Desarrollo De Un Módulo Que Implemente Las Funcionalidades Del Protocolo RTPS Para Ser Utilizado en Aplicaciones de Tiempo Real

Andrés X. RUBIO[1], Alejandra B. TELLO[2], Xavier A. CALDERÓN[3]

Facultad de Ingeniería Eléctrica y Electrónica, Escuela Politécnica Nacional

[1]andresrubiop@msn.com
[2]alejitat_28@hotmail.com
[3]xavier.calderon@epn.edu.ec

***Resumen*—El Proyecto de titulación comprende el desarrollo de un módulo que implementa funcionalidades del protocolo RTPS para aplicaciones distribuidas de tiempo real.**

El módulo contempla una implementación básica del DDS, RTPS, y la interacción de los mismos, para el funcionamiento adecuado del middleware.

Primeramente se presenta el estado actual de los middleware de comunicaciones de tiempo real; realizando un breve estudio de cada tecnología incluyendo al middleware DDS[1]. Además se analiza las características y funcionalidades más específicas definidas en el estándar publicado por la OMG[2] sobre DDS y RTPS.

Posteriormente se definen los requisitos necesarios para integrar el protocolo RTPS con el middleware DDS, realizando previamente un análisis de los diferentes paquetes RTPS[3].

Se presenta un breve resumen del diseño que permite la interacción entre DDS y RTPS, el cual posteriormente es implementado.

Finalmente se presenta un ambiente de pruebas donde una implementación del protocolo es probada dentro de cuatro computadores que intercambian información.

I. Introducción

En la actualidad los sistemas distribuidos de tiempo real se encuentran en una etapa de desarrollo donde se puede tomar varios caminos, es decir diferentes arquitecturas y tecnologías, de las cuales el protocolo de comunicación es parte vital en cada una de ellas, por lo que se requiere que al implementar la comunicación exista una correcta interacción entre el sistema distribuido y el protocolo de comunicación.

El presente proyecto se enfoca en diseñar, implementar y probar, el protocolo de comunicaciones RTPS, el cual es parte del middleware DDS.

II. Marco Teórico

A. *Middleware de Comunicaciones*

El Middleware es una capa intermedia de software, el cual se encarga de simplificar el manejo y la programación de aplicaciones, tratando de mantener la complejidad de redes y sistemas heterogéneos transparentes al usuario, por lo que se ha convertido en una herramienta esencial para el desarrollo de sistemas distribuidos.

Los Middleware de comunicaciones, son una abstracción de los detalles de bajo nivel relacionados con la distribución y la comunicación, el cual proporciona las bases para el desarrollo de Middlewares de alto nivel. Este maneja internamente los detalles del proceso de interconexión entre nodos que por lo general incluyen las siguientes características básicas:

- Direccionamiento o asignación de identificadores a entidades con la finalidad de indicar su ubicación.
- Marshalling[4] o la transformación de los datos en una representación adecuada para la transmisión sobre la red.
- Envío o la asignación de cada solicitud a un recurso de ejecución para su procesamiento.
- Transporte o establecimiento de un enlace de comunicaciones para el intercambio de mensajes entre redes vía unicast o multicast.

En la Figura 1, se aprecia los servicios básicos que provee un Middleware.

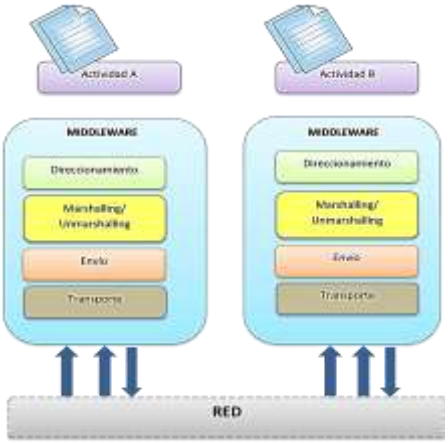

Figura 1. Servicios básicos provistos por el Middleware de distribución [1]

[1] DDS, Data Distributed System
[2] OMG, Object Management Group
[3] RTPS, Real-Time Publish-Suscribe Protocol

[4] Marshalling, es un mecanismo ampliamente usado para transportar objetos a través de una red.

B. Middlewares de Tiempo Real

Un sistema de tiempo real se define como un tipo especial de sistema cuya corrección lógica se basa tanto en la exactitud como también en la disminución de retardos en la información. En los sistemas de propósito general, el uso de la tecnología de Middlewares tiene como objetivo facilitar la programación de aplicaciones distribuidas. Con este fin, el Middleware proporciona una abstracción de alto nivel de los servicios ofrecidos por los sistemas operativos, sobre todo los relacionados con la comunicación.

Los desarrolladores son responsables de definir que parte de la aplicación puede ser accesible de forma remota, mientras el Middleware establece y gestiona transparentemente la comunicación entre los nodos del sistema distribuido. Además, los sistemas en tiempo real también se benefician de estas abstracciones de alto nivel.

1) CORBA y RT-CORBA[5]

CORBA [2] es un Middleware basado en el modelo de sistema distribuido DOM, el cual utiliza el paradigma Cliente-Servidor y cuya característica principal es facilitar la interoperabilidad entre aplicaciones heterogéneas[6]. Esta arquitectura está integrada por los siguientes componentes:

- Object Request Broker (ORB), representa el núcleo del Middleware y es responsable de coordinar la comunicación entre los nodos cliente y servidor.
- Interfaces del Sistema, estas consisten en un conjunto de interfaces agrupadas en función de su ámbito de aplicación.

A continuación se presenta en la Figura 1 una visión general de la arquitectura CORBA.

Figura 2. Arquitectura CORBA [1]

RT-CORBA nace a partir de las falencias de CORBA para tiempo real, las cuales se concentran en creación y destrucción de entidades en tiempo real, en los mecanismos de sincronización para controlar el acceso a recursos compartidos, y en los mecanismos para controlar el grado de concurrencia durante la ejecución de las llamadas remotas.

2) The Ada Distributed System ANNEX [3]

Es un estándar internacional que incluye un anexo dedicado al desarrollo de aplicaciones distribuidas, la principal fortaleza de DSA es que el código fuente está escrito sin tener en cuenta de si va a ser ejecutado en una plataforma distribuida o en un solo procesador.

En el lenguaje de programación Ada cada parte de la aplicación se asigna de forma independiente a cada nodo la cual es llamada partición. Las particiones se comunican entre sí mediante el intercambio de datos a través las RPC y de los objetos distribuidos. La DSA define dos tipos de particiones: activos, los cuales pueden ser ejecutados en paralelo uno con otro; y pasivos, los cuales son particiones sin una tarea o hilo de control.

Los componentes de alto nivel del modelo de distribución de la DSA están ilustrada en la Figura 3 la cual representa el diagrama de secuencia de una llamada remota sincrónica entre dos particiones.

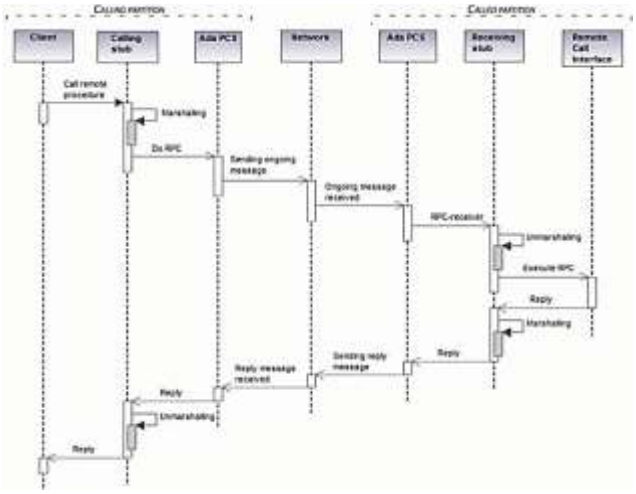

Figura 3. Diagrama de secuencia de una llamada remota síncrona [1]

3) The Distributed Real-Time Specification for Java

Java fue diseñado inicialmente como un lenguaje de programación para sistemas de propósito general y por lo tanto tiene varios inconvenientes para el desarrollo de aplicaciones previsibles, especialmente en aspectos relacionados con la gestión de los recursos internos como la memoria o la programación del procesador [4]. Para los sistemas distribuidos de tiempo real, uno de los trabajos de investigación más notable es la Distributed Real-Time Specification for Java o DRTSJ [5], que integra las dos tecnologías existentes de Java:

- Real-Time Specification for Java [6], el cual logró el soporte en tiempo real a través de nuevas bibliotecas, un mejor mecanismo de Java, y una Máquina de Java en tiempo real.
- Remote Method Invocation [7], el cual define un modelo DOM basado en objetos Java que definen una nueva interfaz, llamada remota, permitiendo la diferenciación de objetos distribuidos de los locales. La arquitectura RMI se muestra en la Figura 4, que representa el diagrama de secuencia de una llamada remota asincrónica entre un cliente y un servidor.

[5] RT-CORBA, CORBA de Tiempo Real

[6] Aplicaciones Heterogéneas, se refiere a las aplicaciones codificadas en diferentes lenguajes de programación, ejecución en diferentes plataformas y/o las implementaciones de Middlewares desarrolladas por diferentes empresas.

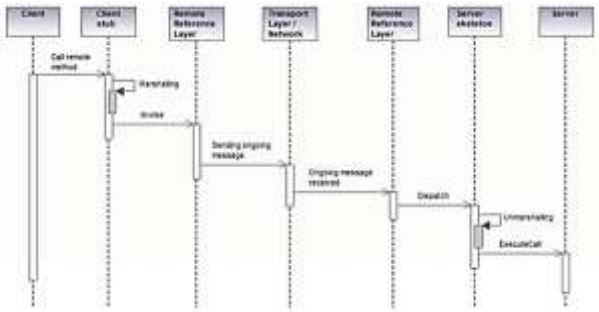

Figura 4. Diagrama de secuencia de una llamada remota asíncrona [1]

- Cliente Stubs o Proxy y Servidor Skeleton, los cuales representan la interfaz entre la capa aplicación y el resto del sistema RMI.
- Remote reference Layer, encargado de la semántica de las invocaciones remotas.
- Capa Transporte, utilizada para establecer las conexiones y gestionar los detalles de comunicación de bajo nivel.

4) The Data Distriution Service for Real-Time Systems

El DDS [8] facilita el intercambio de datos en sistemas distribuidos a través del paradigma publicador-suscriptor. Una arquitectura publicador suscriptor promueve un bajo acoplamiento en la arquitectura de datos y es flexible y dinámica; es fácil de ser adaptada y extendida en sistemas basados en DDS, esta se conecta a los generadores de información anónima (el publicador) con los consumidores de información (el suscriptor).

Una visión general de la arquitectura se muestra en la Figura 5, está describe dos niveles de interfaces y un nivel de comunicaciones:

- Un nivel bajo denominado data-centric Publish-Subscribe o DCPS, el cual está orientado a la entrega eficiente de información adecuada a los destinatarios correctos.
- Un nivel opcional alto denominado data-local reconstruction layer o DLRL, el cual permite una integración simple a la capa aplicación.
- El nivel de comunicaciones se denomina DDS Interoperability Wire Protocol, el cual opera con el Protocolo RTPS.

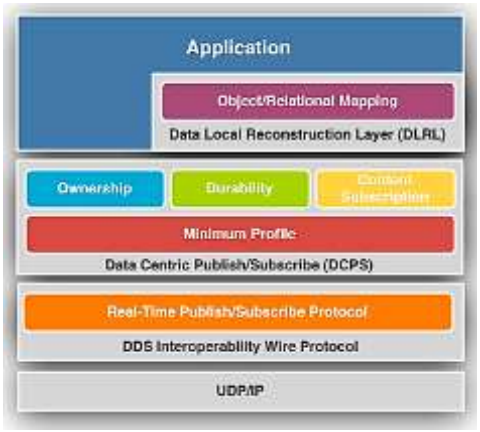

Figura 5. Arquitectura del Middleware DDS [9]

C. Real-Time Publish-Subscribe Protocol

RTPS fue específicamente desarrollado para soportar los requerimientos únicos de los sistemas de datos distribuidos. La comunidad de automatización industrial define los requerimientos para un protocolo de Publicación-Suscripción que trabaje con DDS.

Las principales características del protocolo RTPS son:

- Rendimiento y Calidad de Servicio, los que permiten tener comunicación segura entre el Publicador y Suscriptor.
- Tolerancia a fallos para permitir la creación de redes sin puntos de fallos
- Extensibilidad para permitir que le protocolo sea extendido y mejorado con nuevos servicios.
- Conectividad Plug and Play para que las nuevas aplicaciones y servicios estén automáticamente descubiertos.
- Configurabilidad para permitir el balanceo de requerimientos para la confiabilidad y la puntualidad de cada entrega de datos.
- Capacidad para permitir que los dispositivos implemente un subconjunto del protocolo y que aun así participen en la red.
- Escalabilidad para sistemas que potencial mente escalen en redes extensas.
- Seguridad de tipo de datos para prevenir errores en la programación de aplicaciones.

En la Figura 6 se puede observar la interacción del protocolo RTPS con DDS por medio de los diferentes submódulos que lo componen.

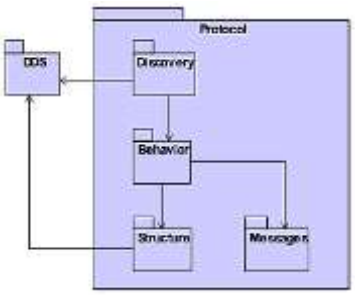

Figura 6. Módulos RTPS [1]

III. Análisis de Requerimientos

Los requerimientos necesarios para soportar el protocolo RTPS con el Middleware DDS, son los siguientes:

- Se deberá implementar los componentes básicos que conforman el DDS, tales como el publicador, suscriptor y el topic.
- Se deberá implementar los mecanismos y técnicas para el alcance de la información, es decir se deberá organizar los datos dentro de cada dominio.
- Se deberá implementar los mecanismos de Lectura y Escritura de datos, y el ciclo de vida de los Topic.
- Se deberá implementar los componentes necesarios del protocolo RTPS y el mecanismo de descubrimiento proporcionado por el mismo.

A continuación se presenta una breve explicación sobre el contenido de cada requisito.

1) Módulo DDS

Dentro del módulo DDS se encuentra al Publicador, al Suscriptor, y al Topic.

- El **Publicador** será el responsable de la distribución de datos, podrá publicar de los diferentes tipos de datos. Un *DataWriter* se encargará de comunicar a un publicador la existencia de datos de un tipo dado.
- El **Suscriptor** será el responsable de recibir los datos publicados y podrá recibir y despachar datos. Para acceder a los datos recibidos, la aplicación deberá utilizar un *DataReader* adjunto al suscriptor.
- Un **Topic** representará la unidad de información que puede ser producida o consumida; estará compuesta por un tipo, un nombre único y un conjunto de políticas de calidad de servicio, como se muestra en la Figura 7

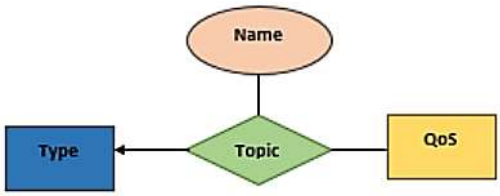

Figura 7. Objeto Topic y sus componentes. [9]

2) Mécanismo y Técnicas Para el Alcance de la Información

Los dominios y las particiones serán la manera de organizar los datos. El Topic DDS permitirá crear *topics*, que limitará los valores que pueden tomar sus instancias. Al suscribirse a un *topic* una aplicación, sólo recibirá, entre todos los valores publicados aquellos valores que coincidan con el filtro del *topic*. Los operadores de los filtros y condiciones de consultas se muestran en la Tabla 1.

Tabla 1. Operadores para Filtros DDS y Condiciones de Consulta

Operador	Descripción
=	Igual
<>	Diferente
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
BETWEEN	Entre y rango inclusivo
LIKE	Búsqueda para un patrón

Estos, limitarán la cantidad de memoria utilizada por el Middleware.

3) Lectura y Escritura de Datos

a) Escritura de Datos

Escribir datos dentro del DDS será un proceso simple ya que solo se deberá llamar al método *write* del *DataWriter*. Este permitirá a una aplicación establecer el valor de los datos para ser publicados bajo un determinado Topic.

El ciclo de vida del *topic-instances*, podrá ser manejado implícitamente a través de la semántica implicada por el *DataWriter*, o podrá ser controlada explícitamente por la API *DataWriter*. La transición del ciclo de vida de *topic-instances* puede tener implicaciones en el uso de recursos locales y remotos.

b) Lectura de Datos

El DDS tendrá dos mecanismos diferentes para acceder a los datos, cada uno de ellos permitirá controlar el acceso a los datos y la disponibilidad de los mismos. El acceso a los datos se realizará a través de la clase *DataReader* exponiendo dos semánticas para acceder a los datos: *read* y *take*.

- La semántica del *read* implementado por el método *read*, dará acceso a los datos recibidos por el *DataReader* sin sacarlo de su caché local. Por lo cual estos datos serán nuevamente legibles mediante una llamada apropiada al *read*.
- La semántica del *take* implementado por los métodos *take* que permitirá acceder a los datos recibidos por el *DataReader* para removerlo de su caché local.

c) Notificaciones

Una forma de coordinar con DDS será tener un sondeo de uso de datos mediante la realización de un *read* o un *take* de vez en cuando. El sondeo podría ser el mejor enfoque para algunas clases de aplicaciones, el ejemplo más común es en las aplicaciones de control. En general, las aplicaciones podrán ser notificadas de la disponibilidad de datos o tal vez esperar su disponibilidad. El DDS apoyará la coordinación por medio de *waitsets* y los *listeners*.

- Los Waitsets, proporcionan un mecanismo genérico para la espera de eventos. Uno de los tipos soportados de eventos son las condiciones de lectura, las cuales podrán ser usadas para esperar la disponibilidad de los datos de uno o más *DataReaders*.
- Los Listeners, aprovechan al máximo los eventos planteados por el DDS y asincrónicamente notificar a los *handler* (controladores) registrados. Por lo tanto, si se quiere un *handler* sea notificado de la disponibilidad de los datos, se deberá conectar el *handler* apropiado con el evento *on_data_available* planteado por el *DataReader*.

4) Módulo RTPS

Los módulos RTPS están definidos por la PIM. La PIM describe el protocolo en términos de una "máquina virtual." La estructura de la máquina virtual está construida por clases, las cuales están descritas en el módulo de estructura, además este incluye a los *endpoints* de los *Writer* y *Reader*. Estos extremos se comunican usando los mensajes descritos en el módulo de mensajes. También es necesario describir el comportamiento de la máquina virtual, por medio del módulo de comportamiento, en el cual se observa el intercambio de mensajes que debe tomar lugar entre los extremos. Por último se encuentra el protocolo de descubrimiento usado para configurar la máquina virtual con la información que esta necesita para comunicar a los pares remotos, este protocolo se encuentra descrito en el módulo de descubrimiento.

a) Módulo Estructura

El propósito principal de este módulo es describir las clases principales que serán utilizadas por el protocolo RTPS, como se puede observar en la Figura 8.

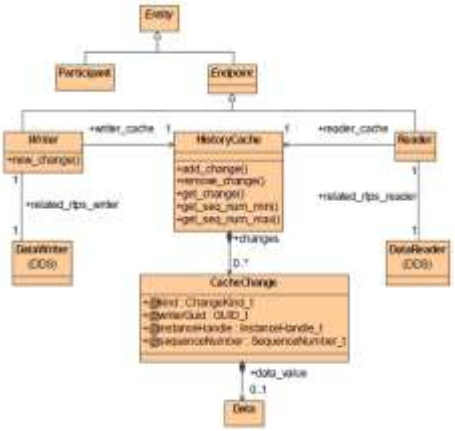

Figura 8. Módulo Estructura

b) Módulo Mensajes

Este módulo describe la estructura y contenidos lógicos globales de los mensajes que se intercambian entre los puntos finales del *Writer* RTPS y los puntos finales del *Reader* RTPS. Los mensajes RTPS son de diseño modular y se pueden ampliar fácilmente para apoyar tanto nuevas características del protocolo, así como extensiones específicas del proveedor.

c) Estructura general del Mensaje RTPS

Consta de una cabecera RTPS de tamaño fijo, seguido de un número variable de Submensajes RTPS. Cada submensaje a su vez consta de un *SubmessageHeader* y un número variable de *SubmessageElements.* Esto se muestra en la Figura 9.

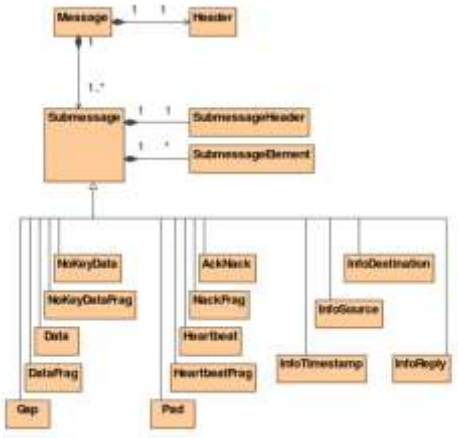

Figura 9. Estructura del mensaje RTPS

d) Elementos del submensaje RTPS

Cada mensaje RTPS contiene un número variable de submensajes RTPS. Cada submensaje RTPS a su vez, se construye a partir de un conjunto de bloques llamados *SubmessageElements*. El RTPS versión 2.2 define los siguientes elementos: submensaje *GuidPrefix, entityId, sequenceNumber, SequenceNumberSet, FragmentNumber, FragmentNumberSet, VendorID, ProtocolVersion, LocatorList, TimeStamp, Count, SerializedData* y *ParameterList.*

e) Submensajes RTPS

El protocolo RTPS de la versión 2.2 define varios tipos de submensajes. Se clasifican en dos grupos: EntitySubmessages e Interpreter-Submessages.

El *Entity submessage* se dirige a una Entidad RTPS. El *Interpreter-Submessages* modifica el estado del receptor RTPS y proporcionará el contexto que ayuda a los procesos posteriores del *Entity submessage*.

Las entidades del submensaje son:

- El submensaje *Data,* contiene información sobre el valor de un objeto fecha de la aplicación. Los submensajes de datos son enviados por el *Writer* a un *Reader*.
- El *DataFrag*, equivale a los datos, pero sólo contiene una parte del valor (uno o más fragmentos). Permite que los datos se transmitan como varios fragmentos para superar las limitaciones de tamaño de mensajes de transporte.
- El *HeartBeat*, describe la información que está disponible en un *Writer*. Los mensajes *HeartBeat* son enviados por un *Writer* a uno o más *Reader*.
- El *HeartbeatFrag*, sirve para los datos fragmentados, describe que fragmentos están disponibles en un Writer. Los submensajes *HeartbeatFrag* son enviados por un *Writer* a uno o más *Reader*.
- El *GAP*, describe la información que ya no es relevante para el *Reader*. Los mensajes *GAP* son enviados por un *Writer* a uno o más *Reader*.
- El *AckNack*, proporciona información sobre el estado de un *Reader* a un *Writer*. Los mensajes *AckNack* son enviados por un *Reader* a uno o más *Writer*.
- El *NackFrag*, proporciona información sobre el estado de un *Reader* a un *Writer*, específicamente los fragmentos de información que siguen perdidos en el *Reader*. Los submensajes *NackFrag* son enviados por un *Reader* a uno o más *Writer*.

Los submensajes de interpretación son:

- El *InfoSource*, proporciona información acerca de la fuente de donde se originaron los Entity Submessage posteriores. Este submensaje se utiliza principalmente para la retransmisión de los submensajes RTPS.
- El *InfoDestination*, proporciona información sobre el destino final de los submensajes que le acompañan. Este submensaje se utiliza principalmente para la retransmisión de submensajes RTPS.
- El *InfoReply*, proporciona información sobre donde responder a las entidades que figuran en submensajes posteriores.
- El *InfoTimestamp*, proporciona una marca de tiempo a los submensajes que le acompañan.
- El *Pad*, se utiliza para agregar relleno a un mensaje, siempre y cuando sea necesaria la alineación de la memoria.

5) Módulo Comportamiento

Una ver que el *Writer* RTPS sea asociado a un *Reader* RTPS, es responsabilidad de ambos, asegurarse que los cambios en el *CacheChange* que existen en la *HistoryCache* de los diferentes *Writers* sean propagados a la *HistoryCache* de los diferentes *Readers*.

Este módulo describe como los pares de *Writer y Reader* RTPS asociados deben comportarse para propagar los cambios en el *CacheChange*. Este comportamiento está definido en términos de los mensajes intercambiados usando a los mensajes RTPS.

a) Requerimientos Generales

Los siguientes requerimientos aplican a todas las entidades RTPS.

- Todas las comunicaciones deberán tomar lugar usando mensajes RTPS, es decir que ningún otro mensaje que no está definido en los mensajes RTPS puede ser usado.
- Se debe implementar un *Message Receiver* RTPS, es decir que para interpretar a los submensajes RTPS se deberá usar esta implementación.
- Las características de tiempos en todas las implementaciones deben ser configurables.
- Se debe implementar el protocolo de descubrimiento denominado *Simple Participant and Endpoint Discovery Protocols*, es decir a los protocolos de descubrimientos que cubre el estándar.

6) Módulo Descubrimiento

El módulo descubrimiento definirá el protocolo de descubrimiento RTPS. El propósito del protocolo de descubrimiento permitirá que cada *participante* RTPS descubra otros relevantes *participantes* y sus *endpoint*. Una vez que el *endpoint* ha sido descubierto, las implementaciones podrán configurar *endpoint* locales para establecer la comunicación.

La especificación RTPS divide al protocolo de descubrimiento en dos protocolos independientes:

- *Participant* Discovery Protocol (PDP)
- *Endpoint* Discovery Protocol (EDP)

Un PDP especifica como los participantes se descubren entre sí en la red. Una vez que dos *participantes* se han descubierto, intercambian información sobre los *endpoint* que los contienen utilizando un EDP. Aparte de esta relación de causalidad, ambos protocolos se pueden considerar independientes.

A fin de la interoperabilidad, todas las implementaciones RTPS deben proporcionar al menos los siguientes protocolos de descubrimiento:

- Simple Participant Discovery Protocol (SPDP), el objetivo es descubrir la presencia de otros participantes en la red y sus propiedades
- Simple Endpoint Discovery Protocol (SEDP), define la información intercambiada requerida entre dos *participantes* para descubrir *Writer* y *Reader Endpoint.*

Ambos son protocolos básicos de descubrimiento que bastan para pequeñas redes de mediana escala. Los PDP adicionales y EDP que están orientados hacia las redes más grandes se pueden añadir a las futuras versiones de la especificación.

Finalmente, el rol de un protocolo de descubrimiento será proporcionar información sobre *endpoint* remotos descubiertos.

IV. Diseño e Implementación de un Módulo que permita interactuar al Protocolo RTPS con DDS

Se diseña un módulo que permita la interacción entre RTPS y DDS y que trabaja con el modelo publicador/suscriptor, por medio de diagramas de clases del módulo RTPS.

Seguidamente se muestra la implementación del diseño, utilizando normas de convención, implementación del protocolo RTPS, implementaciones necesarias DDS.

A. Diseño

A partir del API proporcionado por la OMG el cual es el API-RTPS, y de la recopilación de información necesaria, y establecidos los requerimientos para soportar el protocolo RTPS con el Middleware DDS, se procede con el diseño del módulo.

1) Submódulo de Mensaje y Encapsulación

El submódulo de mensaje y encapsulamiento es el encargado de los codificadores, del encapsulamiento y de la administración del encapsulamiento, para lo cual se define una serie de clases que permiten la codificación y decodificación de los diferentes mensajes RTPS, cabeceras, e identificadores.

A partir del submódulo de mensaje y encapsulación del API-RTPS, se creó nuevas clases, las cuales son necesarias para la correcta implementación del submódulo, por ejemplo se definió los codificadores y decodificadores de cada tipo de submensaje, como también las clases necesarias para serializar los campos de un mensaje RTPS. En la Figura 10, se muestra algunas de las clases de este submódulo.

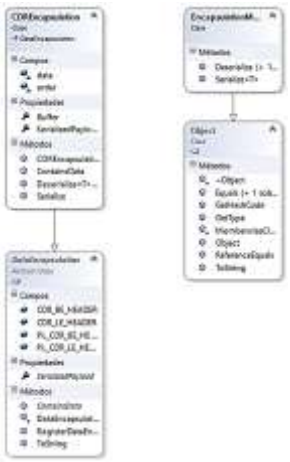

Figura 10. Clases Submódulo de Mensaje y Encapsulación

2) Submódulo de Descubrimiento

El submódulo de descubrimiento es el encargado de los mensajes de descubrimiento y de finalizar el descubrimiento de los participantes, para lo cual se define una serie de clases que permiten el descubrimiento de mensajes RTPS a partir del API-RTPS. En la Figura 11, se muestra algunas de las clases de este submódulo.

Figura 11. Clases Submódulo de Descubrimiento

3) Submódulo de Comportamiento

El submódulo de comportamiento es el encargado de la interfaz con el módulo DDS, de los *writer* y *reader* y del funcionamiento con estado y sin estado, para lo cual se define una serie de clases en base al API-RTPS que permiten mostrar el comportamiento de la comunicación. En la Figura 12, se muestra algunas de las clases de este submódulo.

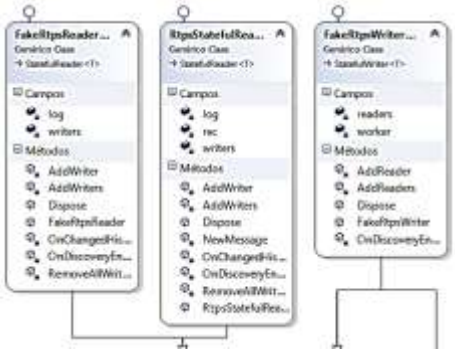

Figura 12. Clases Submódulo de Comportamiento

4) Submódulo de Transporte

Se diseñó un sistema de transporte, el que se encarga del envío de datos, trabajando a nivel local, luego a nivel remoto. Esto se realizó tanto para los mensajes de descubrimiento como para los mensajes RTPS. Finalmente teniendo las clases necesarias para poner mensajes sobre la red se procedió a implementarlas para el envío de mensajes RTPS y de mensajes de descubrimiento RTPS trabajando en conjunto con el envío de mensajes sobre la red. En la Figura 13, se muestra algunas de las clases de este submódulo.

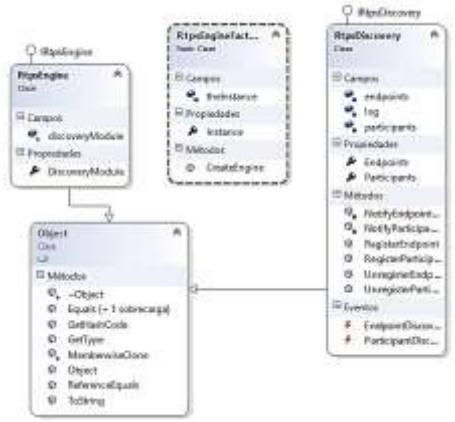

Figura 13. Clases Submódulo de Transporte

5) Submódulo de Configuración

El submódulo de configuración es el encargado de interpretar un archivo de configuración, el cual tiene parámetros configurables tanto del módulo DDS como del módulo RTPS. A continuación se muestra en Figura 14 la configuración básica de RTPS.

Figura 14. Diseño Submódulo de Configuración

B. Implementación

1) Submódulo de Transporte

a) Implementación del transmisor UDP

Dentro de la implementación del transmisor UDP, el inicio de la comunicación del lado del transmisor constituye parte vital del proyecto. Se observa que tanto en el transmisor, como en el receptor se utiliza el protocolo que trabaja sobre IP, denominado UDP, el cual solo es útil por su facilidad de uso, ya que RTPS es un protocolo que trabaja en un nivel superior al de la capa transporte.

b) Implementación de la Maquinaria RTPS

La maquinaria RTPS, se refiere a toda la configuración de perfiles necesarios para trabajar con RTPS, para lo cual primeramente se obtiene del archivo de configuración las instancias, es decir los parámetros del funcionamiento del protocolo, donde se encuentran tiempos, retardos y QoS.

2) Submódulo de Mensaje y Encapsulación

a) Implementacion de los mensajes RTPS

Para poner mensajes en la red se utiliza un método que permite alinear el mensaje a los 32 bits como define la norma, luego se define el *Endianess* y finalmente el tipo de submensaje a enviar.

b) Implementación de Descubrimiento

Para poder realizar el descubrimiento es necesario primeramente añadir a los participantes, estos tienen asociada una QoS, y se envían dentro de algún protocolo de descubrimiento. Estos participantes deben tener una identificación que es autogenerada en el programa.

Así como para el protocolo SEDP, es necesario tener la correspondiente clase que implemente el protocolo SPDP en base a una configuración de transporte predefinida, la cual configura varios parámetros de configuración de calidad de servicio.

3) Submódulo de Comportamiento

a) Implementacion Reader RTPS

Para la implementación de un lector RTPS con estado y sin estado, se permite la creación de un mensaje, escogiendo el tipo de mensaje, después almacenando el mensaje en un *buffer*, seguidamente se lo encapsula y serializa. Luego del proceso de creación del mensaje, el *Reader* se encarga de informar al *HistoryCache* que se ha generado un cambio por medio del objeto *CacheChange*. El *buffer* cuando se trabaja sin estado, tiene un tamaño limitado.

b) Implementacion Writer RTPS

Tanto para los *Writer* con estado y sin estado, es importante la implementación del método *PeriodicWork*, el cual se encarga de anunciar repetitivamente la disponibilidad de datos por medio

del envío de submensajes *Heartbeat.* Con la diferencia que en los escritores sin estado no se guarda más que un cambio.

V. Pruebas

A. Prueba de una aplicación con RTPS y DDS

La siguiente aplicación de la cual se muestra su funcionamiento en las figuras siguientes, trata de la implementación de un sistema de comunicación de datos básico, en este caso un chat, el cual trabaja con las librerías generadas en este proyecto, es decir que trabaja con la implementación de DDS y RTPS.

A continuación, se muestra la interfaz de usuario y las capturas de paquetes.

1) Interfaz de Usuario.

a) Ejecutar la aplicación

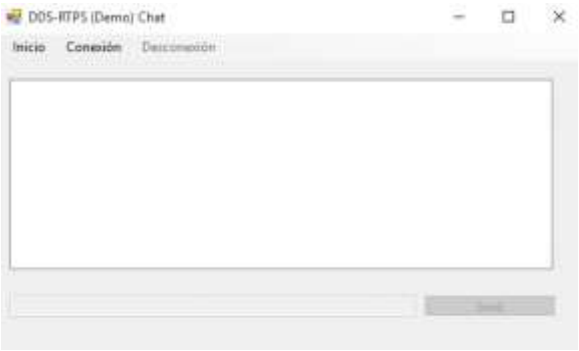

Figura 15. Ejecución del Chat RTPS

b) Conectarse al servicio de chat

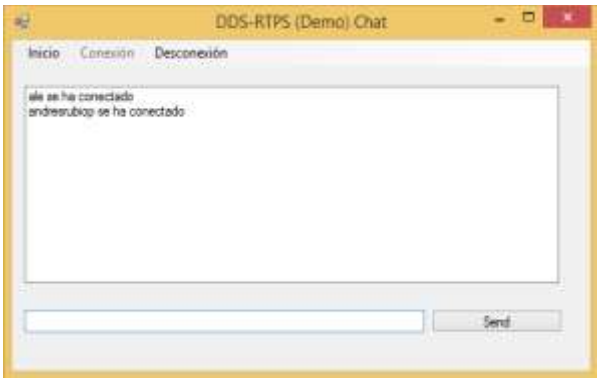

Figura 16. Conexión al servicio de Chat.

c) Intercambio de mensajes

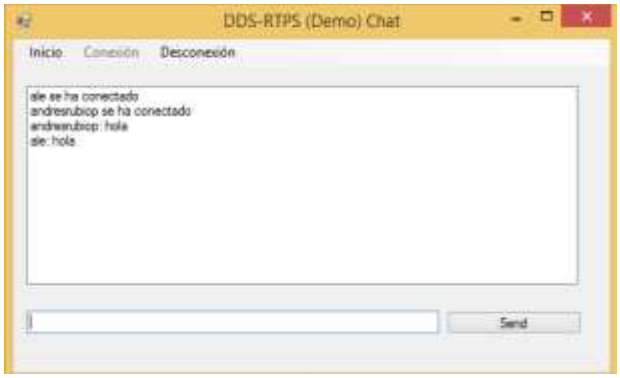

Figura 17. Intercambio de mensajes

d) Desconexión del servicio de chat

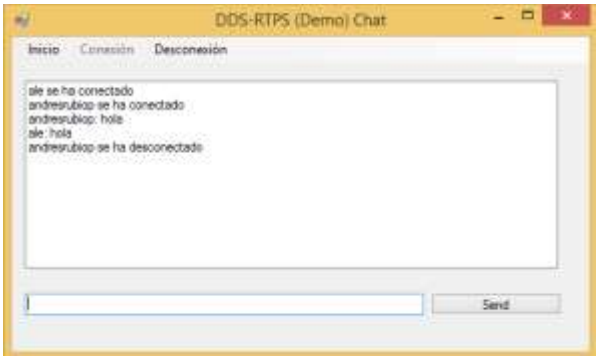

Figura 18. Desconexión del servicio de Chat.

e) Salida de la aplicación

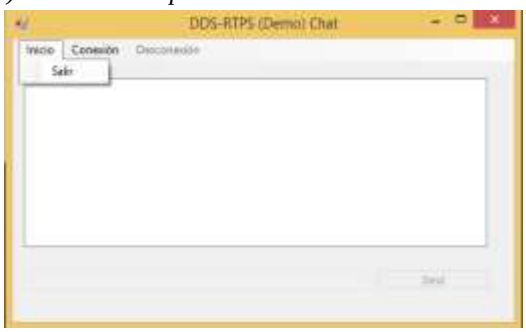

Figura 19. Salida de la aplicación.

f) Captura de paquetes.

Figura 20. Captura paquetes RTPS

VI. Conclusiones

- El modelo de abstracción de datos dentro del Middleware DDS, en el cual se tiene al Publicador y al Suscriptor, centraliza la obtención de datos, es decir, se independiza el origen; facilitando la difusión asincrónica de la información la cual es un requisito común en varias aplicaciones distribuidas.
- La gestión de recursos del procesador propuesto por el Middleware DDS-RTPS, no incluye planificación en los procesadores, pero define varios parámetros de tiempo los cuales sirven para construir el hilo o los hilos en el procesador, que son necesarios para la escucha, o estructura de espera y establecimiento, o pedido de la disponibilidad de datos.
- La gestión de recursos de red presentes en el Middleware DDS-RTPS, puede provocar un incremento en los tiempos de respuesta de las aplicaciones, aunque esta sobrecarga depende de casi exclusivamente de cada aplicación, este efecto es más significativo dentro de DDS-RTPS ya que define un conjunto de entidades que consumen recursos del procesador y de la red.
- El estándar DDS fue diseñado explícitamente para construir sistemas distribuidos en tiempo real, añadiendo un conjunto de parámetros de calidad de servicio para configurar propiedades no funcionales y también permitiendo la

reconfiguración dinámica del sistema, es decir, modificar parámetros en tiempo de ejecución.

- El uso de pruebas unitarias permite comprobar que los componentes de la aplicación trabajen de la manera esperada, además permite mejorar el código para que tenga un mejor funcionamiento, se las pueden realizar independientemente del lenguaje de programación o de la plataforma de desarrollo utilizada.
- Dentro del comportamiento de la interacción entre las entidades DDS y sus correspondientes entidades RTPS, en lo concerniente a la escritura de datos el DataWriter DDS es el encargado de añadir y remover cambios del tipo CacheChange desde y hacia el HistoryCache del Writer RTPS asociado, es decir el Writer RTPS no está en control cuando un cambio es removido desde HistoryCache.
- La implementación sin estado está optimizada para la escalabilidad, esta mantiene virtualmente un estado sumamente simple en las entidades remotas y por lo tanto esta puede escalar de manera adecuada en sistemas grandes. La implementación sin estado es ideal para las comunicaciones con mejor esfuerzo, ya que al trabajar sin estado se requiere menos uso de memoria y por lo tanto la comunicación más rápida.
- La implementación con estado mantiene un total estado en las entidades remotas, esto minimiza el uso de ancho de banda, pero requiere una mayor capacidad de la memoria, y la escalabilidad es reducida, por lo tanto garantiza una comunicación confiable.
- Dentro del descubrimiento existen dos fases, la primera concerniente al protocolo SPDP el cual se encarga de descubrir y anunciar de los participantes y la segunda al protocolo SEDP el cual se encarga de descubrir y anunciar de los servicios que publica el participante.
- El DataWriter es la cara del Publicador, el cual representa a los objetos responsables de la emisión de datos, lo usan los participantes para comunicar el valor y los cambios de los datos; una vez que la nueva información ha sido comunicada al Publicador, es responsabilidad de este determinar cuándo es apropiado emitir el correspondiente mensaje, es decir, lo realiza de acuerdo a su calidad de servicio asociada al correspondiente DataWriter o a su estado interno.
- Para acceder a los datos recibidos, el participante debe utilizar un tipo DataReader asociado al suscriptor, este recibe los datos publicados y los hace disponibles al participante. Un Suscriptor debe recibir y despachar datos de diferentes tipos especificados y asociar a un objeto DataWriter, el cual representa a una publicación, con el objeto DataReader, que representa la suscripción, es hecha por la entidad Topic.
- El Topic tiene el propósito de asociar un nombre único en el dominio, es decir, el conjunto de aplicaciones que se comunican entre sí.
- La semántica proporcionada por las operaciones read y take permite usar el DDS como una caché distribuida o como un sistema de cola, o ambos. Esta es una poderosa combinación que raramente se encuentra en la misma plataforma Middleware. Esta es una de las razones porque DDS es usado en una variedad de sistemas, algunas veces como una caché distribuida de alto rendimiento, otras como tecnología de mensajería de alto rendimiento, y sin embargo, otras veces como una combinación de las dos.
- El uso del lenguaje de programación C#, a pesar de no ser un lenguaje común para programar middleware de comunicación, este nos permite tener una interacción más directa con las herramientas que comúnmente son requeridas al momento de desarrollar aplicaciones dirigidas a los usuarios. Además permite la creación simple de aplicaciones multi-tarea y permite utilizar fácilmente las sobrecargas en métodos.
- La tecnología DDS-RTPS permite desarrollar aplicaciones de comunicación que no se ven afectadas de una manera significativa al momento en que la red de datos deba crecer, es decir que a comparación de otras tecnologías como CORBA-RT al tener un mayor número de usuarios la eficiencia de la comunicación es mínimamente afectada.

VII. REFERENCIAS

[1] H. Pérez y J. J. Gutiérrez, «A survey on standards for real-time distribution middleware,» *ACM Computing Surveys*, vol. 46, nº 49, p. 39, Marzo 2014.

[2] OMG, «Corba Core Specification. v3.2.,» 2011.

[3] ISO/IEC, «Ada 2012 Reference Manual. Language and Standard Libraries—International Standard,» *ISO/IEC*, vol. 8652, 2012.

[4] P. Basanta-Val, M. García-Valls y I. Estévez-Ayres, «An architecture for distributed real-time Java based on RMI and RTSJ.,» de *In Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*, 2010.

[5] Sun Microsystems, «JSR-50: Distributed Real-Time Specification,» 2000.

[6] G. Bollella y J. Gosling, «The real-time specification for Java,» IEEE Computer, 2000.

[7] Sun Microsystems, «Java Remote Method Invocation (RMI),» 2004.

[8] OMG, «Data Distribution Service for Real-Time Systems. v1.2.,» 2007.

[9] A. Corsaro, «Advanced DDS Tutorial,» [En línea]. Available: http://www.prismTech.com/dds-community.

VIII. BIOGRAFÍA