Universidad de las Ciencias Informáticas

Centro de Informática Industrial

Facultad 5

Firmware para tarjeta de adquisición basada en un microcontrolador STM32

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas.

Autor: Manuel Alejandro Oliva Labaut

Tutores:

Ing. Amides Rodríguez Rodríguez

Ing. Julio Alberto Leyva Durán

La Habana, junio de 2012

DECLARACIÓN DE AUTORÍA

Declaro ser el único autor de este trabajo y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter no exclusivo.

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

_____

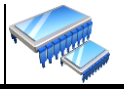
Manuel Alejandro Oliva Labaut
Autor

_____

Amides Rodríguez Rodríguez
Tutor

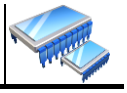
_____

Julio Alberto Leyva Durán
Tutor



"Al fin y al cabo, somos lo que hacemos para cambiar lo que somos"

Eduardo Galeano
(Uruaguay, 1940)



AGRADECIMIENTOS

A mi hijo Royber por darme fuerzas cuando la oscuridad se acercaba, por ser mi asignatura pendiente y próxima tesis.

A mi madre por dármelo todo, sin reservas, incluso su vida si fuese necesario; traerme hasta aquí a base de esfuerzo, sacrificios, ejemplo, humildad, por darme todo el amor del mundo; por deberle todo lo fui, lo que soy y lo que seré.

A mi abuela Dulce por criarme con tanto esfuerzo y voluntad, soportarme tantas malacrianzas y alumbrarme el camino.

A mi padre por ser mi amigo y permitirme ser su padre; por ser mi ejemplo desde niño e impulsarme a alcanzar mis metas. Por su apoyo extraordinario en las buenas y en las malas.

A mi prima Mailén por ser mi hermanita del alma, tan especial e incondicional y por regalarme mi primera sobrina.

A mi hermano Sergio (Rafelín) por demostrarme que la amistad puede más que cualquier distancia, por darme siempre su consejo oportuno, por su apoyo invaluable.

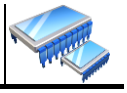
A mis amigos Alkaid y Solangel sin los cuales este resultado no hubiese sido posible.

Al resto de mis amigos de la Universidad y de la Vocacional por aguantarme con todos mis defectos.

A mi familia en general por tanta preocupación y apoyo.

A Amides, mi tutor, por confiar en mí y representarme cuando la situación más lo requería.

ÍNDICE



INTRODUCCIÓN

La Automatización Industrial ha devenido en las últimas décadas como uno de los principales medios para elevar la eficiencia y la eficacia de las empresas industriales modernas. Entre las principales ventajas que la automatización proporciona al universo empresarial están el significativo aumento de la producción, el aumento de la calidad de sus productos, el ahorro de recursos, etc., elementos que se traducen directamente en mayor utilidad. Otros beneficios que se pueden mencionar son:

- ✓ Acumular un conocimiento más detallado sobre el proceso de producción, mediante la recopilación de información y datos estadísticos.
- ✓ Lograr un mejor conocimiento del funcionamiento y rendimiento de equipos y máquinas que intervengan en el proceso.

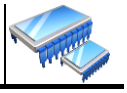
A pesar de los aspectos positivos mencionados anteriormente, pequeñas empresas se ven imposibilitadas de implantar sistemas que automaticen sus procesos industriales por el gran capital necesario para la inversión, así como por temor al aumento de las tareas de reparación y mantenimiento a maquinarias y medios, actividades que consumen no pocos recursos financieros.

La necesidad de automatizar pequeños procesos, reducir el tamaño y los costos de los grandes sistemas de control, ha propiciado la migración hacia sistemas de control empotrados que son implementados a través de *firmware*. Un *firmware* es "el software que establece la lógica de bajo nivel y controla los circuitos electrónicos de algún tipo de dispositivo", (Rushinek, 1985).

En la Universidad de las Ciencias Informáticas (UCI), el Centro de Informática Industrial (CEDIN) perteneciente a la Facultad 5, tiene la misión de "**generar soluciones integrales para la Industria**" y precisamente se ha propuesto ofrecer una solución integral (hardware + software), denominada PLC[1] – HMI[2], que sea genérica para la supervisión y control de procesos industriales con la particularidad de que, por su bajo costo económico, sea viable para la mayoría de las empresas.

La solución PLC – HMI no pretende ser tan ambiciosa como el SCADA Guardián del ALBA[3] (producto insignia del Centro) que tiene capacidad para procesar 12,500 variables, sino que la reducción en el costo viene exactamente de la fácil adaptabilidad del producto entre los distintos clientes, la posibilidad de emprender un despliegue rápido del sistema por su sencillez, una capacidad de procesamiento de hasta 1,000 variables y una arquitectura centralizada con un importante ahorro de capital por la supresión de los nodos que contempla la arquitectura distribuida del sistema Guardián del ALBA.

[1] [3] [4] Véase Glosario de Términos.



Para esta solución, la línea de Sistemas Empotrados perteneciente al CEDIN cuenta con una tarjeta de escasos recursos de hardware basada en un procesador ARM Cortex-M3. La misma fue diseñada por integrantes de la línea a partir de un microcontrolador STM32, con el propósito de desarrollar una tarjeta de adquisición de datos para procesos industriales. Esta tarjeta no puede ser integrada a la solución prevista pues no cuenta con el componente necesario para que sea funcional en la adquisición de datos y en la ejecución de tareas de control.

La situación problemática presentada anteriormente plantea como **Problema Científico** la inexistencia de un mecanismo que permita la integración funcional de la tarjeta de adquisición de datos basada en un microcontrolador STM32 al producto PLC – HMI.

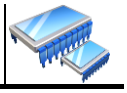
El **objeto de estudio** del presente trabajo son las acciones de control y adquisición de datos sobre procesos industriales, teniendo como **objetivo general** desarrollar un *firmware* para la tarjeta de adquisición de datos basada en un microcontrolador STM32 que permita la adquisición de datos y el flujo de la información digitalizada desde y hacia el PLC – HMI.

El **campo de acción** está enmarcado en el control y la adquisición de datos sobre procesos industriales haciendo uso de tarjetas de adquisición de datos.

El principal aporte práctico esperado del trabajo lo constituye un *firmware* para la tarjeta basada en un microcontrolador STM32, que posibilite la conversión de señales analógicas en señales digitales y viceversa. El *firmware* debe permitir, además, el muestreo y el acceso a esta información por el módulo de adquisición del PLC – HMI mediante un protocolo de comunicación, así como la recepción y ejecución de comandos que sean enviados desde dicho módulo hacia la tarjeta de adquisición de datos.

Para complementar el objetivo general de este trabajo se definen las siguientes **tareas de investigación**:

- ✓ Definir conceptos asociados a microcontroladores, sistemas empotrados, sistemas operativos de tiempo real, hardware de adquisición de datos y *firmware* como base de la investigación a desarrollar.
- ✓ Analizar y sintetizar métodos, técnicas y las herramientas de software libre empleadas en el desarrollo de *firmware*.
- ✓ Establecer y describir los requisitos funcionales y no funcionales del *firmware*.
- ✓ Establecer la arquitectura basada en patrones aplicables a sistemas empotrados.
- ✓ Implementar y validar la solución propuesta a partir de los requerimientos críticos.



Para desarrollar las tareas especificadas anteriormente, se combinarán los métodos y técnicas de investigación científica que se mencionan a continuación, con el objetivo de que conduzcan la búsqueda y el procesamiento de la información.

Métodos teóricos

Método analítico – sintético: Se utilizará para el estudio de cada una de las características y el desempeño de los distintos módulos que integrarán el *firmware*, así como la interrelación de estos para soportar las funcionalidades que deben ser implementadas.

Método de la abstracción – modelación: Será empleado en la comprensión profunda sobre cómo se comporta el hardware del microcontrolador STM32, estableciendo ese conocimiento como punto de partida para el desarrollo del *firmware* que permita manejar, manipular, interactuar de forma abstracta con el hardware.

Métodos empíricos

Método de consulta a expertos: Se aplicará para aprovechar la experiencia del equipo de la línea Sistemas Empotrados, con su juicio intuitivo acerca de técnicas y herramientas útiles para el desarrollo del *firmware*; para dar respuesta a las posibles dificultades técnicas que se presenten durante el desarrollo. Este método permitirá contar con criterios confiables al valorar alternativas en la toma de decisiones, involucrando plenamente a los expertos en la implementación de la solución.

Método experimental: Será aplicado durante todo el proceso de construcción de la solución analizando el resultado de la interacción de la tarjeta de adquisición de datos con el *firmware*, realizando pruebas intermedias que permitan y aseguren el avance de la solución.

Este presente trabajo está estructurado en 4 capítulos. En el **Capítulo 1 Fundamentación Teórica** se definen los principales conceptos asociados a la problemática así como el estado del arte de las tecnologías sintetizadas. Dentro del **Capítulo 2 Métodos y Herramientas** se examinan los principales métodos, técnicas y herramientas asociadas al tema en cuestión. Ya en el **Capítulo 3: Descripción de la Solución** se muestra una visión teórica del sistema desarrollado y por último en el **Capítulo 4: Validación de la Solución** se detallan las actividades realizadas para la validación del *firmware*.

Capítulo 1

FUNDAMENTACIÓN TEÓRICA

En este capítulo se abordan los elementos teóricos necesarios para la comprensión del problema a resolver. Se enuncian los conceptos asociados a la problemática, así como el estado del arte de las tecnologías relacionadas con la adquisición de datos sobre procesos industriales a través de sistemas empotrados.

1.1. Microcontroladores

Han pasado más de 30 años desde la aparición de los microcontroladores y pocos circuitos integrados han tenido tan buena acogida o han sido tan versátiles como estos. Son esenciales en áreas tan disímiles como la industria, el transporte, las comunicaciones, la medicina y los podemos encontrar en dispositivos tan simples como calculadoras de bolsillo, reproductores de DVD y cámaras digitales. Las secciones siguientes tienen como fin hacer una introducción de los microcontroladores y sus principales características.

1.1.1. Definición

La generalidad de las definiciones de "microcontrolador" planteadas por los expertos sobrepasan el ámbito electrónico y presentan a los microcontroladores como computadoras miniaturizadas. Las siguientes definiciones aclaran lo mencionado anteriormente:

- ✓ "El microcontrolador es un computador completo, aunque de limitadas prestaciones, que está contenido en un chip de un circuito integrado", (Cohen Manrique, 2005).
- ✓ "Es un sistema de control completo basado en un microprocesador pero construido en un solo chip", (Crisp, 2004)

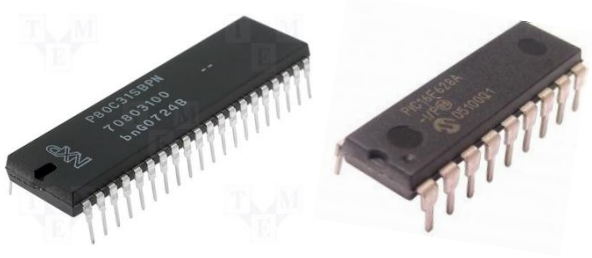


Fig. 1 Microcontroladores: a) NXP b) PIC

En resumen un microcontrolador puede definirse como una microcomputadora, un sistema cerrado que contiene en su interior la mayoría de los chips necesarios para su funcionamiento, dígase procesamiento, memoria volátil y no volátil, y soporta la conexión de varios dispositivos de entrada y salida (E/S).

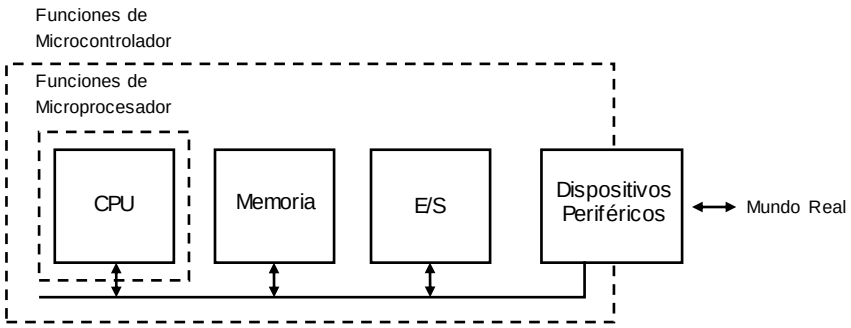


Fig. 2 Esbozo de la arquitectura de un microcontrolador

1.1.2. Características

Los microcontroladores son populares por ser pequeños y de bajo costo, sus componentes son elegidos con el fin de minimizar su tamaño y producirlos tan barato como sea posible. Entre sus principales características podemos encontrar que son:

- ✓ **Empotrados:** Se encuentran "empotrados" en el interior de dispositivos para controlar las características o acciones de este.
- ✓ **Dedicados:** Están programados, generalmente, para realizar una sola tarea y ejecutar un programa en específico, el que se almacena en la memoria no volátil y normalmente no varía.
- ✓ **Bajo consumo de energía:** Un microcontrolador que funcione con una batería consume aproximadamente 50 miliwatts, (Calcutt, y otros, 2004). Que sean bajo consumidores de energía garantiza un funcionamiento a largo plazo en ambientes extremos.

Arquitectura

Durante la evolución de los microcontroladores, fueron demarcándose dos tendencias en el diseño de su arquitectura: abierta o cerrada. Estos diseños responden directamente a las necesidades de los usuarios.

Los microcontroladores que implementen una arquitectura de diseño cerrada no admiten expandir los recursos de hardware disponibles. Poseen determinada cantidad de memoria volátil y memoria no volátil, un número establecido de E/S y cierta cantidad de recursos auxiliares. Estos elementos son prácticamente invariables y el programa que se ejecute debe de ajustarse a la estructura y los recursos del microcontrolador.

Por otro lado, los microcontroladores con una arquitectura de diseño abierta se identifican porque además de contar con una estructura interna determinada, aprovechan sus líneas de E/S (pines) para ofrecer buses de control, buses de direcciones y buses de datos con los cuales ampliar la memoria volátil y/o conectar circuitos integrados externos.

En cuanto a *arquitectura interna de memoria,* los microcontroladores habitualmente están basados en la arquitectura **Harvard** que propone memorias separadas para programas y datos, en contraposición, los sistemas basados en microprocesadores suelen tener una arquitectura **Von Neumann** que establece una sola memoria para uso indistinto de programas y datos.

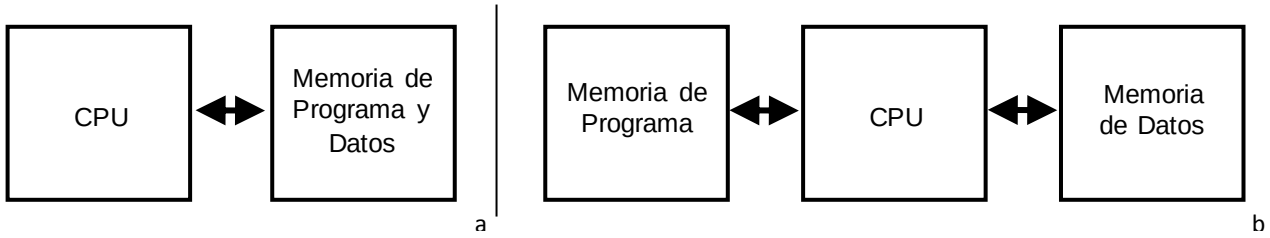


Fig. 3 Arquitecturas de los microcontroladores: a) Von Neumann b) Harvard

La enorme ventaja que aporta la arquitectura Harvard a las aplicaciones empotradas tiene que ver con los dos tipos de memoria que comúnmente se utilizan en este tipo de sistemas. El programa y las constantes se almacenan en la memoria no volátil mientras que el almacenamiento de datos y variables puede realizarse en la memoria volátil. Esta última pierde su contenido cuando se interrumpe el suministro eléctrico mientras que la memoria no volátil siempre mantiene los datos almacenados, incluso después de desconectar la alimentación eléctrica.

La arquitectura de Harvard también tiene la ventaja de contar con interfaces separadas para las operaciones en memoria, (Arnold, 2000), permitiendo el doble de velocidad en la transferencia desde ambas memorias (datos y programas) pues la búsqueda de las instrucciones a ejecutar ocurre en paralelo con la transferencia de datos.

Tamaño de registros de datos

Otro elemento importante sobre los microcontroladores es la cantidad de datos con que pueden operar en una instrucción atómica. Este parámetro los clasifica en de 4, 8, 16, 32 o 64 bits. Un microcontrolador de 8 bits tiene que realizar más de una operación para trabajar con datos de 16 bits, proceso que no ocurre así en un microcontrolador de 16, 32 o 64 bits, lo que hace a los últimos más potentes.

A pesar de que los microcontroladores de 32 y 64 bits son superiores al resto (lógicamente realizan menos operaciones al realizar cálculos con datos de 64 bits), estos solo se reservan para aplicaciones que requieran grandes prestaciones de hardware, como la reproducción de vídeo o el procesamiento de imágenes. Los microcontroladores que hoy dominan el mercado son los de 8 bits. Destacar que los primeros en surgir, los de 4 bits, todavía son ampliamente usados en sectores como el automovilismo, pues se adaptan a las necesidades del usuario y se hace irracional emplear microcontroladores más

potentes y en consecuencia más caros.

Interrupciones

Por sus aplicaciones en la industria y en otras ramas, los microcontroladores deben garantizar una respuesta predecible a los eventos que acontezcan. Por tal motivo, es característico de estos proporcionar una gama de interrupciones de hardware (en inglés denominada signal) y en muchos casos también brindan interrupciones de software (instrucción en ensamblador). (Zurrell, 2000) comenta que comúnmente los microcontroladores disponen de pines asociados a Interrupts ReQuest (IRQ, siglas en inglés de petición de interrupción) pero en sistemas con escasa disponibilidad de pines, las señales de IRQ no pueden estar expuestas o deben ser multiplexadas con otras señales E/S.

En la mayoría de los procesadores al lanzarse una interrupción estas son reconocidas de forma sincrónica: el procesador completa la instrucción en curso antes de ocuparse de la interrupción. Análogamente durante el reconocimiento asíncrono, la ejecución del procesador se detiene en la instrucción en curso y atiende el servicio de interrupción. Si bien el reconocimiento asíncrono es más rápido que el sincrónico, deja abierta la posibilidad de que el código de la interrupción interfiera con la instrucción en curso, proceso bastante riesgoso para sistemas críticos como son los sistemas empotrados.

Interrupciones vectorizadas y no vectorizadas

El tratamiento no vectorizado de las interrupciones constituye uno de los esquemas más simples para su tratamiento. Siempre que se lance una interrupción, el programa salta a una dirección específica donde el programador del microcontrolador chequea, solo una interrupción a la vez y cuál periférico ha causado la interrupción. Este método es comúnmente usado en microcontroladores que tienen diseñadas menos de 5 interrupciones. El método vectorizado es un poco más rápido y fácil de configurar, pero el programador del microcontrolador tiene menos control sobre el sistema, (Kalra, y otros, 2010). Cuando ocurre una interrupción, el manejador de interrupciones de hardware pasa automáticamente a una dirección específica dependiendo del dispositivo que produjo la interrupción.

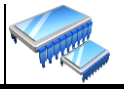
Componentes y periféricos

Memoria no volátil

Anteriormente se señaló que en los microcontroladores el programa normalmente se almacena en una memoria no volátil (ROM, siglas en inglés de *Read Only Memory*).

Para soportar esta función existen varios tipos de memorias no volátiles. Las más utilizadas actualmente para la programación de microcontroladores son:

EEPROM: La EEPROM (siglas en inglés de *Electrical Erasable Programmable Read Only Memory*)



constituye una memoria de sólo lectura, programable y borrable eléctricamente. Tanto el grabado como el borrado, se realizan eléctricamente desde una computadora a través de una interfaz. El número de veces que pueden realizarse ambas operaciones en una memoria EEPROM es finito, por lo que no es aconsejable una reprogramación continua. Se va extendiendo en los fabricantes la tendencia de incluir una pequeña zona de memoria EEPROM en los circuitos programables para guardar y modificar cómodamente una serie de parámetros que adecuan el dispositivo a las condiciones del entorno, (Zurrell, 2000). Los módems, por ejemplo, hacen uso del almacenamiento EEPROM para grabar las opciones de la configuración actual.

Flash: Se trata de una memoria no volátil de bajo consumo energético, que se puede escribir y borrar. Funciona como una ROM y una RAM (siglas en inglés de *Random Access Memory*), pero consume menos energía y es más pequeña. Es más rápida y de mayor densidad que la EEPROM. La alternativa Flash está recomendada frente a la EEPROM cuando se necesita gran cantidad de memoria no volátil, también el acceso de lectura es más rápido y soporta más ciclos de escritura/borrado, (Aguayo, 2004).

Memoria volátil

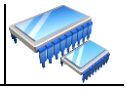
El almacenamiento de datos y de variables se realiza en la memoria volátil (RAM). Algunos microcontroladores cuentan con memoria RAM estática (SRAM), igualmente existen otros disponen de una memoria de datos del tipo EEPROM para estas funciones, (Cohen Manrique, 2005).

Puertos de E/S

Los puertos de E/S permiten al microcontrolador establecer una comunicación y controlar, a través de interfaces u otros dispositivos, a relés, interruptores, teclados, pantallas LCD (siglas en inglés de *Liquid Crystal Display*, pantalla de cristal líquido), sensores, etc. Estos puertos son la principal utilidad de los pines de un microprocesador. Dentro de ellos se encuentran los puertos de comunicación:

Puerto serie: Este periférico existe en los microcontroladores como UART (*Universal Asynchronous Receiver Transmitter*) o USART (*Universal Synchronous/Asynchronous Receiver Transmitter*) dependiendo de si permiten o no el modo sincrónico de comunicación. El objetivo de este periférico es la comunicación con otro microcontrolador o con una computadora. Se implementa a través de las interfaces RS-232, RS-485, RS-422.

Puerto serie sincrónico: Este tipo de periférico se utiliza para comunicar al microcontrolador con otros microcontroladores o con periféricos externos conectados a él a través de las interfaces SPI (*Serial Peripheral Interface*) o I^2C (*Inter-Integrated Circuit*). Su carácter es únicamente sincrónico y no están diseñados para interconectar el sistema con otros dispositivos independientes como una computadora,



(Valdés Pérez, y otros, 2007), sino para conectar dispositivos tales como memorias, pantallas LCD, conversores análogo/digital (A/D) o digital/análogo (D/A), entre otros.

Ethernet: La interfaz de comunicación Ethernet se utiliza mayormente para la comunicación del microcontrolador con otro microcontrolador o con una computadora. Es muy útil en la integración de los sistemas de control basado en microcontroladores para realizar tareas de monitoreo y control remoto de los sistemas a través de HMI (siglas en inglés de *Human – Machine Interface*, interfaz hombre - máquina).

Universal Serial Bus (USB): Mientras que los periféricos sintetizados anteriormente se usan indistintamente para la comunicación entre microcontroladores, microcontroladores con otros periféricos o con una computadora, la interfaz USB normalmente se utiliza para la comunicación del microcontrolador con una computadora y con la finalidad de acceder a memoria no volátil en los casos que esto sea técnicamente posible.

Recursos especiales

Temporizadores y contadores (Timers)

Son circuitos sincrónicos utilizados en el control de períodos de tiempo (temporizadores) o para el conteo de los pulsos externos que llegan a una entrada (pin) del microcontrolador (contadores). En ambos casos se carga un valor en los registros de conteo y seguidamente este valor va incrementando o disminuyendo. Esto sucede al ritmo de los impulsos de reloj para los temporizadores o por cambios de nivel en una entrada del microcontrolador para los contadores. Cuando el valor sobrepasa el límite o llega a 0 se lanza una interrupción.

Conversor A/D (ADC)

Muchas de las magnitudes físicas son representadas por señales analógicas y estas no pueden ser interpretadas por sistemas de control basado en microprocesadores. Normalmente los microcontroladores incorporan un conversor A/D con el objetivo de procesar señales analógicas provenientes, por ejemplo, de sensores. El conversor A/D es uno de los periféricos más codiciados en el mundo de los microcontroladores y llegan a marcar la diferencia entre un fabricante y otro.

Conversor D/A (DAC)

Contrariamente a los conversores A/D que son utilizados para obtener datos, el conversor D/A transforma los datos digitales resultantes de un proceso en su correspondiente señal analógica que puede ser exportada a través de un pin del microcontrolador y estar disponible para alguna interfaz de comunicación.

Modulador de ancho de impulsos (PWM)

Los PWM (Pulse Width Modulator) ofrecen al exterior impulsos de anchura variable (basado en funciones sinusoidales), muy útiles sobre todo para el control de la velocidad de giro de motores de paso.

Controlador de área de red (CAN)

CAN es un protocolo de bus serie para conectar sistemas individuales y sensores. La interfaz de bus CAN utiliza un esquema de transmisión asíncrono controlado por bits de inicio y de parada al principio y al final de cada carácter.

1.1.3. Estado del arte de los microcontroladores

La demanda de microcontroladores ha aumentado enormemente a partir del año 2000. El desarrollo de dispositivos cada vez más inteligentes, pequeños y sencillos ha convertido el mercado de microcontroladores en una de las industrias más rentables del mundo.

Datos de la compañía DataBeans[4] aseguran que el mercado de los microcontroladores alcanzó los 12.3 billones de dólares durante el 2010 y pronostican un aumento del 33.52 % para el 2015 llegando a 18.5 billones de dólares, (Dhadiwal Baid, 2010). Los sectores en que más avance se refleja son:

Gestión energética: Aplicaciones para sistemas y productos basados en energía solar (sistemas fotovoltaicos e inversores de potencias DC – AC), turbinas eólicas, sistemas de calefacción, redes inteligentes que dirijan eficientemente la distribución y gestión energética.

Electrónica automotriz: Control de sistemas de seguridad como bolsas de aire anti choque, sistemas de dirección ABS, control avanzado del motor, sistemas de encendido sin llave (keyless entry), navegación basada en GPS. Uso en vehículos híbridos de sistemas de control basados en microcontroladores para emisión de gases y para garantizar el consumo eficiente de combustible.

Electrónica médica: Dispositivos de asistencia médica portables como: monitores para la presión arterial, glucómetros y sistemas de ultrasonidos portables, equipamiento para gimnasios, miniaturización de dispositivos médicos como: marcapasos, riñones artificiales, etc.

Automatización industrial: Sistemas de adquisición de datos, sensores avanzados, control de turbinas, sistemas para mediciones inteligentes, proyección de HMI para sistemas de control.

Principales proveedores

La siguiente tabla muestra un ranking ordenado por los ingresos de los principales proveedores de microcontroladores durante el 2008:

[4] Firma dedicada a la investigación de mercados y centrada en la industria de semiconductores y la electrónica.

Compañías	2008		
	RANK	$ MILLONES	% MERCADO
Renesas Technology	1	2770	20.1 %
Freescale Semiconductor	2	1518	11.0 %
NEC	3	1330	9.7 %
Fujistsu	4	1065	7.7 %
Infineon Technologies	5	983	7.2 %
Microchip Technology	6	812	5.9 %
STMicroelectronics	7	645	4.7 %
Texas Instruments	8	601	4.4 %
Atmel	9	511	3.7 %
NXP Semiconductors	10	286	2.1 %
Otras	-	3229	23.5 %
TOTAL	-	13749	-

Tabla 1 Listado de los diez fabricantes de microcontroladores más grandes del mundo (Año 2008)

A continuación se muestra una breve reseña de tres de ellas:

Microchip

Microchip Technology fabrica los populares microcontroladores PIC con más de 700 productos diferentes que abarcan una amplia gama con microcontroladores de 8, 16 y 32 bits y controladores de señales digitales (DSC). Estos cubren el mercado de microcontroladores multipropósito y brindan varios periféricos especializados para hacer frente a aplicaciones tales como control de motores, la detección táctil, gráficos, fuentes de alimentación USB y Ethernet conmutadas. Todos los microcontroladores son compatibles con el Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) MPLAB de Microchip.

Renesas

Los microcontroladores Renesas de 8 y 16 bits así como las familias M16C R8C y H8S, han sido muy bien recibido en el mercado por su rendimiento y compatibilidad electromagnética. Renesas también ha creado (después de un gran análisis de las aplicaciones en términos de códigos de operación, arquitectura y consumo de energía para satisfacer las necesidades del mercado) un nuevo núcleo de 32 bits, llamado Renesas eXtreme (RX). Este núcleo puede alcanzar 200 MHz y 0,03 mA / MHz.

STMicroelectronics

El portafolio de 8-bit (STM8) de STMicroelectronics consta de tres familias: STM8S, STM8L y STM8A.

Mientras que los dispositivos STM8S son los más adecuados para aplicaciones industriales, de ultra bajo consumo de energía, los dispositivos STM8L son perfectos para la medición, pequeños equipos médicos y otras aplicaciones portátiles. La familia STM8A está dedicada para aplicaciones de automoción. En la gama de 32-bit, la familia STM32 se basa en el núcleo ARM Cortex-M que combina un alto rendimiento, bajo consumo de energía y operación a bajo voltaje. La serie STM32F aborda una amplia gama de aplicaciones con diseños destinados a ambientes extremos con un alto consumo de memoria. Los precios de venta son accesibles.

1.2. Sistemas empotrados

Los microcontroladores comúnmente vienen alojados en dispositivos o equipos formando parte de sistemas empotrados. (Qing, 2003) ofrece una definición amplia del término:

> *"Los sistemas empotrados son sistemas informáticos con el hardware fuertemente acoplado y con integración de software, que están diseñados para llevar a cabo una función específica. La palabra empotrado refleja el hecho de que estos sistemas son generalmente una parte integral de un sistema más grande, conocido como el sistema de empotramiento".*

Y (Heath, 2003) aporta los siguientes elementos:

> *"Están construidos para controlar una función o serie de funciones y no están diseñados para ser programados por el usuario final de la misma manera en que lo es una PC".*

En resumen un sistema empotrado no es más que aquella aplicación o software que se diseña e implementa para ejecutarse en dispositivos (generalmente microcontroladores) y que se enfrenta a fuertes restricciones de hardware.

Seguidamente se muestra un resumen de las características de los sistemas empotrados aportadas por (Berger, 2002):

- ✓ Están dedicados a tareas específicas, mientras que las computadoras constituyen plataformas genéricas.
- ✓ Compatibles con una amplia gama de procesadores y arquitecturas de procesador.
- ✓ Sensibles económicamente.
- ✓ Tienen restricciones de tiempo real.
- ✓ Las implicaciones de un fallo de software es mucho más grave en los sistemas empotrados que en los sistemas de escritorio.
- ✓ Suelen tener limitaciones en el consumo de energía.
- ✓ A menudo deben operar bajo condiciones ambientales extremas.

- ✓ Tienden a tener menos recursos del hardware que los sistemas de escritorio.
- ✓ Suelen almacenar todo su código objeto en la memoria ROM.
- ✓ Requieren para su desarrollo de herramientas y técnicas especializadas junto a métodos de diseño simples y eficientes.

1.2.1. Sistemas empotrados de tiempo real

(Laplante, 2004) define a los sistemas de tiempo real como:

> *"Sistemas que deben satisfacer explícitamente exigencias en el tiempo de respuesta a determinados eventos o corre el riesgo de sufrir graves consecuencias, incluyendo el colapso del sistema, […] son aquellos sistemas cuya exactitud lógica se basa tanto en la exactitud de los resultados como en su precisión temporal".*

Esta definición conceptualmente está relacionada con las definiciones de sistemas empotrados dadas anteriormente, dando vida a otra área del conocimiento: *los sistemas empotrados de tiempo real*, aunque es necesario acotar que no todos los sistemas empotrados exhiben comportamientos en tiempo real ni todos los sistemas en tiempo real son empotrados.

En la fig. 4 se muestran equipos y vehículos donde se emplean sistemas empotrados que poseen fuertes restricciones en cuanto a tiempo de respuesta y exactitud del resultado. Cualquier error puede acarrear grandes consecuencias en: control de aviones teledirigidos, máquina de hemodiálisis, sistemas de freno ABS, desfibriladores cardiacos. De la exactitud de estos sistemas depende la vida de muchas personas en todo el mundo.

Fig. 4 Aplicaciones de los sistemas empotrados

1.3. Sistemas operativos de tiempo real

Los sistemas operativos son entornos de software que funcionan como intermediarios entre el usuario y las interfaces de bajo nivel del hardware. A la definición anterior basada en la enunciada por (Heath, 2003), se le podría añadir que los sistemas operativos proporcionan un interfaz constante y un conjunto de utilidades para permitir a los usuarios utilizar el sistema de forma rápida y eficiente.

Una de las funciones principales de un sistema operativo es asignar recursos de hardware a las diversas tareas que se deben realizar. A las tareas se le asignan prioridades para que un planificador disponga de su ejecución de acuerdo a un esquema. Estos elementos están presente tanto en los sistemas operativos multitareas y multiusuario (Linux por ejemplo), como en los Sistemas Operativos de Tiempo Real (RTOS, siglas en ingles de *Real Time Operating System*).

Algunas de las particularidades de los RTOS radican en la planificación de estas tareas. El proceso es más complejo en dichos sistemas debido al hecho de que las tareas son críticas en función del tiempo y unas tienen mayor prioridad que otras. Otra característica que resalta en los RTOS es la asignación de tiempo de CPU (periodo de tiempo en el cual una tarea hace uso del CPU). Un RTOS debe ser capaz de responder a eventos externos de forma determinista y en intervalos de tiempo prefijados, minimizando otras funciones. En un RTOS las tareas críticas deben recibir los recursos que necesiten y cuando lo necesiten, sin importar el impacto que esto produzca.

Los RTOS se caracterizan por presentar requisitos especiales descritos por (Stallings, 2006) en aspectos como:

- ✓ **Determinismo:** realiza las operaciones en instantes fijos o en intervalos de tiempo predeterminados. Estos intervalos pueden estar desde microsegundos a pocos milisegundos.
- ✓ **Sensibilidad:** respuesta rápida en la gestión de interrupciones (ISR, siglas en inglés de *Interrupt Service Routine*).
- ✓ **Fiabilidad:** controlan procesos que ocurren en su entorno, las pérdidas o degradaciones en el sistema puede tener desenlaces catastróficos.
- ✓ **Tolerancia a fallos:** deben diseñarse para responder a cualquier tipo de fallo que ocurra dentro del entorno del sistema, siempre optando por corregir el problema y minimizar su impacto que detener la ejecución del sistema. Los sistemas tienen que garantizar ser estables.

1.3.1. Sistemas Operativos Empotrados de Tiempo Real

Para la mayoría de los sistemas empotrados se requiere de un sistema operativo que pueda ejecutar varias aplicaciones simultáneamente permitiendo el control y la comunicación entre las tareas. Entre sus características podemos encontrar (Barr, y otros, 2006):

- ✓ **Soporte de ejecución basado en tareas:** Las tareas son como pequeños programas con su propio contexto de ejecución.
- ✓ **Planificación por prioridad:** Garantía de que una tarea que esté lista para ejecutarse se ejecute por encima de una tarea con menor prioridad.
- ✓ **Sincronización entre tareas:** Uso de semáforos binarios y de exclusión mutua con el objetivo de asegurar que las tareas se ejecuten como entidades independientes entre ellas.
- ✓ **Comunicación entre tareas:** Implementación de mecanismos de comunicación entre tareas como semáforos y colas de mensajes (o buzón).

1.3.2. Estado del Arte de los Sistemas Operativos Empotrados de Tiempo Real

En la actualidad en el mundo existen cientos de sistemas operativos empotrados de tiempo real. La siguiente reseña los clasifica en privativos o de código abierto a través de la licencia con que son distribuidos. Se especifica, además, la empresas que lo desarrolla y si son gratis o comerciales.

AMX (Privativo con ciertas libertades, comercial, KADAK Products Ltd.)

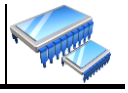
El sistema AMX fue lanzado por primera vez en 1980. Los años de desarrollo continuo le han servido para ser ampliamente reconocido como un RTOS de gama alta.

AMX está disponible para distintas arquitecturas como son: Intel x86, ColdFire, PowerPC y ARM, entre otras. Entre las características que lo diferencian de otros RTOS está el soporte que brinda la compañía, apoyando a los desarrolladores con herramientas de desarrollo específicas para cada plataforma.

Según la página oficial de la compañía KADAK, AMX se ofrece bajo un contrato de licencia de sitio. Este tipo de licencia le permite al usuario “adquirir software para varias ubicaciones con base en un acuerdo global con el proveedor”, (Dell Inc.). La empresa incluye el código fuente, sólo que prohíbe la distribución a terceros del mismo y del código objeto, bibliotecas de objetos o documentación de cualquier software de KADAK. Otra elemento positivo es que provee actualizaciones automáticas durante el periodo de garantía así como el soporte técnico sin cargo adicional.

FreeRTOS (GPLv2, gratis, Real Time Engineers Ltd.)

Algunos especialistas no consideran a FreeRTOS como un RTOS en sí, sino como un micro-núcleo de



tiempo real. Aunque cumple con muchas de las características de un RTOS, no es calificado como tal debido a su sencillez, lo que lejos de ser una desventaja frente a otros RTOS, le aporta escalabilidad convirtiéndolo en la opción idónea para pequeños sistemas empotrados.

Según datos de su página oficial "es líder en el mercado de los sistemas operativos de tiempo real, soporta 31 arquitecturas y recibe 77500 descargas al año".

Diseñado para ser pequeño, simple y fácil de usar. Típicamente el consumo de memoria del núcleo está en el orden de los 4 a 9 Kbytes. Cuenta con una estructura en el código muy portable y predominantemente escrito en C. En la página oficial también se ofrecen herramientas gratuitas de desarrollo para muchas de las arquitecturas soportadas.

Paralelamente a FreeRTOS, Real Time Engineers Ltd. ofrece OpenRTOS y SafeRTOS. Estas son versiones comerciales de FreeRTOS con las cuales la empresa brinda soporte a los clientes, mientras que FreeRTOS cuenta con un foro de asistencia gratis por parte de la comunidad de desarrollo y soporte comercial opcional por parte de la empresa.

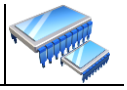
eCos (GPLv2, gratis, eCosCentric Limited)

eCos es un sistema operativo de tiempo real, de código abierto y orientado a aplicaciones empotradas. Los desarrolladores tienen acceso completo y sin restricciones a todos los módulos del sistema operativo incluyendo derecho al libre desarrollo y distribución de aplicaciones basadas en eCos.
Según la página oficial del sistema operativo, su característica principal es que es altamente configurable. Permite personalizar el sistema operativo para ajustarse a los requerimientos de las aplicaciones, brindando buen rendimiento de hardware y un consumo de memoria ajustable.

eCos cuenta con una creciente comunidad de desarrollo que garantiza el soporte para distintas arquitecturas, 13 en total, entre las que se encuentran: ARM, Intel x86, PowerPC, SPARC.

Este sistema proporciona funcionalidades básicas que lo diferencian de los mencionados anteriormente como son:

- ✓ Capa de abstracción de hardware (HAL).
- ✓ Manejo de interrupciones.
- ✓ Manejo de excepciones.
- ✓ Amplio conjunto de primitivas de sincronización.
- ✓ Temporizadores, contadores y alarmas.
- ✓ ISO C, librerías matemáticas y C++ Standard Template Library (STL).



- ✓ Soporte a las interfaces y periféricos: Serie, Ethernet, SPI, I^2C, CAN, USB, ADC.
- ✓ Pilas TCP / IP en red.

1.4. Adquisición de datos

Tradicionalmente en los sistemas de control se han utilizado los dispositivos de medición (sensores, transductores, etc.) como punto de referencia para las acciones de control. La tarea de registrar las mediciones y procesar los datos recogidos para la visualización se ha convertido en fuente de ocupación para ingenieros. Desde hace varios años el uso de tarjetas de adquisición de datos se ha transformado en la vía más efectiva de medir las señales y transferir los datos hacia las computadoras.

La adquisición de datos es el proceso mediante el cual los fenómenos físicos del mundo real se transforman en señales eléctricas que se miden y se convierten a un formato digital para su procesamiento, análisis y almacenamiento por una computadora, (Data-Acquisition, 2011). En el entorno de la automatización industrial el proceso de adquisición de datos se encarga de recolectar la información asociada a las variables del proceso, entiéndase por variables los valores relacionados con magnitudes, estados que definen el comportamiento de un sistema (ej. velocidad, temperatura, encendido, etc.).

En la gran mayoría de las aplicaciones, la adquisición de datos (DAQ, siglas en inglés de *Data AcQuisition*) se ha diseñado no sólo para adquirir datos, sino también para actuar, por lo que es conveniente ampliar la definición enunciada anteriormente para incluir los aspectos de *control*.

Mientras, el control es el proceso mediante el cual las señales digitales de control, provenientes de sistemas de hardware, son convertidas a un formato de señal para su uso en dispositivos de control, (Data-Acquisition, 2011), tales como actuadores y relés. Entonces, cuando un sistema se define como sistema de adquisición de datos o sistema DAQ, es posible que incluya funciones de control.

1.4.1. Hardware para la adquisición de datos

El hardware para la adquisición de datos puede ser definido como el componente de un sistema completo de adquisición de datos y control, que realiza cualquiera de las siguientes funciones, (Data-Acquisition, 2011):

- ✓ La entrada, el procesamiento y la conversión a formato digital (utilizando ADC) de señales analógicas (datos de las mediciones) a partir de un sistema o proceso. Los datos se transfieren a otro dispositivo que permita su visualización, almacenamiento y análisis.
- ✓ La entrada de señales digitales, que contienen información de un sistema o proceso.
- ✓ El tratamiento, la conversión a formato analógico (usando DAC) de las señales digitales provenientes

de algún dispositivo. Las señales de control analógicas se utilizan para controlar algún sistema o proceso.

- ✓ La salida de señales de control digitales.

Como hardware para adquisición de datos podemos encontrar muchos dispositivos totalmente diferentes en el mercado mundial. Entre las más populares, y relacionadas con este trabajo, están las tarjetas conectables al bus de expansión, que se acoplan directamente en dicho bus en una computadora, comúnmente nombradas tarjetas de adquisición de datos.

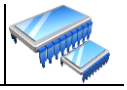
Fig. 5 Tarjeta de adquisición de 12 bit con 8 entradas análogas, 2 salidas análogas y 24 E/S digitales.

1.5. Firmware

Llamado como "el nivel más bajo de las aplicaciones empotradas", los conceptos de *firmware* son breves y pocas veces abarcan toda la dimensión de lo que realmente significa *firmware*. Seguidamente se expone el concepto incluido en el Diccionario FOLDOC de Computación: "Software guardado en memoria ROM o ROM programable". Ciertamente la palabra surge en 1968 por la unión del adjetivo "firm" (firme, rígido, estable) y la terminación de software "ware", (Harper, 2010).

El *firmware* tiene la responsabilidad de "la conducta de un sistema cuando se enciende por primera vez", (Imperial College Department of Computing, 2010). Concretamente el *firmware* es el software que trabaja directamente con hardware y se utiliza como plataforma, interfaz para que los sistemas empotrados hagan uso de los recursos de hardware.

El desarrollo de *firmware* es un proceso un tanto diferente al desarrollo de aplicaciones para computadoras personales. El código debe de ser minimalista, eficiente, en tiempo real, estable, fácil de leer, etc. Estas características están estrechamente relacionadas con las particularidades de los sistemas empotrados enunciadas anteriormente.



1.6. Conclusiones del capítulo

En este capítulo se analizaron los conceptos asociados al problema, detallando de cada uno de ellos definiciones, características y en algunos casos el estado del arte. La información abordada facilita la comprensión de la problemática y aporta los siguientes elementos:

- ✓ Los microcontroladores constituyen herramientas versátiles que contienen periféricos que resultan ventajosos para el desarrollo de tarjetas de adquisición de datos.
- ✓ Los sistemas empotrados están sometidos a fuertes restricciones de hardware; el sistema a desarrollar debe de estar orientado a ejecutarse bajo esas condiciones.
- ✓ A partir del estado del arte expuesto acerca de los Sistemas Operativos Empotrados de Tiempo Real, se decidió emplear en la solución a **FreeRTOS**. La decisión se basa principalmente en que es de código abierto y por los escasos recursos de hardware que requiere, esto unido a que, como se mencionó anteriormente, es pequeño, simple y fácil de usar.

Capítulo 2

MÉTODOS Y HERRAMIENTAS

En este capítulo se examinan las principales técnicas, métodos y herramientas asociadas al tema en cuestión, con el objetivo de obtener información que permita establecer el marco de trabajo favorable para el desarrollo del *firmware*. Es por ello que, a través de la investigación realizada, en esta sección se persigue fundamentar la selección de métodos, técnicas y herramientas a partir de otros estudios relacionados con la materia que se indaga.

2.1. Metodologías de desarrollo

La (Real Academia Española) define la palabra *metodología* como: "*Conjunto de métodos que se siguen en una investigación científica (…)*", y es que en el campo del software, una metodología de desarrollo no es más que un proceso que tiene como fin organizar exitosamente la producción del software a través de, precisamente, conjuntos de métodos, técnicas y herramientas. A continuación se enuncia una definición formulada por (Charvat, 2003):

> *"Una metodología es un conjunto de directrices o principios que pueden ser adaptados y aplicados a una situación específica"*.

En la actualidad catalogar o comparar a las metodologías de desarrollo de software es una tarea bien compleja, pues existe un sinfín de ellas que abarcan los más diversos enfoques. Igualmente existen muchos factores en los que incide la selección de cierta metodología: tamaño del equipo de desarrollo, tiempo disponible para el desarrollo, complejidad del software, implicación del cliente, etc.

Para iniciar el análisis de las metodologías asociadas al desarrollo de *firmware* es necesario describir primeramente las características del proceso de desarrollo de *firmware*. Las mismas fueron esbozadas por (Punkka, 2005):

Tabla 2 Características del proceso de desarrollo de *firmware*

Característica	Descripción
Equipos pequeños y múltiples roles	Son comunes los proyectos unipersonales. La misma persona puede ser responsable del diseño e implementación del software asumiendo múltiples roles.
Co-diseño	Fuerte dependencia entre el software, el hardware y los mecanismos de diseño. El hardware se diseña para que realice una función específica y el software se adapta a los requerimientos de hardware.
Restricciones de recursos	Frecuentemente existen limitaciones de recursos económicos que restringen el costo. Esto significa que los microcontroladores para los que se desarrolla suelen ser de gama baja con limitaciones de hardware.
Exactitud y robustez	Es necesario para el sistema ejecutarse de forma fiable y determinista, incluso en situaciones inesperadas y de manera ininterrumpida.

2.1.1. Métodos tradicionales

A continuación se analizarán dos metodologías de desarrollo. La primera de ellas está asociada directamente al desarrollo de sistemas empotrados y la segunda es – quizás –, una de las metodologías que existan más completas y abarcadoras.

ROPES (Rapid Object-Oriented Process for Embedded Systems)

La metodología ROPES define un proceso para el desarrollo de sistemas empotrados partiendo de los aspectos generales del desarrollo de software. Como proceso en sí contempla la Ingeniería de sistemas, la Integración de sistemas y las actividades de prueba. ROPES puede ser aplicado tanto en equipos pequeños (una a tres personas) como a grandes proyectos, conformados por equipos de trabajo con cientos de miembros, (Douglass, 1999). Es un proceso escalable logrando un equilibrio entre las llamadas metodologías pesadas y las metodologías ágiles o livianas. En la fig. 6 se muestra el modelo en espiral que propone ROPES.

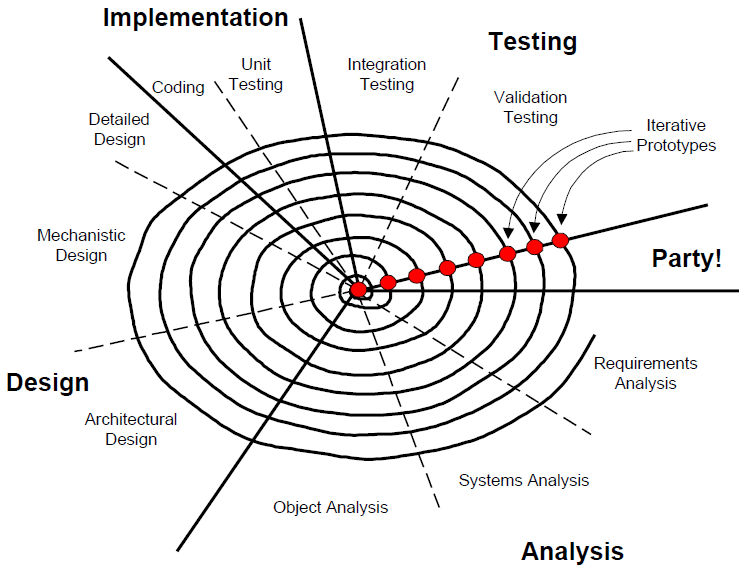


Fig. 6 Ciclo de desarrollo de ROPES

El proceso puede ser conceptualizado en tres plazos de tiempo o escalas diferentes: macro-, micro- y nano-ciclo. El macro-ciclo se produce a lo largo de meses o quizás años y constituye el ciclo rector del desarrollo en general. Tiene cuatro fases primarias, que se superponen, las que tienen como concepto clave centrarse en el perfeccionamiento de los conceptos, en el diseño e implementación y enfocadas en la optimización y el despliegue. El micro-ciclo se contempla de cuatro a seis semanas. El resultado de

cada micro-ciclo es la producción de un prototipo iterativo. El nano-ciclo es la más limitada y por lo general dura de 30 minutos a un solo día de trabajo.

ROPES plantea como punto de partida la planificación inicial del proyecto, así como la reorientación o replanificación del proyecto en caso de ser necesario. Otro elemento importante en el proceso es que la subfase de Ingeniería del sistema, que se incluye en la fase de Análisis, está dirigida a ayudar en los proyectos donde se requiere co-diseño de software y hardware. La fase de requisitos trata de acaparar la definición de los requisitos generales del proyecto durante la primera iteración, aunque deja espacio para que los requisitos se desarrollen durante todo el proyecto. El análisis de requisitos se centra en el uso de casos de uso y todo el proceso se enfatiza en el uso de UML (siglas en inglés de *Unified Model Language*) como lenguaje de modelado.

En el libro *"ROPES: Rapid Object-Oriented Process for Embedded Systems"* el autor, y creador de la metodología Bruce P. Douglass, presenta e ilustra una larga lista de artefactos que se producen en cada fase de la espiral. Igualmente describe en detalle los métodos para distribuir los flujos del proceso y referencia el uso de patrones de diseño en la fase de diseño. Como se mencionó anteriormente, el proceso en general puede ser escalado de acuerdo a la composición del equipo de trabajo y esto se realiza mediante la limitación de los artefactos producidos, pero en el libro no se describe en detalle cómo realizarlo.

RUP (Rational Unified Process)

RUP es un marco de trabajo desarrollado por *Rational Software Corporation*. Los tres pilares de RUP consisten en que es iterativo e incremental, centrado en la arquitectura y dirigido por casos de usos. El proceso de desarrollo se divide en cuatro fases: Concepción, Elaboración, Construcción y Transición. (Ver fig. 7).

RUP como metodología describe los elementos claves del proceso de desarrollo: roles, actividades, artefactos, flujos de trabajo, disciplinas y las relaciones entre éstos. La metodología básica contempla bastante información recogida en una gran cantidad de artefactos. Por otro lado el marco se sustenta en un poderoso conjunto de herramientas CASE (siglas de *Computer Aided Software Engineering*, en español Ingeniería de Software Asistida por Computadora): el Rational. La herramienta puede ser personalizada y configurada mediante el uso de plugins. Estos plugins pueden aportar nuevos elementos, artefactos, actividades y redefinición de roles a RUP en dependencia de las necesidades de cada proyecto. Uno de los plugins que puede estar asociado al desarrollo de sistemas empotrados es el RUP SE (siglas en inglés de *Rational Unified Process to Engineering System*).

Según (Cantor, 2003), RUP SE se aplica a los proyectos de software que:

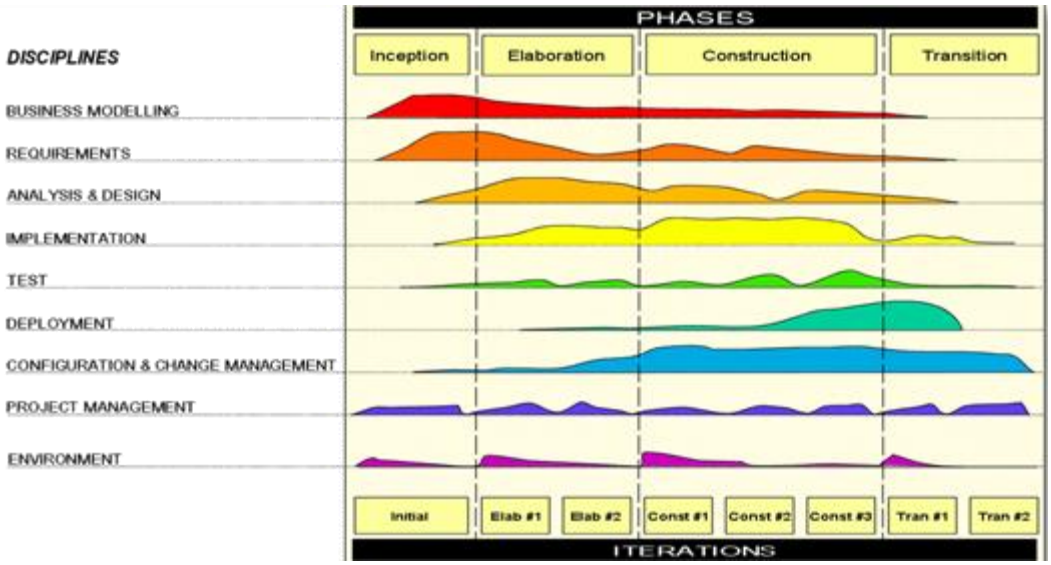


Fig. 7 Proceso de desarrollo de RUP

- ✓ Son suficientemente grandes como para demandar varios equipos de trabajo desarrollando simultáneamente en cada uno de ellos.
- ✓ Implementan de manera concurrente componentes de hardware y el software.

2.1.2. Métodos ágiles

Los métodos ágiles ganaron mucha atención en la comunidad de software durante los últimos años de la década de los '90, pues surgían como resistencia ante la excesiva documentación de las metodologías tradicionales. Varios especialistas en metodologías de desarrollo, todos defensores de los modelos ágiles se reunieron en el 2001 y surge la Alianza Ágil. Sus puntos de vistas fueron compilados en el Manifiesto Ágil. Entre varios principios (hasta llegar a 12) el Manifiesto Ágil (Alianza Ágil, 2001) plantea:

- ✓ Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software valioso.
- ✓ Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- ✓ Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
- ✓ El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- ✓ La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, son esenciales.
- ✓ Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.

En las secciones siguientes se irán analizando las metodologías ágiles que más han llamado la atención de los programadores en el mundo. Destacar que las metodologías que se reseñarán a continuación

parten básicamente de los principios del Manifiesto Ágil. Al finalizar este apartado como conclusión se analizarán la vinculación de las metodologías ágiles con el desarrollo de *firmware*.

XP (eXtreme Programming)

De todas las metodologías ágiles, ésta es la que ha recibido mayor atención, lo que ha provocado que XP sea una de las metodologías más documentadas hoy en día, (Punkka, 2005).

El modelo de desarrollo propuesto por XP, se basa en el desarrollo iterativo e incremental, donde los requerimientos, el diseño y el código fuente evolucionan continuamente a través de las iteraciones.

Como metodología ágil, XP acepta los riesgos minimizando el costo ante los cambios de los requerimientos. La metodología en sí se fundamenta en cuatro valores: Comunicación, Retroalimentación, Simplicidad y Coraje, (Beck, 2000). Sobre estos 4 elementos se construye una docena de prácticas para el desarrollo de software en los proyectos que apliquen XP. Estas prácticas consisten en técnicas antiguas, conocidas y probadas como revisiones de código, la atención a las pruebas, el fuerte diseño de la arquitectura, etc., todas con un nuevo enfoque. Igualmente las prácticas antes mencionadas convierten a XP en una buena opción para los equipos de desarrollo pequeños (hasta diez y 15 personas) aunque la mayoría de las prácticas de XP también son apropiadas para los programadores individuales. La metodología propone 40 horas de desarrollo semanal sostenible.

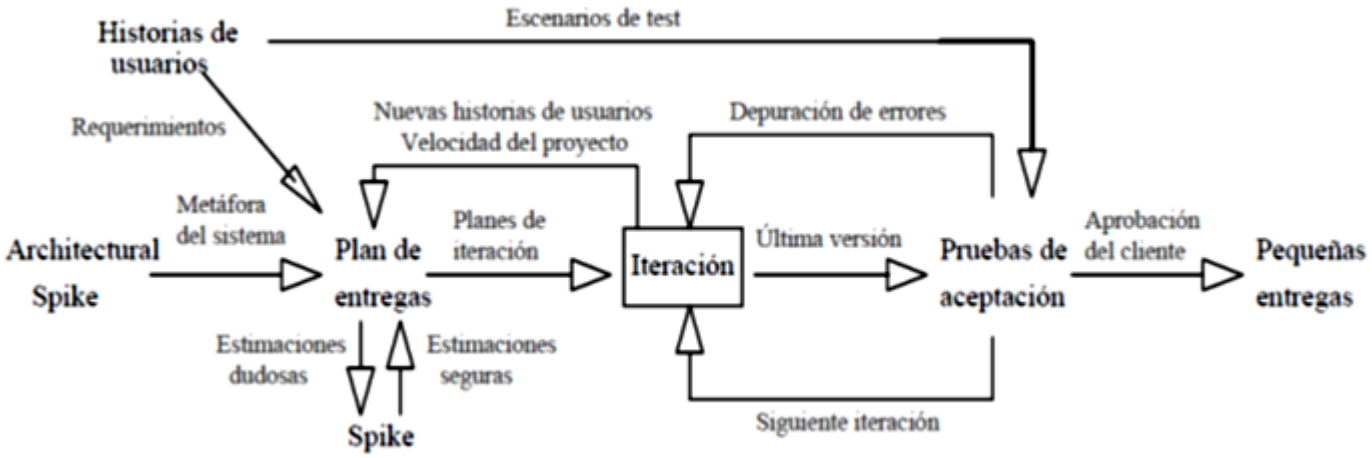


Fig. 8 Ciclo de desarrollo de XP

Fig. 9 Actividades propuestas por XP en cada una de las fases del desarrollo

SCRUM

Scrum se refiere a la gestión del desarrollo de software desde un punto de vista adaptable, enfocándose principalmente en la planeación iterativa y el seguimiento de los procesos, (Fowler, 2003). No contempla una práctica de desarrollo de software en particular sino que funciona estableciendo un marco de trabajo. Se centra más en los cambios de los requisitos que en una planificación general del proyecto, lo que le permite reaccionar favorablemente ante algún cambio, teniendo como premisa que en un comienzo el cliente no sabe lo que quiere.

El papel de la gestión de requisitos es realizado a través de la Pila del Producto donde se inserta cada funcionalidad o requisito funcional que el sistema debe implementar. Luego, como una pila en sí, cada funcionalidad es extraída de la Pila del Producto y se le dedica una iteración.

En Scrum cada iteración es llamada Sprint, la cual tiene una duración de 30 días. Cada Sprint comienza con una reunión de planificación donde se define la funcionalidad requerida junto a otros parámetros. A partir de esa reunión los desarrolladores tienen la responsabilidad de concluir la funcionalidad en el plazo establecido. Diariamente el equipo sostiene una reunión corta (quince minutos), llamada scrum, donde el equipo discute lo que hará al día siguiente.

En muchos libros podemos encontrar casos de éxitos donde se combinan XP y Scrum (ej. "*Scrum y XP desde las trincheras*").

Familia de metodologías Crystal

Alistair Cockburn propone una familia de metodologías a partir de la idea de que diferentes proyectos necesitan diferentes metodologías, (Cockburn, 2000), esto es una muestra de cuan escalable puede ser. Fundamentalmente las cataloga por dos aspectos: la cantidad de miembros del proyecto y la criticidad basada en la prioridad para la productividad, la responsabilidad legal, el costo, la previsibilidad y la agilidad, (Punkka, 2005). Representa esa variación a través de un gráfico con dos ejes: el número de personas en el proyecto y criticidad. Cada metodología está representada por un recuadro dentro de la gráfica.

Cockburn describe a la familia Crystal como "centrada en las personas y la comunicación entre ellas", (Cockburn, 2000). En ese aspecto coincide con la metodología XP, solo que el enfoque es abierto. Puede ser altamente disciplinada (lo más cercano a XP) o aprovechando la facilidad de ejecución (tomando elementos de Proceso de Software Personal, PSP). Cockburn considera que las personas encuentran difícil seguir un proceso tan disciplinado como XP, pero da la posibilidad de alternar entre un método u otro, (Fowler, 2003).

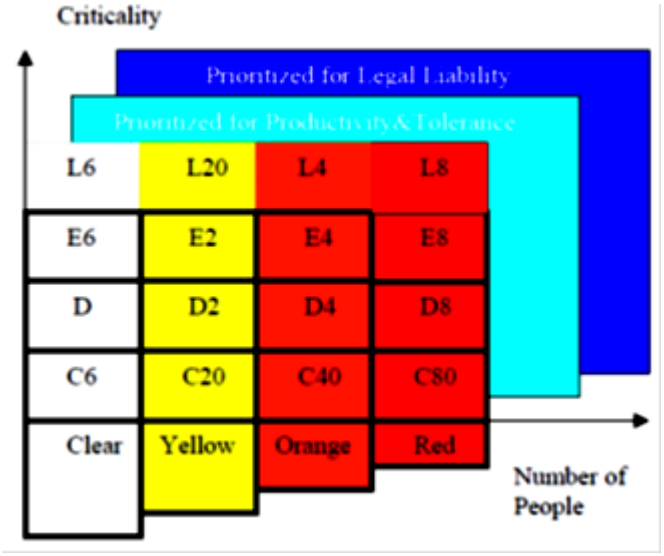


Fig. 10 Representación de la familia de metodologías Crystal

Otro elemento particular de las metodologías Crystal es que hace énfasis en la revisión rigurosa al final de cada iteración. Propone realizar talleres que permitan aprovechar al máximo las experiencias de cada iteración para poco a poco ir refinando los procesos a medida que avanza el proyecto.

2.1.3. Resumen de metodologías

En los apartados anteriores fueron repasadas un conjunto de metodologías con la idea de mostrar las que están posiblemente asociadas al desarrollo de *firmware* y sistemas empotrados. A continuación se muestra una tabla resumen.

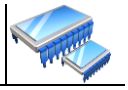


Tabla 3 Resumen de las metodologías ágiles

Metodología	Referencias	Alcance	Elementos particulares	Limitantes
ROPES	Poca	Desde el marco de trabajo a las prácticas de ingeniería	Fase "Party" dentro del modelo en espiral.	Falta documentar adaptación a proyectos pequeños
RUP SE	Bastante	Marco para el desarrollo de software y la gestión de proyectos	Soporte de herramientas CASE. Énfasis en el modelado	Prevista para grandes proyectos con múltiples equipos
XP	Bastante	Prácticas de programación	Programación en parejas	Alcance limitado. Carga de trabajo extrema
SCRUM	Bastante	Marco para el desarrollo de software y la gestión de proyectos	Sprint mensual y scrum diario	No dispone de prácticas ingenieriles
Familia Crystal	Poca	Escalable de acuerdo al proyecto	Colección de acuerdo a tamaño del proyecto	Poca documentación

En realidad la adopción de una metodología de desarrollo de software es un hecho bien ajustable a las características del proyecto objetivo. Comúnmente no se suele encontrar un proyecto con una metodología pura, sino que estas se combinan y se ajustan a las necesidades del proyecto. Se suelen combinar en dependencia del alcance de cada una.

2.1.4. Metodologías y desarrollo de *firmware*

El desarrollo de *firmware* no escapa a los retos que se le imponen a los otros productos de software, aunque debido a sus características mencionadas con anterioridad, sí suelen presentarse problemas propios de este sector. Estos inconvenientes están más asociados a la inestabilidad de los requerimientos y a la necesidad de una rápida salida del producto al mercado.

(Punkka, 2005) hace un detallado análisis de la aplicación de metodologías ágiles al desarrollo de *firmware*, y demuestra por que las metodologías ágiles son más aplicables al desarrollo de *firmware* que cualquier otro tipo de metodología. A continuación se muestra una tabla extraída de dicho artículo, donde se presentan las características de los proyectos según las metodologías que se apliquen y en la última columna se muestra cómo se comportan estas particularidades en los proyectos de desarrollo de *firmware*.

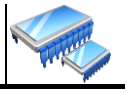


Tabla 4 Comparación de entornos de desarrollo

Características del proyecto		Entornos ágiles	Entornos basados en planificación	Entorno del firmware
APLICACIÓN	**Objetivos primarios**	Estimación rápida, respondiendo a los cambios.	Gran previsibilidad, garantía y gran estabilidad.	Los proyectos están desarrollando algo nuevo, la retroalimentación rápida del cliente es esencial.
	Tamaño	Equipos pequeños con proyectos pequeños	Equipos grandes con grandes proyectos	Son comunes los proyectos desarrollados por una sola persona.
	Entorno	Agitado, muchos cambios, enfocados en el proyecto	Estable, pocos cambios, enfocados en la planificación y la organización	Los proyectos están desarrollando algo nuevo, probabilidad alta de que los requisitos cambien, es común que apliquen nuevas tecnologías, la evolución es basada en la experiencia.
TÉCNICA	**Requerimientos**	Historias priorizadas informales y casos de prueba, experimenta cambios imprevistos.	Proyecto formalizado. La capacidad, la interfaz, la calidad y las necesidades son previsibles en la evolución del proyecto.	Cambios imprevistos debido a los requisitos de incertidumbre y las nuevas tecnologías.
	Desarrollo	El diseño simple, incrementos a corto plazo, la refactorización resulta barata.	Diseño extenso, incrementos a largo plazo, la refactorización resulta cara.	El diseño simple es esencial. La refactorización puede ser difícil debido a los plazos de entrega.
	Pruebas	Los casos de prueba posibles definen los requisitos.	Se documentan los planes de pruebas y los procedimientos.	Las pruebas finales sólo se pueden ejecutar cuando el hardware está disponible. Las pruebas oficiales son caras.

La tabla presentada muestra como los aspectos relacionados con los proyectos de *firmware* se acercan más a las metodologías ágiles que a las tradicionales basadas en la planificación. Por último a continuación se muestra una tabla donde se presentan los principios del Manifiesto Ágil y cómo es posible la aplicación de estos a proyectos de desarrollo de *firmware*.

Tabla 5 Principios ágiles para entornos de desarrollo de *firmware*

Principio del Manifiesto Ágil	Aplicabilidad	Esfuerzo en la aplicación	Comentarios
Satisfacer al cliente mediante la entrega temprana y continua de software valioso.	Necesita adaptación	Alto	El valor de la respuesta rápida e inmediata es esencial.
Aprovechar los cambio en los requerimientos	Necesita adaptación	Medio	Algunos cambios pueden afectar el diseño de hardware en tiempo real o violar requerimientos de hardware.

Entregas en ciclos cortos	Necesita adaptación / Inaplicable	Alto	Gran parte del desarrollo del *firmware* no es visible al cliente en absoluto.
Clientes y desarrolladores trabajando juntos	Necesita adaptación	Bajo	Alta barrera cultural entre los clientes y los desarrolladores.
Desarrollar el proyecto alrededor de individuos motivados	Aplicable	Ninguno	Los métodos ágiles debe mejorar la motivación entre las personas orientadas hacia el desarrollo.
Hacer énfasis en la comunicación cara a cara	Necesita adaptación	Medio	A menudo, los equipos distribuidos.
Las entregas continuas son el mejor indicador de progreso	Necesita adaptación	Alto	El *firmware* necesita una persona técnica para evaluar el estado de funcionamiento del software.
Promover el desarrollo sostenible	Necesita adaptación	Bajo	El desarrollo del *firmware* tiene algunas fechas fijas (pruebas oficiales, ferias).
Atención continua a la excelencia técnica y al buen diseño	Aplicable	Ninguno	Pruebas exhaustivas es esencial para el *firmware* de los requisitos de robustez.
La simplicidad es esencial	Aplicable	Ninguno	La sencillez es natural que en tiempo real, porque el *firmware* de los recursos limitados.
Los mejores resultados provienen de equipos auto-organizados	Aplicable	Ninguno	Promueve el trabajo en equipo. Los desarrolladores del *firmware* propensos al heroísmo pueden carecer de estas habilidades.
Los equipos reflejan cuando y donde "mejorar"	Aplicable	Ninguno	Punto importante para las pequeñas organizaciones para la adopción de la mejora de procesos.

La tabla nos da una medida de que no es tan sencillo adaptar estos principios a los escenarios de *firmware*, es decir, no se evidencia con certeza que los métodos ágiles sean la solución para el desarrollo de *firmware*, pero sí es una realidad que las metodologías ágiles tienen más puntos en común con el entorno para *firmware* que las metodologías tradicionales. La mayoría de las prácticas ágiles pueden ser adaptadas, con menor o mayor esfuerzo.

2.2. Lenguaje Unificado de Modelado (UML)

(Rumbaugh, y otros, 1998) definen a UML como:

> *"Lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software."*

UML es una de las especificaciones más utilizadas en la industria del software y no sólo se limita a

especificar la estructura de la aplicación sino también el comportamiento, la arquitectura, los procesos del negocio y la estructura de los datos, (Rumbaugh, y otros, 1998). Este lenguaje de modelado no define un proceso estándar pero está pensado para ser útil en los procesos de desarrollo iterativo y está dirigido principalmente a dar apoyo a los procesos de desarrollo orientados a objetos aunque, como se verá más adelante, esta última característica es adaptable.

La creciente complejidad de los sistemas empotrados de tiempo real requiere de un enfoque de diseño más refinado y sofisticado para lograr una implementación exitosa. Para alcanzar ese objetivo UML se convierte en una herramienta poderosa para describir los aspectos estructurales y de comportamiento críticos de sistemas de tiempo real.

En el 2002, el *Object Management Group* (OMG, consorcio internacional que está detrás de las especificaciones UML) aprobaba el *Real-Time Profile* (RTP). Este perfil proporcionaba un medio estandarizado para las especificaciones de tiempo, los aspectos de rendimiento y la planificación de tareas para sistemas de tiempo real. En el mismo año se le agregó un complemento al RTP que detallaba un framework general para la Calidad de Servicio (QoS) referida a las especificaciones de tiempo, los aspectos de rendimiento y la planificación de tareas, así como un perfil específico para los mecanismos de tolerancia a fallos, (Douglass, 2004).

Entre los elementos que contempla RTP podemos encontrar:

- ✓ Modelado de acciones y de concurrencia:
- ✓ Modelado de recursos.
- ✓ Modelado de líneas de tiempo.
- ✓ Modelado de planificaciones de tareas.
- ✓ Modelado del rendimiento.

Todos estos aspectos permiten modelar haciendo uso de RTP, (Woodside, y otros, 2004):

- ✓ Una secuencia determinista de pasos, a través de un diagrama de secuencia o haciendo uso de un diagrama de actividades.
- ✓ Hilos asociados a procesos en un diagrama de implementación.
- ✓ Operaciones externas al sistema como los retrasos de la red, operaciones de E/S, operaciones de bases de datos, etc.

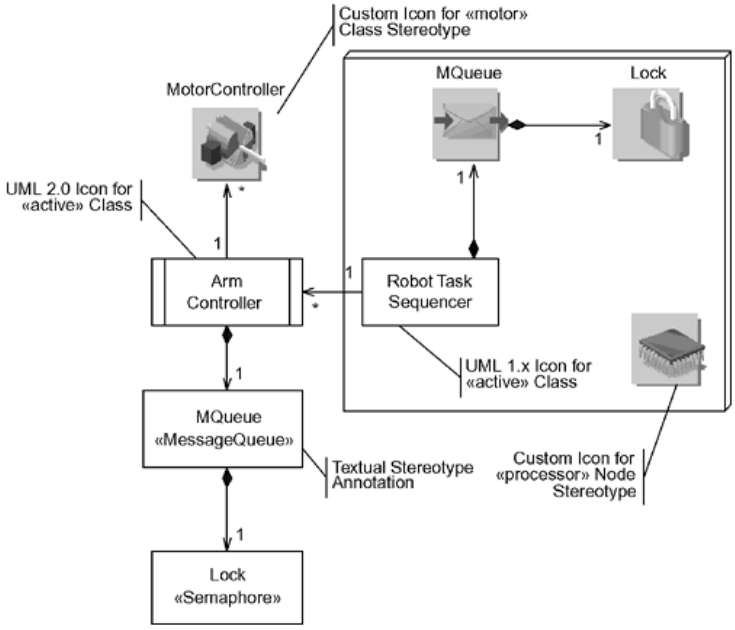


Fig. 11 Ejemplo de diagrama elaborado con UML-RT

2.2.1 UML para programación estructurada

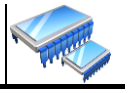
Como se planteaba anteriormente, UML fue diseñado para ser aplicado con el paradigma orientado a objetos, entonces surge la duda ¿cómo aplican UML los proyectos de sistemas empotrados cuando la mayoría de estos sistemas son implementados en C, siendo este un lenguaje estructurado?

(Douglass, 2009) describe tres estrategias para el uso de UML en el desarrollo de aplicaciones empotradas y de tiempo real:

- ✓ Modelado Orientado a Objetos.
- ✓ Modelado basado en Objetos.
- ✓ Modelado operativo.

El Modelado Orientado a Objetos utiliza UML a profundidad para representar el sistema y luego generar el código de C haciendo uso de herramientas CASE. El problema mayor llega a la hora de hacer uso del Paradigma Orientado a Objetos (POO) que formula conceptos como polimorfismo, herencia, relaciones de agregación y composición, etc. Aquí es cuando se utilizan trucos sencillos para lograr la transformación:

- ✓ Las clases se convierten en estructuras que contienen los atributos y punteros a funciones miembros.
- ✓ La generalización se implementa anidando las estructuras que representan a la clase base y la clase hija.



- ✓ El polimorfismo se genera a través de funciones virtuales con el nombre de las funciones miembro.

La otra variante el Modelado basado en Objetos es más simple aún, aunque no utiliza toda la riqueza de UML limitándose a no usar la generalización y el polimorfismo.

Por último el Modelado operativo aprovecha al máximo las ventajas de UML para representar varios aspectos de los sistemas empotrados y de tiempo real como: funcionamiento, comportamiento, interactividad y la estructura. A continuación se muestra una tabla donde se resume como hacer el mapeo entre un paradigma y otro.

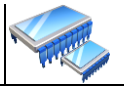
La introducción de los conceptos naturales de UML como clases, componentes, casos de uso, etc., en C ayuda a los desarrolladores con los beneficios que aporta UML como lenguaje de modelado. Esto ayuda a reducir aún más el riesgo que implica la adopción de nuevas tecnologías en el desarrollo de sistemas empotrados y de tiempo real, contribuyendo a acortar el tiempo de desarrollo.

Tabla 6 UML para la programación estructurada

Tipo de diagrama	Modelado operativo	Representación en UML	Descripción
Requerimientos	Diagrama de caso de uso	Diagrama de caso de uso	Representa el uso de un sistema y su relación con los actores
Estructura	Diagrama de construcción	Diagrama de componentes	Muestra un conjunto de artefactos construidos a partir del código fuente como bibliotecas y ejecutables
	Diagrama de fichero	Diagrama de clases	Muestra un conjunto de ficheros .c y .h y sus relaciones
Comportamiento	Diagrama de mensajes	Diagrama de secuencia	Muestra las secuencias de llamadas y eventos enviados entre un conjunto de archivos, incluidos los valores pasados por parámetros
	Diagrama de estado	Diagrama de estado	Muestra una máquina de estado para los archivos y la forma en que sus funciones se incluyen y las acciones son ejecutadas como eventos (ya sea sincrónica o asincrónica)
	Diagrama de flujo	Diagrama de actividades	Detalles del flujo de control de una función o un caso de uso

2.3. Entornos Integrados de Desarrollo

Un entorno de desarrollo integrado (IDE por sus siglas en inglés) es un programa compuesto por una serie de herramientas de programación como son: un editor de código, un compilador, un intérprete, un depurador y un constructor de interfaces gráficas de usuario, aunque en muchos casos incluyen otros componentes.



Muchos de los vendedores de microcontroladores disponen de sus propios IDE. Este es el caso de Microchip Technology, fabricante de los populares microcontroladores PIC. Esta empresa cuenta con el entorno MPLAB, el que se distribuye libremente y el multiplataforma (Windows, Mac y Linux). La desventaja de este sistema es que solo soporta el desarrollo de aplicaciones empotradas para sus microcontroladores PIC.

Igualmente existe toda una gama de IDE que no están sujetos a un fabricante en específico. Entre los más populares podemos encontrar a Keil µVision, IAR Embedded Workbench y Eclipse. A continuación se identificarán sus principales características.

IAR Embedded Workbench

Con ya casi 30 años, IAR Embedded Workbench ofrece toda una suite de herramientas para el desarrollo de aplicaciones empotradas que tengan como destino microcontroladores de 8, 16 y 32 bits. El IDE cuenta con un compilador y un depurador para C/C++, además de las características comunes que incluye un IDE. Está desarrollado para ejecutarse en Windows y puede ser adaptado a Linux con el emulador WINE.

Quizás el elemento más significativo de IAR es la integración que logra la empresa con los fabricantes de microcontroladores, colaborando estrechamente con estos con la intención de marcar una pauta en el mercado. Soporta las arquitecturas ARM, Atmel, STM, 8051, entre otras.

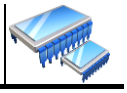
Keil µVision

µVision es una plataforma de software para Windows que combina la gestión de proyectos, la edición y depuración del código fuente, así como la simulación de entornos empotrados. Integra todas las herramientas necesarias para desarrollar aplicaciones empotradas, incluyendo un compilador de C/C++ y un generador de archivos HEX, entre otras características comunes de los IDE.

Eclipse CDT

Eclipse es un IDE multiplataforma de código abierto desarrollado por la Eclipse Foundation, el cual es extensible mediante la incorporación de *plugins*[5]. Uno de los más reconocidos y usados es el CDT. El proyecto CDT ofrece un entorno de desarrollo completo y funcional para C y C++. Incluye las siguientes características: soporte para la creación de proyectos, administración de *toolchains*[6], navegación a través del código fuente. Igualmente cuenta con editor de código con resaltado de sintaxis, hipervínculos de navegación, facilita la refactorización de código fuente, ofrece herramientas visuales de depuración

[5] [8] Véase Glosario de Términos.



permitiendo monitorizar la memoria y los registros.

Eclipse proporciona infinidad de ventajas para el desarrollo de sistemas empotrados, planteando como primer término que es posible lograr un entorno de desarrollo altamente funcional solo haciendo uso de tecnologías libres. (Abbot, 2009) presenta todo un conjunto de herramientas útiles para el desarrollo de aplicaciones empotradas, entre las más necesarias podemos encontrar:

- ✓ Uso de GNU GCC como compilador.
- ✓ Uso de GNU GDB como *debugger* lo que incluye la depuración de programas multi-hilos con visor de interrupciones o señales.
- ✓ Incorporar Makefiles definidos por los desarrolladores.

2.4. Lenguajes de programación

Durante los años, han surgido muchos lenguajes de programación desarrollados para aplicaciones empotradas y de tiempo real. Para la selección de estos lenguajes existe un conjunto de criterios que los diferencian entre unos y otros, estos son solo algunos, (Williams, 2006):

- ✓ Legibilidad.
- ✓ Flexibilidad.
- ✓ Portabilidad.
- ✓ Comprensibilidad
- ✓ Confiabilidad.
- ✓ Madurez.
- ✓ Eficiencia.

Basado en esos parámetros existen tres lenguajes que se han sido la elección de la mayoría de los desarrolladores de sistemas empotrados y de tiempo real. A continuación se describen sus características y usos.

2.4.1 Ensamblador

El lenguaje ensamblador es un lenguaje de bajo nivel en el que cada enunciado produce precisamente una instrucción en lenguaje máquina. Este lenguaje es específico para cada arquitectura, por lo que para dominar el lenguaje ensamblador, se requiere comprender muchos más elementos que para dominar un lenguaje de programación de alto nivel. Las complejidades de lenguaje ensamblador no son tanto en el lenguaje en sí, sino en el conocimiento requerido de la plataforma subyacente del microcontrolador, es decir, cada plataforma de destino tiene su propio lenguaje ensamblador.

El lenguaje en general sobresale por dos aspectos: la velocidad y la eficiencia en el tamaño del código. Lo

antes mencionado unido a que constituye la base de muchos lenguajes de alto nivel, lo convierten en una herramienta poderosa para el desarrollo de sistemas empotrados. Otro elemento a favor del lenguaje ensamblador es que brinda acceso directo al hardware mientras que otros lenguajes de alto nivel requieren precisamente traducir sus instrucciones a este lenguaje para operar con el hardware.

Como aspectos negativos podemos mencionar la alta complejidad, lo que hace lento el proceso de codificación. Los ficheros fuentes en Ensamblador suelen ser bastante grandes lo que dificulta el mantenimiento del código e impacta nuevamente en la productividad del equipo de desarrollo.

La falta de portabilidad es otro asunto a considerar. El conjunto de instrucciones del lenguaje varía entre las distintas arquitecturas de microprocesadores por lo que se hace casi imposible ejecutar el código programado para una plataforma en otra distinta.

2.4.2 ADA

Durante la década de 1970, el Departamento de Defensa de EE.UU. convocó a una prolongada competencia internacional para seleccionar un lenguaje de alto nivel que cubriera sus necesidades crecientes en el campo de los sistemas empotrados y de tiempo real. El resultado final fue ADA.

Este es un lenguaje extenso y estáticamente tipado que soporta programación orientada a objetos. Aunque es muy similar a Pascal sintácticamente, tiene muchas más comodidades pues cuenta con las siguientes características:

- ✓ **Paquetes:**

  Los tipos de datos, objetos de datos y las especificaciones del procedimiento se puede encapsular en un paquete. Esto apoya el diseño del programa y la abstracción de datos.

- ✓ **Manejo de excepciones:**

  Tiene la capacidad de manejo de excepciones que permite al programa operar con sus propios errores de ejecución.

- ✓ **Procesamiento paralelo y concurrente:**

  Soporta la ejecución paralela y concurrente de tareas.

Adicionalmente cuenta con primitivas intrínsecas multitareas y de sincronización, las llamadas rendezvous. Ada debido al alto costo de los compiladores, no ha sido muy adoptado por el sector no militar, aunque aparece en el lugar 16 del ranking de los lenguajes de programación (mes de mayo, 2012) más utilizados según TIOBE Software BV, (TIOBE Software BV, 2012).

2.4.3 C

En 1978, Dennis Ritchie y Brian Kernighan presentaron el libro "*El lenguaje de programación C*" en el cual describían al lenguaje con la siguiente frase:

> *"C es un lenguaje de programación de propósito general que ofrece la economía de expresión, el flujo de control moderno, estructuras de datos y un rico conjunto de operadores. C no es un lenguaje de muy alto nivel, ni uno grande, y no está especializado en cualquier área particular. Sin embargo, la ausencia de restricciones y su generalidad que lo hacen más conveniente y eficaz para muchas tareas que supuestamente otros lenguajes poderosos"*, *(Kernighan, y otros, 1991)*.

C carece de muchas características que poseen los lenguajes de programación modernos. No es orientado a objetos, no cuenta con un apoyo firme en la tipificación y asignación de tareas, no soporta el manejo de excepciones, la protección de punteros, la comprobación de los límites en las variables y los tipos de matriz dinámica de cadenas, (Zurrell, 2000). A pesar de sus limitaciones, "C se ha convertido en el lenguaje de los programadores de sistemas empotrados", (Barr, y otros, 2006).

El lenguaje de programación C tiene una larga lista de ventajas. Solo se mencionarán algunas que lo relacionan directamente con los sistemas empotrados, (Barr, y otros, 2006):

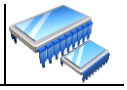
- ✓ Es pequeño y bastante sencillo de aprender, los compiladores están disponibles para casi todos los procesadores en uso hoy en día, lo que garantiza portabilidad.
- ✓ Tiene la ventaja de ser independiente del procesador, lo que permite a los programadores concentrarse en algoritmos y aplicaciones en lugar de los detalles particulares de la arquitectura del procesador.
- ✓ Posibilita la producción de código compacto, eficaz para casi todos los procesadores.

Tal vez la mayor fortaleza de C, y lo que lo diferencia de lenguajes como Pascal y FORTRAN, radica en que es "un lenguaje de muy bajo nivel" de alto nivel. C proporciona a los programadores de sistemas empotrados un grado extraordinario de control directo de hardware sin sacrificar los beneficios de un lenguaje de alto nivel.

2.5. Conclusiones del capítulo

El componente de la investigación que se detalla en este capítulo posibilitó la selección de métodos, técnicas y herramientas que permitirán lograr un ambiente estable y satisfactorio para la implementación del sistema. A continuación se detalla la elección:

- ✓ Se decidió que la metodología **XP** guiara el desarrollo del *firmware.* Los elementos expuestos en el



presente capítulo, demuestran la relación existente entre las características de los entornos de desarrollo de *firmware* y los principios ágiles de **XP**.

- ✓ Se dispuso emplear el enfoque del Modelado Basado en Objetos de **UML** para representar el sistema y generar los principales artefactos que ayuden en la especificación de la arquitectura y el comportamiento del sistema.
- ✓ En cuanto a lenguaje de programación la decisión fue por **C** por su portabilidad y sus características de lenguaje de bajo nivel, además de ser el lenguaje de preferencia universal para el desarrollo de sistemas empotrados.
- ✓ Como IDE se seleccionó **Eclipse CDT** que, junto a un conjunto de herramientas, conforman un entorno de desarrollo basado en tecnologías libres y/o de código abierto.

Capítulo 3

DESCRIPCIÓN DE LA SOLUCIÓN

En el presente capítulo se muestra una visión teórica del sistema diseñado. En el mismo se hace referencia a elementos generales del producto PLC – HMI del que forma parte este trabajo. Igualmente se describen las principales características que debe poseer el *firmware* y se identifican los *requerimientos funcionales y no* funcionales que regirían el desarrollo del *firmware*. También se detalla la arquitectura propuesta especificando los patrones arquitectónicos a utilizar, definiendo modelos que fundamenten el diseño del sistema. Estos elementos permiten establecer los aspectos básicos del *firmware* para la tarjeta de adquisición.

3.1. Problema y propuesta de solución

Con casi seis años de experiencia en el desarrollo de sistemas para la automatización industrial, surge en el 2010 el CEDIN, centro que cuenta hoy con varios productos desplegados en Cuba así como en la hermana República Bolivariana de Venezuela.

Tanto el SCADA Guardián del ALBA como el SCADA UX[7], productos del centro, constituyen sistemas grandes que necesitan considerables recursos de hardware para su implementación, lo que los convierte en soluciones costosas y poco eficientes para la automatización de pequeños procesos industriales. Estos inconvenientes ahuyentaban a pequeñas empresas que veían fuera de sus posibilidades el poder financiarse un proyecto de automatización.

A raíz de esta situación, en la línea de Sistemas Empotrados surge la idea de desarrollar un sistema completamente nuevo, análogo a las soluciones existentes, pero que entre sus principales características estuviese la sencillez, la escalabilidad, el bajo costo de los componentes de hardware, de las tareas de despliegue y que fuese desarrollado por un pequeño equipo de trabajo. La conjunción de todos esos aspectos resultaría en el producto ideal para cualquier empresa interesada en automatizar su producción industrial, así como supervisar cualquier proceso donde interviniesen mecanismos de supervisión y/o de control. Con este fin surge conceptualmente el PLC – HMI.

Este sistema está basado principalmente en dos nodos o recursos de ejecución: el nodo principal que consiste en una tarjeta CID 300-9 y el nodo complementario es una tarjeta diseñada por especialistas de

[7] Véase Glosario de Términos.

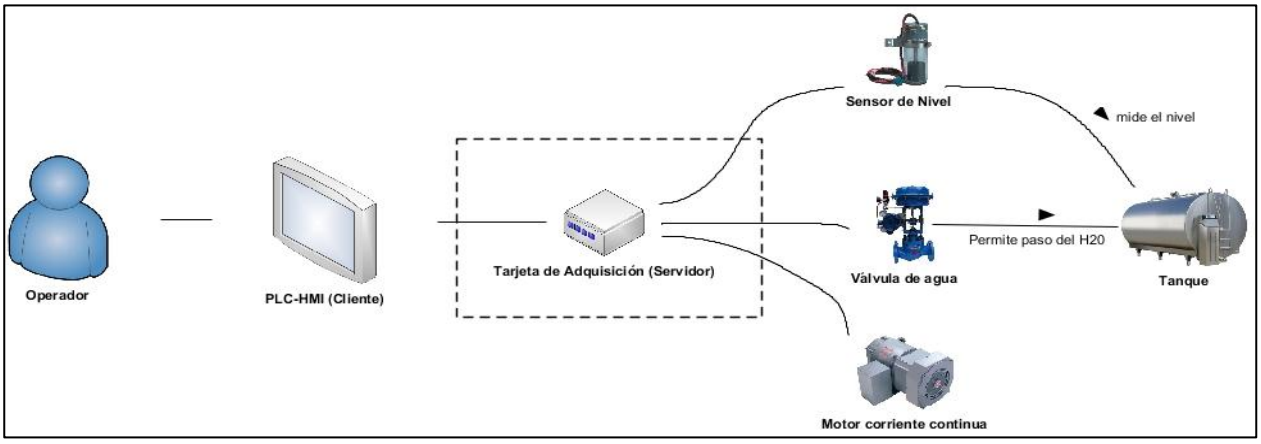
la línea Sistemas Empotrados basada en basada en un microcontrolador STM32.

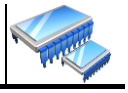
La CID 300-9 es desarrollada en el Instituto Central de Investigación Digital (ICID) radicado en La Habana. Esta tarjeta está basada en el microprocesador S3C2440 con arquitectura ARM9, (Jiménez, y otros, 2010). Según información extraída del artículo anteriormente referenciado, "*la tarjeta fue desarrollada para ser usada en equipos médicos pero sus características de bajo consumo de potencia, bajo costo, pequeño tamaño, alta potencia de cómputo, y el empleo del sistema operativo Linux*" la convierten en la opción ideal para el PLC – HMI: recursos de hardware idóneos, de bajo costo y producido en Cuba. Este dispositivo hospedará al sistema encargado de monitorear y controlar los procesos a automatizar.

Para complementar al PLC – HMI, se dispone de una tarjeta de procesamiento. Esta tiene como fin crear un mecanismo que conecte funcionalmente la CID 300-9 y los dispositivos de campo. La misión de la tarjeta de adquisición de datos basada en un microcontrolador STM32 es precisamente, viabilizar el proceso de recolección de datos así como las tareas necesarias para el control de los dispositivos.

Esta última tarjeta posee menores prestaciones de hardware que la CID 300-9 y cuenta con una arquitectura ARM sobre un núcleo Cortex-M3. Los dispositivos de campo serán conectados a esta tarjeta e indistintamente de la tecnología que los dispositivos implementen, la tarjeta debe posibilitar que la información fluya en ambos sentidos, desde la CID 300-9 hacia los dispositivos de campo y viceversa. Se cuenta con el componente de hardware (la tarjeta) pero no se dispone de un software que viabilice el proceso. Como resultado de este trabajo se diseñó un *firmware* que implementa un servidor para responder a las peticiones provenientes del PLC – HMI a través del puerto serie, utilizando un protocolo de comunicación específico. Entre los datos que viajarán por el canal de comunicación están los comandos enviados desde el PLC – HMI, así como la información asociada a las variables a monitorizar, la cual debe ser recolectada por el *firmware.*

Fig. 12 Escenario de ejemplo que representa la supervisión y el control de un proceso a través del PLC – HMI y la tarjeta de adquisición





3.2. Entorno de desarrollo

Para la selección de las tecnologías y herramientas que soportaran el desarrollo del *firmware*, prevaleció el criterio de que fueran tecnologías libres y gratis, lo cual apoya el concepto de bajo costo del producto PLC – HMI en general.

Como herramienta CASE se seleccionó **Visual Paradigm** para Linux. Como sistema operativo de desarrollo se escogió **Debian GNU/Linux**, el cual es mundialmente conocido por su estabilidad, lo cual lo convierte en la elección de muchos desarrolladores que utilicen entornos libres. Se decidió utilizar el sistema de control de versiones **Subversion**.

Como se comentaba en el capítulo anterior, **Eclipse CDT** fue seleccionado como IDE. Este entorno tiene como aspecto más importante que es ejecutado sobre Linux teniendo como base el *Open Development Environment for Embedded Application* (ODeV, según acrónimo del autor). El ODeV pretende aprovechar la integración de *plugins* que convierten a Eclipse en un poderoso IDE para el desarrollo de sistemas empotrados. ODeV incluye los siguientes componentes:

- ✓ **CodeSourcery GNU ARM EABI** - versión Lite 2011.03-42:

  El *toolchain* ARM EABI es el *port*[8] por defecto en Debian para la arquitectura ARM. CodeSourcery proporciona un *toolchain* basado en ARM EABI que permite que el código compilado escrito para la arquitectura ARM se ejecute en la arquitectura hospedera.
- ✓ **OpenOCD** - versión 0.5.0.

  El *Open On-Chip Debugger* (OpenOCD) permite descargar el binario del *firmware* hacia la memoria FLASH de la tarjeta.
- ✓ **Subclipse:**

  Subclipse es un *plugin* de Eclipse suministrado por CollabNet como soporte al sistema de control de versiones Subversion. El software se distribuye bajo la Licencia Pública Eclipse (EPL) 1.0 Licencia de código abierto.

3.3. Descripción del sistema propuesto

En este apartado se definen los aspectos claves del sistema a diseñar, enmarcándose en las fases de Planificación y Diseño que propone la metodología XP. Como resultado de la adaptación de XP a este trabajo, en este apartado se expondrán las historias de usuario, los requerimientos no funcionales del sistema así como lo referente al Plan de Entregas.

[8] Véase Glosario de Términos.

3.3.1 Historias de Usuario

Las Historias de Usuarios (HU) constituyen el remplazo de los grandes documentos de especificaciones de requerimientos. Son redactadas por los clientes para especificar lo que necesitan del sistema. Igualmente son aprovechadas por el equipo de desarrollo para, durante las reuniones de planificación, crear estimaciones de tiempo. Estos artefactos apoyan la posterior creación de las pruebas de aceptación, es decir, se deben crear una o varias pruebas de aceptación que verifiquen que la HU ha sido desarrollada correctamente.

Para el presente trabajo las HU se definieron mediante tormentas de ideas entre el equipo de desarrollo y los especialistas de la línea Sistemas Empotrados. Como resultado de estas tormentas de ideas se ha logrado confeccionar las HU, las cuales se derivan en una lista de tareas de programación. Si bien las HU tienen como punto de vista lo que el usuario necesita, las tareas de programación definen que se debe hacer para satisfacer esa necesidad.

Tabla 7 Historia de Usuario CM-01

Historia de Usuario	
Número: CM-01	**Nombre:** Efectuar comunicación con el PLC – HMI.
Usuario: PLC – HMI	**Iteración asignada:** 1
Prioridad en el Negocio: Alta	**Riesgo en el desarrollo:** Alto
Descripción: El sistema debe comunicarse eficientemente con el PLC – HMI para responder a las peticiones sobre la información de las variables y recibir el envío de comandos. Esto ha de realizarse usando comunicación serie.	
Observaciones: Se necesita diseñar un protocolo de comunicación que soporte la transferencia y recepción de datos y comandos entre el PLC – HMI y la tarjeta de adquisición.	

Tabla 8 Historia de Usuario CR-01

Historia de Usuario	
Número: CR-01	**Nombre:** Recolectar y enviar datos.
Usuario: PLC – HMI	**Iteración asignada:** 2
Prioridad en el Negocio: Alta	**Riesgo en el desarrollo:** Alto
Descripción: Desarrollar un mecanismo que permita acceder y obtener los valores de las variables que ofrece la tarjeta de adquisición sin tener algún dispositivo conectado y enviar estos datos según las peticiones del PLC – HMI.	
Observaciones: La tarjeta ofrece información acerca de la temperatura y el voltaje del microcontrolador.	

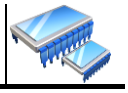


Tabla 9 Historia de Usuario TI-01

Historia de Usuario	
Número: TI-01	**Nombre:** Mostrar información en la pantalla LCD.
Usuario: PLC – HMI	**Iteración asignada:** 3
Prioridad en el Negocio: Baja	**Riesgo en el desarrollo:** Media.
Descripción: La tarjeta debe mostrar por la pantalla LCD trazas de las operaciones que se van ejecutando.	
Observaciones:	

3.3.2 Requerimientos no funcionales

(Rumbaugh, y otros, 1998) explican que:

> *"los requerimientos no funcionales especifican propiedades del sistema, como restricciones del entorno o de la implementación, rendimiento, dependencia de la plataforma, facilidad de mantenimiento, extensibilidad y fiabilidad, entre otros".*

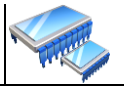
Por otro lado, (Sommerville, 2005) clasifica los requerimientos no funcionales en tres tipos: requerimientos del producto, requerimientos organizacionales y requerimientos externos. Igualmente dentro de los requerimientos del producto y los requerimientos organizacionales menciona ciertas categorías que son útiles para la especificación de los requerimientos no funcionales del *firmware* diseñado. Estas categorías son:

1. Requerimientos de eficiencia:
    a. Requerimientos de rendimiento.
    b. Requerimientos de espacio.
2. Requerimientos de fiabilidad.
3. Requerimientos de implementación.

Características del hardware

La tarjeta basada en el microcontrolador STM32F107VCT cuenta con los siguientes recursos de hardware *(Para más información véase Anexo 1)*:

- ✓ Microcontrolador basado en un microprocesador ARM Cortex – M3 con una frecuencia máxima de 72 MHz.
- ✓ Memoria SRAM: 64 Kb.



- ✓ Memoria FLASH: 256 Kb.
- ✓ Pantalla LCD de 3.2”, 240x320 con color TFT y táctil.
- ✓ 4 LED.
- ✓ 1 Interfaz de comunicación RS-232.
- ✓ 1 controlador Ethernet.

Requerimientos de rendimiento

1. El sistema debe garantizar una respuesta rápida en la interacción con los dispositivos y el PLC – HMI, siempre en el orden de los milisegundos.
2. El tiempo para la recolección de las variables debe ser establecido por configuración a partir de la frecuencia de muestreo que se determine.

Requerimientos de espacio

1. El binario resultante de la compilación debe de ajustarse a la cantidad de memoria ROM disponible, (*Véase Características del Hardware*).
2. El consumo de memoria RAM debe ajustarse a cantidad de memoria disponible (*Véase Características del Hardware*).

Requerimientos de fiabilidad

Dentro del dominio del *firmware* a desarrollar la palabra “fallo” se refiere a alguna situación anómala, un comportamiento diferente a lo esperado. Partiendo de lo expuesto anteriormente se definen los siguientes requerimientos de fiabilidad:

1. El sistema debe implementar un mecanismo que permita al PLC – HMI determinar cuándo ha ocurrido un fallo en la comunicación o en el estado de los dispositivos.
2. En caso de algún fallo en la comunicación con el PLC – HMI o con algún dispositivo, el sistema debe efectuar un número determinado de intentos para restablecer dicha comunicación.

Requerimientos de implementación

1. El código resultante de la implementación debe de estar regido por un Estándar de Codificación, permitiendo ser legible con el objetivo de facilitar las mejores posteriores del *firmware*. El Estándar de Código debe de establecerse y describirse en el presente documento.

3.3.3 Plan de Entregas

eXtreme Programming se basa en ciclos de liberación cortos, unos pocos meses como máximo,

desarrollando primeramente las funcionalidades más importantes, lo cual permite la reducción de riesgos pues para el final van quedando las funcionalidades menos complejas y menos importantes, (Beck, 2000).

En este apartado se define el Plan de Entregas que incluye las iteraciones a realizar para las entregas intermedias y la entrega final. Para distribuir las HU por iteraciones se tiene en cuenta la prioridad que definió el cliente para cada HU y respetando uno de los principios de la metodología XP que plantea que las mejoras deben implementarse al final. Como resultado de la priorización de las HU se definió la siguiente planificación:

Tabla 10 Plan de Entregas

No. Liberación	Historias de Usuario	Iteraciones	Tiempo estimado (semanas)
1	CM-01, CR-01	1 y 2	14
2	TI-01	3	4

3.4. Descripción de la Arquitectura

La Arquitectura de Software es una vista del sistema que incluye los componentes principales del mismo, el comportamiento de esos componentes y la forma en que estos componentes interactúan y colaboran para alcanzar la misión general del sistema, (Clements, 1996). Por tanto la vista arquitectónica se reduce a una vista abstracta, aportando el más alto nivel de comprensión y suprimiendo detalles como, por ejemplo, algoritmos y estructuras de datos.

Los estilos y patrones arquitectónicos aplicables a los sistemas empotrados de tiempo real son muy particulares debido a las características especiales de este tipo de sistemas. Estos patrones están generalmente orientados a solucionar problemas y optimizar áreas como:

- ✓ Rendimiento.
- ✓ Reducción al mínimo el consumo de recursos como memoria, procesamiento y energía.
- ✓ Seguridad.
- ✓ Tolerancia a fallos.

3.4.1 Arquitectura Cliente – Servidor

El modelo Cliente – Servidor constituye un patrón para software distribuido, el cual define dos tipos componentes principales: el Cliente o Maestro y el Servidor o Esclavo. Igualmente se define el modo de interacción entre ellos: la comunicación basada en un protocolo a través del mecanismo de encuesta - respuesta.

Cliente

- ✓ Constituye el componente activo, inicia la comunicación con el servidor enviando las peticiones o encuestas.
- ✓ Debe de obtener una respuesta a partir de la petición.

Servidor

- ✓ Constituye el componente pasivo, espera por las peticiones de los clientes.
- ✓ Procesa las peticiones al recibirlas y ofrece una respuesta.

3.4.2 Patrones arquitectónicos

Five – Layer

El patrón *Five – Layer* (del inglés Cinco Capas) es específicamente útil para la estructuración general de los sistemas empotrados y de tiempo real. Constituye una propuesta de (Douglass, 2002) basada en el patrón arquitectónico Multi – Capas. Este patrón organiza el sistema en un conjunto de capas donde cada una de ellas asume una responsabilidad en específico. El patrón aporta como ventaja la portabilidad del sistema, pues las capas al ser independientes una de otras, cualquiera puede ser recodificada sin que las demás capas se vean afectadas.

El patrón agrupa en una capa común de abstracción los componentes afines al dominio de la capa. Plantea las siguientes capas:

1. Aplicación.
2. Interfaz de Usuario.
3. Comunicación.
4. Abstracción de RTOS.
5. Abstracción de Hardware.

En la fig. 13 se muestra la vista lógica de la propuesta de arquitectura para el *firmware* después de aplicado el patrón arquitectónico Five – Layer. Las capas presentadas representan las abstracciones fundamentales a partir del dominio del sistema. A continuación se detallan brevemente las funciones de cada una de ellas:

1. **Núcleo (Core):**

   Tiene la responsabilidad de la recolección de los datos y procesar las variables asociadas a los dispositivos conectados a la tarjeta. Constituye la capa más importante del *firmware* pues implementa toda la lógica del funcionamiento de la tarjeta de adquisición.

2. **Interfaz de Seguimiento (Tracking Interface):**

   Constituye la interfaz externa de la tarjeta de adquisición. Contiene los componentes que posibilitan a los "*operadores*" dar seguimiento a las operaciones que se van realizando en la tarjeta. Se basa en el uso de una pantalla LCD y los diodos LED para transmitir de forma sencilla el estado de la tarjeta.

3. **Comunicación (Communication):**

   Capa responsable de la comunicación entre la tarjeta de adquisición y el PLC – HMI. Brinda una interfaz con ese objetivo que es utilizada por el **Núcleo** para el envío y la recepción de los datos. Se responsabiliza por la implementación de la lógica del protocolo de comunicación.

4. **Sistema Operativo de Tiempo Real (RTOS):**

   Se centra en brindar una interfaz que abstrae al resto de las capas de la interacción directa con el hardware. Se encarga de brindar un soporte para la programación basada en tareas, la administración de memoria y el uso de todo un conjunto de herramientas de programación que facilitan la implementación del sistema como son: semáforos de sincronización, semáforos de exclusión, colas, interrupciones y el manejo de recursos.

5. **Capa de Abstracción de Hardware (HAL):**

   En el caso específico de este sistema, la capa de abstracción de hardware provee un mecanismo genérico para la interacción con los distintos dispositivos conectados a la tarjeta independientemente del fabricante y de la naturaleza de los dispositivos.

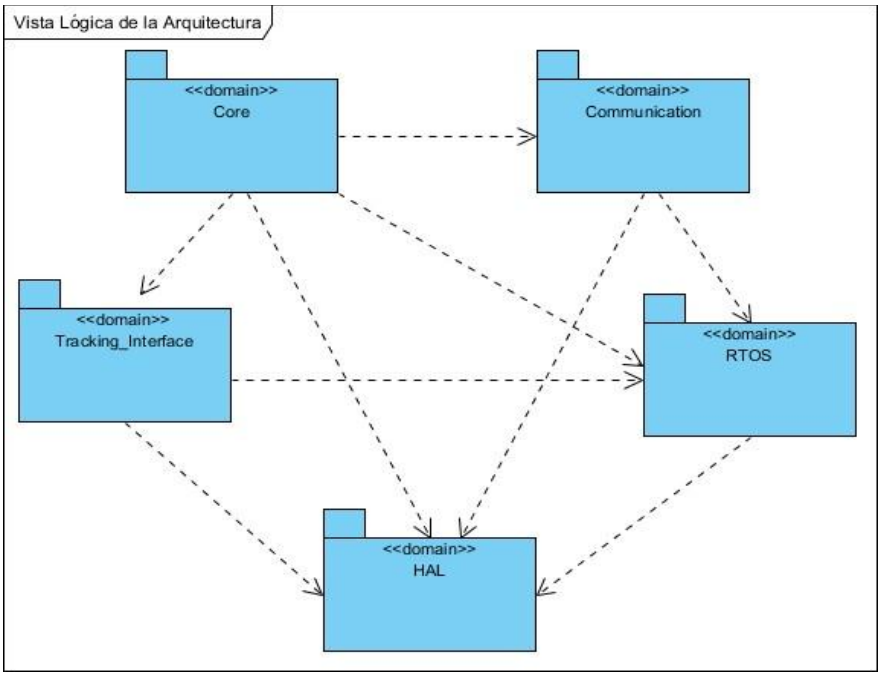


Fig. 13 Vista lógica de la Arquitectura

Hardware Abstraction Layer

Este patrón es ampliamente utilizado en sistemas operativos de tiempo real y en sistemas empotrados. Es referenciado por (Eloranta, y otros, 2009) donde los autores proponen diseñar una interfaz para una capa de abstracción de hardware que contenga las funciones para el control de los dispositivos de hardware.

La capa aporta una interfaz genérica para los dispositivos de hardware, con la finalidad de que los mismos puedan ser accedidos indistintamente por el **Núcleo**. La capa de abstracción de hardware puede ser recodificada sin la necesidad de cambios en la interfaz, igualmente permite la adición de nuevos dispositivos solamente con la implementación del comportamiento para esos nuevos dispositivos.

Virtual Timestamps

Es catalogado por (Eloranta, et al., 2009) como un patrón enfocado en la tolerancia de fallos. El patrón propone establecer una marca de tiempo (estampa) a los valores recolectados. Las estampas ayudarían a determinar el orden de las acciones de recolección.

Aportes

- ✓ Capacidad de prueba: permite elaborar un historial de ejecución para poder realizar análisis del comportamiento del sistema.
- ✓ Precisión: Permite la trazabilidad de los acontecimientos.

3.5. Protocolo de comunicación

ProC-ARM es el protocolo que especifica las reglas para el intercambio de información entre las tarjetas de adquisición basadas en microcontroladores STM32 y el PLC – HMI. Este protocolo se vincula con los niveles 1 y 2 del modelo OSI para la transmisión de datos a través del puerto serie y la interfaz Ethernet, teniendo como principio el ser práctico para cualquiera de los dos tipos de comunicación.

El protocolo diseñado se detalla con profundidad en el *Anexo 2*, donde se especifica la estructura y los elementos necesarios para establecer la comunicación entre los nodos cliente y servidor.

3.6. Artefactos modelados

A pesar de que la metodología XP no define claramente artefactos que modelen el diseño del software, se determinó emplear dos artefactos, el Diagrama de Estados y el Diagrama de Flujo, con el propósito de especificar la estructura y el comportamiento del sistema. (Zurrell, 2000) identifica a los diagramas de estados y los diagramas de flujo como una de las herramientas básicas para el diseño de sistemas empotrados.

3.6.1 Diagrama de Estados

El comportamiento de los sistemas empotrados suele ser modelado mediante una máquina de estados. Según (Rumbaugh, y otros, 1998) una máquina de estados es un comportamiento que especifica las secuencias de estados por las que pasa un objeto durante su vida, junto con las respuestas a determinados eventos.

El diagrama de estados de UML proporciona una representación gráfica para los tres elementos que se definen a continuación, (Rumbaugh, et al., 1998):

- ✓ **Estado:** Condición o situación en la vida de un objeto durante la cual satisface alguna condición, se realiza alguna actividad o se espera algún evento.
- ✓ **Evento:** es la especificación de un acontecimiento significativo que ocupa un lugar en el tiempo y en el espacio. En el contexto de las máquinas de estados, un evento es la aparición de un estímulo que puede disparar una transición de estados.
- ✓ **Transición:** Es una relación entre dos estados que indica que un objeto que esté en el primer estado realizará ciertas acciones y entrará en el segundo estado cuando ocurra un evento especificado y se satisfagan unas condiciones especificadas.

Para el *firmware* a empotrar en la tarjeta de adquisición se han diseñado 3 estados: **Inicialización**, **Configuración** y **Recolección**, considerado a este último un macro estado que a su vez ha sido descompuesto en otros tres estados: **Recepción**, **Envío** y **Procesamiento**. A continuación se describirán cada uno de estos estados.

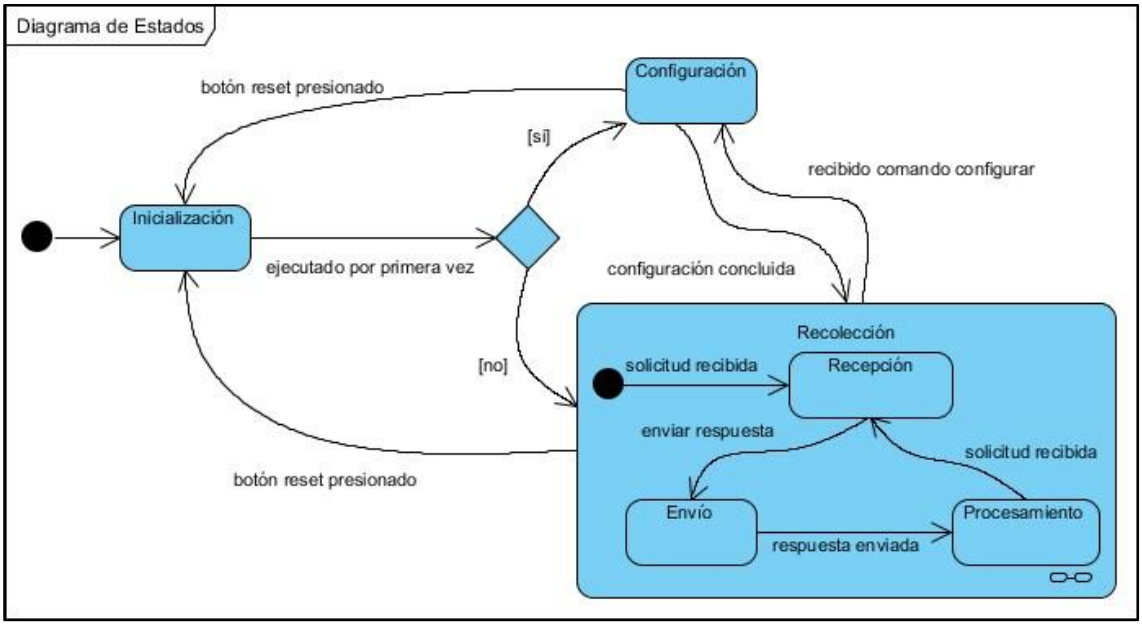


Fig. 14 Diagrama de Estados

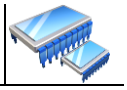


Tabla 11 Descripción de los Estados, Eventos y Transiciones

Elemento	Clasificación	Descripción
Inicialización	Estado	El *firmware* inicia su ejecución. Es el estado donde se inicializan las interfaces de hardware, las estructuras de datos internas así como las variables globales.
Ejecutado por primera vez / [Si]	*Evento*	Si es la primera vez que se ejecuta el *firmware*, ocurre una transición del estado Inicialización hacia el estado Configuración
Ejecutado por primera vez / [No]	*Evento*	Si el *firmware* ha sido ejecutado en otras ocasiones la transición ocurre desde el estado Inicialización hacia el estado Recolección.
Configuración	Estado	Se establecen los parámetros de configuración asociados a las interfaces de hardware y los mecanismos internos para el funcionamiento del *firmware*.
Botón Reset presionado	*Evento*	Si el botón Reset de la tarjeta es presionado se reinicia el sistema y ocurre una transición hacia el estado Inicialización.
Configuración concluida	*Evento*	Ocurre una transición hacia el estado Recolección una vez establecidos los parámetros de configuración.
RECOLECCIÓN	Macro estado	Es el estado donde se desarrolla toda la lógica del proceso de recolección. Es el estado más importante y complejo dentro de la ejecución del *firmware*.
Botón Reset presionado	*Evento*	Ídem.
Inicio / *Dato recibido*	Seudo – Estado / *Evento*	Seudo – estado al cual se llega al ocurrir las transiciones desde los estados Inicialización y Configuración. No constituye propiamente un estado. Al ocurrir la recepción de una solicitud a través de los controladores UART/USART o Ethernet ocurre la transición hacia el estado Recepción.
Recepción	Estado	Se realizan un conjunto de actividades a partir de la información recibida del nodo cliente (PLC – HMI).
Dato para enviar	*Evento*	Si está disponible la información solicitada por el cliente ocurre la transición hacia el estado Envío.
Envío	Estado	Se envía la información solicitada por el cliente.
Dato enviado	*Evento*	Al ser enviada para el cliente la información solicitada ocurre una transición hacia el estado procesamiento.
Procesamiento	Estado	Estado donde se recolecta y trata la información de las variables asociadas de los procesos industriales.
Dato recibido	*Evento*	Al recibir una solicitud del cliente ocurre una transición hacia el estado Recepción para responder la solicitud.

3.6.2 Diagrama de Flujo

Los Diagrama de Flujo son utilizados en la Informática para representar gráficamente las actividades por los que atravieza un sistema, algoritmo o proceso durante su ejecución. Seguidamente se muestra un diagrama de flujo que amplía la información descrita anteriormente en el diagrama de estado.

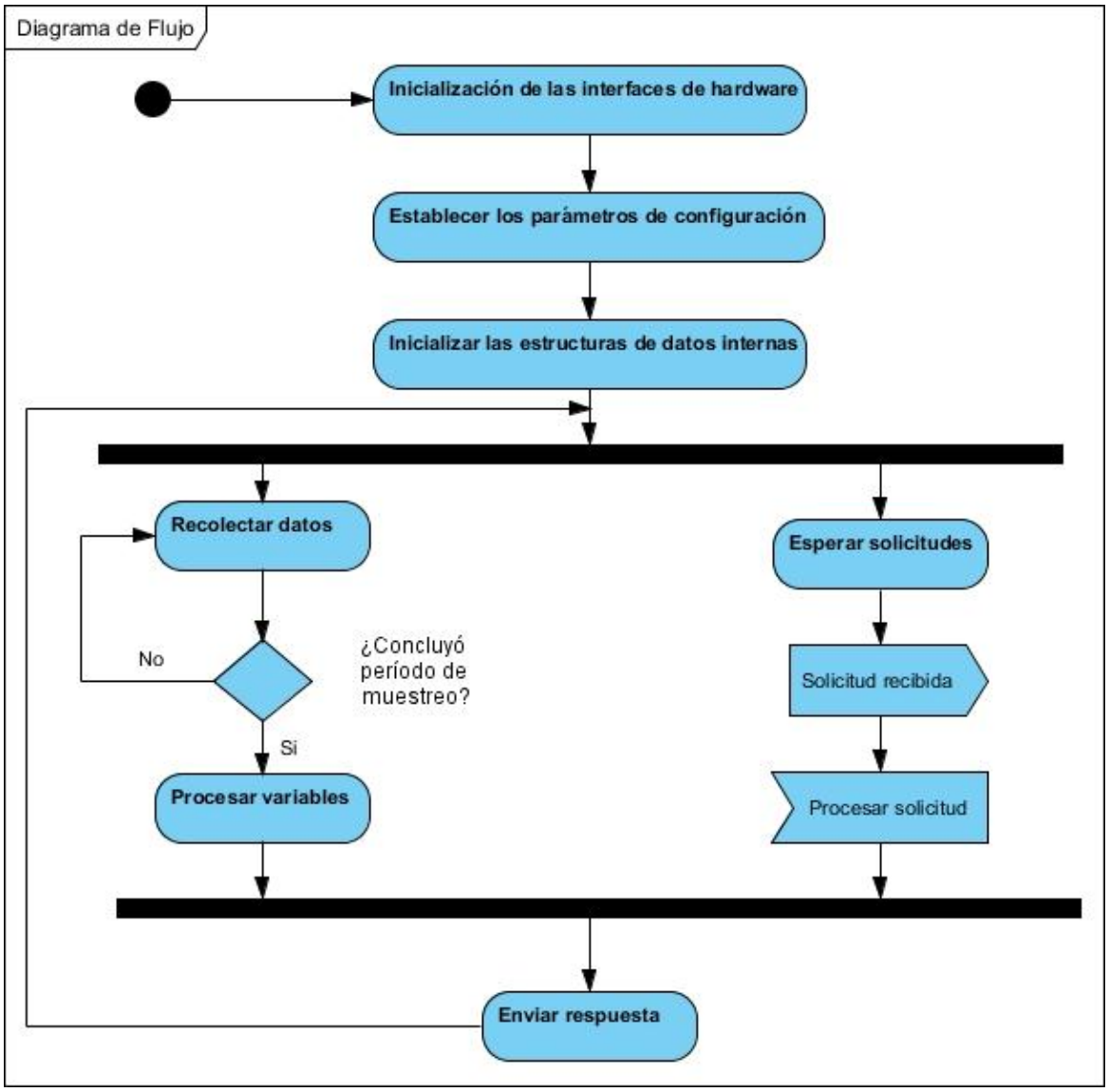
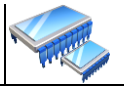


Fig. 15 Diagrama de Flujos

3.7. Conclusiones del capítulo

Como conclusiones de este capítulo podemos mencionar que se establecieron los aspectos básicos con los cuales debe contar el *firmware* para dar respuesta a la problemática planteada. Esto fue posible pues:



- ✓ Se describió el entorno de desarrollo que apoye la implementación del sistema el cual permitirá alcanzar el objetivo de forma rápida y cómoda.
- ✓ Los requerimientos no funcionales especificados y los requerimientos funcionales, obtenidos a partir de las Historias de Usuarios, asegurarán la obtención de un sistema confiable, funcional, portable, eficiente y fácilmente mantenible.
- ✓ La arquitectura propuesta está orientada a satisfacer los requerimientos especificados.

Capítulo 4

VALIDACIÓN DE LA SOLUCIÓN

En el presente capítulo se hace una pequeña introducción a los principales conceptos asociados con el proceso de Verificación y Validación de software. A partir del de los elementos expuestos se especifica la técnica seleccionada para la validación del este trabajo, así como se describe el prototipo funcional que valida el diseño propuesto. Por último se reseñan las pruebas aplicadas al prototipo funcional.

4.1. Verificación y Validación

Durante el ciclo de vida del desarrollo de software se hace necesario chequear que el software que se construye es el mismo que el que el cliente necesita. Esa misión le corresponde al proceso de chequeo y análisis denominado *Verificación y Validación*.

La Verificación y Validación (V&V de aquí en adelante) del software constituyen prácticas para el Aseguramiento de la Calidad del Software (SQA, siglas en inglés de Software Quality Assurance). Según (Sommerville, 2005), la V&V es una actividad que se ejecuta dentro de cada fase de desarrollo del software y contempla actividades como la revisión de los requerimientos, la revisión del diseño, inspecciones al código fuente y por último la ejecución de las pruebas.

(Pressman, 2005) se basó en una definición de Barry W. Boehm para referenciar las dos frases siguientes que definen la V&V:

- ✓ Verificación: ¿Estamos construyendo el producto correctamente?
- ✓ Validación: ¿Estamos construyendo el producto correcto?

Es decir, la Verificación tiene como objetivo garantizar que el producto cumple con los requerimientos funcionales y no funcionales especificados. Por el otro lado la Validación persigue *"demostrar que un producto o componente del mismo satisface el uso para el que se creó al situarlo sobre el entorno previsto",* (INTECO, 2009). La Validación constituye un concepto más amplio que la Verificación pues se encarga de demostrar que el software que se especificó es el que realmente cumple con las expectativas del cliente.

Durante la realización de las actividades asociadas a V&V se pueden utilizar distintos tipos de técnicas. En general, estas técnicas se agrupan en dos categorías, (Juristo, y otros, 2005):

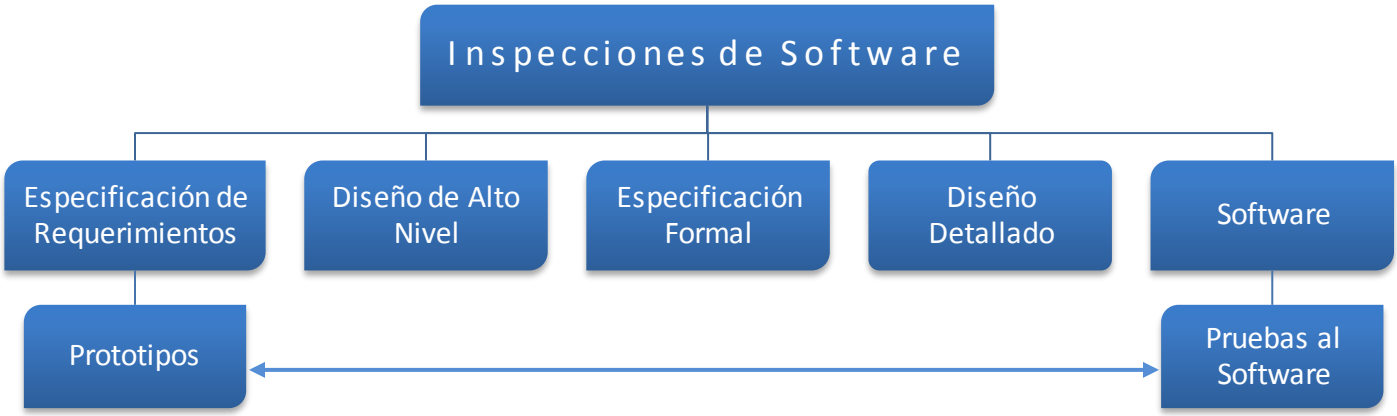


Fig. 16 Objetivos de las Inspecciones de Software

- ✓ **Técnicas estáticas:** Buscan inconformidades sobre el sistema en reposo. Estudian los distintos modelos que componen el software buscando posibles faltas en los mismos. Estas técnicas se pueden aplicar a la especificación de requisitos, al análisis, el diseño y hasta al código fuente.
- ✓ **Técnicas dinámicas:** Se somete el sistema funcionando a un conjunto de entradas con el objetivo de detectar fallos. Se verifican si existen incongruencias entre la salida esperada y la salida real. El proceso de aplicación de técnicas dinámicas es comúnmente denominado "pruebas de software" y se ejecutan indirectamente sobre el código.

Como ya se ha mencionado, para la ejecución de las actividades de V&V a través de técnicas estáticas no es totalmente necesario tener el código fuente de la aplicación. Por consiguiente, este tipo de técnica es utilizada en las fases previas a la codificación de sistema. En cambio, las técnicas dinámicas son realizadas sobre el sistema funcional, por lo que es necesario que componentes, módulos o el sistema en general permitan ser ejecutados.

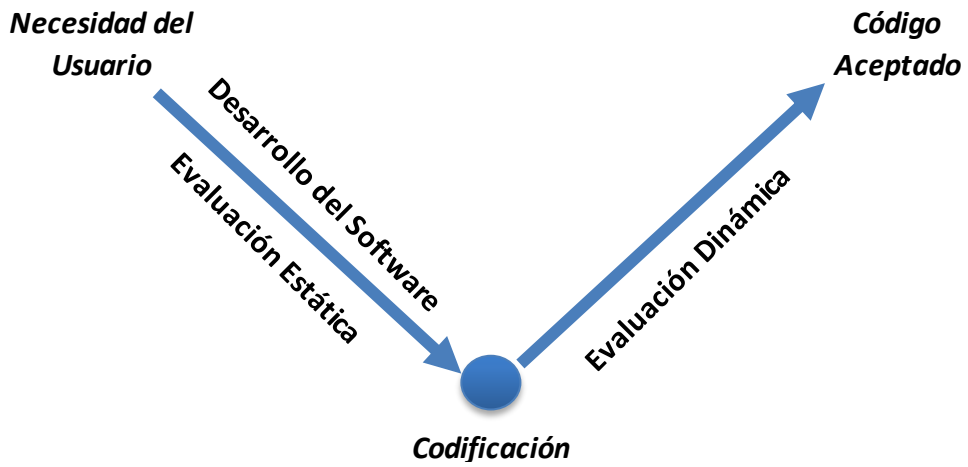
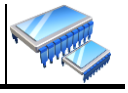


Fig. 17 La Evaluación del software dentro del proceso de desarrollo



Al diseñar un *firmware* para la tarjeta de adquisición de datos basada en microcontroladores, se necesitó validar y verificar que los requerimientos y el diseño propuesto se correspondían con las necesidades del cliente. La técnica de validación escogida tuvo como base el desarrollo de un prototipo funcional al que se aplicaron técnicas dinámicas.

Teniendo en cuenta la necesaria continuidad del desarrollo del *firmware*, la generación de un prototipo funcional constituye un medio eficaz para lograr que los futuros programadores comprendan en su totalidad la especificación de los requerimientos y el diseño del sistema. Es más fácil para un programador construir un sistema desde un modelo en funcionamiento y documentado que a partir de un documento de diseño.

4.2. Prototipos de software

Un prototipo de software es un borrador de un sistema o de una parte del mismo, que entre sus objetivos se encuentran el desarrollar un modelo de trabajo de los componentes funcionales y el poder validar los requerimientos del cliente, (INTECO, 2008) .

El modelo de desarrollo de la metodología utilizada (XP) se basa en la filosofía de construir prototipos funcionales. Estos prototipos deben ir incrementando poco a poco las funcionalidades del sistema, provocando que se entregue una versión del producto operativo con cada incremento. Durante las primeras iteraciones se obtienen versiones incompletas del producto final, pero que proporcionan al usuario la funcionalidad que precisa. Estos prototipos constituyen una plataforma para la evaluación del futuro sistema, (INTECO, 2009).

4.3. Conceptos utilizados

Durante el modelado del prototipo se hizo necesario utilizar conceptos que permitieran estructurar el sistema, así como lograr una abstracción de la información a recolectar de los dispositivos.

Variable

El uso del concepto de *variables* en el contexto del sistema parte básicamente de la definición de variable para los procesos industriales. Las variables de proceso son magnitudes físicas tales como temperatura, presión, flujo, nivel, etc., que deben ser controladas o supervisadas por medio de la instrumentación de los procesos.

Dentro del sistema se representó la información de esas variables a través de estructuras para tener conocimiento del estado del proceso industrial. Las variables definidas pueden ser analógicas o digitales.

Las variables analógicas están asociadas a valores continuos en el tiempo (ej. temperatura, voltaje, presión) mientras que las variables digitales representan aquellos valores discretos comúnmente asociado a estados (ej. estado de un motor: encendido).

Mensajes

Los mensajes son el mecanismo que especifica la comunicación entre la tarjeta de adquisición y el PLC – HMI. Es decir, la tarjeta recibe mensajes que encapsulan la información solicitada por el cliente y responde a través de mensajes que contienen la información solicitada.

Comandos

Los comandos son instrucciones que recibe y envía el *firmware* y que conllevan a que se realice alguna acción. En el sistema los comandos están representados por códigos que han sido establecidos tanto en el cliente como en el servidor.

Tramas

Las tramas constituyen la unidad para el envío de los datos por el canal de comunicación. Es un conjunto de bytes consecutivos que tienen un orden lógico. Los mensajes constituyen la abstracción de las tramas, es decir, los mensajes son convertidos a tramas para su envío.

4.4. Descripción del prototipo

Para desarrollar el prototipo funcional se hizo una selección de las Historias de Usuarios (HU) con mayor prioridad. Estas HU se tradujeron directamente en los requerimientos funcionales más críticos del sistema. Durante el proceso de desarrollo se utilizaron las tecnologías y herramientas planteadas en el capítulo anterior.

Las HU implementadas son las siguientes:

- ✓ CM-01 Efectuar comunicación con el PLC – HMI.
- ✓ CR-03 Recolectar y enviar datos.

Efectuar comunicación con PLC – HMI

Se implementó la lógica del protocolo de comunicación ProC – ARM. El sistema es capaz de recibir las solicitudes provenientes del PLC – HMI, realizar las tareas relacionadas con la petición y responder a la solicitud.

La comunicación a través del puerto serie fue codificada garantizando que el mecanismo de comunicación desarrollado fuese independiente del transporte utilizado. En futuras versiones se debe incorporar la comunicación TCP, para lo cual solo es necesario codificar dentro de la capa de *Communication* los

aspectos referentes a la pila TCP **lwIP**, la cual fue integrada al código fuente del prototipo.

Recolectar y enviar datos

El sistema es capaz de recolectar los datos y enviarlos en respuesta a las solicitudes realizadas por el PLC – HMI. Se simularon valores desde la tarjeta pues no se contó con los dispositivos necesarios para la recolección de "datos reales en tiempo real". El sistema es capaz de recolectar los valores a través de uno de los ADC con los que cuenta la tarjeta.

A través del siguiente diagrama de secuencia se muestra la lógica del funcionamiento del *firmware* cuando recibe un comando determinado. Para modelar el comportamiento del sistema se utilizaron los siguientes estereotipos de UML para los sistemas de tiempo real:

Tabla 12 Estereotipos usados de la extensión UML-RT

Estereotipo	Representación	Descripción
Tarea		Una tarea constituye un hilo de ejecución.
Cola		Mecanismo FIFO (First In First Out). Son la forma principal de comunicación entre las tareas.
Estructura para almacenar datos		Generalmente zonas críticas donde se almacena información necesaria para la ejecución del sistema.
Semáforo Mutex (Exclusión Mutua)		Semáforo que garantiza la exclusión mutua entre las tareas para el acceso a zonas críticas. El semáforo otorga un *token*[9] de acceso cuando la zona a la que se intenta acceder está libre. El *token* debe ser devuelto al salir de la zona crítica.
Semáforo Binario		Tipo de semáforo utilizado para la sincronización entre tareas. Este se encarga de notificar a una tarea cuando ocurra el evento deseado. A diferencia del *Mutex* el *token* no necesita ser devuelto.

[9] Véase Glosario de Términos

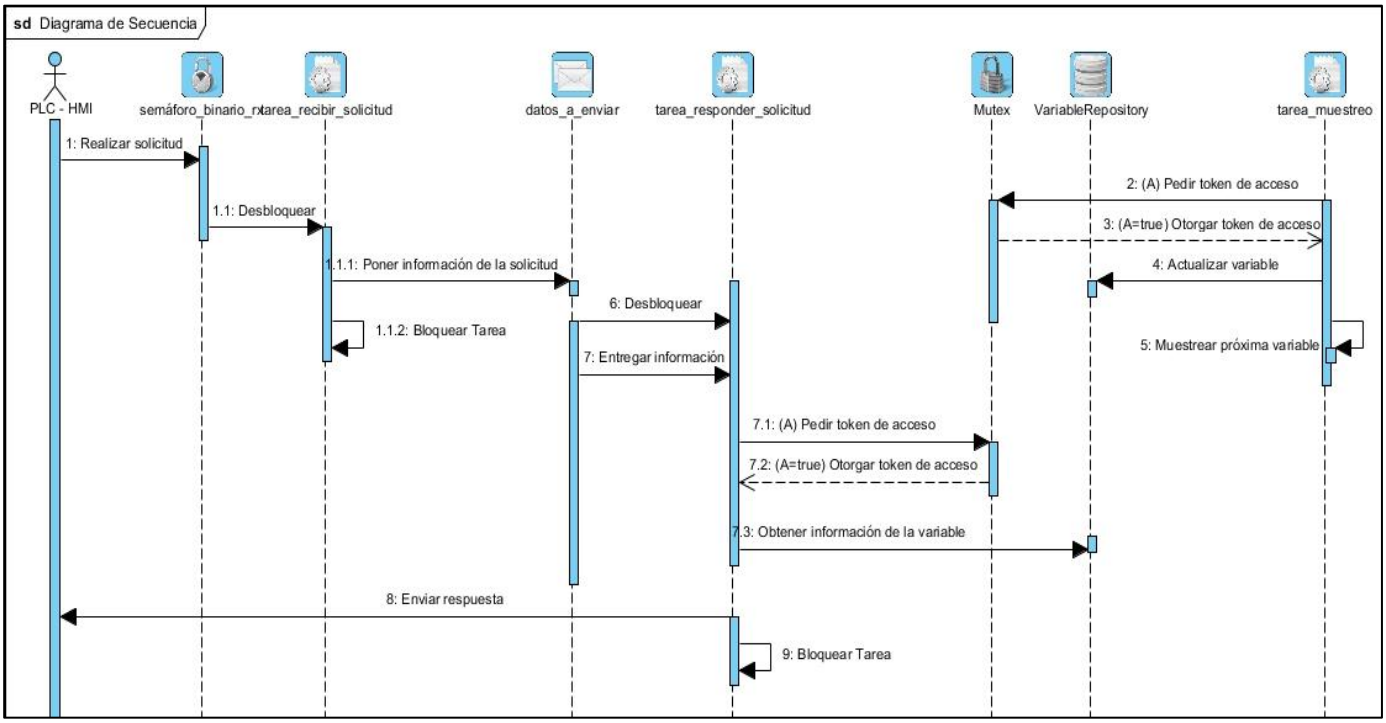


Fig. 18 Diagrama de Secuencia

4.4.1 Técnicas utilizadas

Programación basada en objetos

En el capítulo 2 se hizo referencia a la programación basada en objetos. Para la implementación del prototipo funcional que valida este trabajo, se utilizaron esos conceptos. Como C no soporta explícitamente la POO, se usaron varias técnicas que lograron un acercamiento a conceptos propios de este paradigma como son la encapsulación, la abstracción, la herencia y el polimorfismo.

Para lograr encapsular los atributos privados dentro de las estructuras se utilizó una técnica conocida como *forward declaration* (declaración adelantada). Esta técnica plantea la declaración de un identificador que pueda ser utilizado antes de ser definido. La declaración adelantada le permite al compilador conocer el espacio en memoria a ocupar por el identificador pero oculta la información que contiene.

En cuanto a la abstracción se utilizaron estructuras que se comportaran como clases, definiendo constructores y destructores. Se especificaron métodos de acceso para los atributos encapsulados en estas estructuras. Igualmente se logró encapsular métodos en el interior de las estructuras haciendo uso de punteros a funciones, aspecto que C como lenguaje procedural y estructurado no soporta.

Para lograr herencia entre estructuras se agruparon en una estructura “padre” los atributos comunes entre los “hijos”. Esta estructura “padre” se incluyó como un puntero dentro de las estructuras hijas y se definieron métodos de acceso para los atributos del padre.

Referente al polimorfismo se implementó un polimorfismo estático. Se definieron variables que permitieran conocer en cualquier momento el tipo de dato de la estructura en cuestión. Para cada tipo de estructura se implementaron métodos en específico. Así los punteros a funciones incluidos como atributos (definidos todos con el mismo nombre para estructuras “hermanas”) almacenaron las direcciones de sus respectivos métodos.

Destacar que aunque se logró una aproximación a la POO, el resultado no es el mismo. Este paradigma OO proporciona muchas ventajas que no fue posible aprovechar dado el requerimiento de programar el sistema con el lenguaje C.

Árbol Trie

Los árboles Trie (nombre proveniente de la palabra en inglés re*trie*val, en español recuperación) son una estructura de datos ordenada basada en árboles que se utiliza para almacenar un arreglo asociativo donde las llaves suelen ser cadenas. (Webster's Online Dictionary). El *trie* utiliza las partes de la clave para organizar y buscar entre su colección de datos.

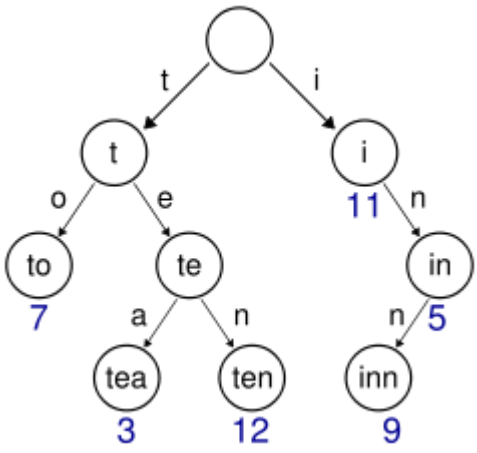


Fig. 19 Representación de un trie para "to", "tea", "ten", "inn"

Uno de los requerimientos no funcionales para el *firmware* especifica que el tiempo de respuesta a las peticiones realizadas por el cliente tiene que ser en el orden de los milisegundos. La búsqueda en un trie, para una clave de longitud m, es de Θ(m) en el peor caso, lo que garantiza el rápido acceso a la información solicitada.

4.4.2 Estándar de código

El código fuente tiene dos funciones igualmente importantes: tiene que funcionar y debe comunicar claramente la forma en que trabaja para que los futuros programadores comprendan fácilmente el código implementado.

XP como metodología ágil promueve la codificación basada en estándares, de manera que el código legado sea fácilmente entendible por todo el equipo de desarrollo facilitando así la recodificación,

(Joskowicz, 2008).

Como estándar de código se utilizó el estándar propuesto por Jack G. Ganssle para el desarrollo de *firmware*, (Ganssle, 2004). El estándar utilizado proporciona los niveles mínimos de legibilidad y facilidad de mantenimiento. El estándar se especifica en el *Anexo 3*.

4.4.3 Estructura

El código fuente del prototipo se organizó de forma tal que mantuviera una estructura física y lógica que se correspondiera con el estándar de código mencionado anteriormente.

Estructura física

El código está separado en carpetas independientes que, a pesar de ser un aspecto físico para la organización del código, tienen un significado lógico.

Como se observa en la fig. 20, los archivos cabeceras fueron almacenados en la carpeta *headers,* mientras que los archivos fuente fueron recogidos en la carpeta *source.* También se incluyó una carpeta *libs* donde se ubicó el código relacionado con las bibliotecas utilizadas en el sistema.

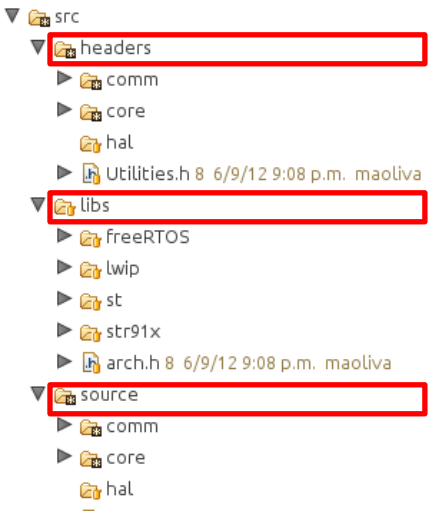


Fig. 20 Estructura del código

El código dentro de las carpetas *headers* y *source* se organizó según las capas de la arquitectura mencionadas en el capítulo 3. Solo se crearon carpetas relacionadas con las capas *Core* y *Communication* pues las capas *RTOS*, *Tracking Interface* y *HAL* están relacionadas con las bibliotecas utilizadas.

Para una mayor comprensión se modeló el siguiente Diagrama de Construcción donde se muestran los componentes que describen la estructura física del sistema así como las relaciones entre ellos.

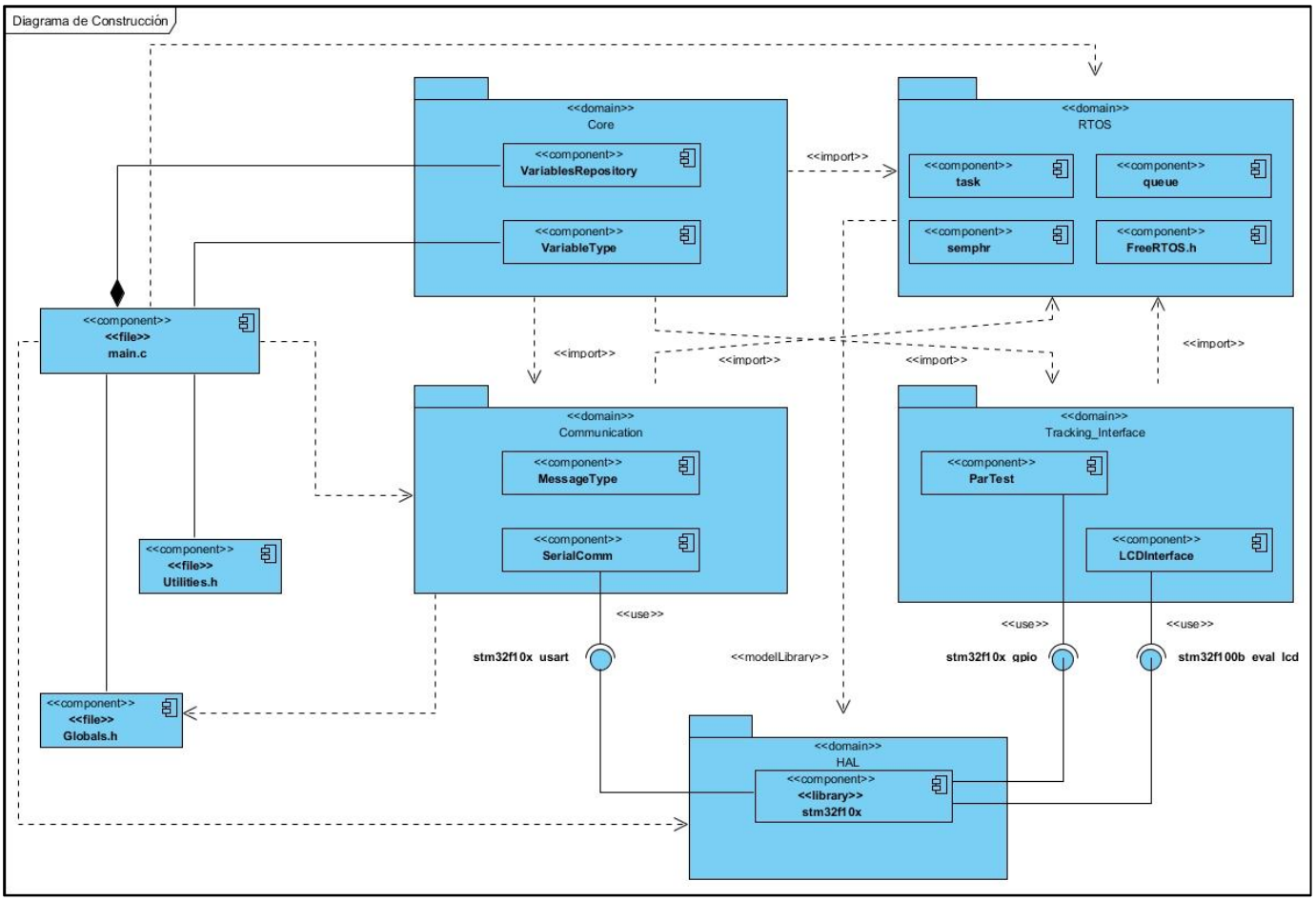


Fig. 21 Diagrama de Construcción

Estructura lógica

En el siguiente diagrama de ficheros muestra el modelado de las estructuras haciendo uso del estereotipo de *clases*. A continuación se hará una descripción de cada elemento que se representa en el diagrama.

Tabla 13 Descripción del Diagrama de Archivos

Elemento	Estereotipo	Descripción
variable_t	Enumeración	Se utiliza para clasificar a las variables en analógicas o digitales.
analog_var_t	Enumeración	Determina el tipo de dato del atributo "valor" para las variables analógicas. Este puede ser entero o real.
digital_var_state	Enumeración	Determina los estados en que pueden estar las variables digitales. Este puede ser encendido o apagado.
CompoundData	Tipo de estructura	Representa la información solicitada por el cliente.

Trie	Tipo de estructura	Definición del Repositorio de Variables; la implementación del árbol Trie descrito anteriormente. Se define a través de un *forward declaration.* El Repositorio de Variables se utiliza para almacenar las variables que deben ser monitoreadas. Estas están asociadas a un proceso industrial en específico.
VarNode	Tipo de estructura	Definición de los nodos del Repositorio de Variables. Se define a través de un *forward declaration.*
Data	Tipo de estructura	Tipo de Estructura contenida por los nodos del Repositorio de Variables. Tiene como atributos el tipo de variable que almacena y la variable.
VariableType	Tipo de estructura	Tipo de Estructura que recoge las características que comparten todos los tipos de variables.
AnalogVariable	Tipo de estructura	Representación de las variables analógicas. Contiene los atributos que definen a este tipo de variable. Encapsula los atributos de su "padre".
DigitalVariable	Tipo de estructura	Representación de las variables digitales. Contiene único atributo el estado en que se encuentra el objeto representado por la variable. Encapsula los atributos de su "padre".
_Trie	Estructura	Instancia del Trie. Representa un Repositorio de Variables. Contiene un conjunto de métodos que permiten al programador acceder eficaz y eficientemente a los datos que almacena.
_VarNode	Estructura	Instancia de tipo de estructura VarNode. Representa un nodo del Repositorio de Variables. Como nodo al fin contiene un Data que a su vez contiene la variable almacenada en el nodo.
MessageType	Tipo de estructura	Estructura que recoge las características que comparten los distintos mensajes. Contiene el método *assembleFrame,* especie de método virtual que se redefine para cada mensaje o "hijo".
AckResponseMsg	Tipo de estructura	Tipo de mensaje que se envía al cliente como confirmación de la solicitud.
ReadResponseMsg	Tipo de estructura	Tipo de mensaje que se envía al cliente como respuesta a la solicitud de información acerca de determinada variable.

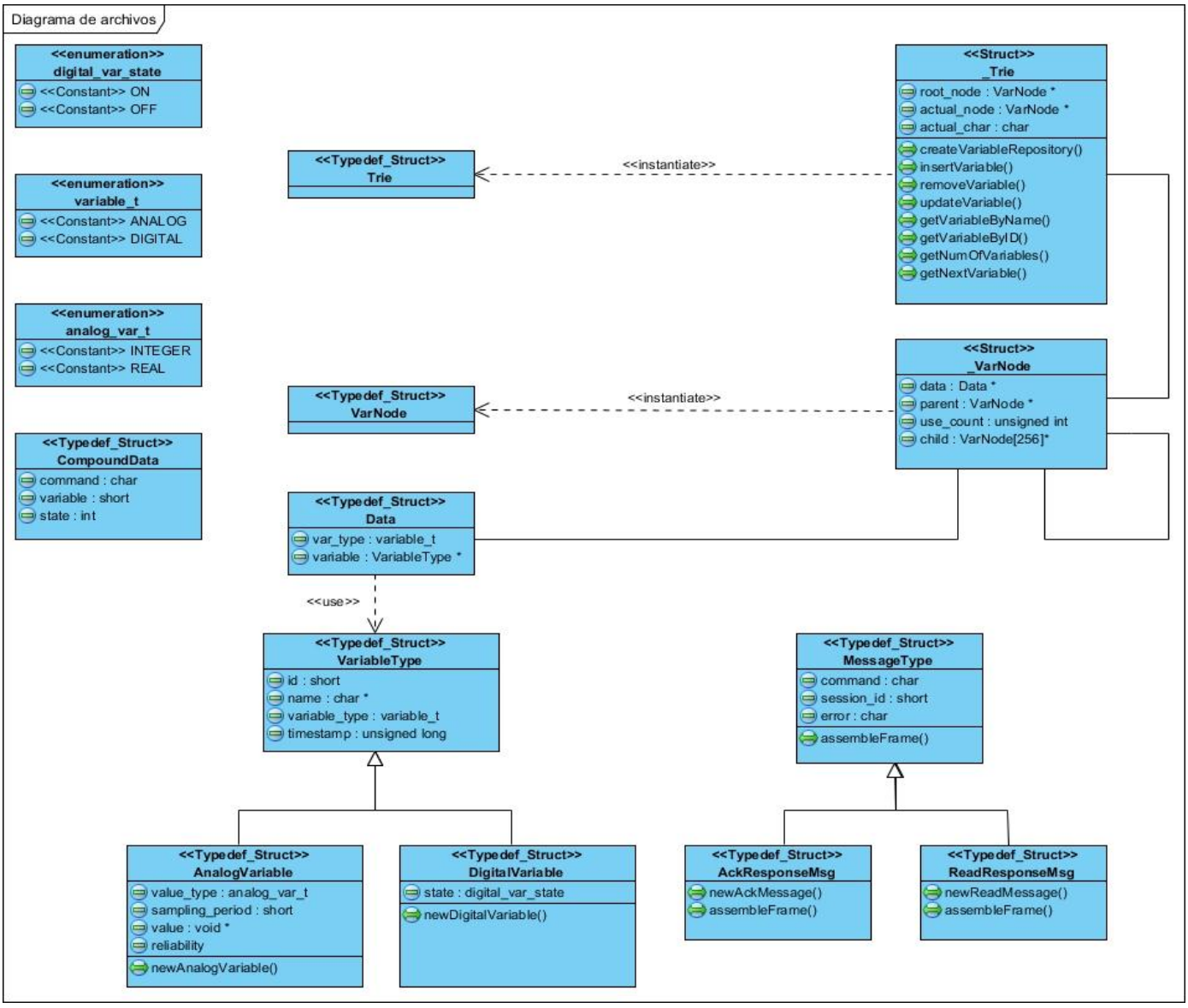


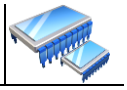
Fig. 22 Diagrama de Archivos

4.5. Fase de pruebas

Entre las características que se resaltaron anteriormente de la metodología XP se señaló que contemplaba un modelo de desarrollo basado en pruebas, lo que se conoce como *Test Driven Development* (TDD). Las pruebas de unidad, de integración y las pruebas de aceptación constituyen los pilares de todo software desarrollado con XP.

Pruebas unitarias y de integración

Generalmente a la hora de desarrollar un sistema que debe de ser empotrado, no se cuenta con un



ambiente idóneo. Pocas veces se cuenta con un mecanismo en el hardware que informe en que parte del código existe un error. En la mayoría de las ocasiones que existe un fallo sencillamente el sistema no funciona, sin permitirle al programador saber dónde se encuentra la falta. También, sin tener en cuenta errores léxicos, sintácticos y semánticos, en el código programado pueden existir "errores conceptuales". Esto sucede cuando se viola alguna restricción relacionada con el RTOS, al interactuar con alguna interfaz de hardware o cuando se omite algún paso importante en la concepción de un algoritmo o proceso. Por todo lo mencionado anteriormente (vicisitudes experimentadas durante la implementación del *firmware*) se concluyó que es imprescindible la realización de pruebas unitarias para el desarrollo de sistemas empotrados.

Las pruebas unitarias son un tipo de pruebas caja blanca y consisten en hacer pruebas modulares mientras que se programa. Las pruebas de caja blanca son aquellas que examinan el código a evaluar, implican conocer la lógica del programa.

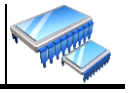
Durante todo el ciclo de desarrollo del prototipo funcional se dispuso de una herramienta que simulaba al RTOS hospedero del *firmware* en el sistema operativo Linux, es decir un ambiente estable. Esto permitió probar la totalidad del código que formaría parte del sistema. Una vez probado y confirmado el resultado, se procedía a integrar el código con el resto de la solución y seguidamente se realizaban las pruebas con el código desplegado en el hardware, logrando integrar continuamente el nuevo código al ya existente.

Una herramienta que simplificó el proceso de prueba fue el ***Subversion*** (SVN). Siempre que se lograba probar aisladamente los nuevos elementos desarrollados, estos eran integrados a la solución y al realizarle nuevamente las pruebas de integración y resultar estas satisfactorias entonces el código era almacenado en el servidor de SVN manteniendo una copia de la versión anterior almacenada. Esto posibilitó tener siempre "el software que funcionara como medida primaria del progreso", cumpliendo con uno de los principios del Manifiesto Ágil.

Al concluir cada una de las HU desarrolladas en el prototipo funcional se realizaron pruebas de aceptación donde se validó que la funcionalidad identificada en la HU se adecuara a los intereses del cliente.

Pruebas de aceptación

Las pruebas de aceptación son consideradas pruebas de caja negra. Este tipo de prueba se realiza sobre la interfaz del programa a probar, cuando, a partir de un conjunto de entradas, se comparan los resultados obtenidos con los resultados esperados. Al contrario de las pruebas de caja blanca, para aplicar pruebas de caja negra no es necesario conocer la lógica del programa, solamente cómo interactuar con el sistema a través de la funcionalidad a probar. Siguiendo la metodología XP, los clientes como primeros



beneficiarios del software desarrollado, son responsables de chequear que los resultados de estas pruebas sean correctos.

Se sometió el sistema a pruebas de caja negra para comprobar su funcionamiento. La herramienta ***CuteCom*** para GNU/Linux fue empleada para simular un cliente, y de esta forma comprobar las respuestas del *firmware* ante determinadas solicitudes. También fue utilizada la pantalla LCD como apoyo para visualizar el proceso de recolección.

Seguidamente se describirán los Casos de Pruebas y sus resultados para las Pruebas de Aceptación como conclusión del proceso de prueba:

Tabla 14 Caso de Prueba para el proceso de recolección

Caso de Prueba de Aceptación	
Código Caso de Prueba: 1	**No. Historia de Usuario:** CR-01
Descripción de la Prueba: La tarjeta debe mostrar en la pantalla LCD el avance de las muestras tomadas, y como estas reflejan el valor de la variable simulada ante un cambio en el potenciómetro. El valor establecido para el periodo de muestreo de la variable es de 1 milisegundo.	
Condiciones de Ejecución: Tener el cable de alimentación de energía conectado a la tarjeta y a la toma de alimentación de energía.	
Pasos de Ejecución: 1. Cuando el *firmware* se inicializa se muestran los datos en la pantalla. En caso de que la tarjeta no inicie la ejecución del sistema automáticamente, presionar el botón *Reset*. 2. Mover el potenciómetro con que cuenta la tarjeta.	**Entradas:**
Resultados esperados: Que varíen los valores de la variable mostrada en la pantalla LCD. Los valores deben cambiar en un intervalo de aproximadamente 50 milisegundos con un margen de error de 10 milisegundos para 50 muestras.	
Evaluación de la Prueba: Prueba insatisfactoria. La variación de los valores se realiza en un periodo mayor.	

A partir del resultado adverso de la prueba realizada se procedió a la inspección del código. Se comprobó que el mecanismo implementado para el semáforo que garantiza la exclusión mutua para el conversor AD y los accesos innecesarios al Repositorio de Variables ralentizaban la ejecución del sistema.

Luego de codificado el nuevo algoritmo, se procedió a realizar la segunda iteración de la Prueba de Aceptación y esta resultó satisfactoria.

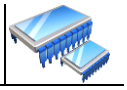


Tabla 15 Caso de Prueba para el envío del comando OpenSession

Caso de Prueba de Aceptación	
Código Caso de Prueba: 2	**No. Historia de Usuario:** CM-01
Descripción de la Prueba: Se envía un mensaje con el comando OpenSession hacia la tarjeta y esta debe responder un mensaje de respuesta ACK. La trama del mensaje está definida en un fichero de texto.	
Condiciones de Ejecución: Tener la tarjeta conectada a una PC a través del puerto serie. En la PC debe de estar corriendo el ***CuteCom*** y la tarjeta debe de estar encendida.	
Pasos de Ejecución: 1. Especificar el dispositivo a utilizar y los demás parámetros de configuración para la comunicación por el puerto serie. 2. Cargar el fichero que especifica la trama OpenSession.	**Entradas:** Trama del mensaje para el comando OpenSession.
Resultados esperados: Los datos recibidos se correspondan con la trama del mensaje de respuesta para el comando OpenSession	
Evaluación de la Prueba: Prueba satisfactoria.	

Tabla 16 Caso de Prueba para el envío del comando CloseSession

Caso de Prueba de Aceptación	
Código Caso de Prueba: 3	**No. Historia de Usuario:** CM-01
Descripción de la Prueba: Se envía un mensaje con el comando CloseSession hacia la tarjeta y esta debe responder un mensaje de respuesta ACK. La trama del mensaje está definida en un fichero de texto.	
Condiciones de Ejecución: Tener la tarjeta conectada a una PC a través del puerto serie. En la PC debe de estar corriendo el ***CuteCom*** y la tarjeta debe de estar encendida.	
Pasos de Ejecución: 1. Especificar el dispositivo a utilizar y los demás parámetros de configuración para la comunicación por el puerto serie. 2. Cargar el fichero que especifica la trama CloseSession.	**Entradas:** Trama del mensaje para el comando CloseSession.
Resultados esperados: Los datos recibidos se correspondan con la trama del mensaje de respuesta para el comando CloseSession	
Evaluación de la Prueba: Prueba satisfactoria.	

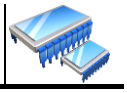


Tabla 17 Caso de Prueba para el envío del comando ReadVariable

Caso de Prueba de Aceptación	
Código Caso de Prueba: 4	**No. Historia de Usuario:** CM-01
Descripción de la Prueba: Se envía un mensaje con el comando ReadVariable hacia la tarjeta y esta debe responder un mensaje de respuesta ReadResponse. La trama del mensaje está definida en un fichero de texto para la lectura de la variable "Presión".	
Condiciones de Ejecución: Tener la tarjeta conectada a una PC a través del puerto serie. En la PC debe de estar corriendo el ***CuteCom*** y la tarjeta debe de estar encendida.	
Pasos de Ejecución: 1. Especificar el dispositivo a utilizar y los demás parámetros de configuración para la comunicación por el puerto serie. 2. Cargar el fichero que especifica la trama ReadVariable.	**Entradas:** Trama del mensaje para el comando ReadVariable.
Resultados esperados: Los datos recibidos se correspondan con la trama del mensaje de respuesta para el comando ReadVariable. Dentro de la trama se reciben las propiedades de la variable.	
Evaluación de la Prueba: Prueba satisfactoria.	

Tabla 18 Caso de Prueba para el envío del comando WriteVariable

Caso de Prueba de Aceptación	
Código Caso de Prueba: 5	**No. Historia de Usuario:** CM-01
Descripción de la Prueba: Se envía un mensaje con el comando WriteVariable hacia la tarjeta y esta debe responder un mensaje de respuesta ACK. La trama del mensaje está definida en un fichero de texto especificando la variable a la cual se le desea cambiar el valor.	
Condiciones de Ejecución: Tener la tarjeta conectada a una PC a través del puerto serie. En la PC debe de estar corriendo el ***CuteCom*** y la tarjeta debe de estar encendida.	
Pasos de Ejecución: 1. Especificar el dispositivo a utilizar y los demás parámetros de configuración para la comunicación por el puerto serie. 2. Cargar el fichero que especifica la trama WriteVariable.	**Entradas:** Trama del mensaje para el comando WriteVariable.
Resultados esperados: Los datos recibidos se correspondan con la trama del mensaje de respuesta para el comando WriteVariable y se encienda el LED especificado en la trama de escritura.	
Evaluación de la Prueba: Prueba satisfactoria.	

Las pruebas de aceptación realizadas constituyeron el colofón del proceso de V&V. De esta forma se verificó el grado de satisfacción del cliente a partir de la entrega del diseño y el prototipo funcional del

firmware para la tarjeta de adquisición basada en microcontroladores.

4.6. Conclusiones parciales

A lo largo del presente capítulo se expusieron las actividades realizadas para la validación de este trabajo. Seguidamente se sintetizan las conclusiones sobre lo expuesto en este capítulo:

- ✓ La descripción de las técnicas aplicadas aportó claridad y legibilidad a la solución propuesta.
- ✓ Los detalles del prototipo sintetizados en esta sección facilita que las futuras mejoras del *firmware* sean un proceso más intuitivo.
- ✓ Las pruebas de aceptación realizadas con la participación del cliente comprobaron la satisfacción de este y permitieron demostrar el cumplimiento de los requerimientos especificados para el *firmware*.

CONCLUSIONES GENERALES

Cumplidos los objetivos del presente trabajo podemos arribar a las siguientes conclusiones:

- ✓ Por la novedad del tema en la UCI, los métodos, técnicas y herramientas empleadas en este trabajo constituyen una guía para el desarrollo de aplicaciones empotradas.
- ✓ El sistema diseñado y las funcionalidades implementadas sientan las bases para la obtención de un producto competitivo en el mercado mundial.
- ✓ El uso de tecnologías libres permitiría economizar los recursos financieros evitando el pago de licencias de software y de esta forma se maximizarían las ganancias obtenidas por la comercialización de la solución.

RECOMENDACIONES

A partir de las experiencias obtenidas a lo largo de la investigación se proponen las siguientes recomendaciones:

1. Establecer un marco de trabajo para el desarrollo de sistemas empotrados en la Línea Sistemas Empotrados radicada en el CEDIN.
2. Continuar con el desarrollo del *firmware* para incorporar nuevas funcionalidades como:
   - ✓ Comunicación TCP.
   - ✓ Gestión de memoria.
   - ✓ Mejorar la presentación de la información en la pantalla LCD.

BIBLIOGRAFÍA REFERENCIADA

1. **Abbot, Doug. 2009.** *Embedded Linux Development using Eclipse.* Oxford : Newnes. Elseiver, 2009. 978-0-7506-8654-9.
2. **Aguayo, Paul. 2004.** *Introducción al Microcontrolador. 2004. Disponible en:.* [Documento PDF] 2004. Disponible en: http://usuarios.lycos.es/sfriswolker.
3. **Alianza Ágil. 2001.** Manifiesto por el Desarrollo Ágil de Software. [En línea] 2001. [Citado el: 2012 de 04 de 04.] http://agilemanifesto.org/iso/es/manifesto.html.
4. **Arnold, Ken. 2000.** *Embedded Controller Hardware Design.* Eagle Rock : LLH Technology Publishing, 2000. ISBN 1-878707-87-6.
5. **Barr, Michael y Massa, Anthony. 2006.** *Programming Embedded Systems.* s.l. : O'Reilly, 2006. ISBN: 0-596-00983-6.
6. **Beck, Kent. 2000.** *Extreme Programming Explained: Embrace Change.* s.l. : Addison-Wesley Professional, 2000. ISBN-13: 978-0321278654.
7. **Berger, Arnold S. 2002.** *Embedded Systems Design: An Introduction to Processes, Tools and Techniques.* Lawrence : CMP Books, 2002, 2002. ISBN: 1578200733.
8. **Calcutt, David, Cowan, Fred y Parchizadeh, Hassan. 2004.** *8051 Microcontroller an Applications based introduction.* Oxford : Newnes, 2004. ISBN 0-7506-5759-6.
9. **Cantor, Murray. 2003.** *Rational Unified Process for Systems Engineering. Part I: Introducing RUP SE Version 2.0.* [En línea] 2003. http://www.ibm.com/developerworks/rational/library/content/RationalEdge/aug03/f_rupse_mc.pdf.
10. **Charvat, Jason. 2003.** *Project Management Methodologies — Selecting, Implementing and Supporting Methodologies and Processes for Projects.* Hoboken : John Wiley & Sons, Inc., 2003. ISBN 0-471-22178-3.
11. **Jiménez González, Daniel J., y otros. 2010.** CID 300-9: Procesador orientado a aplicaciones médicas. *Bioingeniería y Física Médica Cubana*. Enero - Abril 2010, 11 (1). ISSN: 1606-0563.
12. **Clements, Paul. 1996.** *A Survey of Architecture Description Languages.* Schloss Velen, Alemania : Proceedings of the International Workshop on Software Specification and Design, 1996.
13. **Cockburn, Alistair. 2000.** *Agile Software Development. Draft version. 3b.* s.l. : Addison-Wesley, 2000. ISBN 0-201-69969-9.
14. **Cohen Manrique, Carlos. 2005.** *Diferencias entre microcontroladores y microprocesadores.* Sincelejo: Corporación Universitaria del Caribe, 2005.
15. **Crisp, John. 2004.** *Introduction to Microprocessors and Microcontrollers. 2nd Ed.* Oxford : Newnes, 2004. ISBN 0-7506-5989-0.

16. **Data-Acquisition. 2011.** Guide to Data Acquisition. [En línea] 1 de Mayo de 2011. [Citado el: 6 de Marzo de 2012.] http://www.data-acquisition.us/.

17. **Dell Inc.** Resources Glossary. [En línea] [Citado el: 30 de 03 de 2012.] content.dell.com/es/es/domestica/vsl-resources-glossary.aspx.

18. **Dhadiwal Baid, Shweta. 2010.** *Microcontrollers, Leading the Way in Innovations.* [Documento PDF] s.l. : EFY Report, 2010. Disponible en: http://www.efymag.com.

19. **Douglass, Bruce P. 2004.** *Real Time UML: Advances in The UML for Real-Time Systems. 3rd Edition.* s.l. : Addison Wesley, 2004. 0-321-16076-2.

20. **—. 2002.** *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time.* s.l. : Addison Wesley, 2002. ISBN: 0-201-69956-7.

21. **—. 1999.** *ROPES: Rapid Object-Oriented Process for Embedded Systems.* [Whitepaper] Los Angeles : I-Logix Inc., I-Logix Inc., 1999.

22. **—. 2009.** *UML for the C programming language.* s.l. : IBM Rational Software, 2009.

23. **Eloranta, Veli-Pekka, y otros. 2009.** *Patterns for Distributed Embedded Control System.* Tampere. Finlandia : Tampere University of Technology. Department of Software Systems, 2009.

24. **Fowler, Martin. 2003.** *The New Methodology.* s.l. : ThoughtWorks, 2003.

25. **Ganssle, Jack G. 2004.** *A Firmware Development Standard v1.2.* Baltimore : Ganssle Group, 2004.

26. **Harper, Douglas. 2010.** Online Etymology Dictionary. [En línea] 2010. [Citado el: 6 de Marzo de 2012.] www.etymonline.com/.

27. **Heath, Steve. 2003.** *Embedded Systems Design. 2nd Ed.* Oxford : Newnes, 2003. ISBN 0-7506-5546-1.

28. **Imperial College Department of Computing. 2010.** FOLDOC Computing Dictionary. [En línea] LLC, 2010. [Citado el: 6 de Marzo de 2012.] http://foldoc.org/firmware.

29. **INTECO. 2008.** *Guía avanzada de Gestión de Requisitos.* 2008.

30. **—. 2009.** *Guía de Validación y Verificación.* 2009.

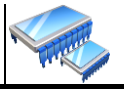
31. **Joskowicz, José. 2008.** Reglas y Prácticas en eXtreme Programming. 02 de Mayo de 2008. Trabajo realizado en el marco de la asignatura "Nuevas Técnicas de Desarrollo de Software en Ingeniería Telemática", del Doctorado de Ingeniería Telemática. Universidad de Vigo, España.

32. **Juristo, Natalia, Moreno, Ana María y Vegas, Sira. 2005.** *Técnicas de Evaluación de Software.* 2005.

33. **Kalra, Alka y Kalra, Sanjeev Kumar. 2010.** *Architecture and Programming of 8051 Microcontroller.* Nueva Delhi : University Science Press, 2010. ISBN-13: 9789380386317.

34. **Kernighan, Brian W. y Ritchie, Dennis M. 1991.** *El lenguaje de programación C.* s.l. : Pearson Educación, 1991. 9688802050.

35. **Laplante, Phillip A. 2004.** *Real-Time System: Design and Analysys: an engineer's handbook. 3th Ed.* Hoboken : IEEE Press, 2004. ISBN 0-471-22855-9..



36. **Rushinek, Avi y Rushinek, Sara F. 1985.** Performance monitoring tools for computer systems: Hardware, software, firmware and hybrid controls. *Computers & Industrial Engineering*, 9 (4), Elsevier Ltd., 1985. Disponible en: http://www.sciencedirect.com/science/article/pii/0360835285900294.
37. **Pressman, Roger S. 2005 .** *Ingeniería de Software. Un enfoque práctico. 6ta Ed.* Madrid : McGraw-Hill, 2005 . ISBN: 9701054733.
38. **Punkka, Timo. 2005.** *Agile Methods and Firmware Development.* Helsinki : Helsinki University of Technology, Software Business and Engineering Institute, 2005.
39. **Qing, Li. 2003.** *Real-time concepts for embedded systems.* Lawrence : CMP Books, 2003. ISBN: 1-57820-124-1.
40. **Real Academia Española.** Diccionario de la Lengua Española. [En línea] [Citado el: 2 de Abril de 2012.] http://buscon.rae.es/draeI/SrvltGUIBusUsual?TIPO_HTML=2&TIPO_BUS=3&LEMA=metodolog%C3%ADa.
41. **Rumbaugh, James, Jacobson, Ivar y Booch, Grady. 1998.** *UML. Manual de referencia.* s.l. : Addison Wesley, 1998.
42. **Sommerville, Ian. 2005.** *Ingeniería de Software, 7ma Ed. .* Madrid : Addison Wesley, 2005.
43. **Stallings, William. 2006.** *Sistemas Operativos: aspectos internos y principios de diseño.* Madrid : Pearson, Prentice Hall, 2006. ISBN: 8420544620.
44. **STMicroelectronics. 2009.** *UM0600 User Manual. STM3210C-EVAL evaluation board.* [Documento PDF] s.l. : STM, 2009.
45. **TIOBE Software BV. 2012.** TIOBE Software BV . [En línea] TIOBE Software BV, 2012. [Citado el: 10 de 05 de 2012.] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.
46. **Valdés Pérez, Fernando E. y Pallas Areny, Ramón. 2007.** *Microcontroladores: Fundamentos y Aplicaciones con PIC.* Barcelona : MARCOMBO, S.A., 2007. ISBN 84-267-14XX.
47. **Webster's Online Dictionary.** Webster's Online Dictionary. *Trie.* [En línea] [Citado el: 21 de Mayo de 2012.] http://www.websters-online-dictionary.org/definitions/TRIE?cx=partner-pub-0939450753529744%3Av0qd01-tdlq&cof=FORID%3A9&ie=UTF-8&q=TRIE&sa=Search#906.
48. **Williams, Rob. 2006.** *Real-Time Systems Development.* Oxford : Elseiver, 2006. 0-7506-6471-1.
49. **Woodside, Murray y Petriu, Dorina . 2004.** *Capabilities of the UML Profile for Schedulability Performance and Time (SPT).* Ottawa : Carleton University, 2004.
50. **Zurrell, Kirk. 2000.** *C Programming for Embedded Systems.* Lawrence : R&D Books. CMP 1-929629-04-4, 2000. ISBN 1-929629-04-4.

BIBLIOGRAFÍA CONSULTADA

1. **Ballesteros Jordán, Freddy Marcelo. 2007.** *Adquisición de datos de un sistema maestro– esclavo utilizando microcontroladores mediante comunicación serial para "M&B Automatización".* Ambato : s.n., 2007. Tesis de grado, Universidad Técnica de Ambato, Ecuador.

2. **Barry, Richard. 2009.** *FreeRTOS Manual Reference.* [Documento PDF] s.l. : Real Time Engineers Ltd., 2009.

3. **—. 2009.** *Using the FreeRTOS Real Time Kernel. A practical guide.* [Documento PDF] s.l. : Rea, Real Time Engineers Ltd., 2009.

4. **Bedón Salazar, Juan Carlos y León Guerrero, Iván Rodrigo. 2009.** *Diseño y construcción de un PLC modular a base de microcontroladores.* Latacunga : s.n., 2009. Tesis de grado, Escuela Politécnica del Ejercito, Ecuador.

5. **Cantor, Murray. 2003.** Organizing RUP SE projects. [En línea] 2003. http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jul03/m_organizing_mc.pdf.

6. **Ganssle, Jack G. 2004.** *The firmware handbook.* Oxford : Newnes, 2004. ISBN: 0-7506-7606-X.

7. **IEEE. 1990.** *610.12-1990 Standard Glossary of Software Engineering Terminology.* Washington DC : IEEE Press, 1990.

8. **Jiménez Ríos, Félix Vicente y Rivero Juárez, Joaquín. 2006.** *Diseño y construcción de una tarjeta programable de adquisición, procesamiento de datos y control.* Cuernavaca : CENIDET, 2006. Tesis de Maestría, Centro Nacional de Investigación y Desarrollo Tecnológico, Cuernavaca, México.

9. **Möckel, Rico. 2010.** *How to design and implement firmware.* s.l. : BIOROB. EPFL Biorobotic Laboratory, 2010.

10. **Patrick, Dale R. y Fardo, Stephen W. 2009.** *Industrial Process Control System. 2nd Ed.* Lilburn : The Fairmont Press Inc., 2009. ISBN-13: 0-88173-592-2.

11. **Kruchten, Philippe. 1995.** Planos Arquitectónicos: El Modelo de "4+1" Vistas de la Arquitectura del Software*. IEEE Software.* 1995, 12 (6).

12. **Reynoso, Carlos Billy. 2004.** *Introducción a la Arquitectura de Software.* [Documento PDF] Buenos Aires : Universidad de Buenos Aires, 2004.

13. **STMicroelectronics. 2009.** *UM0600 User Manual. STM3210C-EVAL evaluation board.* [Documento PDF]. STM, 2009.