Manejador para la comunicación fiable en un sistema de supervisión y control basado en microcontroladores.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autor: Héctor Ernesto León García.

Tutor: Ing. Antonio Cedeño Pozo.

La Habana junio 2012
"Año 54 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro ser el único autor del presente trabajo de diploma y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

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

_____

Firma del Autor.
Héctor Ernesto León García.

_____

Firma del Tutor.
Antonio Cedeño Pozo.

DATOS DE CONTACTO.

Nombre y apellidos del tutor: Antonio Cedeño Pozo.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

e-mail: acedeno@uci.cu. Especialista graduado en la UCI, en el año 2009. Profesor Instructor. Seis años de experiencia en la producción de software en el área de la informática aplicada a la automática.

Agradecimientos.

A Yadiel, por todo lo que ha hecho por muchos de nosotros y en especial por todo lo que hizo por mí.

Al Lito y a Jonnys por mantener el piquete unido.

A mi novia Amanda por quererme lo que nadie en estos últimos años y aguantar todas mis pesadeces.

A mi familia por apoyarme y quererme como soy.

A la gente del grupo 5101 que ha sido por mucho el mejor grupo de todos durante estos 5 años aunque ya no estén todos.

A la gente del proyecto Ale, Karel, Alvaro, Vilches, Manolo, el profesor Espí por siempre ayudarme cuando los necesité y ellos pudieron.

A toda la gente que me tiene entre sus amigos.

A mi tía Rosa por ser la persona más fuerte que conozco, por enseñarme de la forma más difícil que la vida dispara balas a las que sólo queda meterle el pecho.

A mi abuelita Cuca por estar este día y darme la satisfacción de que pudiera verme graduado.

A mi nueva familia Yacnelys, Ricardo, Karla, Ricardito por cuidar y querer a mi mamá y a mí.

A mi suegra Ana que más que mi suegra ha sido otra madre.

Al piquete del fútbol por todas las tardes que estuvieron ahí y sin saberlo me ayudaron a seguir cuando los ánimos estaban por el suelo.

A la persona que todos estos años a tenido que buscar tiempo para resolver sus problemas y los míos, a la que a dedicado su vida entera a mí, a la que le debo todo lo que soy y lo que seré, a quien con una sonrisa pone fin a todos mis problemas, a mi faro, mi guía, a mi madre.

Dedicatoria.

A mi madre, la mayor responsable de este trabajo.
A Amanda por quererme tanto.

Resumen.

En el presente trabajo se describe el desarrollo de un manejador a partir del diseño de un protocolo de comunicación. La necesidad del desarrollo del mismo surge, debido a que en la Línea de Sistemas Empotrados del Centro de Desarrollo de Informática Industrial (CEDIN), de la Facultad 5, de la Universidad de las Ciencias Informáticas (UCI) se está desarrollando un sistema de supervisión y control denominado **G-Hawk** haciendo uso de dos tarjetas basadas en microcontroladores, la STM32 y la CID-300/9. Una parte importante del sistema que se pretende desarrollar es la comunicación entre las dos tarjetas, por lo que existe la necesidad de lograr el intercambio de información de manera fiable entre los dispositivos mencionados.

El proceso de desarrollo comienza con un análisis de los principales sistemas SCADA en Cuba y el resto del mundo, haciendo énfasis en las características de los manejadores de dispositivos. Se realiza un estudio de los protocolos más usados del modelo TCP/IP. Se hace uso de las características esenciales de una serie de protocolos de comunicación, con el objetivo de alcanzar conocimientos conceptuales y estructurales para el diseño del protocolo **ProC-ARM**, ideado para ésta investigación. Seguidamente se lleva a cabo la justificación del grupo de tecnologías y herramientas de desarrollo a utilizar en el diseño e implementación del manejador. El desarrollo del manejador se basa en el protocolo antes mencionado, que se encarga de regir las características de la comunicación entre los dispositivos del sistema en cuestión. Por último se concluye el proceso de desarrollo con pruebas que garanticen el funcionamiento correcto del manejador.

Palabra claves: manejador, microcontroladores, protocolo, SCADA.

Índice general

Índice de figuras

Índice de cuadros

Introducción

Prácticamente todas las esferas de la vida del hombre han sido tocadas de una u otra forma por el fenómeno denominado transferencia de datos. Los avances mostrados por las tecnologías de punta en la transferencia de información han revolucionado el mundo en los últimos años. La producción industrial, determinado en gran medida por el grupo de actividades que realiza como son el envío, la recolección, el manejo o el almacenamiento de datos, es el sector que ha percibido con mayor fuerza la llegada de estas nuevas tecnologías. Algunos años atrás se hubiera tildado de utopía que valores obtenidos por sensores en algún lugar del amplio ambiente industrial pudieran ser enviados hasta el ordenador del operador que estuviera controlando dicha actividad, pero actualmente, el conjunto de herramientas y tecnologías existentes permite que este proceso, antes soñado, se realice en cuestiones de segundos.

La adquisición, monitoreo y control de datos son utilizados en el panorama industrial con el fin de dar solución a problemas que afectan la producción, la eficiencia y la calidad de los procesos en los que las industrias se ven enroladas. La adquisición de información se desarrolla a partir de una combinación de hardware y software que permite poner a disposición del operador, datos para su posterior estudio, evaluación o para el control de los mismos.

La automatización de labores y el control automático de tareas, a partir de dispositivos para un desarrollo más rápido, óptimo y susceptible a errores, son elementos necesarios para satisfacer las necesidades de las industrias en lo que a calidad y control de sus procesos se refiere.

La Universidad de las Ciencias Informáticas (UCI) es un centro de altos estudios creado por la Revolución Cubana para la formación de profesionales a partir de un modelo pedagógico novedoso y la vinculación del estudio con la producción y la investigación. La Universidad cuenta con una estructura de producción formada por centros que se enfocan en una temática para dirigir su investigación y desarrollar productos de software que puedan satisfacer necesidades reales, ya sea para clientes nacionales o foráneos. La Facultad 5 de la Universidad cuenta con dos centros: el Centro de Desarrollo de Arquitectura Empresarial (CDAE) y el Centro de Desarrollo de Informática Industrial (CEDIN en lo adelante). El CEDIN tiene entre sus

objetivos brindar soluciones para la automatización de procesos industriales, aunque la mayoría de las soluciones sólo contemplan el componente de software sobre computadoras; pero, para ser capaces de brindar soluciones integrales es necesario desarrollar sistemas que puedan completar la pirámide de automatización.

La Línea de Sistemas Empotrados del CEDIN tiene como principal objetivo el desarrollo de productos de software para sistemas con pequeña capacidad de cómputo. En la Línea se cuenta con dos tarjetas basadas en microcontroladores, la de menos recursos de hardware fue diseñada a partir de un microcontrolador STM32 y puede ser utilizada para desarrollar tarjetas de adquisición de datos de procesos industriales; por otro lado, la segunda tarjeta tiene mayores prestaciones de hardware, hasta niveles en que se puede empotrar algún tipo de sistema operativo basado en Linux. Esta última fue desarrollada completamente por el ICID (Instituto Central de Investigaciones Digitales) y se denomina CID-300/9.

En la Línea se está desarrollando un sistema de supervisión y control denominado **G-Hawk** utilizando las tarjetas basadas en microcontroladores mencionadas anteriormente. Se pretende utilizar al dispositivo de menos recursos de hardware (basada en el microcontrolador STM32) como tarjeta de adquisición de datos, y a la CID-300/9 como núcleo de procesamiento y módulo de visualización aprovechando las potencialidades de la misma y la pantalla LCD que trae integrada. Una parte importante del sistema que se pretende desarrollar es la comunicación entre las tarjetas mencionadas para el intercambio de información desde lo que sería la tarjeta de adquisición de datos hacia el núcleo de procesamiento y visualización, en ese sentido, existe la necesidad de lograr intercambiar información de manera fiable entre las partes mencionadas.

Ante la situación problemática expuesta se formula el siguiente **problema científico**:

¿Cómo garantizar el intercambio de información, de manera fiable, entre el núcleo de procesamiento y la tarjeta de adquisición de datos en un sistema de supervisión y control basado en microcontroladores?

Objeto de estudio:

Sistemas de supervisión, control y adquisición de datos.

Campo de acción:

Manejadores de dispositivos en sistemas de supervisión, control y adquisición de datos.

De acuerdo con el problema científico planteado se define la siguiente **idea a defender**: Desarrollando un manejador a partir del diseño de un protocolo de comunicación, se garantizará el intercambio de información de manera fiable entre el núcleo de procesamiento y la tarjeta de adquisición en un sistema de supervisión y

control basado en microcontroladores.

Se propone como **objetivo general** de la investigación: Desarrollar un manejador a partir del diseño de un protocolo que permitirá la comunicación en un sistema de supervisión y control, basados en microcontroladores.

Para dar cumplimiento al objetivo planteado, se propone el desarrollo de las siguientes **tareas de investigación**:

1. Estudio de los sistemas de supervisión y control existentes a nivel mundial haciendo énfasis en los manejadores de dispositivos.
2. Estudio de los protocolos de la capa de transporte del modelo TCP/IP.
3. Selección de las tecnologías y herramientas a utilizar para el desarrollo del manejador.
4. Diseño del protocolo de comunicación.
5. Planificación del proyecto.
6. Diseño del manejador de dispositivo a partir del protocolo de comunicación.
7. Implementación del manejador.
8. Realización de pruebas funcionales.

Para el desarrollo de las tareas se emplean los siguientes métodos de investigación científica:

Métodos teóricos:

- Análisis histórico y lógico: Con el objetivo de analizar la situación actual de los manejadores en los sistemas de supervisión y control, su funcionamiento y principales características.
- Análisis y síntesis: En el análisis de las bibliografías existentes sobre los manejadores en los sistemas de supervisión y control, obteniendo de manera sintetizada el contenido necesario y suficiente para la realización del presente trabajo.
- Modelación: En la elaboración de los diagramas necesarios para el diseño e implementación del manejador que permitirá la comunicación fiable en el sistema G-Hawk.

- Hipotético deductivo: Contribuye a la revisión y justificación del modelo y las tecnologías que se utilizará para desarrollar el manejador que permitirá la comunicación fiable en el sistema G-Hawk .

Métodos empíricos:

- Observación científica: Para realizar el diagnóstico del problema a investigar y para complementar el diseño de la investigación.

- Revisión de la documentación: Para seleccionar la información necesaria en la investigación a partir del estudio de documentos y diferentes bibliografías.

Al finalizar la investigación se obtendrá como **resultado**:

Un manejador que permitirá la comunicación, de manera fiable, entre el núcleo de procesamiento y las tarjetas de adquisición para sistemas de supervisión y control basados en microcontroladores.

El documento, en el que se expone todo el desarrollo de este trabajo de diploma, está estructurado en tres capítulos; los cuales están divididos en epígrafes y subepígrafes. El primer capítulo aborda la fundamentación teórica y la selección de las herramientas y tecnologías utilizadas. El segundo capítulo trata el diseño de la solución propuesta y las características con que contará el sistema. En el tercer capítulo se expone la implementación del manejador de dispositivos así como las pruebas realizadas a las funcionalidades desarrolladas.

1 Fundamentación teórica

Introducción

En este capítulo se realiza un estudio de los sistemas de supervisión y control, haciendo especial énfasis en sus principales características, así como en sus manejadores de dispositivos. Además, se realiza la selección de las herramientas para el diseño, la modelación y posterior implementación de la solución.

1.1. Sistemas de supervisión y control.

SCADA, proviene de las siglas Supervisory Control And Data Acquisition (Supervisión, Control y Adquisición de Datos) y constituye un software de control de producción, que se comunica con dispositivos de campo y controla el proceso de forma automática desde la pantalla del ordenador. Estos sistemas proporcionan información del proceso a diversos usuarios: operadores, supervisores de control de calidad, supervisión y mantenimiento [12].

Un sistema SCADA permite que un operador, en una estación central a grandes distancias de la ubicación de los procesos industriales, pueda hacer ajustes o cambios en los controladores locales de los procesos. Tal es el caso de abrir o cerrar válvulas a distancias, conocer el estado de los interruptores de seguridad de un sistema, monitorizar el estado de las alarmas del proceso y obtener información de las variables del proceso involucradas. Cuando la distancia llega a ser muy grande: cientos o miles de kilómetros desde un punto a otro, los beneficios en reducir costos de visitas de rutinas pueden ser apreciados. Los programas necesarios y en su caso, el hardware adicional que se necesite, se denomina, en general, sistema SCADA. Existen diversos tipos de SCADA dependiendo del fabricante y sobre todo de la finalidad con que se va a hacer uso del sistema. Pero, existen una serie de características con que debe cumplir todo sistema SCADA [12]:

- Arquitectura abierta, debe permitir su crecimiento y expansión, así como deben poder adecuarse a las necesidades futuras del proceso y de la planta.

- La programación e instalación no debe presentar mayor dificultad, debe contar con interfaces gráficas que muestren un esquema básico y real del proceso.
- Deben permitir la adquisición de datos de todo equipo, así como la comunicación a nivel interno y externo (redes locales y de gestión).
- Deben ser programas sencillos de instalar, sin excesivas exigencias de hardware, y fáciles de utilizar, con interfaces amigables para el usuario.

Un SCADA cuenta con una serie de características que dependen del proceso en el cual será usado. El apoyo en las características de estos sistemas y la realización de un estudio de sus principales funciones permite crear una base desde el punto de vista funcional y estructural para el sistema a desarrollar en la investigación.

1.1.1. Funciones principales de los sistemas SCADA

Este tipo de software es utilizado fundamentalmente en el sector industrial. Abarca desde las soluciones para la captura de información de un proceso o planta industrial hasta aquellos análisis con los que se pueden obtener importantes indicadores que permiten una realimentación del proceso. De forma general se consideran como funciones básicas de un sistema SCADA las enunciadas a continuación [12]:

- Supervisión remota de instalaciones y equipos: Permite al operador conocer el estado de desempeño de las instalaciones y los equipos alojados en la planta, lo que permite dirigir tareas de mantenimiento y estadísticas de fallas.
- Control remoto de instalaciones y equipos: Mediante el sistema se pueden activar o desactivar los equipos remotamente (por ejemplo abrir válvulas, activar interruptores, prender motores, etc.), de manera automática y también manual. Además es posible ajustar parámetros, valores de referencia, algoritmos de control, etc.
- Procesamiento de datos: El conjunto de datos adquiridos conforman la información que alimenta el sistema, esta información es procesada, analizada, y comparada con datos anteriores, y con datos de otros puntos de referencia, dando como resultado una información confiable.
- Visualización gráfica dinámica: El sistema es capaz de brindar imágenes en movimiento que representen el comportamiento del proceso, dándole al operador la impresión de estar presente dentro de una planta real. Estos gráficos también pueden corresponder a curvas de las señales analizadas en el tiempo.

- Generación de reportes: El sistema permite generar informes con datos estadísticos del proceso en un tiempo determinado por el operador.
- Representación de señales de alarma: A través de las señales de alarma se logra alertar al operador frente a una falla o la presencia de una condición perjudicial o fuera de lo aceptable. Estas señales pueden ser tanto visuales como sonoras.
- Almacenamiento de información histórica: Se cuenta con la opción de almacenar los datos adquiridos, esta información puede analizarse posteriormente, el tiempo de almacenamiento dependerá del operador o del autor del programa.
- Programación de eventos: Esta referido a la posibilidad de implementar subprogramas que brinden automáticamente reportes, estadísticas, gráfica de curvas, activación de tareas automáticas, etc.

El procesamiento de datos es una de las principales funciones de los sistemas SCADA. El control sobre los datos que se manejan y el nivel de fiabilidad que se pueda obtener en el proceso de transmisión es proporcional a la calidad del sistema. La eficiencia de un SCADA estará dada en gran medida por la fiabilidad de su módulo de adquisición y procesamiento de datos.

1.1.2. Ejemplos de sistemas SCADA

Los sistemas SCADA son en su mayoría privativos y se ejecutan sobre Windows[1], por tanto suelen tener un alto costo en el mercado. Algunos como el SCADA "Guardián del Alba" (GALBA en lo adelante) fueron creados bajo los preceptos de software libre.

El GALBA surge a partir de la necesidad de la empresa Petróleos de Venezuela - Sociedad Anónima (PDVSA) de automatizar y gestionar sus procesos industriales, controlar oleoductos, sistemas de transmisión de energía eléctrica, yacimientos de gas y petróleo y el control de todos los procesos de esta rama, redes de distribución de gas natural, subterráneos, generación energética, sistemas de distribución de agua, entre otros.

Lookout, National Instruments: Lookout es una poderosa herramienta de software (SCADA) de fácil uso para la automatización industrial. Se comunica con

[1]Sistema Operativo

dispositivos E/S[2] ubicados en campo mediante PLC[3], RTU[4] y otros dispositivos. Algunos proyectos típicos de Lookout incluyen control, monitoreo y supervisión continua de procesos, fabricación discreta, y sistemas de telemetría remota. Lookout permite representaciones gráficas sobre la pantalla de una computadora de dispositivos reales tales como interruptores, escalas gráficas, registradores de eventos, botones pulsadores, etc. Enlaza sus imágenes a los actuales instrumentos de campo usando PLC, RTU, u otros dispositivos de E/S[12].

Fast/Tools: Fast/Tools se ha desarrollado a lo largo los últimos veinte años mediante la combinación de experiencia en supervisión, control y adquisición de datos con los requisitos de usuarios y las industrias líderes. El sistema tiene éxito en las aplicaciones críticas debido a las características de su diseño, el apoyo de los sistemas de parada y no redundantes y su rendimiento muy alto en línea, así como sus posibilidades de configuración. Estas propiedades contribuyen a una alta eficiencia y producción de procesos de alta calidad. Fast/Tools se ha evolucionado con la idea de proporcionar un sistema que comienza pequeño y que puede crecer con las necesidades del cliente, tanto en corto como en un largo plazo de tiempo. Las características de ser un proveedor independiente del hardware, significa compromiso con los estándares de los sistemas operativos, los protocolos de red estándar, interfaces de usuario estándar y el estándar de herramientas de desarrollo de software[25].

RSView32: RSView32 es un software para crear y ejecutar aplicaciones de adquisición de datos, supervisión y control. Diseñado para su uso en diferentes sistemas operativos tales como Microsoft Server 2003, Windows XP y entornos de Windows 2000. Contiene las herramientas necesarias para la creación de una interfaz hombre-máquina, incluido los dibujos de animación en tiempo real, elaboración de gráficos de tendencias, alarmas y resúmenes. Permite la integración con los productos de otros fabricantes, con el fin de maximizar tecnologías[24].

Actualmente los SCADA son empleados en una gran variedad de aplicaciones de diversas industrias: pesadas, ligeras o de bienes. El elevado costo de los SCADA en el mercado mundial se ha convertido en uno de los factores revolucionarios de la industria de la producción de software, influenciado a algunas empresas a trabajar en el desarrollo de sistemas bajo los preceptos del software libre. Otras industrias deciden por alternativas diferentes, dedican sus esfuerzos y recursos a adquirir sistemas que si bien no llegan a cumplir con todas las funcionalidades de un SCADA, permiten realizar supervisión y control sobre sus procesos sin tener que emplear en ello elevadas sumas de dinero.

[2]Entrada/Salida
[3]Controlador Lógico Programable siglas en inglés
[4]Unidad Terminal Remota siglas en inglés

1.1.3. Manejadores de dispositivos en sistemas SCADA

El subsistema de adquisición de datos es el encargado de la transferencia de los datos entre el sistema SCADA y el conjunto de dispositivos que se encuentran en el campo. Este subsistema está formado por diferentes componentes entre los que podemos encontrar los manejadores de dispositivos y el módulo de recolección de datos. Un manejador o driver (en inglés) es un programa que permite al sistema operativo interactuar con un dispositivo, haciendo una abstracción del hardware y proporcionando al usuario una interfaz para usarlo[27].

La mayoría de las empresas utilizan estándares internacionales para definir los protocolos mediante los cuales será posible la comunicación con los elementos de hardware que fabrican. Las formas en que los instrumentos de campo se comunican con otros dispositivos son muy variadas, típicamente tienen integradas interfaces de comunicación, tales como: RS232, RS485, RS222 y Ethernet; sobre estos tipos de interfaces las comunicaciones se establecen utilizando diferentes protocolos o lenguajes de comunicaciones, por ejemplo: Modbus, EthernetIP, Profibus, DH+, DF1, DNP, Device Net, Control Net, etc[5].

Generalmente los fabricantes proveen uno o varios manejadores para sus dispositivos. Conociendo la especificación del protocolo utilizado, los controladores pueden ser desarrollados por terceros, en algunas oportunidades los fabricantes de elementos de hardware no siguen exactamente estándares de protocolos conocidos; sino que realizan algunas modificaciones en su implementación, en esos casos sería muy complicado desarrollar el manejador correspondiente, resultando necesario conocer los cambios realizados o los elementos introducidos[5].

1.2. Protocolos de comunicación.

Un protocolo de comunicación es un grupo de reglas que garantizan el intercambio de información entre un conjunto de dispositivos siempre que estos conformen una red. Entre los protocolos propios de una red de área local podemos distinguir dos principales grupos. Por un lado están los protocolos de los niveles, físico y de enlace del modelo OSI, estos definen las funciones que se asocian con el uso del medio de transmisión. Se refieren al envío de los datos a nivel de bits y trama; y el modo de acceso de los nodos al medio. Además están determinados sólo por el tipo de red que utilizan. En el segundo grupo se encuentran aquellos que realizan las funciones de los niveles de red y transporte de OSI, estos son los encargados del envío y recibo de la información y de garantizar una comunicación extremo a extremo, libre de errores. Estos protocolos transmiten la información a través de la red utilizando pequeños

segmentos denominados paquetes. Cada uno define su propio formato de paquetes, en el que se especifican elementos como: el origen, destino, longitud, tipo del paquete y la información redundante para el control de errores[22].

En los protocolos se pueden reconocer algunas propiedades, por ejemplo:

- Rigen los pasos para comenzar la comunicación entre dos nodos.
- Determinan el inicio y el fin de los mensajes, así como el resto del formato de los mismos.
- Realizan la corrección de los errores.
- Marcan la terminación de la sesión de conexión.
- Plantean estrategias de seguridad[5].

El diseño de un protocolo que se encargue de regir los aspectos esenciales de la comunicación entre el manejador y los dispositivos, con los que se comunique, es un eslabón fundamental en el proceso de lograr la transferencia de información fiable entre éstos. Los protocolos de comunicación se encargan de definir la estructura de los paquetes o tramas que se utilizan en la transferencia de información. Fundamentalmente especifican destino, longitud de los datos, los datos y la información fundamental para el control de errores.

1.2.1. Modbus

El protocolo de comunicaciones industriales Modbus fue desarrollado en 1979 por la empresa norteamericana MODICON. Debido a que es público, relativamente sencillo de implementar y flexible, se ha convertido en uno de los protocolos de comunicaciones más populares en sistemas de automatización y control. Éste funciona siempre en modo maestro-esclavo (cliente- servidor), siendo el maestro (cliente) quien controla en todo momento las comunicaciones con los esclavos (servidores) que pueden ser hasta 247. Además las comunicaciones MODBUS se pueden realizar en modo ASCII[5] o en modo RTU[6]. En modo ASCII, por cada byte a transmitir se envían dos caracteres con su representación hexadecimal y en modo RTU se envían en binario. En el modo ASCII las tramas comienzan por 3AH (carácter ':'), y terminan en 0DH-0AH (CR LF Carrier Return Line Feed). En modo RTU no se utiliza indicador de inicio y final de trama. Se utiliza un sistema

[5]Código estándar para el intercambio de información
[6]Código binario

de detección de errores diferente dependiendo del tipo de codificación utilizado. En el caso de la codificación ASCII es el checksum (LRC[7]). En la codificación RTU se utiliza el método de CRC[8] [13].

1.2.1.1. Modbus/TCP

Modbus/TCP es una variante de la familia de protocolos de comunicación Modbus. En concreto, se refiere al uso de la mensajería Modbus en un medio ambiente "Intranet" o "Internet" , utilizando los protocolos TCP/IP. El uso más común del protocolo en este momento es para la conexión Ethernet de PLC, módulos E/S de dispositivos de campo y módulos de E/S de redes. Modbus/TCP hace uso de una representación "big-endian" para las direcciones y datos. Esto significa que cuando un cantidad numérica mayor que un solo byte se transmite, el byte más significativo se envía primero. Además basa su modelo de datos en una serie de tablas que tienen características distintivas. Todas las peticiones y respuestas Modbus/TCP están diseñadas de tal manera que el destinatario puede verificar que un mensaje está completo. La información adicional de longitud se encuentra en la cabecera del mensaje para que el destinatario reconozca los límites, incluso si el mensaje tuvo que ser dividido en varios paquetes para la transmisión. La existencia de normas explícitas e implícitas de longitud, y el uso de un código de comprobación de error (CRC-32) ofrece como resultado una posibilidad infinitesimal de que una solicitud o un mensaje de respuesta llegue con un formato corrupto[26].

1.2.2. BSAP

El Protocolo BSAP (Bristol Synchronous/Asynchronous Protocol en inglés) es un protocolo industrial utilizado para el control y supervisión de sistemas SCADA. Es muy completo con una topología tipo árbol con un máximo de seis niveles y 127 nodos por nivel; a su vez, cada nodo puede controlar hasta 127 dispositivos remotos, cuenta con una dirección única basada en su posición en la red y puede ser maestro de los niveles inferiores o esclavo de los niveles superiores [15].

Características del Protocolo:

- Control por caracteres (Modo transparente).
- Transmisión Asíncrona/Síncrona.

[7]Chequeo de Longitud Redundante siglas en inglés
[8]Chequeo Cíclico Redundante siglas en inglés

- Topología tipo árbol; en la raíz se encuentra la MTU[9].
- Carácter básico codificado en ASCII sin bit de paridad.
- Interfaces de capa física: RS-232C, RS-422A, RS-423A y RS-485.
- Velocidades de transmisión: Síncrona: 187,5 kbps, 1 Mbps Asíncrona: 300 bps a 38,4 kbps.
- Medios de transmisión: Par trenzado, cable coaxial, radio.

1.2.3. DF1

El DF1 es un protocolo ASCII, asíncrono, orientado a bytes que se utiliza para comunicarse con la mayoría de los módulos de la interfaz del Allen Bradley RS232[10]. DF1 es un protocolo que se puede encontrar en modo Full-dúplex y modo Half-duplex.

El protocolo DF1 Full-dúplex se usa sobre un enlace punto a punto que permite dos formas de transmisión simultánea y sobre un enlace multipuntos donde los módulos de la interfaz son capaces de transmisiones arbitrarias en el enlace. Además para aplicaciones en las que es necesario obtener el más alto rendimiento posible del medio de comunicación[22].

El Protocolo DF1 Half-duplex es una versión multi-niveles del protocolo para un maestro y uno o más esclavos. Se pueden tener de 2 a 255 nodos conectados simultáneamente en un único enlace, este enlace opera con todos los nodos interconectados a través de módems Half-duplex. En este tipo de comunicación se designa un nodo como maestro para controlar qué nodos tienen acceso al enlace, todos los otros nodos son esclavos y tienen que esperar por el permiso del maestro antes de transmitir. El maestro puede enviar y recibir mensajes hacia y desde cada nodo en el enlace multipunto y hacia y desde cada nodo de la red conectada al enlace multipunto[22].

1.2.4. DNP3

Distributed Network Protocol 3 (DNP3 en lo adelante) es un estándar de telecomunicaciones que define las comunicaciones entre estaciones maestras, unidades remotas y otros Dispositivos Electrónicos Inteligentes (IED siglas en inglés). Fue desarrollado para lograr la interoperabilidad entre los sistemas de empresas eléctricas,

[9]Unidad Terminal Maestra siglas en inglés
[10]Controlador Lógico Programable

petroleras, de agua, entre otras. DNP3 ha sido diseñado específicamente para aplicaciones SCADA. Esto implica la adquisición de información y envío de comandos de control entre dispositivos computarizados separados físicamente. Está diseñado para la transferencia de paquetes pequeños de datos de manera fiable con los mensajes que lleguen a participar de una secuencia determinista. En este sentido se diferencia de otros protocolos de uso general, tales como FTP[11], que es parte de TCP/IP, que puede enviar archivos de gran tamaño, pero de una manera que generalmente no es tan adecuada para el control SCADA. Una característica clave de DNP3 es que es un protocolo de estándar abierto y ha sido adoptado por un número significativo de fabricantes de equipos. La ventaja de un estándar abierto es que permite la interoperabilidad entre equipos de diferentes fabricantes, esto significa por ejemplo que un usuario puede comprar el equipo del sistema, como una estación principal de un fabricante, y ser capaz de agregar equipos RTU que provienen de otro fabricante. DNP3 ofrece características importantes, tales como la flexibilidad y la seguridad. Esto se resume a continuación[3]:

- Los mensajes se separan en varias tramas para proporcionar un control óptimo de errores y secuencias rápidas de comunicación.
- Permite topología igual-igual, así como maestro-esclavo.
- Permite topología de múltiples maestros.
- Solicitudes y respuestas con múltiples tipos de datos en un solo mensaje, y permite objetos definidos por el usuario.
- Permite comunicar excepciones/eventos sin necesidad de encuestas por parte del maestro.
- Permite mensajes en "Broadcast"[12].
- Transferencia segura de configuración/archivos.
- Permite direccionar más de 65 000 dispositivos en un solo enlace.
- Proporciona sincronización de tiempo y eventos con marca de tiempo.
- Permite confirmaciones al nivel de la capa de Enlace y/o capa de Aplicación garantizando así alta integridad en la información.
- Asigna prioridades a un grupo de datos.

[11] File Transport Protocol en inglés
[12] Mensaje que es enviado por el maestro a todos sus esclavos

1.3. Modelo TCP/IP.

El modelo TCP/IP fue desarrollado y presentado por el Departamento de Defensa de EE.UU. En 1972 y fue aplicado en ARPANET (Advanced Research Projects Agency Network en inglés), que era la red de área extensa del Departamento de Defensa como medio de comunicación para los diferentes organismos de EE.UU. La transición hacia TCP/IP en ARPANET se concretó en 1983.

Al conjunto de protocolos de red que son implementados por la pila sobre los cuales se fundamenta Internet y que permiten la transmisión de datos entre las redes de computadoras se le conoce como la familia de protocolos de Internet. Los dos más importantes son TCP(Transmission Control Protocol en inglés) e IP (Internet Protocol en inglés), de ahí que se denomine como conjunto de protocolos TCP/IP. Los tipos de protocolos existentes superan los cien, ente los cuales podemos mencionar como los más conocidos a HTTP, FTP, SMTP, POP y ARP.

TCP/IP es la plataforma que permite la comunicación entre diferentes sistemas operativos en diferentes computadoras, ya sea sobre redes de área local (LAN siglas en inglés) o redes de área extensa (WAN siglas en inglés)[6].

Figure 1.1: Pila TCP/IP

1.3.1. Capa de Aplicación.

La capa de aplicación maneja protocolos de alto nivel, aspectos de representación, codificación y control de diálogo. El modelo TCP/IP combina todos los aspectos relacionados con las aplicaciones en una sola capa y garantiza que estos datos estén

correctamente empaquetados para la siguiente capa. La capa de aplicación incluye programas de aplicación que utilizan la red. Es la encargada de pasar la solicitud a la Capa de Transporte cuando un usuario inicia una transferencia de datos. Como ejemplos de estas aplicaciones se tienen al correo electrónico, los programas de transferencia de archivos o cualquier aplicación que tenga entre sus funcionalidades la de transmitir datos por la red como es el caso de la aplicación que se pretende desarrollar para dar solución a la problemática expuesta.

1.3.2. Capa de Transporte

La principal tarea de la capa de transporte es proporcionar la comunicación entre una aplicación y otra. Este tipo de comunicación se conoce frecuentemente como comunicación punto a punto. La capa de transporte regula el flujo de información y puede también proporcionar un transporte confiable, asegurando que los datos lleguen sin errores y en secuencia. Para hacer esto, el software de protocolo de transporte tiene el lado de recepción enviando acuses de recibo de retorno y la parte de envío retransmitiendo los paquetes perdidos. La capa de transporte divide el flujo de datos que se está enviando en pequeños fragmentos (por lo general conocidos como paquetes) y pasa cada uno, con una dirección de destino, hacia la siguiente capa de transmisión[16].

Debido a que la solución a desarrollar se encuentra enmarcada esencialmente en la capa de aplicación y hará uso del servicio TCP de la capa de transporte del modelo TCP/IP, en este trabajo, se decidió indagar solamente en estos dos niveles.

1.3.3. TCP

El protocolo TCP constituye uno de los más conocidos y empleados del modelo TCP/IP. Para entender el funcionamiento de los protocolos TCP/IP debe tenerse en cuenta la arquitectura que éstos proponen para comunicar redes. Dicha arquitectura se basa en ver como semejantes todas las redes con posibles conexiones, sin tener en cuenta tamaño amplitud o localización. Define que todas las redes que intercambiarán información deben estar conectadas a una misma computadora y denomina a tales ordenadores como compuertas, enrutadores o puentes.

Una entidad de transporte TCP acepta mensajes de longitudes arbitrariamente grandes, procedentes de los procesos de usuario, los separa en pedazos que no excedan de 64 octetos y transmite cada pedazo como si fuera un datagrama separado. La capa de red, no garantiza que los datagramas se entreguen apropiadamente, por lo que TCP deberá utilizar temporizadores y retransmitir los datagramas si es necesario.

Los datagramas que consiguen llegar, pueden hacerlo en desorden; y dependerá de TCP el hecho de reensamblarlos en mensajes, con la secuencia correcta[21].

1.3.4. Protocolo UDP

El Protocolo de Datagramas de Usuario (UDP siglas en inglés) es un estándar TCP/IP. Algunos programas utilizan UDP en lugar de TCP para el transporte de datos rápido, compacto y no confiable entre ordenadores de una red. UDP proporciona un servicio de datagramas sin conexión que ofrece entrega de mejor esfuerzo, lo que significa que UDP no garantiza la entrega ni comprueba la secuencia de los datagramas. Un ordenador de origen que necesita comunicación confiable debe utilizar TCP o un programa que proporcione sus propios servicios de secuencia y confirmación. Normalmente, utilizan UDP los programas que transmiten pequeñas cantidades de datos a la vez o que tienen requisitos de tiempo real. En estas situaciones, las capacidades de carga pequeña y multidifusión de UDP (por ejemplo, un datagrama, muchos destinatarios) resultan más apropiadas que TCP[1].

Los mensajes UDP están encapsulados y se envían en datagramas IP, como se muestra en la Figura 1.2:

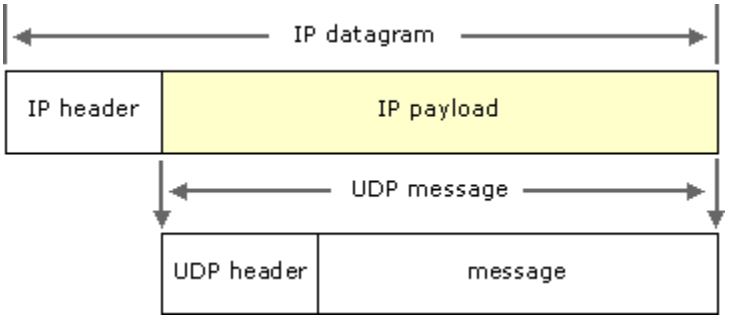


Figure 1.2: Estructura de un mensaje al usar UDP

1.3.5. Comunicación serial

La comunicación serial es un protocolo muy común para comunicación entre dispositivos que se incluye de manera estándar en prácticamente cualquier computadora.

El concepto de comunicación serial es sencillo. El puerto serial envía y recibe bytes de información bit a bit. Aún y cuando esto es más lento que en la comunicación en paralelo, que permite la transmisión de un byte completo por vez, este método de

comunicación es más sencillo y puede alcanzar mayores distancias. La especificación IEEE 488 para la comunicación en paralelo determina que el largo del cable para el equipo no puede ser mayor a 20 metros, con no más de 2 metros entre cualquier par de dispositivos; por el otro lado, utilizando comunicación serial el largo del cable puede llegar a los 1200 metros.

Debido a que la transmisión es asíncrona, es posible enviar datos por una línea mientras se reciben datos por otra. Las características más importantes de la comunicación serial son la velocidad de transmisión, los bits de datos, los bits de parada y la paridad. Para que dos puertos se puedan comunicar, es necesario que las características sean iguales[2].

1.4. Herramientas.

La búsqueda de la eficiencia, la productividad y la competitividad por parte de las empresas ha traído consigo que estas hagan uso de herramientas de apoyo. En el mundo se han creado muchas herramientas de modelamiento así como de desarrollo, realizar la selección de éstas es un problema para muchos desarrolladores de software. En este epígrafe se presentan las herramientas de las que se hace uso en el diseño e implementación de la solución y sus principales características.

1.4.1. Bibliotecas

En ciencias de la computación, una biblioteca (library en inglés) es un conjunto de subprogramas utilizados para desarrollar software. Las bibliotecas contienen código y datos, que proporcionan servicios a programas independientes, es decir, pasan a formar parte de éstos. Esto permite que el código y los datos se compartan y puedan modificarse de forma modular. Algunos programas ejecutables pueden ser a la vez programas independientes y bibliotecas, pero la mayoría de estas no son ejecutables.

En el desarrollo de la solución se hará uso principalmente de la biblioteca QT, con el fin de facilitar el desarrollo de las funcionalidades del manejador. QT que es una biblioteca multiplataforma de código abierto de alta calidad, estable y gratuito. Es usada para la construcción de aplicaciones de software con un interfaz gráfica de usuario (GUI siglas en inglés), y también se utiliza para el desarrollo de programas sin interfaz gráfica, tales como herramientas de línea de comandos y las consolas de los servidores. Además ofrece herramientas que facilitan el manejo de cadenas y el trabajo con hilos.

Se hará uso además la biblioteca **TransportLayer** desarrollada por integrantes de la Línea.

1.4.1.1. Biblioteca TransportLayer

TransportLayer es una biblioteca ligera y robusta que se encuentra enmarcada en la capa de transporte del modelo OSI. Cuenta con una API muy cómoda de utilizar así como con una interfaz de C++ que hace explote de las potencialidades de la Programación Orientada a Objetos (POO). La misma contiene un instalador con la documentación HTML generada con doxygen, ejemplos de uso de cada una de las vías de comunicación, con un cliente y un servidor en cada caso. También puede ser recompilada para otras arquitecturas, específicamente arquitecturas ARM características fundamental que impulsó a su implementación. La biblioteca en su versión actual tiene dependencia del sistema operativo Linux.

Esta biblioteca se implementó haciendo uso de la metodología ágil de desarrollo XP con el objetivo principal de lograr la compatibilidad a la hora de compilar para sistemas embebidos y que la dependencia de grandes framework sea mínima o nula.

1.4.2. IDE

Un Entorno de Desarrollo Integrado (IDE siglas en inglés) es un entorno de programación que ha sido empaquetado como un programa de aplicación, por lo general consiste en un editor de código, un compilador, un depurador, y una interfaz gráfica de usuario del constructor. Un IDE puede ser una aplicación independiente o puede ser incluido como parte de una o más aplicaciones existentes y compatibles. El lenguaje de programación BASIC, por ejemplo, se puede utilizar en aplicaciones de Microsoft Office, lo que hace posible escribir un programa de WordBasic dentro de la aplicación Microsoft Word. Un IDE proporciona un marco de uso fácil para muchos lenguajes de programación modernos[4].

Se decide utilizar QT Creator, el cual es un IDE creado bajo los preceptos del software libre y el código abierto. QT Creator es multi-plataforma, presenta una alta calidad al combinar edición, depuración y herramientas de compilación. Tiene todo lo necesario para crear programas para ordenadores. Cuenta con una gran comunidad en el mundo. Además luego de 18 años, se ha conseguido acumular una basta y abarcadora cantidad de documentación, que va desde tutoriales hasta fragmentos de código.

1.4.3. Lenguaje de programación.

Los lenguajes de programación son lenguajes creados por el ser humano para poder comunicarse con las computadoras. Así un lenguaje de programación es el

conjunto de símbolos y palabras que permiten al usuario de una computadora darle instrucciones y órdenes para que la computadora los ejecute[10].

C++ es un lenguaje versátil, potente, procedural[13] y orientado a objetos. Como lenguaje procedural se asemeja al C y es compatible con él. Como lenguaje orientado a objetos se basa en una filosofía completamente diferente, que exige del programador un completo cambio de mentalidad. Las características propias de la Programación Orientada a Objetos(POO) de C++ son modificaciones mayores que cambian radicalmente su naturaleza[8].

Este lenguaje de alto nivel posee una serie de propiedades, las cuales se muestran a continuación:

- Uniformidad. Ya que la representación de los objetos lleva consigo tanto el análisis como el diseño y la codificación de los mismos.
- Comprensión. Tanto los datos que componen los objetos, como los procedimientos que los manipulan, están agrupados en clases, que se corresponden con las estructuras de información que el programa trata.
- Flexibilidad. Al tener relacionados los procedimientos que manipulan los datos con los datos a tratar, cualquier cambio que se realice sobre ellos quedará reflejado automáticamente en cualquier lugar donde éstos aparezcan.
- Estabilidad. Dado que permite un tratamiento diferenciado de aquellos objetos que permanecen constantes sobre aquellos que cambian con frecuencia, permite aislar las partes del programa que permanecen inalterables en el tiempo.
- Reusabilidad. La noción de objeto permite que programas que traten las mismas estructuras de información, reutilicen las definiciones de objetos empleadas en otros programas e incluso los procedimientos que los manipulan. De esta forma, el desarrollo de un programa puede llegar a ser una simple combinación de objetos ya definidos donde estos están relacionados de una manera particular.

1.4.4. Metodología.

Las metodologías ágiles tienen como objetivo principal que todos los esfuerzos del equipo de trabajo se empleen en el desarrollo de productos que cumplan con los requisitos que el cliente exige. El equipo de trabajo en su totalidad se centra

[13]Orientado a algoritmos.

solo en tareas y procesos que tributen al producto que se encuentra en creación, mejoramiento o implementación. A su vez cada cierto tiempo el cliente recibe versiones del producto a medida que se va desarrollando, esto le permite evaluar el trabajo realizado hasta ese punto, detectar posibles problemas, aportar nuevas ideas a la solución y sugerir nuevas funcionalidades que no se habrían tenido en cuenta hasta el momento y podrían se importantes. Los métodos ágiles son menos orientados al documento, son más bien orientados al código: sugiriendo en todo momento que el código fuente es la parte más importante de la documentación.

1.4.4.1. Extreme Programming (XP)

Se selecciona la metodología ágil XP. Ésta no exige un régimen muy estricto para darle seguimiento, permite ver a la Línea como el cliente y garantiza además a partir de sus principales características, factibilidad en el desarrollo del ciclo de vida del software.

A continuación se brindan algunas características de XP:

- El equipo de trabajo y el cliente se comunican entre ellos con el objetivo de un desarrollo de software que finalice con los resultados deseados.
- El cliente o el usuario se convierten en miembro del equipo de trabajo.
- El software empieza en pequeño y añade funcionalidades con la retroalimentación continua.
- Las pruebas garantizan desde el primer día retroalimentación entre usuarios y clientes.
- Los diseños del software serán sencillos y libres de pretensiones innecesarias.
- Se realizan entregas del software de forma constante y frecuente tan pronto como sea posible.
- Se garantiza una implementación rápida de los cambios que ocurran durante el proceso de desarrollo del software.
- El manejo del cambio en el proceso se convierte en su parte más importante.
- La fase o etapa en la que se encuentre el desarrollo del software no determina el costo del cambio.

Los programadores deben entender que en el ciclo de vida del desarrollo de un producto de software siempre aparecen cambios inevitables, pueden cambiar requisitos, personal, tecnología, etc. Por tanto lo mejor es prepararse para enfrentar estos cambios. A continuación se describen las faces del ciclo de vida del software planteadas por XP.

Faces del ciclo de vida del software planteadas por XP:

Exploración: En la fase de Exploración los clientes escriben las historias de usuario (funcionalidades con que debe contar el sistema). Cada plantilla describe las características que deben ser adicionadas al programa. Al mismo tiempo el equipo del proyecto se familiariza con las herramientas, la tecnología y las prácticas que utilizarán en el proyecto. La tecnología ha ser usada será probada y las posibles arquitectura para el sistema son exploradas construyendo un prototipo del sistema.

Planeación: La fase de planeación configura la prioridad para las historias de usuario, contenidas en las tarjetas CRC (Clase-Responsabilidad-Colaboración, una técnica que reemplaza a los diagramas para la representación de modelos, en las que se escriben las responsabilidades) y se realiza un contrato del contenido para la primera entrega. Los programadores primero estiman cuánto esfuerzo requieren para cada historia y se hace una programación de acuerdo a esta estimación. El tiempo de la programación de la primera entrega normalmente no excede dos meses y el tiempo de la fase como tal toma un par de días.

Desarrollo: La fase de desarrollo hacia la entrega incluye varias iteraciones del sistema antes de la primera entrega. La programación que se determinó en la etapa de planeación es dividida en un número de iteraciones donde cada una tomará de una a cuatro semanas para ser implementada. La primera iteración crea la arquitectura de todo el sistema; esto es logrado seleccionando las historias que hacen cumplir la estructura para todo el sistema. Las pruebas funcionales creadas por los clientes son para correr al final de cada iteración. Al final de la última iteración, el sistema estará listo para ser entregado y llevarlo a producción.

Pruebas: La fase de pruebas requiere chequeos extras de ejecución del sistema antes de que sea entregado al cliente. En ésta fase, se pueden encontrar nuevos cambios y se toma la decisión si serán incluidos en la entrega actual. Durante esta fase, las iteraciones pueden necesitar ser recortadas de tres semanas a una semana. Después que la primera entrega es producida para el uso del cliente, el proyecto XP debe mantener el sistema en producción corriendo mientras que también se estén produciendo nuevas iteraciones.

Mantenimiento: La fase de mantenimiento requiere también un esfuerzo para soportar las tareas de los clientes. Así, la velocidad del desarrollo puede disminuir después de que el sistema está en producción. La fase de mantenimiento puede

requerir incorporar nuevas personas al equipo y cambiar la estructura del equipo. Dentro de esta fase se llega a un estado llamado "de muerte", que sucede cuando el cliente no tiene más historias para ser implementadas. Esto requiere que el sistema satisfaga también las necesidades en otros aspectos, como por ejemplo lo concerniente a la ejecución y la confiabilidad. Éste es el momento en el proceso XP cuando la documentación necesaria del sistema es finalmente escrita porque no habrá más cambios en la arquitectura, diseño o código.

XP está descrito por medio de 4 valores: Comunicación, Sencillez, Retroalimentación y Valentía.

Los cuatro valores de XP:

1. **Comunicación.** Cuantas veces aparecen problema en nuestro equipo de desarrollo por falta de comunicación, por no comentar un cambio crítico en el diseño, por no preguntar al cliente. XP ayuda mediante sus prácticas a fomentar la comunicación.

2. **Sencillez.** Siempre hay que preguntarse ¿Qué es lo más simple que puede funcionar?. Existe cierta tendencia a pensar en mañana, la próxima semana y el próximo mes. XP enseña a apostar, apuesta por hacer una cosa sencilla hoy y dejar un poco más para mañana, si es necesario, que hacer una cosa complicada hoy y no utilizarla después. La sencillez y la comunicación se complementan, cuanto más simple sea el sistema menos hay que comunicar sobre el.

3. **Retroalimentación.** "No me preguntes a mi, pregúntale al sistema", es la primera clave de la retroalimentación, por medio de pruebas funcionales al software éste logra mantener informado sobre su grado de fiabilidad al grupo de trabajo. Los clientes y las personas que escriben pruebas tienen una retroalimentación real de su sistema. La retroalimentación actúa junto con la sencillez y la comunicación, cuanta mayor retroalimentación más fácil es la comunicación. Escribir pruebas garantiza que se aprenda como simplificar un sistema.

4. **Valentía.** Asumir retos, ser valientes antes los problemas y afrontarlos. XP propone que si es necesario borrar un código que tomó mucho tiempo pero que no todos pueden entender por lo engorroso del mismo, se debe comprender que es lo mejor para el desarrollo del software. Cuando no se afronta el problema y se parchea un código que positivamente se sabe que esta mal se termina odiando el sistema [19].

1.4.4.2. Prácticas XP

La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione [19]. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación estricta de las 12 prácticas que define XP, Figura 1.4:

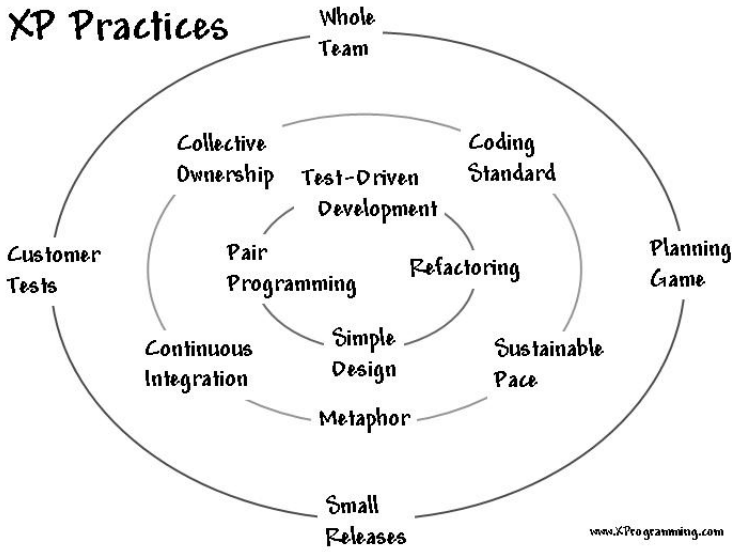


Figure 1.3: Prácticas de XP

- El juego de la planificación (Planning Game): Hay una comunicación frecuente el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.

- Entregas pequeñas (Small Releases) : Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión ya constituye un resultado de valor para el negocio. Una entrega no debería tardar más 3 meses.

- Programación en parejas (Pair Programming): La XP incluye, como una de sus prácticas estándar, la programación en parejas. Nadie programa en solitario, siempre hay dos personas delante del ordenador. Ésta es una de

las características que más se cuestiona al comienzo de la adopción de la XP dentro de un equipo, pero en la práctica se acepta rápidamente y de forma entusiasta. El hecho de que todas las decisiones las tomen al menos dos personas proporciona un mecanismo de seguridad enormemente valioso.

- Metáfora (Metaphor): El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema (conjunto de nombres que o actúen como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema).

- Diseño simple (Simple Desing): Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.

- Pruebas (Customer Test): La producción de código está dirigida por las pruebas unitarias. Éstas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.

- Refactorización (Refactoring): Es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.

- Propiedad colectiva del código (Collective Ownership): Cualquier programador puede cambiar cualquier parte del código en cualquier momento.

- Integración continua (Continuous Integration): Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.

- 40 horas por semana (Sustainable Pace): Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. El trabajo extra desmotiva al equipo.

- Cliente in-situ (Whole Team): El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Éste es uno de los principales factores de éxito del proyecto XP. El cliente conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver

de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita.

- Estándares de programación (Code Standard): XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.[18]

XP como metodología de desarrollo será factible ya que la solución a desarrollar es relativamente pequeña en cuanto a funcionalidades y personal en el equipo de trabajo. Además permite que el principal esfuerzo realizado por el grupo de trabajo se centre en garantizar el desarrollo de las funcionalidades exigidas por el cliente. Permite que la documentación fundamental que respalde el proceso de desarrollo del software sea el código fuente de la aplicación, así como las tarjetas de clase, responsabilidad y colaboración y los diagramas que sea necesario generar en el transcurso del ciclo de vida del software.

1.4.5. UML

Se necesitaba un lenguaje para comunicar las ideas a otros desarrolladores y también para servir de apoyo en los procesos de análisis de un problema. Con este objetivo se creo el Lenguaje Unificado de Modelado (UML en lo adelante). UML es un estándar para representar y modelar la información con la que se trabaja en las fases de análisis y, especialmente, de diseño. Un modelo es una simplificación de la realidad. El objetivo del modelado de un sistema es capturar las partes esenciales del sistema. Para facilitar este modelado, se realiza una abstracción y se plasma en una notación gráfica. Esto se conoce como modelado visual[7].

UML está compuesto por una notación muy específica y por reglas semánticas relacionadas que permiten la construcción de sistemas de software. Por sí mismo no prescribe ni aconseja cómo usar esta notación en el proceso de desarrollo o como parte de una metodología de diseño orientada a objetos, en su lugar describe cómo modelar la relación entre esos elementos ya sean clases, componentes, nodos, actividades, flujos de trabajo, casos de uso, objetos o estados. También soporta la idea de extensiones personalizadas a través elementos estereotipados y provee beneficios significativos para los ingenieros de software al ayudarles a construir modelos rigurosos que soporten el ciclo de vida de desarrollo de software completo[7].

1.4.6. Herramientas de Modelado

Las Herramientas CASE (Computer Aided Software Engineering en inglés) proporcionan un conjunto de herramientas semi-automatizadas y automatizadas que brindan ayuda y dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software[9].

Beneficios de las herramientas CASE:

- Potencia la mejora del producto final.
- Facilita el desarrollo de los procesos.
- Mejora la calidad del sistema.
- Disminuye tiempo de trabajo.
- Garantiza la consistencia de los procedimientos.
- Captura de los datos del sistema[9].

Las Herramientas CASE fueron diseñada para:

- Soportar un entorno personal dedicado.
- Utilizar Gráficos para especificar y documentar los sistemas.
- Unir todas las fases del ciclo del software.
- Utilizar la inteligencia artificial para realizar automáticamente muchas de las rutinas, tareas de desarrollo y mantenimiento del software[9].

Como herramienta CASE se decide el uso de Visual Paradigm que es una herramienta de modelado para todo tipo de diagramas UML basada en los preceptos del software libre. Se puede encontrar disponible en múltiples plataformas. Visual Paradigm cuenta con un diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad. Facilita la conversión modelo a código, diagrama a código. Además permite que el modelo y el código permanezcan sincronizados en todo el ciclo de desarrollo.

Conclusiones

Al finalizar este capítulo se concluye que realizar un estudio de los sistemas de supervisión y control existentes en Cuba y en el resto del mundo, haciendo énfasis en los manejadores de dispositivos así como de los protocolos del modelo TCP/IP, permitió determinar que un medio factible para la comunicación entre el manejador y los dispositivos del sistema G-Hawk sería TCP.

Se realizó la selección de las herramientas a utilizar para el desarrollo del manejador; C++ como lenguaje de desarrollo, XP como metodología de desarrollo, UML como lenguaje de modelado, QT Creator como IDE de desarrollo y Visual Paradigm como herramienta CASE.

2 Diseño de la solución

Introducción

En este capítulo se exponen las características del sistema, así como la descripción y modelación de los procesos, detallando las historias de usuarios que brindan una mejor visión sobre los requerimientos del cliente, además se hace alusión a las fases de planificación y diseño propias de la metodología de desarrollo utilizada. De igual forma se exponen los artefactos generados durante el transcurso de las mismas.

2.1. Protocolo de comunicación entre dispositivos basados en microcontroladores sobre arquitectura ARM (ProC-ARM).

Debido a la necesidad existente de garantizar el intercambio de información fiable entre las tarjetas CID-300/9 y STM32, se decide a partir del estudio realizado a los protocolos de comunicación en el capítulo **Fundamentación Teórica**, diseñar un protocolo de comunicación (ProC-ARM) que sirva como rector de dicho proceso.

ProC-ARM en su primera versión contará con las siguientes características:

- **Comunicación:** ProC-ARM podrá ser implementado para la comunicación a partir de una conexión TCP, UDP o Serie teniendo en cuenta las características de cada una.
- **Ubicación:** ProC-ARM será un protocolo perteneciente a la capa de aplicación del modelo TCP/IP y será el encargado de organizar el flujo de información entre el manejador y los dispositivos con los que se comunique.
- **Estructura:** ProC-ARM será un protocolo binario, los datos que conformarán las tramas de lectura y escritura tendrán valores binarios, respetando en todo momento la estructura de las mismas.

- **Arquitectura:** ProC-ARM será diseñado sobre una arquitectura Cliente-Servidor. El manejador ubicado en la tarjeta CID-300/9 se comportará en todo momento como cliente con respecto a la aplicación servidor que se encontrará en la tarjeta STM32.
- **Datos:** ProC-ARM permitirá la lectura y escritura de datos en los servidores. El tipo de dato que se utilizará para el intercambio de información se denominará **Registro** y tendrá un tamaño de 2 bytes.

Estructura y especificaciones de los comandos ProC-ARM:

- **Comando openSession:** Comando que tiene como fin el inicio de una nueva sesión en alguno de los servidores. Cada nueva sesión será creada con un identificador para poder diferenciarlas unas de otras. La respuesta al comando **openSession** contendrá el id de la sesión creada y OK si la sesión se creó con éxito o un error en caso de que no se pudiera crear.

comandType

Table 2.1: TRAMA-REQUEST (**openSession**)

comandType	idSession	error

Table 2.2: TRAMA-RESPONSE (**openSession**)

- **comandType:** En un byte este campo corresponde al tipo de **comando** que se lanza (openSession, closeSession, readRegister, writeRegister, deviceInfo).
- **idSession:** En dos bytes este campo corresponde al **ID** de la nueva sesión que se creó.
- **error:** En un byte este campo contendrá **OK** si se creó una sesión con éxito o un **error** (CONNECTION FAILURE, ILEGAL COMAND TYPE) en caso contrario.

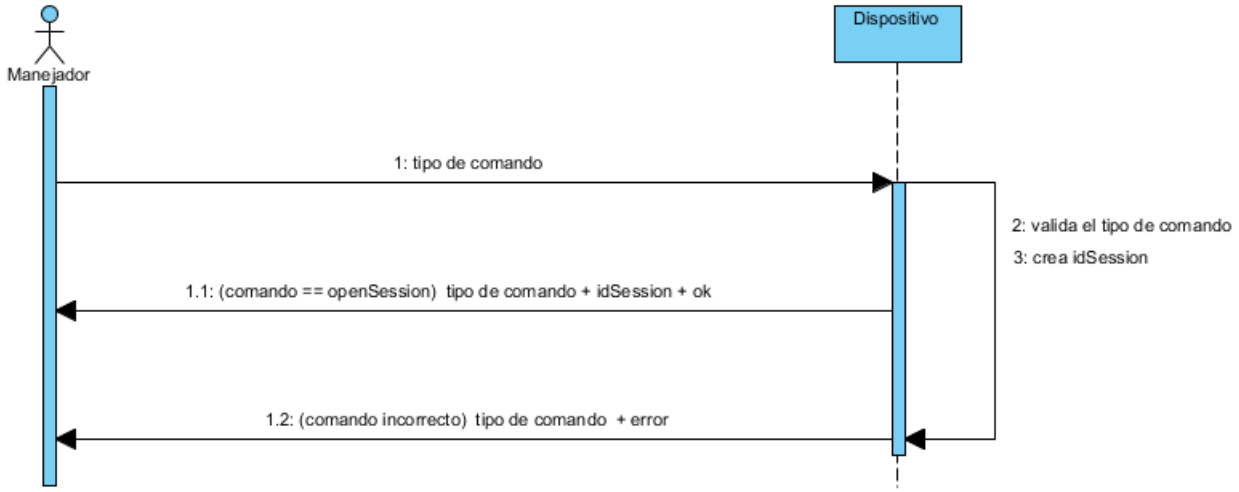


Figure 2.1: Secuencia - openSession

- **Comando closeSession:** Comando que tiene como fin el término de una sesión existente en alguno de los servidores. Cada sesión deberá ser finalizada a partir del identificador con el que fue creada. La respuesta al comando **closeSession** retornará **OK** si la sesión fue cerrada con éxito o un **error** en caso de que no se pudiera finalizar.

comandType	idSession

Table 2.3: TRAMA-REQUEST (**closeSession**)

comandType	error

Table 2.4: TRAMA-RESPONSE (**closeSession**)

- **comandType:** En un byte este campo corresponde al tipo de **comando** que se lanza (openSession, closeSession, readRegister, writeRegister, deviceInfo).
- **idSession:** En dos bytes este campo responde al **ID** de la sesión que será cerrada .
- **error:** En un byte este campo contendrá **OK** si se cerró la sesión con éxito o un **error** (CONNECTION FAILURE, ILEGAL COMAND TYPE, ILEGAL SESSION ID) en caso contrario.

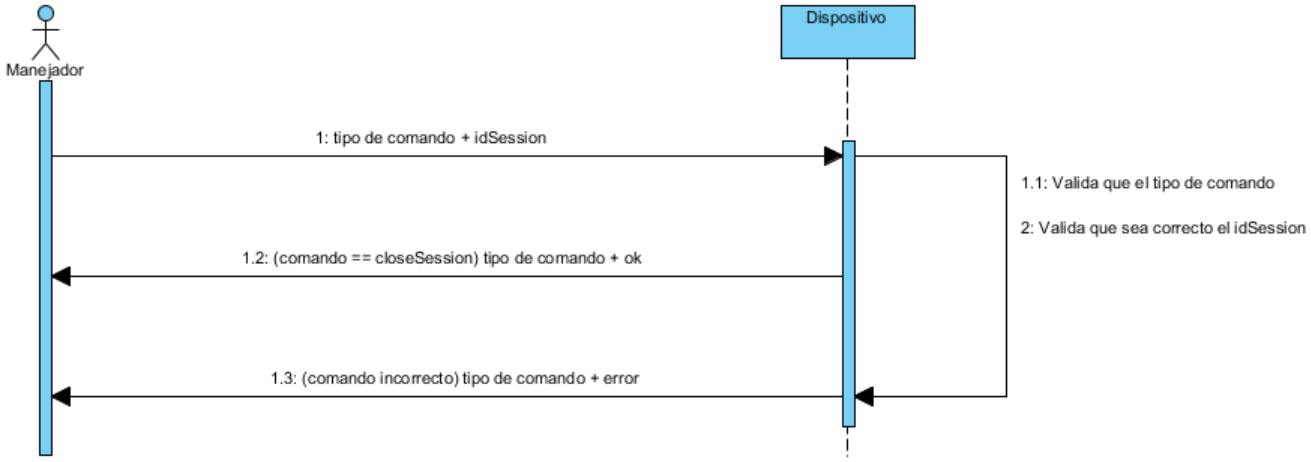


Figure 2.2: Secuencia - closeSession

- **Comando readRegister:** Comando que tiene como fin la lectura de registros existentes en direcciones especificas de los servidores. La lectura de registros se realizará a partir de la direcciones de los mismos siempre y cuando se cuente con una sesión abierta en dicho servidor. La respuesta al comando **readRegister** contendrá la cantidad de registros leídos y los registros, si se logró la lectura, de lo contrario retornará un **error**.

comandType	idSession	reference	registerCount

Table 2.5: TRAMA-REQUEST (**readRegister**)

comandType	idSession	reference	registerCount	Registers

Table 2.6: TRAMA-RESPONSE (**readRegister**)

- **comandType:** En un byte este campo corresponde al tipo de **comando** que se lanza (openSession, closeSession, readRegister, writeRegister, deviceInfo).
- **idSession:** En dos byte este campo corresponde al **ID** de la sesión en que se trabaja.
- **reference:** Este campo corresponde a la **dirección** en el servidor de donde se realizará la lectura de los datos.

- **registerCount:** Este campo corresponde a la **cantidad** de registros que se desea leer del servidor.
- **Registers:** Este campo corresponde a los **datos** leídos del servidor.
- **error:** En un byte este campo contendrá **OK** si la lectura de los datos se realizó con éxito o un **error** (CONNECTION FAILURE, ILEGAL COMAND TYPE, ILEGAL SESSION ID, ILLEGAL READ REFERENCE) en caso contrario.

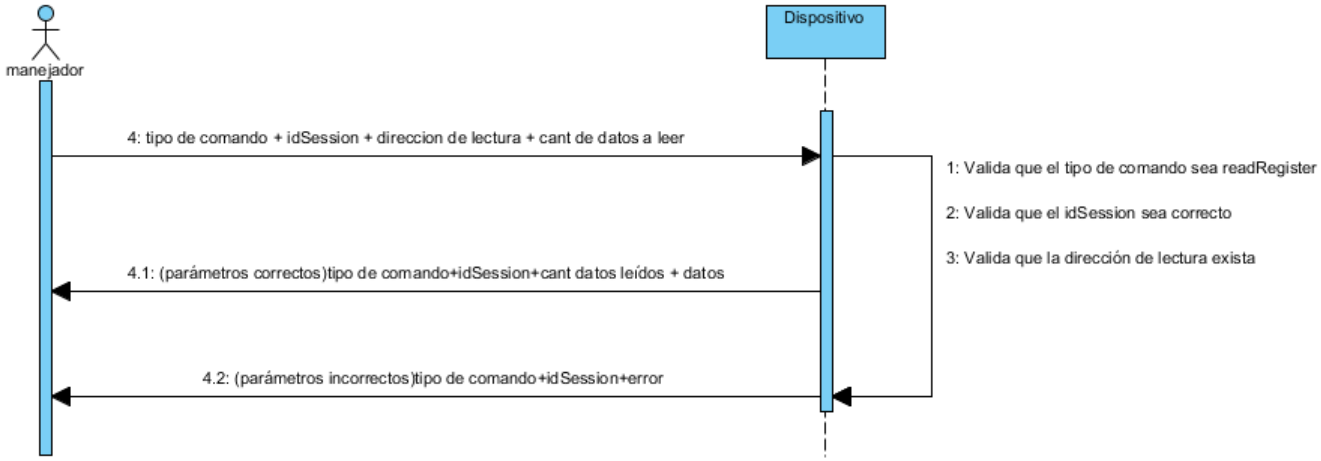


Figure 2.3: Secuencia - readRegister

- **Comando writeRegister:** Comando que tiene como fin la escritura de registros en alguno de los servidores a partir de una sesión ya creada. La escritura de registros se realizará a partir del identificador de la sesión creada en el mismo así como de la dirección donde se desea escribir el registro, se envirará la cantidad de datos a escribir y los datos. La respuesta al comando **writeRegister** contendrá OK si los datos fueron escritos con éxito de lo contrario retornará un **error**.

comandType	idSession	reference	registerCount	Registers

Table 2.7: TRAMA-REQUEST (**writeRegister**)

comandType	idSession	error

Table 2.8: TRAMA-RESPONSE (**writeRegister**)

- **comandType:** En un byte este campo corresponde al tipo de **comando** que se lanza (openSession, closeSession, readRegister, writeRegister, deviceInfo).
- **idSession:** En dos byte este campo corresponde al **ID** de la sesión en que se trabaja.
- **reference:** Este campo corresponde a la **dirección** en el servidor donde se escribirán los datos.
- **registerCount:** Este campo corresponde a la **cantidad** de datos que se desea escribir en el servidor.
- **Registers:** Este campo corresponde a los **datos** a escribir en el servidor.
- **error:** Este campo contendrá **OK** si la escritura de los datos se realizó con éxito, de lo contrario un **error** (CONNECTION FAILURE, ILEGAL COMAND TYPE, ILEGAL SESSION ID, ILLEGAL WRITE REFERENCE)**.**

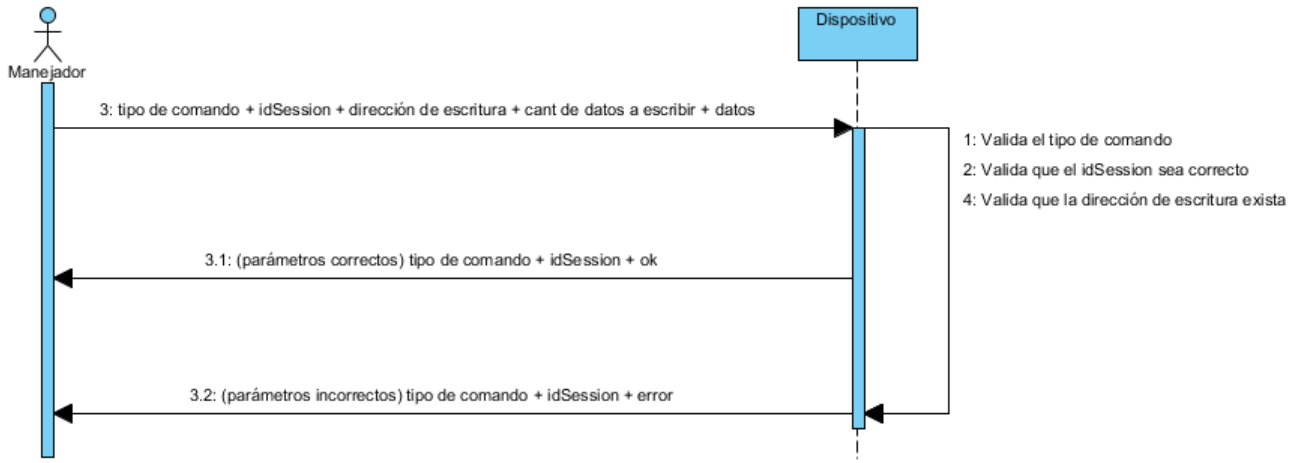


Figure 2.4: Secuencia - writeRegister

- **Comando deviceInfo:** Comando que tiene como fin obtener información del dispositivo que actúa como servidor. La acción de obtener información del servidor se realizará a partir del identificador de la sesión abierta. La respuesta al comando **deviceInfo** contendrá la cantidad de datos leídos que conforman la información del dispositivo, y los datos si el proceso se desarrolló con éxito, de lo contrario retornará un **error**.

comandType	idSession

Table 2.9: TRAMA-REQUEST (**deviceInfo**)

comandType	idSession	datalength	data

Table 2.10: TRAMA-RESPONSE (**deviceInfo**)

- **comandType:** En un byte este campo corresponde al tipo de **comando** que se lanza(openSession, closeSession, readRegister, writeRegister, deviceInfo).
- **idSession:** En dos byte este campo responde al **ID** de la sesión en que se trabaja.
- **datalength:** En dos byte este campo responde a la **cantidad** de información que se obtuvo del servidor.
- **data:** Este campo responde a la **información** obtenida del servidor.
- **error:** Este campo contendrá **OK** si la lectura de los datos se realizó con éxito de lo contrario un **error** (CONNECTION FAILURE, ILEGAL COMAND TYPE, ILEGAL SESSION ID)**.**

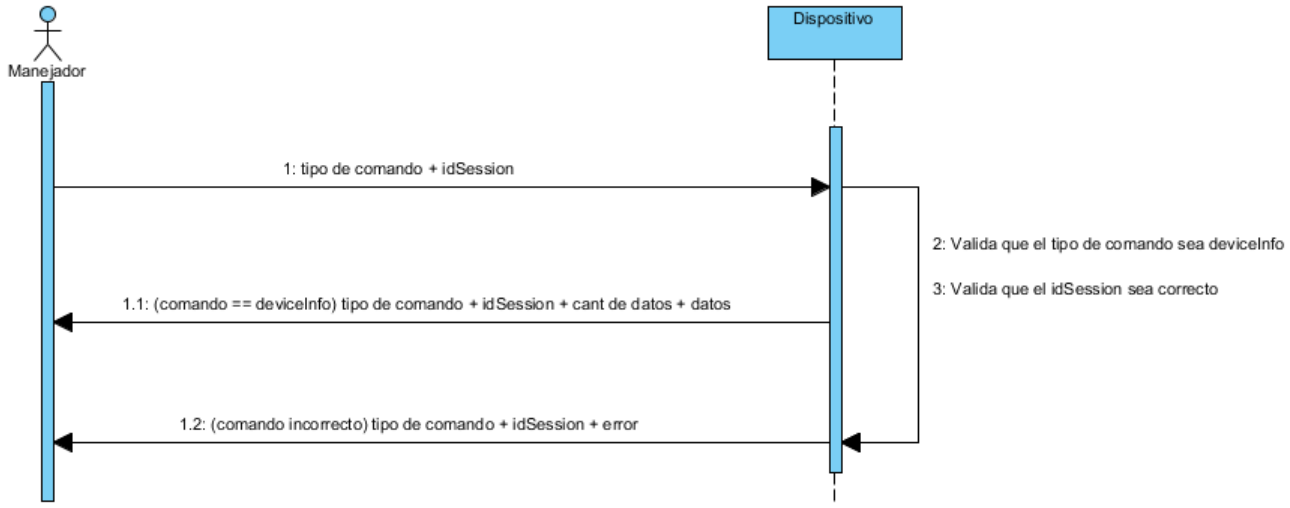


Figure 2.5: Secuencia - deviceInfo

Especificación de errores de ProC-ARM:

Cuando se produce un error en la ejecución de un comando en el dispositivo conectado, este responde al cliente ubicando en el último byte de la trama de respuesta el código del error ocurrido. Este código permitirá al manejador conocer que error se ha producido en el servidor, garantizando que se advierta al usuario y se trabaje en la corrección del mismo. A continuación se brinda la relación de errores del protocolo ProC-ARM.

Código	Nombre	Significado
1	ILEGAL SESSION ID	El ID de sesión indicado en la trama no se corresponde con el ID de sesión válido del esclavo.
2	ILEGAL COMAND TYPE	El tipo de comando recibido no se corresponde a ninguno posible.
3	ILLEGAL READ REFERENCE	La dirección de lectura especificada no existe.
4	ILLEGAL WRITE REFERENCE	La dirección de escritura especificada no existe.
5	SLAVE DEVICE FAILURE	El esclavo ha recibido la trama y la ha comenzado a procesar, pero se ha producido algún error y no ha podido termina la tarea.
6	CONNECTION FAILURE	Ha ocurrido un fallo en la conexión con el servidor

Table 2.11: Tabla de errores del protocolo ProC-ARM

2.2. Características del sistema

A las características o cualidades que todo sistema debe poseer se le denomina requerimiento no funcional. Los requerimientos no funcionales son aquellos que no se refieren directamente a las funciones específicas que proporciona el sistema, para la realización del sistema propuesto estas son las características no funcionales.

2.2.1. Características de seguridad.

- Confiabilidad: La información manejada debe estar protegida en todo momento en caso de que ocurran errores en el sistema.

- Integridad: La información manejada por el sistema durante todo el proceso debe ser objeto de una cuidadosa protección.
- Disponibilidad: Los mecanismos utilizados para lograr la seguridad en la transferencia de datos no deben ser un obstáculo para obtener los servicios deseados en cada momento.

2.2.2. Características de hardware.

La tarjeta CID-300/9 cuenta con las siguientes especificaciones de hardware:

1. Frecuencia CPU Máx: 400 MHz.
2. Puertos: 4 USB, 1 Ethernet, 2 Serie.
3. RAM: 128MB.
4. Audio: 2 entrada/salida de 3.5mm.
5. Tarjeta SD: 1 puerto.
6. Video: Pantalla táctil y puerto LVDS.

2.2.3. Características de software.

La tarjeta CID-300/9 contiene embebido un sistema operativo basado en GNU/Linux con las siguientes características:

1. Tamaño del sistema 48 MB.
2. Tamaño en SD 90 MB.
3. RAM consumida en consola 9.7 MB.
4. RAM consumida con QT 25 MB.
5. RAM consumida con XServer 16.7 MB.
6. Tiempo de arranque 5.6 seg.
7. Tamaño de la caché 261 MB.

2.3. Patrón arquitectónico.

Los patrones arquitectónicos son los que definen la estructura de un sistema, los cuales a su vez se componen de subsistemas con sus responsabilidades. También tienen una serie de directivas para organizar los componentes del sistema, con el objetivo de facilitar la tarea del diseño del mismo.

Se ha decidido el uso de una arquitectura en capas para el desarrollo del manejador. Este patrón define cómo organizar el modelo de diseño en capas, que pueden estar físicamente distribuidas, lo cual quiere decir que los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores [22].

Principales estilos de este patrón arquitectónico:

- Arquitecturas de dos capas.
- Arquitecturas de tres capas.
- Arquitecturas de n capas.

Para el diseño del manejador se utiliza específicamente el estilo arquitectónico tres capas, ya que el modelo de diseño está organizado en las capas de Manejador, Protocolo y Transporte, como se muestra a continuación:

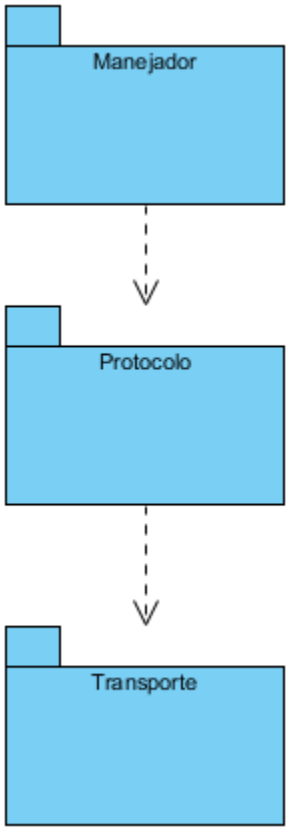


Figure 2.6: Arquitectura 3 capas del manejador.

Capa Manejador:

En la capa Manejador, se definen e implementan clases para obtener un comportamiento acorde a las características del protocolo sobre el cual se está implementando el manejador. La clase Collector que se encuentra en esta capa es la encargada de garantizar las características e interfaces de acceso a las principales funcionalidades del sistema propuesto.

Capa Protocolo:

La capa Protocolo tiene como objetivo garantizar la lógica de comunicación entre el manejador y el dispositivo que se supervisa, definiéndose para esto una clase Device. En esta clase se controlan todos los pasos que define el protocolo para lograr una comunicación satisfactoria, segura y fiable. Para llevar a cabo esto es necesario el uso de un transporte, que permita el envío y recepción de las tramas a través del medio físico que utilice el protocolo.

Capa Transporte:

En la capa de transporte es utilizada la biblioteca dinámica TransportLayer. Esta biblioteca permite la creación de transportes TCP, UDP y Serie, para cada unos de los transportes existen dos clases: una clase interfaz que hereda de ITransport, adquiriendo sus características y una clase en la que se implementan las funcionalidades de su interfaz, correspondiente con las características específicas del transporte a que pertenece. En el caso de TCP existen las clases ITransportTCP y TransportTCP, para UDP son la ITransportUDP y TransportUDP y para el caso de Serie son la ITransportSPort y TransportSPort como se muestra en la Figura 2.7:

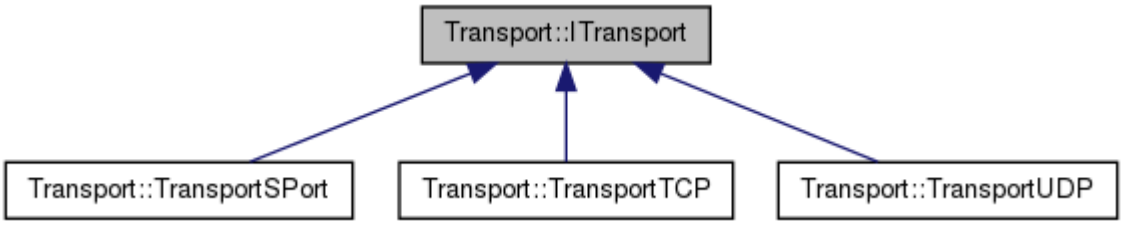


Figure 2.7: Diagrama de clases de la biblioteca TransportLayer

2.4. Diagrama de clases del diseño.

Las clases que son utilizadas dentro de un sistema y las relaciones que existen entre ellas son representadas en el diagrama de clases. El diagrama de clases sirve para visualizar las relaciones entre las clases de un sistema, las cuales pueden ser asociativas, de herencia, de uso y de convencimiento.

En la Figura 2.8 se muestra el diagrama de clases del sistema donde se encuentra representadas las relaciones entre las mismas así como sus atributos y métodos.

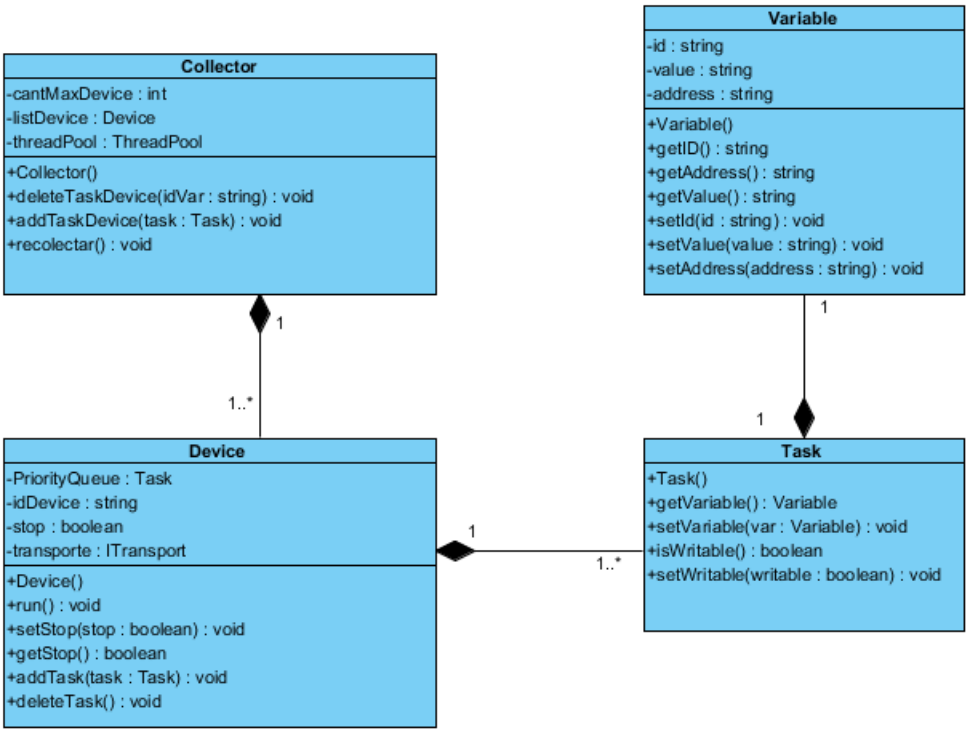


Figure 2.8: Diagrama de clases del sistema

2.5. Fase de Planificación.

La fase de Planificación es la fase inicial en cualquier proyecto XP, tiene entre sus principales objetivos dejar recogidas las historias de usuario, estas historias son expuestas por los clientes a grandes rasgos. Se realiza un cronograma de entregas donde se establece qué historias de usuario serán agrupadas para conformar una entrega, y el orden de las mismas. La duración de esta fase es proporcional con el nivel de familiaridad que tengan los programadores con las tecnologías a utilizar. En la metodología XP, la creación del sistema normalmente se divide en tres etapas, cada una de las cuales toma el nombre de **Iteración**[18].

2.5.1. Procesos de sistema.

Como se muestra en la Figura 2.9 los procesos en el sistema inician con la configuración de los parámetros de conexión para la comunicación con el dispositivo a encuestar. A continuación se procede a crear y adicionar variables al manejador. Las

variables contendrán la información a intercambiar. Luego comienza el proceso de transferencia de información entre el manejador y el dispositivo, es válido aclarar que el proceso de crear y adicionar variables continúa sin importar que se haya iniciado el intercambio de información. El sistema debe permitir la adición y eliminación de variables en caliente. A continuación se intenta establecer la comunicación con el dispositivo a partir de los parámetros configurados previamente, al lograr establecer comunicación, se procede a atender las tareas existentes en la cola del mismo. Si la comunicación no se completa entonces se mostrará el error ocurrido y se detendrá el proceso con el dispositivo.

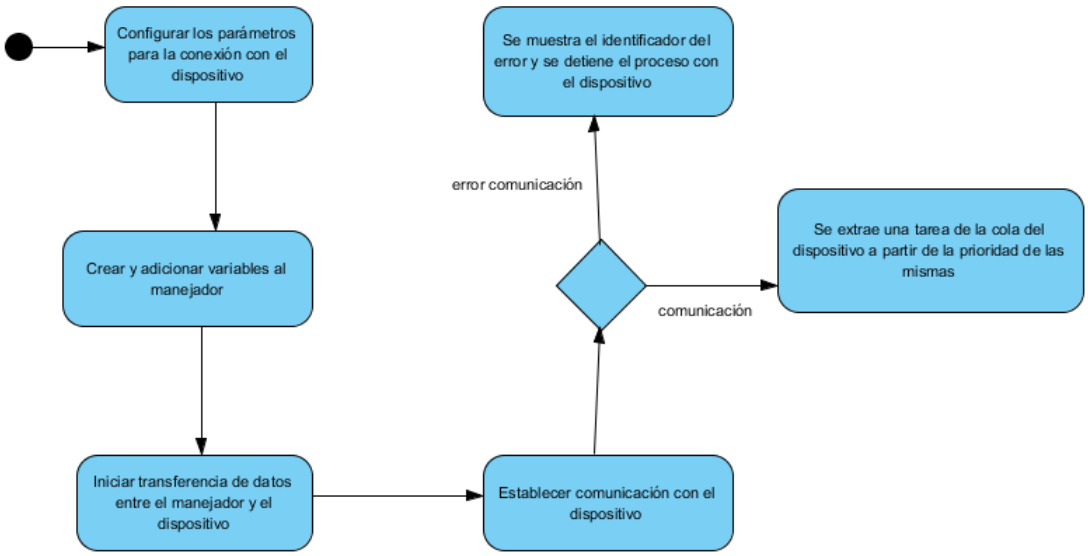


Figure 2.9: Procesos del sistema I

Como se muestra en la Figura 2.10 el proceso de atender las tareas que se encuentran en la cola del dispositivo comienza con la extracción de cada una a partir de la prioridad de las mismas. Luego de extraída la tarea se procede a realizar sobre el dispositivo la acción que la misma especifica (leer, escribir). De corresponder a una tarea de escritura se escriben los datos en el dispositivo, a partir del parámetro de dirección especificado en la variable. De lo contrario si corresponde con una tarea de lectura entonces se procede a leer del dispositivo la cantidad de datos especificados en la configuración inicial de la variable que se encuentra relacionada con dicha tarea, al finalizar el proceso de lectura la tarea se vuelve a ubicar en la cola del manejador.

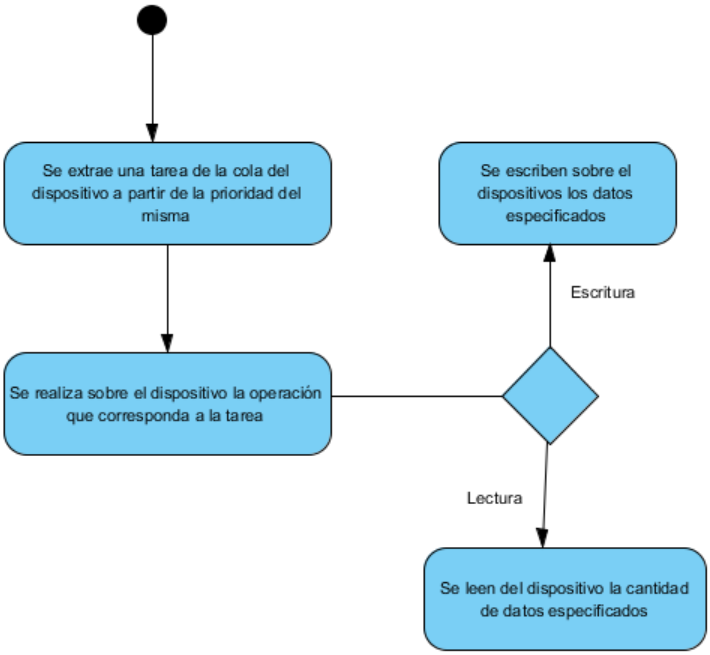


Figure 2.10: Procesos del sistema II

2.5.2. Historias de usuario (HU).

Las historias de usuario tienen el mismo propósito que los casos de uso, son escritas por los propios clientes, tal y como ven ellos las necesidades del sistema. Las historias de usuario son similares al empleo de escenarios, con la excepción de que no se limitan a la descripción de la interfaz de usuario, conducen el proceso de creación de los test de aceptación (empleados para verificar que las historias de usuario han sido implementadas correctamente). Existen diferencias entre éstas y la tradicional especificación de requisitos, la principal diferencia es el nivel de detalle. Las historias de usuario solamente proporcionaran los detalles sobre la estimación del riesgo y cuánto tiempo conllevará la implementación de dicha historia de usuario[11].

Historia de Usuario	
Número: 1	**Usuario:** Línea Sistemas Empotrados
Nombre historia: Configurar dispositivo.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 4	**Iteración asignada:** 1
Programador responsable: Héctor E. León García.	
Descripción: Se configurará el dispositivo en lo referente al medio por el cual se establecerá la comunicación (TCP) así como la dirección IP y el puerto de conexión.	
Observaciones: Es muy importante dejar bien configurados los parámetros de conexión a partir de los que se establecerá la comunicación con el dispositivo.	

Table 2.12: Configurar del dispositivo.

Historia de Usuario	
Número: 2	**Usuario:** Línea Sistemas Empotrados
Nombre historia: Gestionar variables.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 4	**Iteración asignada:** 1
Programador responsable: Héctor E. León García.	
Descripción: Adición y eliminación de variables que contendrán la información para intercambiar con el dispositivo.	
Observaciones: La adición y eliminación de variables del manejador podrá realizarse antes y durante el proceso de transferencia de información.	

Table 2.13: Gestionar variables.

Historia de Usuario	
Número: 3	**Usuario:** Línea Sistemas Empotrados
Nombre historia: Diseñar protocolo de comunicación.	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Media
Puntos estimados: 2	**Iteración asignada:** 2
Programador responsable: Héctor E. León García.	
Descripción: El protocolo de comunicación va a proveer a los usuarios las características para lograr transferir información de forma fiable entre los dispositivos del sistema.	
Observaciones: Este protocolo es uno de los eslabones esenciales del manejador de dispositivos que se desea implementar.	

Table 2.14: Diseñar protocolo de comunicación.

Historia de Usuario	
Número: 4	**Usuario:** Línea Sistemas Empotrados
Nombre historia: Comunicar dispositivos.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 3	**Iteración asignada:** 2
Programador responsable: Héctor E. León García.	
Descripción: El proceso de comunicación entre el manejador y los dispositivos se basa en el los parámetros especificados.	
Observaciones: Durante el proceso de conexión con los dispositivos en todo momento se realiza un control para en caso de ocurrir un error poder a partir del identificador del mismo informar al usuario y proceder a darle solución, garantizando fiabilidad en el proceso.	

Table 2.15: Comunicar dispositivos.

Historia de Usuario	
Número: 5	**Usuario:** Línea Sistemas Empotrados
Nombre historia: Gestionar transferencia de datos.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 4	**Iteración asignada:** 3
Programador responsable: Héctor E. León García.	
Descripción: La gestión de la transferencia de datos se garantiza atendiendo las tareas del manejador. Se deben atender las tareas dependiendo de la prioridad de las mismas.	
Observaciones: La operación de atender tareas estará basada en una cola con prioridad.	

Table 2.16: Atención a tareas.

2.5.3. Plan de Iteraciones

Todo el trabajo de la iteración es expresado en tareas de programación, las cuales son asignadas a un programador como responsable. Normalmente el plan de iteraciones consta de tres iteraciones.

- **Primera iteración:** Esta iteración tiene como objetivo la implementación de la primera y la segunda historia de usuario. Al final de ésta se contará con una primera versión básica que recogerá las funcionalidades antes expuestas. El cliente interactuará con ésta versión, con el objetivo de obtener una retroalimentación a partir de sus opiniones, para el grupo de trabajo.

- **Segunda iteración:** El objetivo de esta iteración se basa en la implementación de las historias de usuario tres y cuatro. Al finalizar se contará con una versión de prueba con las funcionalidades antes expuestas. Esta será mostrada al cliente con el objetivo de realizar cambios en base a las necesidades expresadas por mismo.

- **Tercera iteración:** Durante el transcurso de esta iteración se implementará la historia numero 5. Al finalizar la misma se contará con una versión funcional básica del producto final. Como resultado de ésta, el sistema será puesto en funcionamiento, para que a partir de las opiniones del cliente y del grupo de desarrolladores, se proceda a trabajar en mejoras respecto a su diseño y funcionalidades.

2.5.4. Plan de duración de las iteraciones.

El plan de duración de iteraciones como parte del ciclo de vida de un proyecto XP contendrá la relación entre las historias de usuario y la iteración en que se abordará cada una, así como la duración de cada una de las iteraciones la cual debe encontrarse entre 1 y 3 semanas y el orden en que serán implementadas las historias[18].

Iteración	Orden de la historias de usuario a implementar	Duración total de la iteración
1era	Configurar dispositivo	
	Gestionar variables	3 semanas
2da	Diseñar Protocolo de comunicación	
	Comunicar dispositivos	3 semanas
3era	Gestionar transferencia de datos	3 semanas

Table 2.17: Plan de duración de iteraciones

2.5.5. Plan de entregas

El plan de entregas recoge las fechas aproximadas del fin de cada iteración y las entregas del sistema que se harán en cada una.

HU	Final 1ra Iteración.	Final 2da Iteración.	Final 3ra Iteración.
1-2-3	3ra semana de marzo (0.1)		3ra semana de mayo (1.0)
4-5		3ra semana de abril (0.2)	3ra semana de mayo (1.0)
6			3ra semana de mayo (1.0)

Table 2.18: Plan de entregas

2.6. Patrones de diseño

Los patrones de diseño son un conjunto de estrategias, o buenas prácticas, que pueden facilitar el trabajo en muchas situaciones a la hora de realizar una aplicación orientada a objetos. Además son los encargados de identificar clases, instancias, roles, colaboraciones y la distribución de responsabilidades[17].

2.6.1. Patrones de diseño utilizados en el manejador.

- **Singleton:** Mediante el patrón Singleton se garantiza que el sistema posea una instancia única disponible, esto permite una vía de acceso única al manejador de dispositivos. En la clase **Collector** se encuentran las funcionalidades principales del sistema y se garantiza y que solo exista una instancia de la misma.

- **Facade:** El patrón Facade en el sistema se evidencia al lograr utilizar una interfaz común para todas las interfaces del manejador. Esta interfaz facilita el acceso a las funcionalidades de la aplicación. La clase **Collector** contiene las principales funcionalidades del sistema, la misma será el punto de acceso a todas las funciones del software.

2.6.2. Patrones para asignar responsabilidades presentes en el manejador.

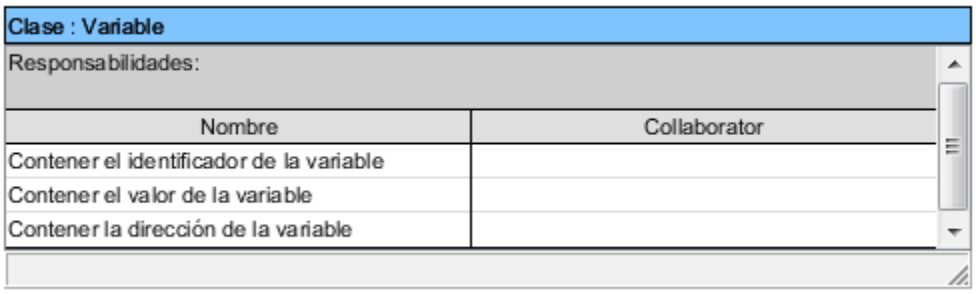
- **Creador:** En el sistema se aplica en la asignación de responsabilidades a las clases relacionadas con la creación de objetos, de forma tal que una instancia de un objeto sólo pueda ser creada por el objeto que contiene la información necesaria para ello. El uso de este patrón va a garantizar que existan las dependencias mínimas entre las clases que componen el manejador, lo que favorece al mantenimiento del sistema.

- **Experto:** En el sistema se aplica en la asignación de responsabilidades a las clases de forma tal que las mismas contengan la información necesaria para poder ejecutar una acción específica. El uso de este patrón permitirá contar con un manejador robusto y fácil de mantener ya que las relaciones entre las clases serán las menores posibles y los objetos del sistema harán uso de su propia información para hacer lo que se les pide.

2.7. Tarjetas de Clase, Responsabilidad y Colaboración.

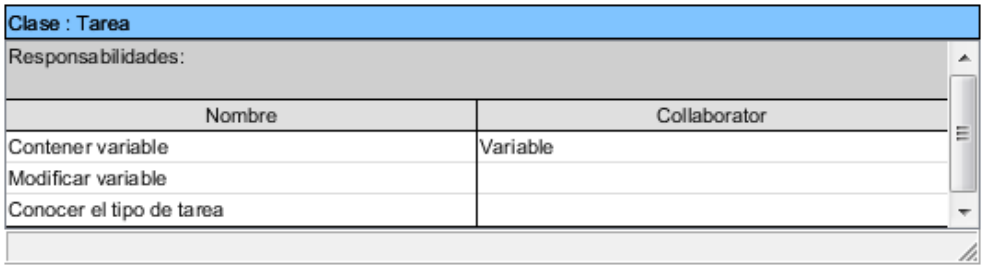
El uso de las tarjetas de Clase, Responsabilidad y Colaboración (CRC siglas en inglés) permite al programador enfocarse en el desarrollo orientado a objetos evitando los malos hábitos de la programación procedural clásica. Las tarjetas CRC representan objetos del sistema; la clase a la que pertenece el objeto se puede escribir

en la parte de arriba de la tarjeta, en una columna a la izquierda se pueden escribir las responsabilidades u objetivos que debe cumplir el objeto y a la derecha, las clases que colaboran con cada responsabilidad[18].



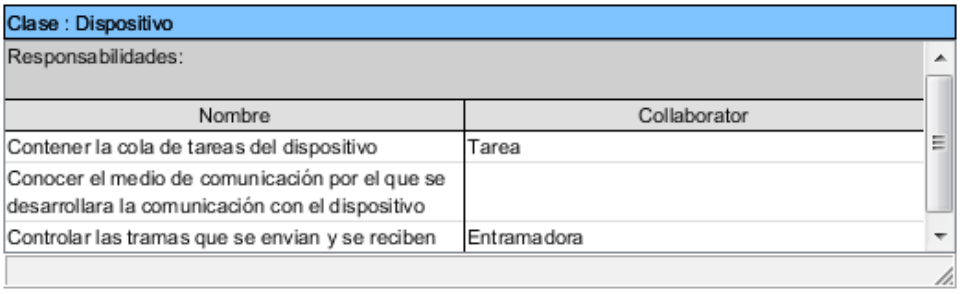
Clase : Variable	
Responsabilidades:	
Nombre	Collaborator
Contener el identificador de la variable	
Contener el valor de la variable	
Contener la dirección de la variable	

Figure 2.11: Tarjeta CRC-Variable



Clase : Tarea	
Responsabilidades:	
Nombre	Collaborator
Contener variable	Variable
Modificar variable	
Conocer el tipo de tarea	

Figure 2.12: Tarjeta CRC-Tarea



Clase : Dispositivo	
Responsabilidades:	
Nombre	Collaborator
Contener la cola de tareas del dispositivo	Tarea
Conocer el medio de comunicación por el que se desarrollara la comunicación con el dispositivo	
Controlar las tramas que se envian y se reciben	Entramadora

Figure 2.13: Tarjeta CRC-Dispositivo

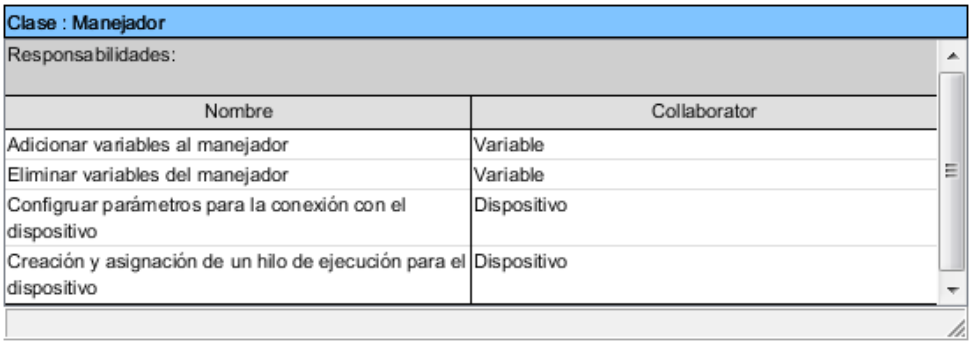


Figure 2.14: Tarjeta CRC-Manejador

Conclusiones

Basado en la información obtenida de los diagramas de clases y de paquetes generados en la fase de Planificación, la arquitectura 3-capas diseñada, el diseño del protocolo de comunicación entre el manejador y los dispositivos del sistema G-Hawk así como las historias de usuario recogidas, se puede decir que se obtuvo la información necesaria para comenzar con la implementación de la solución que dará respuesta a la problemática existente.

3 Implementación y Pruebas

Introducción

En XP se plantea que el proceso de implementación debe realizarse de forma iterativa, con el objetivo de que al alcanzar el fin de cada iteración se obtenga un producto funcional que después de ser probado y mostrado al cliente, ofrezca al equipo de trabajo una retroalimentarse a partir de la opinión de éste. En el presente capítulo se describen las tres iteraciones especificadas durante la etapa de diseño del sistema, exponiendo las tareas de ingeniería correspondientes a cada historia de usuario. Además se hace alusión al proceso de pruebas realizadas al sistema.

3.1. Iteración I

Esta iteración tiene como objetivo la implementación de las historias de usuario referente a la configuración del dispositivo y la gestión de variables. Al final de ésta se contará con una primera versión básica que recoja las funcionalidades antes expuestas. El cliente interactuará con esta versión con el objetivo de hacer uso de su punto de vista como retroalimentación para el grupo de trabajo.

Tarea de ingeniería	
Número de tarea: 1	**Número historia: 1**
Nombre tarea: Implementación de función con el objetivo de configurar y crear el dispositivo.	
Tipo de tarea: Desarrollo	**Puntos estimados: 4**
Fecha inicio:	**Fecha fin:**
Programador responsable: Héctor E. León García.	
Descripción: Esta función permitirá la configuración de la dirección IP y el puerto por el que se establecerá la comunicación con el dispositivo.	

Table 3.1: Tarea de historia de usuario: Configurar dispositivo

Tarea de ingeniería	
Número de tarea: 2	**Número historia: 1**
Nombre tarea: Implementación de función con el objetivo de crear y asignar al dispositivo un hilo de ejecución.	
Tipo de tarea: Desarrollo	**Puntos estimados: 4**
Fecha inicio:	**Fecha fin:**
Programador responsable: Héctor E. León García.	
Descripción: Se creará y asignará al dispositivo un hilo de ejecución.	

Table 3.2: Tarea de historia de usuario: Configurar dispositivo

Tarea de ingeniería	
Número de tarea: 1	**Número historia: 2**
Nombre tarea: Implementación de función para adicionar variables al manejador.	
Tipo de tarea: Desarrollo	**Puntos estimados: 4**
Fecha inicio:	**Fecha fin:**
Programador responsable: Héctor E. León García.	
Descripción: Las variables deben contener un identificador y una dirección.	

Table 3.3: Tarea de historia de usuario: Gestionar variables

Tarea de ingeniería	
Número de tarea: 2	**Número historia: 2**
Nombre tarea: Implementación de función para eliminar variables del manejador.	
Tipo de tarea: Desarrollo	**Puntos estimados: 4**
Fecha inicio:	**Fecha fin:**
Programador responsable: Héctor E. León García.	
Descripción: Las variables se eliminan a partir del identificador de las mismas.	

Table 3.4: Tarea de historia de usuario: Gestionar variables

Tarea de ingeniería	
Número de tarea: 3	**Número historia: 2**
Nombre tarea: Implementación de función para adicionar en caliente variables del manejador.	
Tipo de tarea: Desarrollo	**Puntos estimados: 4**
Fecha inicio:	**Fecha fin:**
Programador responsable: Héctor E. León García.	
Descripción: Se adicionan las variables del manejador luego de iniciado el proceso de intercambio de información.	

Table 3.5: Tarea de historia de usuario: Gestionar variables

Tarea de ingeniería	
Número de tarea: 4	**Número historia: 2**
Nombre tarea: Implementación de función para eliminar en caliente variables del manejador.	
Tipo de tarea: Desarrollo	**Puntos estimados: 3**
Fecha inicio:	**Fecha fin:**
Programador responsable: Héctor E. León García.	
Descripción: Se eliminan las variables del manejador luego de iniciado el proceso de intercambio de información.	

Table 3.6: Tarea de historia de usuario: Gestionar variables

3.2. Iteración II

El objetivo de esta iteración se basa en la implementación de las historias de usuario referentes al diseño del protocolo que regirá las comunicaciones entre el recolector y los dispositivos, así como la que se refiere a comunicar los dispositivos. Al finalizar se contará con una versión de prueba con las funcionalidades antes expuestas. Esta será mostrada al cliente con el objetivo de realizar cambios en base a las necesidades expresadas por mismo.

Tarea de ingeniería	
Número de tarea: 1	**Número historia: 3**
Nombre tarea: Diseño de protocolo de comunicación entre el manejador y los dispositivos.	
Tipo de tarea: Diseño	**Puntos estimados: 4**
Fecha inicio:	**Fecha fin:**
Programador responsable: Héctor E. León García.	
Descripción: Este protocolo brindará a los usuarios las características de la comunicación entre el manejador y los dispositivos. El protocolo contará con una lista de los posibles errores que pudiesen ocurrir en el proceso de transferencia de datos. El protocolo de comunicación será la base del manejador que se desea implementar.	

Table 3.7: Tarea de historia de usuario: Diseñar protocolo de comunicación.

Tarea de ingeniería	
Número de tarea: 1	**Número historia: 4**
Nombre tarea: Implementación de función para establecer comunicación con dispositivos.	
Tipo de tarea: Desarrollo	**Puntos estimados: 4**
Fecha inicio:	**Fecha fin:**
Programador responsable: Héctor E. León García.	
Descripción: Usando como base el protocolo de comunicación realizará la conexión del recolector con los dispositivos. La conexión se realizará por el medio especificado por el usuario en el dispositivo, configurando los parámetros específicos en cada caso.	

Table 3.8: Tarea de historia de usuario: Comunicar dispositivos

3.3. Iteración III.

Durante el transcurso de esta iteración se implementará la historia que se refiere al proceso de atender las tareas de los dispositivos. Al finalizar la misma se contará con una versión funcional básica del producto final. Como resultado de esta, el sistema será puesto en funcionamiento para que a partir de su comportamiento y las opiniones del cliente y del grupo de desarrolladores, trabajar sobre las mejoras en su diseño y funcionalidades.

Tarea de ingeniería	
Número de tarea: 1	**Número historia: 5**
Nombre tarea: Implementación de función para escribir datos sobre los dispositivos.	
Tipo de tarea: Desarrollo	**Puntos estimados: 5**
Fecha inicio:	**Fecha fin:**
Programador responsable: Héctor E. León García.	
Descripción: El proceso de escribir datos se basa en lo especificado en el protocolo de comunicación.	

Table 3.9: Tarea de historia de usuario: Gestionar transferencia de datos

Tarea de ingeniería	
Número de tarea: 1	**Número historia: 6**
Nombre tarea: Implementación de función para leer datos de los dispositivos.	
Tipo de tarea: Desarrollo	**Puntos estimados: 5**
Fecha inicio:	**Fecha fin:**
Programador responsable: Héctor E. León García.	
Descripción: El proceso de leer datos se basa en lo especificado en el protocolo de comunicación.	

Table 3.10: Tarea de historia de usuario: Gestionar transferencia de datos

3.4. Estilo de código

A la hora de programar es necesario seguir un estilo. Este permitirá revisar, mantener y actualizar el código de una manera más sencilla y ordenada. Seguir estas normas también evitará caer en errores y malas prácticas que dificultan la comprensión de las líneas de código[14].

Un código con un formato desordenado se hace más difícil de leer por lo que no es aceptable, lo más importante es la consistencia del código dentro de un programa o una biblioteca por lo que se debe seguir un estilo consistente. Seguidamente se muestra el estilo utilizado en el código fuente del manejador de dispositivos:

3.4.1. Nombres:

- Los nombres de las clases se escriben en singular.
- Los nombres de las clases garantizan que no exista una interpretación subjetiva.
- Los nombres de las clases serán siempre distintos a sus elementos para evitar redundancia.
- Se emplea minúscula para el inicio de los nombres de las variables, atributos y métodos ya que estos generalmente son el producto de concatenar varias palabras.
- De contener más de una palabra la estructura del nombre, las siguientes se escriben con mayúscula.
- Los nombres de constantes deben contener solo letras mayúsculas.
- Las abreviaturas serán mínimas se debe ser consistente en su uso y cada abreviación debe significar solo una cosa.
- Los nombres de los métodos serán denotados por frases que estén formadas por verbos.
- Los nombres de los atributos están formados por frases que incluirás sustantivos.
- Evitar el uso de nombres idénticos para distintos propósitos.

3.4.2. Codificación:

- Se establece un tamaño de indentación estándar de tres espacios, sin tabulaciones.
- Se alinean secciones del código.
- Se alinean verticalmente llaves de apertura y cierre.
- Se evita colocar más de una sentencia por línea.
- Se minimiza el alcance de las variables para evitar confusión y facilitar el mantenimiento.

- Se emplear cada variable y rutina solo para un propósito.
- Se evitar el uso de variables públicas.
- Se emplea i, j, k, l, p, q, r para contadores en ciclos.
- Se mantiene la modularidad del código bajo el criterio de la lógica que encierra.
- Se emplean correctamente los tipos de ciclos: si es al menos una vez usar do-while, si es ninguna o más veces usar while-do, y si se conoce el número exacto de ciclos usar for.
- Se inicializan todas las variables.

3.5. Pruebas del sistema

XP define que no debe existir ninguna funcionalidad en el programa que no haya sido probada, los desarrolladores escribirán pruebas para chequear el correcto funcionamiento del programa y los clientes realizarán pruebas funcionales. El resultado será un programa más seguro, capaz de soportar nuevos cambios a medida que pase el tiempo. Se deben hacer pruebas que verifiquen a fondo el funcionamiento del sistema, permitan revelar la calidad del producto desarrollado e identifiquen posibles fallos de implementación, calidad o usabilidad de un software.

3.5.1. Especificación del escenario de prueba.

El presente escenario de prueba permite validar la fiabilidad de la transferencia de datos que ofrece el manejador de dispositivo de manera remota, así como el cumplimiento de las funcionalidades descritas en el diseño de la aplicación. Se utilizó un escenario lo más real posible, solamente para probar las funcionalidades del manejador de dispositivo. Para la correcta realización de las pruebas se utilizaron una serie de recursos que se exponen a continuación:

Recursos físicos: Se utiliza una máquina de 128MB de RAM, con un microcontrolador ARM-9 a una velocidad de 400 MHz, con red cableada.

Recursos lógicos: El sistema operativo sobre el cual se realizan las pruebas está basado en GNU/Linux, cuenta con un tamaño de la caché 261 MB, consume una RAM en consola de 9.7 MB. Debe tener instalado la biblioteca TransportLayer.

3.5.2. Caso de prueba #1

Descripción: El caso de prueba permite verificar la configuración del dispositivo con el cual el manejador realizará la transferencia de información. Se configuran los parámetros (IP, Puerto) que exige la comunicación por la cual se realizará la transferencia.

Caso #	Clases válidas	Clases inválidas	Resultados esperados	Resultados obtenidos	Observaciones
1	Al iniciarse la aplicación se configuran los parámetros IP y Puerto para la conexión con el dispositivo.	Realizar otra operación antes de configurar los parámetros del dispositivo.	Quedará configurada la conexión con el dispositivo con el cual se realizará el intercambio de información.	Queda configurada la conexión con el dispositivo que se comunicará con el manejador.	El parámetro IP se refiere a una cadena que emule una dirección IP real. El parámetro Puerto se refiere a un número entero que emule un puerto de conexión del sistema operativo.

Cuadro 3.11: Caso de prueba #1

3.5.3. Caso de prueba #2

Descripción: El caso de prueba permite verificar la gestión de las variables del manejador. Se adicionan y eliminan objetos de tipo variable al manejador.

Caso #	Clases válidas	Clases inválidas	Resultados Esperados	Resultados obtenidos	Observaciones
2	Se procede a adicionar variables al manejador.		Se adicionarán variables al manejador.	Quedan adicionadas variables al manejador de dispositivos	
2.1	Se procede a eliminar variables del manejador.		Se eliminarán variables del manejador.	Quedan eliminadas variables del manejador de dispositivos	
2.2	Se procede a adicionar variables en caliente al manejador.		Se adicionarán variables en caliente al manejador.	Quedan adicionadas en caliente variables al manejador de dispositivos	
2.3	Se procede a eliminar variables en caliente al manejador.		Se eliminarán variables en caliente del manejador.	Quedan eliminadas en caliente variables del manejador de dispositivos	

Table 3.12: Caso de prueba #2

3.5.4. Caso de prueba #3

Descripción: El caso de prueba permite verificar que se establezca correctamente la comunicación del manejador con el dispositivo. Además garantiza conocer a partir de la lectura y escritura de datos la fiabilidad que brinda el manejador en el proceso de transferencia con el dispositivo.

caso#	Clases válidas	Clases inválidas	Resultados esperados	Resultados obtenidos	Observaciones
3	Se procede a establecer comunicación con el dispositivo.		Se establecerá comunicación con el dispositivo con el que se realizará el intercambio de información.	Se establece comunicación con el con el dispositivo con el que se realizará el intercambio de información.	
3.1	Se procede a escribir datos sobre el dispositivo.		Se escribirán datos correctamente sobre el dispositivo	Se escriben datos correctamente sobre el dispositivo	
3..2	Se procede a leer datos del dispositivo.		Se leerán datos correctamente del dispositivo	Se leen datos correctamente del dispositivo	

Table 3.13: Caso de prueba #3

Conclusiones

Al finalizar el presente capítulo donde se hace alusión a las etapas de implementación y pruebas del software, se puede afirmar que el protocolo de comunicación ProC-ARM rige con eficiencia el intercambio de información fiable entre el manejador y los dispositivos que encuesta. Además es válido afirmar que el manejador garantiza el control eficiente de los dispositivos con los que se comunica.

4 Conclusiones.

Luego de finalizada la investigación para el desarrollo del manejador para la comunicación fiable en un sistema de supervisión y control basado en microcontroladores se concluye que:

- A pesar de la existencia de otros protocolos, TCP es factible para comunicar los dispositivos del sistema G-Hawk.
- El diseño de un protocolo para la comunicación entre los dispositivos del sistema G-Hawk, garantiza fiabilidad en el proceso de transferencia de información.
- El manejador realiza un control eficiente sobre los dispositivos que encuesta.
- El desarrollo de un manejador a partir del protocolo ProC-ARM para la comunicación entre dispositivos, permite el intercambio de datos de forma fiable en el sistema de supervisión y control G-Hawk.

5 Recomendaciones.

- Incorporar a la implementación del manejador, lo necesario para garantizar el intercambio de información de manera fiable a partir de una conexión Puerto Serie.
- Incorporar a la implementación del manejador un planificador de tareas, con el objetivo de optimizar la frecuencia de la lectura y escritura de registros en los dispositivos.
- Implementar al manejador el concepto de variables en bloque, con el objetivo de mejorar las funcionalidades de lectura y escritura de registros en los dispositivos.
- Incorporar a la implementación del manejador la lectura y escritura de registros a partir de los tipos de datos **String y Float.**

Bibliografía

[1] Protocolo de datagramas de usuario (UDP). http://technet.microsoft.com/es-es/library/cc785220(v=ws.10).aspx.

[2] Comunicación serial. http://digital.ni.com/public.nsf/allkb/039001258CEF8FB686256E0F005888 June 2006.

[3] Protocolo de comunicaciones DNP3. http://www.micros-designs.com.ar/protocolo-de-comunicaciones-dnp3/, 2010.

[4] Anne Furey and Arjan Pottjewijd. What is integrated development environment (IDE). http://searchsoftwarequality.techtarget.com/definition/integrated-development-environment, July 2001.

[5] Antonio Cedeno Pozo. *Módulos de adquisición y análisis para la interacción con dispositivos de campo en un SCADA.* PhD thesis, Universidad de las Ciencias Informáticas, 2009.

[6] Barrios Dueñas, Joel. Introducción a TCP/IP. http://www.alcancelibre.org/staticpages/index.php/introduccion-tcp-ip.

[7] Bautista,Osire. Lenguaje de modelado unificado - documentos, July 2009.

[8] Bjarne Stroustrup. Historia del lenguaje c++ : Informatica full. http://informatica-full2.blogspot.com/2009/06/historia-del-lenguaje-c.html, June 2009.

[9] De Nobrega, Maria. Herramientas CASE: rational rose. por maria de nobrega | curso sistemas de información II. http://curso_sin2.blogia.com/2005/060401-herramientas-case-rational-rose.-por-maria-de-nobrega.php.

[10] Felipe U. Pérez García. Clasificación de los lenguajes de programación. http://www.articulandia.com/premium/article.php/06-09-2006Clasificacion-de-los-lenguajes-de-programacion.htm, June 2006.

[11] Gerardo Fern ández Escribano. Introducci ón a extreme programming, 2002.

[12] Ing. Henry Mendiburu Díaz. Sistemas SCADA.

[13] Jordi Bartolome. El protocolo MODBUS. http://www.tolaemon.com/site/protocolo_modbus, 2011.

[14] José Antonio Aragón Cáceres and Beatriz Llanes Jiménez. *Servicio de Integración con Terceros para el acceso a variables del sistema SCADA Guardián del ALBA.* PhD thesis, May 2009.

[15] José Briseno Márquez. Transmisión de datos, 2005.

[16] Julio César Chavez Urrea. Protocolo TCP/IP - monografías.com. http://www.monografias.com/trabajos/protocolotcpip/protocolotcpip.shtml, 2009.

[17] Luis H. Fernandez. Patrones de diseno. http://software.guisho.com/patrones-de-diseno, 2009.

[18] Ma Carmen Penadés, Patricio Letelier, and José H. Canós. Metodologías Ágiles en el desarrollo de software.

[19] Manuel Calero Solís. Una explicación de la programación extrema (XP), 2003.

[20] Martin Fowler. La nueva metodologia. http://www.programacionextrema.org/articulos/newMethodology.es.html, 2003.

[21] Miguel Alejandro Soto. Protocolos TCP/IP. http://usuarios.multimania.es/janjo/janjo1.html.

[22] Pedro Alberto Uriarte Rodríguez. *Manejador para la comunicación con dispositivos de campo mediante el protocolo DF1.* PhD thesis, Universidad de las Ciencias Informáticas, March.

[23] Ricardo Colusso and Juan Gabardini. Desarrollo Ã¡gil de software. http://knol.google.com/k/desarrollo-%C3%A1gil-de-software#Caracter(C3)(AD)sticas_del_Desarrollo_(C3)(81)gil.

[24] Rockwell Automation. PRODUCT PROFILE RSVIEW32, 2007.

[25] Supervisory Systems Fast/Tools. FAST/TOOLS, 2003.

[26] Swales Andy. OPEN MODBUS/TCP SPECIFICATION, March 1999.

[27] Tim Ficher. Driver. *About.com PC Support.*