Revista Iberoamericana de Automática e Informática Industrial
CEA IFAC comité español de automática
RIAI
ISSN: 1697-7912. Vol. 3, Núm. 2, Abril 2006, pp. 61-70
http://riai.isa.upv.es

rtfCANopen: IMPLEMENTACIÓN DEL BUS CAN PARA SISTEMAS EMPOTRADOS

Javier Portillo, Marga Marcos, Aitor Olarra, Itziar Cabanes

Escuela de Ingenieros de Bilbao (Universidad del País Vasco)
Alameda Urkijo s/n 48013 Bilbao
jtppobej@bi.ehu.es, jtpmamum@bi.ehu.es, aolarra@tekniker.es, jtpcaaxi@bi.ehu.es

Resumen: Este artículo describe el diseño y la implementación (hardware y software) de *rtfCANopen*, un sistema de comunicación basado en el bus CAN y adaptado a las necesidades de los sistemas empotrados. Se discuten las necesidades que impone el desarrollo de sistemas distribuidos de tiempo real al sistema de comunicación y, partiendo de las conclusiones obtenidas, se diseña rtfCANopen. Entre las características de esta implementación cabe destacar: se basa en el protocolo de alto nivel estándar CANopen; requiere de pocos recursos; tiene capacidades de descarga, configuración y puesta en marcha automática de los nodos del sistema distribuido; permite que el ingeniero de control se abstraiga de los problemas inherentes a las comunicaciones. Además, rtfCANopen, aunque puede usarse de forma independiente, encaja en un proceso de desarrollo coherente soportado por el conjunto de herramientas RTF (*Real Time Framework*) con lo que aspectos como el cálculo de tiempos de ejecución de peor caso y su uso en el análisis, la asignación de prioridades a mensajes ó el empaquetado de señales en mensajes se tratan de forma global y coherente. Como caso de estudio se aplica rtfCANopen en el control de un robot móvil autónomo.

Palabras Clave: buses de campo, sistemas de control distribuido, sistemas basados en microprocesador.

1. INTRODUCCIÓN

El protocolo de comunicación CAN (Controller Area Network) se desarrolló inicialmente para su aplicación en la industria del automóvil (CAN, 1991) aunque, posteriormente, se ha utilizado en multitud de aplicaciones industriales. Entre sus características cabe destacar:

- Fiabilidad a interferencias electromagnéticas
- Bajos tiempos de latencia
- Soporte de los modelos Cliente/Servidor y Productor/Consumidor
- Capacidad *multi-maestro* (cualquier nodo puede iniciar la comunicación)
- Baja probabilidad de errores residuales

A estas características cabe añadir el comportamiento temporal determinista del bus CAN (Tindell y Burns, 1994), ya que 'es posible' acotar el tiempo de respuesta de peor caso de todos los mensajes de una red CAN. Esta propiedad de CAN lo hace muy apropiado para el desarrollo de sistemas distribuidos con requisitos temporales. Sin embargo, el mero hecho de utilizar el protocolo CAN no asegura el comportamiento determinista del sistema o el cumplimiento de requisitos temporales. Para convertir en realidad el 'posible' comportamiento determinista del sistema se requiere de ciertos niveles de abstracción y del seguimiento de un proceso estricto en el diseño y desarrollo de las comunicaciones.

Es más, el uso extensivo de CAN (especialmente en el mundo de la automoción) ha provocado que afloren otros requisitos que se deben satisfacer en dicho proceso de desarrollo (Johansson, *et al*,. 2000):

- Bajo coste. Los sistemas eléctricos representan hoy en día el 30% del coste total de un vehículo.
- Comportamiento ante cargas elevadas. El crecimiento en el número de nodos (ECUs, *Electrical Control Units*) que comparten el bus hace necesario predecir los retardos máximos que puede sufrir cada mensaje para dar al integrador de sistemas un control total sobre el comportamiento de la red.

– Configuración y actualización global y automática. La configuración manual independiente de cada dispositivo es muy ineficiente
– Herramientas de soporte. Deben existir herramientas coherentes con la filosofía del proceso y que automaticen las labores relacionadas.

En resumen, se impone la adopción de un proceso de desarrollo adecuado que utilice eficientemente los recursos disponibles y que asegure el cumplimiento de requisitos temporales por concepto y diseño. El hardware y software empleados en el sistema de comunicación, objeto del presente artículo, deben ser coherentes con la especificación y diseño del sistema, los análisis y simulaciones temporales que se realicen y la generación automática de código si la hubiera. En este sentido, los autores trabajan en la integración de diferentes herramientas de modelado, especificación, análisis temporal (Marcos y Portillo, 2000) y generación de código (Marcos, *et al*, 2000; Portillo y Marcos, 2001) en un único entorno. Este entorno de herramientas se denomina RTF (*Real Time Framework*) y está dedicado al desarrollo de sistemas de control distribuidos con requisitos de tiempo real.

Por lo tanto, el sistema de comunicación será una de las piezas que deba encajar en el conjunto. Su nombre, *rtfCAN*, indica que se basa en el protocolo CAN pero que se concibe dentro de RTF, de forma que los sistemas especificados, modelados, diseñados, analizados y el código generado en RTF sea directamente ejecutable sobre el hardware del sistema de comunicación rtfCAN y con el software de rtfCAN. Esta integración en un proceso completo para el desarrollo de sistemas distribuidos con requisitos de tiempo real plantea los siguientes objetivos para rtfCAN:

– Protocolo de alto nivel adecuado para la filosofía de desarrollo
– Toma de decisiones de diseño acordes a los criterios de optimización en el uso de recursos y cumplimiento de restricciones temporales. Ejemplos: algoritmo de asignación óptima de prioridades para los mensajes, selección del conjunto de señales empaquetadas en cada mensaje,…
– Conocimiento de tiempos de ejecución de peor caso de todas las rutinas
– Diseño modular del software que aísle los aspectos de la comunicación de los del control
– Descarga de código, configuración y puesta en marcha automática de todos los nodos

En definitiva, se plantea el desarrollo de un sistema de comunicación (hardware y librería de comunicación software) basado en el bus CAN apropiado para su integración en el proceso formal de desarrollo de Sistemas de Control Distribuidos con requisitos de Tiempo Real soportado por el conjunto de herramientas RTF. No obstante, este sistema de comunicación podrá ser empleado por cualquier desarrollador con independencia de RTF.

El siguiente apartado presenta trabajos de otros autores con objetivos similares a los de rtfCAN (definición de sistemas de comunicación coherentes con procesos de desarrollo) y se discute sobre la conveniencia de adoptar un protocolo de alto nivel estándar. En los apartados 3 y 4 se describe el hardware y software del sistema de comunicación desarrollado, puntualizándose con mayor detalle en el apartado 5 la funcionalidad de descarga, configuración y puesta en marcha automática. El apartado 6 ilustra la utilización del sistema de comunicación, junto con el resto de herramientas del entorno, en la generación de un sistema concreto de control: un robot móvil autónomo. El apartado 7 ubica el papel de rtfCAN como una pieza más necesaria dentro del proceso de construcción de un sistema de control distribuido. Finalmente, se enumeran algunas conclusiones y trabajo futuro.

2. TRABAJO RELACIONADO

2.1 Conceptos de sistemas de comunicación

Se han realizado algunos esfuerzos en dirección a garantizar, desde las primeras etapas del diseño, la corrección de sistemas distribuidos sobre el bus CAN. En (Johansson, *et al*, 2000) se apuntan algunas propiedades necesarias para que un sistema de comunicación lo permita:

– Niveles de abstracción para el proceso de desarrollo
– API (*Application Programming Interface*) a nivel de señales
– Comportamiento temporal predecible
– Construcción de sistemas verificables en base a unidades verificadas

En esta línea, se pueden apuntar las aproximaciones de Volcano y OSEK. El sistema de software Volcano garantiza unas comunicaciones predecibles a través de los modelos matemáticos desarrollados en (Tindell y Rajnak, 1998) y basados en DMA (*Deadline Monotonic Analysis*). Volcano asegura la llegada de todas las señales dentro de plazos, así como la asignación de prioridades y empaquetamiento óptimo de señales en mensajes CAN para lograr el rendimiento deseado con las necesidades mínimas de potencia de cómputo. Se ha aplicado con éxito en el desarrollo de varias familias de vehículos Volvo (Specks y Rajnák, 1998). Por su parte, la especificación del sistema operativo OSEK (OSEK/VDX, 2004) engloba la descripción de un Sistema Operativo, un Sistema de Comunicación independiente de protocolos y un Sistema de Gestión de Red. OSEK no está ligado al protocolo CAN ni a ningún otro en concreto, puesto que pretende servir

como marco estructural para cualquier sistema distribuido de la industria del automóvil.

Ahora bien, aunque en ambos casos se propone el desarrollo de un sistema de comunicación para sistemas empotrados, uno de ellos es propietario (Volcano es propiedad intelectual de la empresa Volcano Automotive) y el otro (OSEK) resulta ser un estándar abierto pero demasiado genérico, y además no contempla las peculiaridades de CAN. Por otra parte, ambos están totalmente centrados en las necesidades de la industria del automóvil.

2.2 Protocolos de alto nivel

El estándar de CAN sólo define hasta la capa de enlace ó nivel 2 (Modelo de Referencia OSI de ISO). Por tanto, es abierto a protocolos que regulen el nivel de aplicación y el estándar no especifica nada acerca de temas tales como la gestión de red, la planificación de mensajes ó el formato de los datos. Esto significa que se puede construir 'a medida' un protocolo de alto nivel concreto sobre la capa de enlace CAN.

Este enfoque flexible ha permitido a CAN adaptarse a campos muy diferentes. De hecho, los fabricantes de dispositivos CAN se han unido en diferentes asociaciones para definir protocolos estándar de alto nivel: CAL/CANopen de la asociación *Can in Automation* (CiA), DeviceNet de Open DeviceNet Vendor Association (ODVA), SDS de Honeywell, etc. Además, en algunas de estas asociaciones se han desarrollado perfiles concretos dirigidos a un sector específico de mercado, como la industria textil, el sector de automoción, etc.

En la definición del sistema de comunicación rtfCAN, objeto del presente trabajo, se planteó una disyuntiva entre adoptar un protocolo de alto nivel estándar ó definir uno propio que respondiera a ciertos requisitos específicos.

Es habitual la adquisición de librerías en código objeto (en código fuente resultan muy caras) de implementaciones acordes con protocolos de alto nivel. Los beneficios de adoptar un protocolo de alto nivel estándar se pueden resumir en los siguientes:

- interconexión e *intercambiabilidad* de productos de diferentes fabricantes
- sencilla conexión y configuración de nuevos dispositivos
- disponibilidad de servicios estándar para el arranque y configuración de la red

Por su parte, la creación de un protocolo de alto nivel particular permite una flexibilidad total para optimizar la asignación de prioridades, el mapeo de señales a mensajes y la gestión de red, haciendo coherente todo ello con los estudios temporales que se apliquen. Además, sólo un conocimiento exhaustivo de las capas de abstracción software empleadas permite calcular con rigurosidad los tiempos de ejecución de peor caso para la transmisión y recepción de mensajes. La utilización de software construido por terceros dificulta la medida de estos tiempos (especialmente si no se tiene el código fuente).

En definitiva, ante la disyuntiva planteada en cuanto a la elección del protocolo de alto nivel, se adoptó una solución intermedia o de compromiso entre ambas propuestas para beneficiarse de las ventajas de ambas. Por tanto, no se desarrolló desde cero el protocolo de alto nivel, sino que se optó por basarse en la especificación de CANopen (CiA, 1996a) para construir una implementación particular que mantuviera las ventajas de basarse en el estándar CANopen ofreciendo a la vez el control absoluto sobre las comunicaciones. De esta forma, es posible medir tiempos de respuesta, así como modificar el protocolo de alto nivel de acuerdo a las necesidades del proceso software que se siga. Por tanto, el nombre completo del sistema de comunicación resulta rtfCANopen.

CANopen (CiA, 1996a) define varios tipos de mensajes para tratar con datos de diferente naturaleza:

- PDO (*Process Data Object*), datos de tiempo real
- SDO (*Service Data Object*), datos de configuración de los nodos
- NMT (*Network Management Service*), datos de gestión de red

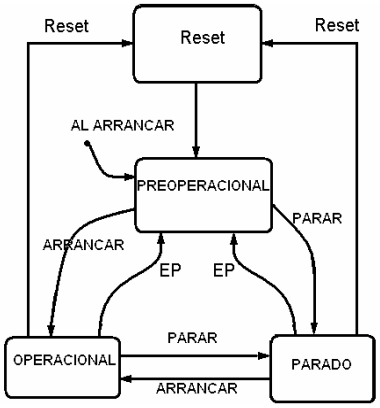

Figura1. Estados NMT de los nodos CANopen (EP, Entrar en Preoperacional)

Los servicios NMT (CiA, 1996b) proporcionan mecanismos de control sobre el estado de cada nodo. La fig. 1 muestra los posibles estados del nodo y cómo los comandos NMT permiten pasar de un estado a otro. También es interesante comentar la función del mensaje SYNC, que permite el funcionamiento síncrono de los PDO. Sólo el nodo maestro NMT puede enviar mensajes de sincronismo, y los PDO sincronizados sólo podrán ser transmitidos después de un mensaje SYNC (o un número múltiplo de ellos). De esta forma, un nodo puede ser programado para enviar un mensaje PDO con datos en tiempo real una vez recibidos un

número ‘n’ concreto de mensajes de sincronismo, donde ‘n’ depende de la frecuencia con la que estos datos se refrescan.

3. DESCRIPCION DEL HARDWARE

Los nodos rtfCANopen, se han diseñado e implementado para formar parte de una red CANopen empotrada, integrada en el prototipo de un robot móvil (Portillo, *et al*., 2002). Las librerías de comunicación rtfCANopen controlarán y gestionarán las comunicaciones en estos nodos, mientras que el código específico de las aplicaciones de control se encargará de gobernar los dispositivos de entrada y salida conectados a cada nodo CAN. Los nodos diseñados se basan en los microcontroladores PIC16F87X (Microchip, 1998) tal y como se muestra en la fig. 2. Éstos utilizan el interfaz SPI (*Serial Peripheral Interface*) para comunicarse con un controlador CAN MCP2510 (Microchip, 1999) y acceder físicamente a la red por medio de un *transceiver* PCA82C250 (Philips, 1997).

Las principales características técnicas de los microcontroladores empleados son (concretamente se utilizan el 876 y el 873 en el caso del robot móvil):

- CPU de 8 bits y 5 MIPS a 20Mhz
- Hasta 8K x 14 palabras de memoria de programa tipo FLASH
- Hasta 368 x 8 bytes de memoria RAM
- Hasta 256 x 8 bytes de memoria EEPROM
- Tres contadores, interfaz SPI, USART y conversor analógico/digital.

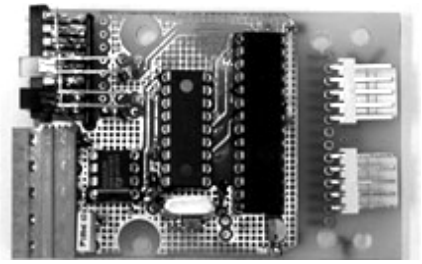

Figura 2. Nodo rtfCANopen.

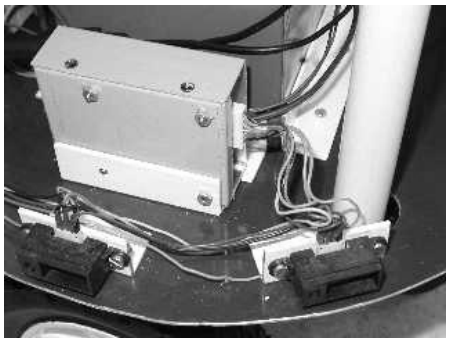

Figura 3. Nodo rtfCANopen en red empotrada.

El controlador CAN gestiona los mensajes transmitidos y recibidos. Este dispositivo dispone de filtros de recepción (configurados por el PIC) y disparará una interrupción dirigida al PIC cuando se reciba un mensaje esperado.

Los parámetros de un nodo básico (velocidad de red, identificador del nodo y la velocidad del microprocesador) se establecen manualmente con interruptores externos. Un LED indicará el estado actual del nodo conforme al estándar CANopen (pre-operacional, operacional o parado).

Los nodos se protegen con una caja, dejando acceso a los interruptores, entradas / salidas y conectores CAN para que sea posible su instalación en la red. La fig. 3 muestra un nodo rtfCANopen que envía datos a los sensores infrarrojos de proximidad a través de la red. Cabe destacar el bajo coste económico de cada nodo.

4. ARQUITECTURA SOFTWARE

El resumen técnico de la implementación CANopen desarrollada para los nodos es la siguiente (acorde con el formato estándar de CiA):

- Servicios NMT: Esclavo
- Control de error: No
- ID de los nodos: HWswitch
- Nº PDOs: 2 de recepción y 2 de transmisión
- Modos de PDO: Sync (cíclico) y petición remota
- PDO linking: No
- Mapeo de PDO: Dinámico
- Nº de SDOs: 1 como servidor ó cliente
- Mensaje de emergencia: No
- Versión CANopen: DS 301 V3.0 Communication Profile for Industrial Systems
- Framework: DSP 302 V3.0 Framework for Programmable CANopen Devices.
- Memoria FLASH ocupada: 3KB (Librería rtfCANopen)

Considerando que se busca la abstracción y el diseño modular para aislar los aspectos de comunicación y aplicación, el software que ejecutan los microcontroladores PIC está separado en dos módulos. El módulo de comunicación (COM) es el mismo para todos los nodos: configura y gestiona el MCP2510 y se encarga de las comunicaciones a través del bus CAN de acuerdo al estándar de CANopen. Por otro lado, el código de aplicación (APP) es diferente en cada nodo y lleva a cabo la funcionalidad del mismo. Por último, y dado que los dispositivos conectados a cada nodo son diferentes, las labores de configuración también serán distintas.

Los módulos COM y APP se coordinan a través de una interfaz estándar, ya que el módulo COM es siempre el mismo y tiene que colaborar con diferentes códigos de aplicación. En la fig. 4 se muestra esta interfaz estándar, donde las flechas sombreadas muestran el flujo de datos, las líneas curvas señalan las interrupciones y las líneas simples las llamadas entre módulos. Las llamadas realizadas por el módulo COM encontrarán puntos de entrada

estándar en algún módulo APP. Es decir, el módulo APP se utiliza siempre de la misma forma.

El módulo COM está formado por varios sub-módulos (ver fig. 5):

- ISR (*Interrupt Service Routine*). Este módulo se encarga de procesar las interrupciones. Las interrupciones pueden ser debidas a los dispositivos conectados al nodo, en cuyo caso ejecuta la entrada *appISR* del módulo APP, como se observa en la figura 5, o pueden ser generadas por el controlador CAN con la recepción un mensaje. En este caso, ISR decide qué submódulo de COM debe ejecutar en función del tipo de mensaje recibido.
- NMT (*Network Management*). En este módulo se implementa la máquina de estados finitos que controla la operación del nodo, según lo establecido en el estándar CANopen (CiA, 1996b).
- PDO (*Process Data Object*). Gestiona los mensajes de tipo PDO y SYNC.
- SDO (*Service Data Object*). Gestiona los mensajes que se utilizan para configurar el nodo, según lo establecido en el estándar CANopen.
- SPI. Rutinas para acceder al controlador CAN a través de la interfaz de periféricos serie.
- EEPROM. Rutinas para acceder a la EEPROM en el PIC.

El módulo APP lo constituyen los siguientes sub-módulos:

–OBJ. Estructuras de datos que definen los objetos implementados en el nodo.

–APP. Código para acceder a los datos de la aplicación concreta. También realiza la configuración y el control de los dispositivos empleados por la aplicación.

–

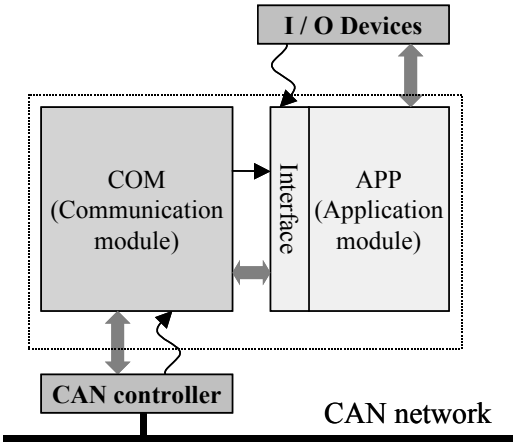

Figura 4. Arquitectura Software (nivel 1).

Por lo tanto, ISR gestiona todas las interrupciones del microcontrolador provenientes tanto de los dispositivos como del controlador CAN. Dado que la relación entre los módulos ISR y APP se restringe a llamar a *appIsr* cuando el origen de la interrupción no es el controlador CAN, ISR ha sido incluida en el módulo COM.

La estructura propuesta facilita la codificación de un módulo APP que haga uso y se coordine con la librería rtfCANopen (módulo COM). El único requisito es respetar la interfaz estándar mostrada en la figura 5 (permite la comunicación con los sub-módulos NMT, PDO y SDO) y el punto de entrada *appIsr* (permite sincronizar los dispositivos y aplicaciones específicas).

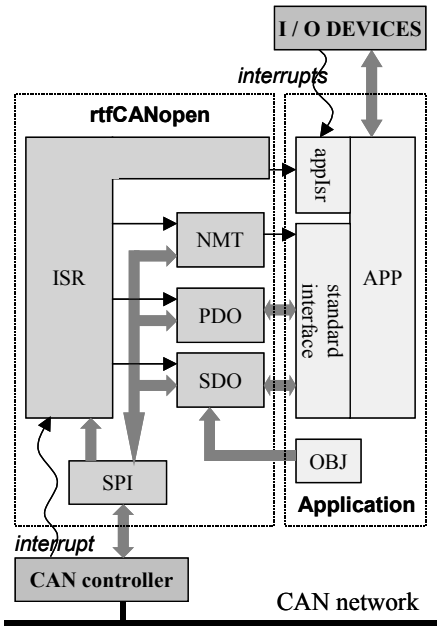

Figura 5. Arquitectura Software (nivel 2).

5. DESCARGA, CONFIGURACIÓN Y PUESTA EN MARCHA AUTOMÁTICA

En principio, la programación del software de aplicación debería hacerse siguiendo los siguientes pasos: extraer el nodo de la red empotrada, sacar el micro del nodo y utilizar los dispositivos específicos para la programación de la memoria Flash. Estas operaciones deberían realizarse de forma individual cargando un módulo APP diferente en cada uno de los nodos de la red. De igual forma, habría que configurar adecuadamente por separado cada nodo. Por último, cuando se precisara cambiar el código del módulo COM (común a todos los nodos) también habría que reprogramar manualmente todos los dispositivos.

Teniendo en cuenta que rtfCANopen está expresamente diseñado para su uso en sistemas empotrados, todas estas operaciones serían muy costosas en tiempo y serían origen de muchos errores. En algunos sistemas, incluso, sería imposible desmontar todos los dispositivos. Por estas razones, el sistema de comunicación rtfCANopen dispone de las siguientes capacidades:

–**Descarga automática**. Es capaz de actualizar de forma global el software de todo el sistema, tanto el software de comunicación como el de

aplicación de cada nodo, sin necesidad de extraer los nodos. Se emplea la propia red CAN y los recursos de rtfCANopen para descargar, desde un nodo maestro (a estos efectos), la parte de aplicación específica de cada nodo (APP) y la librería de comunicación común a todos (COM).

– **Configuración y puesta en marcha automática.** El mismo nodo maestro mencionado anteriormente dispone de toda la información necesaria para configurar, a través de mensajes SDO, cada uno de los nodos. Finalmente, desde este nodo se inicializará el funcionamiento programado para cada uno de los nodos y el conjunto de la red, lanzándose la ejecución de la aplicación de control distribuida.

Para desarrollar estas capacidades es necesario centralizar todos los módulos a descargar y la información de configuración en el nodo maestro, así como disponer de la funcionalidad de auto-programación en cada uno de los nodos de la red.

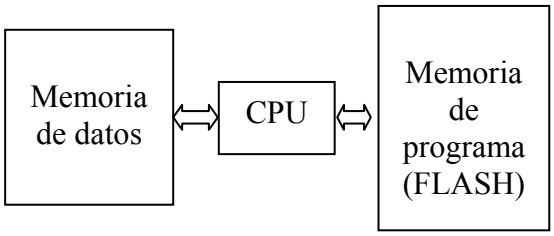

Figura 6. Arquitectura de los microcontroladores PIC.

Los microcontroladores PIC son de arquitectura HARVARD por lo que tienen la memoria de datos y la de programa separadas, como se ve en la fig. 6. En esta arquitectura, los datos y el programa no comparten memoria, lo que impide en un principio su comunicación y fuerza a realizar de forma diferente la transferencia del programa al micro. Para programar o escribir en la memoria de programa hay que utilizar un dispositivo programador dedicado. En un intento por paliar esta limitación es habitual que los microcontroladores con memoria de programa FLASH incorporen una funcionalidad con la que poder aceptar nuevo código por uno de los canales estándares de comunicación y, una vez almacenado éste en memoria de datos, transferir el código a la memoria de programa. Los microcontroladores empleados en la implementación de los nodos (PIC16F87X) incorporan esta funcionalidad a través de un juego de registros comunes para el acceso a la memoria EEPROM (datos) y a la memoria FLASH (programa). El empleo de esta funcionalidad permite desarrollar la descarga automática del código de aplicación (módulo APP) para cada nodo.

En definitiva, la funcionalidad de descarga automática se consigue manteniendo residente en la memoria de programa de los nodos el código mínimo que permite recibir mensajes a través de la red CAN, extraer de ellos los módulos APP y COM (cada vez que se actualice la versión de rtfCANopen) y auto-programar el nodo (ubicando los módulos en la memoria de programa para su ejecución subsiguiente).

Las entradas del diccionario de objetos que se recomiendan en (CiA, 1998) para la implementación de un nodo CANopen que pueda actualizar su software, son las siguientes:

– [1F50h,xxh]: *Download program data*. Este objeto se usa para aceptar directamente el código a programar en la memoria. El subíndice 0 se usa para conocer cuántos programas distintos o áreas de memoria disponibles hay y los demás subíndices se usan para recibir el código siendo cada subíndice el correspondiente a un área de memoria o a un programa.

– [1F51h,xxh]: *Program control object*. Este objeto se usa para el control del programa cargado o bajado en el objeto [1F50h, xxh]. Los comandos esenciales a implementar son *Start program*, que requiere la escritura de un 1 en el objeto y el comando *Stop program*, que requiere la escritura de un 0 en el objeto.

La fig. 7 muestra los pasos básicos que el software residente debe seguir, cumpliendo el protocolo CANopen. Este código será lo primero en ejecutarse tras iniciar el nodo.

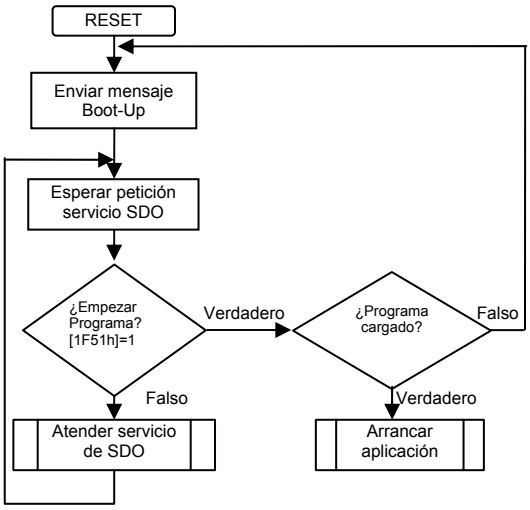

Figura 7. Diagrama de flujo del código residente.

Una vez que empieza a ejecutarse, debe enviar el mensaje de Boot-Up de CANopen para informar a los otros nodos y al maestro de la red (en aspectos de configuración) que está inicializado, entonces activará el servidor SDO para atender adecuadamente todos los servicios o peticiones SDO que se requieran. Esto incluye esperar una nueva configuración y actualización del software mientras el objeto [1F51h,xxh] valga 0, así como una petición de empezar el programa de la aplicación con la escritura de un 1 en el objeto [1F51h,xxh]. Antes de llamar o de saltar al código de la aplicación tendrá que verificar si hay algún programa cargado en memoria.

6. CASO DE USO: ROBOT MÓVIL AUTÓNOMO

6.1 Descripción del prototipo

El sistema de comunicaciones CAN desarrollado se ha utilizado en la construcción del robot móvil autónomo que ilustra la figura 8. Los servos, sensores infrarrojos, motores, sónar, LCD y teclado se controlan empleando los nodos micro-controladores diseñados. Una CPU PC-104 a través de una tarjeta comercial CAN (ESD, 1998) configura tanto la red como los nodos microcontroladores, además de coordinar y supervisar el sistema global.

El objetivo es controlar los movimientos del robot conforme a cuatro conductas de comportamiento: 'explorar el entorno', 'evitar obstáculos', 'seguir línea' y 'control remoto'. El control del robot está basado en las cuatro conductas, cada una de ellas con un nivel de activación específico en cada momento. Por tanto, todas las conductas están siempre presentes pero con diferentes intensidades. El movimiento final del robot es el resultado de la suma ponderada de las conductas, en función de su nivel de activación. Una descripción más detallada del diseño del sistema de control puede encontrarse en (Portillo *et al*., 2002).

6.2 Especificación de la aplicación en RTF

Como ya se ha mencionado anteriormente, el proceso de desarrollo de la aplicación se ha realizado en RTF (Marcos y Portillo, 2000; Marcos, *et al.*, 2000; Portillo y Marcos, 2001), un entorno para construir aplicaciones distribuidas en tiempo real basadas en el bus CAN. RTF abarca las diferentes fases involucradas en la creación de un Sistema de Control Distribuido en Tiempo Real, desde la especificación hasta la generación de código. RTF se fundamenta en el uso de Matlab y *toolboxes* relacionadas (Mathworks, 1999a, 1999b, 1999c, 1999d). Las características ofrecidas por *Simulink* y *Stateflow* se han completado con nuevos bloques, agrupados en la librería RTF, que permiten especificar el sistema distribuido completo.

En el nivel superior, el usuario define la arquitectura general del sistema distribuido en términos de los nodos que se conectarán a través del bus CAN y los enlaces de comunicación. Para ello, se hace uso de un conjunto de bloques de librería que han sido integrados como nuevos bloques de *Simulink*. Los nodos pueden tener capacidad de procesamiento (CPUs donde se ejecutarán los algoritmos de control definidos por el usuario), o bien pueden ser nodos de entrada/salida (dispositivos que conectan los sensores y actuadores a la red CAN). La figura 9, muestra el nodo inteligente (CPU-104) en el que residen las tareas de control y los nodos CAN de entrada/salida que se conectan a los sensores y actuadores.

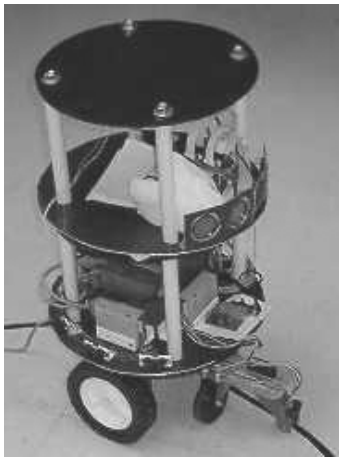

Figura 8. Robot móvil autónomo

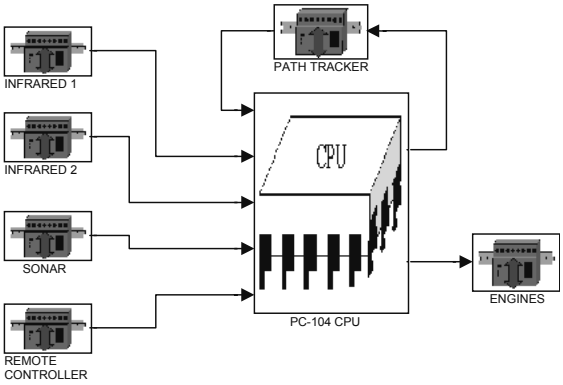

Figura 9. Nodos del robot autónomo (librería RTF)

Siguiendo un enfoque jerárquico, el usuario puede definir la arquitectura software para cada nodo en términos de: tareas concurrentes existentes, comunicación y sincronización entre dichas tareas y comunicación entre el nodo y la red CAN (mensajes enviados y/o recibidos). Para ello, se dispone de un conjunto de bloques que permiten una especificación intuitiva y gráfica del modelo.

La figura 10 ilustra las tareas concurrentes que forman el sistema de control del robot móvil autónomo. Las cuatro conductas ('explorar el entorno', 'evitar obstáculos', 'seguir línea' y 'control remoto') se implementan como tareas concurrentes y un conjunto de tareas de ponderación realizan la compensación para obtener la consigna de velocidad. Así mismo, se observa que alguna de las tareas utiliza los bloques de librería para recibir y enviar mensajes a la red CAN.

Finalmente, es necesario especificar la funcionalidad de cada tarea del sistema. En este sentido, Simulink y Stateflow facilitan la especificación de los algoritmos de control y la del comportamiento dirigido a eventos discretos en cada tarea. Así, la conducta 'seguir línea' puede ser especificada en Simulink, tal y como se presenta en la figura 11. De la misma forma, se han especificado el resto de conductas.

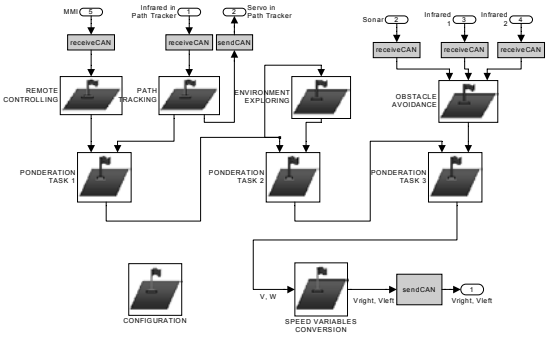

Figura 10. Arquitectura software del robot móvil.

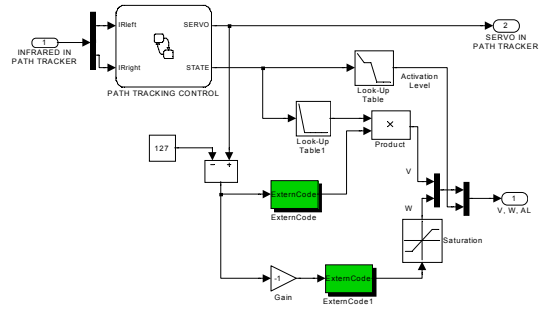

Figura 11 Conducta seguir línea

6.3 Generación del código de la aplicación

Una vez especificado el sistema de control distribuido, y antes de proceder a generar el código de la aplicación, es necesario detallar aquellos aspectos relativos a las plataformas hardware y software que se emplearán en la implementación del sistema: nodos CAN concretos, tipos de CPU, sistemas operativos, etc. Así, para cada nodo inteligente se indica el sistema operativo (en este caso VxWorks), el protocolo de red (rtfCANopen) y el tipo de bus interno (PCI, PC-104). Los nodos de entrada salida quedan identificados indicando el fabricante y modelo (e.g. ESD, Selectron,...) lo que permite seleccionar el *driver* que se encarga de su configuración y operación.

Real Time Workshop (Mathworks, 1999d) genera un fichero ASCII intermedio con la representación del diagrama de bloques del modelo Simulink. La generación de código la realiza invocando un programa escrito en el lenguaje de marcado *Target Language Compiler* (TLC). Este programa interpreta el fichero ASCII del modelo y lo transforma en código fuente para la plataforma específica. Posteriormente, obtiene el código ejecutable para la plataforma destino utilizando las librerías necesarias. En particular, en este caso de uso se genera código suponiendo que los nodos de procesamiento operan bajo sistema operativo VxWorks y que el protocolo de alto nivel del bus es *rtfCANopen*.

RTF realiza mejoras sobre el proceso de generación de Real Time Workshop para habilitar la construcción de código de forma acorde a la arquitectura de tareas especificada en el modelo. Para ello, RTF utiliza un programa TLC propio que genera el código fuente de cada nodo inteligente basándose en la arquitectura software especificada en el modelo. Como en el caso del robot móvil la aplicación se ejecuta en nodos de procesamiento bajo el sistema operativo de tiempo real *VxWorks*, la librería RTF correspondiente define los procesos y los mecanismos de comunicación y de sincronización utilizados por este sistema operativo (semáforos, colas, señales,...).

Por tanto, el código para la CPU PC-104 se genera automáticamente desde RTF. Para el robot móvil autónomo este código incluye:

- Código para programar las tareas de coordinación y supervisión.
- Código para controlar la tarjeta CAN, conectada a la CPU PC-104.
- Código para la puesta en marcha de la red, descarga de todos los módulos APP a las tarjetas y configuración de los nodos a través de mensajería SDO.

7. APORTACIÓN DE rtfCANopen EN EL DESARROLLO DE SISTEMAS DISTRIBUIDOS

Este apartado intenta clarificar las implicaciones que caben esperar del uso de los nodos CAN y la librería *rtfCANopen*, presentados en este trabajo, en el desarrollo de sistemas distribuidos con requisitos temporales.

En primer lugar, es claro que el uso del protocolo CAN no garantiza en sí mismo el cumplimiento de los requisitos temporales que se le impongan al conjunto de la aplicación. Se podrá asegurar el correcto comportamiento temporal del sistema en función de la forma en que se reflejen los requisitos globales de la aplicación en la elección del número, naturaleza y estructura de los mensajes, así como del número y planificación de las tareas especificadas en los distintos nodos.

Ahora bien, *rtfCANopen* supone una abstracción de las comunicaciones, que facilita al usuario de RTF concentrarse en los aspectos propios del diseño de los algoritmos de control. Esta abstracción permite tomar decisiones en el diseño de la aplicación acordes a los criterios de optimización en el uso de recursos y cumplimiento de restricciones temporales. Por ejemplo, se puede deducir automáticamente la estructura, periodicidad y prioridad de un mensaje CAN a partir del conjunto de señales que, desde el diseño de alto nivel de la aplicación, se decida que deban enviarse de un nodo a otro. Es decir, *rtfCANopen* constituye una pieza más de RTF, entendido este último como un entorno que integra herramientas de ayuda al diseño de sistemas de control distribuido. El proceso de desarrollo que propugna RTF permite que diferentes herramientas (y los diferentes expertos que las usan) colaboren

para alcanzar un objetivo común que cumpla los requisitos globales impuestos a la aplicación, tanto los funcionales como los no funcionales.

En este sentido, el conocimiento de tiempos de ejecución de peor caso de todas las rutinas es necesario, pues deben asegurarse los plazos de todas las tareas de tiempo real crítico de una aplicación a través de un análisis temporal. La disponibilidad de la totalidad del código fuente ofrecida por rtfCANopen permitirá calcular los tiempos de ejecución que requiere una herramienta de análisis temporal, como por ejemplo *BERTA* (Marcos y Portillo, 2000).

Por otro lado, algunos requisitos temporales impuestos a la globalidad de la aplicación resultan en plazos temporales asociados a comunicaciones extremo a extremo, en las que se ven envueltos varios nodos de la red y varios mensajes. *rtfCANopen* permitirá imponer estas restricciones a todos los mensajes implicados.

Para alcanzar este objetivo, es necesario disponer de herramientas que traduzcan las especificaciones globales de la aplicación en implicaciones concretas para los mensajes CAN (estructura, naturaleza y prioridad), que permitan analizar el comportamiento temporal del sistema distribuido en su conjunto y que permitan la posterior generación automática del código adecuado. Este es el objetivo final de un entorno como RTF para el que rtfCANopen constituye una de sus piezas.

8. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se presenta el desarrollo de una implementación CANopen de bajo costo que es adecuada para su uso en aplicaciones distribuidas empotradas y ha sido desarrollada en consonancia con los estándares asociados.

Su diseño modular (módulos de comunicación y aplicación) hace que rtfCANopen suponga una abstracción ventajosa para los ingenieros de control que pueden centrarse en los aspectos propios de su disciplina, dejando la resolución de la comunicación a una librería ya desarrollada. De cualquier forma, cuando surja la conveniencia de modificar el protocolo de comunicaciones para mejorar el rendimiento de los algoritmos de control, rtfCANopen permitirá realizar dichos cambios. Es más, la actualización del software de toda la red se ve simplificada gracias a las funcionalidades de descarga y configuración global y automática de rtfCANopen.

Por último, hay que destacar que no se trata de una implementación aislada sino que este sistema de comunicación se integra dentro de un proceso coherente para el desarrollo de sistemas distribuidos con requisitos de tiempo real en el que colaboran otras herramientas desarrolladas por los autores para cubrir las fases desde la especificación hasta la generación del código final del sistema de control.

El trabajo futuro incluye modificaciones de rtfCANopen que mejoren los requisitos de tiempo real (aunque manteniendo la compatibilidad con CANopen) y midan el tiempo de cómputo de peor caso (WCET). En esta línea, se facilitará la realización de estudios temporales y cálculos de tiempos de respuesta de peor caso para los sistemas basados en rtfCANopen.

El código fuente de rtfCANopen y la documentación correspondiente está disponible en: www.disa.bi.ehu.es/rtfCANopen.htm.

AGRADECIMIENTOS

Este trabajo se ha realizado gracias a la ayuda del Ministerio de Ciencia y Tecnología dentro del marco del Proyecto TAP 98-0585-C03-02, por la UPV/EHU 146-345-TB196/98 y GV PI-1998-42.

REFERENCIAS

CAN (1991). Bosch CAN specification, V 2.0 PartA. R. Bosch Gmbh, Germany.

CiA (CAN in Automation) (1996a). CANopen Communication profile for Industrial Systems. CiA Draft Standard 301. CiA, Germany.

CiA (CAN in Automation) (1996b). NMT service specification. CiA Draft Standard 203-1.CiA, Germany.

CiA (CAN in Automotion) (1998). *Framework for Programmable CANopen Devices*. DSP302

ESD gmbh (1998). *CAN-PC104/331 software manual*. Electronic System Design gmbh. Hannover (Germany).

Johansson, L., Broqvist, P., Ohlsson, J. y Torin, J. (2000). Efficient CAN based automotive control systems. www.volcanoautomotive.com

Marcos, M. M., Portillo, J. (2000). Basic Environment for Real Time Systems Analysis using CAN bus. Proceeding of the Workshop on Real Time Programming. Palma de Mallorca (Spain), May 2000.

Marcos, M. M., Portillo, J., Bass J.M. (2000). Matlab-based real-time framework for distributed control systems. Proceeding of the Workshop on Algorithms and Architectures for Real-Time Control. Palma de Mallorca (Spain), May 2000.

Mathworks (1999a). Using Matlab version 5.3.

Mathworks (1999b). Simulink: Dynamic System Simulation for Matlab V2.1

Mathworks (1999c). Stateflow: For use with Simulink. User's Guide

Mathworks (1999d). Real Time Workshop: For use with Simulink. User's Guide

Microchip (1998). *PIC16F87X 28/40 pin 8-bit CMOS* FLASH *microcontrollers.* Microchip, USA.

Microchip (1999). *MCP2510 Stand-Alone CAN controller with SPI interface.* Microchip, USA.

OSEK/VDX (2004). www.osek-vdx.org

Philips semiconductors (1997). PCA82C250 CAN controller interface. Philips Semiconductors, The Netherlands.

Portillo, J., Marcos, M. (2001). Contributions to the design of real time distributed control systems. Proceedings of the European Control Conference, September, 2001. Porto, Portugal.

Portillo J., Marcos M., Olarra A., Cabanes I. (2002) rtfCANopen: Una implementación modular y económica aplicada al control de un robot móvil autónomo. Proceedings XXIII Jornadas de Automática, Tenerife, septiembre 2002.

Specks, W., Rajnák, A. (1998). The Scaleable Network Architecture of the Volvo S80. 8th Intl. Conference on Electronic Systems for Vehicles, Baden-Baden, Oct 1998, pp. 597-641

Tindell, K. and A. Burns (1994). Guaranteeing Message Latencies on Controller Area Network (CAN). Proceedings 1st International CAN Conference, Mainz (Germany), September 1994.

Tindell, K. and Rajnak A. (1998). A CAN Communications Concept with Guaranteed Message Latencies, Volcano Communications Technologies, Lennart Casparsson Volvo Car. Corp. SAE Technical Paper Series, 98C50.