VISUALIZADOR DE LA INTERFAZ HOMBRE-MAQUINA DEL SISTEMA ROFLEXIN/LC PARA DISPOSITIVOS MOVILES CON ANDROID.

Ing. Luis Andrés Valido Fajardo[1]

1. Universidad de Matanzas – Sede "Camilo Cienfuegos"- Departamento de Desarrollo de Recursos para Aprendizaje, Vía Blanca Km.3, Matanzas, Cuba. luis.valido@umcc.cu

Resumen

El Centro de Estudios de Fabricación Avanzada y Sostenible (CEFAS) de la Universidad de Matanzas desarrolla el proyecto ROFLEXIN/LC (Robusto, flexible e inteligente y de bajo costo) un sistema de monitoreo de bajo costo para procesos y sistemas mecánicos.

La presente investigación surge a partir de la necesidad de los directivos, supervisores y personal que utilizan el sistema ROFLEXIN/LC, de visualizar la información recopilada por dicho sistema en dispositivos móviles y *tablets*. El objetivo fundamental es desarrollar una aplicación para móviles capaz de visualizar en tiempo real la información del sistema ROFLEXIN/LC. Para el desarrollo de la solución propuesta se realiza un análisis de las principales herramientas, tecnologías y metodologías que se utilizan en la construcción de un software. El proceso estuvo guiado por el uso de las siguientes herramientas y tecnologías: *Visual Paradigm* como herramienta CASE, UML como lenguaje de modelado, JSON como formato de intercambio de datos y Java como lenguaje de programación. Desarrollado con el IDE Eclipse con la extensión ADT. Para validar que los resultados alcanzados fueron los esperados se realizó un conjunto de pruebas: pruebas de aceptación, de compatibilidad, de usabilidad, de campo, de rendimiento y pruebas de satisfacción de usuarios. Se logró corregir todas las no conformidades detectadas lo que significa que la solución propuesta está lista para ser desplegada. Finalmente se obtuvo un Visualizador para dispositivos móviles y *tablets* con sistema operativo con Android capaz de representar en tiempo real la información proveniente del sistema ROFLEXIN/LC.

***Palabras claves**: Android, Interfaz Hombre-Máquina, Sistema de monitoreo, Visualizador*

ISBN: 978-959-16-4235-6

Introducción

En poco tiempo las Tecnologías de la Información y las Comunicaciones (TIC) han evolucionado considerablemente, hasta el punto de volverse casi imprescindible para algunos sectores de nuestra sociedad. La automatización de los procesos industriales se ha convertido en un gran avance para el sector empresarial, con su uso, se han revolucionado las producciones a diferentes escalas, reduciendo el peligro de catástrofes, además de obtener grandes ahorros en tiempo, dinero y recursos.

Dentro de esa gama de productos que han revolucionados se encuentra los sistemas de monitoreo los cuales contribuyen significativamente al aumento de la productividad y la calidad, además de garantizar una mayor seguridad para el equipamiento y los operadores de procesos. Sin embargo, los altos costos de estos sistemas, junto a la necesidad de importarlos, hacen prohibitiva su aplicación, especialmente para pequeñas y medianas empresas.

La informatización de la sociedad cubana es una voluntad política del gobierno que está expresada en los Lineamientos de la Política Económica y Social, aprobados en el Sexto Congreso del Partido Comunista de Cuba (2011). En varios sectores de la industria y la economía cubana se necesita un sistema robusto, flexible y de bajo costo para el monitoreo de sistemas y procesos mecánicos, que sea posible de implementar con un mínimo de componentes importados. El Centro de Estudios de Fabricación Avanzada y Sostenible (CEFAS) de la Universidad de Matanzas vinculado al Programa Nacional de Ciencia, Tecnología e Innovación desarrolla el proyecto ROFLEXIN/LC un sistema de monitoreo de bajo costo para procesos y sistemas mecánicos.

Dicho sistema está compuesto por varios módulos que interactúan entre sí, comenzando el proceso por los controladores de dispositivos y finalizando en la Interfaz Hombre-Máquina (HMI).

El HMI está compuesto por un ambiente de edición o editor, que es donde se realizan las configuraciones para el monitoreo, y el ambiente de ejecución o visualizador, que permite al operador supervisar el sistema o los procesos mecánicos.

El ROFLEXIN/LC está orientado al trabajo en estaciones fijas como una aplicación de escritorio, el cual debe ser instalado sobre el sistema operativo de cada ordenador destinado a la monitorización de los procesos.

Las condiciones planteadas anteriormente provocan que los directivos, supervisores y el personal autorizado de la empresa, no puedan visualizar la información de los procesos en tiempo real mediante el Ambiente de Ejecución del HMI en estaciones de trabajo que no tengan instalado el sistema, limitando la disponibilidad de la aplicación. Esto ocasiona que los directivos necesiten trasladarse hacia las oficinas para poder supervisar los procesos,

ISBN: 978-959-16-4235-6

estando éstas en muchas ocasiones alejadas de las entidades administrativas, provocando además gastos innecesarios.

Teniendo en cuenta la situación planteada con anterioridad se define el siguiente problema de investigación: ¿Cómo lograr la disponibilidad del visualizador del HMI del ROFLEXIN/LC en los dispositivos móviles? En concordancia con lo anterior se propone como objetivo general: Desarrollar una aplicación para dispositivos móviles que permita la disponibilidad del visualizador del HMI del ROFLEXIN/LC.

Materiales y métodos o Metodología computacional

Conceptos asociados a la investigación

ROFLEXIN/LC: Es un sistema de monitoreo de bajo costo para procesos y sistemas mecánicos. El mismo se basa en la utilización de tecnologías de hardware más baratas que las convencionales con un software desarrollado con herramientas libres y código abierto bajo una arquitectura abierta que solo use aquellos módulos que sean necesarios en dependencia de las necesidades de clientes. Todo esto hace que sea un proyecto que este dirigido fundamentalmente aunque no exclusivamente, a pequeñas y medianas empresas, donde el uso de las alternativas disponibles en el mercado mundial no es financieramente factible.

La arquitectura que se propone para la conexión del hardware se muestra en la figura, la misma está compuesta por un sistema de acondicionamiento de señal que está formado por una placa con elementos circuitales que trasforman la salida de los sensores conectados al sistema, señal la cual se conecta al conversor análogo-digital (ADC) el cual se encuentra conectado por el bus i2c a la Raspberry Pi, la cual se encarga de la recolección de los datos para su envió a través de un interface *ethernet* que se encuentra conectada a un servidor en el cual una aplicación de escritorio recepciona los datos y brinda opciones de visualización de gráficos de históricos y tendencias, y permite además la configuración del hardware de manera remota, así como la obtención de modelos del proceso a través de la utilización de algoritmos de modelación basados en elementos de inteligencia artificial. Además, dicho servidor también brinda la posibilidad de realizar la supervisión vía web mediante un servidor web con el cual se pueden comunicar todos los ordenadores conectados a su red a través del protocolo de trasferencia de hipertexto (http).

ISBN: 978-959-16-4235-6

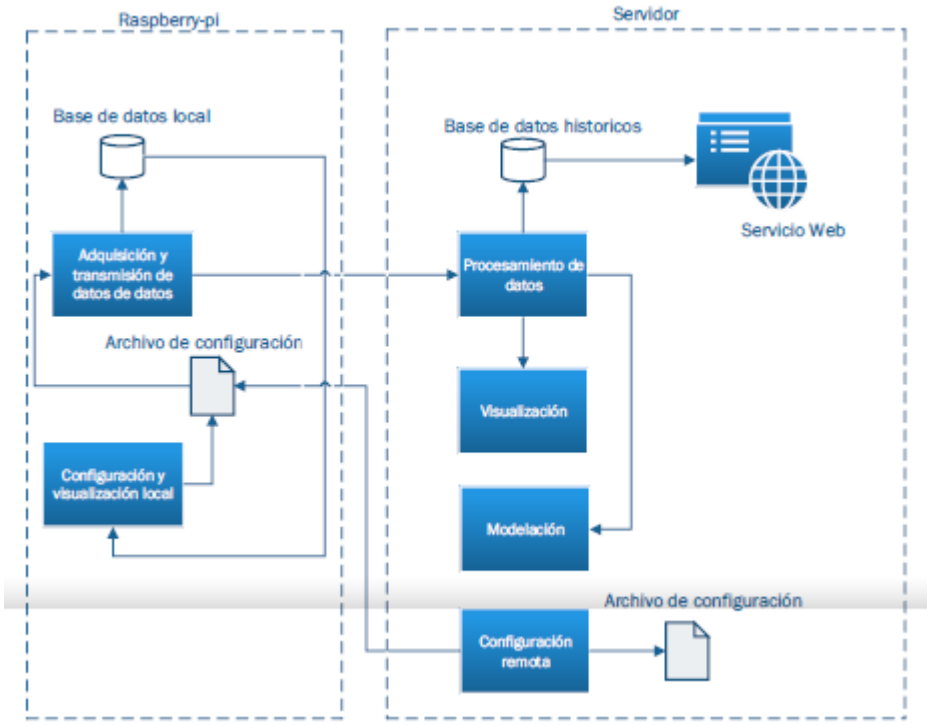

Figura 1. Arquitectura de conexión del hardware.

Soluciones de software analizadas

Slicetex Virtual HMI (Virtual HMI): Virtual HMI es un panel HMI (Interfaz Hombre-Máquina) virtual que le permite controlar el PLC remotamente y al mismo tiempo visualizar mensajes provenientes del PLC en un dispositivo Android con conexión a la red Ethernet. La aplicación Virtual HMI en conjunto con una *tablet*, puede ser una alternativa económica frente a los terminales HMI *touch* existentes en el mercado. Está diseñada para mostrar múltiples configuraciones visuales prediseñadas. Cada pantalla se llama panel. Se puede elegir diferentes paneles de acuerdo a su aplicación. Por ejemplo un panel que combine *display* LCD y teclas, o un panel que solo contenga teclas o barras deslizantes.

Iconics' MobileHMI Brings HMI/SCADA: El nuevo *MobileHMI* de Iconics lleva la potencia de las soluciones GENESIS64 HMI / SCADA a cualquier teléfono inteligente, tableta, navegador web o dispositivo habilitado para la web. La innovadora tecnología *Smart Tile*, *AppHub* ™ y la configuración ayudan a simplificar la implementación en cualquier dispositivo Microsoft, Apple o Android a través de la innovadora tecnología WinRT y XAML-to-HTML5 para entregar soluciones HMI/SCADA

EasyAccess 2.0: Es una herramienta de acceso remoto para su máquina o HMI industrial. Le permite monitorear o actualizar los controladores o proyectos conectados de HMI. Cuenta con su versión para dispositivos móviles con sistema operativo Android con una

ISBN: 978-959-16-4235-6

versión 4.1 o superior. Aunque las funcionalidades en un dispositivo Android y su interfaz de usuario pueden ser ligeramente diferentes a las de la PC.

EZ RMC Remote HMI App: La aplicación EZ RMC Remote HMI es una aplicación diseñada para que sus dispositivos Android habiliten la supervisión y el control de sus HMI EZTouch desde EZAutomation.net. Esta La aplicación permite el acceso directo y total a su HMI EZTouch. Permite la visualización de los datos como hojas de cálculo, gráficos de barra o de líneas.

TeslaSCADA2 Runtime: Es el ambiente de visualización para proyectos desarrollados en TeslaSCADA IDE. El mismo requiere del sistema operativo Android en su versión 3.0 o superior. Cuenta con una interfaz gráfica para la configuración de la herramienta.

Métodos teóricos

Analítico-sintético: Para el estudio de los conceptos empleados en los sistemas de monitoreo desde dispositivos móviles, analizando todos los documentos elaborados, para la extracción de los elementos más importantes.

Histórico-lógico: Para la comprensión de los antecedentes y las tendencias actuales referidas la evolución en el mundo de los sistemas de monitoreo desde dispositivos móviles.

Métodos empíricos

Consulta bibliográfica: Empleada para consultar las fuentes de información relacionados con los tipos de los sumarios y gráficos visualizados en los entornos móviles de los HMI en sistemas de monitoreo.

Observación: Se puso en práctica este método para conocer el funcionamiento existente en los despliegues del ROFLEXIN/LC mediante el comportamiento de los dispositivos de campo en las propiedades de los gráficos y sumarios empleados para la toma de decisiones de los operadores. Se emplea para estudiar las características y comportamientos de las soluciones similares, permitiendo obtener información relevante sobre el funcionamiento de dichas soluciones.

Metodología de desarrollo de software: *Extreme Programming*

Con una buena metodología se pretende reducir costos y retrasos de proyectos, así como mejorar la calidad del software. *Extreme Programming* o Programación Extrema es una de las llamadas metodologías ágiles de desarrollo de software más exitosas y controversiales de los tiempos recientes. *Extreme Programming* (XP) surge como una nueva manera de encarar proyectos de software, proponiendo una metodología basada esencialmente en la simplicidad y agilidad (Beck, 2002). La metodología propuesta en XP está diseñada para

ISBN: 978-959-16-4235-6

entregar el software que los clientes necesitan en el momento en que lo necesitan. Si bien el ciclo de vida de un proyecto XP es muy dinámico, se puede separar en fases.

- **Fase de exploración**. Es la fase en la que se define el alcance general del proyecto. En esta fase, el cliente define lo que necesita mediante la redacción de sencillas historias de usuarios. Los programadores estiman los tiempos de desarrollo en base a esta información.

- **Fase de planificación**. La planificación es una fase corta, en la que el cliente, los gerentes y el grupo de desarrolladores acuerdan el orden en que deberán implementarse las historias de usuario, y, asociadas a éstas, las entregas.

- **Fase de iteraciones**. Esta es la fase principal en el ciclo de desarrollo de XP. Las funcionalidades son desarrolladas en esta fase, generando al final de cada una un entregable funcional que implementa las historias de usuario asignadas a la iteración.

- **Fase de puesta en producción**. Si bien al final de cada iteración se entregan módulos funcionales y sin errores, puede ser deseable por parte del cliente no poner el sistema en producción hasta tanto no se tenga la funcionalidad completa.

Lenguaje de Modelado: UML

UML (*Unified Modeling Language*) es desde finales de 1997 un lenguaje de modelado visual que se utiliza para especificar, visualizar, construir y documentar artefactos de un sistema de software (Rumbaugh, 2007).

Es válido destacar que UML es un lenguaje de modelado, no un método o un proceso. UML constituye un lenguaje más expresivo, claro y uniforme que los anteriores definidos para el diseño Orientado a Objetos, no brinda el total éxito de los proyectos, pero si mejora sustancialmente el desarrollo de los mismos al posibilitar una nueva y fuerte integración entre las herramientas, los procesos y los dominios.

Herramienta CASE: *Visual Paradigm*

Herramienta CASE (del inglés, *Computer Aided Software Engineering*) con licencia gratuita, que propicia un conjunto de ayudas para el desarrollo de programas informáticos dando soporte al modelado visual con UML (del inglés, *Unified Modeling Language*), desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación (Oscar, 2013).

JSON

ISBN: 978-959-16-4235-6

JSON (*JavaScript Object Notation*) es un formato ligero para el intercambio de datos. Es sencillo de leer y de escribir así como de interpretar por máquinas. Está basado en un subconjunto de JavaScript. El formato de JSON es ampliamente reconocido por una gran variedad de lenguajes como Java, PHP, JavaScript, C++, C# y muchos otros. Esto hace que JSON sea un lenguaje de gran potencial para el intercambio de datos (Academy, 2017) (Keller, 2016).

Lenguaje de programación: Java

Las aplicaciones nativas para la plataforma Android se desarrollan utilizando el lenguaje de programación Java. Java es un lenguaje de programación orientado a objetos, desarrollado por *Sun Microsystems* a principios de los años 90. El lenguaje es muy parecido en sintaxis a C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel como la manipulación directa de punteros a memoria. La memoria es gestionada mediante un recolector de basura lo que aísla al programador de la necesidad de hacer liberaciones explícitas de la memoria asignada dinámicamente y, de esta manera, contribuye a eliminar las fugas de memoria que comúnmente afectan a programas desarrollados en C y C++ .Una de las características más peculiares de este lenguaje es la posibilidad que ofrece de crear aplicaciones independientes a la plataforma donde va a ser ejecutada (Goytia, 2014).

Entorno de desarrollo: Eclipse

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para *VisualAge*. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios (Burd, 2004).

Eclipse es un entorno de desarrollo compuesto por herramientas de programación de código abierto multiplataforma. Se le añaden extensiones para programar en Java por ejemplo *Java Development Tools*, *JDT* (Gallardo, 2003), o, en el caso de Android (*Android Development Tools, ADT*). Bundle ADT contiene los componentes esenciales de desarrollo Android. Diseñado para ampliar las características de Eclipse y permitir configurar y codificar nuevos proyectos Android.

ADT amplía las capacidades de Eclipse para que pueda configurar rápidamente nuevos proyectos de Android, se crea una interfaz de usuario de aplicación, se agregan los paquetes basados en la API de Android, se depuran las aplicaciones utilizando las herramientas del SDK de Android, e incluso permite exportar el archivo *apk* con firma con el fin de distribuir la aplicación.

Android SDK

ISBN: 978-959-16-4235-6

El Android SDK *Android Software Development Kit*}, contiene herramientas necesarias y gratuitas para el desarrollo Android en Eclipse. Combinado con el Bundle ADT antes mencionado forman la combinación perfecta para este tipo de desarrollo.

Requisitos funcionales

Los requisitos funcionales son funciones del sistema de software o sus componentes. Cada función se describe como un conjunto de entradas, comportamientos y salidas. Los requisitos funcionales pueden ser: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que se supone, un sistema debe cumplir. Son complementados por los requisitos no funcionales, que se enfocan en cambio en el diseño o la implementación. Establecen los comportamientos del sistema (Sommerville, 2015).

Por lo anteriormente explicado se definen como requisitos funcionales:

1. Iniciar la aplicación.
    a. Comprobar conexión del dispositivo.
    b. Cargar configuración definida para la terminal.
2. Visualizar datos de la terminal.
3. Visualizar fecha y hora actual.
4. Visualizar estado actual de la batería del dispositivo.
5. Interacción con el módulo de Seguridad.
    a. Autenticar terminal.
    b. Cerrar sesión.
6. Visualizar sumario de variables.
7. Visualizar detalles de una variable.
8. Visualizar comportamiento de una variable en un gráfico de tendencia.
9. Visualizar gráfico de tendencia.
    a. Adicionar serie a la gráfica.
    b. Eliminar serie de la gráfica.
    c. Mostrar u ocultar serie de la gráfica.
    d. Limpiar gráfico de tendencia.
    e. Invertir los colores de texto y fondo de la gráfica.
    f. Exportar el gráfico a formato de imagen.
    g. Mostrar u ocultar leyenda de las series representadas en la gráfica.
    h. Editar propiedades de una serie.
10. Visualizar estado actual de conexión del dispositivo.

Requisitos no funcionales

Los requerimientos no funcionales son requisitos que imponen restricciones en el diseño o la implementación. Son propiedades o cualidades que el producto debe cumplir. Sin embargo, estos requisitos no definen el éxito del producto, pero influyen considerablemente

ISBN: 978-959-16-4235-6

en la evaluación del mismo. Teniendo en cuenta las características del sistema se definieron los siguientes requerimientos no funcionales:

Interfaz: Clara y concisa. No debe dar lugar a la confusión del usuario y debe seguir los estándares de diseño de interfaces de Google. Las interfaces visuales del sistema deben ser diseñadas para ser visualizadas en modo *landscape*. La interfaz de usuario del sistema deberá ser diseñada de forma tal que permita el aprovechamiento del espacio. En la pantalla principal se tendrá acceso a la mayoría de las funcionalidades con que contará el sistema de manera sencilla y con un solo clic. Los componentes Android de interfaz de usuario para la entrada de datos, de ser posible, deberán ser configurados de manera que el riesgo de entrada de datos inválidos por parte del usuario disminuya.

Estabilidad: El sistema debe ser capaz de manejar los errores ocurridos durante la ejecución de la misma y avisando de la naturalidad del error.

Rendimiento: El sistema debe desempeñar su función de una manera fluida. Se debe buscar la experiencia de uso más agradable para el usuario.

Optimización: El consumo de batería y de datos debe ser adecuado, y nunca dejar procesos sueltos que consuman memoria y batería. El tiempo de ejecución debe ser mínimo, para mejorar los tiempos de respuesta y la experiencia de uso del usuario.

Usabilidad: El sistema deberá visualizarse correctamente en cualquier dispositivo Android con las siguientes dimensiones de pantalla: pequeña, normal o grande. La interfaz visual del sistema debe ser atractiva y sencilla, permitiendo al usuario facilidad de uso y entrenamiento. El sistema deberá ser soportada en la mayor cantidad posible de versiones del SO Android. El sistema deberá ser capaz de brindar información al usuario mediante mensajes, para ayudarlo y guiarlo durante su interacción con la aplicación.

Ayuda y documentación: Se brindarán manuales de ayuda que documenten cómo trabajar de forma adecuada con el sistema.

Arquitectura

La selección de un patrón arquitectónico es una decisión fundamental de diseño en el desarrollo de un sistema de software ya que provee un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos. En el desarrollo de un software, se hace necesario seleccionar diferentes patrones los cuales ayudan a que esté presente una buena estructura y organización que hace eficiente su funcionamiento. Estos patrones son una guía para cometer una determinada acción. Especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones, para organizar los distintos componentes.

ISBN: 978-959-16-4235-6

En el caso de la solución propuesta en este documento vamos a partir de que forma parte de una arquitectura cliente servidor tal y como se muestra en la figura.

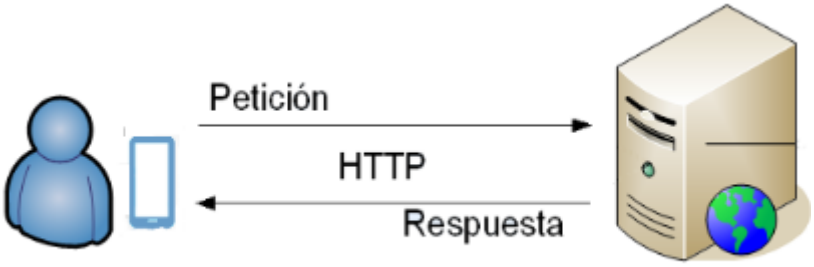

Figura 2. Arquitectura cliente servidor

La arquitectura cliente servidor es una arquitectura de aplicación distribuida en la que un prestador de servicio, el servidor, responde a las peticiones de los clientes. Una máquina servidora es una máquina que está ejecutando uno o más programas servidores que comparten sus recursos con los clientes. Un cliente no comparte sus recursos sino que solicita al servidor su contenido o servicio. Por tanto son los clientes quienes inician el proceso de comunicación. En nuestro caso el Visualizador constituirá la parte cliente. El mismo presenta una arquitectura basada en capas.

Arquitectura N-Capas

La arquitectura N-Capas se enfoca en dividir una aplicación en capas separadas que desempeñan diferentes roles y funcionalidades. Su principal objetivo es separar la lógica del negocio de la lógica de diseño. Una aplicación N-Capas tradicional incluye la capa de presentación, la capa de negocios y la capa de datos. Al estar los componentes separados y localizados, es más sencillo darle mantenimiento al software. Además, la capacidad de realizar pruebas mejora considerablemente debido a que las capas solo interactúan entre ellas mediante interfaces bien definidas y es posible añadir implementaciones alternativas a cada capa.

En el caso del sistema el mismo se divide en dos capas tal y como se muestra en la figura. Una primera capa Aplicación donde va estar todo lo referente a la lógica de negocio definida para el sistema, así como las entidades que sirven de apoyo para esto. Mientras en una segunda capa esta Comunicación capa encargada de permitir la comunicación entre las entidades que componen la capa de Aplicación y el servidor HMI, con el uso de interfaces bien definidas en la capa de Comunicación.

ISBN: 978-959-16-4235-6

Figura 3. Arquitectura N-Capas en el sistema.

Patrones de diseño

Los patrones de diseño expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas de software. La utilización de patrones de diseño, permite ahorrar grandes cantidades de tiempo en la construcción de software. El software construido es más fácil de comprender, mantener y extender. Estos se pueden clasificar en: (Gamma, 2003)

Comportamiento: Los patrones de comportamiento están relacionados con los algoritmos y la asignación de responsabilidades entre los objetos. Son utilizados para organizar, manejar y combinar comportamientos.

Estructurales: Los patrones estructurales se ocupan de cómo las clases y objetos se combinan para formar grandes estructuras y proporcionar nuevas funcionalidades.

Creacionales: Los patrones creacionales abstraen el proceso de creación de instancias y ocultan los detalles de cómo los objetos son creados o inicializados.

Para la implementación de la solución expuesta en este trabajo se hará uso de un grupo de estos como son:

Instancia Única (*Singleton*): Proporciona una forma de agrupar el código en una unidad lógica, que se puede acceder a través de una sola variable. Al asegurar que sólo existe un objeto único, se sabe que todo el código hace uso de los mismos recursos globales. El sistema desarrollada hace uso de este patrón específicamente en la clase *ControllerRequest* encargada de controlar las peticiones que se le realiza al servidor (Gamma, 2003).

Cajón de navegación (*Navigation Drawer*): Con el fin de conseguir una aplicación intuitiva se va a implementar el patrón de diseño *Navigation Drawer*. Es un panel lateral que se desliza desde el borde izquierdo de la pantalla y muestra las principales opciones de navegación de la aplicación (Smyth, 2017). El *Navigation Drawer* es una vista que actúa como menú para cambiar entre los distintos *Fragments* o pantallas de la aplicación. El *drawer* se oculta automáticamente cuando el usuario está usando la aplicación, por lo que permite a los *Fragments* ocupar el mayor tamaño de pantalla posible.

Modelo Vista Controlador: El patrón de diseño MVC es uno de los patrones existentes más citados. Desde su creación ha jugado un papel influyente en la mayoría de los

ISBN: 978-959-16-4235-6

frameworks de interfaz de usuario y en la manera de pensar acerca del diseño de la interfaz de usuario. Este patrón define tres componentes principales: el Modelo, la Vista y el Controlador. A continuación se describen brevemente la funcionalidad de cada componente (Sommerville, 2015) (Fowler, 2012):

- **El Modelo:** Maneja el sistema de datos y las operaciones asociadas a dichos datos.
- **La Vista:** Representa la visualización del Modelo. Define y maneja como los datos serán mostrados al usuario.
- **El Controlador:** Maneja las interacciones del usuario, manipula el Modelo y causa la actualización de la Vista.

El *Framework* de Interfaz de Usuario de Android está organizado alrededor del patrón MVC. Este *framework* provee la estructura y herramientas para construir un Controlador que trate las entradas del usuario (ej. toques de pantalla) y una Vista que muestre de forma gráfica información en la pantalla (Gargenta, 2014).

ViewHolder: Éste patrón es usado en un caso muy particular en el desarrollo para dispositivos Android: los *ListView*. El patrón *ViewHolder* tiene un único propósito en su implementación: mejorar el performance. Y es que cuando hablamos de aplicaciones móviles, el rendimiento es un tema de gran importancia (Calvo, 2015).

Lo que hace el *ViewHolder* es mantener una referencia a los elementos del *ListView* mientras el usuario realiza *scrolling* en la aplicación. Así que cada vez que se obtiene la vista de un *item* evitamos las frecuentes llamadas a *findViewById*, la cual se realizaría únicamente la primera vez y el resto llamaría a la referencia en el *ViewHolder*, ahorrando procesamiento.

Observador (Observer): Es un patrón de diseño de comportamiento que define una dependencia de uno-a-muchos o de uno-a-uno entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él (Gamma, 2003). Dentro del sistema es utilizado para que determinadas entidades sean notificadas cuando el usuario realiza determinadas acciones sobre algunos de los componentes visuales que integran las interfaces visuales del sistema. Permite establecer la lógica programable de las interfaces visuales y de los componentes que la integran.

Adaptador (*Adapter*): Es un patrón de diseño estructural que se encarga de adaptar una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla. Este patrón es conocido como Envoltorio (*Wrapper*) (Gamma, 2003) (Ruiz, 2015).En el sistema determinadas entidades implementa este patrón para que entidades del modelo del negocio puedan ser visualizadas en componentes gráficos como *ListView*, *GridView*, ect.

Fachada (*Facade*): Este es un patrón de diseño estructural que provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema (Gamma, 2003).Este patrón es utilizado en el sistema para proveer un único canal de intercambio de datos entre los dos subsistemas o capas que componen el sistema.

Resultados y discusión

Aunque la metodología XP no comprende el trabajo con diagrama de paquetes, se realizó el diagrama de paquetes de la solución para detallar su funcionamiento.

Este diagrama es el encargado de representar las dependencias entre los paquetes que componen un modelo. Es decir, muestra cómo un sistema está dividido en agrupaciones lógicas y las dependencias entre ellas (Jiménez, 2016).

La estructura del código fuente del sistema está organizado dentro la carpeta *src* tal y como muestra la siguiente figura.

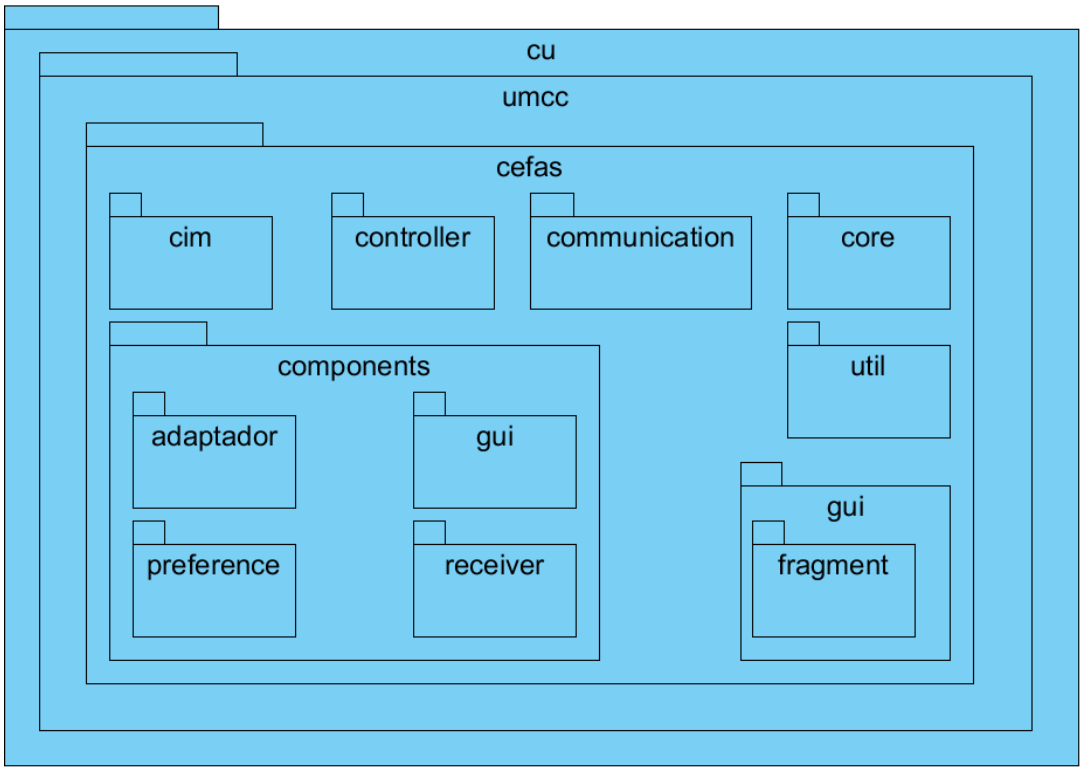

Figura 4. Diagrama de paquetes del sistema.

La misma se organiza de la siguiente manera:

ISBN: 978-959-16-4235-6

- Paquetes *cu*, *umcc* y *cefas*, encierran toda la solución. La primera hace referencia al código del país, mientras la segunda y la tercera hacen referencia a la entidad y dentro de esta el grupo que lleva a cabo el desarrollo del sistema.
- Paquete *cim*: Contiene la entidades que representan el modelo común de información que van compartir el sistema con el servidor.
- Paquete *controller*: Contiene las entidades encargadas de controlar la lógica del negocio del sistema.
- Paquete *communication*: Agrupa las entidades encargadas de establecer comunicación con el servidor, así como de elaborar las peticiones y procesar las respuestas.
- Paquete *core*: Contiene aquellas entidades que sirven de interfaces, los enumerativos que se utilizarán en el sistema.
- Paquete *util*: En este paquete están aquellas entidades que no persiguen o tienen un objetivo específico sino con sus funcionalidades ayudan a otras a cumplir sus propósitos u objetivos.
- Paquete *gui*: Compuesta por entidades que representa las pantallas o interfaces gráficas del sistema y el subpaquete *fragments*.
    - Paquete fragments: Entidades que heredan de la entidad *Fragment* propia de Android representan fragmentos de las interfaces visuales del sistema.
- Paquete *components*: Integrada por cuatro subpaquetes que albergan aquellas entidades que heredan y redefinen comportamientos de entidades propias de Android.
    - Paquete *gui*: Conformado por las entidades que heredan y redefinen el comportamiento de determinados componentes visuales propios de Android.
    - Paquete *receiver*: Almacena las entidades que heredan y redefinen el comportamiento de la clase *BroadcastReceiver* propia de Android.
    - Paquete *preference*: Constituida por las entidades que heredan y redefinen el comportamiento de las clases destinadas a la confección de las preferencias de las aplicaciones desarrolladas con Android.
    - Paquete *adaptador*: Alberga las entidades que heredan y redefinen el comportamiento de las clases *ArrayAdapter* y *BaseExpandableListAdapter* propias de Android.

En las siguientes figuras se muestran las principales interfaces gráficas con que cuenta el sistema:

ISBN: 978-959-16-4235-6

Figura 5. Ventana principal

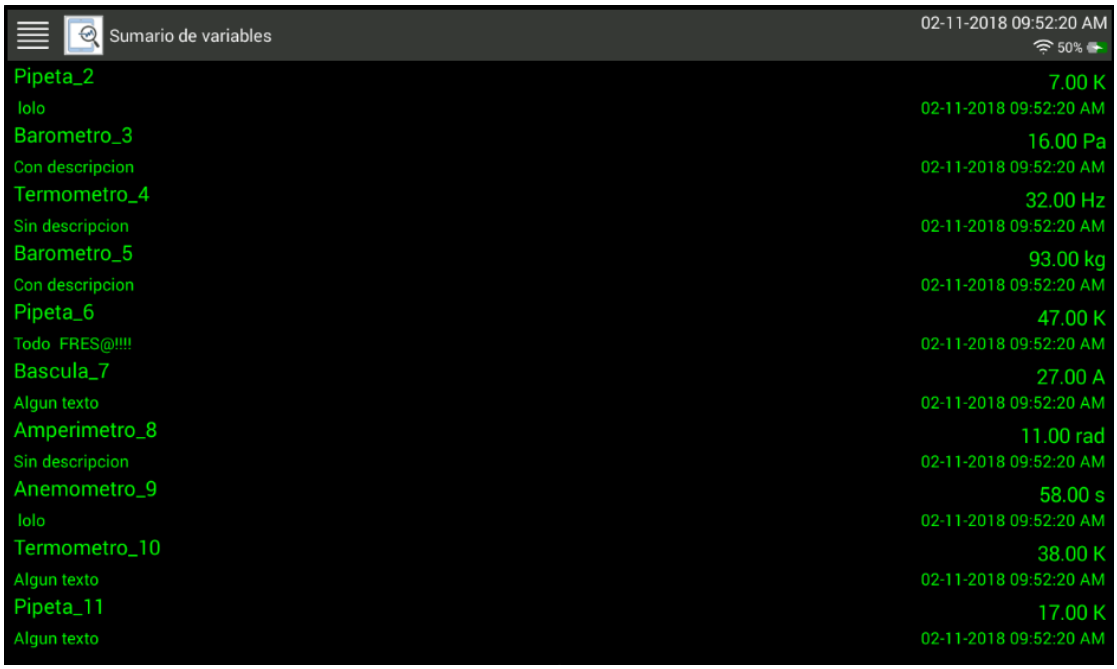

Figura 6. Sumario de variables

ISBN: 978-959-16-4235-6

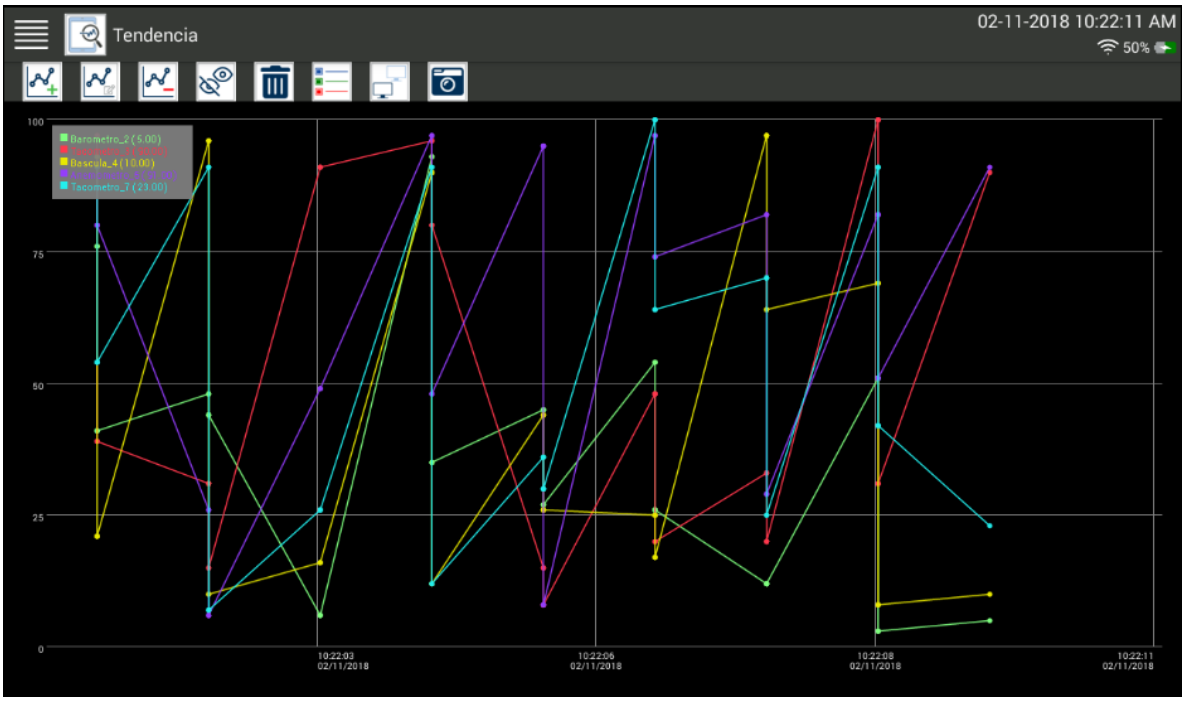

Figura 7. Tendencia.

El sistema fue sometido a diversas pruebas con el fin de corregir las posibles fallas presentes y con esto disminuir el margen de error de este. Una vez analizado los resultados arrojados por las diferentes pruebas realizadas y solucionado las fallas detectadas se puede concluir que el sistema Visualizador está listo para ser desplegado y ser utilizado como la interfaz Hombre-Máquina del ambiente de ejecución o visualización del sistema ROFLEXIN/LC desde dispositivos móviles y *tablets*.

Conclusiones

El desarrollo de una aplicación móvil que permita la disponibilidad del visualizador del HMI del sistema de monitoreo ROFELXIN/LC es un gran reto y una oportunidad que sin dudas posee un alto impacto social, ya que no existe otra solución similar en el CEFAS. El hecho además de que el sistema esté desarrollado para una arquitectura móvil, posibilita explotar los beneficios de la red, minimizando los costos de hardware en las organizaciones. Con la realización de la presente investigación se logró:

1. Establecer de los fundamentos teórico-metodológicos para el desarrollo de sistemas informáticos de monitoreo en el entorno móvil.

2. Analizar del estado del arte de las principales tecnologías, metodologías y herramientas para el desarrollo de aplicaciones móviles en tiempo real.

3. Analizar y diseñar una solución informática.

ISBN: 978-959-16-4235-6

4. Implementar una solución informática.

5. Validar de los resultados obtenidos mediante la realización de pruebas.

6. Valorar cualitativamente de los resultados obtenidos en la validación de la solución propuesta.

Destacar que todo el desarrollo del trabajo se utilizó herramientas y tecnologías libres.

Bibliografía

Academy, ICode. 2017. *Json for Beginners: Your Guide to Easily Learn Json in 7 Days.* 2017.
Beck, K. and Molina, J.G. 2002. *Una explicación de la programación extrema: aceptar el cambio.* 2002.
Burd, B.A. 2004. *Eclipse For Dummies.* 2004.
Calvo, A. 2015. *Beginning Android Wearables: With Android Wear and Google Glass SDKs.* 2015.
Fowler, M. 2012. *Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch.* 2012.
Gallardo, D. and Burnette, E. and McGovern, R. 2003. *Eclipse in Action: A Guide for Java Developers.* 2003.
Gamma, E. and Helm, R. and Johnson, R. and Vlissides, J. 2003. *Design Patterns: Elements of Reusable Object-Oriented Software with Applying Uml and Patterns:An Introduction to Object-Oriented Analysis and Design and the Unified Process.* 2003.
Gargenta, M. and Nakamura, M. 2014. *Learning Android: Develop Mobile Apps Using Java and Eclipse.* 2014.
Goytia, L.L. and González, A.G. 2014. *Programación Orientada a Objetos C++ y Java.* 2014.
Jiménez, J.L.A. 2016. *UF2406 - El cliclo de vida del desarrollo de aplicaciones.* 2016.
Keller, S. 2016. *Json Book: Easy Learning of Javascript Standard Object Notation.* 2016.
2011. Lineamientos de la política económica y social del partido y la revolución. 2011.
Ruiz, A.P. 2015. *Mastering Android Application Development.* 2015.
Smyth, N. 2017. *Kotlin / Android Studio 3.0 Development Essentials - Android 8 Edition.* 2017.
Sommerville, I. 2015. *Software Engineering.* 2015.

ISBN: 978-959-16-4235-6