UNIVERSIDAD POLITÉCNICA DE PUEBLA

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Programa Académico de Posgrado

Sistema SCADA para la interoperabilidad de dispositivos médicos en terapia intensiva neonatal

TESIS
QUE PARA OBTENER EL GRADO DE

MAESTRÍA EN INGENIERÍA EN AUTOMATIZACIÓN DE PROCESOS INUSTRIALES

PRESENTA:
Blanca Arely Carballo Almendra

DIRECTOR
Dra. Rita Marina Aceves Pérez

Juan C. Bonilla, Puebla, México, Diciembre 2016.

El presente trabajo fue realizado en el área de Posgrado de la Universidad Politécnica de Puebla, ubicada en Tercer carril del Ejido "Serrano" S/N, San Mateo Cuanalá, Municipio Juan C. Bonilla, Puebla CP 72640.

Apoyo del CONACYT, Beca No. 634814, Programa de Maestría perteneciente al Programa Nacional de Posgrados de Calidad (PNPC-CONACYT).

La presente tesis titulada "**SISTEMA SCADA PARA LA INTEROPERABILIDAD DE DISPOSITIVOS MÉDICOS EN TERÁPIA INTENSIVA NEONATAL"** y realizada por la Ing. Blanca Arely Carballo Almendra, ha sido revisada y aprobada por el Jurado para obtener el Título de:

MAESTRO EN INGENIERÍA EN AUTOMATIZACIÓN DE PROCESOS INDUSTRIALES

UNIVERSIDAD POLITÉCNICA DE PUEBLA

PNPC-CONACYT

Jurado integrado por:

Director: Dra. Rita Marina Aceves Pérez

Revisor: Dr. Salvador Arroyo Díaz

Revisor: Dr. Rafael Rojas Rodríguez

Revisor: Dr. Luis A. Sánchez Gaspariano

Juan C. Bonilla, Puebla, México, Diciembre 2016.

Agradecimientos

Agradezco en primer lugar a Dios por haberme ayudado durante estos años, el esfuerzo fue grande, pero él siempre me dio la fuerza necesaria para continuar y lograrlo.

A mi esposo por la compresión, tolerancia y apoyo económico y moral durante este periodo y creer en mis capacidades para obtener este nuevo logro.

A mis padres por sus consejos y el amor que me brindan a diario a pesar de la distancia.

A mi asesora por su confianza depositada en mí, por su apoyo profesional y por lo más importante por la amistad que me brindó.

Tabla de contenido

Índice de Tablas

Índice de figuras

Nomenclatura

DDS	Servicio de distribución de datos
ICE	Ambiente clínico integrado
QoS	Servicios de calidad
SCADA	Supervisión, control y adquisición de datos
SysML	Lenguaje de modelado de sistemas
CIMIT	Centro para la integración de medicina y tecnología innovadora
MDPnP	Conexión plug `n play de dispositivos médicos
OMG	Grupo de modelado de objetos

Resumen

En el presente escrito se muestra el desarrollo de un sistema realizado para la interconectividad de tres dispositivos médicos a través del estándar DDS el cual fue propuesto por la OMG para sistemas de tiempo real. Este estándar fue implementado por la empresa RTI mediante un software llamado Connext DDS, este mismo es utilizado en este documento para desarrollar las clases correspondientes a la comunicación de la red LAN entre los dispositivos y el ordenador del sistema a implementar. El sistema está compuesto de los dispositivos médicos integrando las tarjetas BeagleBone Black las cuales ejecutarán la aplicación Publicador que será encargada de transmitir los datos a través de la red para que sean recibidos por la aplicación llamada IUM-DDS la cual es programada en java. Esta aplicación integra las clases de comunicación realizadas con Connext DDS, las cuales son encargadas de recibir los datos publicados por las tarjetas y después mandarlos a la interfaz gráfica.

Capítulo 1 Planteamiento del problema de investigación

1.1 Introducción

Hoy en día, existe la necesidad de desarrollar sistemas con la característica de la interoperabilidad de dispositivos médicos ya que ésta es casi inexistente en un medio de trabajo clínico. Los fabricantes de dispositivos médicos por ejemplo Philips, Dräger, Siemens, Teleflex, Medtronic por mencionar algunos, tienen su propio procesamiento de datos y manejo de la información, por lo tanto integrando múltiples dispositivos de diferentes marcas en una área de terapia intensiva neonatal es complicado atender las señales que emiten todos los dispositivos y esto perjudica a la atención del paciente, por lo que se requiere automatizar el proceso de la adquisición de datos, ya que tradicionalmente las enfermeras son encargadas de checar información de todos los dispositivos médicos conectados a un paciente, cada determinado tiempo, en lugar de eso es más óptimo centrar la información obtenida de los dispositivos médicos en una sola interfaz, para que sea más fácil de visualizar. Este requisito no es sólo por comodidad, sino para evitar la posibilidad de errores debido a la complejidad de las interacciones entre los dispositivos y operadores humanos.

Un sistema que soporte dicha interoperabilidad está supuesto para proporcionar los medios para interconectar dispositivos en un medio distribuido, por lo tanto, debe ser diseñado para dar cuenta de los fallos de red. En este documento se presenta el tema denominado "Sistema SCADA para la interoperabilidad de dispositivos médicos en terapia intensiva neonatal", el cual se desarrollará para satisfacer los requerimientos de la empresa Arroba Ingeniería S.A. y del proyecto denominado “Unidad de terapia intensiva

neonatal basada en una arquitectura modular interoperable", financiado por CONACYT. La empresa antes mencionada se dedica a fabricar equipos neonatales para terapia intensiva y desea lograr la interoperabilidad de un oxímetro de pulso y un monitor de signos vitales, por lo cual el objetivo principal de este tema es la implementación de un Sistema SCADA [1] (Supervisión, control y adquisición de datos) con la arquitectura de plataformas de investigación clínica abierta en conjunto con la empresa Arroba para integrar los dispositivos médicos lo cual se ve reflejado en la figura 1, donde se muestra que está conformado de una interfaz capaz de procesar información obtenida de los dispositivos médicos, generar alarmas de las variables que se midan y representar gráficos de las señales obtenidas, además de dispositivos médicos con adaptadores para transmitir los datos y se utiliza un *middleware* encargado de la transmisión y recepción de datos a la interfaz, esta imagen consta de dos etapas, la primera corresponde a la configuración del modelo de datos en los dispositivos médicos y a la realización de la aplicación adaptador, la segunda es correspondiente a la implementación de la comunicación con la interfaz diseñada.

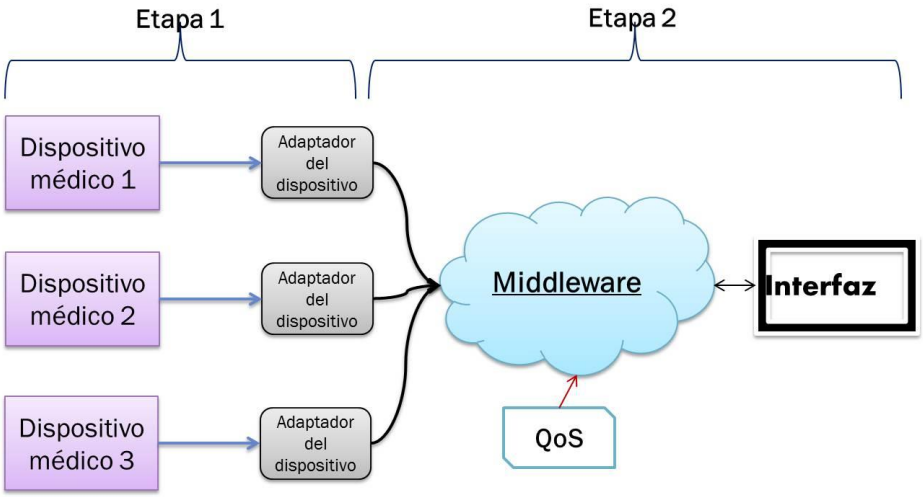

Figura 1. *Diagrama interno del sistema*

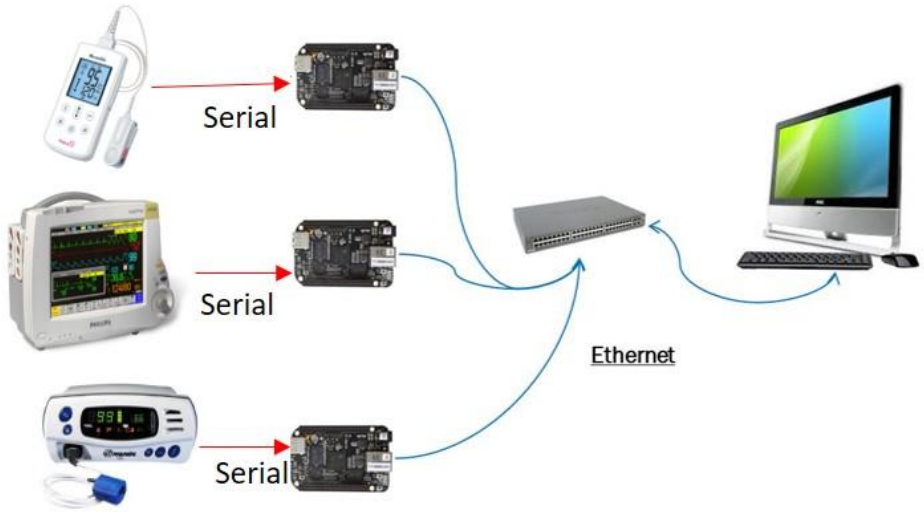

Figura 2. *Sistema físico*

En la figura 2 se pueden observar los componentes físicos que serán ocupados en el sistema donde se encuentran los dispositivos médicos a utilizar, las tarjetas de desarrollo BeagleBone Black que trabajan como adaptadores de los dispositivos y éstas van conectadas al switch al igual que una computadora que será donde se mostrarán los datos de los dispositivos.

Las plataformas de investigación clínica abierta en la que se va a basar este tema son OpenICE [2] y openSDC [3] que son de libre acceso y sirven para la interoperabilidad de dispositivos médicos de marcas específicas. A continuación, se describe de manera muy general cada una.

OpenICE fue creada por el CIMIT (Centro para la integración de tecnología y medicina innovadora), a partir del programa de interoperabilidad MD PnP establecido en el 2004 donde inicializaron los requerimientos para desarrollar el estándar abierto ICE por sus siglas en inglés (Integrated Clinical Environment), que define las condiciones en que la interoperabilidad puede permitir la integración de dispositivos médicos para crear nuevos sistemas con mayor capacidad de seguridad y rendimiento que un solo dispositivo pueda ofrecer [4].

OpenSDC es desarrollada por la empresa Dräger de Alemania en un proyecto de tecnología interno llamado “Device & System Connectivity”, que tenía el objetivo de satisfacer la creciente demanda de interoperabilidad de dispositivos médicos en un entorno clínico integrado (ICE); estas plataformas pretenden que sea un estándar adjunto a las normas del IEEE-11073 [5].

Ambas plataformas se basan en el estándar de comunicación de dispositivos de punto de cuidado comprendidos en un sistema, IEEE-11073 y en el estándar abierto ICE que fue publicado como ASTM F2761-09; estos estándares son una fuente de información relativamente joven en este tema, los trabajos publicados relativos al primer estándar se presentan a partir del año 2005 y son realizados para dispositivos de salud en casa, mas no en ambientes

clínicos integrados, esto permite que la investigación de este proyecto sea significativa en este campo de estudio.

El middleware que se usará es DDS, el cual es un estándar de comunicación para sistemas de tiempo real propuesto por la OMG (Grupo de modelado de objetos) y que satisface los requerimientos de conectividad plug and play, ser en tiempo real, capaz de controlar eventos e incluir el servicio de calidad (QoS).

1.2 Objetivos

1.2.1 Objetivo general

Implementar un sistema SCADA con la arquitectura de una plataforma de investigación clínica abierta en conjunto con la empresa Arroba para lograr la interoperabilidad y control de eventos de las variables medidas de tres dispositivos médicos.

1.2.2 Objetivos específicos

- Desarrollar el modelo de datos de los dispositivos médicos a ocupar para su posterior tratamiento.
- Realizar el modelado de la arquitectura funcional del sistema con SysML para un mejor entendimiento de la operación del sistema.
- Implementar una interfaz utilizando el estándar DDS para que permita monitorear las variables obtenidas de cada uno de los dispositivos médicos en tiempo real.

1.3 Justificación

Actualmente, una unidad de terapia intensiva neonatal es un ambiente altamente complejo donde se mezclan diversas tecnologías médicas tales como monitores de paciente, bombas de infusión, bombas peristálticas, ventiladores pulmonares, sistemas de succión, cuna térmica, etc. Un ambiente de esta manera presenta, entre otros, los siguientes problemas:

- Cada instrumento tiene una filosofía de operación y control distinta mediante teclados de membrana, perillas, botoneras, pantallas táctiles, etc. lo que hace extremadamente difícil controlar los dispositivos de forma coherente.
- La información está distribuida en cada instrumento por separado y con diferentes formatos haciendo inviable la integración de los datos, los cuales frecuentemente y a pesar de la gran cantidad de tecnología presente, deben ser capturados manualmente.
- Los sistemas de alarma están distribuidos en cada instrumento, con señales visuales y auditivas de distinta naturaleza, haciendo difícil identificar en la práctica diaria el origen y significado de cada señal en medio de múltiples opciones concurrentes provocando serios riesgos a la salud del paciente.
- No es posible capacitar al personal de forma integral, cada proveedor da su capacitación de manera independiente, esto se complica considerando la rotación de personal muy frecuente en estas áreas lo que en la práctica degrada la capacidad del personal para operar todos los dispositivos de manera eficiente.
- Hay gran cantidad de redundancia en diversos instrumentos y equipos los cuales cuentan cada uno con sus propias conexiones de alimentación eléctrica, gabinetes y fuentes de poder para la atención de un solo paciente, el estándar actual de diseño requiere un mínimo de 12 conexiones eléctricas disponibles.
- Las áreas físicas son generalmente ruidosas y con iluminación intensa las 24 horas, esto provoca estrés e incomodidad al paciente y sobre todo al personal, en un ambiente médico crítico por sí mismo agresivo, se incrementa aún más la posibilidad del error humano.

Con base en la problemática antes señalada, y observando que en terapia intensiva una enfermera tiene que atender las señales emitidas por todos los dispositivos médicos, se automatizó el proceso de adquisición de datos, diseñando e implementando un sistema para el monitoreo de tres dispositivos médicos, adquiriendo los datos a partir de las tarjetas Beaglebone Black las cuales obtienen las variables clínicas de los dispositivos y son transmitidos a una interfaz en la cual se centra la información de cada uno de

los dispositivos haciendo más rápida la visualización de datos importantes para la asistencia, esto facilita el proceso de gestión del cuidado del paciente.

También se realizará el tratamiento de la información mediante el *middleware* utilizado de RTI donde se controlarán los eventos de las variables clínicas medidas por los dispositivos médicos ocupados en el área de terapia intensiva neonatal. Cabe señalar que sistema propuesto es la base para integrar muchas áreas de aplicación en la automatización futura, como lo son dispositivos clínicos móviles, enviar la información al celular del doctor, telemedicina, enviar la información al registro clínico electrónico, entre otras.

1.4 Contribuciones esperadas

Las contribuciones de esta tesis son las siguientes:

1. Investigación pionera en México de la interoperabilidad de dispositivos médicos.

2. Estructura funcional de un sistema para la interoperabilidad de dispositivos médicos.

3. Interfaz única para monitoreo y control de eventos de tres dispositivos médicos.

1.5 Entregables

- Tabla comparativa de las plataformas OpenICE y OpenSDC.

- Documento con la especificación de los datos que desean adquirir de los dispositivos médicos definiendo su estructura.

- Representación del sistema mediante los siguientes diagramas: de casos de uso, de secuencia, de requerimientos, de clases el cual comprende la jerarquía del programa que se realizará y el de bloques el cual clasifica los componentes del sistema. Estos diagramas serán realizados en SysML [6], donde se plasmarán las características funcionales del sistema.

- Códigos realizados para controlar los eventos de las variables obtenidas de los dispositivos médicos y el tratamiento de la información que integran la interfaz, tomando en cuenta las plataformas de OpenICE y OpenSDC.

- Documento con los resultados de las pruebas del sistema, integrando los dispositivos médicos para verificar la funcionalidad de la interfaz, esperando que cubra las necesidades del ambiente clínico integrado y pueda haber interoperabilidad entre aplicaciones.

Capítulo 2 Marco teórico

Resumen

El capítulo del marco teórico proporciona una realimentación de los principales recursos que se toman en cuenta para el desarrollo del proyecto; se compone de las secciones siguientes: en la primera se tratan las características y partes que componen a un sistema SCADA, como lo son el hardware, software y middleware descritos subsecuentemente y proporcionando ejemplos de los mismos, por último, se mencionan los modelos de comunicación en los que se puede basar un sistema de comunicación. También se mencionan los dispositivos médicos que se tomaron en cuenta para este proyecto. Posteriormente se abordan los trabajos relacionados donde se exponen las plataformas de investigación clínica existentes OpenICE y OpenSDC que son estudiadas con el fundamento de basarse en ellas en este tema de tesis.

2.1 Sistema SCADA

El nombre SCADA significa: (*Supervisory control and data adquisition*, supervisión, control y adquisición de datos) [1].

Aunque inicialmente sólo era un programa que permitía la supervisión y adquisición de datos en procesos de control, en los últimos tiempos han ido surgiendo una serie de productos hardware y buses especialmente diseñados o adaptados para los sistemas SCADA. La interconexión de estos sistemas también es propia, se realiza una interfaz del PC a la planta centralizada, cerrando el lazo sobre la computadora principal de supervisión [7].

2.1.1 Características y funciones de un SCADA

Las características principales de un SCADA [8] son:

- Ser sistemas de arquitectura abierta (capaz de adaptarse de acuerdo a las modificaciones de un proceso o de una empresa)

- Comunicarse con facilidad al usuario con el equipo de planta

- Ser programas sencillos de instalar, sin excesivas exigencias de hardware

Las funciones que rigen a un SCADA se definen en los siguientes puntos [9]:

- **Supervisión remota de instalaciones y equipos;** permite al operador conocer el estado de desempeño de las instalaciones y los equipos alojados en la planta, lo que hace que se puedan dirigir las tareas de mantenimiento y estadística de fallas.

- **Control remoto de instalaciones y equipos;** mediante el sistema se pueden activar o desactivar los equipos remotamente de manera automática y también manual. Además, es posible ajustar parámetros, valores de referencia, algoritmos de control, entre otras características.

- **Procesamiento de datos;** los conjuntos de datos adquiridos conforman la información que alimenta al sistema, ésta es procesada, analizada y comparada con datos anteriores y con datos de otros puntos de referencia, dando como resultado una información.

- **Visualización gráfica dinámica;** el sistema es capaz de brindar imágenes en movimiento que representen el comportamiento del proceso, dándole al operador la impresión de estar presente dentro de una planta real. Estos gráficos también pueden corresponder a curvas de señales.

- **Representación de señales de alarma;** a través de las señales de alarma se logra alertar al operador frente a una falla o la presencia de una condición perjudicial o fuera de lo aceptable. Estas señales pueden ser tanto visuales como sonoras.

- **Almacenamiento de información histórica;** se cuenta con la opción de almacenar los datos adquiridos, esta información puede analizarse posteriormente.

- **Programación de eventos;** se refiere a la posibilidad de programar subprogramas que brinden automáticamente reportes, estadísticas, gráfica de curvas, activación de tareas automáticas, etc.

2.1.2 Operación de un SCADA

Un sistema SCADA se compone de hardware/software que permiten la adquisición de datos de sensores o dispositivos de campo usados en el monitoreo y control de procesos industriales, y permiten la transmisión de comandos/instrucciones para los dispositivos de campo remotos o actuadores [10].

Sus principales elementos son mostrados en la Figura 3.

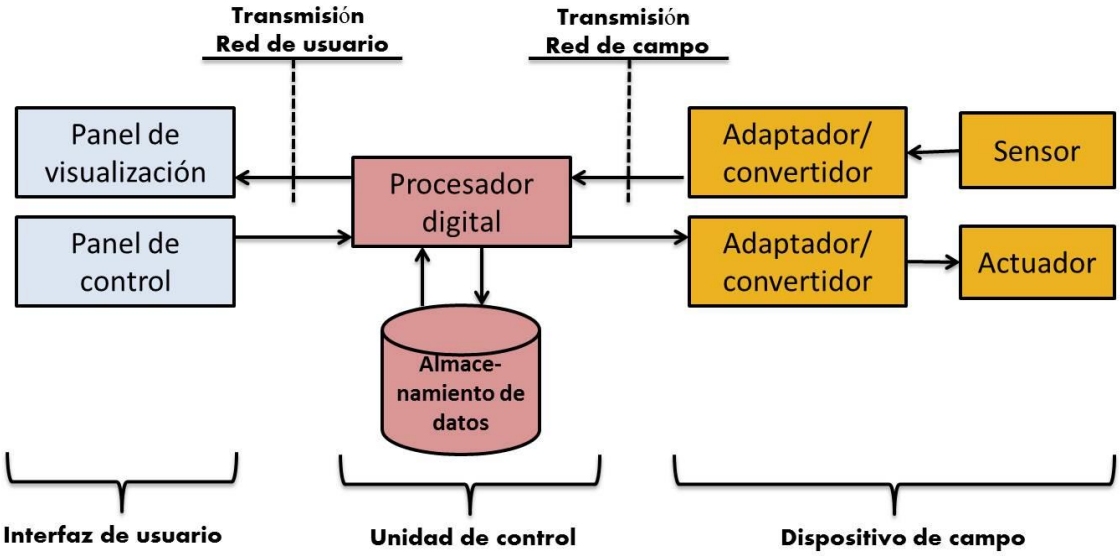

Figura 3. *Estructura de un sistema SCADA*

Se tiene la HMI (interfaz hombre-máquina) la cual proporciona al operador las funciones de control y supervisión de la planta. Seguido de la unidad de control donde se procesan los datos y los dispositivos de campo. A partir de la figura 3 tenemos que para que la comunicación sea viable en este sistema debe tener parte de software, hardware y un middleware que comunique a los dos anteriores.

2.1.2.1 Hardware

La parte de hardware que identifica a un SCADA son los dispositivos de campo, los adaptadores y/o convertidores y los buses de campo o redes. Los adaptadores pueden ser de diferentes tipos para proyectos tecnológicos se toman como tarjetas de desarrollo para adaptar los datos tomados de los dispositivos y adaptarlos a otro tipo de datos que pueda ser leído por una aplicación. Algunos ejemplos de tarjetas de desarrollo más eficientes son Raspberry Pi y BeagleBone Black las cuales tienen las características de una computadora a menor costo, a continuación, se describirán brevemente.

Raspberry Pi; es una tarjeta de desarrollo con placa base de 85 x 54 milímetros, en el que se aloja un chip Broadcom BCM2835 con procesador ARM hasta a 1 GHz de velocidad, GPU VideoCore IV y hasta 512 Mbytes de memoria RAM. En cuanto a su precio, suele estar por debajo de los 40 euros, una de las razones que explica su popularidad. Esta tarjeta dispone de conexión Ethernet para conectar un cable RJ-45 directamente al router o recurrir a adaptadores inalámbricos WiFi. En este caso, conviene que se obtenga una Raspberry Pi que incorpora dos puertos USB, ya que, si no, no se puede conectar un teclado y ratón [11].

Para el almacenamiento, Raspberry Pi recomienda utilizar una tarjeta SD con una capacidad mínima de 4 Gbytes y de clase 4 (este valor aparece siempre impreso en la tarjeta, e indica su rendimiento en cuanto a velocidad se refiere).

BeagleBone Black; el BeagleBone Black es una plataforma de desarrollo de bajo costo del tamaño de una tarjeta de crédito con un buen apoyo de una comunidad de rápido crecimiento. El BeagleBone Black difiere ligeramente de la versión normal que le proporciona un puerto micro HDMI a bordo, 512 MB de DRAM DDR3L, 4GB de memoria interna flash, un procesador AM3358 a 1GHz, y haciendo JTAG opcional con un suministro de usuario principal. En última instancia, la BeagleBone Black sigue siendo perfecta para la computación física y pequeñas aplicaciones embebidas. A continuación, se muestra una tabla donde se exponen las principales características de la BeagleBone normal y la Black (Tabla 1) [12].

Tabla 1. *Comparación de BeagleBone y BeagleBone Black*

	BeagleBone Black $55 Dolares	BeagleBone $89 Dolares
Procesador	AM3358BZCZ100, 1GHZ	AM3359ZCZ72, 720MHz
Salida de video	HDMI	Nula
DRAM	512MB DDR3L 800MHZ	256MB DDR2 400MHz
Flash	4GB eMMC, uSD	uSD
Sobre tarjeta JTAG	Opcional	Si, sobre USB
Serial	Header	Via USB
PWR Exp Header	No	Si
Fuente	210-460 mA@5V	300-500 mA@5V

Para iniciar las tarjetas de desarrollo antes mencionadas es necesario instalar un sistema operativo Linux el más común para estas tarjetas es Debian. Este proceso puede darse de dos maneras la primera es instalarlo sobre una SD y siempre cargar el sistema mediante esta, la segunda opción es cargar la memoria SD con una imagen del sistema operativo y posteriormente flashear la memoria SD sobre la memoria interna de la tarjeta y así se arrancará el sistema desde la memoria interna quedando la SD libre para espacio compartido con la tarjeta. Después de cargar el sistema operativo se debe actualizar la imagen para obtener controladores más recientes [11 y 12].

2.2 Lenguajes utilizados en el sistema

El software utilizable en un SCADA viene dado por una interfaz HMI por sus siglas en inglés, la cual puede ser programada por diferentes lenguajes por ejemplo C, C++, C#, IDL y Java. A continuación, se describen los dos últimos ya que serán utilizados en la presente tesis debido a que java tiene mayor soporte de plataformas y pueden hacerse múltiples aplicaciones con este lenguaje. Por otro lado, IDL es utilizado para la descripción de la interfaz y es requerido por el estándar DDS de RTI.

2.2.1 IDL

IDL (Interface Description Language) es un lenguaje interpretado, orientado al arreglo, de análisis matemático y de despliegue gráfico. Interpretado significa que existe un programa intérprete que procesa el código escrito en este lenguaje. Por esta razón no es posible directamente generar un

programa ejecutable a partir de un programa hecho en IDL, y los programas entonces dependen de la existencia del intérprete en la plataforma sobre la cual se deseen correr los programas [13].

IDL es un estándar de la OMG utilizado en sistemas de computación distribuida para la descripción de los componentes de las interfaces. IDL permite especificar los datos que se intercambiarán en un lenguaje neutral, pudiendo generar componentes en diversos lenguajes, sin que se pierda la interoperabilidad de los mismos [14].

2.2.2 Lenguaje Java

El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems. Java es ampliamente visto como un lenguaje de programación para aplicaciones de Internet orientado a objetos. El lenguaje Java es un derivado del lenguaje C++, por lo que sus reglas de sintaxis se parecen mucho a C++: por ejemplo, los bloques de códigos se modularizan en métodos y se delimitan con llaves ({y }) y las variables se declaran antes de que se usen. Estructuralmente, el lenguaje Java comienza con paquetes. Un paquete es el mecanismo de espacio de nombres del lenguaje Java. Dentro de los paquetes se encuentran las clases que son categorías de objetos que a su vez tienen dentro métodos, variables, constantes, entre otros, [15].

La popularidad de Java, sus características y su independencia de la plataforma le han convertido en un lenguaje de interés para las comunidades de tiempo real y de sistemas empotrados. Esta situación ha motivado el desarrollo de RTSJ (*Real-Time Specification for Java*), que es una extensión del lenguaje para permitir el desarrollo de sistemas de tiempo real. Entre las extensiones que se han incluido, se encuentran las hebras de tiempo real, eventos asíncronos y un modelo de memoria nuevo que permite evitar o limitar el efecto del recolector de memoria. Las ventajas potenciales de Java han motivado su interés en el desarrollo de sistemas de alta integridad [16].

2.3 Middleware

Para lograr que haya transmisión de datos de los dispositivos de campo a la interfaz en un sistema SCADA es necesario ocupar un middleware, el cual se puede implementar en un sistema básico como el que se muestra en la Figura 4.

En la literatura de los sistemas distribuidos existen varias definiciones con respecto al término middleware, pero cada una de ellas se encuentra acorde con la siguiente definición:

Middleware es una capa de software que se encuentra entre el sistema operativo y la aplicación. Esta facilita la comunicación, coordinación e integración de componentes u objetos, mediante un conjunto de servicios que pueden ser internos o externos y se encarga de resolver los problemas de heterogeneidad existentes entre sistemas operativos. El middleware consiste en un conjunto de mecanismos reutilizables que ofrecen soluciones a problemas como, por ejemplo, la heterogeneidad, interoperabilidad, seguridad, fiabilidad, etc. [17].

Hoy en día, la mayoría de los sistemas de middleware se basan en invocación y por lo tanto siguen un paradigma de petición / respuesta: Un cliente pide un servicio particular de un servidor, ya sea enviando un mensaje de solicitud o la realización de una invocación de método remoto (RMI) y luego recibe una respuesta a cambio. Aunque el modo de operación de este tipo funciona bien en una red de área local (LAN) de contexto con un número moderado de los clientes y servidores, no escala a grandes redes como Internet [18].

Sistemas middleware como CORBA o Java RMI han demostrado dar una abstracción útil para la construcción de aplicaciones distribuidas complejas. Proporcionan una interfaz común de nivel superior para el programador de aplicaciones y ocultan la complejidad de tratar con una variedad de plataformas y redes subyacentes, en los siguientes temas se da una descripción más detallada de cada uno de estos sistemas middleware [16].

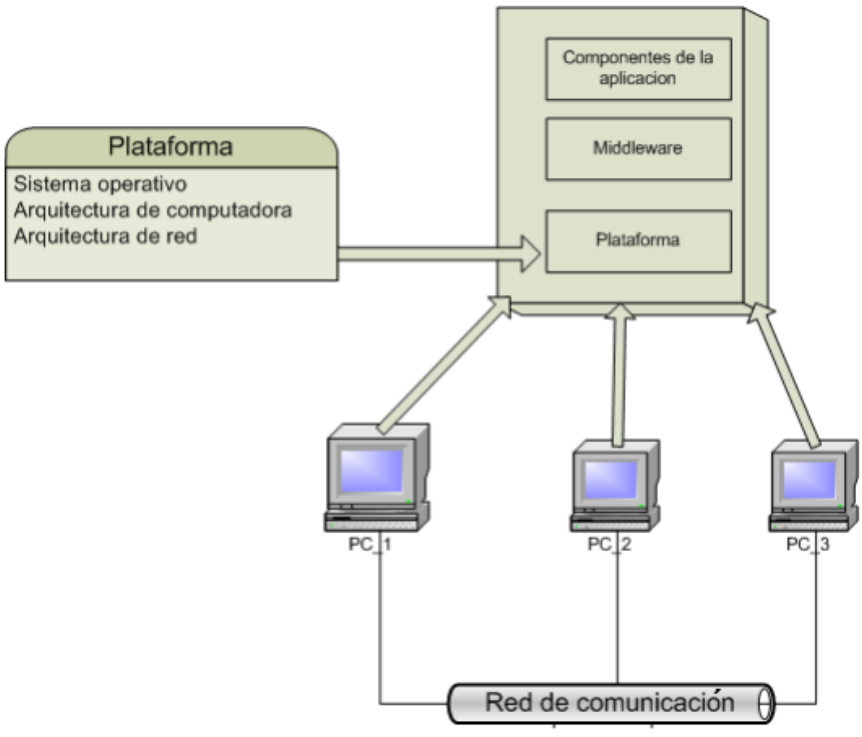

Figura 4. *Arquitectura general de un middleware*

2.3.1 CORBA

CORBA (Common Object Request Brocker Architecture) es la tecnología que propone OMG [19] para:

- Constituir la base de la tecnología emergente DOM (Distributed Object Management)
- Facilitar el diseño de aplicaciones basadas en el modelo Cliente/servidor.
- Facilitar la integración de sistemas legados (Off-the-Shell).

CORBA es un Middeware o marco de trabajo estándar y abierto de objetos distribuidos que permite a los componentes en la red interoperar en un ambiente común sin importar el lenguaje de desarrollo, sistema operacional, tipo de red, etc. En esta arquitectura, los métodos de un objeto remoto pueden ser invocados “transparentemente” en un ambiente distribuido y heterogéneo a través de un ORB (Object Request Broker). Además del objetivo básico de ejecutar simplemente métodos en objetos remotos, CORBA adiciona un conjunto de servicios que amplían las potencialidades de estos objetos y conforman una infraestructura sólida para el desarrollo de aplicaciones críticas de negocio [20].

Con CORBA se facilita el diseño de aplicaciones en plataformas heterogéneas sin necesidad de conocer los detalles de los recursos y servicios que ofrece cada elemento de la plataforma, la capacidad de diseñar aplicaciones desarrolladas en diferentes lenguajes de programación, supliendo los recursos necesarios para implementar las interfaces entre ellas y la interoperabilidad entre aplicaciones desarrolladas por diferentes fabricantes, para que un componente sea interoperable sólo se requiere que ofrezcan las interfaces y los patrones de interacción basados en la especificación CORBA [21].

La especificación CORBA está basada en tres conceptos fundamentales [21]:

- Modelo orientado a objetos.
- Entorno de computación distribuido y abierto.
- Integración y reutilización de componentes.

2.3.2 Java RMI

Java RMI [22] (*Java Remote Method Invocation*) es un mecanismo ofrecido por Java para invocar un método de manera remota. Forma parte del entorno estándar de ejecución de Java y proporciona un mecanismo simple para la comunicación de servidores en aplicaciones distribuidas basadas exclusivamente en Java. Si se requiere comunicación entre otras tecnologías debe utilizarse CORBA o SOAP en lugar de RMI.

RMI se caracteriza por la facilidad de su uso en la programación por estar específicamente diseñado para Java; proporciona paso de objetos por referencia (no permitido por SOAP), recolección de basura distribuida y paso de tipos arbitrarios (funcionalidad no provista por CORBA). A través de RMI, un programa Java puede exportar un objeto, con lo que dicho objeto estará accesible a través de la red y el programa permanece a la espera de peticiones en un puerto TCP. A partir de ese momento, un cliente puede conectarse e invocar los métodos proporcionados por el objeto.

2.3.3 Middleware para sistemas distribuidos de tiempo real

Cuando se trata de sistemas distribuidos con requisitos de tiempo real, CORBA no constituye una solución del todo adecuada, al no estar preparado para responder a las necesidades de mínimo retardo y determinismo que dichos sistemas requieren. Como respuesta a esta necesidad, en junio de 2004 la OMG finaliza la especificación de DDS para sistemas de tiempo real. En dicha especificación, se unificaron las mejores prácticas presentes en los middlewares de distribución de datos en tiempo real desarrollados de forma independiente por las empresas RTI y Thales. Este hecho fue definido por aquel entonces como “el avance más significativo para sistemas de tiempo real llevado a cabo por la OMG en los últimos años”. Aunque inicialmente sólo existían las versiones de RTI y Thales, actualmente son numerosas las implementaciones del estándar DDS de la OMG, algunas de ellas de código abierto. Entre ellas, destaca RTI Data Distribution Service, una implementación comercial de la capa DCPS (Data-Centric Publish-Subscribe) del estándar DDS que soporta múltiples arquitecturas (incluyendo entre ellas varios sistemas empotrados) y múltiples lenguajes (C, C++, C#, Java…) [14].

2.3.3.1 Sistemas de tiempo real

Un sistema de tiempo real es aquél en el que la corrección del sistema depende tanto del valor lógico de los resultados, como del instante de tiempo en el que se producen. Dentro de los sistemas de tiempo real podemos distinguir diversos sistemas, los cuales pueden ser clasificados en tres tipos diferentes

dependiendo de las causas que producen no respetar los requisitos de tiempo [16]:

- **Críticos:** Un incumplimiento puede provocar daños en seres humanos.
- **Firmes:** Una respuesta fuera de plazo es inútil pero el sistema puede seguir funcionando, aunque la calidad de servicio se vea degradada.
- **Acríticos:** una respuesta fuera de tiempo tiene una validez relativa, pero el sistema sigue operativo.

2.3.3.2 DDS

El servicio de distribución de datos para sistemas de tiempo real (DDS) es el primer estándar middleware internacional recientemente adoptado por la OMG [23].

Esta norma ha experimentado una adopción record de espacio dentro de la aeronáutica y dominio de defensa y está expandiendo con rapidez a los nuevos dominios, tales como transporte, servicios financieros y SCADA. La especificación OMG DDS ha sido diseñada para apoyar con eficacia los modelos de datos definidos estáticamente. Este modelo mostrado en la figura 5 requiere que los tipos de datos utilizados por temas DDS sean conocidos en tiempo de compilación y que todos los miembros del espacio de datos globales DDS están de acuerdo precisamente en la misma asociación de tipo tema [24].

El DDS está adoptado para la conectividad de datos céntricos. Se integra de componentes de un sistema en conjunto, proporcionando conectividad de baja latencia de datos, fiabilidad extrema, y una arquitectura escalable, las cuales negocios y aplicaciones de internet de las cosas necesitan [25]. Los vendedores de esta tecnología se presentan en la Tabla 2.

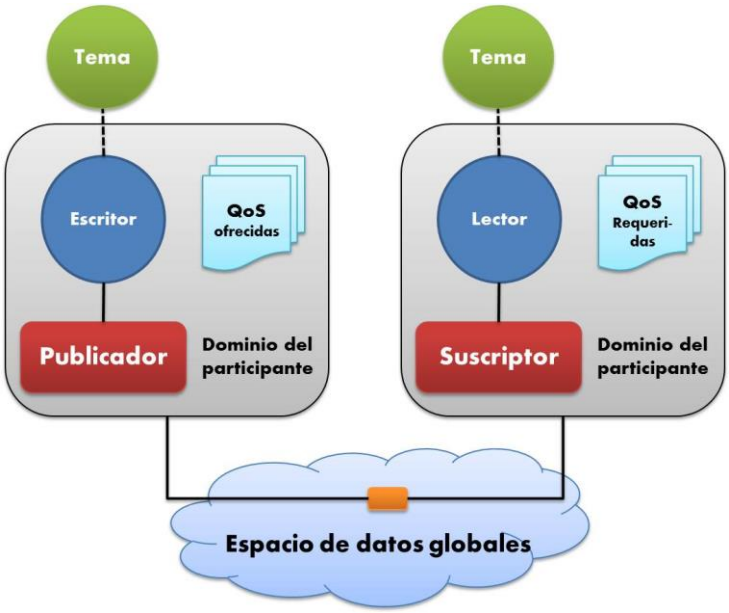

Figura 5. *Modelo de datos DDS*

Tabla 2. *Principales distribuidores de productos DDS*

Empresas	Producto
Kongsberg Gallium	InterCOM DDS
MilSOFT	MilDDS
Object Computing	OpenDDS
Thales PrismTech	SpliceDDS, Vortex
RTI	RTI Connext® DDS
Twin Oaks Computing	CoreDX DDS

2.3.4 Modelos de comunicación

Para que un sistema SCADA sea capaz de interconectar sus componentes de hardware, software y el middleware antes mencionados y transmitir datos entre ellos, es necesario conocer el modelo de comunicación que se va a utilizar.

Los modelos generales de comunicación que existen son paso de mensajes, cliente/servidor y sistemas de mensajes para los middlewares [26].

- **Paso de mensajes.** Es el modelo más sencillo de comunicación a nivel de procesos. Consiste en que dos procesos se comuniquen con mensajes básicos entre ellos. Un proceso envía una solicitud a otro que, al recibirla, puede generar una respuesta. Para este tipo de comunicación, sólo son necesarias dos operaciones de gestión de la comunicación: conexión y desconexión, y dos operaciones de transferencia de la información: enviar y recibir. La programación basada en sockets está basada en este paradigma.

- **Cliente-Servidor.** El modelo cliente-servidor se basa en la asignación de dos funcionalidades diferentes a cada uno de los componentes que se comunican. El proceso servidor proporciona una serie de servicios y espera la llegada de peticiones por parte de los clientes. En este modelo, el servidor tiene un papel pasivo, dedicándose a esperar las solicitudes de los clientes, que son los que toman el papel activo. Los paradigmas de comunicación derivados de este modelo son llamados a servicio, llamada a procedimiento remoto, objetos distribuidos, componentes distribuidos y servicios distribuidos.

- **Sistema de mensajes.** Los sistemas de mensajes, constituyen un modelo basado en una capa intermedia (middleware) que gestiona el paso de mensajes entre componentes. Por ello, se suele hablar de una capa intermedia del modelo de paso de mensajes. El nombre más conocido de este modelo es “Message-Oriented Middleware (MOM)”. Existen dos modelos de implementación de este paradigma: punto a punto (peer to peer) y publicación/suscripción (publish/subscribe).

2.3.4.1 Paradigmas de comunicación

Como se observó en el tema anterior los modelos están relacionados con los paradigmas de comunicación. En la Figura 6, se puede observar la clasificación de los modelos y paradigmas de comunicación. El paradigma más importante es el de las comunicaciones únicas de datos, donde se pueden ubicar prácticamente todos los modelos y paradigmas clásicos de comunicación. La transferencia de código precisa del soporte de comunicaciones de los paradigmas anteriores, y adicionalmente la infraestructura necesaria para la ejecución del código. Como se puede observar, en la transmisión de datos, hay tres grupos importantes, en función del papel que toman cada uno de los componentes de la comunicación. Estos son, de menor a mayor complejidad, los pasos de mensajes, el modelo cliente-servidor y los sistemas de mensajería. Algunos middlewares existentes se ven

clasificados en la imagen de acuerdo al paradigma de comunicación que utilizan, éstos se encuentran entre paréntesis.

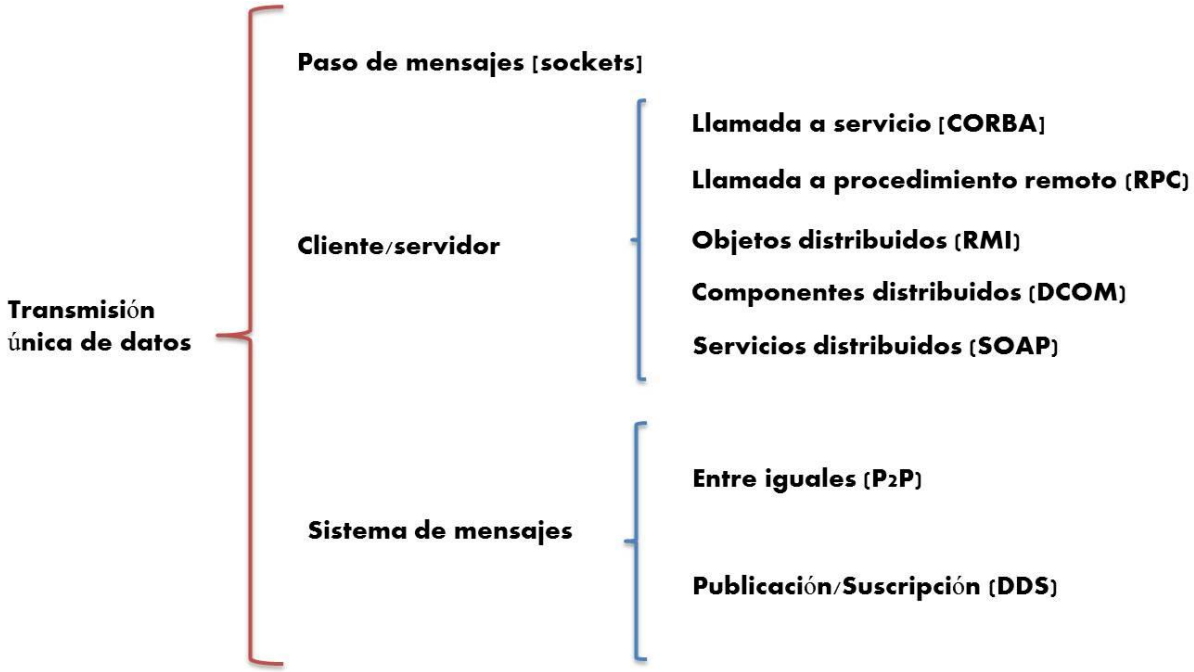

Figura 6. *Modelos y paradigmas de comunicación*

2.4 Dispositivos médicos utilizados

2.4.1 Oxímetro de pulso

Es uno de los dispositivos que se ocupan en este sistema y obtiene la saturación de oxígeno en la sangre. Al medir la saturación de oxígeno estamos midiendo la cantidad de oxígeno que se encuentra combinado con la hemoglobina, es por eso que esta medida es una medida relativa y no absoluta ya que no indica la cantidad de oxígeno en sangre que llega a los tejidos, sino, qué relación hay entre la cantidad de hemoglobina presente y la cantidad de hemoglobina combinada con oxígeno (oxihemoglobina). Este parámetro se puede, y usualmente se mide óptimamente, dado que la cantidad de oxihemoglobina está relacionada con la coloración roja de la sangre, siendo esta más fuerte cuando más oxihemoglobina contiene la sangre y más tenue cuando menos oxihemoglobina hay presente.

Los valores típicos de la saturación de oxigeno (SpO_2) andan entre 95% y 97% con un rango de variación del 2%. Valores por debajo del 90% se asocian con situaciones patológicas e insuficiencia respiratoria.

2.4.1.1 Principio de funcionamiento del sensor

El principio de funcionamiento del sensor óptico se basa en el hecho de que la absorción de la sangre a una cierta longitud de onda es dependiente de la saturación de oxihemoglobina. Entonces refiriéndose al sensor dactilar, emitiendo una luz a esta longitud de onda a través del dedo y recibiendo la cantidad de luz que no fue absorbida en un receptor diametralmente opuesto al emisor, se puede conocer la cantidad de luz absorbida por el dedo, que es mayoritariamente absorbida por la sangre.

Una vez llegado a este punto se presenta un problema, la sangre y por tanto la SpO2 es pulsátil, por ende, al ser variable esta, no se puede determinar a priori si la variación de la medida es debido a una variación de la variable misma o debido a la pulsatilidad del flujo sanguíneo. Es por esta razón que estos sensores contienen en realidad dos emisores a dos longitudes de diferentes y un receptor como se puede ver en la figura 1, de manera que a una de las longitudes de onda la absorción es muy dependiente de la SpO2, y a la otra longitud de onda la absorción teóricamente no varía con la SpO2 pero si con la cantidad de sangre, es decir, varía con el pulso.

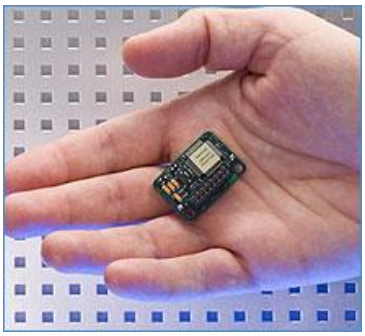

Figura 7. *Módulo OEM de NONIN*

En este caso se hace uso del módulo de medición oximetría NONIN III OEM mostrado en la figura 7, el cual realiza la medición de la saturación de oxígeno, entregando las mediciones realizadas en binario de la siguiente forma:

El modulo OEM envía 3 paquetes de 25 frames (cada frame consta de 5 bytes) cada segundo, por lo que envía 75 frames por segundo (375 bytes por segundo). De cada frame se usa el byte 2 (estatus) y el byte 3 (pletismografia) de los cuales se tienen 75 por segundo.

Además, en los primeros frames de cada paquete se incluye información acerca de la frecuencia cardiaca (HR) y de la saturación de oxigeno (SpO_2), por lo que estos valores se obtienen únicamente 3 veces por segundo.

Esta información es obtenida de [27].

2.4.2 Monitor de signos vitales

Es un dispositivo que permite detectar, procesar y desplegar en forma continua los parámetros fisiológicos del paciente, algunos de estos parámetros corresponden al electrocardiograma (ECG), frecuencia respiratoria, presión no invasiva (PNI), presión invasiva (PI), temperatura corporal, saturación de oxígeno (SpO2), saturación venosa de oxígeno (SvO2), gasto cardiaco, dióxido de carbono (CO2), presión intracraneana (PIC), entre otros.

En este caso no se tuvo la oportunidad de utilizar uno real, pero se realizó una simulación en una tarjeta núcleo F446, con parámetros oficiales de un monitor de signos vitales obtenidos de [28].

2.5 Trabajos relacionados

En el presente trabajo se revisó en la literatura de las posibles arquitecturas implementadas en un ICE. Para comprender este término, una implementación-ICE compatible puede ser vista como una plataforma informática (arquitectura, hardware y servicios de software) que permite que los dispositivos médicos sean heterogéneos integrados para crear sistemas médicos para el cuidado de un solo paciente en ambientes de alta agudeza y criticidad [29].

Obteniendo de la exploración dos plataformas de investigación clínica; OpenICE y OpenSDC que están en desarrollo y que a continuación se describe cada una de ellas.

2.5.1 OpenICE

OpenICE [2] es una plataforma de investigación clínica abierta para sistemas distribuidos que permite la colaboración de personas interesadas en contribuir a la aplicación actual, fue propuesta por el CIMIT a partir del programa de interoperabilidad MD PnP establecido en el 2004 [4]. Mediante este programa se desarrolló el estándar ICE (Ambiente Clínico Integrado por sus siglas en inglés) del cual la primera parte fue publicada bajo el nombre de ASTM F2761-09 [29] donde describe los componentes que son requeridos para hacer una implementación ICE.

El estándar ICE proporciona servicios que exponen datos y aspectos de control de dispositivos integrados a una aplicación supervisor ICE. Los dispositivos médicos individuales, el suministro de datos y la administración de tratamiento están conectadas a un sustrato de red compartida, gestionada por la red del controlador ICE, esto se realiza para aumentar la efectividad de la operación “Plug and Play” de un ICE en ambientes de alta agudeza, así como el modo en que la interoperabilidad puede permitir la integración de dispositivos para crear nuevos sistemas de dispositivos médicos con mayores capacidades

de seguridad y rendimiento que cualquier dispositivo individual; en la figura 8 se muestra la arquitectura de este estándar la cual consiste de la implementación de varios sistemas ICE que pueden ser enlazados al registro médico electrónico (EMR) o a la central de servicios de alarma (ADT), esto con la finalidad de darle más seguridad al paciente. La plataforma OpenICE está relacionada con múltiples estándares [30] tales como la familia IEEE-11073 [31] de la cual se deriva la IEEE-11073-10101 donde obtienen de esta toda la terminología utilizada para las variables sensadas por los dispositivos médicos, la IEEE-11073-10201 la cual rige el modelo del dominio de información y la IEEE-11073-20601 que define la comunicación del dispositivo de salud personal. Entre otros estándares se encuentran el HL7 y EHR que son un conjunto de estándares que facilitan el intercambio electrónico de información clínica y el estándar internacional DDS que es propuesto por la OMG para sistemas de tiempo real.

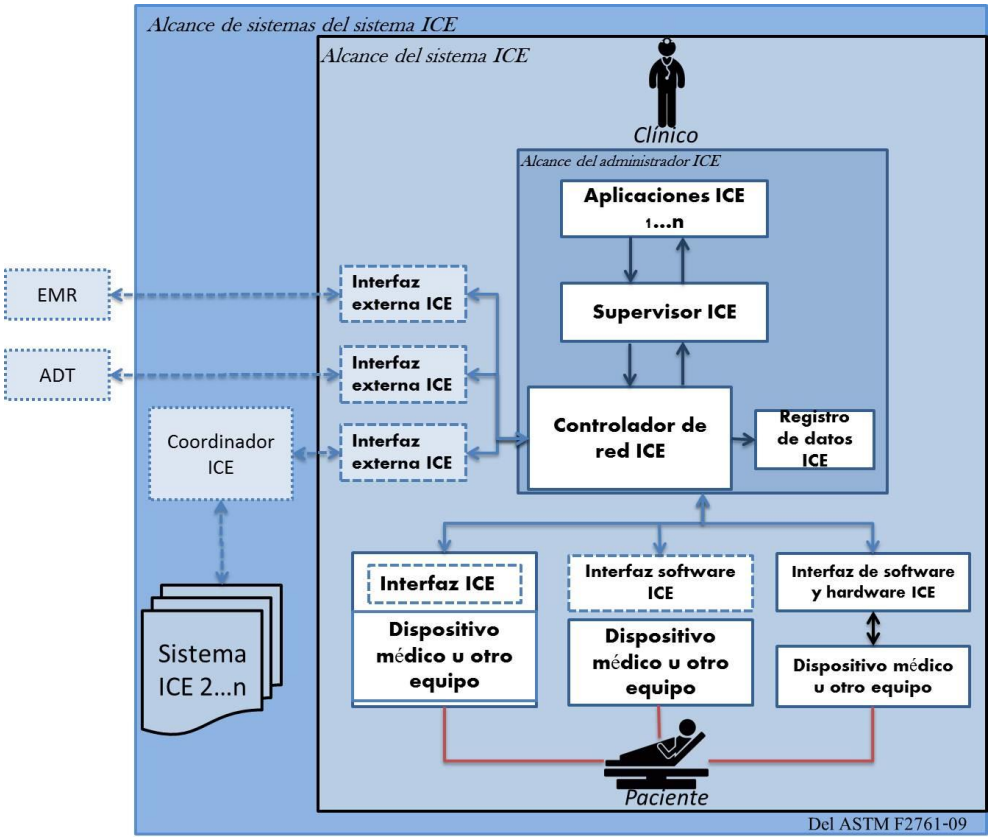

Figura 8. *Arquitectura propuesta por el estándar ASTM F2761*

El paradigma en el que se basa OpenICE es publicador/suscriptor, para facilitar todas las comunicaciones entre los dispositivos médicos y aplicaciones. OpenICE integra los dispositivos médicos y las aplicaciones clínicas de un ecosistema de salud existente [32]; esto significa que es el punto de entrada y salida de los datos de los dispositivos médicos dentro y fuera de una red. OpenICE tiene dos modos de funcionamiento: El supervisor que se ejecuta en una PC y suscribe a todos los dispositivos médicos en el mismo dominio y

contexto del paciente y el software adaptador-dispositivo el cual puede ser instalado en sistemas embebidos, laptops, Raspberry Pis, Intel Galileos y Beagleboards; el cual actúa como un puente entre los dispositivos médicos y OpenICE, traduciendo los propios protocolos de comunicación del dispositivo en las estructuras de datos estándar de OpenICE y el protocolo de comunicación.

Los dispositivos que actualmente son compatibles con OpenICE son [31]:

- Philips MPxx Intellivue - Ethernet Connection/ Serial Connection
- Masimo Radical 7
- Dräger Apollo
- Dräger Evita XL
- Dräger Evita 4
- Dräger V500
- Ivy 450C
- Nonin Bluetooh
- Capnostream 20
- Nellcor N595
- Bernoulli

Los Simuladores de dispositivos incluidos en la plataforma son:

- Monitor de paciente multiparámetros
- Presión arterial no invasiva
- Electrocardiograma
- End Tidal CO2
- Temperatura
- Bomba de infusión
- Oxímetro de pulso

Esto quiere decir que ofrecen unos dispositivos simulados para que personas que son principiantes en temas de comunicación en un ambiente clínico integrado (ICE) y sino tiene a la mano dispositivos médicos de las marcas antes mencionados, puedan correr los softwares como pruebas para investigaciones futuras.

Resultados: Los resultados que tienen obtenidos los miembros del CIMIT en el desarrollo de esta plataforma quedan dados en la interconexión de los 11 dispositivos antes mencionados a través de una tarjeta BeagleBoard la que actúa como adaptador del dispositivo y que están conectados a una red WAN mediante la cual se transmiten datos bajo un mismo dominio determinado por el software supervisor el cual corre en una PC y que contiene los datos de los dispositivos conectados a la red. La plataforma sigue en desarrollo por que esperan que pueda ser escalable, característica que aún no se logra.

2.5.2 OpenSDC

OpenSDC [3] es una plataforma que se ha desarrollado en un proyecto interno de tecnología en la empresa Dräger denominado “Device & System Connectivity“ (Conectividad de sistemas y dispositivos), que tenía el objetivo de satisfacer la creciente demanda de interoperabilidad de dispositivos médicos en un ambiente clínico integrado; mientras el objetivo del proyecto era desarrollar una arquitectura eficiente, una pila de protocolos y middleware que satisfaga los requerimientos derivados para facilitar la implementación del concepto ICE y posteriormente ser adoptada por el comité de estándares IEEE 11073 [31] para su inclusión [4].

OpenSDC abarca la arquitectura de dispositivos médicos orientados al servicio (SOMDA) en un lugar de trabajo clínico a través de servicios web mostrado en la figura 9, SOMDA transfiere el concepto de arquitectura orientado al servicio (SOA) para el dominio de sistemas distribuidos de dispositivos médicos [34].

Figura 9. *Arquitectura SOMDA*

La plataforma OpenSDC expone un pequeño conjunto de servicios estándar (por ejemplo, obtener, ajuste / acción, informe de eventos, formas de onda, PHI (información médica protegida)), junto con un predefinido conjunto de operaciones, esta plataforma es abierta y es utilizable para la evaluación de los protocolos propuestos en el proyecto de dräger, sólo fue una implementación de referencia para introducir los protocolos [35].

Esta plataforma se basa en el protocolo DPWS como protocolo de transporte fundamental, un modelo de mensajes descrito en IEEE 11073-DIM y las nomenclaturas las obtiene de IEEE 11073-10101, para el registro futuro de datos del paciente contempla los protocolos HL7 y EHR. OpenSDC propone una pila de protocolos para el procesamiento de datos mostrados en la figura 10

en los cuales se encuentra BICEPS el cual se plantea como un estándar adjunto en la nomenclatura IEEE 11073-10207, a su vez otro protocolo llamado SDC equivalente al protocolo de aplicación inteligente correspondiendo a IEEE 11073-20701, por último, el denominado MDPWS y un protocolo interno a este llamado JDPWS ambos estarán adjuntos en IEEE 11073-20702 [4].

Resultados: Implementaron su propio middleware BICEPS que trabaja conjuntamente con MDPWS, también realizaron un escenario de prueba para sus protocolos obteniendo una conexión ethernet directa de un cliente (PC) y una raspberry pi simulando ser un dispositivo médico, no hay fundamentos de que hayan trabajado con dispositivos reales, han utilizado las librerías de OpenSDC que son para la comunicación y que facilitan el desarrollo de sistemas distribuidos de dispositivos médicos en ambientes de alta agudeza.

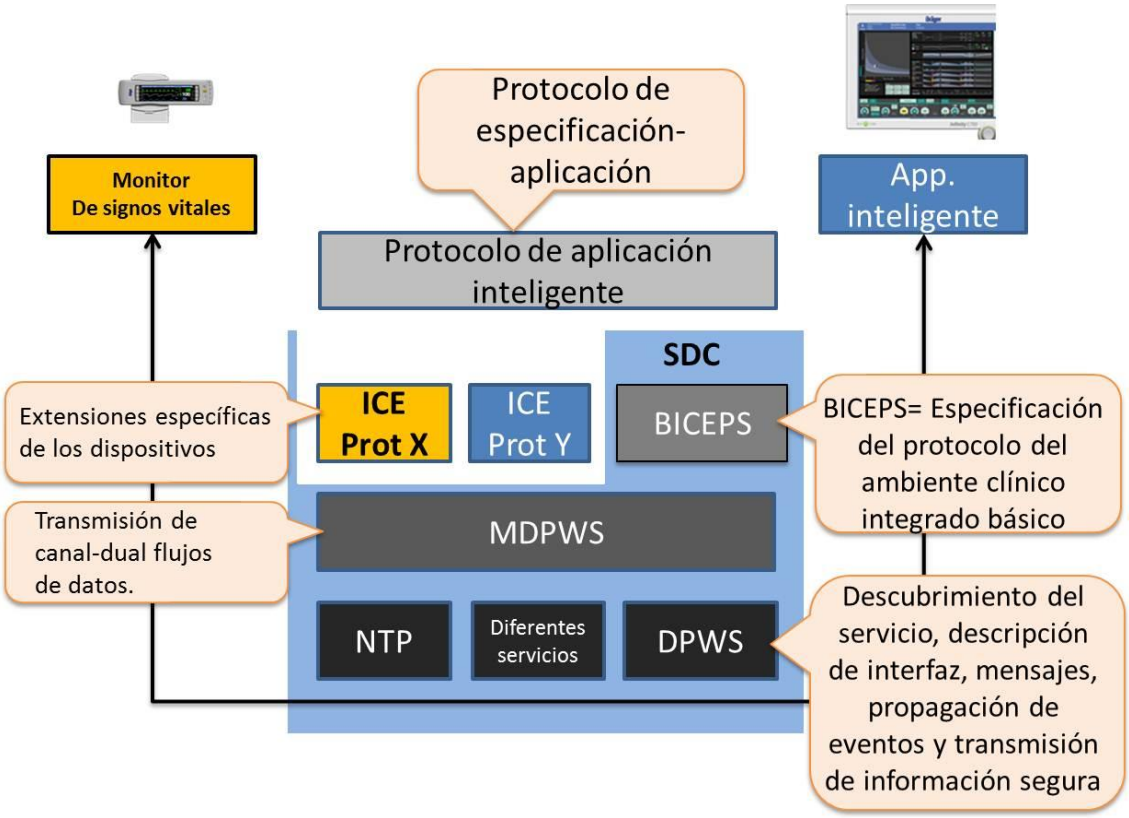

Figura 10. *Pila de protocolos propuestos por OpenSDC*

2.5.3 Resultado

Teniendo estudiadas las plataformas anteriores obtenemos la siguiente tabla comparativa (Tabla 3) donde se resumen las características principales de openICE y openSDC, habiéndose cotejado en las referencias correspondientes que es lo que contiene cada una y en que se basan.

Tabla 3. *Características de OpenICE y OpenSDC*

Características	OpenICE	OpenSDC
Arquitectura	ASTM F2761-09	SOMDA
Estándar de nomenclatura	IEEE11073-10101	IEEE 11073-10101
Protocolos propuestos	_____	BICEPS;SDC;MDPWS
Protocolos de seguridad de información	HL7/EHR	HL7/EHR
Año publicado	2013	2014
Tarjeta de desarrollo utilizada	BeagleBone Black	Raspberry Pi
Núm. De dispositivos agregados	11	No especificado
Red	Ethernet	Ethernet
Protocolo de interfaz	IEEE 802.3	IEEE 802.3
Middleware	DDS	BICEPS
Modelo del dominio de información de los dispositivos	IEEE 11073-10201	IEEE 11073-10201
Paradigma de comunicación	Publicador/suscriptor	Servicios distribuidos
Sistema operativo	Linux Debian	Linux Debian
Compatibilidad de ejecución	Windows	Windows
Repositorio	GitHub	Sourceforge
Lenguaje de programación	Java	Java

Organización	CIMIT	Dräger
Proyecto en el que fueron realizadas	MD PnP	Conectividad de sistemas y dispositivos
Conectividad	UDP/IP&TCP/IP V4	UDP/IP&TCP/IP V4
Protocolo o software que integra QoS	RTI connext DDS	MDPWS
Lenguaje de descripción de servicios web	XML	XML
Propósito de implementar la plataforma	Implementar una arquitectura ICE	Verificar la funcionalidad de los protocolos propuestos

Observando las características correspondientes a cada plataforma se pudo seleccionar la plataforma openICE como base de este tema de tesis, ya que su desarrollo fue realizado con la implementación del estándar DDS por la empresa RTI y también consideraron los estándares de la familia IEEE-1073, también comparten información sobre su desarrollo y permiten aportaciones por parte de los desarrolladores.

Capítulo 3 Modelado del sistema

3.1 Introducción

El modelado de software con SysML permite crear un prototipo fácil de seguir y muy práctico a la hora de planificar, diseñar y documentar el sistema que se está desarrollando. Este sistema puede incluir hardware, software, información, procesos, personal e instalaciones.

En el presente capítulo se muestran los diagramas realizados correspondientes al modelado del sistema, se encuentran divididos en dos etapas la de análisis y posteriormente la de diseño. La primera comprende los diagramas de requerimientos que son producto del análisis de requisitos, posteriormente los diagramas de casos de usos para explicar que uso tienen las partes que integran el sistema y que son producto de los escenarios que se plantean. Más adelante se presentan los diagramas correspondientes al diseño del sistema donde interviene el diagrama de paquetes que dan una vista general de la composición del modelo del sistema, seguido se encuentra el diagrama de bloques que muestra las piezas que integran al sistema, consecutivamente tendremos los diagramas de secuencia que explican las interacciones entre objetos del sistema y por último es presentado el diagrama de clases que proporciona una interpretación y comprensión de las clases, métodos y objetos del sistema. Todos los diagramas planteados en este capítulo fueron realizados en la herramienta Enterprise Architect y bajo el lenguaje de SysML.

3.2 Análisis

En este apartado se analizan los requerimientos del sistema con respecto a la propuesta de solución al problema que se planteó en el capítulo uno, representando en diagramas estos requisitos y posteriormente se presentan los escenarios que intervienen en el funcionamiento del sistema y desglosando el diagrama de casos de uso correspondiente a las actividades del sistema.

3.2.1 Requerimientos del sistema

Los diagramas de requerimientos son clasificados de acuerdo a los requisitos que se presentaron. Estos diagramas son unos de los más importantes para el modelado de un sistema, ya que en éstos se definen de manera objetiva los requisitos que se deben cubrir al diseñar el sistema y algunas restricciones que se deben tomar en cuenta, se dan algunas relaciones entre requerimientos en los que se especifica si un requisito depende de otro, se deriva (deriveReqt), si lo extiende (refine) o simplemente para que se cumpla un requisito se debe satisfacer otro (satisfy), o si un requisito sirve para comprobar la efectividad de otro (verify); todas estas relaciones entre requisitos son muy importantes para poder entender este diagrama y verificar que si se estén atendiendo cada uno de ellos. Por ejemplo, para lograr el requerimiento de la conexión plug and play se debe satisfacer un requerimiento que es utilizar estándares de la IEEE 1073.

A continuación, se exponen los diagramas de requerimientos realizados los cuales se dividen en requerimientos por facilidad de uso (figura 11), de certificación (figura 12) y el de funcionalidad (figura 13).

En el diagrama de requerimientos por facilidad de uso se clasificaron los siguientes requisitos: interfaz única ya que se requiere visualizar los gráficos de tres dispositivos médicos en una sola pantalla, escalabilidad se debe cumplir ya que es una de las garantías del estándar DDS y extensibilidad cuya característica del sistema hace que sea factible incorporar nuevas aplicaciones. Estos requisitos se verifican mediante casos de prueba los cuales se visualizan en la imagen de color amarillo.

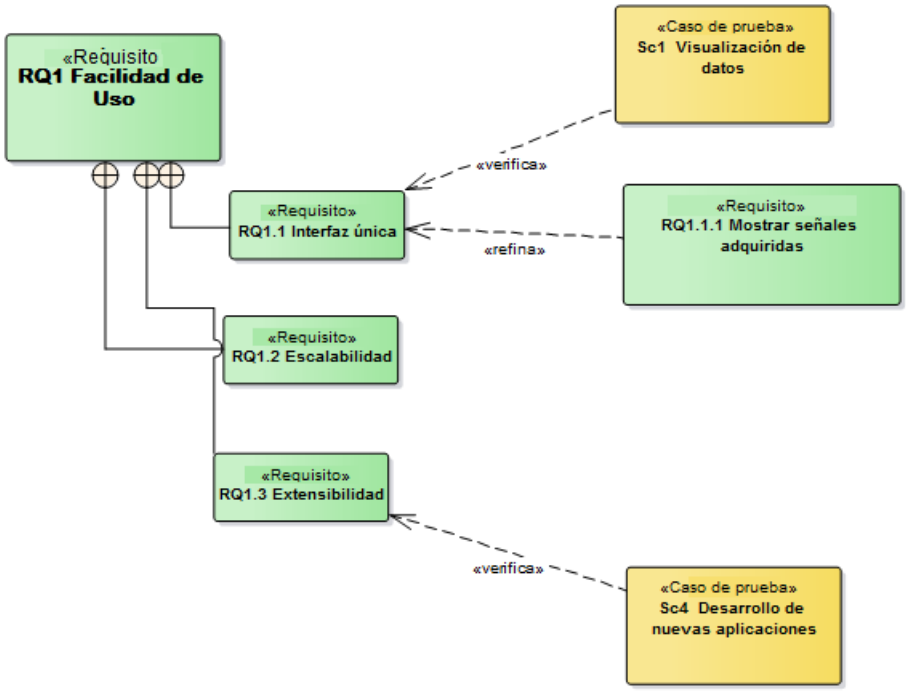

Figura 11. *Diagrama de requerimientos (facilidad de uso)*

En el diagrama de requerimientos de la figura 12 se contemplan los requisitos necesarios para que este sistema siga la ruta de la certificación, por ende, debe cumplir ciertos estándares y normas que permitan documentar este sistema, iniciando por la norma que rige el desarrollo de sistemas ocupados en el área médica (IEC-62304) y posteriormente implementando algunos estándares de la familia IEEE-11073, seguido del ASTMF-2761 el cuál describe la plataforma de un ambiente clínico integrado. Por último, se debe exponer un sistema adaptable al ambiente médico a futuro. La verificación de este diagrama será dada por la liberación del sistema.

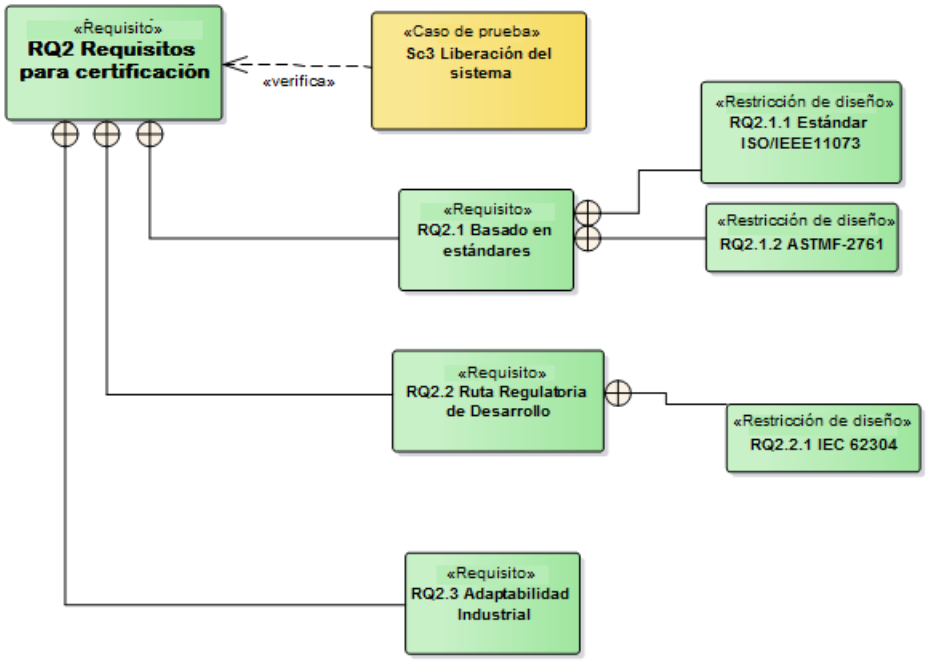

Figura 12. *Diagrama de requerimientos de certificación*

El diagrama de la figura 13 expone los requerimientos funcionales del sistema de los cuales son conexión plug and play ya que el sistema debe detectar la disponibilidad de un nuevo dispositivo el cual debe ocupar los puertos RJ45 y el serial, se debe considerar cierta seguridad lo cual se toma en cuenta en la inclusión de políticas de calidad (QoS), se pretende que el sistema sea tolerante a fallas esto es una propiedad para que el sistema permanezca funcionando a pesar de que un nodo falle; otro de los requerimientos es que se puedan visualizar las alarma y el registro de actividades que se cumplen en la implementación del middleware.

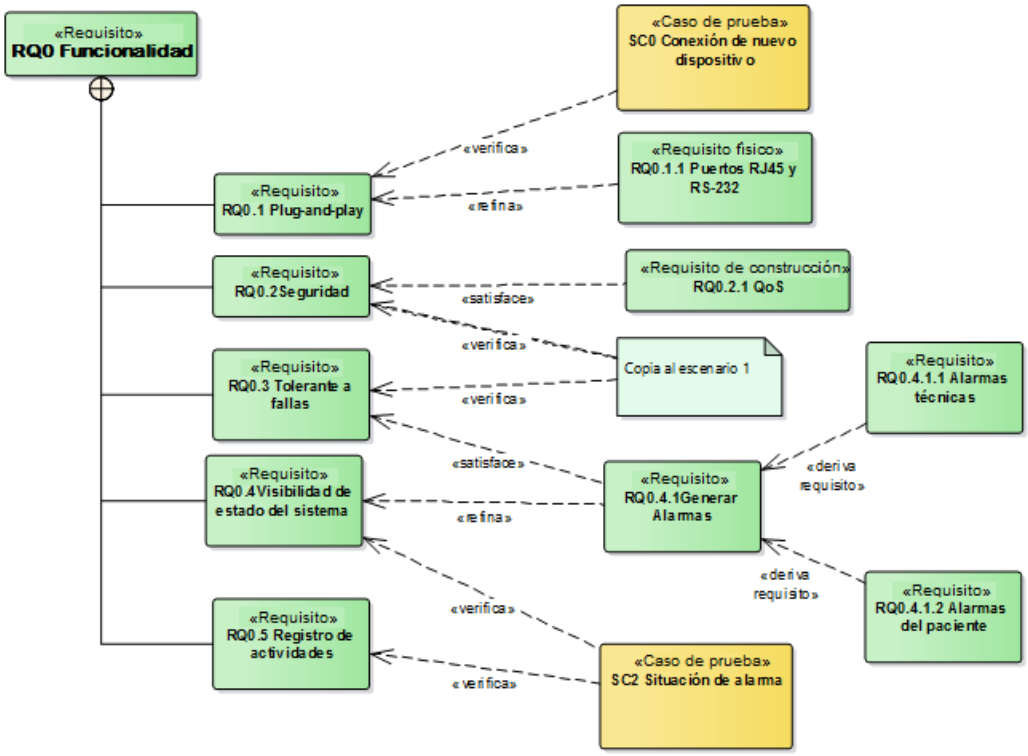

Figura 13. *Diagrama de requerimientos de funcionalidad*

3.2.2 Escenarios y casos de uso

En modelado de sistemas un escenario es una circunstancia o situación en las que se puede encontrar un sistema, en este documento se analizan cuatro escenarios que conciernen al sistema que se está realizando y son presentados a continuación, cabe señalar que estos escenarios tienen relación directa con el diagrama de casos de uso.

Escenario 1: “Interactuar con aplicación”

Se presenta el individuo a operar el sistema y se encuentra con una aplicación la cual contendrá algunos objetos de interacción como por ejemplo activar la recepción de datos de un dispositivo médico.

Escenario 2: “Transferencia de datos”

Interviene el middleware que se utiliza para hacer el procesamiento de datos.

Escenario 3: “Situación de alarma”

Este escenario se presenta cuando se obtienen medidas fuera del rango normal o el dispositivo se desconecta, y se crean eventos en el middleware.

Escenario 4: “Estableciendo QoS”

El sistema debe contar con las políticas QoS adecuadas para que exista la transmisión de datos en los tiempos adecuados. De no ser así la comunicación se ve afectada y el middleware es el encargado de notificar estos eventos.

3.2.2.1 Diagrama de caso de uso

Los diagramas de caso de uso nos ayudan a tener una idea clara de lo que pueden hacer nuestras aplicaciones y descartar lo que no.

En este caso se obtuvo el diagrama de caso de uso desde la funcionalidad de los usuarios de nuestro sistema hasta la comunicación que se realiza en el middleware ocupado.

En la figura 14 se muestra el diagrama de caso de uso, en éste se presentan cuatro actores, los cuales se tomaron así debido a que son los elementos importantes en la interacción del sistema. El primer actor es el usuario del sistema el cual podrá interactuar con la interfaz, el siguiente actor es el dispositivo médico, el cual contendrá la aplicación “adaptador”, la cual será encargada de publicar los datos en red. Por último, se presentan los actores publicador/suscriptor, estos nos reflejan la funcionalidad del estándar DDS el cual es un middleware que proporciona las conexiones necesarias entre estas aplicaciones. Cada uno de los actores adjuntan las actividades que corresponden a la función que tienen dentro del sistema.

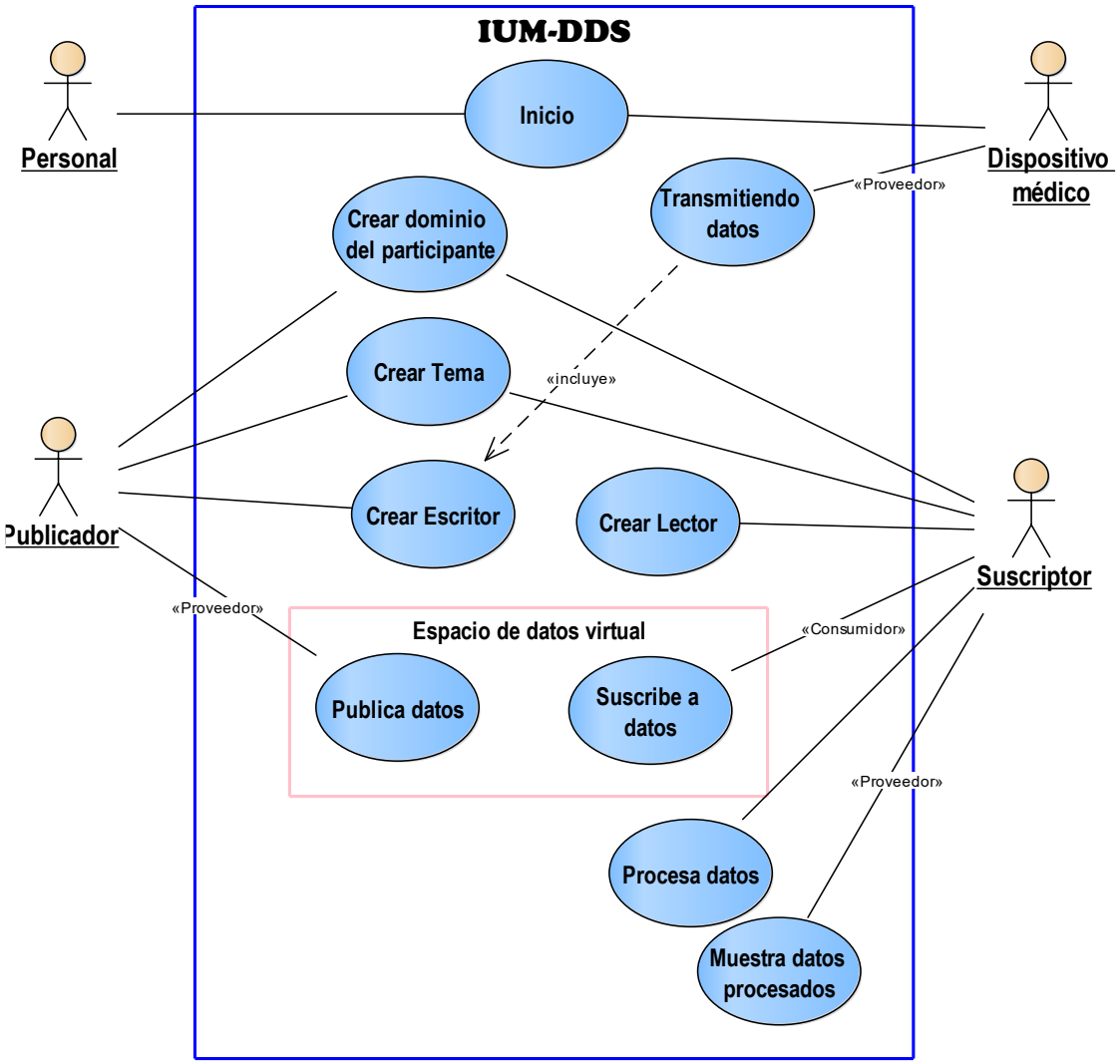

Figura 14. *Diagrama de caso de uso*

3.3 Diseño del sistema

En el diseño del sistema se describen, organizan y estructuran los componentes del sistema, tomando en cuenta los requerimientos de las funcionalidades definidas en la fase de análisis (entrada, procesamiento, salida y control de eventos) y los expone en múltiples diagramas que serán mostrados en los siguientes apartados, los cuales incluyen la funcionalidad del sistema, el hardware utilizado y la plataforma de software del sistema.

3.3.1 Diagrama de paquetes

Primero se mostrará la vista general del sistema a través del diagrama de paquetes (figura 15), el modelo de paquetes permite agrupar los elementos que integran todo el sistema y crea una estructura jerárquica de los mismos. En este diagrama se observan los modelos llamados de requerimientos en este se encuentran también los diagramas de casos de uso, el modelo del dominio operacional el cual incluye los diagramas de secuencia, el modelo de diseño y por último el modelo de implementación de clases que integra el diagrama de clases.

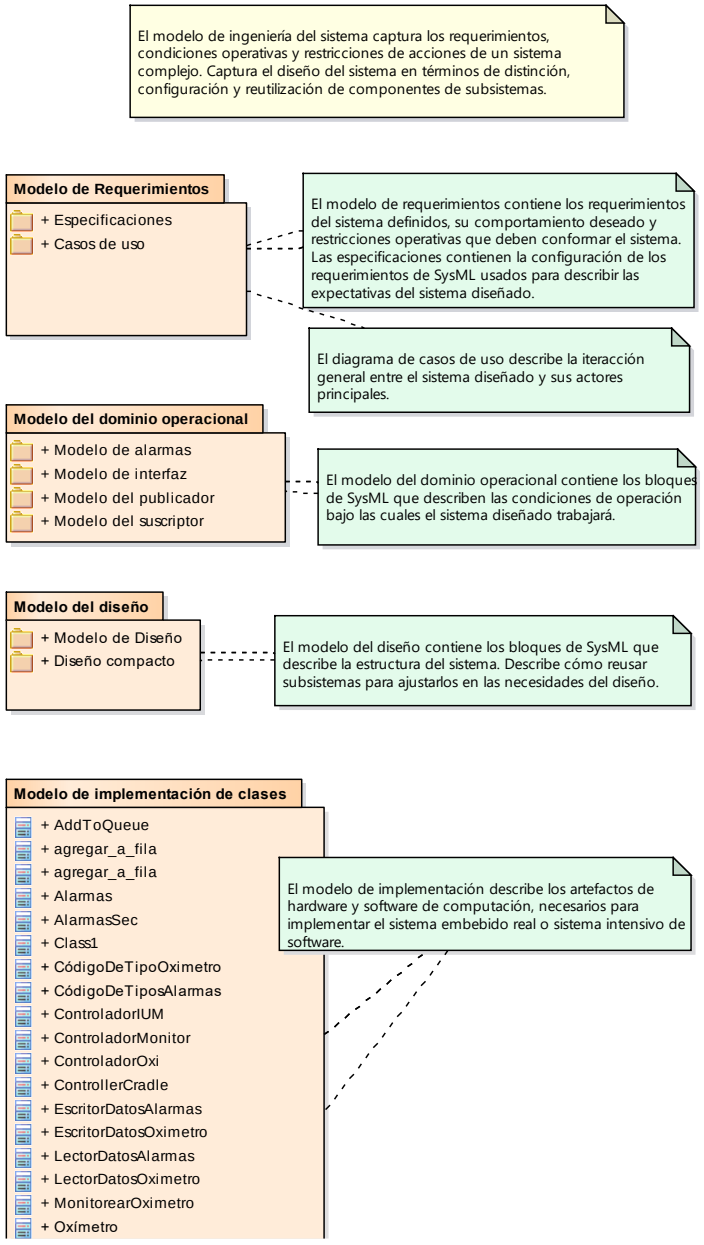

Figura 15. *Organización del modelo del Sistema.*

3.3.2 Diagramas de bloques

El diagrama mostrado en la figura 16 es de bloques y es utilizado para modelar el diseño del sistema, es decir, describe detalladamente los componentes lógicos de la aplicación como lo son las aplicaciones que puede contener, el controlador de datos y la conexión “plug and play”, los cuales crean algunas relaciones de dependencias entre bloques y se muestran algunas jerarquías derivadas de estos componentes de acuerdo a sus funciones. Como se puede observar el único bloque que no es obligatorio es el llamado otros dispositivos debido al cumplimiento de uno de los requerimientos del sistema que se detallaron anteriormente.

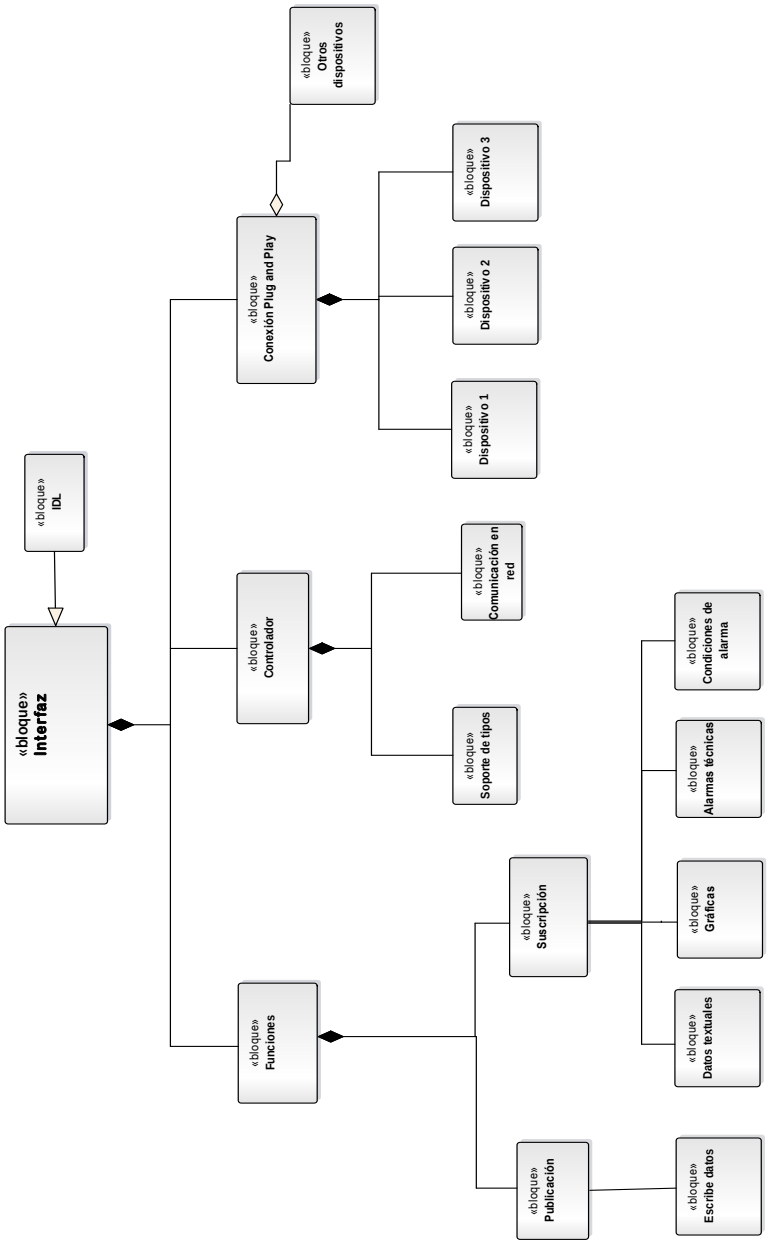

Figura 16. *Sistema de alto nivel*

En el siguiente diagrama (figura 17) se muestra una forma más compacta del diseño del sistema mostrando los componentes externos y el lugar que toma el modelo de transferencia de datos dentro del sistema que está basado en el estándar DDS, como se puede ver se tiene una restricción para utilizar la tarjeta BeagleBone Black la cual será utilizada para ejecutar la aplicación Adaptador. Tenemos dos actores que son los que podrían interactuar con nuestro sistema en una vista futura, el paciente y el personal médico. Y por último la aplicación que será descrita en los diagramas de secuencia.

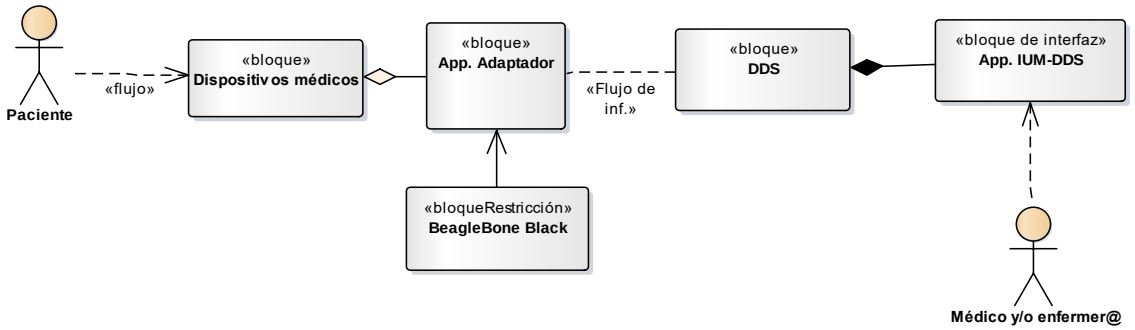

Figura 17. *Diseño compacto del modelo*

3.3.3 Diagramas de secuencia

Los diagramas de secuencia nos muestran una interacción, que representa la secuencia de mensajes o actividades entre instancias de clases, componentes del sistema o actores y se puede visualizar el flujo consecutivo de eventos. En los diagramas mostrados en las figuras 18, 19, 20 y 21, se despliegan las interacciones que se tienen entre los elementos del sistema como lo son el dispositivo médico encargado de sensar las variables médicas, la aplicación adaptador la cual tendrá el elemento publicador encargada de leer datos del dispositivo y hacer que se pueda comunicar a la aplicación IUM-DDS, el espacio virtual de datos es donde llegan todos los datos haciendo una partición por dominio seleccionado y la aplicación IUM-DDS es la encargada de recibir todos los datos y mandarlos a graficar. Por lo tanto, en el diagrama se especifican las actividades o interacciones que se realizan en cada elemento subsecuentemente y en qué tiempo o parte del proceso se introduce el estándar DDS el cual ocupa un espacio de datos virtual parecido a una base de datos donde se pueden mantener los datos publicados por el adaptador hasta que otra aplicación se suscriba al mismo dominio para acceder a ellos y estos puedan ser entregados de manera segura y confiable.

Cabe señalar que se presenta cuatro diagramas de secuencia correspondientes a los elementos más importantes que se presentan en el sistema.

En el diagrama de la figura 18 se muestra la secuencia de actividades que se realizan en las clases de la aplicación adaptador que es instalado en la tarjeta Beaglebone Black. En la clase serialOxi se realiza la habilitación del puerto y se empiezan a leer los datos del dispositivo, posteriormente esos datos serán obtenidos por la clase OximetroPublisher la cual se encargará de crear un dominio y un tema para que se instancien los datos y puedan ser publicados estos parámetros que rigen la comunicación fiable.

En la figura 19 se presenta el diagrama de secuencias correspondiente a la ejecución de la aplicación IUM-DDS la cual se divide en dos partes primero las clases correspondientes a la recepción de datos a través de la red que son mostradas en este diagrama. La secuencia inicia en la clase SubscriberOxi la cual tiene como función crear un dominio del participante y un tema para iniciar la comunicación de los datos de interés, posteriormente se sigue a la clase Data reader y typesuport para crear primero un objeto de datareader y que pueda estar atento de la llegada de datos correspondientes al tema de interés.

Para continuar con la ejecución de la aplicación se detalla la segunda parte que la compone, esta se muestra en el diagrama de la figura 20. En la imagen se exponen las clases correspondientes al procesamiento de datos y de la interfaz gráfica. Cabe mencionar que existe una clase controladora de datos por cada dispositivo y una general para implementar cada una de ellas en una interfaz gráfica.

Por último, se muestra en la figura 21, el diagrama de secuencias correspondiente al modo de operación de las clases de la alarma, esto se desarrolla para tener más control de eventos de estos datos. Las clases de la alarma tienen relación directa con el Publicador y el suscriptor de los diagramas de comunicación.

Empezando comunicación serial

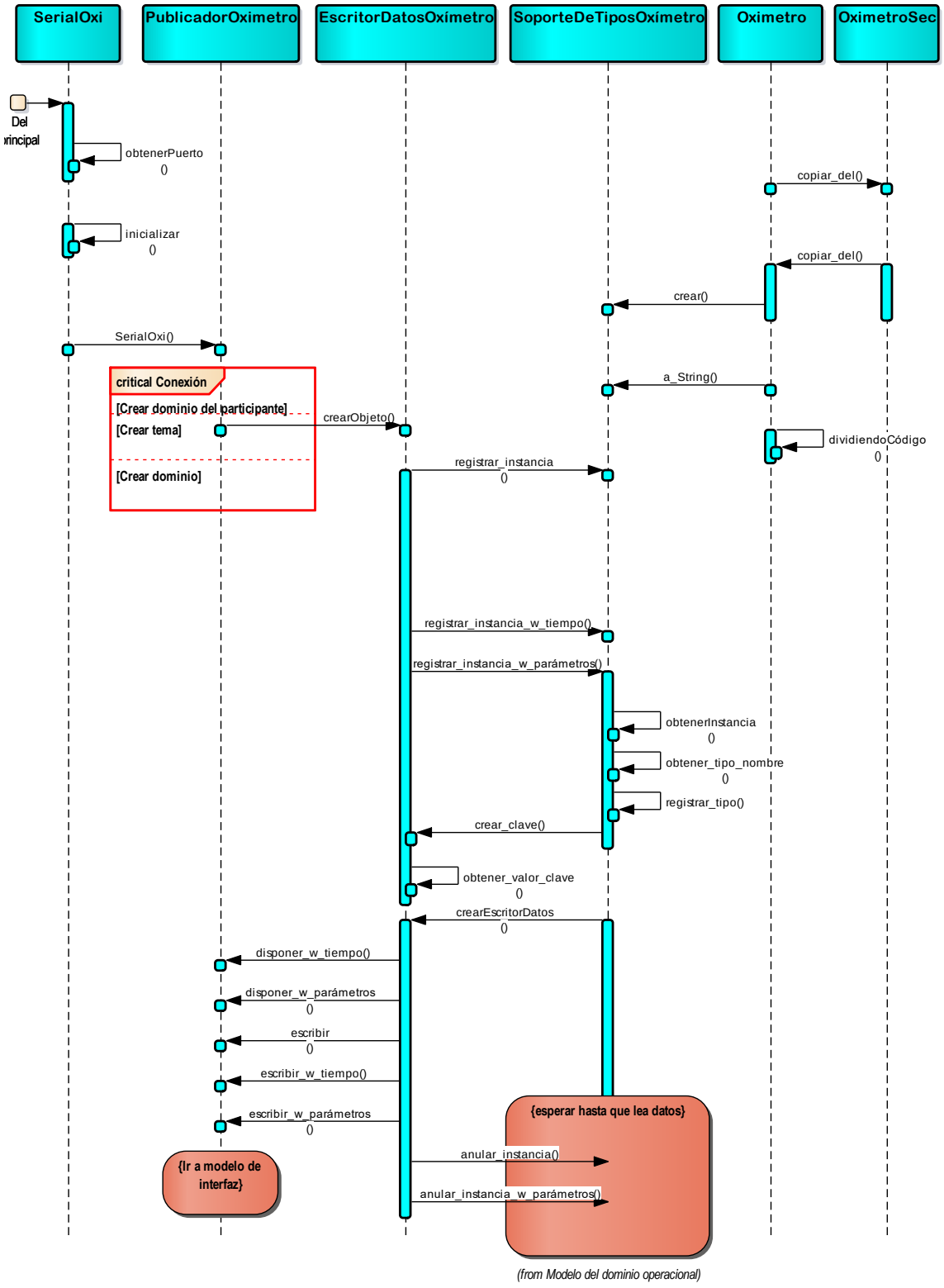

Figura 18. *Diagrama de secuencia (1)*

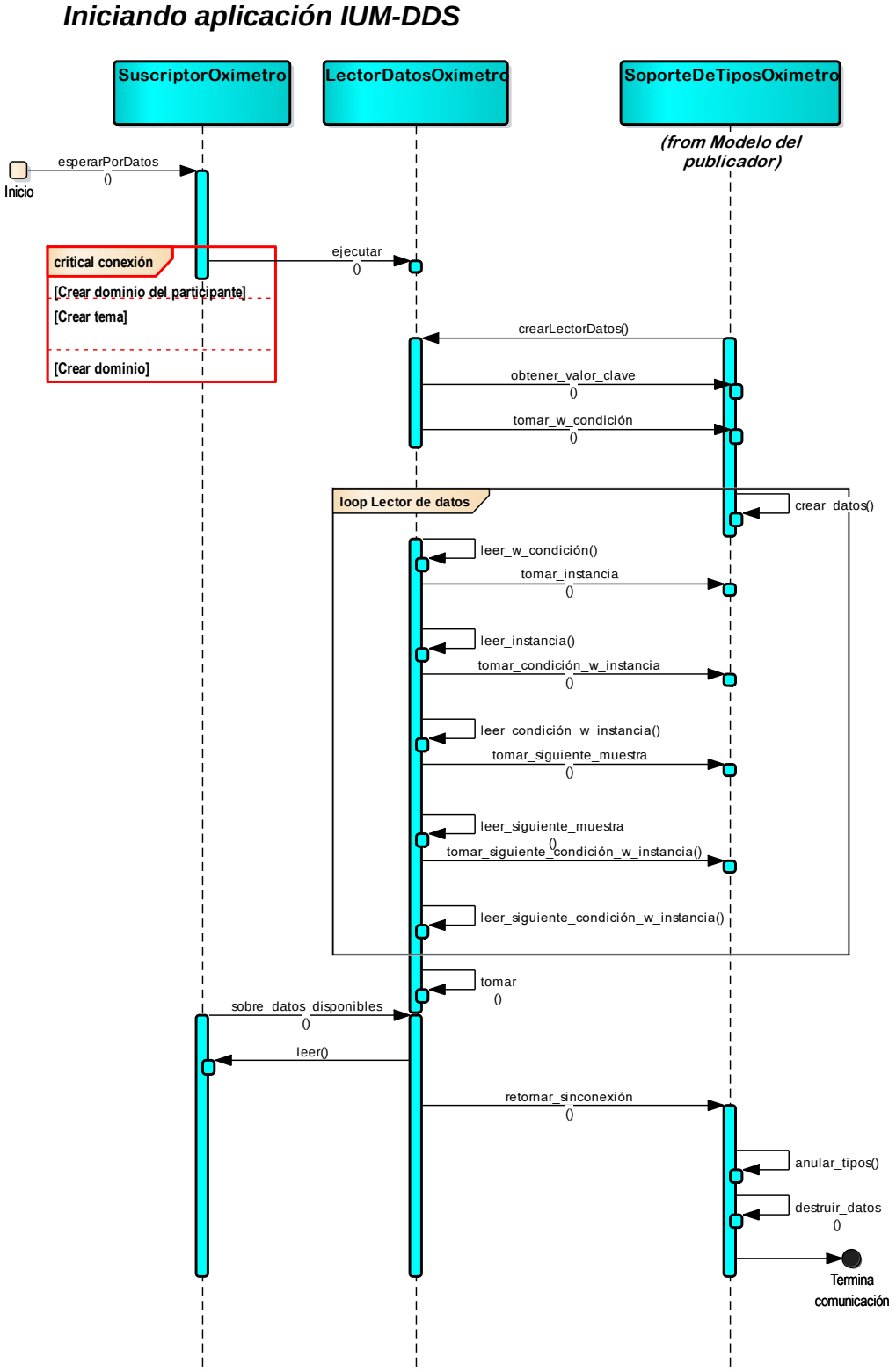

Figura 19. *Diagrama de secuencia (2)*

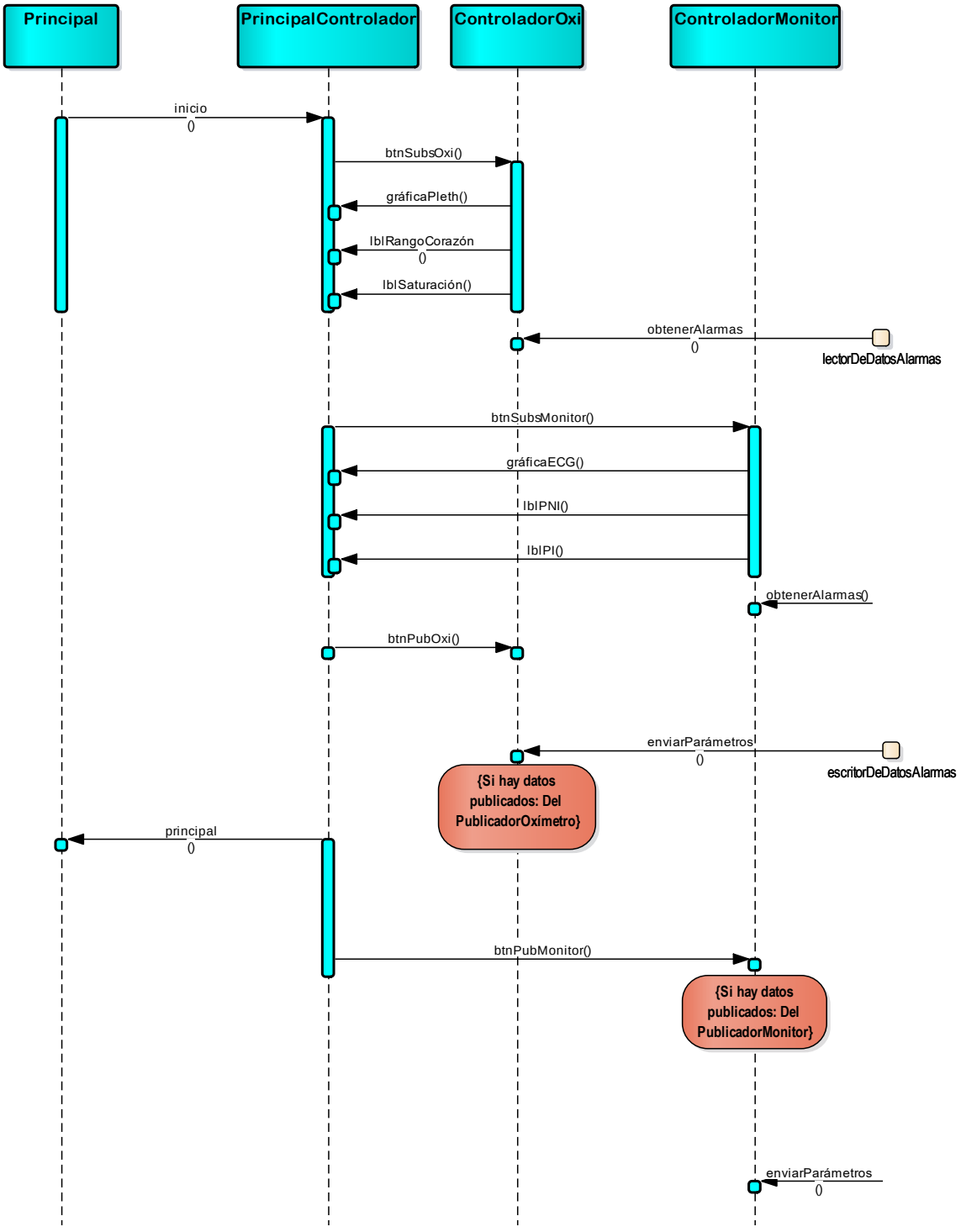

Figura 20. *Diagrama de secuencia (3)*

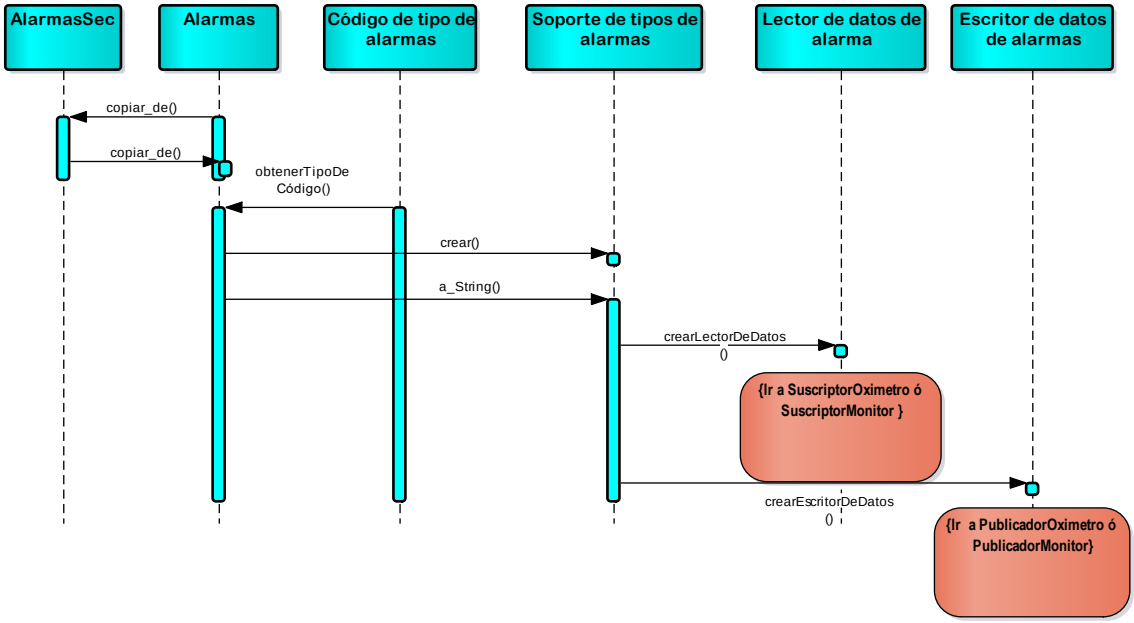

Figura 21. *Diagrama de secuencia (4)*

3.3.4 Diagrama de clases

El diagrama de clases es un tipo de estructura estática que describe la estructura de un sistema mostrando las clases que lo componen, sus atributos, operaciones o métodos implementados y las relaciones entre objetos.

En la figura 22 se expone el diagrama de clases correspondiente al sistema implementado, también se observan los paquetes a los que pertenecen las clases y los tipos de relación que se dan entre ellas. Por otra parte, se integran los métodos que se desarrollan en cada una de ellas y en algunos el tipo que retornan.

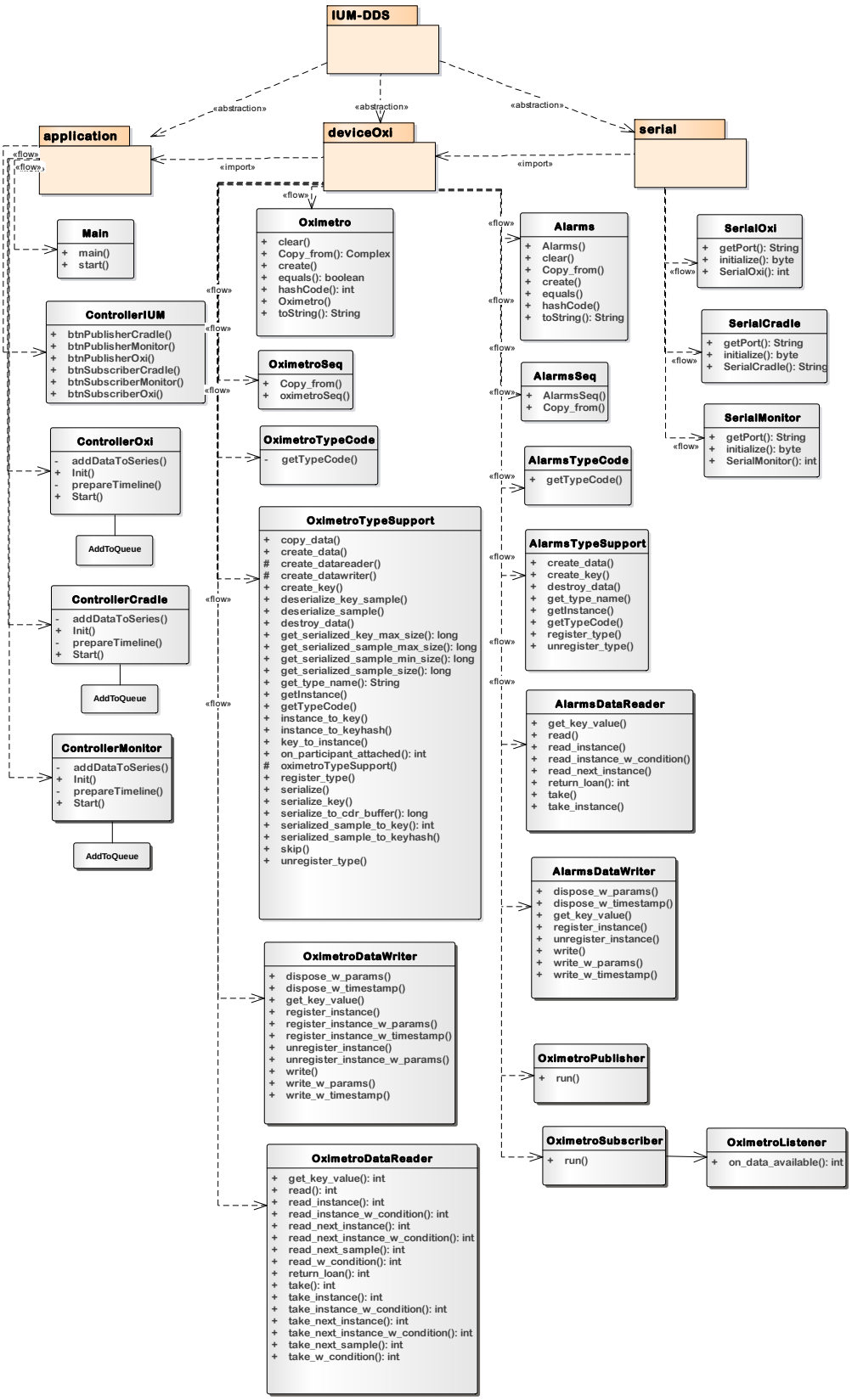

Figura 22. *Diagrama de clases*

3.3.5 Diagramas de implementación

El diagrama de implementación de software sirve para proporcionarnos información acerca del lenguaje de programación que se ocupa en el sistema, en la figura 23 se puede apreciar este diagrama mostrando el lenguaje java para la implementación de la interfaz IUM-DDS que es la primaria para el sistema, posteriormente en el hardware se utilizaron dos lenguajes primero C para la simulación de los dispositivos y después java para la implementación de la aplicación “Publicador” en la tarjeta BeagleBone Black.

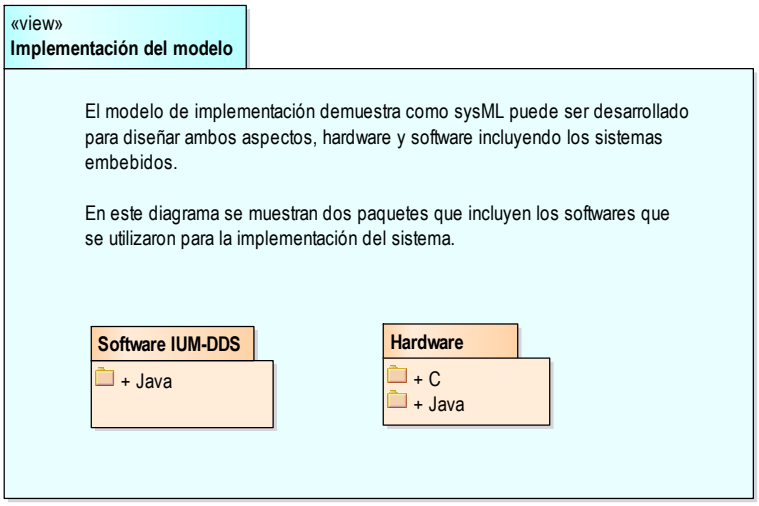

Figura 23. *Diagrama de descripción de software*

Otra perspectiva de ver al sistema es con el diagrama de relación del hardware y software que lo componen, ya que se observan cómo se implementan e interactúan las aplicaciones que se realizaron con los componentes que se utilizaron y también que relaciones o conectores son los que los unen; esto puede verse en la figura 24.

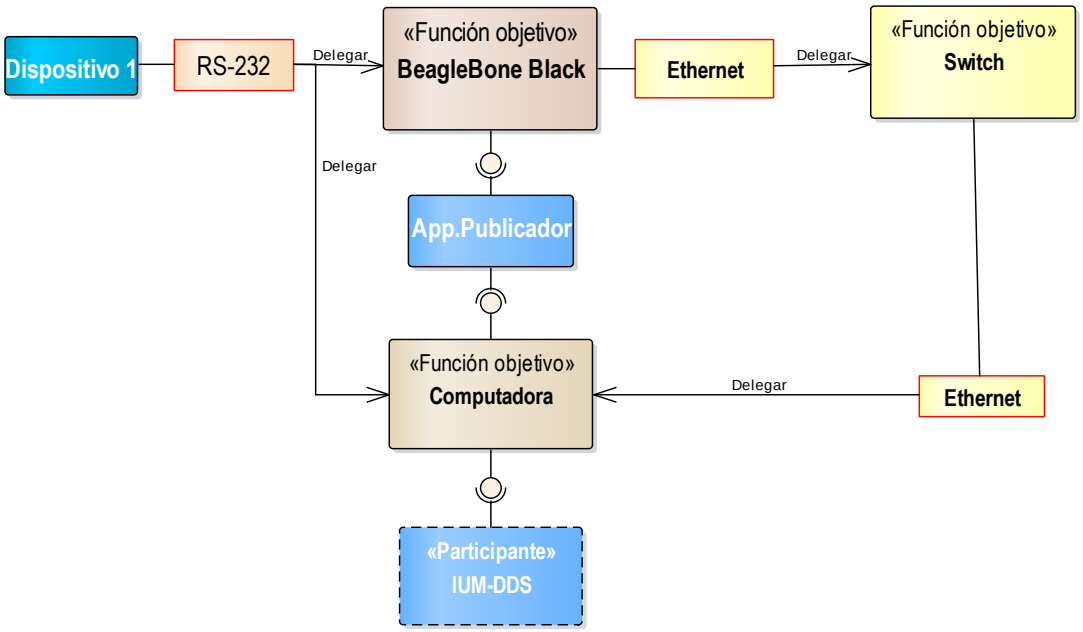

Figura 24. *Relación H-S*

Capítulo 4 Implementación

4.1 Introducción

En este nuevo capítulo se tratará el desarrollo de las aplicaciones de DDS. Como primer punto es necesaria la instalación del propio software de DDS por lo que se presentan las herramientas que son necesarias para lograr esta tarea; fue requerido realizarse para los sistemas operativos Linux y Windows. Para poder implementar la aplicación en la tarjeta beaglebone black se requiere utilizar el sistema operativo linux y para ejecutar la aplicación interfaz se va a realizar en Windows para verificar la adaptabilidad del estándar.

4.2 Herramientas para la implementación del sistema

Las herramientas para la implementación se dividen en dos partes, software y hardware. Dentro de la primera se incluyen las siguientes:

Lenguaje de programación; el que se utiliza para la implementación del sistema es java por sus múltiples recursos y está fuertemente ligado a android lo que se espera utilizar a futuro. Para definir el modelo de datos ocupados se utiliza el lenguaje IDL.

GNU Compiler Collection (GCC); este compilador es utilizado para lograr el buen funcionamiento del estándar DDS.

JDK; es el software que proporciona herramientas de desarrollo para realizar programas en java y es utilizado por eclipse.

Eclipse; es el IDE utilizado para la implementación del sistema ya que es adaptable y se pueden instalar múltiples plugins de acuerdo a las necesidades requeridas.

Putty; es un emulador de terminal, un programa que permite conectar con máquinas remotas y ejecutar programas a distancia.

RTI Connext DDS; es el middleware de red para aplicaciones distribuidas de tiempo real utilizado, ya que cuenta con la implementación del estándar DDS de la OMG, además de sus herramientas para el análisis del transporte de paquetes.

Sistemas operativos; el sistema operativo Linux es requerido para la implementación del sistema en la Beaglebone Black y Windows es utilizado para la implementación de la interfaz.

Por otro lado, tenemos las herramientas necesarias correspondientes al hardware para la implementación del sistema, éstas se listan abajo.

PC; esta será necesaria para el monitoreo de los dispositivos médicos, aquí se instalará el software correspondiente a la interfaz y hará la actividad de suscribirse a los dispositivos conectados. Esta computadora deberá tener un mínimo de 8 GB de memoria RAM, 500 GB de disco duro con una velocidad mínima de 2.6 GHz.

Tres tarjetas Beaglebone Black; estas tarjetas embebidas hacen el papel de adaptador de dispositivo, se le da ese nombre por que sirven para adaptar los parámetros recibidos en un formato que pueda publicarse a través del estándar DDS.

Cableado Ethernet; es definido generalmente como el estándar más popular creado para la intercomunicación de componentes de red fabricados por varias compañías, considerando la manera en que cada componente trabaja e interactúa con los demás componentes de la red, por esta razón será ocupada en este proyecto para la comunicación entre publicadores y suscriptores.

Switch; es ocupado para la interconexión de las tarjetas embebidas y el pc, creando una red LAN.

Oxímetro de pulso y monitor de signos vitales; son los dispositivos que se ocupan en este sistema, y los cuales se describen en el capítulo 2.

4.3 Procedimiento para la instalación de RTI Connext DDS en PC

PC con Ubuntu instalado

Primero se tratará sobre las instalaciones requeridas en una PC con Linux Ubuntu de 32 bits, esta computadora es utilizada para hacer los programas que se ejecutarán en la beaglebone black.

Se accedió al centro de software de Ubuntu y se buscó el compilador de GNU, seguido se dio clic en el botón instalar. De la misma manera se instaló el IDE de eclipse.

Para instalar java 8 en Ubuntu, se ingresaron los siguientes comandos.

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

Después se verifica si se instaló correctamente java mediante el siguiente comando.

```
Java -version
```

Y en la terminal se mostró la versión instalada de java (figura 25).

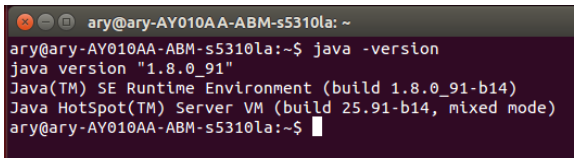

Figura 25. *Versión de java*

Luego de instalar los programas anteriores nuestro equipo está listo para la descarga e instalación de DDS de RTI.

El software RTI connext DDS está disponible en el sitio web de RTI [36], en donde es necesario registrar los datos y correo de la persona para realizar la respectiva descarga. En esta página se encontrarán algunas versiones de DDS de RTI compatibles tanto para sistemas Red Hat como para sistemas Windows. RTI proporciona una licencia de prueba 30 días para realizar pruebas de DDS. Nosotros disponemos de una licencia de un año debido a que se registró el proyecto por parte de la universidad y dura hasta noviembre del 2016 por lo que se utilizará para finalizar el proyecto y ver el funcionamiento de DDS de RTI.

Después de haberse descargado el archivo de RTI correspondiente a la plataforma Linux de 32 bits, aparecerá un archivo llamado *"rti_connext_dds-5.2.0-eval-i86Linux3gcc4.8.2.run",* se da clic secundario sobre el archivo, seleccione propiedades y en la pestaña permisos active el recuadro que permite

ejecutar el archivo como un programa y cierre esa ventana. Posteriormente se vuelve a dar clic secundario sobre el archivo y esta vez seleccione ejecutar, seguido se presentará una guía de instalación de RTI y solo se seguirán los pasos solicitados hasta finalizar. Cuando se haya finalizado se abrirá el RTI Launcher y en la pestaña de instalación hará falta la licencia que proporciona la comunidad de RTI mediante un correo al momento de tu registro. Cuando se hay obtenido el archivo de licencia este deberá ser pegado en la carpeta donde se instaló RTI. En mi caso la ruta es la siguiente

/home/usuario/rti_connext_dds-5.2.0/rti_license.dat

Cuando realices este paso automáticamente será detectado el archivo de la licencia en el Launcher como se observa en la figura 26.

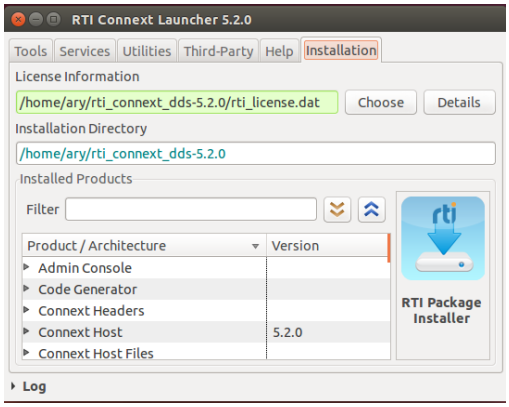

Figura 26. *RTI Launcher*

Para realizar aplicaciones que sean ejecutables en la beaglebone black, es necesario aplicar la librería del estándar DDS correspondiente al procesador *ARM*, ya que como son arquitecturas diferentes no se puede ejecutar la aplicación implementada para la arquitectura de la computadora en la tarjeta.

Conociendo esto, debemos descargar e instalar la librería del sitio oficial [35], esto nos proporcionará un archivo con nombre *rti_connext_dds-5.2.0-core-target-armv6vfphLinux3.xgcc4.7.2. rtipkg*. Como se puede observar la extensión de este archivo no es convencional, por lo que se requiere del RTI Launcher para ejecutarlo.

En la figura 27 vemos un botón con el nombre *RTI Package Installer*, damos clic ahí y podremos instalar el archivo de la librería asignándole la ruta donde se encuentra la librería y posteriormente dando clic en el botón *install* mostrado en la figura 28.

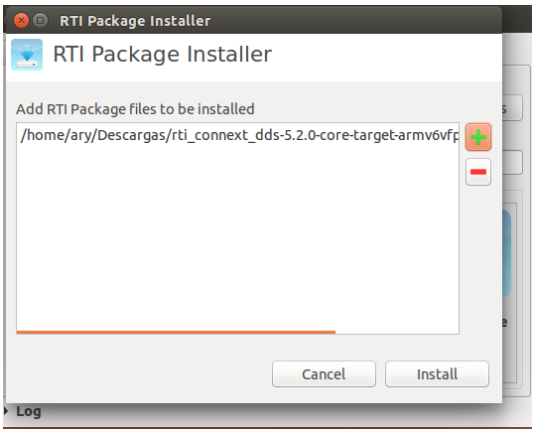

Figura 27. *Instalador de RTI*

Cuando haya terminado el proceso deberemos ver la carpeta correspondiente a la plataforma de *arm* ubicada en la carpeta de librerías de la instalación de RTI, esto se muestra en la figura 28.

Figura 28. *Plataformas de RTI*

El paso que falta para que se pueda trabajar con el estándar DDS de RTI es aplicar las variables de entorno. Una variable del entorno es un valor dinámico cargado en la memoria, que puede ser utilizado por varios procesos que funcionan simultáneamente.

Antes de explicar la configuración de las variables de entorno vamos a descargar el software *VNC viewer-5.3.1* de la referencia [38], este programa nos servirá más adelante para acceder al escritorio de la tarjeta y la biblioteca del puerto serial del sitio web [39], que contiene los *jar* necesarios para poder establecer comunicación vía serial con la tarjeta. Una vez descargado este último recurso, deberá extraer el archivo e identificar la carpeta del sistema operativo al que se vaya a instalar, en este caso Linux.

Copiar el archivo *librxtxSerial.so* a la siguiente ruta, donde el valor de la variable JAVA_HOME se proporciona en la configuración de las variables de entorno

JAVA_HOME/jre/lib/i386

El archivo *RXTXcomm.jar* será copiado en la siguiente ruta

JAVA_HOME/jre/lib

En el caso de que el archivo con extensión *jar* sea ocupado en un IDE, deberá incluirse en un folder dentro del proyecto.

A continuación, se explica cómo declarar las variables de entorno.

Configuración de variables de entorno

Debe abrir una nueva terminal y posteriormente escribir el siguiente comando para acceder a un editor de textos con permisos de administrador

```
sudo gedit
```

Al dar *enter* solicitará la contraseña del usuario y deberá proporcionarse, obteniendo como resultado un editor de texto abierto. En este archivo se da clic en el botón abrir y desplegará una nueva ventana solicitando la ruta del archivo que se desea abrir. En la etiqueta “Lugar” se escribirá

/etc/environment

Y esta ruta abrirá el archivo donde se encuentra la variable PATH del sistema. Entonces en ese archivo se agregarán las siguientes variables de entorno; JAVA_HOME con valor de la ruta donde se encuentra instalado java, NDDSHOME con valor de la ruta donde se encuentra instalado RTI, LD_LIBRARY_PATH con valor de la ruta donde se encuentra la plataforma en la que serán construidas las aplicaciones de RTI. Por último, se deberá agregar la ruta de la carpeta *bin* de java en la variable PATH al igual que la ruta donde se encuentra el archivo .so de la biblioteca serial. A continuación, se muestran como quedaron definidas en el archivo las variables de entorno (figura 29).

Figura 29. *Variables de entorno en Linux*

Con estas instalaciones ya podemos crear aplicaciones con el estándar DDS de RTI sobre Linux en la PC.

PC con Windows instalado

Para poder instalar RTI en Windows es necesario tener instalado *visual studio 2010 y java*, por consiguiente, haremos la instalación de ambas herramientas.

El programa de *visual studio ultimate 2010* fue descargado de [40] y posteriormente se ejecutó el instalador desde la ubicación del equipo haciendo doble clic sobre el archivo .exe y se siguieron las instrucciones de la pantalla. Para instalar este programa es necesaria la conexión a internet.

Posteriormente se descargó *JDK* del sitio web oficial [41] y se ejecutó de la misma manera que el software anterior.

Ya teniendo instalados estos softwares se procede a la instalación de RTI connext DDS. La descarga del programa se obtiene del mismo sitio web explicado para Linux [36], solo que esta vez será descargada la versión correspondiente a la plataforma de Windows 7 de 64 bits, la cual lleva por nombre *rti_connext_dds-5.2.0-eval-x64Win64VS2010*, inmediatamente se procede a su instalación dando doble clic y siguiendo las instrucciones de instalación. Cuando se haya instalado se copia el archivo de licencia a la carpeta de instalación de RTI.

Para el caso de Windows no es necesario instalar el archivo target de la plataforma *arm*, ya que esta computadora se va a ocupar para la aplicación de la interfaz, lo que si es necesario es obtener la librería para crear los gráficos.

La librería que se utiliza para obtener los gráficos se llama *JfreeChart* y puede ser descargada de [42], esta descarga contiene dos archivos importantes *JComm.jar y Jfreechart.jar*, ambos archivos deberán ser incluidos en el proyecto que se realice en Eclipse.

Configuración de variables de entorno

Para Windows se requiere declarar más variables de entorno que en Linux.

Para configurar las variables de entorno nos dirigimos a *Equipo > propiedades del sistema > configuración avanzada del sistema > Variables de entorno > nueva*

Agregamos las variables correspondientes en este caso quedaron de la siguiente manera.

JAVA_HOME => C:\Program Files\java\jdk1.8.0_60

NDDSHOME => C:\Program Files\rti_connext_dds-5.2.0

REPLAY_HOME => C:\Program Files\rti_connext_dds-5.2.0\resource\app\app_support\recording_service

RTI_LICENSE_FILE => C:\Program Files\rti_connext_dds-5.2.0\rti_license.dat

Y en la variable *Path* se agregan las siguientes rutas.

C:\Program Files\rti_connext_dds-5.2.0\lib\x64Win64VS2010; C:\Program Files\java\jre1.8.0_60\bin

Con estos programas instalados ya se puede ocupar RTI connext DDS, pero también necesitamos descargar eclipse ya que es el IDE que se va a ocupar para el desarrollo de la aplicación y *putty* el emulador de terminal para acceder a la tarjeta beaglebone black más adelante.

Eclipse puede ser descargado de [43], este sitio proporciona un archivo comprimido con el nombre *eclipse-jee-mars-1-win32-x86_64*, el cual deberá ser extraído para acceder al ejecutable de eclipse.

El programa *Putty* es gratuito y puede ser descargado de [44]. La descarga proporciona un archivo ejecutable, el cual puede ser instalado haciendo doble clic sobre el archivo.

4.4 Procedimiento para la instalación de RTI Connext DDS en BeagleBone Black

De inicio necesitamos una memoria microsd ya que en ésta se instalará el sistema operativo, lo podemos instalar en la memoria interna de la tarjeta, pero esto nos restringe en el espacio de la memoria. Los pasos que continúan se realizaran en la PC con Linux instalado.

Deberemos descargar una versión de Linux compacta y que sea compatible con DDS, por lo tanto, la versión descargada fue Ubuntu 14.04 y fue encontrada en [45], en este sitio hay múltiples versiones del sistema operativo que le puedes instalar a la tarjeta, pero en este caso se escogió que fuera compatible con el estándar y que se pudiera instalar en microsd, el nombre completo del archivo descargado es *bone-ubuntu-14.04.3-console-armhf-2016-02-11-2gb.img.xz*

Para formatear la microsd será necesario el programa *Gparted* sino se tiene instado podrá dirigirse al centro de software de Ubuntu y obtenerlo. Seguido se abrirá este programa, al abrirlo solicitará la contraseña de usuario por lo que deberás proporcionarla y teniendo conectada la microsd buscarás cual es el sufijo que le corresponde y seleccionarlo en el programa, ya que fue seleccionado darás clic secundario sobre la partición y seleccionar desmontar. A continuación, volver a dar clic secundario y elegir formatear como *ext4.*

Una vez finalizados los pasos anteriores deberás desplazarte a la carpeta donde descargaste la versión de Ubuntu para la tarjeta, se da clic secundario sobre el archivo y se elige abrir con *Escritor de imágenes de disco*, posteriormente te aparecerá una ventana solicitando que elijas el dispositivo sobre el que quieras escribir la imagen por lo tanto se seleccionará el correspondiente a la microsd conectada y aceptar; esperar hasta que finalice el proceso y expulsar la memoria.

Ahora conectamos el cable de red, la alimentación, el monitor y un teclado en la Beaglebone Black e insertamos la microsd. Un punto muy importante es que, para el *booteo* desde la SD, deberás presionar el botón que se encuentra junto al conector USB al momento de conectar a la alimentación, cuando los leds comiencen a parpadear rápidamente ya se puede soltar este botón.

En la terminal que aparezca en la pantalla se hará lo siguiente:

- Ingresar usuario/contraseña, *ubuntu/temppwd* respectivamente.

Ahora bien, el espacio de la instalación asume que se están ocupando 2GB, para ocupar todo el espacio de la tarjeta microsd, se debe realizar el procedimiento una sola vez, el cual mostramos a continuación. Para esto modificamos la partición de la tarjeta de la siguiente manera.

- Ingresar el siguiente comando para ver cuál identificador pertenece a la tarjeta microsd en la beaglebone

```
ls –l /dev/mmcblk*
```

- Se examinará la partición de la tarjeta microsd

```
fdisk /dev/mmcblk0
```

- Digitar **p** para visualizar la tabla de particiones
- Posteriormente digitar d para borrar la partición creada por la escritura de la imagen
- Volver a crear la partición **1**, con la letra **n**, seguido el valor por defecto **1** y especificar inicio y sectores extremos de la nueva partición, en este caso utilizamos los valores por defecto ya que queremos todo el espacio de la tarjeta disponible, seguido un *enter.*
- Se escriben los datos seleccionados con la letra **w**

```
sudo reboot
```

- Por último, se reinicia la tarjeta con el siguiente comando

Ya que se volvió a iniciar la tarjeta digitar el siguiente comando para que acepte los cambios de la memoria.

```
sudo resize2fs /dev/mmcblk0p1
```

Para verificar que se esté tomando todo el tamaño de la microsd se escribe el comando y se verá el tamaño de las particiones.

```
df
```

A continuación, ejecutamos los siguientes comandos para actualizar los paquetes instalados en la tarjeta, para hacer este paso se requiere tener el cable Ethernet conectado.

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

Después de tener actualizado el sistema operativo se procede a instalar el escritorio gráfico de Ubuntu, aquí se instala el *xfce* ya que es ligero para la tarjeta.

```
sudo apt-get install xfce4
```

Una vez instalado el escritorio debemos apagar la tarjeta con el siguiente comando.

```
sudo shutdown –h now
```

Acceso remoto

El acceso remoto tiene algunas ventajas por si no se cuenta con un teclado, mouse y monitor para integrarse a la tarjeta. Por esta razón vamos a explicar una forma de acceder a la tarjeta por medio de esta herramienta.

Primero debemos conectarnos a la tarjeta por *ssh* y posteriormente hacer la conexión via escritorio remoto, para esto deberemos conectar la tarjeta a la computadora por medio del cable usb.

Para lograr el acceso via ssh en windows deberemos abrir el programa *putty* e ingresar la dirección *IP* de tarjeta, por lo que se escribe *192.168.7.2*, ya que esta es la dirección por defecto que trae configurada la tarjeta para accederse via usb, despues de ser ingresada la dirección *IP* se oprime el botón *open*, donde se podrá ingresar a la tarjeta. La pantalla inicial de *putty* se observa en la figura 30.

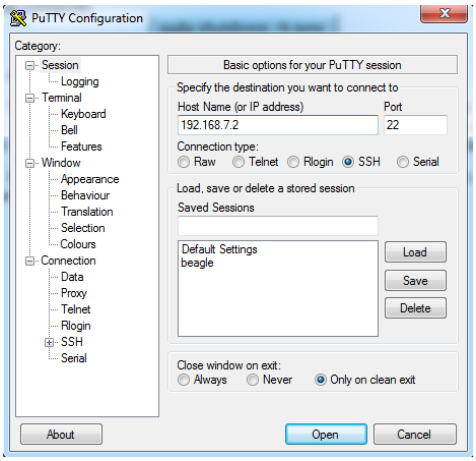

Figura 30. *Ventana inicial de Putty*

Para el caso de Linux no hay necesidad de instalar un programa como éste. Los pasos son muy simples:

- Abrir una terminal de Linux
- Introducir el siguiente comando

```
ssh ubuntu@192.168.7.2
```

Pedirá la contraseña de la tarjeta y deberá proporcionarse para que dé acceso (*temppwd*). Tanto para Linux como para Windows al momento de acceder debe mostrar la información contenida en la figura 31.

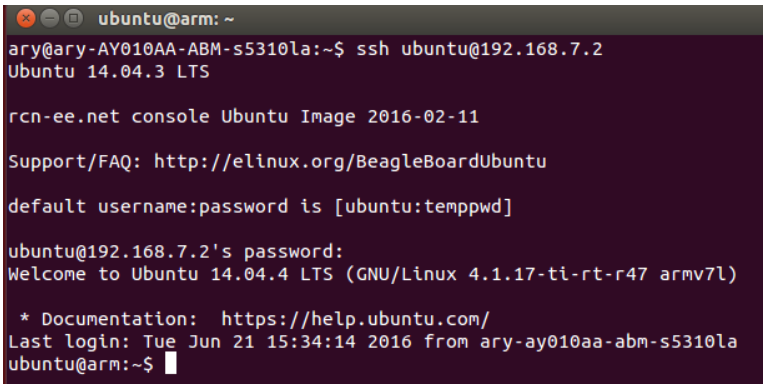

Figura 31. *Inicio de conexión de la beaglebone black*

Puesto que ya se obtuvo el acceso por medio del protocolo ssh los siguientes pasos son genéricos.

Se instala VNC para acceso remoto por medio del siguiente comando

```
sudo apt-get install tightvncserver
```

Luego ejecutamos

```
tightvncserver
```

Para habilitar una conexión debemos ver qué resolución soporta la beaglebone. De acuerdo a la documentación de la tarjeta, la *HDMI* soporta las siguientes resoluciones

- 1280x1024
- 1440x900
- 1024x768
- 1280x720

Por lo tanto, para habilitar el escritorio remoto ingresamos el siguiente comando y ya estará listo para que podamos acceder al escritorio de la tarjeta.

```
vncserver :1 –geometry 1280x1024 –depth 32
```

Para ingresar al escritorio abrimos *vnc_viewer*, este programa está disponible tanto para Windows como para Linux. La pantalla aparecerá como en la figura 32, donde se encuentra la etiqueta *"VNC Server"* pondremos la dirección *IP* de la tarjeta seguido de dos puntos y en este caso el 1, que corresponde al escritorio habilitado en el paso anterior; posteriormente pulsaremos el botón *conectarse* y ya podremos ver el escritorio de la beaglebone black como se muestra en la figura 33.

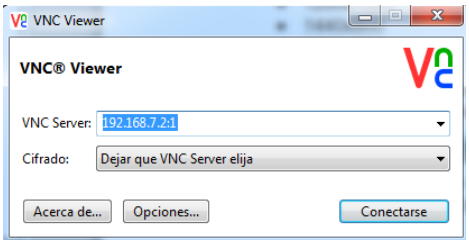

Figura 32. *VNC viewer*

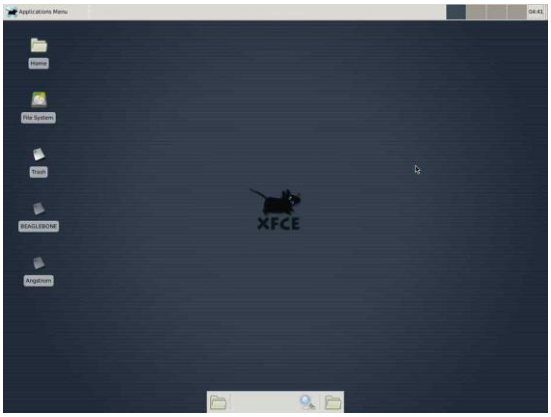

Figura 33. *Escritorio de la beaglebone*

Instalando java

Es requerido instalar java en la tarjeta para poder correr las aplicaciones, por lo que seguiremos los siguientes pasos para obtenerlo. Primero vamos a descargarlo desde Oracle mediante la siguiente línea.

```
wget --no-check-certificate --no-cookies --header "cookie: oraclelicense=accept-securebackup-cookie" http://download.oracle.com/otn-pub/java/jdk/8u33-b05/jdk-8u33-linux-arm-vfp-hflt.tar.gz
```

Posteriormente nos dirigimos a la ubicación del archivo descargado

```
cd /home/Ubuntu/downloads
```

Ya que nos encontramos en la ruta extraemos el archivo

```
gunzip jdk-8u33-linux-arm-vfp-hflt.tar.gz
```

Siguiente paso descomprimir el archivo

```
tar xfv jdk-8u33-linux-arm-vfp-hflt.tar
```

Seguido moveremos la carpeta a la ruta correspondiente.

```
mv jdk-8u33-linux-arm-vfp-hflt /usr/bin/
```

Instalando librerías requeridas

Para poder correr la aplicación requerida en la tarjeta beaglebone black se requiere la librería target de DDS para *arm* y la biblioteca del puerto serial. Estas librerías ya se encuentran instaladas en la computadora con Linux, por lo que se harán los siguientes pasos para copiarlas a la tarjeta.

El primer comando deberá ser escrito en una terminal del servidor de Linux y se ocupa para pasar RTI a la tarjeta, donde se pone la ruta de donde se encuentra el archivo que se quiere copiar y después el destino.

```
scp –r home/usuario/rti_connext_dds-5.2.0 ubuntu@192.168.7.2:
/home/ubuntu/rti_connext_dds-5.2.0
```

Posteriormente se copiará el archivo rxtxSerial.jar

```
scp –r home/usuario/descargas/rxtx.Serial.jar ubuntu@192.168.7.2:
/home/ubuntu/downloads
```

Para poder ejecutar una aplicación que ocupe el puerto serial necesitamos el archivo con extensión *so*, éste es diferente al que se ocupa para la computadora, por la cuestión de la plataforma *arm*, por lo que para obtenerlo haremos lo siguiente.

Primero deberemos estar conectados vía ssh a la beaglebone black y ejecutar el siguiente comando.

```
sudo apt-get install librxtx-java
```

Esto instala *librxtxSerial.so* en */usr/lib/jni*. Esta ruta se incluye automáticamente en la ruta de la biblioteca de Java en algunas plataformas, pero no todas, por lo que necesita actualizar esta propiedad.

Moveremos la librería a la ubicación de java, para esto deberemos acceder como *root.*

```
sudo su
```

Luego movemos el archivo a la ubicación de java.

```
cp /usr/lib/jni/librxtxSerial.so /usr/jdk1.8.0_102/jre/lib/arm
```

Declaración de variables de entorno en la tarjeta

Accedemos a la Beaglebone black con permisos de administrador con el comando anterior *sudo su*.

Posteriormente ingresamos el siguiente comando para saber dónde se encuentra el archivo de las variables de entorno

```
echo $SHELL
```

Después de que nos arroje la ruta, nos ubicamos en la misma con el comando cd. Y seguido editamos el archivo con el siguiente comando

```
sudo --edit .profile
```

Una vez abierto el archivo agregaremos las siguientes líneas con las variables de entorno.

export PATH=$PATH: /usr/jdk1.8.0_102/bin/;/usr/jdk1.8.0_102/jre/lib/arm/

export JAVA_HOME=/usr/jdk1.8.0_102/

export NDDSHOME=/home/Ubuntu/rti_connext_dds-5.2.0

export LD_LIBRARY_PATH=/home/Ubuntu/ rti_connext_dds-5.2.0/lib/armv6vfphLinux3.xgcc4.7.2

Una vez agregadas estas variables guardamos los cambios realizados con

```
ctrl + o
```

Y cerramos el archivo con

```
ctrl + x
```

Verificamos que se hayan guardado los cambios con el siguiente comando

```
more .profile
```

Y éste nos entregará los cambios que se hayan realizado en el archivo. Por último, deberemos reiniciar la tarjeta con el comando ya visto anteriormente *reboot*.

4.5 Modelos y frames de datos

Los frames de datos y el modelo de datos tienen relación directa ya que la estructura desarrollada en IDL corresponde al manejo de datos del frame de cada dispositivo con su tipo. Los modelos desarrollados en IDL deben ser analizados gramáticamente por un constructor de tal manera que puedan crear la estructura de tipos que será entregada por el middleware, para que el sistema pueda controlar los diferentes eventos de cada tipo de dato. En las tablas 4, 5 y 6 se muestran los frames de datos correspondientes a cada dispositivo médico.

Tabla 4. *Frame de datos para el monitor de signos vitales*

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
paquete 1	1	Status	ECG_H	ECG_L	CHK
Paquete 2	1	Status	HR_H	HR_L	CHK
paquete 3	1	Status	TEMP_h	TEMP_L	CHK
paquete 4	1	Status	NIPB_H	NIPB_L	CHK

Tabla 5. *Frame de datos para el oxímetro de pulso*

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Paquete 1	1	Status	PLETH	HR MSB	CHK
Paquete 2	1	Status	PLETH	HR LSB	CHK
Paquete 3	1	Status	PLETH	SpO2	CHK
Paquete 4	1	Status	PLETH	REV	CHK
Paquete 5	1	Status	PLETH	reserved	CHK

Para iniciar con el desarrollo del sistema primero debemos definir los tipos de datos que se ocupan. Se desarrollaron los archivos en IDL donde se describen los tipos de datos de los dispositivos médicos, los cuales se observan en las figuras 34 y 35.

En este código se declaran tres paquetes uno para cada dispositivo, posteriormente se definen los temas que se utilizarán en la comunicación con el estándar DDS y los tipos de datos normales y los correspondientes a las alarmas proporcionadas por cada dispositivo.

```
module deviceOxi{

const string AlarmTopic = "com::rti::medical::Alarm";
const string OximetroTopic = "com::rti::medical::Oximetro";

struct Alarms_range{
    string id1;
    double spo2_high;
    double spo2_low;
    string id2;
    double HR_high;
    double HR_low;
};
struct Oximetro{
    string ID; //@key
    long pleth;
    long spo2;
    long HR_H;
    long HR_L;
};
};
```

Figura 34. *Modelo de datos del Oximetro en IDL*

```
module deviceMonitor{

const string AlarmTopic = "com::rti::medical::Alarm";
const string MonitorTopic = "com::rti::medical::SignMonitor";

struct SignMonitor{
    string ID3;//@key
    long ECG;
    long HR_H;
    long HR_L;
    float TEMP;
    long NIPBH;
    long NIPBL;
};
struct Alarms_range{
    string id4;
    double ECG_H;
    double ECG_L;
    float TEMP_H;
    float TEMP_L;
    double NIPB_H;
    double NIPB_L;
};
};
```

Figura 35. *Modelo de datos del monitor de signos vitales en IDL*

4.6 Implementación de la aplicación

Una vez instaladas las herramientas anteriores seguiremos con la implementación de la aplicación. Lo primero que se hace es implementar un archivo con extensión *IDL*, el cual representa el modelo de datos que utiliza la interfaz, estos archivos fueron mostrados en la sección anterior; estos archivos son muy importantes ya que desglosan los tipos de datos que se manejan en toda la aplicación.

Posteriormente abriremos el *RTI Launcher* y nos desplazamos a la pestaña *Utilities*, seguido seleccionamos *code Generator* lo cual nos dará la ventana como la figura 36, aquí debemos dar como entrada de archivo la ruta donde se encuentra el *archivo.idl* creado, en la opción formato se debe seleccionar *IDL* y en la salida del directorio debemos asignar la ruta donde se guardará el proyecto creado. Continuando seleccionamos el lenguaje, en este caso utilizamos java y la plataforma de salida seleccionamos la correspondiente a *arm* por último oprimimos el botón *run* y aparecerá una terminal donde nos indicará cuando haya terminado el proceso.

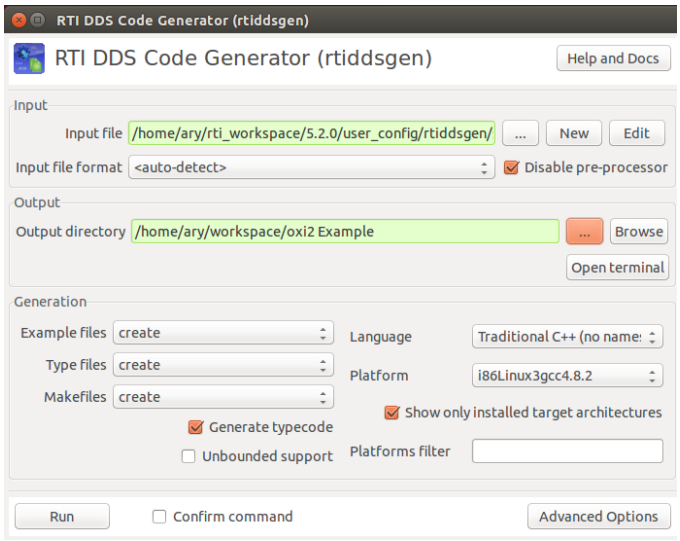

Figura 36. *RTIDDSGEN*

Ahora abriremos el IDE de eclipse, y nos colocaremos en el espacio de trabajo (*workspace),* donde fue creado el proyecto. Una vez abierto damos clic secundario sobre el explorador de paquetes.

Seleccionar *Import > General > Existing Projects into Workspace > next > select root directory > next > finish*

Podremos observar el proyecto incorporado en el explorador de paquetes de la figura 37. Continuamos agregando un nuevo paquete haciendo clic secundario sobre el paquete, seleccionamos *new* y seguido *package*, asignamos el nombre requerido y finalizamos.

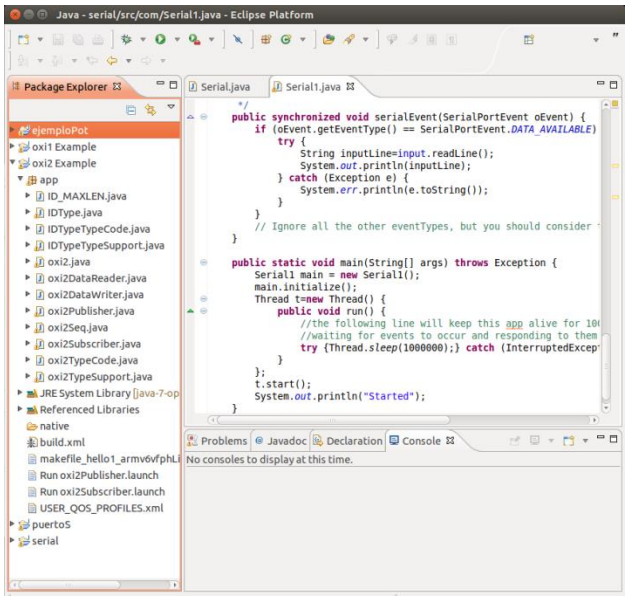

Figura 37. *Proyecto importado*

Nuevamente hacemos clic secundario sobre el proyecto, pero ahora elegiremos *new* > *class*, la cual corresponderá a la comunicación serial.

Uno de los pasos muy importante es agregar las librerías dentro del proyecto por lo que agregaremos un folder dentro del proyecto haciendo clic secundario > *new* > *folder*, a esta carpeta le pondremos el nombre de *lib* y ahí guardaremos los *jar* que necesitemos.

Posteriormente configuraremos el *path* del proyecto, haciendo clic secundario sobre el *proyecto* > *Build path* > *configure Build Path* > *Add JARs y seleccionamos la carpeta donde se encuentra la librería jar* (figura 38).

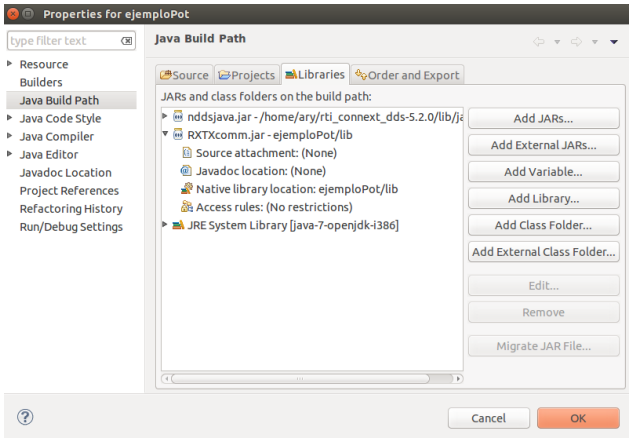

Figura 38. *Agregando librerías jar*

Ahora pasaremos a la clase Publisher, donde se publicarán los datos obtenidos del serial. Para que esto sea posible se debe hacer un objeto a la clase del puerto serial e instanciar los datos. Otro de los parámetros que debe ser correctamente modificado es el tiempo de publicación ya que debe estar acorde a los obtenidos por el dispositivo.

Para poder ejecutar la aplicación realizada se debe generar un *jar* ejecutable (figura 39), para hacer esto debemos dar clic secundario sobre el *proyecto* > *Export* > *java* > *Runnable JAR file* > *next*, se selecciona cual aplicación debe ejecutar, en este caso se requiere que sea la clase Publisher, y se asigna una ruta de destino, se le asigna un nombre el cual puede ser diferente al de la clase, se selecciona la opción *Package required libraries into generated JAR* y oprimir el botón *Finish*. Esto nos dará como producto una *jar* ejecutable el cual podremos correr en la tarjeta.

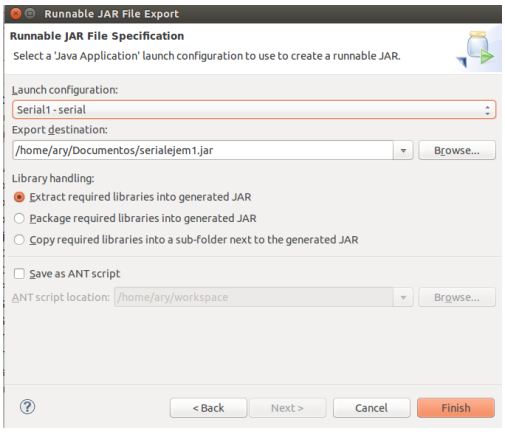

Figura 39. *Exportar Jar ejecutable*

Ejecutando aplicación en la beaglebone black

Necesitamos pasar el archivo *jar* ejecutable a la tarjeta por lo cual conectaremos la beaglebone black al servidor Linux y se ocupará el siguiente comando en la terminal del servidor

Una vez copiado se accede vía *ssh* a la tarjeta, para realizar las configuraciones correspondientes.

```
scp –r /home/usuario/workspace/Publisher.jar ubuntu@192.168.7.2
/home/Ubuntu/lib
```

El primer paso será conectar el dispositivo mediante el puerto serial de la tarjeta y ver como lo identifica la tarjeta.

```
sudo dmesg | grep tty
```

Lo que nos da por resultado lo mostrado en la figura 40. Aquí nos damos cuenta que el único puerto habilitado de la tarjeta es el */dev/ttyS0.*

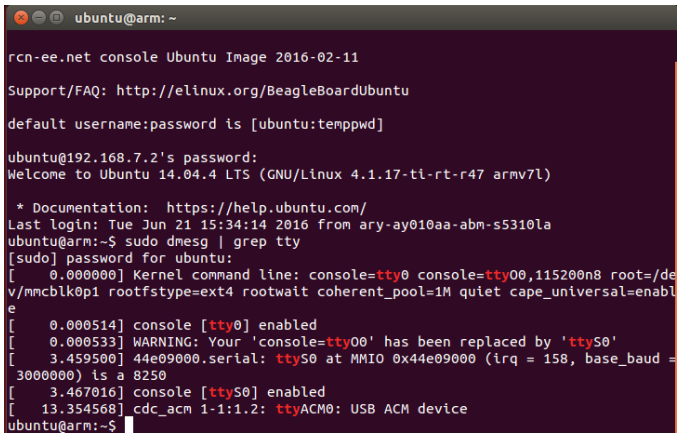

Figura 40. *Puertos disponibles en la tarjeta*

Por esta razón debemos habilitar un puerto más, aquí se ha elegido habilitar el *UART2*. Para este paso es necesario estar con los permisos de administrador.

```
echo BB-UART2 > /sys/devices/platform/bone_capemgr/slots
```

El siguiente comando sirve para verificar la información del archivo (figura 41)

```
cat /sys/devices/platform/bone_capemgr/slots
```

Figura 41. *Contenido del archivo slots*

Para verificar que se haya habilitado el puerto *UART2* ingresemos el siguiente comando, el cual nos dará los puertos habilitados.

```
ls /dev/ttyO*
```

Una vez hechos los pasos anteriores ya se puede salir como administrador.

```
exit
```

Posteriormente ejecutamos el siguiente comando para dar permisos de modificar el identificador del dispositivo.

```
sudo chmod a+rw /dev/ttyACM0
```

Y ejecutamos el siguiente comando para hacer cambio de puertos y que pueda ser válido para la aplicación, ya que como la librería serial solo admite los puertos con id convencionales, pero no los que se identifican de diferente manera como lo es el sufijo ACM.

```
sudo ln –sf /dev/ttyACM0 /dev/ttyS2
```

Para hacer la comunicación entre la beaglebone black y una computadora mediante red Ethernet, debemos identificar la dirección IP asignada a la tarjeta mediante el siguiente comando. Lo que obtendremos se muestra en la figura 42, donde eth0 corresponde a la dirección IP correspondiente al cable Ethernet conectado.

```
ifconfig
```

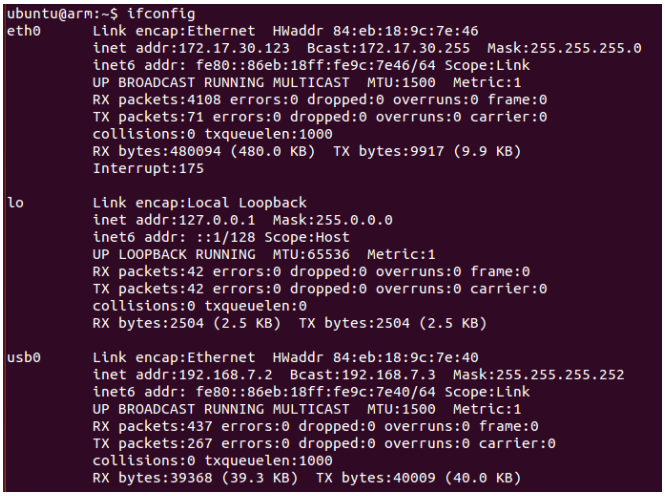

Figura 42. *Dirección IP de la tarjeta*

Una vez que identifiquemos la dirección, haremos un *ping* hacia la dirección *IP* de la computadora a donde queremos que lleguen los datos, para parar la ejecución de este comando se presiona la tecla *ctrl + c*.

```
ping 172.17.30.127
```

Ahora si ya estamos listos para ejecutar la aplicación en la tarjeta, nos moveremos a la ruta donde se guardó el programa.

```
cd /home/ubuntu/lib
```

Y seguido ejecutamos el archivo con el siguiente comando, dando por resultado los datos publicados.

```
java  -jar Publisher.jar
```

Capítulo 5 Resultados

5.1 Introducción

En esta sección se presentan los resultados obtenidos del sistema implementado, de los cuales se pueden destacar la implementación del estándar DDS mediante la distribución de RTI, seguido se obtuvo la conexión en red mediante un switch con las tarjetas beaglebone black y las computadoras. También se puede recalcar la conectividad de varios dispositivos obteniendo conexión plug and play. Todos estos resultados se integran bajo la interfaz de IUM-DDS correspondiente al sistema, que muestra los datos obtenidos de los dispositivos médicos en tiempo real.

5.2 Interfaz gráfica

En la figura 43 se muestra la interfaz principal del sistema, ésta consta de 4 botones y una característica primordial es que puede ser ampliada a muchos dispositivos sin cambiar la estructura que ya fue determinada.

Los botones que inician con la palabra publicador tienen la función principal de publicar los datos procedentes de los dispositivos médicos y transmitirlos bajo un mismo dominio y tema. En este momento solo se corroboró esta parte de la aplicación en memoria compartida es decir en la misma computadora, ya que en la configuración actual de los dispositivos médicos no pueden recibir información.

Los botones con inicio de la palabra suscriptor tienen la función de suscribirse y leer los datos correspondientes al dispositivo de interés. Para la implementación de la comunicación mediante este botón fue configurado el tema y dominio correspondiente en el que fueron publicados los datos del dispositivo para poder recibirlos, posteriormente se envían a las clases controladoras para que se puedan visualizar los datos en pantalla ya sean datos estáticos o gráficas en tiempo real.

Figura 43. *Interfaz principal*

Posteriormente se realizó la prueba de memoria compartida para esto se habilitaron los dos botones uno ejecutando el publicador y el otro el suscriptor bajo la misma computadora, y las pantallas de los resultados son mostrados en la figura 44, respectivamente a las aplicaciones correspondientes.

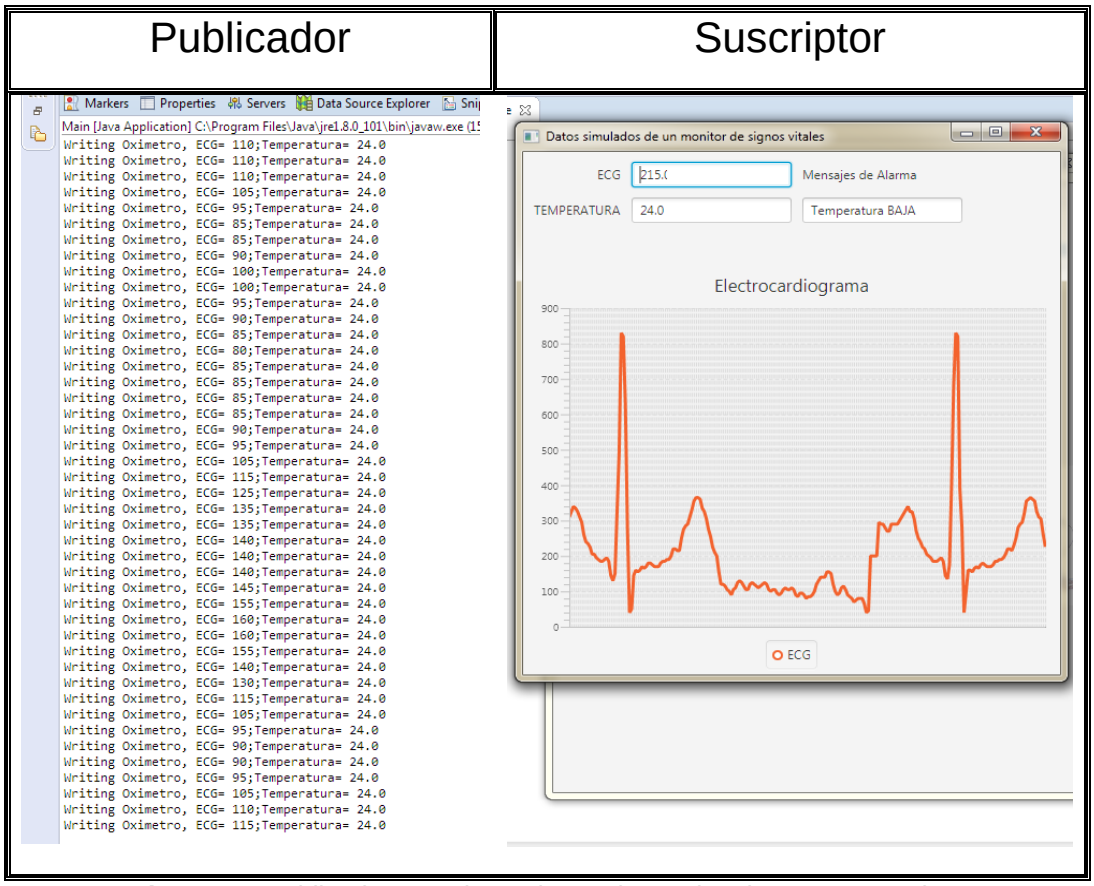

Figura 44. *Publicador-suscriptor ejecutados en la misma computadora*

5.3 Sistema en red

Una vez obtenidos los programas correspondientes a la tarjeta beaglebone Black y al IUM-DDS y siendo instalados como se vio en el capítulo 4, sólo queda ver la conexión en red la primera que se realizó puede observarse en la figura 45, donde se conectan dos computadoras, una funcionando como adaptador del dispositivo y la cual ejecuta la aplicación “publicador” y la otra ejecutando la aplicación “IUM-DDS”, la cual se suscribe a los datos ya publicados de la otra computadora y los muestra gráficamente en la interfaz. La conectividad de la red se probó en la figura 46.

En la figura 47 se muestran los datos del publicador de la primera computadora y se presenta la interfaz gráfica en la segunda computadora correspondiente al oxímetro de pulso, en esta pantalla se muestran los valores de saturación de oxígeno los cuales están en una escala de 0 a 100% y el ritmo cardiaco el cual es proporcionado en latidos por minutos. En el lado derecho se observa el espacio para la visualización de la alarma obtenida del dispositivo médico, por ultimo esta pantalla muestra la forma de onda pletismográfica de una persona.

Figura 45. *Conexión en red 1*

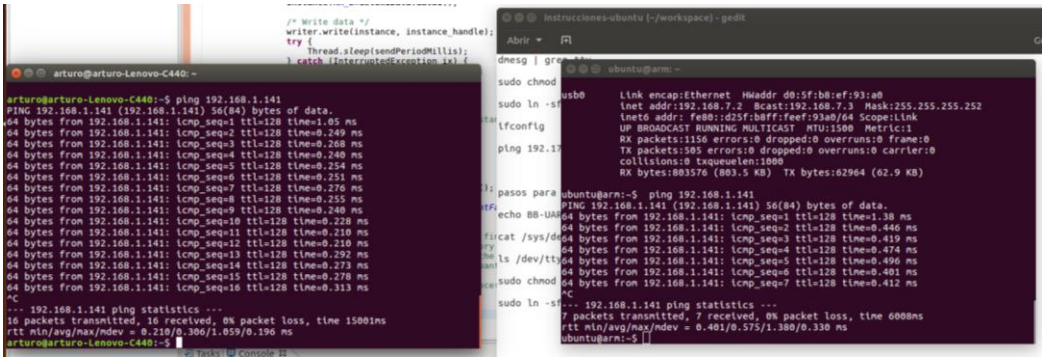

Figura 46. *Conectividad entre puntos de acceso*

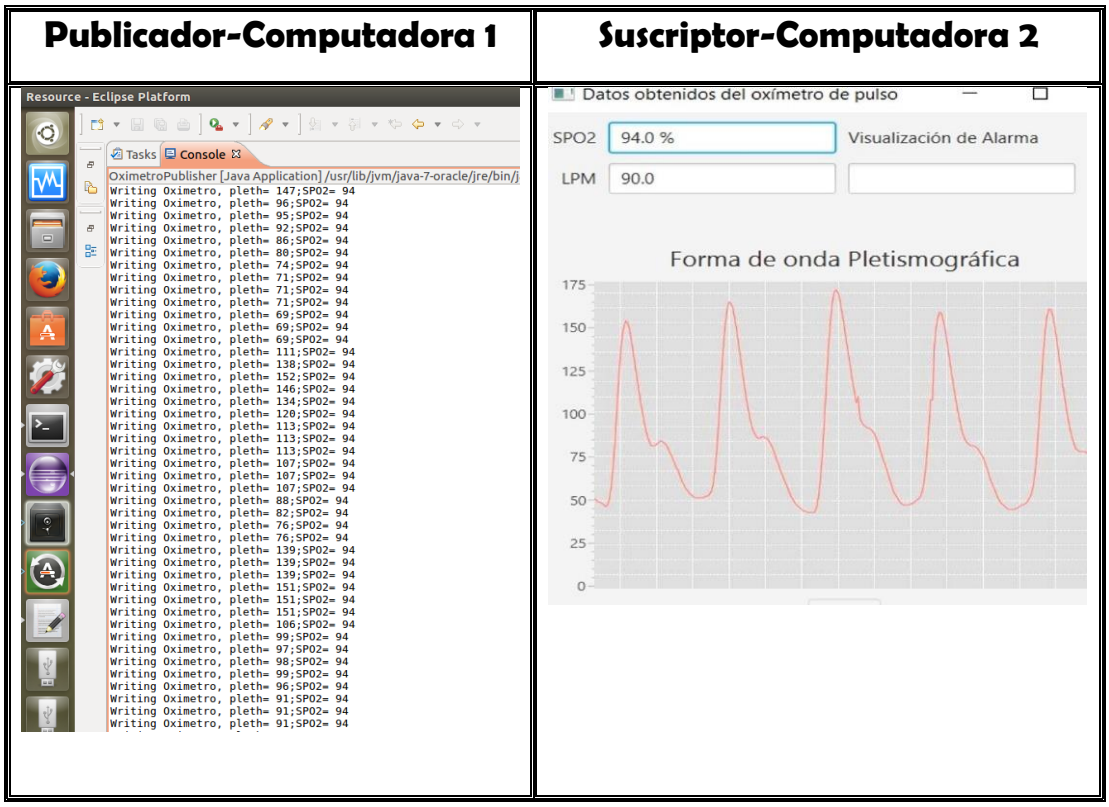

Figura 47. *Publicador-suscriptor en distintas computadoras*

Una vez realizada la red anterior se implementó la misma red, pero ahora con una computadora y una tarjeta beaglebone Black, mostrada en la figura 48, donde también se aprecia una tarjeta núcleo que simula un monitor de signos vitales conectado a la tarjeta beaglebone Black que a su vez es conectada al switch de comunicaciones, en este mismo se conecta vía ethernet a la computadora que recibirá los datos publicados.

En la figura 49 son mostradas las pantallas correspondientes a la tarjeta y a la computadora. También se observan los datos procesados y obtenidos del monitor de signos vitales el cual fue simulado con la tarjeta núcleo F446; se puede observar la interfaz con los parámetros de salida del electrocardiograma y la temperatura con un mensaje de alarma y consecutivamente la gráfica correspondiente al electrocardiograma.

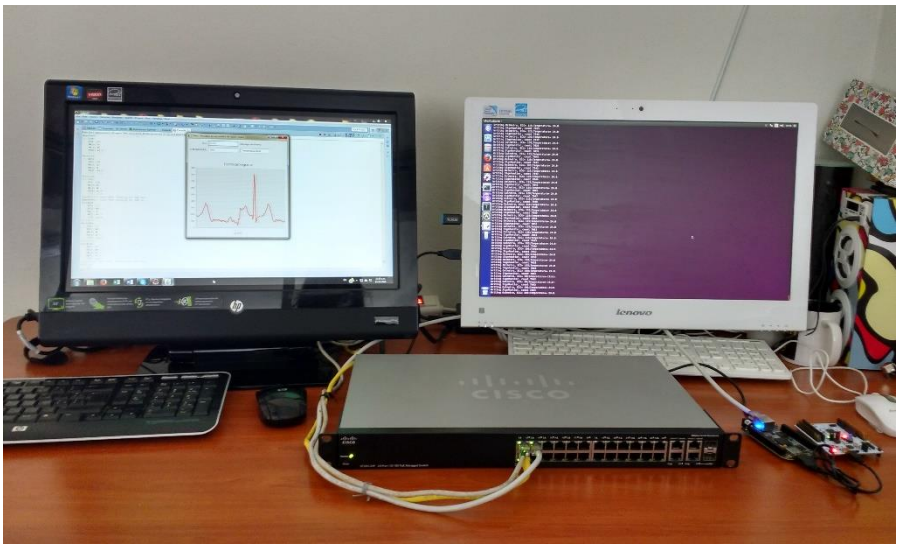

Figura 48. *Conexión en red 2*

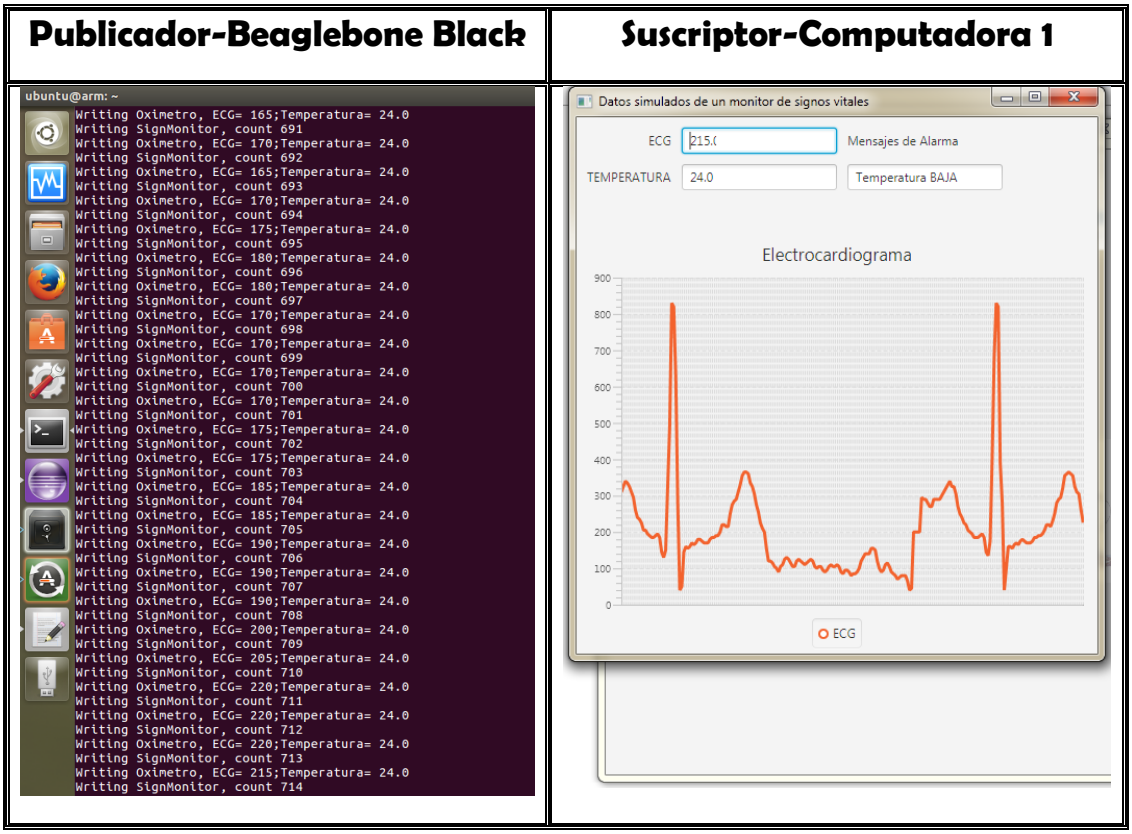

Figura 49. *Publicador-suscriptor /Tarjeta-PC*

Por último, se muestra la conexión realizada para una tarjeta BeagleBone Black y la tarjeta que simula el monitor de signos vitales ambas conectadas a un modem por el cual se transmite la información (figura 50). Cabe señalar que ambas tarjetas son conectadas mediante el puerto serial y la beaglebone black es conectada vía ethernet para transmitir los datos a la interfaz gráfica que se ejecuta en otra computadora conectada a la misma red del modem.

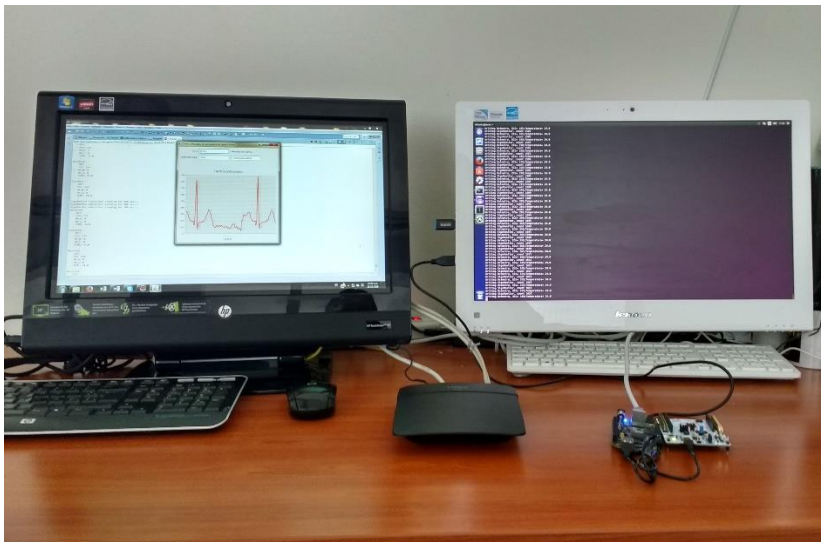

Figura 50. *Conexión de las tarjetas*

Capítulo 6 Conclusiones y trabajo futuro

6.1 Conclusiones

Fueron realizados los modelos de datos de un monitor de signos vitales y un oxímetro de pulso, con lo cual se ve reflejado el resultado del primer objetivo específico.

Se analizaron y diseñaron los diagramas de paquetes, de requerimientos, de casos de uso, de diseño, de secuencia, de clases y de implementación del sistema mediante el software Enterprise architect con el lenguaje SysML. Estos diagramas describen el análisis y operación del sistema con lo cual se ve cubierto el segundo objetivo específico.

Fue desarrollada la interfaz gráfica y la comunicación para una red LAN y/o memoria compartida, mediante el estándar DDS. Esto representa el cumplimiento del tercer objetivo específico.

Posteriormente, se puede mencionar que se conoció a detalle la enorme normatividad que rigen este tipo de sistemas, además se compararon los protocolos de comunicación que existen y con los cuales se desarrolló el sistema.

Conforme a la investigación realizada se puede comentar que existen pocas instituciones a nivel mundial, trabajando en la interoperabilidad de dispositivos médicos. Sin embargo, es preciso mencionar que el estándar DDS

distribuido por RTI es un gran aporte a la falta de interoperabilidad, ya que posee múltiples herramientas y características fundamentales para el soporte de sistemas distribuidos en tiempo real.

6.2 Trabajo futuro

Trabajar variando los dominios correspondientes al estándar DDS, para el control del monitoreo de varios pacientes dentro de un mismo sistema.

Realizar el control a lazo cerrado del sistema, para que haya comunicación de regreso del sistema a los dispositivos médicos, es decir, que si tengo conectada una bomba de infusión pueda enviar los datos del volumen de suministro del medicamento y la velocidad de goteo, de la interfaz al dispositivo.

Trabajar la eficacia de las alarmas y el modo en que éstas se pueden proporcionar.

Dar seguimiento a la ruta regulatoria y migrar el sistema IUM-DDS al sistema operativo QNX ya que es requisito para seguir la norma IEC-62 304, puesto que se menciona que este es específico para sistemas de tiempo real críticos.

Referencias bibliográficas

[1] S. A. Boyer," SCADA: supervisory control and data acquisition", 4rd ed., International Society of Automation, 2009.

[2] M. Reynolds, T. Norgall, y Cooper, T., “State of the art technology- Protocols, Networks and Standards for point -of -care device communication”. 2004. [En línea]. Consultada en junio, 2015. Disponible en: http://www.iso.org/iso/wsc-medtech_23_Melvin_Reynolds.pdf

[3] S. Schlichting y S. Pöhlsen, “An architecture for distributed systems of medical devices in high acuity environments- a proposal for standards adoption”. 2014. [En línea]. Consultada en Junio, 2015. Disponible en: https://www.hl7.org/documentcenter/public_temp_AF8D91DD-1C23-BA17-0C916C024EF34C6A/wg/healthcaredevices/20140116%20An%20architecture%20for%20distributed%20systems%20of%20medical%20devices%20in%20high-acuity%20environments.pdf

[4] J. M. Goldman, “Medical device plug-and-play (MD PnP) interoperability program”. 2012. [En línea]. Consultada en Junio, 2015. Disponible en: http://www.mdpnp.org/uploads/Oct12_MD_PnP_White_Paper.pdf

[5] S. Sclichting, S. Poehlsen y S. Wiley, "SDC update", 2015. [En línea]. Consultada en Julio, 2015. Disponible en: http://www.hl7.org/documentcenter/public_temp_A750610A 1C23-BA17-0C2CE5D6778A21E6/wg/healthcaredevices/20150120%20openSDC%20update.pdf

[6] OMG, “OMG SysML specification”, 2007. [En línea]. Consultada en Julio, 2015. Disponible en: http://sysml.org/docs/specs/OMGSysML-PAS-07-02-03.pdf

[7] H. Galván, “Implementación de un sistema SCADA para la mezcla de dos sustancias en una industria química”, Tesis de maestría, [En línea]. Consultada en Septiembre, 2015. Disponible en: http://itzamna.bnct.ipn.mx/dspace/bitstream/123456789/8555/1/89.pdf

[8] C. Castro y C. Romero, “Introducción a SCADA”, 2006. [En línea]. Consultada en Agosto, 2015. Disponible en: http://www.uco.es/grupos/eatco/automatica/ihm/descargar/scada.pdf

[9] A. Garcia, “Control de procesos automatizados”, 2008. [En línea]. Consultada en octubre, 2015. Disponible en: http://es.slideshare.net/nestorcusco/sistema-scada-24902242

[10] I. Ungurean, N. C. Gaitan y V. G. Gaitan," Transparent interaction of SCADA systems developed over different technologies," in 18th International Conference on System Theory, Control and Computing, Sinaia, Romania, October, pp. 476-481, 2014.

[11] RASPBERRY PI FOUNDATION, "RASPBERRY PI", 2009. [En línea]. Consultado en Noviembre, 2015. Disponible en: https://www.raspberrypi.org/

[12] BeagleBoard Foundation, "BeagleBone Black", 2014. [En línea]. Consultado en Noviembre, 2015. Disponible en: http://beagleboard.org/BLACK

[13] S. Ramirez, "Tutorial de IDL básico", Junio 2001. [En línea]. Consultado en Noviembre, 2015. Disponible en: http://www.sw.cl/~sram/manuals/idl.pdf

[14] J. López, J. Povedano, J. Sánchez y J. M. López," Políticas de QoS en una Plataforma de Trabajo Colaborativo sobre Middleware DDS", XIII Jornadas de Tiempo Real (JTR2010), 2010

[15] W. Savitch. Java an introduction to computer science & programming. 3rd. Ed. Pearson: Prentice Hall, 2004

[16] P. Tejera y A. Alonso," Middleware de Comunicación para Sistemas Distribuidos de Tiempo Real en Java", 2007, [En línea]. Consultado en Noviembre, 2015. Disponible en: http://www.ctr.unican.es/jtr12/articulos/13-daniel-tejera.pdf

[17] J. Estrada," Comparativo de herramientas middleware para el desarrollo de aplicaciones distribuidas bajo el paradigma de la programación orientada a componentes," Tesis de maestría, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, México, D.F. 2008

[18] P. Pietzuch y J. Bacon, "Hermes: a distributed event-based middleware architecture," in Proceedings. 22nd International Conference on Distributed Computing Systems Workshops, pp.611-618, 2002

[19] L. Zhang, y J. Peng, "A platform Based on CORBA for Open Networked Measurement System, in IEEE circuits and systems international conference on Testing and Diagnosis, ICTD, Chengdu, pp. 1-5, 28-29 abril ,2009

[20] S. Pickin, "Introducción al modelo de componentes de CORBA: motivación y visión general", 2004. [En línea]. Consultada en Noviembre, 2015. Disponible en: http://www.it.uc3m.es/mcfp/docencia/si/material/10_ccm.pdf

[21] J. M. Drake, "Sistemas distribuidos de tiempo real", 2009. [En línea]. Consultada en octubre, 2015. Disponible en: http://www.ctr.unican.es/asignaturas/procodis_3_II/Doc/Procodis_8_01.pdf

[22] Oracle, "Tutorial RMI", 2011. [En línea]. Consultada en Noviembre, 2015. Disponible en: https://docs.oracle.com/javase/tutorial/rmi/

[23] OMG, ”Portal”, 2014. [En línea]. Consultada en Septiembre, 2015. Disponible en: http://portals.omg.org/dds/where-can-i-get-dds/

[24] OMG, “Extensible and Dynamic Topic Types for DDS”, U.S.A. standard, Noviembre, 2014.

[25] G. Pardo, “DDSTutorial—Part II”, 2009. [En línea]. Consultado en Septiembre, 2015. Disponible en: http://www.omg.org/news/meetings/GOV-WS/pr/rte-pres/DDS_Tutorial_RTEW09.pdf

[26] J. L. Poza, “Revisión de los Sistemas de comunicaciones más empleados en control distribuido”, 2009. [En línea]. Consultada en octubre, 2015. Disponible en: https://riunet.upv.es/bitstream/handle/10251/6408/Comunicaciones%20en%20los%20sistemas%20distribuidos.pdf

[27] Nonin support, “Pulse oximeters and sensors”, 2016, [En línea]. Consultada en enero, 2016. Disponible en: http://www.nonin.com/Manuals-IFUs

[28] PhysioNet, “The research resource for physiologic signals”, 2009, [En línea]. Consultada en febrero, 2016. Disponible en: https://www.physionet.org/

[29] ASTM International, ASTM F2761-Medical Devices and Medical Systems - Essential safety requirements for equipment comprising the patient-centric integrated clinical environment (ICE), 2009.

[30] R. M. Hofmann, “Modeling medical devices for plug-and-play interoperability”, Tesis de maestría, MIT, Junio, 2007.

[31] ISO/IEEE, ISO/IEEE11073-x, Medical Health Device Communication Standards Family, 2004.

[32] J. Plourde, D. Arney y J. Goldman,"OpenICE: An open, interoperable platform for medical cyber-physical systems", ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS, Berlìn, pág. 221, 14-17 abril,2014

[33] CIMIT, “OpenICE”, 2014, [En línea]. Consultada en Agosto, 2015. Disponible en: https://www.openice.info/

[34] S. Schlichting y Stephan Pöhlsen,” An architecture for distributed systems of medical devices in high acuity environments “, 2014. [En línea]. Consultada en Agosto, 2015. Disponible en file:///C:/Users/ary/Downloads/TechnicalWhitepaper.pdf

[35] Dräger, “OpenSDC”, 2014. [En línea]. Consultada en Agosto, 2015. Disponible en: http://sourceforge.net/projects/opensdc/

Descargas de softwares

[36] http://www.rti.com/downloads/index.html

[37] https://community.rti.com/downloads/rti-connext-dds-raspberry-pi

[38] http://www.realvnc.com/download/get/1866/eula

[39] http://rxtx.qbang.org/wiki/index.php/Download

[40] https://www.microsoft.com/es-mx/download/details.aspx%3Fid%3D23691

[41] http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

[42] http://www.jfree.org/jfreechart/download.html

[43] http://www.eclipse.org/downloads/packages

[44] http://putty.softonic.com/

[45] https://rcn-ee.online/rootfs/2016-02-11/microsd/