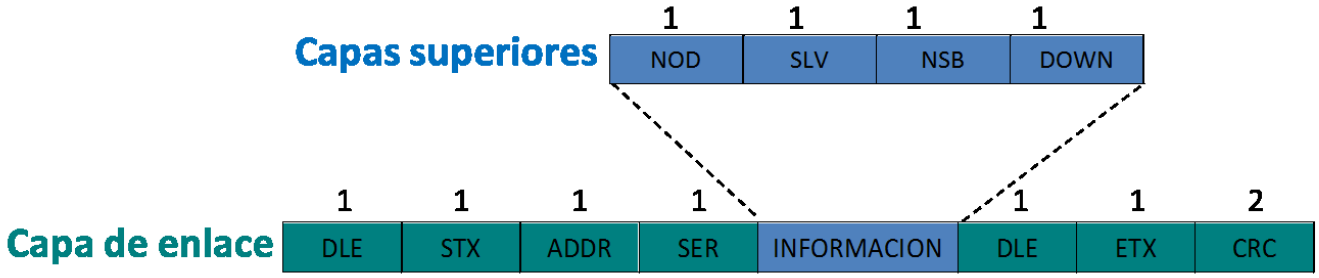

Figura 7 Formato de mensaje de reconocimiento/Nada para Transmitir

Código	Descripción
NOD	Código de Función 87H.

Reconocimiento/No se dispone de espacio en el buffer (NACK), Código 87H.

Con este mensaje de respuesta, el nodo esclavo le indica a su maestro que además de una Interrogación (Poll) ha recibido también otro mensaje pero no dispone de espacio en los buffers.

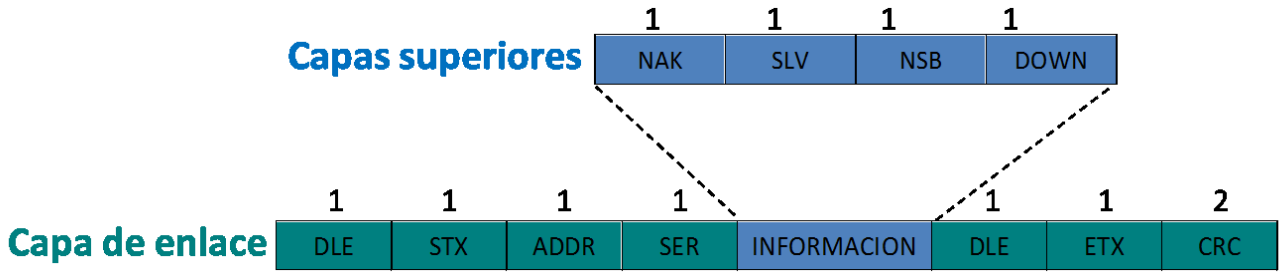

Figura 8 Formato de mensaje Reconocimiento/No se dispone de espacio en el buffer.

Código	Descripción
SER	Número de serie del mensaje reconocido con NAK.
NAK	Código de Función 95H.

Reconocimiento desde arriba (UP- ACK), Código 8BH.

Este mensaje es utilizado por el nodo maestro para informar al esclavo que ha recibido correctamente y almacenado el mensaje.

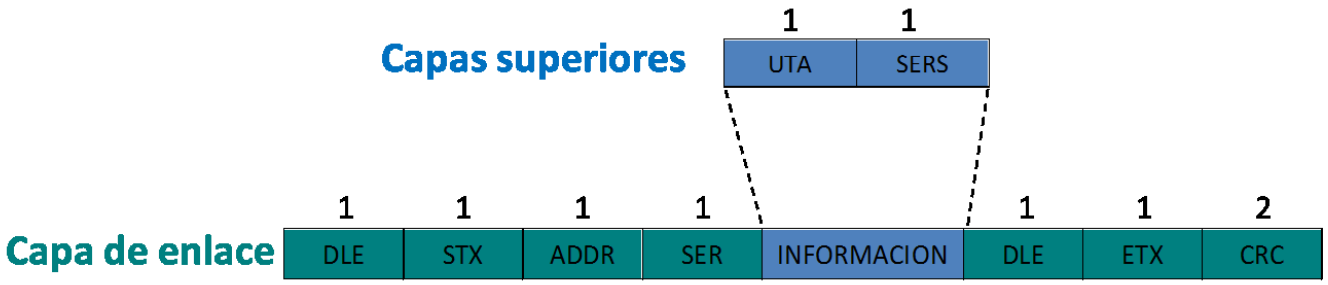

Figura 9 Formato de mensajes de reconocimiento desde arriba

Código	Descripción
ADDR	Dirección local de la esclava.
ERM	Número de serie del mensaje de la maestra.
UTA	Código de Función 8BH.
SERS	Número de serie del mensaje reconocido con UP-ACK.

1.5. Flujo de mensajes.

El nodo maestro inicia el flujo de mensajes haciendo una solicitud al nodo esclavo. Si el direccionamiento es global, el mensaje debe pasar por varios nodos en el camino a su destino, la solicitud se dirige a su nodo intermedio adecuado.

El nodo intermedio responde con un mensaje 'DOWN TRANSMIT ACK' a su maestro con el mismo número de serie de la petición y como dirección local '0', que indica que es el maestro local. El mensaje con la petición entonces es enrutado hacia los próximos niveles con la nueva dirección local generada por el nodo intermedio. [4]

Si la respuesta del esclavo no es inmediata, el maestro local debe interrogar a su esclavo para una respuesta. Si el esclavo responde con 'NO DATA TO TRANSMIT', esto significa que no tiene información disponible terminando la secuencia de interrogación. Cuando el esclavo responde a una interrogación el maestro envía a su esclavo un 'UP TRANSMIT ACK' indicando que recibió la trama satisfactoriamente y terminar el flujo de mensajes. O puede responder con 'UP TRANSMIT ACK' con una interrogación para solicitar más información. [4]

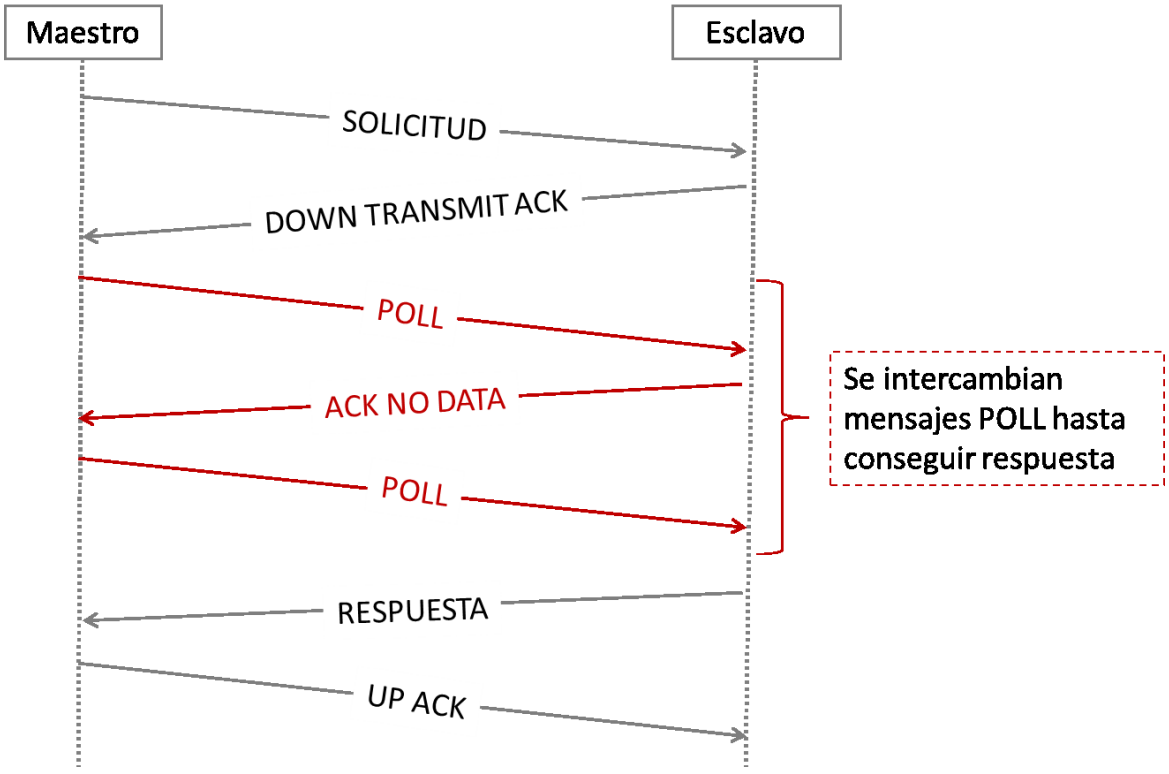

Figura 10 Secuencia de mensajes para una solicitud de datos exitosa.

Después de completar el procesamiento en el nodo de destino global debe enviarse una respuesta al autor del mensaje. Para ello, las direcciones de los mensajes origen y destino se intercambian, haciendo que el mensaje de respuesta retorne a su remitente.

1.6. Tendencias y desarrollo del protocolo BSAP.

Muchos SCADAs utilizan herramientas que proveen una interfaz de comunicación para los dispositivos Bristol, de estas herramientas las más conocidas son: OpenBSI, desarrollada por Bristol Babcock y *Universal Server* de *MTL Open System Technologies LP*. Sin embargo otros SCADAs implementan el protocolo BSAP para comunicarse con este tipo de dispositivos, como es el caso de Oasys de la compañía *Valmet Automation.*

1.6.1 Open Bristol system interface (OpenBSI).

OpenBSI es un software de comunicación que proporciona acceso a la red de dispositivos Bristol Babcock. En la base cuenta con un servidor de comunicaciones para Windows 98/NT y 2000 a través del cual otras aplicaciones cliente se comunican con las redes Bristol. Provee un conjunto de herramientas que se comunican a través del servidor para recopilar y gestionar los datos recogidos en la red, además de generar archivos basados en datos históricos, capturar las alarmas, supervisar y controlar las comunicaciones.

Entre las herramientas principales se encuentran:

LocalView: Este programa permite establecer comunicación localmente con el controlador. A través de LocalView, otros programas de aplicación intercambian datos con él. [5]

NetView: Permite crear una arquitectura de control la cual da la posibilidad de monitorear la conexión en tiempo real de las variables, ver la programación en cada variable, modificar los parámetros de manera remota sin la necesidad de ir al sitio; es decir hay el control remoto y acceso a los datos, configuración, etc. sobre todos las variables. [5]

DataView: Permite recolectar diferentes tipos de información de los dispositivos Bristol. Señales, Listas, arreglos, archivos y registros de alarmas, así como búsqueda con criterios específicos pueden ser realizadas. DataView también permite cambiar los valores de las señales en línea y modificar las banderas

de estado. Es un programa para trabajar en línea con los datos.[5]

StandAlone OPC Server: proporciona una conectividad de OPC (OLE for Process Control) a la familia de controladores Bristol Babcock, de esta manera OpenBSI se puede comunicar con los distintos SCADAs. Se entiende por OPC como la conectividad abierta dentro la automatización industrial que garantiza la interoperabilidad mediante la creación y mantenimiento de especificaciones de estándares abiertos.[5]

1.6.2 Universal server.

Universal Server es un software desarrollado para sistemas basados en Windows, que brinda soporte para varios dispositivos utilizando protocolos como DF1, Modbus y BSAP. Además cuenta con un servidor OPC para transmitir los datos recolectados a cualquier sistema de supervisión y control.

Universal Server provee un excelente soporte para dispositivos Bristol Babcock usando el protocolo BSAP. El módulo BSAP permite al usuario crear configuraciones para dispositivos que usan este protocolo y soporta los dispositivos 3305, 3310, 3330, 3335, 3340 y 3530.

1.6.3 OASyS.

OASyS es un SCADA, que incorpora aplicaciones interoperativas conectadas a través de interfaces estándares. TCP/IP sobre Ethernet y "Valmet´s SQL Backbone Middleware" proveen conexión directa a terminales de servidores, Sistemas Comerciales de Manejo relacional de Base de Datos (RDBMSs), y compuertas (gateways). [6]

OASyS es compatible con diversas plataformas de "software": Microsoft Windows NT y Windows 95/98, Digital UNIX, AIX, HP-UX, y Solaris. Estructuralmente OASyS está basado en tres elementos funcionales: Interfaz al Usuario, Manejador de Base de Datos, y Herramientas de Trabajo. Todos los elementos de OASyS están interconectados vía SQL Backbone; es decir en un ambiente cliente servidor para el transporte de datos y aplicaciones en tiempo real dentro de una arquitectura distribuida de OASyS y externamente incorpora a los otros subsistemas intranet. [6]

Implementa gran variedad de protocolos entre ellos BSAP, este es utilizado para la comunicación con dispositivos de redes 3000. Para el transporte de los datos utilizan una biblioteca llamada libcomm que se integra con la arquitectura distribuida del SCADA garantizando la comunicación de manera eficiente.

1.6.4 Selección de herramientas existentes.

Realizado el estudio de algunas herramientas existentes que implementan el protocolo BSAP se llega a la conclusión de que su uso no es conveniente en el proyecto Guardián del ALBA. Ya que no cumplen los principales requerimientos del proyecto. La mayoría brinda sus funcionalidades solo sobre plataforma Windows, además de ser propietarios y de código cerrado. Por lo que se necesita un manejador que se integre al módulo de recolección del SCADA Guardián del ALBA, usando herramientas libres y multiplataforma.

1.7. Tecnologías para el transporte de datos.

Para la adquisición de datos en redes industriales es necesario el uso de tecnologías que permitan el transporte de los datos entre los SCADAs y los dispositivos de campo a través de un medio físico determinado. Para ello se analizaron las bibliotecas Asio C++ Library, QextSerial y TransportProvider por ser tecnologías multiplataforma y de libre acceso.

1.7.1 Asio C++ Library.

Asio es una biblioteca de C++ libremente disponible y multiplataforma que se utiliza para la implementación de aplicaciones de red, brinda la posibilidad de desarrollar un modelo asíncrono para la Capa de Transporte de forma sencilla y natural. Da soporte para la resolución de direcciones, aceptación de nuevas conexiones, socket orientados a datagrama y funcionalidades de temporizador.

A partir de la versión 1.3.5 la biblioteca Boost incluyó a la biblioteca Asio dentro de sus espacios de nombre (Boost::Asio), lo que demuestra el nivel de robustez y madurez de dicha biblioteca. Boost::Asio incorporó las funcionalidades para el manejo del puerto serie, evitando así el uso de una biblioteca adicional. [4]

1.7.2 Biblioteca de transporte TransportProvider.

En el módulo de recolección del proyecto Guardián del Alba se ha desarrollado una biblioteca basada en

herramientas libres que permite realizar de forma eficiente la conexión al medio físico. La misma permite la lectura y escritura de datos a través de redes Ethernet, sobre protocolos de nivel de transporte TCP/IP, UDP/IP o buses seriales RS232. Basada en Boost::Asio la biblioteca TransportProvider contiene un conjunto de funcionalidades que permiten la abstracción en el intercambio de información entre diferentes nodos dentro de una red.

Actualmente la biblioteca TransportProvider se distribuye en forma de biblioteca dinámica (.so ó .dll), es multiplataforma y por su bajo acoplamiento con otros módulos puede reutilizarse en el desarrollo de diversas aplicaciones. [2]

1.7.3 QextSerial.

QextSerial es una biblioteca desarrollada con el framework Qt, es multiplataforma y brinda las principales funciones para la comunicación serie, posee una clase llamada QextSerialPort que encapsula las funcionalidades de un puerto serie para sistemas UNIX y Windows. Aprovechando las ventajas del framework Qt, la biblioteca QextSerial hace un correcto y eficiente uso de los puertos series para el intercambio de datos entre los distintos dispositivos que conforman una red.

Capítulo2: Características del sistema.

2.1. Herramientas seleccionadas para el desarrollo

A raíz del estudio de las posibles tecnologías a utilizar para el transporte de los datos para el manejador propuesto, se decide utilizar TransportProvider por ser una biblioteca multiplataforma, extensible y flexible, además de tener una interfaz intuitiva y fácil de utilizar, brinda soluciones que ya han sido debidamente avaladas y probadas y ha sido utilizada en el desarrollo de varios manejadores para el SCADA Guardián del ALBA entre ellos Modbus, DF1 y DNP3.

En el proyecto Guardián del Alba, se hizo una selección de herramientas libres y multiplataforma para el desarrollo de manejadores. Herramientas que garantizan una buena integración para la gestión de la configuración tanto del código fuente como de la documentación, el caso del IDE de desarrollo con un buen completamiento de código y que brinda facilidades en la vinculación de bibliotecas y en la configuración del compilador. Los siguientes subepígrafes describen las características de las herramientas seleccionadas.

2.1.1 Entorno Integrado de Desarrollo Eclipse.

Eclipse es una plataforma de herramienta universal, un IDE *(Integreted Development Environment* por sus siglas en inglés*)* de código abierto y extensible, para todo y nada en particular. Su valor real proviene de sus plug-ins que se integran a la plataforma permitiendo el uso de herramientas aprovechando las facilidades y comodidades que provee Eclipse, ejemplo de ello tenemos herramientas para desarrollo Web, herramientas para el diseño de clases, herramientas para el control de versiones, entre otras. La plataforma es muy flexible y extensible.

Eclipse cuenta con un gran corrector de errores que, aparte de identificar las palabras reservadas del lenguaje también puede detectar, y marcar sobre el código de un programa, los lugares donde se pueden producir errores de compilación o de sintaxis. Posee un excelente sistema para completar códigos facilitando el trabajo al programador. Eclipse incorpora una herramienta para realizar automáticamente el formateo del código de acuerdo a unos criterios preestablecidos.

La compilación es una tarea que se lanza automáticamente al guardar los cambios realizados en el

código. Por esta razón es prácticamente innecesario controlar manualmente la compilación de los proyectos. También integra un cliente para el sistema de gestión de versiones CVS. Asimismo, a través de "plug-ins" libremente disponibles es posible añadir otros como Subversion y Git.

2.1.2 Lenguaje de programación C++

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. En este sub-epígrafe se exponen algunas de sus principales características.

Orientado a objetos: La posibilidad de orientar la programación a objetos permite al programador diseñar aplicaciones desde el punto de vista como una comunicación entre objetos en lugar de una secuencia estructurada de código. Además, permite una mayor reutilización de código en una forma más lógica y productiva.

Portabilidad: Prácticamente se puede compilar el mismo código C++ en casi cualquier tipo de ordenador y sistema operativo sin realizar ningún cambio. C++ es el lenguaje de programación muy utilizado en el mundo.

Velocidad: El código resultante de una compilación de C es muy eficiente, por efecto de su dualidad como de alto nivel y lenguaje de bajo nivel y el tamaño reducido del propio lenguaje, pudiéndose utilizar tanto para escribir software de bajo nivel, como drivers y componentes de sistemas operativos

Standard Template Library: Posee una serie de bibliotecas de funciones integradas para la manipulación de datos a nivel más básico. En C++, además de poder usar las bibliotecas de C, se puede usar la nativa STL (Standard Template Library), propia del lenguaje. Proporciona una serie plantillas (templates) que permiten efectuar operaciones sobre el almacenado de datos, procesado de entrada/salida.

2.1.3 Herramienta de modelado Visual Paradigm for UML

Visual Paradigm for UML es una herramienta CASE para modelamiento UML muy potente y de fácil uso. Te permite dibujar todo tipo de diagramas UML, revertir código fuente a modelos UML, generar código fuente desde los diagramas UML, solo por mencionar algunas de sus características.

Visual Paradigm for UML incluye los objetos más recientes de UML además de diagramas de casos de uso, diagramas de clase, diagramas de componentes, reversa instantánea para Java, C++, DotNet Exe/dll, XML, XML Schema, y Corba IDL, ofrece soporte para Rational Rose, integración con Microsoft Visio, además permite generar reportes y documentación en HTML/PDF.

2.1.4 Metodología de desarrollo OpenUP

Se ha seleccionado como metodología de desarrollo OpenUP por ser una metodología ágil que aplica un enfoque iterativo e incremental dentro de su estructura del ciclo de vida y que puede adaptarse para desarrollar diversos tipos de proyectos.

OpenUP es apropiado para proyectos pequeños y de bajos recursos permite disminuir las probabilidades de fracaso en los proyectos pequeños e incrementar las probabilidades de éxito. Permite detectar errores tempranos a través de un ciclo iterativo, evita la elaboración de documentación, diagramas e iteraciones innecesarios requeridos en la metodología RUP. Por ser una metodología ágil tiene un enfoque centrado al cliente y con iteraciones cortas. [7]

OpenUP cuenta con 4 fases de desarrollo como se muestra en la figura.

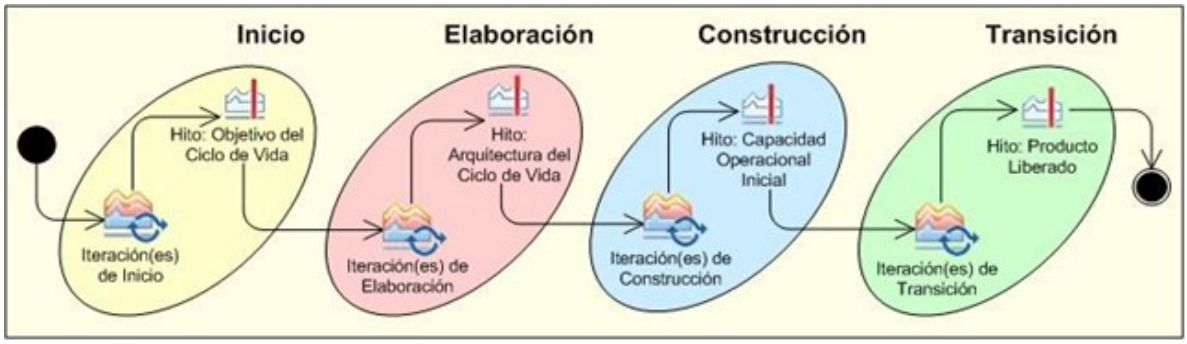

Figura 11 Fases de la metodología de desarrollo OpenUP.

Concepción: Primera de las 4 fases en el proyecto del ciclo de vida, acerca del entendimiento del propósito y objetivos y obteniendo suficiente información para confirmar qué debe hacer el proyecto. El

objetivo de ésta fase es capturar las necesidades de los stakeholders en los objetivos del ciclo de vida para el proyecto.

Elaboración: Es el segundo de las 4 fases del ciclo de vida del OpenUP donde se tratan los riesgos significativos para la arquitectura. El propósito de esta fase es establecer la base de la elaboración de la arquitectura del sistema.

Construcción: Esta fase está enfocada al diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo. El propósito de esta fase es completar el desarrollo del sistema basado en la Arquitectura definida.

Transición: Es la última fase, su propósito es asegurar que el sistema es entregado a los usuarios, y evalúa la funcionalidad y rendimiento del último entregable de la fase de construcción.

2.2. Arquitectura de manejadores

Para la implementación de los manejadores en el SCADA Guardián del Alba se ha definido una estructura multicapa que separa, desde el punto de vista lógico, la capa de transporte, la capa de protocolo y la capa driver (Figura 12). A esta estructura se le asocia un conjunto de clases llamadas DriverCore que facilitan la implementación de los drivers y garantizan la reusabilidad de las diferentes capas en múltiples contextos.

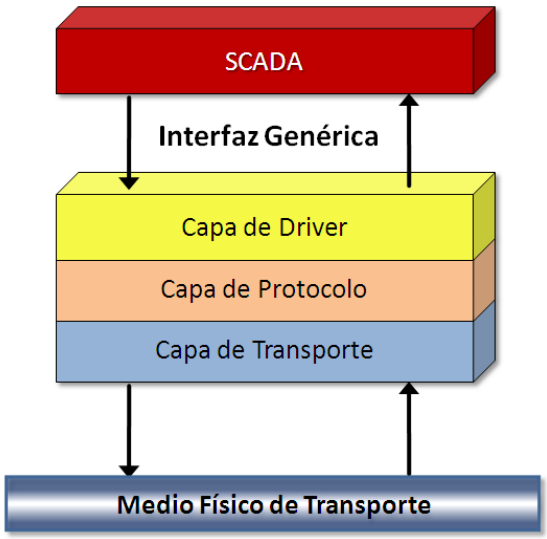

Figura 12 Arquitectura de los manejadores

La capa de transporte es la encargada de manejar la conexión y los detalles de la transmisión del flujo de

datos a través del medio físico. La capa del protocolo proporciona una serie de funciones que encapsulan la construcción e interpretación de los mensajes necesarios para la comunicación con los dispositivos de acuerdo a las reglas del protocolo. Por último la capa driver debe traducir la interfaz del sistema de supervisión y control en términos de llamadas a la biblioteca del protocolo. Esta separación lógica facilita el desarrollo y el mantenimiento de los manejadores de dispositivos.

2.3. Interfaz genérica.

Es difícil que un SCADA contemple, en su código, todos los protocolos posibles para la gran variedad de dispositivos que presenta una planta. Por el contrario lo común es crear un protocolo general (o unos pocos) y la tarea de traducir este protocolo general a los protocolos específicos se le encarga a los manejadores, que, como regla, se programan en módulos independientes al SCADA. [8]

Por esta razón se desarrolló una interfaz genérica que permitiera abstraer al SCADA de las diferencias entre los protocolos y características de los dispositivos que con él se enlazan. A partir de esta interfaz se han desarrollado los manejadores para los protocolos Modbus, OPC, EtherNet/Ip, ABInterchange de Allen Bradley, entre otros.

La Interfaz genérica ha demostrado ser lo suficientemente flexible como para asimilar protocolos de cualquier naturaleza. Por su diseño, la Interfaz genérica posibilita que el recolector de variables sea altamente escalable, ya que no necesita de la creación de contextos de ejecución en cada driver.[8]

2.4. Biblioteca DriverCore.

En la concepción de los manejadores para el SCADA “Guardián del ALBA” hay un conjunto de características y conceptos que son comunes para todos los manejadores. Las abstracciones sobre los conceptos de variables, dirección de una variable, dispositivo, manejador, bloque de variables, etc.; se encuentran implementados en un componente de software que se conoce a nivel de proyecto por el nombre DriverCore.[2]

DriverCore implementa la Interfaz genérica y define la arquitectura de los manejadores, brinda un conjunto de clases abstractas que encapsulan los conceptos y soluciones comunes en el desarrollo de manejadores. Fue desarrollado con el propósito de acelerar el proceso de desarrollo, permitiendo reutilizar

código y promover buenas prácticas de programación como el uso de patrones de diseño. Carga de manera opcional la biblioteca TransportProvider y provee un conjunto de funcionalidades como:

- Introspección de manejadores y dispositivos
- Manejo de las direcciones
- Clasificación de las variables en bloques de lectura o escritura
- Entrada / Salida asíncrona.

Este modelo permite un máximo de 24 manejadores cargados de forma simultánea en el DriverCore, no más de 32 propiedades de configuración por objeto, ya sea driver o dispositivo y no más de 32 parámetros de diagnóstico por objeto (driver o dispositivo). La biblioteca se encuentra en su versión 5 y ha sido utilizada en la implementación de varios manejadores para el SCADA Guardián del ALBA, ha demostrado ser robusta y flexible.

2.5. Flujo actual de los procesos.

Las tareas de recolección de la información procedente del campo deben ser planificadas en función de los tipos de protocolos y redes de comunicación. La adquisición se realiza a través de los manejadores, ya sean por mecanismos de encuesta-respuesta o por mecanismos de escucha.

Los manejadores tienen la responsabilidad de agrupar las variables asociadas a los dispositivos en bloques siguiendo varios criterios, entre ellos el orden de las variables, el tamaño y uno de los más importantes; el periodo de muestreo. Los manejadores crean los bloques con el objetivo de poder recuperar o escribir todas las variables que lo conforman un solo pedido del Protocolo.

La planificación de las tareas de lectura se hace en función del tiempo de consulta de cada bloque de interrogación [9]. A continuación se describe la secuencia del proceso:

1. El SCADA solicita al manejador correspondiente la información de los valores, estampas de tiempo y calidad de las variables para un bloque determinado.
2. El manejador construye las tramas de solicitud para las variables de dicho bloque siguiendo las reglas del protocolo de comunicación. Envía las tramas de solicitud y espera respuesta por parte de los dispositivos en el campo.

3. El dispositivo procesa la solicitud y responde con los datos solicitados.
4. Una vez que el manejador recibe los valores, los transmite a las capas superiores para su posterior procesamiento.

Este es un proceso que se repite en función de la frecuencia de muestreo de las variables a consultar. En la siguiente figura se muestra graficamente este proceso.

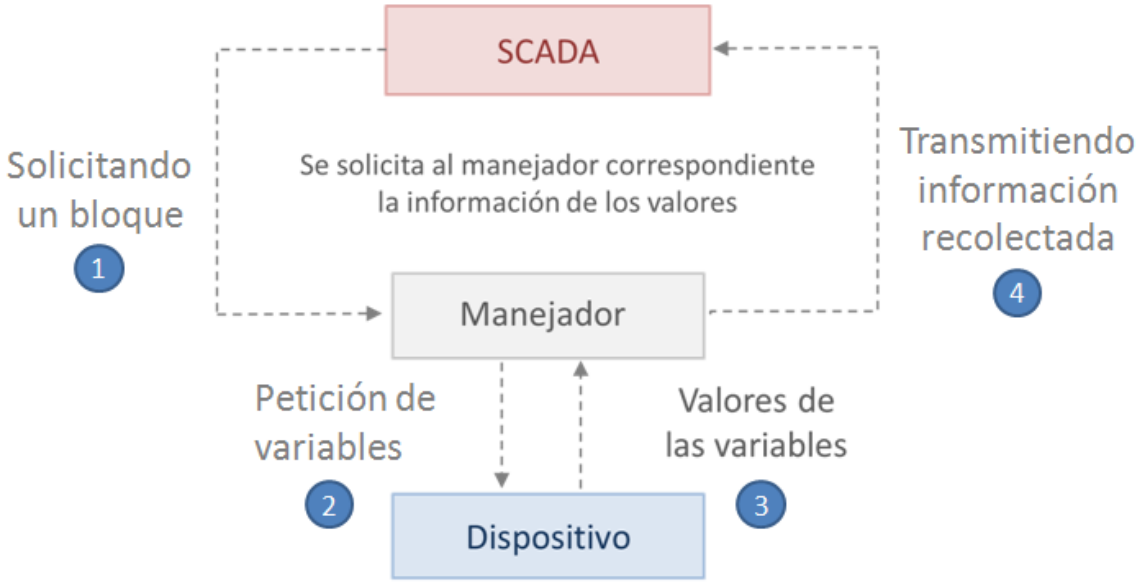

Figura 13 Proceso de recolección de datos

2.5.1. Solicitud de las señales

Como se muestra en la figura 15, una señal con nombre F.MOTOR.CALOR ocupa 13 bytes en el mensaje (un byte por caracter), siendo 249 el máximo para una trama de solicitud de datos. Evidentemente el número de señales a solicitar en un mismo mensaje es muy reducido. Por lo que es necesario construir gran cantidad mensajes y por tanto un gran número de peticiones para recuperar los valores de las variables configuradas en el SCADA.

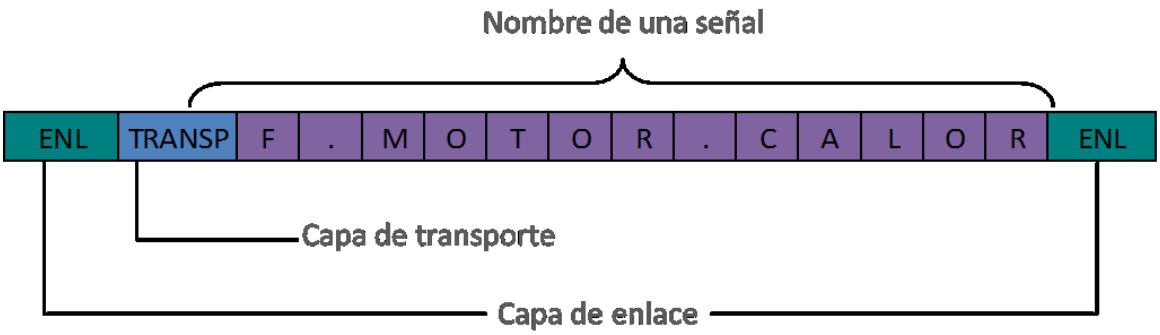

Figura 14 Mensaje local para la solicitud de una señal.

2.5.2. Comunicación con los dispositivos Bristol

La comunicación con los dispositivos Bristol tiene la peculiaridad de ser lenta, ya que cada solicitud de datos debe viajar por toda una jerarquía de dispositivos en forma de árbol para llegar a su destino, ver Figura 2. De la misma manera el dispositivo retorna la información solicitada al emisor.

Por las características de la comunicación con dichos dispositivos, el acceso al medio físico es de forma exclusiva. Esto significa que ningún bloque de variables puede acceder al medio físico, mientras sea usado por otro. Por lo que las peticiones se realizan de manera secuencial, trayendo consigo que si por alguna razón un bloque demora un tiempo mayor que el configurado, se retrasa el momento en que se deben solicitar los próximos bloques. Este retraso se va incrementando a medida que se ejecutan las siguientes peticiones. Por este motivo las variables no se actualizan en el tiempo establecido.

2.6. Especificación de requerimientos de software.

Los manejadores que se desarrollan para el Guardián del ALBA deben cumplir con una serie de requerimientos especificados en el documento: “Especificación de Requerimientos de Software del Módulo de Drivers”, en su versión 1.3. De los requerimientos que se especifican en el documento antes mencionado, el manejador para el protocolo BSAP debe incluir en su funcionamiento, específicamente los siguientes requerimientos:

2.6.1 Requerimientos funcionales.

1. **Lectura de variables del dispositivo** [10]. Se deben brindar funcionalidades para la lectura de variables, según especifica el protocolo BSAP, ya sea por el nombre o la dirección MSD.

2. **Escritura de variables del dispositivo** [10]. La capa de acceso a datos debe soportar funcionalidades de escritura de las variables, según especifica el protocolo BSAP
3. **Configuración de dispositivos y redes.** La capa de acceso a datos debe ofrecer funciones para obtener y modificar la configuración de sus parámetros y de los parámetros de los dispositivos y redes [10].
4. **Variables con información de diagnóstico.** Se deben brindar funcionalidades que permitan obtener información de diagnóstico con relación a las operaciones realizadas a cierto dispositivo, lo cual puede ayudar a determinar el estado del mismo y el comportamiento que ha presentado en el transcurso del tiempo [10].
5. **Mensajes de Error.** La capa de acceso a datos debe proporcionar una función que traduzca los códigos de error específicos que devuelven sus funciones a mensajes textuales [10].

2.6.2 Requerimientos no funcionales.

- **Usabilidad** [10].
- **Confiabilidad** [10].
- **Portabilidad.** La biblioteca debe poder ser ejecutada por los sistemas operativos más utilizados, así como en la mayoría de los compiladores de C++ [10].
- La capa debe de tener un 100% de disponibilidad [10]

2.7. Propuesta del sistema.

Dada situación problémica antes planteada se necesita implementar un manejador que cumpla con los requerimientos del sistema y que permita reducir la demora en el proceso de adquisición de datos. Para lograrlo se debe aprovechar al máximo el canal de comunicación, buscando la manera de encuestar la mayor cantidad de variables en un único pedido. Simplificando así el número de mensajes a transmitir entre los dispositivos y el SCADA.

Los datos asociados a las señales residen en un directorio (Master Signal Directory) que posee cada

dispositivo, estos datos pueden ser accedidos ya sea por el nombre de la señal o por la dirección MSD. Si se solicitan las señales por sus direcciones, que ocupan solamente 2 bytes en trama, se multiplica hasta 8 veces el número de variables a solicitar en una sola petición, reduciendo la cantidad de mensajes a transmitir. En la siguiente figura se representa de forma gráfica lo antes planteado.

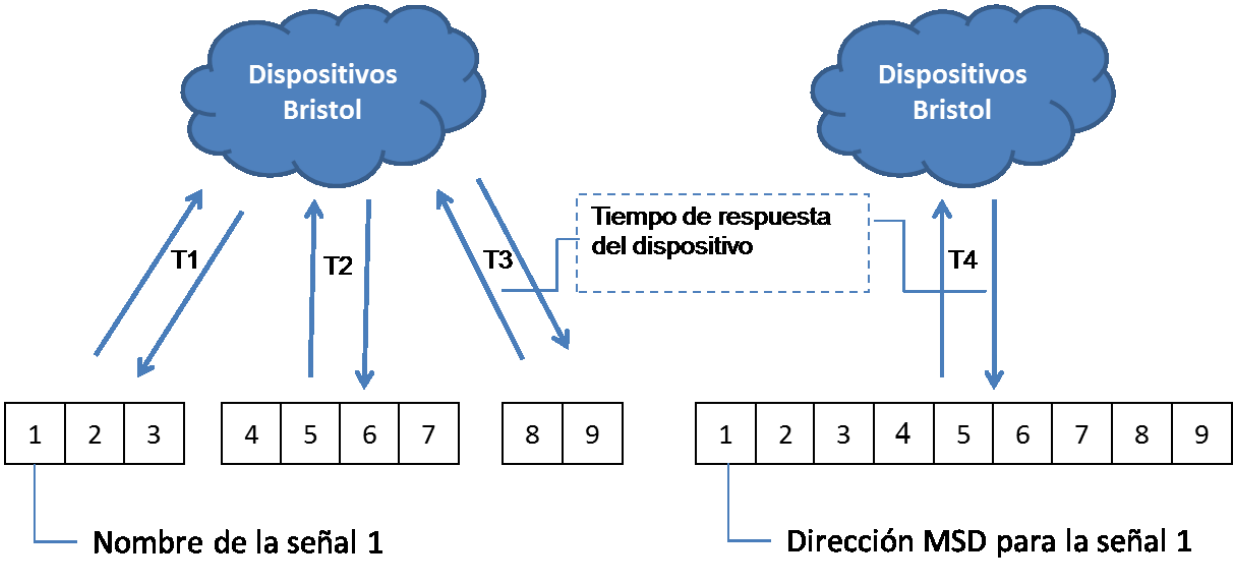

Figura 15 Solicitud de señales por sus nombres y por sus direcciones MSD

En la figura anterior se puede apreciar que para solicitar 9 señales por sus nombres, se necesitan 3 mensajes y el tiempo de actualización de las variables se puede calcular con la fórmula T1 + T2 + T3. Sin embargo con direcciones MSD se solicitan las 9 señales en un solo mensaje con un tiempo de actualización T4, donde T1 + T2 + T3 > T4.

2.7.1 Bloques y segmentos.

Como se explicó anteriormente un bloque es un grupo de variables ordenadas y agrupadas por criterios que define el manejador. Para optimizar el proceso de recolección, se ordenarán las variables que tengan un mismo periodo de encuesta por sus nombres, de manera tal que las señales con la misma base estén lo más cercanas posible, así como la extensión y el atributo. Se cree que de esta manera los dispositivos respondan con mayor prontitud. Por ejemplo, para un grupo de señales con los nombres A.D.7, B.A.1 y A.C.2 se ordenarían de la siguiente manera A.C.2, A.D.7 y B.A.1, para lograr esto se toma como criterio de ordenamiento el orden lexicográfico de los nombres.

Para construir los bloques y poder realizar las solicitudes con direcciones MSD el manejador necesita varios datos con los que no se cuenta inicialmente, como son la dirección y el tipo de la señal. Además la interfaz genérica no permite modificar los bloques una vez creados en el inicio del proceso. Por lo que se decide agrupar las variables sin un límite para el tamaño de los bloques, con el objetivo de que todas las variables de un dispositivo determinado con un mismo periodo de encuesta se encuentren en el mismo bloque. Y de esta manera el manejador puede gestionar internamente los bloques de forma dinámica.

Una vez conformados los bloques el SCADA necesita los valores para cada señal según su periodo de muestreo. Debido a que no se tuvo en cuenta el tamaño ni la cantidad de señales a la hora de conformar los bloques, es muy probable de que todas no puedan ser solicitadas en un mismo mensaje, por lo que el manejador divide los bloques en segmentos.

Se define como segmento a un subconjunto de un bloque como se muestra en la figura 17. Contiene la mayor cantidad de señales que se pueden encuestar en un mensaje BSAP. Los segmentos tienen la mejoría de ser dinámicos, posibilitando cambiar su tamaño y contenido. Se utiliza con el fin de solicitar las variables para un bloque cuando estas no pueden ser pedidas en un solo mensaje.

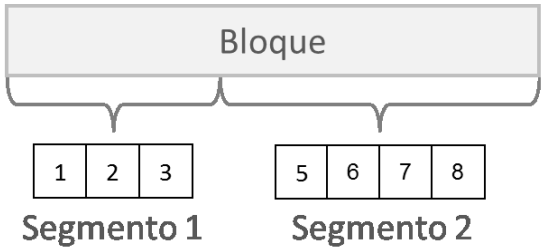

Figura 16 Dos segmentos para un bloque de variables.

2.7.2 Proceso de actualización.

Teniendo en cuenta que todas las variables de un dispositivo con un mismo periodo se encuentran en un bloque, este puede ser demasiado grande para recuperar todos sus valores en una solicitud. Por lo que el manejador debe gestionar internamente las solicitudes de los segmentos correspondientes.

Cuando se realiza la solicitud para los valores de un bloque determinado, este se divide en segmentos, los cuales se van a encuestar de manera secuencial. A medida que se solicita cada segmento se transmiten los datos recolectados a las capas superiores. Al terminar con el último segmento del bloque se devuelve

el control al SCADA para que realice la petición al próximo bloque en la cola de solicitudes.

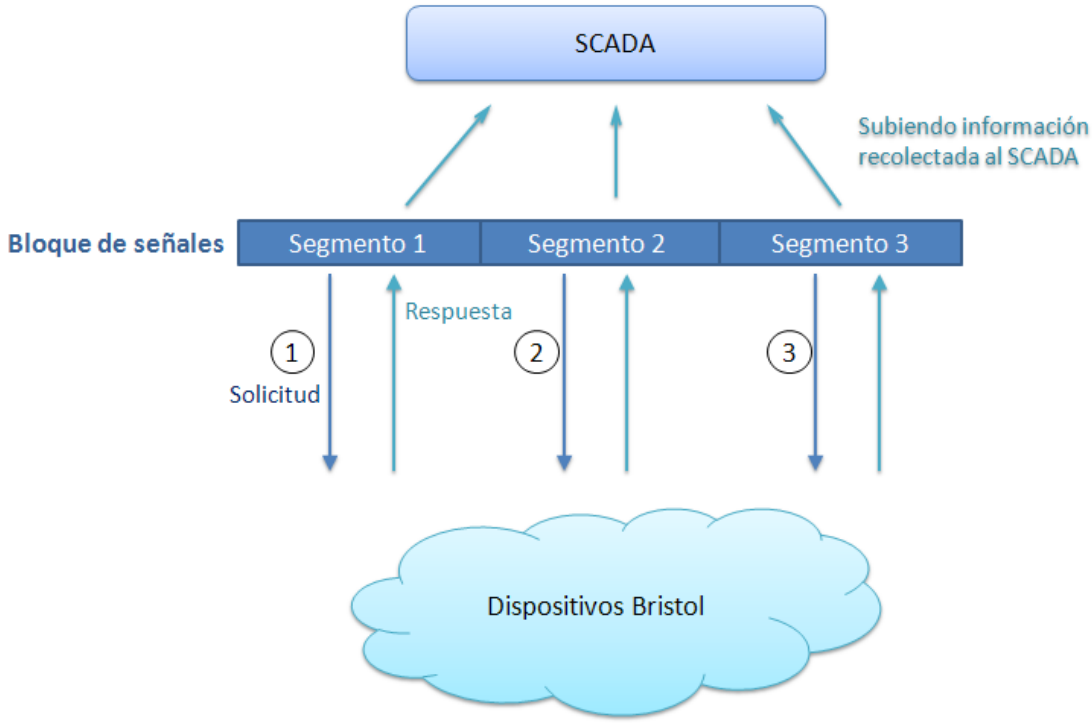

Figura 17 Solitudes para los segmentos de un bloque de señales

2.7.3 Solicitando los datos.

El manejador inicialmente solo conoce los nombres de las señales, por lo que para comenzar el proceso de actualización se necesita la dirección MSD para cada señal. Como se explicó en el epígrafe 1.4.3 el protocolo BSAP permite seleccionar los campos de las señales a solicitar, en este caso se requiere la dirección MSD y el tipo de señal, esta puede ser analógica o digital. Para esto se construyen los segmentos correspondientes a cada bloque y a partir de estos segmentos se ensamblan los mensajes con los nombres de las señales, solicitando los campos antes mencionados. Una vez conocidas las direcciones MSD de cada señal, el manejador no necesita solicitarlas en próximos mensajes.

Posteriormente comienza el ciclo de actualización construyendo los mensajes con las direcciones MSD, solicitando la mayor cantidad de variables que quepan en un mensaje. En cada solicitud se envía la versión de las direcciones MSD que conforman el mensaje, en caso de que no coincidan con las del dispositivo, este retorna un código de error avisando que las direcciones solicitadas no son válidas. Esto

suele suceder cuando la configuración del dispositivo es alterada. Para este caso el manejador vuelve a solicitar las direcciones MSD al dispositivo para continuar el ciclo de actualización.

2.7.4 Construyendo los mensajes.

Al construir los mensajes se debe tener en cuenta el tipo de señal. Ya que puede que no quepa toda la información solicitada, en el mensaje de respuesta del dispositivo. Por ejemplo si todas las señales solicitadas en un mensaje compuesto por direcciones MSD son analógicas, la cantidad de señales para un mensaje local sería un total de (246 bytes / 2 bytes = 123 señales) y la respuesta a esta solicitud tendría (123 señales * 4 bytes = 492 bytes). Es obvio que los valores de todas las señales no caben en un el mensaje de respuesta, por lo que habría que pedir nuevamente las señales que no recibieron sus valores. Para evitar casos como este a la hora de ensamblar las tramas se calcula el tamaño del mensaje de respuesta, conociendo que las señales analógicas ocupan 4 bytes y las señales digitales ocupan 1 byte.

2.7.5 Solicitudes pendientes.

Cuando se realiza una solicitud de datos para un grupo de variables puede darse el caso que el dispositivo no responda por algún motivo desconocido, puede ser por problemas de comunicación o rotura del equipo. Teniendo en cuesta lo antes planteado se establece un límite de solicitudes, cuando se alcanza este límite el manejador ignora al dispositivo y devuelve el control a las capas superiores.

2.8. Modelo de domino.

Este modelo tiene como objetivo lograr un entendimiento mínimo del contexto en que se empleará el manejador.

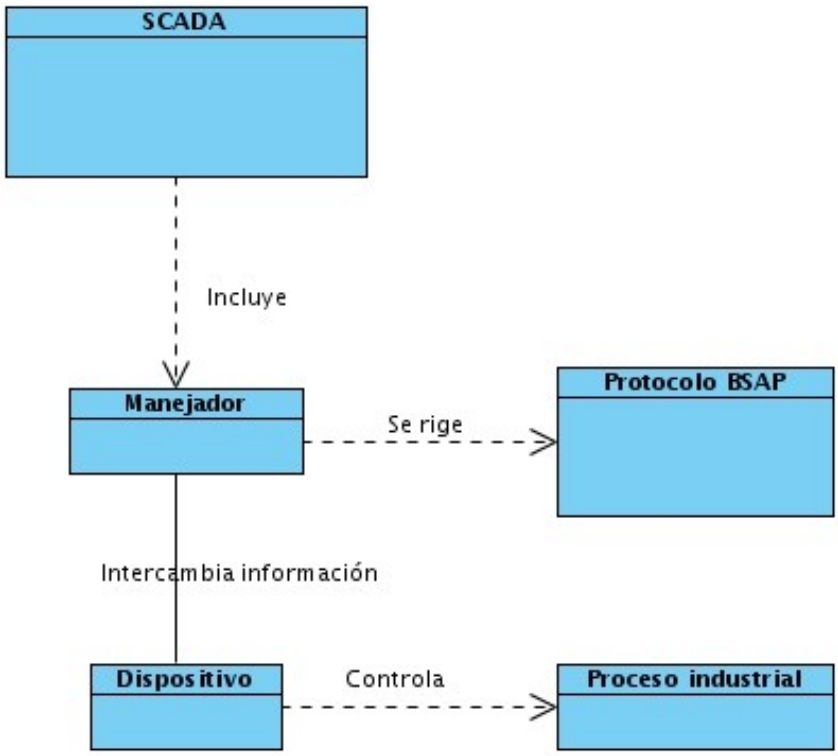

Figura 18 Modelo de dominio

2.8.1 Glosario de términos del dominio.

SCADA: Este concepto engloba a todos los componentes que conforman al Sistema de Adquisición y Supervisión de Datos.

Manejador: Módulo del SCADA encargado del intercambio de información con dispositivos que se comunican por medio de un protocolo de comunicación específico.

Protocolo de comunicación BSAP: Conjunto de reglas o leyes a tener en cuenta para la comunicación con los dispositivos Bristol.

Dispositivo: Equipo electrónico que permite la adquisición de datos.

Proceso Industrial: Proceso que se lleva a cabo en industrias, entre ellos extracción, el refinamiento y el transporte del combustible.

Capítulo3: Diseño, implementación y pruebas.

3.1. Arquitectura en capas.

Para el diseño del manejador se utilizó la arquitectura en tres capas, esta se explica de manera funcional en el epígrafe 2.1. Esta arquitectura permite llevar a cabo el desarrollo en varios niveles y, en caso de que sea necesario algún cambio, sólo se modifica el nivel requerido sin tener que revisar todo el código. El diseño queda de la siguiente manera:

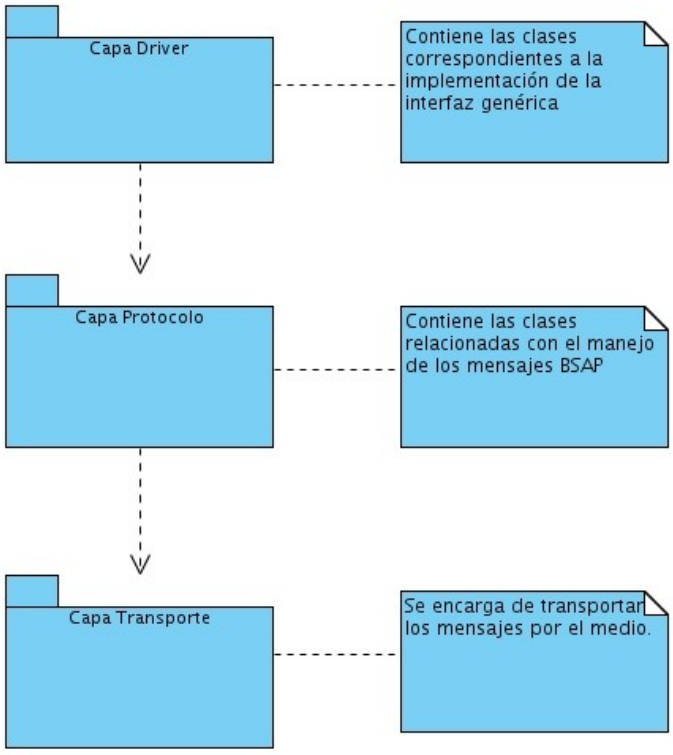

Figura 19 Arquitectura en tres capas del manejador.

En la capa de driver se encuentran aquellas clases que se relacionan directamente con el DriverCore. Ellas representan las características particulares del sistema propuesto. Posibilitan que la información recolectada cumpla con las restricciones impuestas por la Interfaz Genérica de los manejadores y por las particularidades del protocolo en cuestión.

La capa de protocolo se encarga de gestionar la comunicación con el dispositivo. En ella se sitúan las clases que interpretan el sentido y organización de los mensajes recibidos, además de aquellas que realizan las operaciones necesarias para solicitar y escribir datos en el dispositivo, entre otras operaciones.

En la capa de transporte se encuentran las clases que conforman la biblioteca TransportProvider. La misma se encarga de la transmisión de los datos por el puerto serie.

3.2. Diagramas de clases del manejador BSAP.

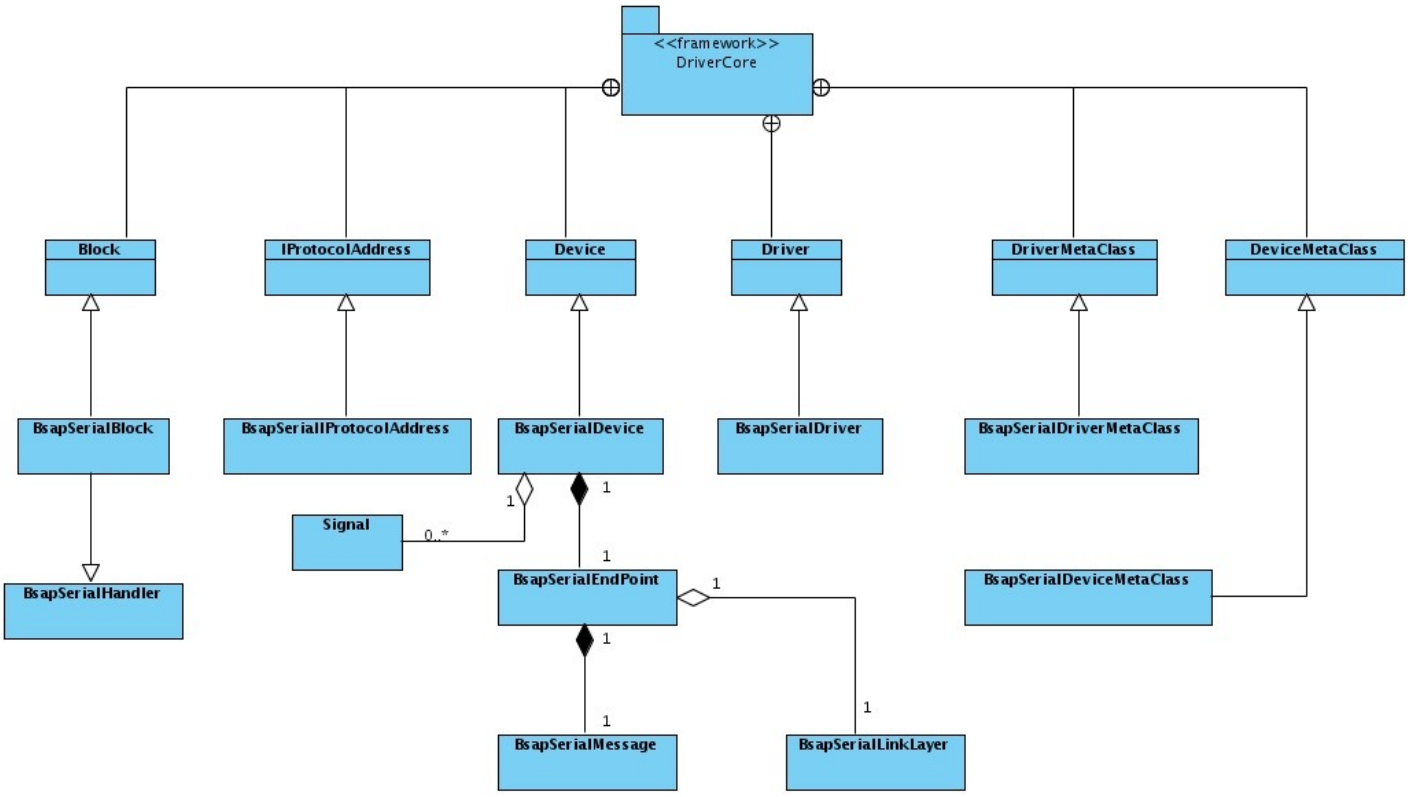

Figura 20 Diagrama de clases general.

3.2.1 Diagrama de clases de la capa protocolo.

Para un mejor entendimiento del diagrama de clases general, a continuación se expone el diagrama de clases que corresponde a la capa de protocolo de una forma más detallada.

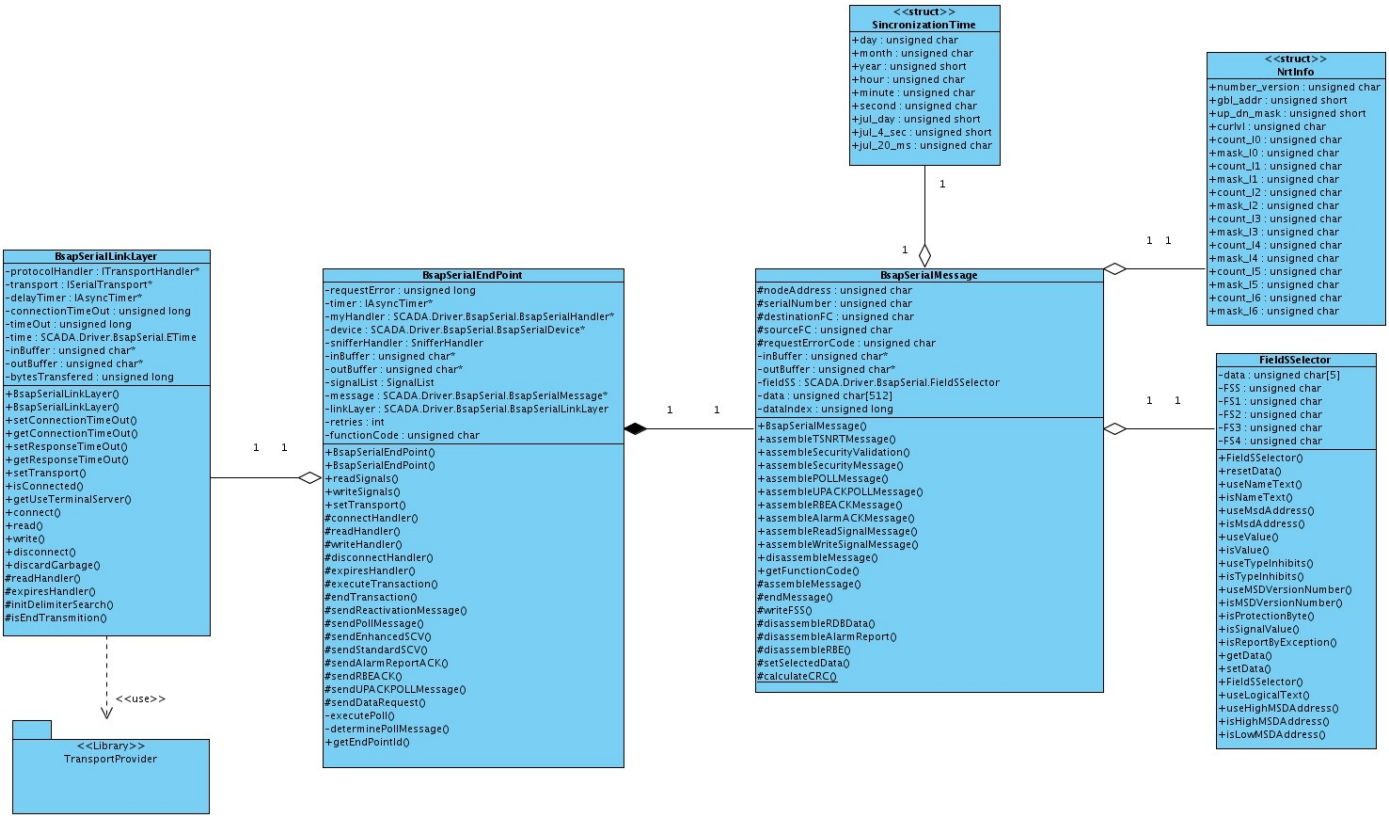

Figura 21 Diagrama de clases de la capa protocolo.

3.2.2 Diagrama de clases de la capa Driver.

A continuación el diagrama de clases de la capa de driver.

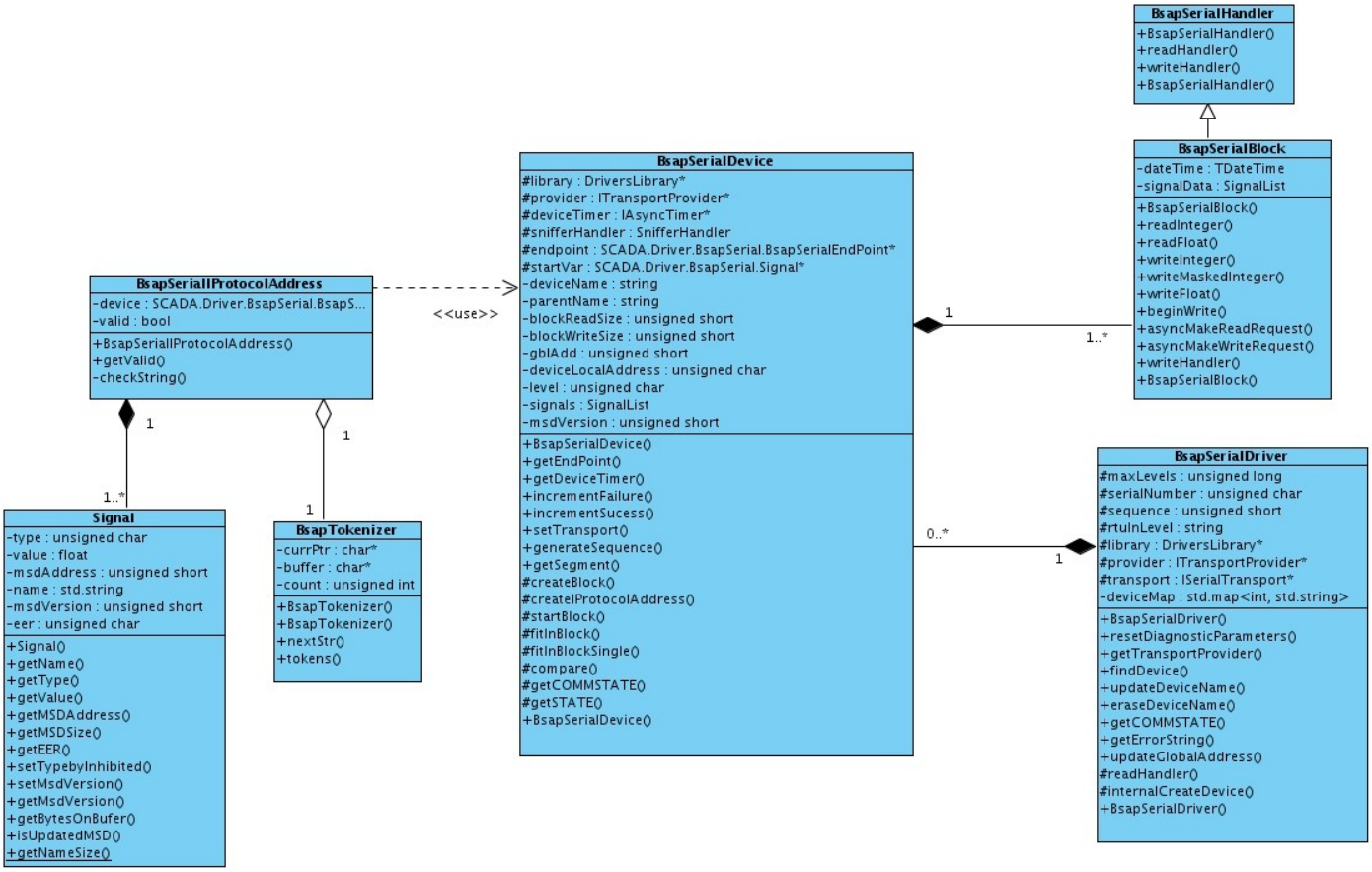

Figura 22 Diagrama de clases de la capa Driver

3.3. Descripción de las clases del diseño

A continuación se describen las principales clases que conforman el manejador, pues no es objetivo de este trabajo plasmar cada clase que lo componen.

3.3.1 Clase BsapSerialEndPoint

Esta clase representa las características comunes y comportamientos en la comunicación con los dispositivos Bristol, además de las funciones necesarias para leer y escribir variables en dichos dispositivos.

Nombre: BsapSerialEndPoint
Tipo de clase: Entidad

Atributo:	**Tipo:**
numberOfRetries	unsigned long
retriesInterval	unsigned long
requestError	unsigned long
protocoloState	ProtocolState
timer	IAsyncTimer*
snifferHandler	SnifferHandler
myHandler	BsapSerialHandler
timeStamp	long long
inBuffer	unsigned char *
outBuffer	unsigned char *
signalList	SignalList
message	BsapSerialMessage*
Linklayer	BsapSerialLinkLayer

Para cada responsabilidad	
Nombre:	BsapSerialEndPoint(BsapSerialDevice* device, SnifferHandler snifferHandler);
Descripción:	Constructor de la clase BsapEndPoint
Nombre:	setEndPointID (unsigned long newEndPointID)
Descripción:	Cambia el identificador del EndPoint.
Nombre:	getEndPointID ()
Descripción:	Devuelve el identificador del EndPoint.
Nombre:	setEnableSniffer (bool enable)
Descripción:	Establece el uso o no del sniffer. En dependencia del uso del sniffer o no se ejecutan los callback de la función sniffer.
Nombre:	connectHandler (unsigned long error)

Descripción:	Handler de conexión del transporte reimplementado.
Nombre:	readHandler (unsigned char* data, unsigned long bytesTransfered, unsigned long error)
Descripción:	Handler de lectura del transporte reimplementado.
Nombre:	void writeHandler (unsigned long bytesTransfered, unsigned long error)
Descripción:	Handler de escritura del transporte reimplementado.
Nombre:	disconnectHandler (unsigned long error)
Descripción:	Handler de desconexión del transporte reimplementado.
Nombre:	expiresHandler (unsigned long error)
Descripción:	Handler del timer asíncrono reimplementado.
Nombre:	executeTransaction ()
Descripción:	Ejecuta las transacciones del endpoint.
Nombre:	endTransaction ()
Descripción:	Termina la transacción
Nombre:	executeTransaction ()
Descripción:	Ejecuta una transacción enviando el mensaje que se encuentra en el buffer.
Nombre:	sendReactivationMessage()
Descripción:	Envía al dispositivo un mensaje de reactivación de POLL.
Nombre:	sendPollMessage()
Descripción:	Envía un mensaje de POLL al dispositivo.
Nombre:	sendEnhancedSCV ()
Descripción:	Envía un mensaje para obtener la llave para encriptar la contraseña necesaria para loguearse.
Nombre:	sendStandardSCV ()
Descripción:	Envía un mensaje para validar el código de seguridad.
Nombre:	sendAlarmReportACK ()

Descripción:	Envía un ACK a previo reporte de alarmas recibido.
Nombre:	sendUPACKPOLLMessage()
Descripción:	Envía un mensaje UP-ACK con POLL al dispositivo.
Nombre:	sendDataRequest()
Descripción:	Envía un mensaje de solicitud de datos.
Nombre:	determinePollMessage()
Descripción:	Cuando la función es invocada ya se ha determinado e interpretado el tipo de mensaje recibido y se depositan los datos en las variables correspondientes.

Tabla 1 : Descripción de la clase BsapSerialEndPoint

3.3.2 Clase BsapSerialMessage

Esta clase es la encargada de formar e interpretar los mensajes del protocolo utilizados por la clase BsapSerialEndPoint.

Nombre: BsapSerialMessage	
Tipo de clase: Entidad	
Atributo:	Tipo:
destinationFC	unsigned char
sourceFC	unsigned char
requestErrorCode	unsigned char
inBuffer	unsigned char*
outBuffer	unsigned char *
fieldSS	FieldSSelector
Para cada responsabilidad:	
Nombre:	BsapSerialMessage(unsigned char *rBuffer, unsigned char *wBuffer);

Descripción:	Las instancias de la clase se crean a partir del buffer de lectura y escritura utilizados por el EndPoint.
Nombre:	assembleTSNRTMessage(const unsigned char nodeAddress, const unsigned char serial, const unsigned short sequence);
Descripción:	Construye un mensaje de sincronización de tiempo, para la reactivación de los nodos. Devuelve la cantidad de bytes que conforman el mensaje.
Nombre:	assembleSecurityValidation(const unsigned char nodeAddress, const unsigned char serial, const unsigned short sequence);
Descripción:	Construye un mensaje para la inicialización del registro de una nueva sesión en el dispositivo. Devuelve la cantidad de bytes que conforman el mensaje.
Nombre:	unsigned long assembleSecurityMessage(
Descripción:	Construye un mensaje para el registro de una sesión en el dispositivo. Devuelve la cantidad de bytes que conforman el mensaje.
Nombre:	assemblePOLLMessage(const unsigned char serial)
Descripción:	Construye un mensaje de tipo poll, utilizado para determinar si el dispositivo está activo y solicitar datos. Devuelve la cantidad de bytes que conforman el mensaje.
Nombre:	assembleUPACKPOLLMessage(const unsigned char serialNumber, const unsigned char ackSerialNumber)
Descripción:	Construye un mensaje UP-ACK, utilizado para informar al dispositivo la recepción de mensajes de datos de forma satisfactoria. Devuelve la cantidad de bytes que conforman el mensaje.
Nombre:	assembleReadSignalMessage(const unsigned char serial, const unsigned short sequence, const unsigned char functionCode, const unsigned char msdVersion, SignalList)

Descripción:	Construye un mensaje de acceso a la base de datos del dispositivo para la lectura de datos. Devuelve la cantidad de bytes que conforman el mensaje.
Nombre:	assembleWriteSignalMessage(const unsigned char serial,
Descripción:	Construye un mensaje de acceso a la base de datos del dispositivo para la escritura de datos. Devuelve la cantidad de bytes que conforman el mensaje.
Nombre:	disassembleMessage (const unsigned long size, SignalList)
Descripción:	Esta función interpreta los mensajes recibidos del dispositivo, el cual se encuentra en el buffer de entrada
Nombre:	getFieldSSelector()
Descripción:	Devuelve la instancia utilitaria que manipula los campos selectos por variables
Nombre:	disassembleRDBData(SignalList signalList)
Descripción:	Interpreta la parte de datos de los mensajes recibidos de parte del dispositivo. Retorna un código de error según la interpretación del mensaje recibido, valor de 0 si el mensaje se interpretó de forma correcta.
Nombre:	setSelectedData(Signal * signal)
Descripción:	Establece a cada señal su valor correspondiente en dependencia de la información que se solicitó, por defecto siempre se solicita el tipo de dato y el valor de las señales.
Nombre:	calculateCRC(const unsigned char* buffer, const unsigned long size)
Descripción:	Calcula la suma de chequeo polinomial CRC-CCITT. Se retorna una palabra con la suma de chequeo calculada.

Tabla 2 Descripción de la clase BsapSerialMessage

3.3.3 Clase BsapSerialBlock

Esta clase puede ser interpretada como un buffer de datos, desde el cual se pueden obtener valores. Hereda de las clases Block del framework DriverCore y BsapSerialHandler.

Nombre: BsapSerialBlock	
Tipo de clase: Entidad	
Atributo:	Tipo:
dateTime	TDateTime
signalData	SignalList
Para cada responsabilidad:	
Nombre:	BsapSerialBlock(Device* device, unsigned long blockStart, unsigned long blockEnd, bool forRead, VarLinkVector* container)
Descripción:	Los bloques son creados con los índices inferiores y superiores de las variables que lo integran.
Nombre:	readInteger(IProtocolAddress* address, DeviceVarType varType, TDateTime* deviceTimeStamp, unsigned short* quality)
Descripción:	Este método permite realizar la lectura de un valor entero desde el bloque o contenedor de datos a partir de un IProtocolAddress y un tipo de dato específico. Devuelve un valor entero de 64 bits que expresa el valor en el dispositivo.
Nombre:	readFloat(IProtocolAddress* address, DeviceVarType varType, TDateTime* deviceTimeStamp, unsigned short* quality)
Descripción:	Este método permite realizar la lectura de un valor flotante desde el bloque o contenedor de datos a de un IProtocolAddress y un tipo de dato específico. Devuelve un valor flotante de 64 bits relacionado con la dirección pasada por parámetros.

Nombre:	writeInteger(IProtocolAddress* address, DeviceVarType varType, long long value)
Descripción:	Permite escribir un valor entero en el bloque de variables, relacionado con la dirección BSAP pasada por parámetros.
Nombre:	writeMaskedInteger(IProtocolAddress* address, DeviceVarType varType, long long value, long long maskParam)
Descripción:	Este método permite modificar bits específicos en el bloque a partir de un IProtocolAddress y de un tipo en el dispositivo. El parámetro mask determina cuales bits serán modificados.
Nombre:	writeFloat(IProtocolAddress* address, DeviceVarType varType, double value)
Descripción:	Este método permite escribir un valor flotante en el bloque a partir de un IProtocolAddress y de un tipo en el dispositivo.
Nombre:	beginWrite()
Descripción:	Realiza las operaciones necesarias para inicializar la escritura del Bloque.
Nombre:	asyncMakeReadRequest()
Descripción:	Realiza una lectura asíncrona al dispositivo. Es aquí donde se invoca al método de lectura del BsapProtocolEndPoint.
Nombre:	asyncMakeWriteRequest()
Descripción:	Realiza la escritura al dispositivo BSAP en cuestión.
Nombre:	getSignals()

Descripción:	Devuelve las señales simples del bloque.
Nombre:	void writeHandler(unsigned long error)
Descripción:	Es invocado en forma de callback, cuando finaliza la escritura de las variables.

Tabla 3 Descripción de la clase BsapSerialBlock

3.3.4 Clase BsapSerialDevice

Hereda de SCADA::Driver::Device re-implementando las funcionalidades para la creación y organización de las variables en los bloques. Además implementa las funcionalidades estáticas para establecer y obtener las propiedades configurables y de diagnósticos del dispositivo.

Nombre: BsapSerialBlock	
Tipo de clase: Entidad	
Atributo:	Tipo:
library	DriversLibrary*
provider	ITransportProvider*
deviceTimer	IAsyncTimer*
snifferHandler	SnifferHandler
endpoint	BsapSerialEndPoint*
deviceName	String
blockReadSize	unsigned short
blockWriteSize	unsigned short
gblAdd	unsigned short

deviceLocalAddress	unsigned char
level	unsigned char
signals	SignalList
msdVersion	unsigned short
Para cada responsabilidad:	
Nombre:	BsapSerialDevice(unsigned long devScadaId, char* transferBuffer, unsigned long bufferSize, ITransportProvider* provider, SnifferHandler nsnifferHandler)
Descripción:	Recibe como parámetro un identificador único para el SCADA y un buffer de transferencia que será utilizado para el envío de los datos que se lean desde el dispositivo hacia el recolector, apuntador del provider utilizado por el driver y apuntador a la función sniffer.
Descripción:	Función estática utilizada por Bsap::BsapSerialDeviceMetaClass para obtener el valor de la propiedad relacionada con la dirección local del dispositivo. Devuelve la dirección local del dispositivo.
Nombre:	getLevel(BaseClass* base)
Descripción:	Función estática utilizada por Bsap::BsapSerialDeviceMetaClass para obtener el valor de la propiedad relacionada con el nivel del dispositivo.
Nombre:	setPollPeriod(BaseClass* base, long long value)
Descripción:	Función estática utilizada por Bsap::BsapSerialDeviceMetaClass para el establecimiento de la propiedad relacionada el periodo de los mensajes de poll.
Nombre:	getPollPeriod(BaseClass* base)

Descripción:	Función estática utilizada por Bsap::BsapSerialDeviceMetaClass para obtener el valor de la propiedad relacionada con el periodo de los mensajes de poll.
Nombre:	getEndPoint()
Descripción:	Devuelve el EndPoint utilizado por la clase.
Nombre:	enableSniffer(bool enable)
Descripción:	Habilita o deshabilita el sniffer.
Nombre:	getDeviceTimer()
Descripción:	Devuelve la instancia del timer asíncrono creada.
Nombre:	void setTransport(ISerialTransport* ntransport)
Descripción:	Establece el transporte usado por el driver al dispositivo en cuestión.
Nombre:	void updateAddress()
Descripción:	Actualiza los valores de la dirección local y dirección global del EndPoint en dependencia de los parámetros de configuración de la red de dispositivos.
Nombre:	getSegment(bool &endDevice)
Descripción:	Determina las próximas señales a leerse en dependencia del nivel y la función de código.
Nombre:	getMsdVersion()
Descripción:	Devuelve la versión MSD de las señales en el dispositivo.

Tabla 4 Descripción de la clase BsapSerialDevice

3.3.5 Clase BsapSerialDriver

Hereda de SCADA::Driver::Driver reimplementando las funcionalidades para la creación de dispositivos. Posee funcionalidades para configurar y obtener los valores de las propiedades del transporte, así como propiedades específicas del driver.

Nombre: BsapSerialDriver	
Tipo de clase: Entidad	
Atributo:	Tipo:
library	DriversLibrary
provider	ITransportProvider
transport	ISerialTransport
deviceMap	std::map<BsapSerialDevice*, std::string>
Para cada responsabilidad:	
Nombre:	resetDiagnosticParameters()
Descripción:	Devuelve el provider que utiliza el driver.
Nombre:	getTransportProvider(BaseClass* base, long long newPackageDensity)
Descripción:	Funcionalidad estática para el establecimiento de la propiedad de densidad de paquete.
Nombre:	setDeviceName(BaseClass* base, string newDeviceName)
Descripción:	Función estática utilizada por BsapSerialDriverMetaClass para obtener el valor de la propiedad relacionada con el nombre del dispositivo serial (ejemplo en Debian GNU/Linux /dev/ttyS0, en Windows COM1). Esta función presenta el prototipo funcional correspondiente al puntero de función SCADA::Driver::StrGetter
Nombre:	getDeviceName(BaseClass* base)

Descripción:	Función estática utilizada por BsapSerialDriverMetaClass para el establecimiento de la propiedad relacionada con la velocidad en baudios del dispositivo serial. Esta función presenta el prototipo funcional correspondiente al puntero de función SCADA::Driver::IntSetter
Nombre:	getTerminalServerPort(BaseClass* base)
Descripción:	Función estática utilizada por Bsap::BsapSerialDriverMetaClass para el establecimiento de la propiedad relacionada con el nivel máximo soportado. Esta función presenta el prototipo funcional correspondiente al puntero de función SCADA::Driver::IntSetter
Nombre:	setMaxLevels(BaseClass* base, long long nmaxLevel)
Descripción:	Función estática utilizada por Bsap::BsapSerialDriverMetaClass para obtener el valor de la propiedad relacionada con el nivel máximo soportado. Esta función presenta el prototipo funcional correspondiente al puntero de función SCADA::Driver::IntGetter
Nombre:	getMaxLevels(BaseClass* base)
Descripción:	Función estática utilizada por Bsap::BsapSerialDriverMetaClass para el establecimiento de la propiedad relacionada con el máximo de rtu por niveles. Esta función presenta el prototipo funcional correspondiente al puntero de función SCADA::Driver::IntSetter
Nombre:	setRtuInLevel(BaseClass* base, string nrtuLevel)
Descripción:	Función estática utilizada por Bsap::BsapSerialDriverMetaClass para obtener el valor de la propiedad relacionada con el nivel máximo soportado. Esta función presenta el prototipo funcional correspondiente al puntero de función SCADA::Driver::IntGetter
Nombre:	getRtuInLevel(BaseClass* base)
Descripción:	Encuentra la dirección local del dispositivo padre de primer nivel. En caso de que el nombre del dispositivo pasado por parámetro no sea de primer nivel, se busca la dirección local del padre que se encuentre en nivel 1.
Nombre:	findRoot(BaseClass* device)
Descripción:	Obtiene la dirección global de un dispositivo padre pasado por parámetro.

Nombre:	findParentGlobalAddress(string nparentName)
Descripción:	Encuentra un device dado el nombre del mismo.
Nombre:	findDevice(const std::string & name)
Descripción:	Actualiza el nombre de un dispositivo en los mapas de asociacion del manejador.
Nombre:	updateDeviceName(BsapSerialDevice * device)
Descripción:	Elimina el nombre de un dispositivo de los mapas con la informacion asociada.
Nombre:	eraseDeviceName(BsapSerialDevice * device)
Descripción:	Devuelve el estado de comunicacion asociado con el manejador.
Nombre:	getCOMMSTATE(TDateTime* dateTime)
Descripción:	El metodo getErrorString permite obtener informacion textual comprensible a partir de un código de error del manejador. Si el valor de errorCode no se corresponde con uno de los valores admisibles se devuelve errInvalidErrorCode.
Nombre:	getErrorString(unsigned long errorCode, const char** ppErrorDescription)
Descripción:	Actualiza la dirección global pasada por parametros en dependencia de la dirección local pasada y el nivel del dispositivo en cuestión. Para que la función tenga efecto se deben pasar cada una de las direcciones de los dispositivos de los niveles superiores del dispositivo en cuestión.
Nombre:	updateGlobalAddress(unsigned short & globalAddress, const unsigned char address, const int & level)
Descripción:	Incrementa el valor del número serie de los mensajes en el poll.

Nombre:	generateSerialNumber()
Descripción:	Incrementa el valor del número de secuencia de los mensajes de datos en el poll.
Nombre:	generateSequence()
Descripción:	Devuelve el número serial generado.
Nombre:	getSerialNumber()
Descripción:	Devuelve el número de secuencia generado.
Nombre:	getSequence()
Descripción:	Actualiza las señales de cada dispositivo. Obtiene las próximas señales a actualizar, y le ordena a los endpoints correspodientes a cada dispositivo a recolectar los datos necesarios.

Tabla 5 Descripción de la clase BsapSerialDriver

3.4. Diagrama de Despliegue.

El diagrama de despliegue muestra la disposición física de los distintos nodos que componen el sistema y el reparto de los componentes sobre dichos nodos. Los nodos no son más que elementos físicos que existen en tiempo de ejecución y representan un recurso computacional que generalmente tienen algo de memoria y capacidad de procesamiento. El diagrama de despliegue modela la topología de hardware sobre la cual se ejecutará el sistema.

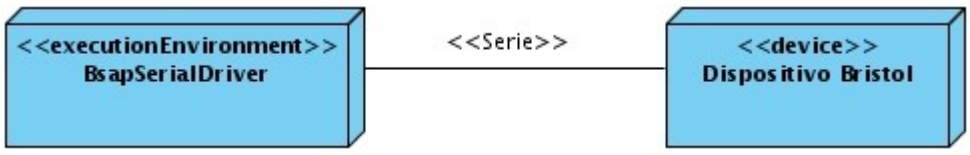

Figura 23 Diagrama de Despliegue del Manejador BsapSerial

El nodo BsapSerialDriver encapsula las funcionalidades del manejador el cual a través de una comunicación serial obtiene los datos provenientes de los dispositivos Bristol.

3.5. Diagrama de Componentes.

El siguiente diagrama de componentes muestra las dependencias lógicas entre los componentes del software.

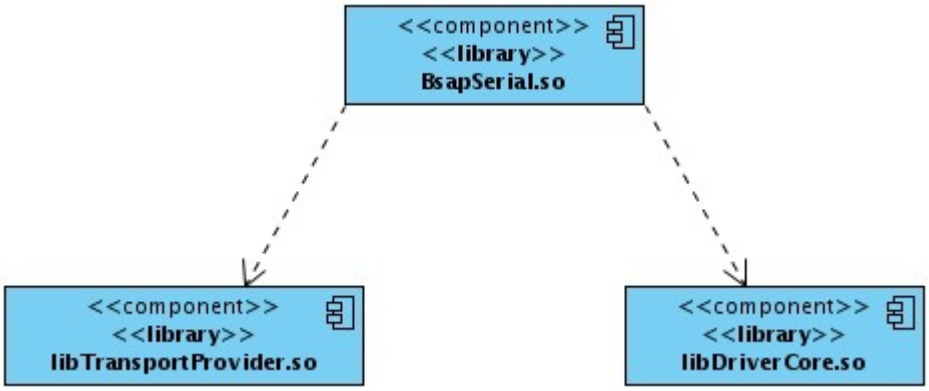

Figura 24 Diagrama de componentes del manejador BsapSerial

El componente *BsapSerial.so* es una biblioteca que encapsula las clases asociadas a las tres capas definidas en la arquitectura del manejador. Gestiona la comunicación con los dispositivos Bristol siguiendo las reglas del protocolo BSAP. Hace uso del componente *libTransportProvider.so* que propicia una interfaz bien definida para la comunicación con el medio físico y *libDriverCore.so* que provee la implementación de funcionalidades comunes en los manejadores.

3.6. Pruebas de software

Las pruebas de software son procesos que permiten verificar la calidad de un producto. Se utilizan para comprobar que el software cumple con los requerimientos planteados, además de identificar errores de implementación, usabilidad o calidad de los programas computacionales. Al manejador desarrollado se le han realizado varias pruebas para evaluar su comportamiento, las mismas siguen el paradigma de las pruebas de Tipo Caja Negra.

Las pruebas realizadas fueron planeadas con el objetivo de comparar la solución propuesta con el manejador implementado anteriormente, en cuanto a los tiempos de actualización de las variables. Estas fueron realizadas en una PC con procesador Core 2 Duo 2.20 Ghz y memoria de 1Gb de RAM. Los resultados de las pruebas pueden variar, en dependencia del contexto en que se realicen las mismas.

Como no se cuenta con un dispositivo real, se desarrolló un programa que simula el comportamiento de los dispositivos Bristol Babcock. Este simulador se ejecuta de forma local en la computadora de prueba siendo despreciado el tiempo en que viajan las solicitudes BSAP hacia los dispositivos. El simulador para el protocolo BSAP responde correctamente a las tramas de solicitud con valores generados aleatoriamente.

Caso #1: Lectura de señales en un dispositivo.

En el caso inicial se midieron los tiempos de recolección para cada uno de los manejadores con una configuración de 100, 300, 500 y 1000 señales en un solo dispositivo, cada una con un periodo de actualización de 1 segundo. Las pruebas realizadas arrojaron como resultado una mejora considerable en los tiempos de recolección logrando reducir la demora en el muestreo de las variables hasta un 80%. Siendo mayor la diferencia cuando se configuran un gran número de variables. Los resultados se muestran gráficamente en la siguiente figura:

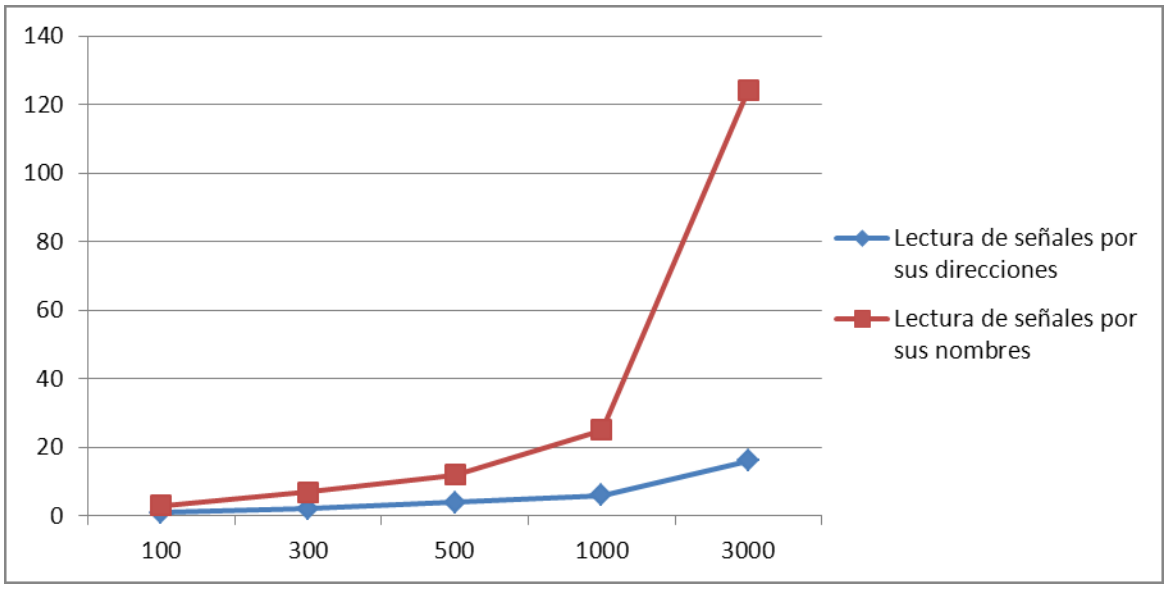

Figura 25 Comparación de los manejadores con un dispositivo.

El eje horizontal de la gráfica representa el número de variables configuradas en el dispositivo y el eje vertical el tiempo en segundos que demora una variable en actualizarse.

Caso #2: Lectura de señales con 2 dispositivos.

En este caso de prueba se configuraron dos dispositivos cada uno con el mismo el mismo número variables, sumando 100, 300, 500 y 1000. Cada una de estas variables tiene un periodo de 1 segundo para su muestreo.

Luego de realizar las pruebas se obtuvieron mejores resultados que el caso anterior, logrando reducir la demora en el proceso de actualización en un 88 %. En la siguiente gráfica de tendencia se muestran los resultados.

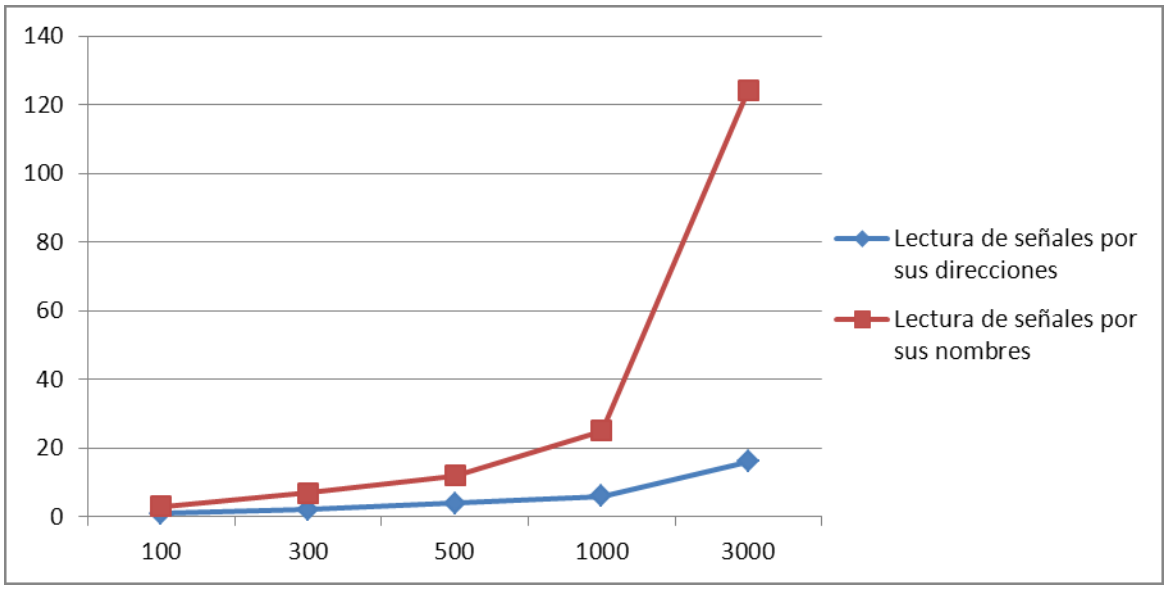

Figura 26 Comparación de los manejadores con dos dispositivos.

3.6.1. Recolector gráfico

Para realizar las pruebas fue necesario el uso del Recolector Gráfico, el cual es un software que permite cargar el manejador como una biblioteca y brindar una interfaz visual para la configuración del manejador y los dispositivos asociados a las variables. Así como la vista de los valores que se recolectan y las tramas

que se envían y reciben por parte del dispositivo. A continuación se muestran algunas capturas en el proceso de adquisición de datos.

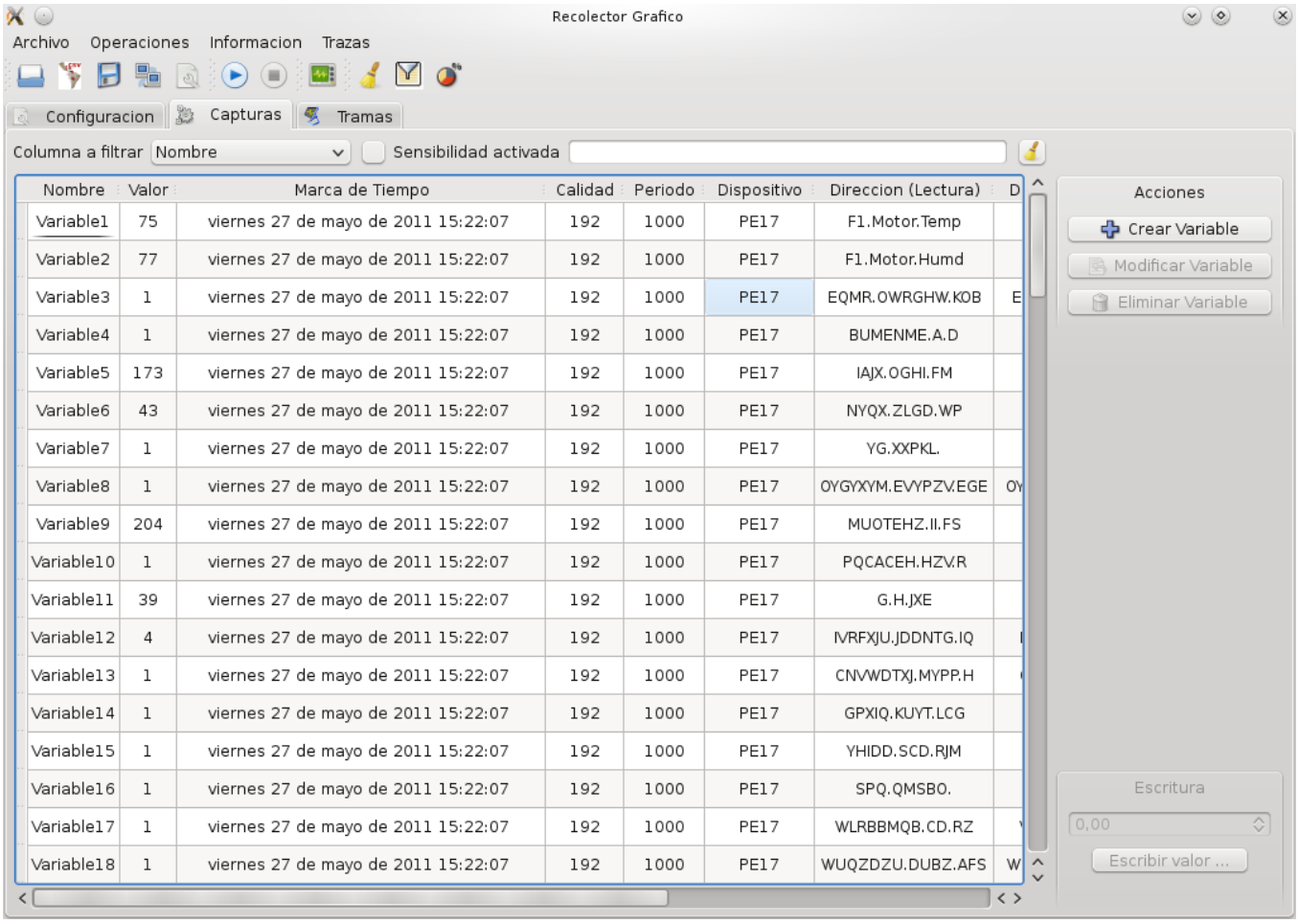

Figura 27 Capturas de variable con el recolector

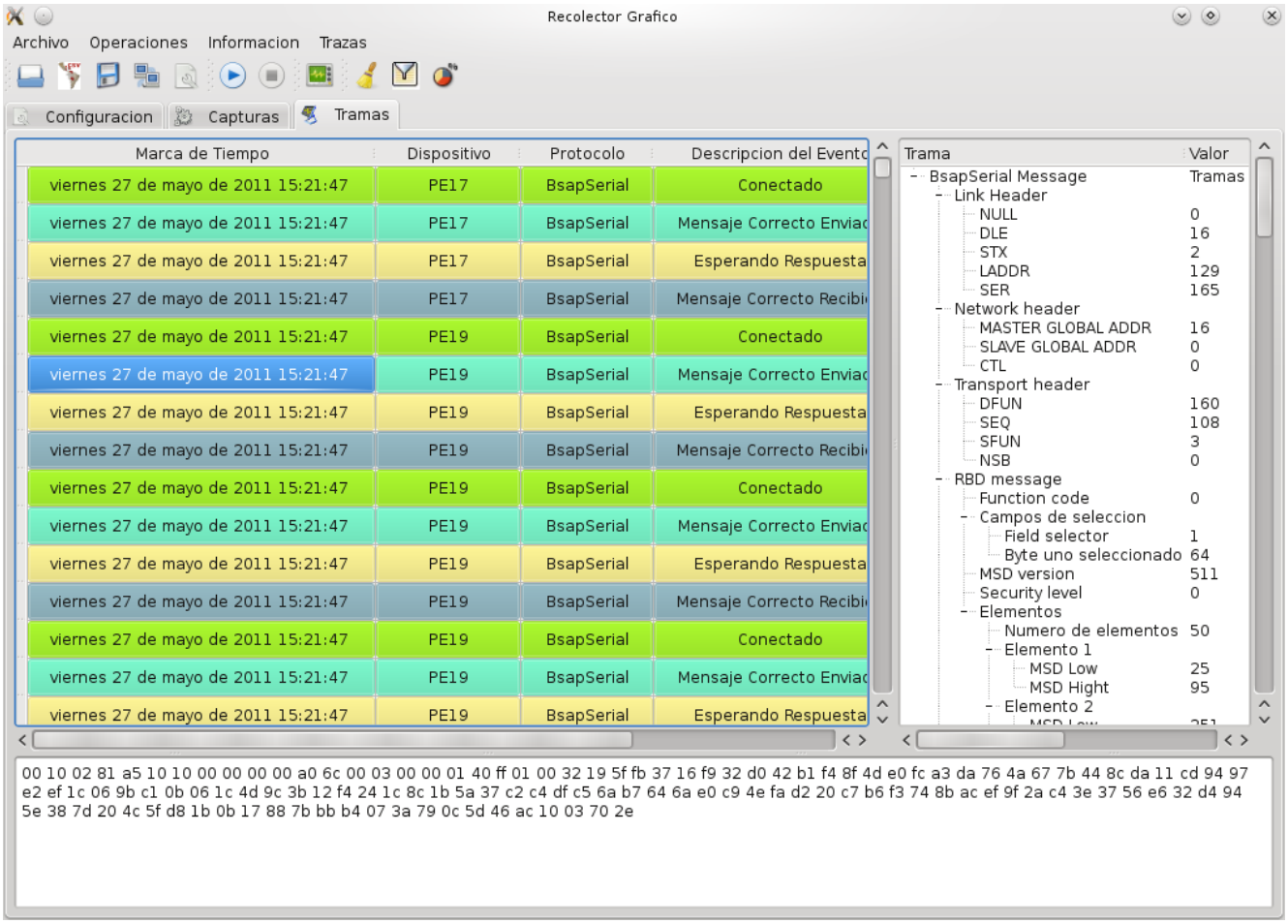

Figura 28 Vista de las tramas que se envían y reciben

CONCLUSIONES GENERALES

Se obtuvo un módulo de clases completamente funcional, que cumple con los requisitos planteados. Logrando como resultado:

- La solución implementada logró reducir ampliamente los tiempos de actualización de las variables, cumpliendo con los objetivos definidos en el inicio del trabajo.
- La capa de acceso a datos desarrollada se adapta a diferentes entornos de acción y ejecución. Además del proyecto Guardián del Alba la solución implementada podría ser usada por cualquier otro sistema que necesite comunicarse con dispositivos Bristol
- La utilización de la biblioteca de transporte TransportProvider permitió acceder a datos dispuestos en dispositivos Bristol de forma fácil y eficiente.

RECOMENDACIONES

Debido a que en este trabajo no se desarrolla todas las funcionalidades del protocolo BSAP se recomienda como trabajo futuro la implementación de las mismas. Estas pueden mejorar aún más el proceso de adquisición en los sistemas de supervisión actuales.

Aspectos que se recomienda para trabajos futuros:

- Agregar funcionalidades para la captura de reportes por excepción.
- Agregar funcionalidades para la captura de reportes de alarmas generadas por el dispositivo.

BIBLIOGRAFÍA

1. Márquez, J.E.B., *Transmisión de datos, Tercera edición*. 2005.
2. Pozo, A.C., *Módulos de adquisición y análisis para la interacción con dispositivos de campo en un SCADA*. 2009, Universidad de las Ciencias Informáticas: La Habana.
3. PDVSA, *Sistema de adquisición supervisión y control para los pueblos del ALBA.* 2009.
4. Babcock, B., *Network 3000 Communications Application Programmer's Reference*. 2005, Watertown.
5. Naranjo, V.H., *DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL Y MONITOREO DE VALVULAS DE BLOQUEO PARA EL PASO DE COMBUSTIBLE*. 2010: Quito
6. Teletrol. *Metso Automation, Sistema avanzado de supervisión, control y gerencia de información OASyS.* 2011.
7. Foundation, E. *The Eclipse Foundation open source community website*. 2011; Available from: http://www.eclipse.org.
8. Codorníu, R.T., A.C. Pozo, and L.E.G. Hernández, *Interfaz Genérica para manejadores de dispositivos en el proyecto SCADA PDVSA*. 2008.
9. PDVSA, *Recolección y Manejadores en el Guardián del ALBA*. 2008.
10. ALBA, P.G.d., *Especificación de Requerimientos de Software del Módulo de Drivers*. 2007.
11. Sons, J.W., *Pattern Oriented Software Architecture*. 1996.
13. Automation, M., *OASyS IS THE ONE SUPERVISORY CONTROL AND REALTIME INFORMATION MANAGEMENT SYSTEM THAT PROVIDES YOU THE FLEXIBILITY AND ADAPTABILITY FOR YOUR CONSTANTLY CHANGING ENTERPRISE NEEDS.* 2001.
14. C.A, S., *Curso de PLC OMRON.* . 2001.
15. Bailey, D. and E. Wright, *Practical SCADA for Industry. Mumbai, India : Vivek Mehra*. 2003: Vivek Mehra.
16. Parr, M.E.A., *Programmable Controllers, An engineer's guide*. 2003.
17. Casanova, J., *Usabilidad y arquitectura del software*. 2004.
18. Instruments, N. *Comunicación Serial: Conceptos Generales*. 2004 06-06-2006; Available from: http://digital.ni.com/public.nsf/allkb/039001258CEF8FB686256E0F005888D1.
19. PDVSA, *Documento de Visión del Proyecto Guardián del ALBA.* . 2007.
20. Trujillo, R.A., *Guía de implementación del Manejador Demo.* 2007.
21. Eclipse, *Open UP/Basic*. 2008.
22. Johnson, R.G., *Capa de acceso a datos síncrona y asíncrona para dispositivos Modbus TCP*. 2008, Universidad de Ciencias Informáticas: Ciudad de La Habana.
23. Hernández, L.E.G., R.G. Johnson, and A.C.M. Borges, *Manejadores GE Fanuc y BSAP Serial para el proyecto SCADA PDVSA*. 2009.
24. Morales, C.L.d.C.y.R., *Introducción a SCADA.* 2010.
25. cplusplus.com. *C++ Library Reference*. 2011; Available from: www.cplusplus.com.

26. Paradigm, V. *Sitio Web de Visual Paradigm*. 2011; Available from: http://www.visual-paradigm.com/product/vpuml/.
27. Systems, Z., *Zator Systems: Tecnología de la información para el conocimiento.* 2011.

[1-5, 7-27]

GLOSARIO

ASCII: (acrónimo inglés de American Standard Code for Information Interchange). Es un código estándar para el Intercambio de Información. Utiliza 7 bits para representar los caracteres, aunque inicialmente empleaba un bit adicional (bit de paridad) que se usaba para detectar errores en la transmisión.

Bus de campo: Es un sólo enlace de comunicaciones, al cual se conectan directamente todos los dispositivos. Existen dos formas de comunicación en esta topología, colisión y maestro/esclavo.

Dispositivo de campo: Son los elementos físicos que miden, monitorean y, en algunos casos almacenan, los datos de las variables del proceso. Estos dispositivos no se conectan directamente al SCADA.

Multiplataforma: Término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de software, que puedan funcionar en diversas plataformas. Por ejemplo, una aplicación multiplataforma podría ejecutarse en Windows en un procesador x86, en GNU/Linux en un procesador x86, y en Mac OS X en uno x86.

Interfaz: Zona de contacto o conexión entre dos componentes de hardware; entre dos aplicaciones; o entre un usuario y una aplicación. Apariencia externa de una aplicación informática.

Introspección: Es la capacidad de los objetos mostrar sus propiedades, obtener el valor de esas propiedades a partir del nombre de la misma y modificar su valor igualmente a partir del nombre [20].

Trama: Es una unidad de envío de datos. Viene a ser sinónimo de paquete de datos.

1