Universidad Central "Marta Abreu" de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

"Interfaz Genérica para Sistemas de Adquisición basada en Software Libre"

Autor: Erick López González

Tutor: Msc. Fidel González Vázquez

Santa Clara

2009

"Año del 50 aniversario del triunfo de la Revolución"

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central “Marta Abreu” de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Automática, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

Firma del Autor

Firma del Jefe de Departamento donde se defiende el trabajo

Firma del Responsable de Información Científico-Técnica

PENSAMIENTO

Los grandes espíritus siempre han encontrado una violenta oposición de parte de mentes mediocres.

Albert Einstein

DEDICATORIA

A mi madre por todo.

A mi abuelo Muñí, ojalá estuviera todavía.

AGRADECIMIENTOS

Resulta muy difícil mencionar en pocas líneas a tantas personas que, de un modo u otro, han influenciado positivamente en mi trabajo, apoyándome y alentándome hasta el momento de su culminación. Pero, teniendo en cuenta esta dificultad, me gustaría citar a continuación a una serie de personas a las que agradezco mucho en estos años, pidiendo al mismo tiempo perdón a aquellos a los que pueda no haber mencionado y que igualmente merecen formar parte de estas líneas.

- ✓ *Le agradezco enormemente a mis padres por hacerme quien soy, además de sus genes me han dado mucho.*
- ✓ *A mi familia toda que siempre han estado ahí cuando lo he necesitado para cualquier cosa.*
- ✓ *A mi abuela Edelma y mis tíos Toto, Neme, el Gallo y Baby que son como mis padres también.*
- ✓ *A la persona que más cerca ha estado de mi durante estos 5 años y siempre me ha apoyado, a Midi "Arigatô gozaimasu". Te adoro.*
- ✓ *A todos mis profesores, que de una forma u otra han influido en mi formación como persona y como profesional, en especial a Boris Luis, Fidel y José Omar, que además de profesores han sido amigos.*
- ✓ *Mis compañeros del grupo gracias, juntos hemos podido con todo, luchando como equipo.*

TAREA TÉCNICA

1. Estudio teórico sobre especificación e implementación de drivers.
2. Estudio sobre lenguajes de programación para software industrial.
3. Diseño de la Interfaz Genérica.
4. Implementación de los Drivers.
5. Validación de los Resultados.
6. Confección del informe final.

Firma del Autor	Firma del Tutor

RESUMEN

El presente Trabajo de Diploma consiste en el diseño e implementación de una Interfaz Genérica que permita la comunicación a otras aplicaciones con dispositivos de campo a través de los controladores de los protocolos usados por éstos.

Se realizó un estudio de las distintas Interfaces en base a métodos y herramientas para la realización de éstas y se diseñó la aplicación. El diseño de la aplicación se realizó basado en una arquitectura multicapa que ofrece ventajas como por ejemplo la posibilidad de reutilización y adaptabilidad del código y de cambiar una capa sin la necesidad de hacerlo a las otras y en particular disponer de las bibliotecas de protocolo permite modificar de manera más simple la interfaz genérica en caso necesario, además cada capa tiene una funcionalidad lógica propia.

Se implementó la Interfaz Genérica y los controladores de Modbus[1] y PPI[2] en objetos compartidos de modo que pueden ser usados por cualquier aplicación que necesite comunicarse con el campo. La Interfaz está implementada de modo que se le pueden añadir controladores de forma dinámica, es decir sin necesidad de parar su funcionamiento.

---

[1] Protocolo abierto usado para la comunicación entre dispositivos en ambientes industriales.

[2] Ídem a 1 pero es creado por la compañía Siemens y no dan sus especificaciones.

TABLA DE CONTENIDOS

INTRODUCCIÓN

La automatización de la industria en Cuba (Herrera, 2008, González, 2008) es un concepto que se ha manejado con perspectiva mucho antes, incluso del siglo XXI, con ideas y hechos puestos en práctica por el Che con su paradigma de la gran industria socialista desarrollada. Pasos en aquella dirección fueron la creación de una red industrial en todo el país, ejemplos tangibles como Planta Mecánica y la INPUD en la ciudad de Santa Clara. En aquel periodo y hasta los años ochenta, como representantes de tan novedosas ideas quedaron varias fábricas y servicios, donde el significado de automatización se materializaba en la instrumentación de medición y algunos controles y servomecanismos. Poco después ocurrió la caída del campo socialista de Europa del Este y con él, la paralización casi por completo de nuestra producción industrial.

La recuperación económica que ha vivido nuestro país después del periodo especial en los años noventa, abrió las puertas a la diversificación económica tanto de exportación como de importación y por ende generó una apertura tecnológica sin precedentes. La necesidad de poner en práctica conceptos novedosos y retomar los olvidados, en función de la eficiencia industrial y el desarrollo del país ha contribuido a fomentar el desarrollo de la automatización de la industria cubana, así como de la investigación en ese campo. Numerosos centros de desarrollo, con las tres principales universidades cubanas a la vanguardia, se han dado a la tarea de lograr aportes significativos en pos de

mejorar la eficiencia de la producción y los servicios, con aportes reconocidos internacionalmente.

Una de las temáticas relacionadas con la automatización, que más aplicación práctica tiene internacional y nacionalmente es la supervisión de procesos tecnológicos, y hacia ella se han volcado investigadores y empresas, logrando un desarrollo vertiginoso que no se agota; ejemplo de ello es la gran demanda de sistemas de supervisión y control o SCADAs[3] (Bailey and Wright, 2003, Boyer, 1999) por sus siglas en ingles, los cuales son ampliamente usados también en Cuba. Como todo elemento nacido y sujeto al calor del mercado capitalista y dada su demanda y complejidad, los sistemas SCADAs son softwares cotizados a altos precios y en casi la generalidad de los casos, son construidos en la plataforma Windows, lo cual eleva aun más su valor comercial, pues es necesario comprar el sistema operativo y después el sistema SCADA con toda su base física de control y campo.

Debido al monopolio del software técnico que poseen algunas empresas y la dependencia que esto genera de las mismas, en el mundo de la informática industrial se viene escuchando cada vez más la palabra: software libre, lo cual realmente constituye una alternativa al acaparamiento de los sistemas informáticos para la industria. Lo anterior, unido a los consabidos problemas de seguridad del sistema operativo Windows (Siechert and Bott, 2005), la proliferación cada vez mayor de virus informáticos, exploits[4], troyanos, y gusanos(Walker, 2006), ha motivado a los tecnólogos a migrar sus aplicaciones informáticas industriales a sistemas más robustos como las distribuciones del sistema GNU/Linux (Krill, 2009, Microsoft, 2008). De esta necesidad tangible a nacido la idea de crear sistemas de supervisión y control sobre plataformas de software libre.

[3] Supervisory Control And Data Acquisition por sus siglas en ingles, son sistemas de adquisición de datos y control supervisor.

[4] Tipo de virus informático.

La ausencia de un sistema SCADA de gama media y buenas prestaciones generales en plataforma libre, es algo que afecta la alternativa de empleo de este tipo de software para la automatización moderna y por lo tanto es un tema que cobra especial relevancia en centros de investigación y desarrollo. En la actualidad existen muy pocas opciones al respecto, ya que las empresas que históricamente han desarrollado estos productos tienen fuertes lazos con Microsoft Corp. la compañía multimillonaria que desarrolla el sistema Windows, por ello el desarrollo de sistemas de supervisión en plataformas libre es muy escaso.

El presente trabajo pretende hacer un aporte en el desarrollo y divulgación de los sistemas SCADA en software libre, al crear una interfaz genérica para la comunicación con los PLC desde los niveles de procesamiento, gestión y visualización de la información de un sistema SCADA, que funciona en el sistema operativo GNU/Linux. Específicamente la interfaz genérica es la capa de software, que permite a las aplicaciones superiores conectarse a través de los protocolos de los PLC y otros dispositivos de campo “inteligentes”, ejecutando simples métodos de lectura y escritura; puede verse, si se quiere, como un intérprete de lenguajes, haciendo una analogía con el ser humano, hablando en varios idiomas con los “extranjeros” (los protocolos) e interactuando con nosotros (capas superiores) en un lenguaje común. Los protocolos (lenguajes de comunicación para hardware) soportados en este trabajo son el PPI para la interacción con los autómatas de la compañías SIEMENS y Modbus, estándar abierto creado por la compañía Modicon y muy usado por los dispositivos de campo para la instrumentación y el control, en versiones posteriores, se pretenden implementar otros protocolos que ampliarán las capacidades y versatilidad de este producto de software.

En el presente trabajo se define como objetivo contribuir a la comunicación entre un Sistema de Adquisición de Datos y los dispositivos de campo, diseñando e implementando la Interfaz Genérica y los controladores de dispositivos Modbus y PPI. Para lograr dicho objetivo se establecen objetivos específicos, los cuales se enumeran a continuación:

1. Hacer un estudio de las interfaces existentes en el mundo y en Cuba.

2. Seleccionar herramientas de Software Libre mas adecuadas para la realización del proyecto.
3. Hacer el análisis, diseño e implementación de la aplicación para que esta sea flexible, escalable y multiplataforma.
4. Implementar una aplicación de prueba.
5. Analizar los resultados obtenidos.

Para el desarrollo de la investigación se utilizarán diferentes métodos y técnicas que en unidad y diferencias particulares nos permitirán el abordaje del problema. Estos métodos y técnicas favorecerán el cumplimiento de las siguientes tareas:

- ✓ Revisión de la bibliográfica técnico-especializada para la construcción del marco teórico de referencia general de la pesquisa y la delimitación y caracterización de las interfaces ya existentes y las tecnologías usadas para su implementación.
- ✓ Selección las herramientas de software a usar como plataforma para el diseño e implementación efectivas del software, así como del lenguaje de programación a usar.
- ✓ Diseño e implementación de la Interfaz.
- ✓ Realización de las pruebas y comparaciones con las interfaces ya existentes.
- ✓ Confección de la documentación sobre el proyecto.

Organización del informe

El informe de la investigación se estructurará en introducción, capitulario, conclusiones, referencias bibliográficas y/o bibliografía y anexos.

En la introducción se dejará definida la importancia, actualidad y necesidad del tema que se aborda y se dejarán explícitos los elementos del diseño teórico.

Capitulario

CAPÍTULO I: Se dedicará a la caracterización de las tecnologías existentes y al estudio comparativo de las interfaces que se usan en la actualidad (estado del arte).

CAPÍTULO II: Se utilizará para expresar el diseño e implementación de la Interfaz Genérica.

CAPÍTULO III: Se dedicará al Software diseñado e implementado y a expresar los resultados de la validación mediante la comparación con otros actualmente en el mercado.

Conclusiones

Recomendaciones

Referencias bibliográficas y/o Bibliografía

ESTUDIO DE LAS INTERFACES

Desde el inicio de la informática ésta se ha desarrollado mucho, pero las metas globales de los programadores continúan siendo las mismas: “hacer que la tarea de realizar programas para ordenadores sea cada vez lo más simple, flexible y portable posible” (Francisco, 2000). Con el objetivo de mejorar cada vez más en ese sentido el desarrollo de software avanza y cada día son mayores los retos de los programadores.

En este capítulo se describe el estudio hecho sobre el tema de los distintos tipos de Interfaces Genéricas existentes, así como su desarrollo. Todo ello para cumplir con el objetivo de darle al lector una base teórica sobre el tema a tratar en el informe, informarlo sobre lo que existe en el mundo en cuanto a este tipo de software y darle a conocer el porqué de la realización de éste.

1.1 Interfaz Genérica, su definición

1.1.1 Interfaz

Según la definición de la palabra buscada en diccionarios en software, parte de un programa que permite el flujo de información entre un usuario y la aplicación, o entre la aplicación y otros programas o periféricos. Esa parte de un programa está constituida por un conjunto de comandos y métodos que permiten estas intercomunicaciones. Interfaz también hace referencia al conjunto de métodos para lograr interactividad entre un usuario y una computadora. Una interfaz puede

ser del tipo GUI[5], o línea de comandos. También puede ser a partir de un hardware, por ejemplo, el monitor, el teclado y el mouse, son interfaces entre el usuario y el ordenador.

En electrónica, un interfaz es el puerto por el cual se envían o reciben señales desde un sistema hacia otros. Por ejemplo, el interfaz USB[6], interfaz SCSI[7], interfaz IDE[8], interfaz puerto paralelo o serial, etc.

Pero en general el concepto más abarcador es: “Punto de interconexión entre dos entidades, sistemas, equipos, conceptos, etc (Babylon, 2009, Wikitionary, 2008).

1.1.2 Genérico, ca

Que es común o se refiere a un conjunto de elementos del mismo género(RAE, 2009).

1.1.3 Interfaz Genérica

Cuando en materia de software hablamos de una interfaz que permita la comunicación con otros medios independientemente de cuáles sean estos, es decir sea capaz de encapsular varios protocolos o cualquier grupo de objetos, realizando la tarea de seleccionar el adecuado en cada caso según se le demande y dando un conjunto de funciones que permitan una interacción con ellos, estamos hablando de una Interfaz Genérica.

1.2. Interfaces Genéricas, ejemplos

Con el objetivo de una mejor comprensión de lo que es una Interfaz Genérica y de conocer sus aplicaciones y el estado de estas en el mundo de hoy se listan a continuación algunos ejemplos de implementaciones y proyectos a realizarse.

[5] Interfaz Gráfica de Usuario.

[6] Puerto Serie Universal

[7] Protocolo de conexión utilizado por discos duros de altas prestaciones y muy caro.

[8] Otro protocol usado para la conexión de medios de almacenamiento al ordenador.

1.2.1 MOCA

MOCA (Ramírez, 2004) es una estructura de servicios adaptables para dispositivos computacionales móviles con memoria limitada, como celulares y PDAs. Para asegurar portabilidad a través de las plataformas heterogéneas de hardware, fue desarrollada en Java[9]. La estructura de MOCA es abierta y es escalable. Esta estructura permite algunas ventajas:

Los dispositivos se adaptan a su ambiente: mediante descubrimientos y descargas dinámicas de los servicios de dispositivos cercanos. El software en los dispositivos es extensible: las aplicaciones pueden residir localmente en el dispositivo o ser descargadas de la red inalámbrica de manera transparente. Las aplicaciones comparten los servicios: MOCA administra eficientemente los recursos de los dispositivos conteniendo diversas aplicaciones en una misma JVM[10]. MOCA tiene como objetivos específicos de aplicación dispositivos computacionales móviles que permitan adaptabilidad dinámica, control de rastreo de memoria, seguridad y portabilidad. El tamaño de los componentes básicos de MOCA es de 50K.

La arquitectura de MOCA está basada en la noción de dos componentes:

- *Servicio:* es un componente de software independiente manejado por MOCA que puede ser
- activo (con hilos de ejecución) o pasivo. Encapsula una función y proporciona una interfaz
- específica a la función que realiza. Existen dos tipos de servicios:
- *_local*: por ejemplo, servicio de cargado de un archivo.
- *_remoto*: por ejemplo, servicio de impresión y servicio de almacenamiento.

[9] Lenguaje de programación que necesita de un programa llamado máquina virtual para poder ejecutar sus binarios, pero esta característica hace que sus programas sean fácilmente multiplataforma.

[10] Máquina Virtual de Java

Como un servicio puede ser accedido por las aplicaciones o por otros servicios que se ejecutan en un dispositivo, todos los servicios residen en un mismo espacio compartido. Es un programa de Java estándar que declara un método principal y se entrega como uno o más archivos de clases. Las aplicaciones pueden solicitar y usar servicios invocando métodos que proporcionan las interfaces de servicios. MOCA permite que varias aplicaciones se ejecuten en una misma Máquina Virtual de Java (JVM, *Java Virtual Machine*).

Los componentes principales de la arquitectura MOCA son:

Registro de servicios. Es el elemento central de la arquitectura MOCA. Realiza dos roles:

- Actúa como un repositorio central de servicios y mantiene el mapeo de la descripción de servicios a las implementaciones correspondientes.
- Encapsula la administración del ciclo de vida de los servicios. En otras palabras, registra, actualiza, y borra el registro de los servicios. Un servicio puede registrarse.

1.2.2 Controlador personal universal (*Personal Universal Controller*, PUC)

Los aparatos de oficina y domésticos son cada vez más complejos debido a que ya casi todos ellos contienen algún tipo de microprocesador. A medida que la complejidad de los aparatos aumenta, las interfaces de los aparatos son más difíciles de usar. Muchas de estas interfaces no presentan opciones para que personas discapacitadas puedan usar los equipos. Para resolver este problema varios grupos de investigación, industriales y académicos, están trabajando para simplificar el control de aparatos y servicios creando un control remoto universal. A diferencia de los controles remotos preprogramables disponibles actualmente, estos nuevos controles descargan una especificación del aparato o servicio y la usan para generar automáticamente una interfaz para el control remoto. Esto promete ser un enfoque útil porque la especificación puede ser lo suficientemente

detallada para generar tanto interfaces de voz como gráficas. Sin embargo, la generación de interfaces puede ser difícil.

En algunos artículos sugieren que separar la interfaz del aparato podría solucionar el problema. Proponen un controlador personal universal (*Personal Universal Controller*, PUC) (Ramírez, 2004), un dispositivo que permite al usuario interactuar con todos los aparatos y servicios de su ambiente.

Un PUC podría tomar varias formas:

- una computadora de mano con interfaz gráfica para una persona discapacitada
- una superficie interactiva de Braille para un invidente
- una diadema que permita síntesis y reconocimiento de voz

Cuando el usuario desee controlar un aparato, el PUC se comunicaría con él, descargaría la especificación con sus funciones y generaría automáticamente una interfaz de control remoto adecuada para el PUC y el usuario. El PUC y el aparato continuarían intercambiando mensajes conforme el usuario manipula la interfaz.

1.2.3 Interfaz Genérica para la estimación de autómatas de estados finitos ponderados

Este es un trabajo donde se define e implementa una metodología novedosa para el entrenamiento de un AEFP[11] (Penagarikano, 2006). Todo modelo que implemente una sencilla interfaz puede ser entrenado independientemente de su estructura y sin conocimiento alguno de su representación interior. Para el caso del entrenamiento de MOM por MV, la reestimación es exactamente la misma que se obtendría mediante el algoritmo de Baum-Welch.

[11] Autómatas de Estados Finitos Pnderados

Por otra parte, los MMC constituyen un formalismo extremadamente útil tanto para la decodificación como para el entrenamiento, ya que permiten la reestimación simultánea de un conjunto de AEFP. Partiendo de esta metodología, se ha desarrollado un módulo de entrenamiento para el sistema *Sautrela*, que simplifica considerablemente el esfuerzo necesario para incorporar nuevos formalismos de modelado a un sistema de procesamiento del habla basado en AEFP.

1.2.4 Interfaz USB genérica para comunicación con dispositivos electrónicos

Este proyecto es una respuesta a la necesidad de comunicar de forma sencilla y genérica dispositivos electrónicos no necesariamente pensados para interactuar con un PC (Tejera, 2006).

La solución se basa en tres puntos:

- Un componente de hardware.
- Un medio de comunicación (USB).
- Una arquitectura (software y firmware).

Con él se utiliza una PC para comunicarse con dispositivos electrónicos, logrando:

- Aumentar la potencialidad de los dispositivos.
- Aprovechar las capacidades de procesamiento, y almacenamiento del PC.
- Aumentar la Interacción con el mundo físico.
- Simplificar el manejo de los dispositivos.
- Uso de microcontroladores como parte de la solución.

Objetivos del proyecto:

- Construcción de hardware y software necesarios para facilitar la comunicación con dispositivos electrónicos por medio del USB.
- Ocultar la complejidad de la tecnología USB.

- Arquitectura modularizada y extensible.
- Firmware[12], API, protocolo de comunicación, Drivers[13].
- Bibliotecas de alto nivel para distintos dispositivos
- Soporte para Linux y Windows.

¿Cómo se comunican los dispositivos?

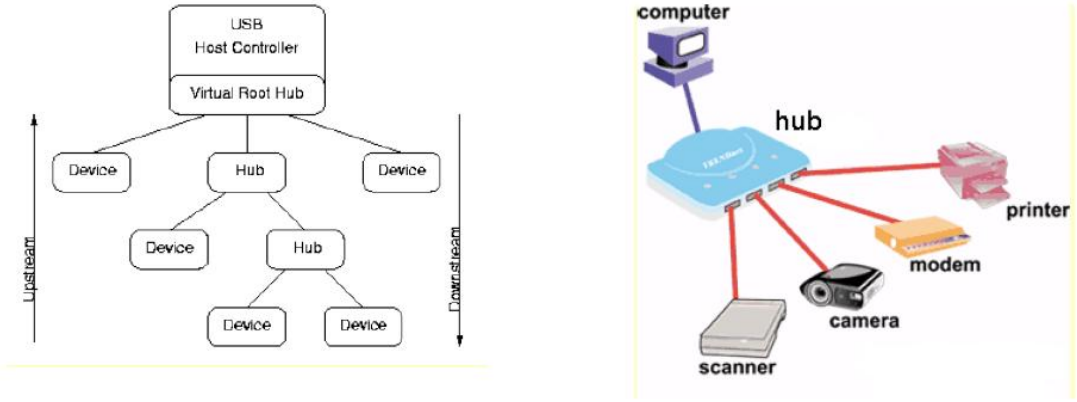

Figura 1.1 Los dispositivos se comunican por marcos con Bus centrado en el host, El maestro (host) inicia todas las transferencias y los esclavos (dispositivos) responden a los pedidos.

1.2.5 Plataforma genérica para desarrollos basados en agentes móviles

En este trabajo se presenta una plataforma de desarrollo genérica (Rivera, 2007), fácilmente modificable, ampliable y abierta, para la prueba y evaluación de sistemas basados en agentes móviles. Los agentes se pueden interconectar mediante gran cantidad de medios físicos como GSM, Wireless, sistemas de radio a 433 o 868 MHz, BlueTooth, etc, y pueden ser controlados y programados desde cualquier sistema operativo o lenguaje de programación. Su control se puede

[12] Sistema Operativo de dispositivos empotrados, como celulares, Mp4, PDAs, etc.

[13] Manejadores o controladores, son programas informáticos que permiten interactuar con dispositivos físicos.

realizar desde la web. Los agentes móviles se pueden adaptar a cualquier entorno, añadiéndose de forma sencilla los nuevos periféricos requeridos.

La Inteligencia Ambiental se caracteriza por ser un área de trabajo especialmente interdisciplinario, en la que se pueden distinguir dos grandes campos de trabajo. El primero de ellos estaría involucrado en la creación de los elementos hardware necesarios para poder estar frente a un "entorno inteligente", tales como sistemas de comunicación, sistemas biométricos, sistemas de localización. El segundo de ellos se encargaría del desarrollo de los algoritmos y aplicaciones software necesarios para dar funcionalidad al hardware anteriormente mencionado.

Esta amplia interdisciplinaridad necesaria para crear sistemas que ayuden a realizar las tareas diarias a los usuarios, acarrea graves problemas a la hora de desarrollarlos. Por un lado los equipos de investigación no suelen estar especializados en todas las áreas de conocimiento necesarias, lo que hace que las investigaciones acaben en meros algoritmos funcionando sobre un simulador, en hardware inservible por si mismo o, en el peor de los casos, en meras especulaciones. Por otra parte, si el equipo se decide por realizar el trabajo completo para el desarrollo del sistema, se enfrenta a muchos problemas en múltiples campos, lo que supone una gran infraestructura y tiempo de desarrollo.

El presente trabajo describe una plataforma funcional sobre la que poder trabajar sobre entornos inteligentes centrándose en problemas concretos de software, hardware o comunicaciones y olvidándose de los demás, pues ya están implementados y funcionando en la plataforma.

Entre las funcionalidades que ofrece esta plataforma están:

- Acceso al Hardware[14] mediante simples rutinas
- Agentes Móviles (Robots) baratos, extensibles y abiertos
- Adición de nuevo Hardware mediante una sencilla interfaz (Servidor Sensorial)
- Intercomunicación de los agentes desde prácticamente cualquier medio físico

[14] Parte dura de los sistemas informáticos, es decir lo físico lo que se puede tocar.

- Acceso al sistema desde cualquier sistema operativo/ lenguaje de programación
- Fácil integración con la web

Desde un punto de vista más amplio podemos ver el sistema como un punto de encuentro entre los distintos equipos de investigación, pues permite compartir desarrollos de una forma sencilla, para así crecer entre todos. A todo esto se suma que la plataforma en su totalidad está basada en estándares y herramientas libres. De esta manera todos los desarrollos realizados para esta plataforma se podrían portar a futuras plataformas abiertas de forma sencilla.

1.2 Importancia de la definición de la Interfaz Genérica

Como ya hemos visto las interfaces genéricas son ampliamente utilizadas en diferentes aplicaciones y existen proyectos destinados a continuar el desarrollo de estas herramientas destinadas a mejorar tanto la experiencia de los usuarios finales como de los programadores.

Pongámonos a pensar en que sería de algunas aplicaciones conocidas por nosotros, que hacen uso de interfaces genéricas, sin estas. Por ejemplo un reproductor de audio como el conocido Winamp[15], la interfaz que posee permite reproducir varios formatos distintos de archivos de audio, incluso en sus versiones más recientes también de video, imagínense ahora, si el Winamp no tuviera esta interfaz genérica que detecta el tipo de archivo que se quiere reproducir y carga el decodificador para dicho archivo, tendríamos nosotros que hacerlo, se imaginan viendo el formato de cada archivo y con un programa aparte para cada uno de ellos, sería un desastre si quisiéramos reproducir nuestra música y la tuviéramos por ejemplo algunos en mp3, otras en wma y otras en ogg(Martín, 2005).

Pues esto se cumple exactamente igual para casi todos los lugares donde sea usada una interfaz, en un Sistema SCADA la definición de la Interfaz Genérica,

[15] Programa informático para reproducir música y las últimas versiones también video que corre sobre el Sistema Operativo Windows y de mucha aceptación y uso.

además de aportar gran flexibilidad y escalabilidad, permite a los desarrolladores de todos los módulos que necesiten la comunicación con dispositivos de campo hacer su trabajo sin la necesidad de conocer las características específicas de cada protocolo usado para la transacción de datos ya que sólo deben conocer las especificaciones y las posibilidades que brinda la interfaz al resto de la aplicación y con esto son capaces de interactuar con cualquier dispositivo soportado por los manejadores.

1.3 Interfaces Genéricas propietarias y libres

Este tipo de software, donde mas usado es en Sistemas Supervisorios y debido a que la inmensa mayoría de estos son privativos, pues no hay documentación sobre ellos, lo cierto es que los SCADAs más usados como Movicon X, Intouch, Wincc, soportan gran cantidad de protocolos en sus interfaces de drivers, esto también es debido a que pagan licencia y usan controladores propietarios de varias empresas que mantienen sus implementaciones cerradas.

En el caso de los pocos SCADAs libres que existen en la actualidad, debido principalmente al poco tiempo que llevan de desarrollo, poseen una interfaz de comunicación aún sin mucho desarrollo y sobre todo muy poco documentada.

1.4 Necesidad del trabajo

La falta de software industrial que sea libre o de código abierto es enorme, es una de las ramas del software que menos desarrollo posee en la comunidad de software libre, esta está dominada por compañías que generalmente son las que hacen el hardware y los protocolos y mantienen sus programas de código cerrado para tener el control total sobre ellos.

Con el desarrollo paulatino que está teniendo la industria nacional después del impulso económico que se ve en el país debido a varios factores, como los diferentes convenios en el marco del ALBA[16], se ha podido ver que grandes partes de las inversiones se utilizan en la automatización de estas industrias y dentro de

[16] Alternativa Bolivariana para las Américas, proyecto de integración de América Latina.

la automatización, una parte considerable en el software supervisor, que todos los existentes poseen licencias extremadamente caras.

En nuestra región existen varias industrias que están en medio de este proceso de revitalización y modernización y han contratado para ello a la Empresa de Automatización Integral (CEDAI), en su sucursal de Villa Clara.

1.5 Problema a resolver

Teniendo en cuenta la necesidad existente la empresa CEDAI Villa Clara comienza la ejecución de un proyecto para la creación de un SCADA basado en tecnologías de software libre para el Instituto de Bioplantas, teniendo además varias industrias interesadas en un producto igual, por lo que nos damos a la tarea de hacer un sistema genérico con el cual se puedan cubrir las necesidades de la mayoría de estos clientes.

Como parte de este proyecto, se da la tarea de hacer el módulo de comunicación el cual consta de la Interfaz Genérica y los manejadores, en principio de Modbus serie y tcp y PPI de Siemens.

1.6 Conclusiones parciales

Después de realizar búsquedas en distintos tipos de bibliografías, tanto físicas como virtuales, se puede decir que hay poca documentación sobre el tema de las Interfaces Genéricas, especialmente en la rama de las interfaces para comunicación industrial, donde la mayoría son realizadas por compañías privadas que mantienen oculto todo lo que desarrollan, incluso protocolos hechos por ellas no dan las especificaciones como es el caso de Siemens.

Los proyectos que existen hoy en día basados en software libre están todavía en etapa de desarrollo no muy avanzada, es decir no son maduros y una de las peores cosas que poseen es precisamente la documentación.

A pesar de ello con lo estudiado se puede comenzar el diseño de la interfaz para la comunicación con dispositivos de campo y mas tarde la implementación de la misma, proceso que se describe en el siguiente capítulo.

DISEÑO E IMPLEMENTACIÓN DE LA APLICACIÓN

Después del estudio exhaustivo realizado en el capítulo 1 estamos listos para entrar en el desarrollo del proyecto, donde primero haremos la definición de las herramientas y lenguajes de modelado e implementación a usar, plataforma de desarrollo y otras herramientas necesarias, después se describe el proceso de diseño e implementación de la Interfaz y sus controladores para terminar con las conclusiones parciales del capítulo.

2.1 Descripción general

El software puede ser usado para cualquier aplicación que necesite comunicación con dispositivos de campo, ya sean estos PLCs, sensores inteligentes, dispositivos empotrados de adquisición cualquiera que sea capaz de comunicarse con los protocolos implementados en los controladores que posea la interfaz, que en su primera fase contará con soporte para MODBUS en sus variantes de comunicación Serie y TCP/IP[17] y codificaciones ASCII y RTU[18].

El programa brinda una interfaz genérica para cualquier protocolo que se vaya a usar, teniendo varias funciones que permiten la interacción con los controladores y dispositivos.

[17] Formas de transmission de datos, por el Puerto serie y a través de la red, TCP/IP es un protocolo.

[18] Formas de codificacion de los datos.

2.2 Diseño e implementación

Después de tener bien definida la tarea que queremos realizar, seleccionamos las herramientas a usar y con ellas implementamos la aplicación.

2.2.1 Lenguajes y herramientas

Siempre que se va a realizar un proyecto de cualquier tipo o complejidad, antes de comenzar a implementarse, se debe hacer un estudio de las posibles herramientas y materiales que estén a nuestro alcance para la confección del mismo y de acuerdo con nuestras metas se seleccionan las más adecuadas para ello.

2.2.1.1 Lenguajes

En el modelado de la aplicación se utilizó el Lenguaje Unificado de Modelado (Unified Modeling Language, UML) (Jacobson et al., 1999, Neustadt and Arlow, 2006), es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Para la implementación del software se usan los lenguajes C/C++ (Eckel, 2000, Stroustrup, 1999), lenguajes usados debido a sus características, tales como eficiencia con respecto a otros, cantidad de proyectos semejantes existentes en ese lenguaje, experiencias propias en el lenguaje, bibliotecas existentes que utilizan el lenguaje y se reutilizan en el proyecto en aras de ganar en eficiencia y portabilidad, además que como forma parte de un proyecto mas grande para aumentar la compatibilidad con las otras partes del proyecto que se implementarán también en C++.

El programa es implementado usando tecnología Qt (Blanchette and Summerfield, 2006, Wikibooks, 2009) para aprovechar todas las ventajas de este IDE para el desarrollo de aplicaciones multiplataforma.

Algunas de sus características son:

● Compatibilidad multiplataforma con un sólo código fuente

● Soporte para C++

● Disponibilidad del código fuente

● Excelente documentación

● Arquitectura lista para plugins

Pero además tenemos que es usada en varios proyectos reconocidos y por compañías conocidas como: Nokia, Google, Epson, AMD (Nokia, 2009a).

2.2.1.2 Herramientas

Para el modelado se usó la herramienta libre Umbrello (Hensgen, 2003). Umbrello es una herramienta libre para crear y editar diagramas UML, que ayuda en el proceso del desarrollo de software. Fue desarrollada por Paul Hensgen, y está diseñado principalmente para KDE[19], aunque funciona en otros entornos de escritorio.

Umbrello maneja gran parte de los diagramas estándar UML pudiendo crearlos, además de manualmente, importándolos a partir de código en C++, Java, Python, IDL, Pascal/Delphi, Ada, o también Perl (haciendo uso de una aplicación externa). Así mismo, permite crear un diagrama y generar el código automáticamente en los lenguajes antes citados, entre otros. El formato de fichero que utiliza está basado en XMI.

[19] Entorno de escritorio mas usado en los sistemas operativos Linux.

También permite la distribución de los modelos exportándolos en los formatos DocBook[20] y XHTML, lo que facilita los proyectos colaborativos donde los desarrolladores no tienen acceso directo a Umbrello o donde los modelos van a ser publicados vía web.

Para la implementación la plataforma de desarrollo que se usó Eclipse Ganymede (Comunity, 2009), que es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido". Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent Azureus.

Algunas características de Eclipse Ganymede son:

- Editor de texto
- Resaltado de sintaxis
- Compilación en tiempo real
- Pruebas unitarias con JUnit
- Control de versiones con CVS
- Asistentes (wizards): para creación de proyectos, clases, tests, etc.
- Refactorización

Asimismo, a través de "plugins" libremente disponibles es posible añadir varias funcionalidades como el soporte de proyectos de Qt, pudiendo trabajar estos fácilmente con el plugin de Qt el cual permite la creación y el manejo de proyectos Qt, se usan bibliotecas de Qt 4.3.

[20] DocBook es un aplicación del estándar SGML/XML e incluye una DTD propia y que se utiliza de manera más destacada en el área de la documentación técnica, especialmente para documentar todo tipo de material y programas informáticos.

Además también se usó el IDE de desarrollo hecho por la compañía creadora de Qt “Qt Creator” (Nokia, 2009b, Trolltech, 2009) versión 1.0, el cual sobresale por el bajo consumo de recursos de hardware.

2.3 Modelado

Generalmente los protocolos de comunicación con los dispositivos definen la semántica de los mensajes, y su estructura dejando la trasmisión de los mismos a capas inferiores de transporte (por ejemplo TCP/IP para Ethernet o comunicación RS-232 para serie). La interfaz debe ser suficientemente general para que a ella puedan conectarse los diferentes controladores independientemente de su naturaleza y suficientemente eficiente como para que se aprovechen al máximo las posibilidades de cada protocolo y el diseño de los controladores sea simple. Basados en esta concepción general los controladores deben desarrollarse como un sistema multicapa (Figura 2.1):

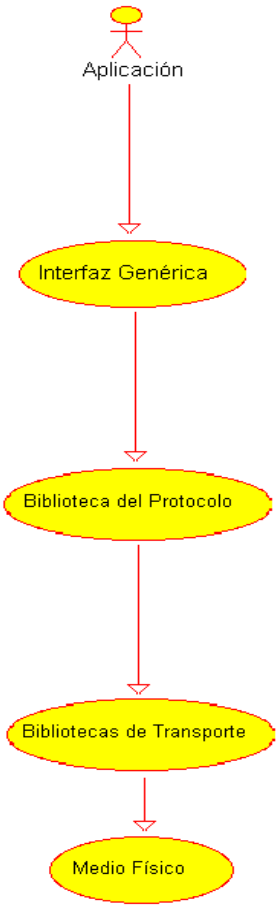

Figura 2.1 Modelo multicapa de la Interfaz Genérica

La biblioteca de transporte es la encargada de manejar la conexión (si existiera) y la trasmisión de un flujo de datos a través del medio físico. La biblioteca del protocolo proporciona una serie de funciones (API) que encapsulan la construcción e interpretación de los mensajes necesarios para la comunicación. La biblioteca del protocolo utiliza la capa de transporte de modo que no debe entrar en las especificidades de cómo se envían o reciben los mensajes. Por último la interfaz genérica debe traducir en términos de llamadas a la biblioteca del protocolo la interfaz genérica.

Ahora hablaremos de las ventajas de esta arquitectura para que se comprenda el porqué de su uso:

Las dos primeras capas de los controladores son reutilizables y tienen valor por si mismos. En particular disponer de las bibliotecas de protocolo permite modificar de manera más simple la interfaz genérica en caso necesario.

Cada capa tiene una funcionalidad lógica propia. La capa de transporte envía y recibe mensajes, la capa del protocolo construye e interpreta los mensajes y la capa de implementación traduce la interfaz genérica en términos del protocolo. A partir de esta separación lógica es posible arrancar en paralelo los trabajos en las tres capas siempre y cuando estén claras las interfaces correspondientes.

2.3.1 Diagramas de caso de uso

La Figura 2.2 presenta el actor principal del sistema y los casos de uso generales de este, así como las relaciones entre ellos. En los próximos epígrafes se abordarán los diagramas y subdiagramas de los casos de uso abstractos.

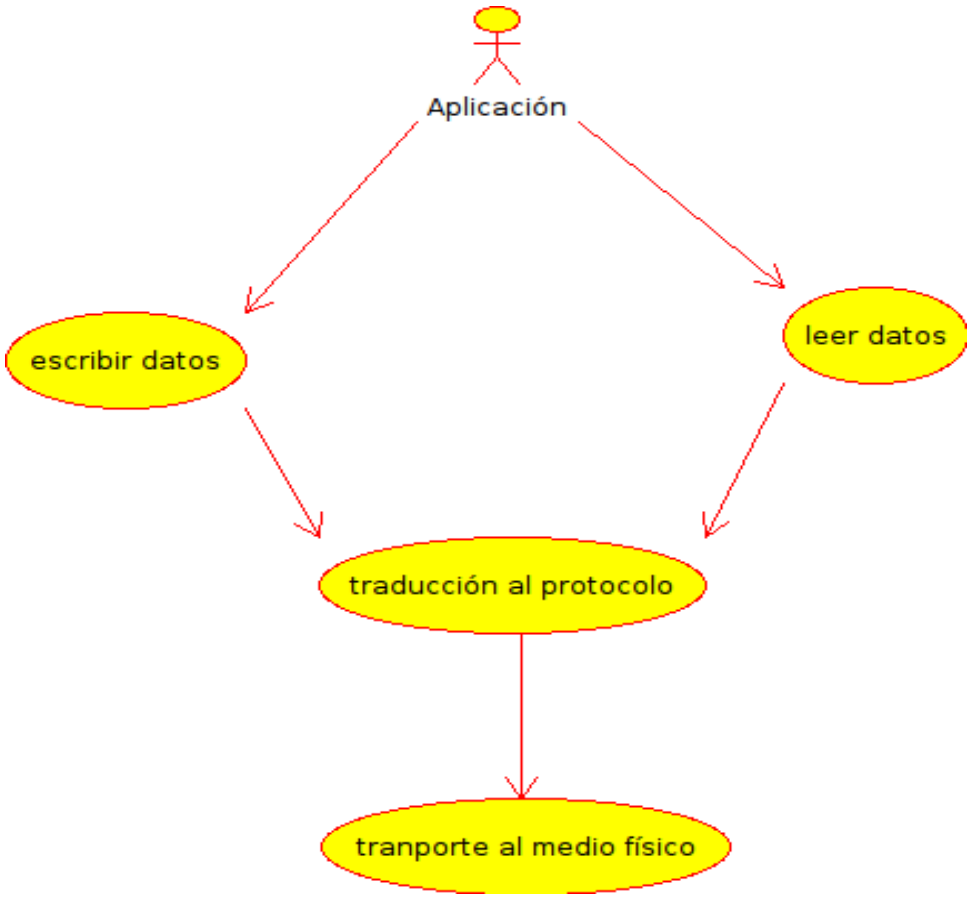

Figura 2.2 Caso de uso general.

Actores:

El sistema cuenta con un actor principal, el cual sería cualquier aplicación que necesite la conexión con dispositivos de campo a través de la Interfaz.

Los principales casos de uso que tiene asociado este actor son los de leer y escribir datos en dichos dispositivos. El actor envía la trama de datos, la función a realizar y el dispositivo, con esto el sistema selecciona el controlador adecuado para el dispositivo en el cual se traducen los datos a su lenguaje específico y se le pasan a la capa de transporte, la cual es la encargada de entregarlo al medio físico.

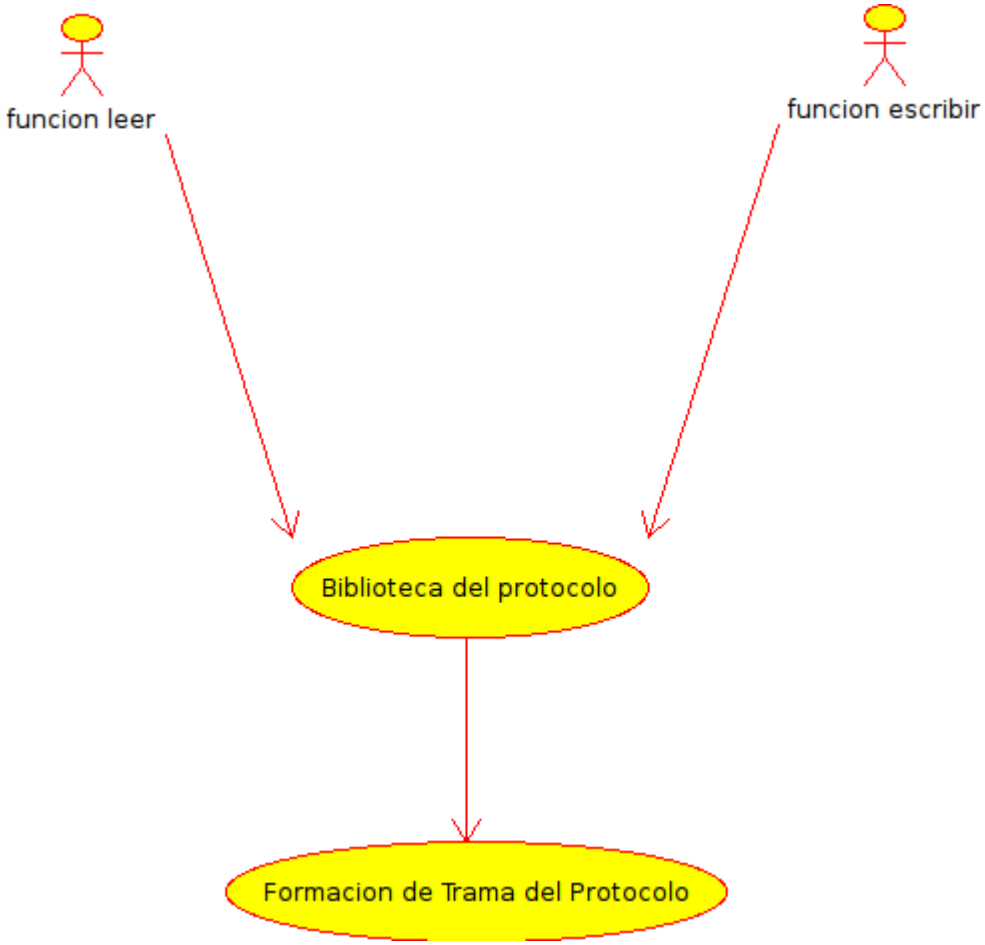

Figura 2.3 Caso de uso Traducción al protocolo

En este ejemplo vemos como los actores serían las funciones de la Interfaz Genérica, las cuales interactúan con las bibliotecas de los protocolos, en dependencia del tipo de dispositivo la interfaz selecciona la biblioteca correspondiente, a las cuales se les envía la trama de datos a ser traducida, esta biblioteca se encarga de traducir estos datos a un lenguaje entendible por el dispositivo que se encarga de manejar, por ejemplo Modbus.

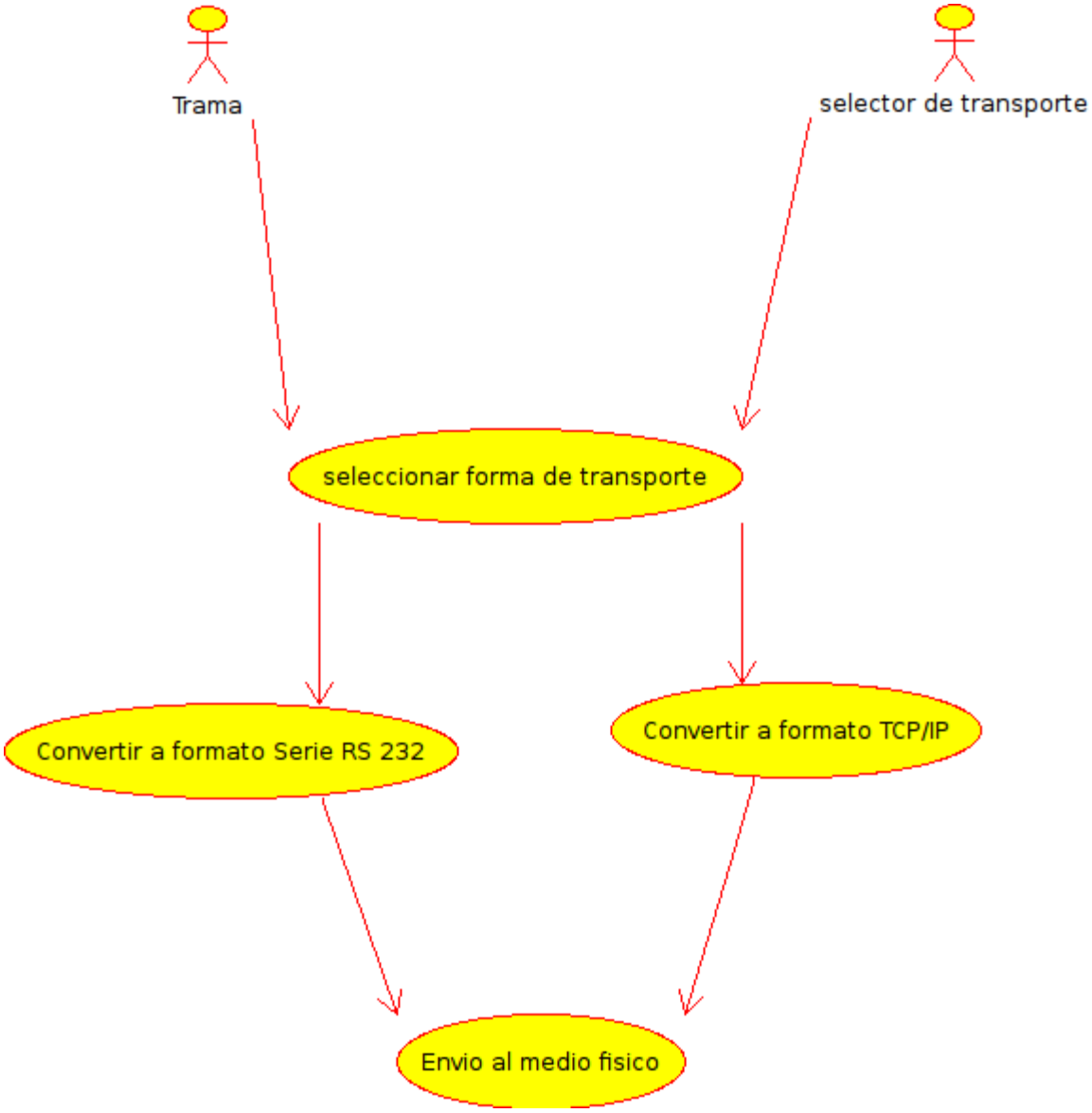

Figura 2.4 Caso de uso transporte.

En el caso de uso de transporte actúan dos actores los cuales son: la trama y el selector de transporte, a continuación se explican su procedencia y sus roles.

Trama: este actor del caso de uso de transporte proviene del controlador específico del protocolo al cual fue traducido de la trama inicial dada por el actor principal del sistema, este actor prácticamente solo pasa por esta capa ya que ella es la encargada de convertirla a formato del medio y enviarla.

Selector de transporte: este actor es generado por el sistema (la Interfaz Genérica) en dependencia del dispositivo con el cual se va a interactuar según los datos

suministrados por el actor principal, en la capa de transporte según este se define a que formato convertir la trama y se envía al medio físico.

A continuación se explican algunas funcionalidades de la aplicación a modo de ejemplos en los cuales los usuarios interactúan con ella desde una interfaz grafica de un SCADA.

2.3.2 Funcionalidades típicas

1 Nombre: Insertar controlador en la configuración.

Breve Descripción: Esta funcionalidad permite adicionar un nuevo controlador desde el despliegue de configuración.

Flujo:

a. La funcionalidad comienza cuando el usuario en el despliegue selecciona la opción insertar controlador.

b. El despliegue emite un diálogo para introducir el nombre de la biblioteca donde se encuentra el controlador. El usuario introduce o selecciona la biblioteca.

c. El Nodo de adquisición carga la correspondiente biblioteca (si no estaba cargada previamente) y a través de la función listDrivers le transfiere al despliegue la lista de controladores empotrados en la biblioteca. Esta lista se le muestra al usuario para que seleccione el controlador correspondiente. El usuario selecciona el controlador deseado.

d. El SCADA crea la instancia del controlador, tanto en la base de datos como en la biblioteca a través de la llamada a la función createDriver.

e. El SCADA invoca a la función getParametersValues para que el controlador creado le informe al despliegue de configuración, cuales son los parámetros que el controlador admite y sus correspondientes valores.

f. El despliegue de configuración le permite al usuario modificar los parámetros de configuración del controlador. Una vez terminado el proceso de edición se salvan los parámetros en la base de datos de configuración y se aplican al controlador invocando a la función setParametersValues.

2 Nombre: Insertar dispositivo en la configuración.

Breve Descripción: Esta funcionalidad permite adicionar un nuevo dispositivo a un controlador existente y cargado desde el despliegue de configuración.

Flujo:

a. La funcionalidad comienza cuando el usuario en el despliegue selecciona la opción insertar dispositivo.

b. El despliegue emite un diálogo para introducir el nombre del fabricante y el modelo del dispositivo a insertar. El usuario introduce estos datos.

c. El SCADA crea el dispositivo correspondiente a partir de la llamada a createDevice en el controlador y crea también el objeto persistente en la base de datos de configuración.

d. El despliegue emite un diálogo para introducir la dirección del dispositivo. El usuario introduce los datos. Cuando se oprime aceptar se verifica (utilizando la función validateDeviceAddress) si es una dirección admisible rechazándose la modificación si el resultado es incorrecto. Una vez introducida una dirección válida el SCADA guarda la misma en la base de datos de configuración y se aplica el cambio en el dispositivo invocando a la función deviceSetAddress.

e. El SCADA invoca a la función deviceGetParametersValues para que el dispositivo creado le informe al despliegue de configuración, cuales son los parámetros que el dispositivo admite y sus correspondientes valores.

f. El despliegue de configuración le permite al usuario modificar los parámetros de configuración del dispositivo. Una vez terminado el proceso de edición se aplican los parámetros al controlador invocando a la función deviceSetParametersValues y en caso de éxito se salvan en la base de datos de configuración.

3 Nombre: Insertar un punto en el dispositivo.

Breve Descripción: Esta funcionalidad permite insertar un punto en un dispositivo nuevo o ya existente desde el despliegue de configuración.

Flujo:

a. La funcionalidad comienza cuando el usuario en el despliegue selecciona la opción insertar un punto.

b. El despliegue emite un diálogo para introducir los atributos del punto. Cuando se desee introducir la dirección del punto el despliegue verifica (a través de la función de la interfaz getCapabilities) si el controlador admite la navegación en el espacio de direcciones del dispositivo.

c. Si no se permite la navegación el usuario introduce manualmente la dirección y se verifica mediante la función deviceValidVariableAddress si la dirección introducida es válida. Una vez introducida una dirección válida se pasa a e.

d. Si la navegación se permite el sistema crea un enumerador para ese dispositivo (Función deviceOpenEnumeration) e invocando repetidamente a enumerate obtiene todo el espacio de direcciones del dispositivo, mostrándoselos al usuario para que seleccione entre ellos la dirección deseada. Cuando se introduce la dirección correcta se pasa a e.

e. Se construye una nueva lista con las variables del dispositivo anteriormente existentes y la nueva variable y se invoca a deviceAttachVariableList para recrear la lista nuevamente con la adición de la nueva variable.

4 Nombre: Arranque del sistema.

Breve Descripción: Esta funcionalidad lleva todos los dispositivos del controlador a estado operativo en el arranque del sistema.

Flujo:

a. Para cada controlador configurado en la base de datos se hace lo siguiente:

b. Se crea el controlador mediante createDriver.

c. Se recuperan las propiedades del controlador desde la base de datos de configuración y se aplican mediante la invocación a setParametersValues.

d. Se recupera de la base de datos la lista de dispositivos asociados al controlador.

e. Para cada dispositivo configurado que está asociado a ese controlador se efectúa:

f. Se crea el dispositivo mediante createDevice.

g. Se recupera la dirección de ese dispositivo y se aplica mediante deviceSetAddress.

h. Se recuperan las propiedades del dispositivo desde la base de datos de configuración y se aplican mediante la invocación a deviceSetParametersValues.

i. Se recuperan los puntos asociados al dispositivo desde la base de datos de configuración.

j. Se asocian las variables al dispositivo mediante la invocación a deviceAttachVariableList.

k. El sistema le reporta el número de Bloques creados para ese grupo al Planificador.

l. Se finaliza el lazo comenzado en f.

m. Se finaliza el lazo comenzado en b.

2.3.3 Descripcion de las principales funciones de la Interfaz Genérica.

listDrivers

Esta función retorna la lista de manejadores que almacena la biblioteca.

DWORD listDrivers(

LPDWORD pBufferSize, //Dirección donde se almacena el tamaño del buffer.

LPMULTI_Z pBuffer); //Apunta al buffer donde se recibe la respuesta.

Parámetros

pBufferSize

Apunta a la variable donde se almacena el tamaño del buffer. La función debe chequear si este espacio es suficiente y devuelve en este parámetro el espacio usado en el buffer.

pBuffer

Apunta a una zona de memoria donde se almacenarán los nombres de los manejadores en formato de multicadena. El formato multicadena tiene múltiples cadenas individuales separadas por el carácter /0 añadiéndole a la última cadena un carácter /0 adicional. Si la función detecta que el espacio en el buffer es insuficiente el contenido del mismo queda indefinido. Este puntero puede ser nulo.

Valor de Retorno

Si la función tiene éxito devuelve cero. Si el tamaño del buffer especificado en el parámetro pBufferSize es insuficiente o si el parámetro pBuffer es nulo se devuelve errInsuficientBufferSize y en ese caso se deposita en la dirección a que apunta pBufferSize el tamaño necesario del Buffer.

Nota

Para asegurar el éxito de la función se puede proceder a llamarla dos veces. En la primera llamada se envía el tamaño del buffer en cero y pBuffer nulo. La función retornaría errInsuficientBufferSize y depositaría en la dirección a que apunta pBufferSize el tamaño requerido del Buffer. Con esta información se puede proceder a reservar el espacio suficiente en el buffer y recuperar la lista de manejadores en la segunda llamada a la función.

createDriver

La función createDriver crea una instancia de un Manejador y retorna un identificador que puede ser usado para acceder al objeto.

DWORD createDriver(

LPSTR drvName, // puntero al nombre del manejador

LPDWORD pDrvHandle); // dirección donde se almacena el identificador del manejador

Parámetros

drvName

Apunta a una cadena con terminador nulo que especifica el nombre del Manejador, cuya instancia se desea crear.

pDrvHandle

Apunta al identificador del objeto manejador que se crea en la llamada. Si la función tiene éxito se almacena en esta dirección un identificador válido. En caso contrario se deposita el valor cero en el parámetro.

Valor de Retorno

Si la función tiene éxito devuelve 0. En caso de que no se encuentre en la biblioteca un manejador cuyo nombre sea igual al parámetro drvName se devuelve el valor errDrvNotFound

Nota

Se admite la creación de varias instancias de un mismo tipo de manejador, por ejemplo si se tienen varias redes o canales que tienen el mismo protocolo y se desean controlar con el mismo manejador.

closeHandle

Esta función destruye la instancia dado su identificador. Permite destruir manejadores, dispositivos, grupos y enumeradores.

DWORD closeHandle(

DWORD handle); // identificador del objeto que se desea destruir.

Parámetros

handle

Identifica el objeto.

Valor de Retorno

Si la función tiene éxito devuelve 0. Si el parámetro handle no se corresponde con un identificador válido de un objeto creado previamente por una de las llamadas a: createDriver, createDevice, deviceCreateVariableList o deviceOpenEnumeration

se devuelve el valor errInvalidHandle. Si ocurre un error interno en la destrucción de la instancia correspondiente se retorna errDestroyError

Nota

Cualquiera sea el resultado de la llamada a closeHandle el identificador dejará de ser válido. Es responsabilidad del manejador destruir, en esta llamada, todas las estructuras asociadas a la instancia del objeto de manera que la memoria quede en un estado consistente. Si el manejador ha creado hilos internos para ese objeto debe finalizarlos y destruirlos. Este proceso no debe durar más de 5 segundos.

createDevice

La función createDevice crea una instancia de un Dispositivo asociada a un Manejador y retorna un identificador que puede ser usado para acceder al objeto.

DWORD createDevice(

DWORD drvHandle // Identificador válido de una instancia de manejador

LPSTR vendor, // puntero a una cadena que identifica al fabricante del equipo

LPSTR model, // puntero a una cadena que identifica el modelo del equipo

LPDWORD pDevHandle); // dirección donde se almacena el identificador del dispositivo

Parámetros

drvHandle

Identifica el objeto manejador.

vendor

Apunta a una cadena con terminador nulo que especifica el nombre del fabricante del dispositivo.

model

Apunta a una cadena con terminador nulo que especifica el modelo del dispositivo.

pDevHandle

Apunta al identificador del objeto dispositivo que se crea en la llamada. Si la función tiene éxito se almacena en esta dirección un identificador válido. En caso contrario se deposita el valor cero en el parámetro.

Valor de Retorno

Si la función tiene éxito devuelve 0. Si el parámetro drvHandle no se corresponde con una instancia del manejador creada previamente con la función createDriver se devuelve el valor errInvalidHandle.

Nota

Los parámetros vendor y model son opcionales. El manejador debe crear un dispositivo genérico si dichos parámetros se omiten o no se corresponden con valores reconocidos. Se admite la creación de varias instancias de un mismo dispositivo.

deviceRead

Esta función permite leer los valores de un conjunto de variables desde el dispositivo.

DWORD deviceRead(

DWORD devHandle, // Identificador válido de una instancia de dispositivo.

WORD blockNumber, // Número del Bloque a encuestar.

ReadyReadFunc readyRead); // CallBack que se llama cada vez que se

//deposita el valor de una variable del bloque.

Parámetros

devHandle

Identifica el dispositivo de encuesta.

blockNumber

Identifica el bloque de variables sobre el que se efectuará la operación de lectura. Debe ser uno de los reportados en la clasificación que realiza la función deviceAttachVariableList.

readyRead

Especifica un puntero a la dirección de la función que se llama cada vez que se deposita el valor de una variable del bloque. Esta función debe tener 2 parámetros, que se enumeran a continuación:

1. DWORD. Identificador (handle) del dispositivo.

2. LPVarLinkInfo. Puntero a la estructura VarLinkInfo de la variable cuyo valor fue cambiado.

La función readyRead no debe retornar ningún valor. Este puntero puede ser nulo, en cuyo caso no se produce el CallBack.

Valor de Retorno

Si la función tiene éxito devuelve 0. Si el identificador del Dispositivo es inválido se retorna errInvalidDevHandle. Si el número del Bloque especificado en blockNumber no se corresponde con uno de los bloques establecidos para ese dispositivo se devuelve errInvalidBlockNumber.

Nota

La función debe actualizar los campos value, timeStamp y quality de cada una de las variables que pertenecen al bloque. Cada vez que se lee una variable y se actualizan los valores mencionados se procede a llamar a la función de tipo callBack readyRead, para informarle al usuario del manejador que está listo el valor de la variable. Esta llamada es importante en el caso en que el protocolo admita la transferencia de secuencias de valores en el tiempo, ya que en este caso, pueden venir del dispositivo varios valores de una misma variable pero con diferentes marcas de tiempo. Si el usuario del manejador no reacciona ante el CallBack, el proximo valor que se reciba sobreescribirá al anterior en la estructura VarLinkInfo y se perderá esa medición.

deviceWrite

Esta función permite modificar el valor de un conjunto de variables que pertenezcan a una misma lista de variables en el dispositivo.

DWORD deviceWrite(

DWORD devHandle, // Identificador válido de una instancia de dispositivo.

DWORD num_items, // Número de variables que se modificarán.

LPVarOutInfo pVars); // Arreglo de estructuras VarOutInfo que contienen el

//identificador de las variables y el nuevo valor.

Parámetros

devHandle

Identifica el dispositivo.

num_items

Número de variables que se pretende modificar. Consecuentemente refleja el tamaño de los arreglos pVars y pResults

pVars

Arreglo de estructuras VarOutInfo que contienen la información de enlace de las variables y el nuevo valor. En el campo writeResult de la estructura se almacena, por el manejador, la aceptación de la salida por parte del dispositivo. Si el byte correspondiente se establece en cero significa que la modificación fue aceptada por el dispositivo (lo que no necesariamente significa que fue ejecutada).

Un valor diferente de cero significa un código de error. Si el campo varId de algún elemento de pVars no se encuentra en la lista de variables del dispositivo se retorna errVarNotFound en el byte correspondiente. Cualquier otro valor diferente de cero en writeResult indica un rechazo de la salida.

Valor de Retorno

Si la función tiene éxito devuelve 0. El éxito se mide por la aceptación por parte del dispositivo de la salida múltiple lo que no implica necesariamente que todas las variables se modifiquen. El criterio real de si las salidas fueron ejecutadas correctamente implica una relectura de los valores, lo cual debe preverse en las

capas superiores al manejador. Si el identificador del Dispositivo es inválido se retorna errInvalidDevHandle.

2.4 Conclusiones parciales

En el desarrollo de este capítulo se hizo la selección de las herramientas para el desarrollo de la aplicación teniendo siempre en cuenta las prestaciones de las computadoras en las cuales se iba a trabajar, los requerimientos para la implementación del proyecto y que fueran herramientas de software libre.

Con dichas herramientas se logra el diseño de un aplicación flexible y multiplataforma, basando este en la tecnología Qt y haciéndolo multicapa, ya que cada capa puede ser cambiada respetando la comunicación entre ellas y la aplicación seguiría funcionando igual.

Después de haber implementado la aplicación se llevarán a cabo pruebas y análisis de la misma y el reporte de estos conformarán el capítulo número 3 del informe.

EXPERIMENTO Y ANÁLISIS DE RESULTADOS

En el presente capítulo se llevarán a cabo experimentos de conexión utilizando la Interfaz Genérica y se analizarán los resultados obtenidos mediante el uso de aplicaciones sencillas con el propósito de probar el software.

3.1 Marco de la prueba

La prueba se realiza en un ordenador con Tarjeta madre AZUZ P5LD2-VM con Chipset Intel 945G, procesador Intel Pentium D a 3 GHz, memoria física 489 MB corriendo un Sistema Operativo Ubuntu versión 9.04 (Jaunty) con Linux 2.6.28-11-genérico y entorno de escritorio GNOME 2.26.1 con un PLC Siemens S7-200 por disponibilidad del equipo.

3.2 Implementación de las pruebas

- La primera prueba se realiza ejecutando una aplicación hecha para probar la función de escribir datos en varias áreas del PLC.

  Esta aplicación utiliza las funciones más básicas de la Interfaz para escribir en las salidas del PLC. Ahora se pasará a explicar paso a paso el código utilizado en la prueba.

```
#include <QtCore/QCoreApplication> // se incluye biblioteca del nucleo de Qt

#include <iostream> //biblioteca de entrada salida estandar de c++

#include "../interfaceG.h" // aquí se incluye el archive cabecera necesario para
```

```
// usar la interfaz.
using namespace std;
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    InterfaceG inter1 = new InterfaceG(); // se crea un objeto de tipo interface que
// tiene acceso a todas las funciones miembro de la Interfaz.
    inter1.createDriver("PPI",ppiIdent); // se crea un manejador de tipo PPI con
// puntero al identificador ppiIdent
    inter1.createDevice(ppiIdent,Siemensp,s7p,s7200ptr); // se crea un dispositivo
 // asociado al driver anteriormente creado y se almacena su identificador en
//  s7200ptr
    inter1.deviceSetAddress(s7200ptr,"/dev/ttyS0","9600",2,salidas); //se le añaden
// al dispositivo la dirección, velocidad, el slot del PLC y el área de memoria que se
// va a usar.
    int dat = 0; // se declara una variable entera y se inicializa en cero
    cout << "\n\n Ejemplo de prueba del driver..."<<endl ; // se imprime una línea en
// pantalla
    while (dat!=1000){ // un ciclo while para seguir escribiendo en las salidas hasta
// que se entre 100
    cout << "Entre el valor a escribir en las salidas del PLC (entero):" <<endl;
//  salida por pantalla
    cout << "Entre 1000 para parar el programa " // otra salida por pantalla
```

```
    cin >> dat; // se guarda el valor en la variable dat
    inter1.deviceWrite(s7200ptr,0,dat); // se llama la función a de escribir en el PLC.
// pasándole como dato lo que se entro por teclado.
    if (dat==1000) { // se comprueba si el numero entrado es 1000
        inter1.deviceWrite(s7200ptr,0,0); // si el numero es 1000 se resetean las
// salidas del PLC antes de salir del ciclo.
}
}
    return a.exec();
}
```

Con la ejecución del programa anterior se pudo comprobar el correcto funcionamiento de las bibliotecas de la Interfaz al poder apreciar el cambio de estado en las salidas del PLC al introducirle distintos números mientras la aplicación estuvo en ejecución.

- La segunda prueba se trata de con el mismo programa anterior, pero se le cambia el área del PLC donde se escribe, es decir en este caso se escribe en un área de memoria del PLC llamada DB que es el área para de datos.

  Se ejecuta el programa y se le pasa la cadena “prueba de escritura y lectura en el plc”.

  El principal objetivo de esta prueba es comprobar el correcto funcionamiento de la lectura del PLC ya que la escritura habíamos visto su correcto funcionamiento den el programa anterior.

- La tercera prueba se ejecuta el siguiente programa:

```
#include <QtCore/QCoreApplication> // se incluye biblioteca del nucleo de Qt
#include <iostream> //biblioteca de entrada salida estandar de c++
#include "../interfaceG.h" // aquí se incluye el archive cabecera necesario para
```

```
// usar la interfaz.
using namespace std;
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    InterfaceG inter1 = new InterfaceG(); // se crea un objeto de tipo interface que
// tiene acceso a todas las funciones miembro de la Interfaz.
    inter1.createDriver("PPI",ppiIdent); // se crea un manejador de tipo PPI con
// puntero al identificador ppiIdent
    inter1.createDevice(ppiIdent,Siemensp,s7p,s7200ptr); // se crea un dispositivo
 // asociado al driver anteriormente creado y se almacena su identificador en
//  s7200ptr
    inter1.deviceSetAddress(s7200ptr,"/dev/ttyS0","9600",2,DB); //se le añaden
// al dispositivo la dirección, velocidad, el slot del PLC y el área de memoria que se
// va a usar.
    cout << "Se va a leer el contenido de el área DB del PLC <<endl; " // salida por
// pantalla
string dat = 0;
inter1.deviceRead(s7200ptr,0,dat); // se llama la función para leer del PLC.
cout  << “el contenido almacenado es”: << dat <<endl;
    return a.exec();
}
```

La salida del anterior programa fue exactamente la entrada en la prueba número 2.

3.3 Análisis económico

Para hacer un análisis económico de este trabajo se debe hablar de su contribución a un proyecto mayor. Con la puesta a punto de este proyecto, un SCADA basado en software libre cuyas prestaciones abarquen las necesidades de las industrias locales y regionales, el país se ahorraría miles de dólares debido, primero a la diferencia de precio con los sistemas actuales del mercado, y después a que no se le estaría pagando a ninguna corporación extranjera sino que todo quedaría en empresas cubanas, por lo que no habría esa fuga de capitales que tanto nos afecta.

3.4 Conclusiones parciales

Como conclusiones se puede decir que la implementación de la aplicación fue satisfactoria, ya que permite la interacción con el dispositivo de campo.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

El trabajo desarrollado ha puesto en evidencia algunos aspectos que permiten enunciar las siguientes conclusiones:

1 En el mundo existe muy poca documentación disponible sobre las Interfaces de Comunicación con dispositivos de campo.

2 Existen muchas herramientas de Software Libre con calidad para la realización de proyectos de software para el control de procesos, las cuales permitieron cumplir con los requisitos planteados en el trabajo y obtener un producto flexible, escalable y multiplataforma.

3 Se logró la implementación de un programa que demuestra la posibilidad de comunicación con dispositivos que soporten la comunicación Modbus o PPI.

4 La realización de este proyecto constituye un punto de partida para el desarrollo de aplicaciones de supervisión y control tan necesarias en la industria.

Recomendaciones

Durante el desarrollo del proyecto se ha visto que las posibilidades van más allá de lo que hasta hoy se ha hecho, por lo que se recomienda:

1 Aumentar el número de protocolos soportados implementando más controladores.

2 Incrementar las funcionalidades de la Interfaz Genérica para dar más facilidades a los programadores de interactuar con los controladores.

3 Divulgar los resultados obtenidos para lograr el interés de otros proyectos en la aplicación.

4 Completar la documentación del software en formato accesible a la mayoría de los desarrolladores.

REFERENCIAS BIBLIOGRÁFICAS

BABYLON (2009) Definición Interfaz. Disponoble en: http://diccionario.babylon.com/Interfaz.

BAILEY, D. & WRIGHT, E. (2003) *Practical SCADA for Industry (IDC Technology)*.

BLANCHETTE, J. & SUMMERFIELD, M. (2006) *C++ GUI Programming with Qt 4*, Prentice Hall.

BOYER, S. A. (1999) *SCADA: Supervisory Control and Data Acquisition*.

COMUNITY, E. (2009) Eclipse Documentation. Disponoble en: http://help.eclipse.org/ganymede/index.jsp.

ECKEL, B. (2000) *Thinking in C++*, Prentice Hall.

FRANCISCO, M. (2000) *Introducción a la OOP*, Grupo EIDOS.

GONZÁLEZ, R. (2008) Comunicación Personal. Santa Clara. Disponoble en:

HENSGEN, P. (2003) Manual de Umbrello UML Modeller. Disponoble en: http://docs.kde.org/kde3/es/kdesdk/umbrello/.

HERRERA, M. (2008) Comunicación Personal. Santa Clara. Disponoble en:

JACOBSON, BOOCH & RUMBAUGH (1999) El Proceso Unificado de Desarrollo de Software. Addison Wesley. Disponoble en:

KRILL, P. (2009) The Windows-versus-Linux server face-off. Disponoble en: http://www.itworld.com/windows/63231/windows-versus-linux-server-face.

MARTÍN, A. L. (2005) Formatos de Audio Digital. Disponoble en: http://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_01_02/formatos_audio_digital/html/frames.htm.

MICROSOFT (2008) Compare Windows to Linux. Disponoble en: http://www.microsoft.com/windowsserver/compare/windows-server-vs-red-hat-linux.mspx.

NEUSTADT, I. & ARLOW, J. (2006) *Uml 2*, ANAYA MULTIMEDIA.

NOKIA (2009a) Clientes. Disponoble en: http://www.qtsoftware.com/products/.

NOKIA (2009b) Qt Creator manual. Disponoble en: http://doc.trolltech.com/qtcreator-1.1/index.html.

SIECHERT, C. & BOTT, E. (2005) *Seguridad En Microsoft Windows Xp Y Windows 2000*, McGRAW-HILL/INTERAMERICANA DE ESPAÑA.

STROUSTRUP, B. (1999) *The Design and Evolution of C++*, Addison-Wesley.

TROLLTECH (2009) Qt Development Tools. Disponoble en: http://www.qtsoftware.com/products/developer-tools.

WALKER, A. (2006) *Seguridad, Spam, Spyware Y Virus*, ANAYA MULTIMEDIA.

WIKIBOOKS (2009) Programación con Qt4. Disponoble en: http://es.wikibooks.org/wiki/Programaci%C3%B3n_con_Qt4.

WIKITIONARY (2008) Definición de Interfaz. Disponoble en: http://es.wiktionary.org/wiki/interfaz

ANEXOS

Anexo 1. Programa de pruebas para el driver de Siemens.