CANopen

Departamento de Informática y Automática

Facultad de Ciencias

Universidad de Salamanca

Raquel Sánchez Díaz

Tabla de Contenidos

Índice de figuras

Índice de tablas

1. Bus CAN y CANopen

El bus de campo CAN sólo define las capas física y de enlace por lo que es necesario definir cómo se asignan y utilizan los identificadores y datos de los mensajes CAN. Para ello se definió el protocolo CANopen, que está basado en CAN, e implementa la capa de aplicación. Actualmente está ampliamente extendido, y ha sido adoptado como un estándar internacional.

La construcción de sistemas basados en CAN que garanticen la interoperatividad entre dispositivos de diferentes fabricantes requiere una capa de aplicación y unos perfiles que estandaricen la comunicación en el sistema, la funcionalidad de los dispositivos y la administración del sistema:

- Capa de aplicación (*application layer*). Proporciona un conjunto de servicios y protocolos para los dispositivos de la red.
- Perfil de comunicación (*communication profile*). Define cómo configurar los dispositivos y los datos, y la forma de intercambiarlos entre ellos.
- Perfiles de dispositivos (*device profiles*). Añade funcionalidad específica a los dispositivos.

La relación entre el modelo OSI y los estándares CAN y CANopen la podemos ver en la siguiente figura:

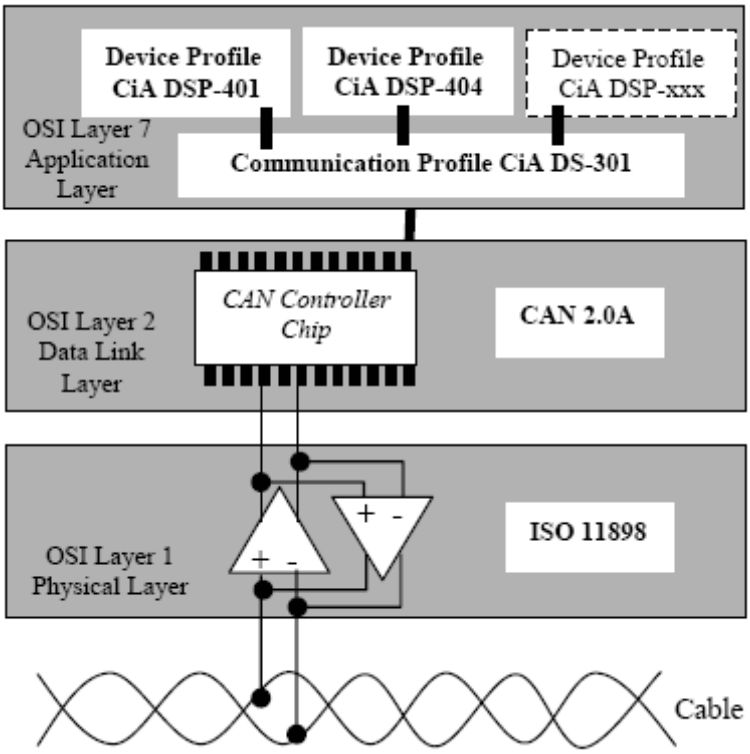


Figura 1: Visión esquemática de los estándares CAN y CANopen en el modelo OSI

A continuación se describirá la capa de aplicación CAL para las redes CAN, en la que se basa el protocolo CANopen. Después se explicará dicho protocolo y los perfiles que define.

1.1. CAL (CAN Application Layer)

Fue una de las primeras especificaciones producidas por CiA (CAN in Automation) [1], basada en un protocolo existente desarrollado originalmente por Philips Medical Systems. Ofrece un ambiente orientado a objetos para el desarrollo de aplicaciones distribuidas de cualquier tipo, basadas en CAN.

Esta capa está compuesta por los siguientes cuatro servicios:

- CMS (*CAN-based Message Specification*): ofrece objetos de tipo variable, evento y dominio para diseñar y especificar cómo se accede a la funcionalidad de un dispositivo través de su interfaz CAN.
- NMT (*Network ManagemenT*): proporciona servicios para la gestión de la red. Realiza las tareas de inicializar, arrancar, parar o detectar fallos en los nodos. Se basa en el concepto de maestro-esclavo, habiendo sólo un NMT maestro en la red.
- DBT (*DistriBuTor*): se encarga de asignar de forma dinámica los identificadores CAN (de 11 bits), también llamados COB-ID (*Communication Object Identifier*). También se basa en el modelo maestro-esclavo, existiendo sólo un DBT maestro.

- LMT (*Layer ManagemenT*): permite cambiar ciertos parámetros de las capas como por ejemplo el identificador de un nodo (Node-ID) o la velocidad del bus CAN.

CMS define 8 niveles de prioridad en sus mensajes, cada uno con 220 COB-IDs, ocupando desde el 1 al 1760. Los identificadores restantes (0, 1761-2031) se reservan para NMT, DBT y LMT, como muestra la Tabla 1: Distribución de los COB-IDs en CAL. En una red CAN el mensaje con el COB-ID más pequeño es el de mayor prioridad.

Este estándar asume CAN2.0A (*CAN Standard Message Frame*) con identificadores de 11 bits. Sin embargo se puede utilizar CAN2.0B (*CAN Extended Message Frame*) con identificadores de 29 bits. En ese caso se mapean los 11 bits definidos anteriormente con los 11 bits más significativos del identificador de tramas extendidas. De esta manera los rangos de la tabla quedan bastante más amplios.

COB-ID	Utilización	Cantidad
0	NMT Start/Stop	1
1-220	Objetos CMS Prioridad 0	220
221-440	Objetos CMS Prioridad 1	220
441-660	Objetos CMS Prioridad 2	220
661-880	Objetos CMS Prioridad 3	220
881-1100	Objetos CMS Prioridad 4	220
1101-1320	Objetos CMS Prioridad 5	220
1321-1540	Objetos CMS Prioridad 6	220
1541-1760	Objetos CMS Prioridad 7	220
1761-2015	Monitoreo dispositivos NMT	255
2016-2031	Servicios de NMT, LMT y DBT	16

Tabla 1: Distribución de los COB-IDs en CAL

2. CANopen

CAL proporciona todos los servicios de gestión de red y mensajes del protocolo pero no define los contenidos de los objetos CMS ni los tipos de objetos. Es decir, define cómo pero no qué. Aquí es donde CANopen entra en juego.

CANopen [3] está por encima de CAL y utiliza un subconjunto de sus servicios y protocolos de comunicación. Proporciona una implementación de un sistema de control distribuido utilizando los servicios y protocolos de CAL.

El concepto central de CANopen es el diccionario de objetos (OD, *Device Object Dictionary*). Es un concepto utilizado por otros buses de campo. No forma parte de CAL.

2.1. Diccionario de objetos de CANopen

Es un grupo ordenado de objetos. Describe completamente y de forma estandarizada la funcionalidad de cada dispositivo y permite su configuración mediante mensajes (SDO) a través del propio bus.

Cada objeto se direcciona utilizando un índice de 16 bits. Para permitir el acceso a elementos individuales de las estructuras de datos también existe un subíndice de 8 bits. En la siguiente tabla podemos ver la estructura general del diccionario:

Indice	Objeto	Descripción
0x0000	No usado	
0x0001 - 0x001F	Tipos de dato estáticos	Contiene definiciones de tipos de dato estándar como boolean, enteros, string, punto flotante, etc. Se incluyen como referencia, no pueden ser leídas ni escritas.
0x0020 - 0x003F	Tipos de dato complejos	Contiene definiciones de estructuras predefinidas, compuestas de tipos estáticos, comunes a todos los dispositivos.
0x0040 - 0x005F	Tipos de dato específicos del fabricante	Contiene definiciones de estructuras predefinidas, compuestas de tipos estáticos, específicas de un dispositivo en particular.
0x0060 - 0x007F	Tipos de dato estáticos específicos del perfil del dispositivo	Definiciones de tipos de dato básicos específicos para el perfil del dispositivo.
0x0080 - 0x009F	Tipos de dato complejos específicos del perfil del dispositivo	Definiciones de estructuras específicas para el perfil del dispositivo.
0x00A0 - 0x0FFF	Reservado	
0x1000 - 0x1FFF	Rango para el perfil de comunicaciones	Contiene parámetros de configuración del bus CAN. Estas entradas del diccionario son comunes a todos los dispositivos.
0x2000 - 0x5FFF	Rango para el perfil específico del fabricante	Contiene las extensiones al perfil estándar, realizadas por el fabricante.
0x6000 - 0x9FFF	Rango para perfiles de dispositivo estandarizados	Contiene todos los objetos de datos comunes a un tipo de perfil que pueden ser leídos o escritos desde la red. Algunas de las entradas son obligatorias (funcionalidad requerida) mientras que otras son opcionales (funcionalidad opcional).
0xA000 - 0xFFFF	Reservado	

Tabla 2: Estructura de un diccionario de objetos estándar en CANopen

El rango relevante de objetos va desde el índice 1000 al 9FFF. Para cada nodo de la red existe un OD, diccionario de objetos, que contiene todo los parámetros que describen el dispositivo y su comportamiento en la red.

En CANopen hay documentos que describen perfiles. Hay un perfil de comunicaciones (*communication profile*) donde están descritos todos los parámetros relacionados con las comunicaciones. Además hay varios perfiles de dispositivos (*device profiles*) donde se definen los objetos de un dispositivo en particular.

Un perfil define para cada objeto del diccionario su función, nombre, índice, subíndice, tipos de datos, si es obligatorio u opcional, si es de “sólo lectura”, “sólo escritura” o “lectura-escritura”, etc.

2.2. Identificadores de mensaje

CANopen define la distribución de los identificadores de mensaje (*Predefined Connection Set*) de manera que hay un mensaje de emergencia por nodo, mensajes de sincronización y *time stamp*, un SDO (ocupando dos identificadores), mensajes NMT y cuatro PDOs de transmisión y cuatro de recepción por dispositivo.

El identificador de 11 bits se divide en dos partes:

- 4 bits para el código de función
- 7 bits para el identificador de nodo (Node-ID)

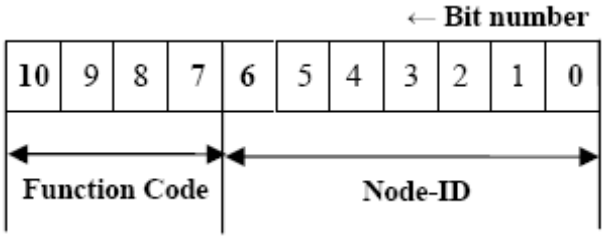


Figura 2: Estructura del identificador de mensajes CAN

La distribución de los identificadores se corresponde con una estructura del tipo maestro-esclavo. El maestro que conoce los Node-IDs de todos los esclavos conectados (máximo 127). Dos esclavos no pueden comunicarse entre sí porque no conocen sus identificadores.

En las siguientes tablas podemos ver la distribución general de los identificadores:

Broadcast objects of the CANopen Predefined Master/Slave Connection Set			
Object	**Function code (ID-bits 10-7)**	**COB-ID**	**Communication parameters at OD index**
NMT Module Control	0000	000h	–
SYNC	0001	080h	1005h, 1006h, 1007h
TIME STAMP	0010	100h	1012h, 1013h

Peer-to-Peer objects of the CANopen Predefined Master/Slave Connection Set			
Object	**Function code (ID-bits 10-7)**	**COB-ID ***	**Communication parameters at OD index**
EMERGENCY	0001	081h - 0FFh	1024h, 1015h
PDO 1 (transmit)	0011	181h - 1FFh	1800h
PDO 1 (receive)	0100	201h - 27Fh	1400h
PDO 2 (transmit)	0101	281h - 2FFh	1801h
PDO 2 (receive)	0110	301h - 37Fh	1401h
PDO 3 (transmit)	0111	381h - 3FFh	1802h
PDO 3 (receive)	1000	401h - 47Fh	1402h
PDO 4 (transmit)	1001	481h - 4FFh	1803h
PDO 4 (receive)	1010	501h - 57Fh	1403h
SDO (transmit/server)	1011	581h - 5FFh	1200h
SDO (receive/client)	1100	601h - 67Fh	1200h
NMT Error Control	1110	701h - 77Fh	1016h, 1017h

Tabla 3: Asignación de los identificadores CAN en CANopen

2.2.1 Modelo de comunicaciones

El modelo de comunicaciones de CANopen define cuatro tipos de mensajes (objetos de comunicación):

- Objetos administrativos: son mensajes administrativos que permiten la configuración de las distintas capas de la red así como la inicialización, configuración y supervisión de la misma. Se basa en los servicios NMT, LMS (LSS) y DBT de la capa CAL.
- *Service Data Objects* (SDO): objetos o mensajes de servicio utilizados para leer y escribir cualquiera de las entradas del diccionario de objetos de un dispositivo. Corresponden a mensajes CAN de baja prioridad.
- *Process Data Objects* (PDO): objetos o mensajes de proceso utilizados para el intercambio de datos de proceso, es decir, datos de tiempo real. Por este motivo, típicamente corresponden a mensajes CAN de alta prioridad.
- Mensajes predefinidos: de sincronización, de emergencia y *time stamp*. Permiten la sincronización de los dispositivos (objetos SYNC) y generar notificaciones de emergencia en forma opcional.

CANopen soporta los modelos de comunicación punto-a-punto, maestro-esclavo y productor-consumidor en sus variantes *push* y *pull*. En el modelo *push* los productores colocan los eventos en el canal de eventos y éste se los envía a los consumidores. En el *pull* el flujo de eventos ocurre en el sentido contrario, es decir, los consumidores solicitan eventos al canal de eventos y éste los solicita a los productores.

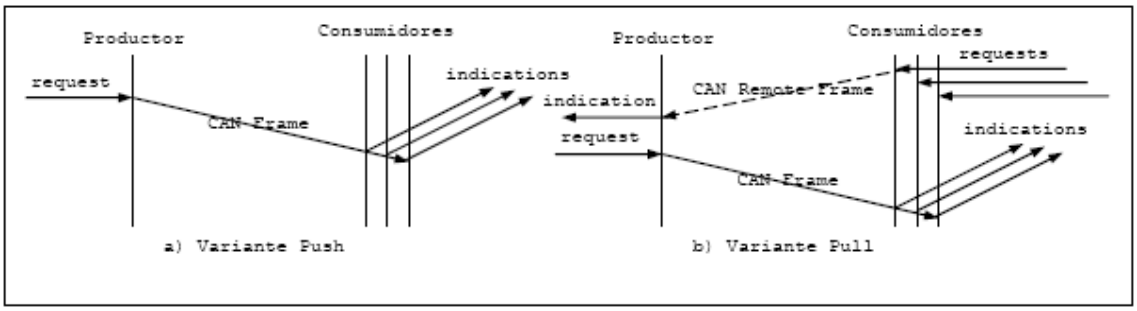


Figura 3: Modelo de comunicación productor-consumidor en CANopen

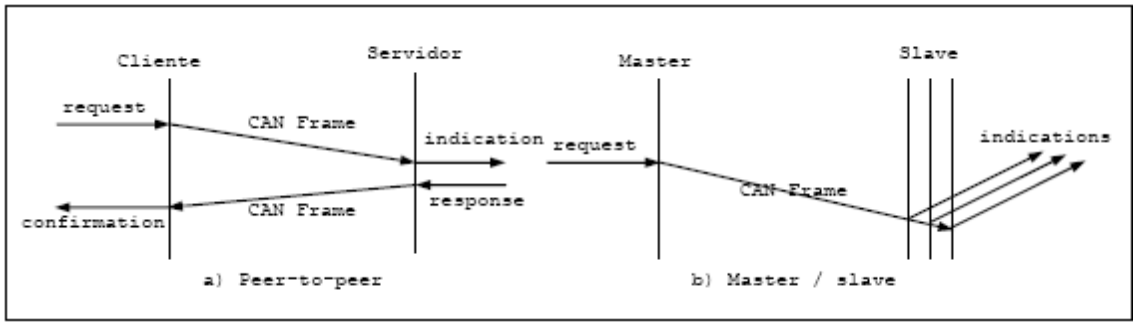


Figura 4: Modelos de comunicación punto-a-punto y maestro-esclavo en CANopen

2.3. Service Data Objects (SDO)

Normalmente este tipo de objetos es usado para configuración de dispositivos y transferencia de grandes cantidades de datos no relevantes en forma directa para el control del proceso. En comparación con los PDOs, son mensajes de baja prioridad.

Los objetos de servicio o SDOs permiten implementar los modelos de comunicación cliente-servidor o punto-a-punto, para acceder a los diccionarios de objetos de los dispositivos. Para un cierto SDO, el dispositivo cuyo diccionario está siendo accedido es el servidor mientras que el otro dispositivo, el que inicia la actividad, es el cliente.

Este tipo de objetos ofrece transferencia de datos sin conexión y con confirmación. Por este motivo, cada SDO involucra el intercambio de dos tramas CAN con diferentes identificadores.

Un SDO es representado en CMS como un objeto de tipo *Multiplexed Domain*. Se definen una serie de protocolos petición-respuesta que se pueden aplicar a los SDOs para su transferencia:

- *Initiate Domain Download*
- *Initiate Domain Upload*
- *Download Domain Segment*
- *Upload Domain Segment*
- *Abort Domain Transfer*

Download significa escribir en el diccionario de objetos y *upload* leer del él.

Al ser *Multiplexed Domains* desde el punto de vista de CMS, los SDOs pueden transferir datos de cualquier longitud. Sin embargo CANopen define dos tipos de transferencia para los SDO basándose en el tamaño de los datos a transferir: transferencia expédita, para datos de longitud menor o igual a 4 *bytes*, y transferencia en segmentos, para datos de longitud mayor de 4 *bytes*.

Los mensajes tanto del cliente como del servidor siempre tienen una longitud de 8 *bytes* aunque no todos contengan información significativa.

Es importante tener en cuenta que en CANopen los parámetros de más de un *byte* se envían siempre en la forma *little endian*, es decir, primero el *byte* menos significativo (LSB).

2.3.1 Transferencia expédita

Usada para transmitir mensajes con longitud de datos menor o igual a 4 *bytes*, para lo cual se usan los protocolos *Initiate Domain Download* o *Initiate Domain Upload*.

No se aplica fragmentación, se envía un único mensaje CAN y se recibe la confirmación del servidor.

Un SDO que se transmite del cliente al servidor bajo este protocolo tiene el siguiente formato:

- *Command Specifier* (1 *byte*): contiene información sobre si es subida o descarga de datos, petición o respuesta, transferencia expédita o en segmentos, tamaño de los datos y el *toggle* bit:
  - *Client Command Specifier* (3 bits): 001 para *download* y 010 para *upload*.

- Ignorado (1 bit): se pone a 0.
- Número de *bytes* / Ignorado (2 bits): número de *bytes* que no contienen datos. Es válido si los siguientes dos bits son 1. Si no, vale 0. En *upload* se ignora.
- *Transfer Expedited* / Ignorado (1 bit): indica si se trata de una transferencia expédita (a 1) o una transferencia en segmentos (a 0). Tiene que estar siempre a 1 para este tipo de transferencia. En *upload* se ignora.
- *Block Size Indicated* / Ignorado (1 bit): si está a 1 es que el tamaño de los datos está especificado, si no, no. En *upload* se ignora.

- *Index* (2 *bytes*): índice de la entrada del diccionario de objetos del servidor que el cliente desea acceder mediante el SDO actual. Este campo y el siguiente forman el multiplexor del dominio.
- *Subindex* (1 *byte*): subíndice de la entrada del diccionario de objetos del servidor que el cliente desea acceder. Sólo tiene sentido si la entrada es de un tipo complejo. Si se trata de una entrada con tipo estático, debe ser 0.
- Datos / Ignorado (4 *bytes*): datos que se desean enviar al servidor (el valor de una entrada en el diccionario si se está escribiendo). Si es *upload* se ignora.

El servidor, al recibir el mensaje anterior, y si ha accedido con éxito al diccionario, contesta con un mensaje con el siguiente formato:

- *Command Specifier* (1 *byte*):
  - *Server Command Specifier* (3 bits): 011 para *download* y 010 para *upload*.
  - Ignorado (1 bit): se pone a 0.
  - Ignorado / Número de *bytes* (2 bits): si es un *download* se ignora.
  - Ignorado / *Transfer Expedited* (1 bit): si es un *download* se ignora.
  - Ignorado / *Block Size Indicated* (1 bit): si es un *download* se ignora.
- Ignorado / Datos (4 *bytes*): si es un *download* (escritura) se ignora ya que es una confirmación. Si es una lectura o *upload*, contiene el valor leído del diccionario de objetos del servidor.

2.3.2 Transferencia en segmentos

Usada para transmitir mensajes con longitud de datos mayor de 4 *bytes*. Se aplica fragmentación en segmentos partiendo los datos en múltiples mensajes CAN. El cliente espera confirmación del servidor por cada segmento.

Para el primer mensaje tanto del cliente como del servidor, se usan los protocolos *Initiate Domain Download* o *Initiate Domain Upload* según corresponda, con el bit *Transfer Expedited* a 0 (para transferencia en segmentos). Si ese bit está a 0 y el siguiente (*Block Size Indicated*) está a 1, significa que el campo de datos (de 4 *bytes*) contiene el número de *bytes* que se van a transmitir (al ser *little endian* el *byte* 4 del mensaje contiene el LSB y el 7 el MSB).

Para los mensajes siguientes se usan los protocolos *Download Domain Segment* o *Upload Domain Segment*, según se quiera leer o escribir una entrada en el diccionario. Los mensajes CAN bajo estos dos protocolos tienen el siguiente formato:

- *Command Specifier* (1 *byte*):
  - *Client Command Specifier* (3 bits): 000 para *download* y 011 para *upload*.
  - *Toggle Bit* (1 bit): es un bit que vale 0 o 1 de forma alternada en segmentos consecutivos. La primera vez vale 0. Teniendo en cuenta que el mecanismo de intercambio sólo permite un mensaje pendiente de confirmación, un único bit es suficiente para esto. El servidor se limita a hacer un eco del valor recibido.
  - Número de *bytes* / Ignorado (3 bits): indica el número de *bytes* que no contienen datos, o cero si no especifica el tamaño. Se ignora en el *upload*.
  - *Last Segment Indication* / Ignorado (1 bit): vale 1 si se trata del último segmento del SDO que se está transmitiendo y 0 si hay más segmentos. Para *upload* se ignora.
- Datos / Ignorado (7 *bytes*): datos que se desean enviar al servidor. Son los *bytes* que no caben en el mensaje CAN inicial. Si es *upload* se ignora.

El servidor, al recibir el mensaje anterior, contesta con un mensaje con el formato:

- *Command Specifier* (1 *byte*):
  - *Client Command Specifier* (3 bits): 001 para *download* y 000 para *upload*.
  - *Toggle Bit* (1 bit): igual que antes.
  - Ignorado / Número de *bytes* (3 bits): se ignora en el *download*.
  - Ignorado / *Last Segment Indication* (1 bit): se ignora en el *download*.
- Ignorado / Datos (7 *bytes*): si es un *download* (escritura) se ignora porque es una confirmación. Si es una lectura o *upload*, contiene el valor leído del diccionario de objetos del servidor.

2.3.3 Abort Domain Transfer

Uno de los protocolos antes mencionados y que aún no se ha explicado es el *Abort Domain Transfer*. Existe la posibilidad que al acceder a las entradas del diccionario de objetos del servidor, se produzca un error. Este protocolo es usado en esos casos para notificar tanto a clientes como a servidores. El formato de los mensajes de este protocolo es el siguiente:

- *Command Specifier* (1 *byte*):
  - *Command Specifier* (3 bits): 100 para *Abort Domain Transfer*.
  - Ignorado (5 bits): se ignoran.
- *Index* (2 *bytes*): índice de la entrada del diccionario de objetos del servidor que causó el error que se está notificando. Este campo y el siguiente forman el multiplexor del dominio.
- *Subindex* (1 *byte*): subíndice de la entrada del diccionario de objetos del servidor que causó el error que se está notificando. Sólo tiene sentido si la entrada es de un tipo complejo. Si se trata de una entrada con tipo estático, debe ser 0.

- Código de error (4 *bytes*): código que identifica el error. En la siguiente tabla podemos ver sus posibles valores:

Abort Code	Description
0503 0000	Toggle bit not alternated
0504 0000	SDO protocol timed out
0504 0001	Client/Server command specifier not valid or unknown
0504 0002	Invalid block size (Block Transfer mode only)
0504 0003	Invalid sequence number (Block Transfer mode only)
0503 0004	CRC error (Block Transfer mode only)
0503 0005	Out of memory
0601 0000	Unsupported access to an object
0601 0001	Attempt to read a write-only object
0601 0002	Attempt to write a read-only object
0602 0000	Object does not exist in the Object Dictionary
0604 0041	Object can not be mapped to the PDO
0604 0042	The number and length of the objects to be mapped would exceed PDO length
0604 0043	General parameter incompatibility reason
0604 0047	General internal incompatibility in the device
0606 0000	Object access failed due to a hardware error
0606 0010	Data type does not match, lengh of service parameter does not match
0606 0012	Data type does not match, lengh of service parameter is too high
0606 0013	Data type does not match, lengh of service parameter is too low
0609 0011	Sub-index does not exist
0609 0030	Value range of parameter exceeded (only for write access)
0609 0031	Value of parameter written too high
0609 0032	Value of parameter written too low
0609 0036	Maximum value is less than minimum value
0800 0000	General error
0800 0020	Data can not be transferred or stored to the application
0800 0021	Data can not be transferred or stored to the application because of local control
0800 0022	Data can not be transferred or stored to the application because of the present device state
0800 0023	Object Dictionary dynamic generation fails or no Object Dictionary is present (e.g. OD is generated from file and generation fails because of a file error)

Tabla 4: Códigos de error para SDO *Abort Domain Transfer*

Los SDOs requieren ser definidos en el diccionario de objetos mediante una estructura que contiene parámetros relacionados con la transmisión de los mismos. La estructura para el primer SDO para servidores tiene un índice de 0x1200 mientras que el primer SDO para clientes, se ubica en la entrada 0x1280. En total, en una red CANopen, se pueden definir hasta 128 SDOs para clientes y 128 SDOs para servidores.

2.4. Process Data Objects (PDO)

Este tipo de objetos permite intercambiar datos del proceso en tiempo real. Implementa el modelo de comunicaciones productor-consumidor. Los datos se transmiten desde un productor a varios consumidores.

Ofrece un servicio de transferencia de datos sin conexión y sin confirmación. No se aplica un protocolo de fragmentación y reensamble de los objetos. Los PDOs están pensados para tráfico de tiempo real de alta prioridad, por lo que es conveniente evitar la sobrecarga que

produciría agregar un protocolo de fragmentación y confirmación como el que se usa en los SDOs.

Los mensajes PDO de un nodo o dispositivo pueden dividirse en dos categorías. Los tPDO son aquello mensajes con información del proceso que el nodo transmite (por ejemplo la lectura de un sensor). Por otro lado, los rPDO son los mensajes con información del proceso que el nodo escucha (por ejemplo un nodo que controle la apertura de una bomba escuchará el bus en busca de órdenes).

El contenido de un PDO está definido tan sólo por su identificador. Tanto el emisor como el receptor deben conocerlo para poder interpretar su estructura interna. Cada PDO se describe mediante dos objetos del diccionario:

- PDO *Communication Parameter*: contiene el COB-ID que utiliza el PDO, el tipo de transmisión, tiempo de inhibición y temporizador.
- PDO *Mapping Parameter*: contiene una lista de objetos del OD contenidos en la PDO, incluyendo su tamaño en bits.

CANopen define varios mecanismos de comunicación para la transmisión de PDOs:

- Transmisión asíncrona:
  - Eventos: la transmisión de un mensaje es causada por la ocurrencia de un evento específico definido en el perfil del dispositivo.
  - Temporizador: existe un temporizador que cada cierto tiempo cause la transmisión.
  - Solicitud remota: la transmisión asincrónica de mensajes PDO puede comenzar al recibir una solicitud remota (trama RTR) enviada por otro dispositivo.
- Transmisión sincrónica: la transmisión sincrónica de mensajes PDO es disparada por la expiración de un período de transmisión, sincronizado mediante la recepción de objetos SYNC. Es decir, cada vez que llega un mensaje SYNC, se abre una ventana de transmisión sincrónica. Los PDOs sincrónicos deben ser enviados dentro de esa ventana. Se distinguen dos modos dentro de este tipo de transmisión:
  - Modo cíclico: son mensajes que se transmiten dentro de la ventana abierta por el objeto SYNC. No se transmiten en todas las ventanas sino con cierta periodicidad, especificada por el campo *Transmission Type* del *Communication Parameter* correspondiente.
  - Modo acíclico: son mensajes que se transmiten a partir de un evento de la aplicación. Se transmiten dentro de la ventana pero no de forma periódica.

En la siguiente tabla podemos ver los distintos modos de transmisión de PDOs, definidos por el *Transmission Type* (entero de 8 bits) del *Communication Parameter*:

Trans-Mission Type	Condition to trigger PDO (B=both needed, O=one or both)			PDO Transmission
	SYNC*	RTR*	Event*	
0	B	-	B	Sync, acyclic
1-240	O	-	-	Sync, cyclic
241-251	-	-	-	*reserved*
252	B	B	-	Sync, after RTR
253	-	O	-	Async, after RTR
254	-	O	O	Async, manufacturer specific event
255	-	O	O	Async, device profile specific event

*SYNC = objeto SYNC recibido
*RTR = recibida trama RTR
*Event = cambio de valor de un dato, temporizador…

Tabla 5: Modos de transmisión de PDOs enCANopen

Un PDO puede tener asignado un tiempo de inhibición que define el tiempo mínimo que debe pasar entre dos transmisiones consecutivas del mismo PDO. Forma parte del *Communication Parameter*. Está definido como un entero de 16 bits en unidades de 100 microsegundos.

2.5. Mensajes adicionales

2.5.1 Mensaje deTime Stamp

Este tipo de objetos representan una cantidad absoluta de tiempo en milisegundos desde el 1 de Enero de 1984. Proporciona a los dispositivos un tiempo de referencia común.La etiqueta temporal o *time-stamp* se implementa como una secuencia de 48 bits.

2.5.2 Mensaje de sincronización (SYNC)

En una red CANopen, hay un dispositivo que es el productor de objetos SYNC y una serie de dispositivos consumidores de objetos SYNC. Cuando los consumidores reciben el mensaje del productor, abren su ventana de sincronismo y pueden ejecutar sus tareas sincrónicas.

Este mecanismo permite coherencia temporal y coordinación entre los dispositivos. Por ejemplo, un conjunto de sensores pueden leer las variables del proceso controlado en forma coordinada y obtener así una imagen consistente del mismo.

El COB-ID usado por este objeto de comunicación puede encontrarse en la entrada 0x1005 del diccionario. Para garantizar el acceso de estos objetos al bus, debería asignárseles un COB-ID bajo. El conjunto predefinido de conexiones de CANopen sugiere usar un valor de 128. El campo de datos del mensaje CAN de este objeto se envía vacío.

El comportamiento de estos mensajes es determinado por dos parámetros:

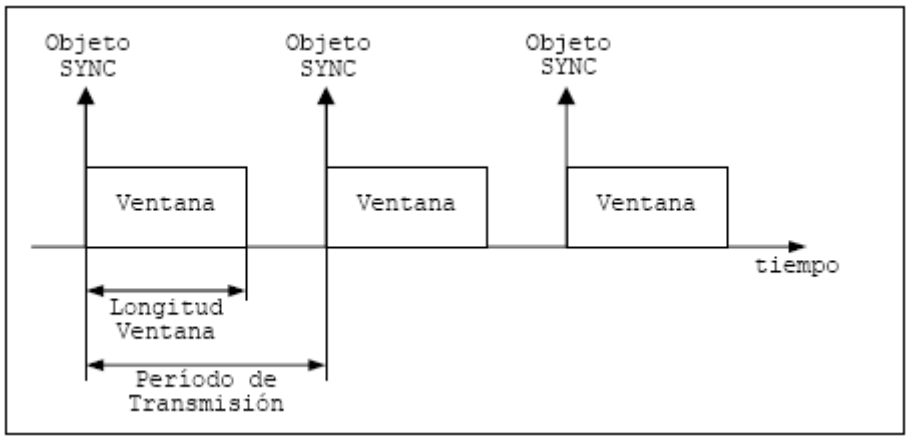


Figura 5: Parámetros de un objeto SYNC en CANopen

La longitud de la ventana o Synchronous Window Length puede ubicarse en la entrada 0x1007 del diccionario. El período de transmisión se encuentra en la posición 0x1006.

2.5.3 Mensaje de emergencia

Estos mensajes se envían cuando ocurre un error interno en un dispositivo. Se transmiten al resto de dispositivos con la mayor prioridad. Pueden usarse como interrupciones o notificaciones de alertas.

Un mensaje de emergencia tiene 8 *bytes*. Su estructura es la siguiente:

COB-ID	Byte 0-1	Byte 2	Byte 3-7
0x080 + Node_ID	**Emergency Error Code**	**Error Register** (Object 0x1001)	Manufac-turer specific error field

Figura 6: Estructura de un mensaje de emergencia en CANopen

- *Emergency Error Code* (2 *bytes*): código del error que causó la generación del mensaje de emergencia. Se trata de fallos internos de los dispositivos por lo cual, los errores se relacionan con fallos de tensión, de corriente, del software del dispositivo, del adaptador del bus CAN, etc. La siguiente tabla nos muestra los códigos correspondientes en hexadecimal:

Emergency Error Code	Meaning
00xx	Error Reset or No Error
10xx	Generic Error
20xx	Current
21xx	current, device input side
22xx	current, inside the device
23xx	current, device output side
30xx	Voltage
31xx	mains voltage
32xx	voltage inside the device
33xx	output voltage
40xx	Temperature
41xx	ambient temperature
42xx	device temperature
50xx	Device hardware
60xx	Device software
61xx	internal software
62xx	user software
63xx	data set
70xx	Additional modules
80xx	Monitoring
81xx	Communication
8110	CAN overrun
8120	Error Passive
8130	Life Guard Error or Heartbeat Error
8140	Recovered from Bus-Off
82xx	Protocol Error
8210	PDO not processed due to length error
8220	Length exceeded
90xx	External error
F0xx	Additional functions
FFxx	Device specific

'xx' es la parte dependiente del perfil del dispositivo

Tabla 6: Códigos de error para los mensajes de emergencia de CANopen

- *Error Register* (1 *byte*): entrada con índice 0x1001 del diccionario de objetos. Cada bit de este registro indica una condición de error distinta cuando está a '1'. El significado de cada bit es:

Bit	Significado
0	Error genérico.
1	Problema de corriente.
2	Problema de tensión.
3	Problema de temperatura.
4	Error de comunicaciones.
5	Específico del perfil de dispositivo.
6	Reservado.
7	Específico del fabricante del dispositivo.

Tabla 7: Bits del Error Register de los mensajes de emergencia de CANopen

- *Manufacturer-specific Error Field* (5 *bytes*): este campo puede usarse para información adicional sobre el error. Los datos incluidos y su formato son definidos por el fabricante del dispositivo.

2.5.4 Mensajes de Node/Life Guarding

Se utiliza para saber si un nodo está operativo. La comunicación se basa en el concepto de maestro-esclavo. El NMT maestro monitorea el estado de los nodos. A esto se le llama *node guarding*. Opcionalmente los nodos pueden monitorear el estado del NMT maestro. A esto se le llama *life guarding*. Comienza en el NMT esclavo después de haber recibido el primer mensaje de *node guarding* del NMT maestro.

Sirve para detectar errores en las interfaces de red de los dispositivos, pero no fallos en los dispositivos en sí (ya que estos son avisados mediante mensajes de emergencia).

Se puede implementar de dos maneras distintas, pudiendo utilizarse sólo una de ellas a la vez: NMT *Node Guarding* o *Heartbeat*.

NMT *Node Guarding*

El maestro va preguntando a los esclavos si están vivos cada cierto tiempo. Si alguno no contesta en un tiempo determinado significa que está caído y se informa de ello. Si los esclavos también monitorean al maestro deben informar de que el maestro está caído si no reciben mensajes de *Node Guarding* de él durante un determinado intervalo.

El maestro interroga a los esclavos mediante una trama remota (RTR) con la siguiente estructura:

COB-ID
0x700 + Node_ID

Figura 7: Trama RTR que el NMT maestro envía a los NMT esclavos

Los NMT esclavos responden con el siguiente mensaje:

COB-ID	Byte 0
0x700 + Node_ID	bit 7: *toggle*, bit 6-0: *state*

Figura 8: Mensaje que los NMT esclavos envían al NMT maestro

El *toggle bit* (bit 7) es un bit que va alternando su valor en cada mensaje de *Node Guarding*. La primera vez vale '0'.

Los bits del 0 al 6 indican el estado del nodo. Su valor lo podemos ver en la siguiente tabla:

Value	State
0	Initialising
1	Disconnected *
2	Connecting *
3	Preparing *
4	Stopped
5	Operational
127	Pre-operational

Los estados marcados con * son para si realiza un *boot-up* extendido

Tabla 8: Valor del campo *state* en un mensaje de NMT *Node Guarding*

Heartbeat

Cada nodo manda un mensaje de *Hearbeat* cada cierto tiempo para informar de que está operativo. En este caso el mensaje de *Boot-up* se considera que es el primer mensaje de *Heartbeat*. Si el NMT maestro deja de recibir estos mensajes durante un tiempo determinado significará que el nodo está caído.

Tienen la siguiente estructura:

COB-ID	Byte 0
0x700 + Node_ID	*state*

Figura 9: Estructura de un mensaje de *Heartbeat*

state	Meaning
0	Boot-up
4	Stopped
5	Operational
127	Pre-operational

Tabla 9: Valor del campo *state* en un mensaje de *Heartbeat*

2.5.5 Gestión de la red (NMT)

Aparte de los mensajes predefinidos, CANopen incluye una serie de mensajes para la administración y monitoreo de los dispositivos en la red. Estos están implementados en la capa CAL y reciben el nombre de servicios de gestión de red (*Network ManagemenT*, NMT). Se trabaja con un modelo de comunicaciones maestro-esclavo en el cual un dispositivo es el NMT maestro y el resto los NMT esclavos.

Un dispositivo NMT esclavo puede encontrarse en alguno de los siguientes estados:

- ***Initialising***: al encender el dispositivo se pasa directamente a este estado. Después de realizar las labores de inicialización el nodo transmite el mensaje de *Boot-up* y pasa al estado *Pre-Operational*.

COB-ID	Byte 0
0x700 + Node_ID	0

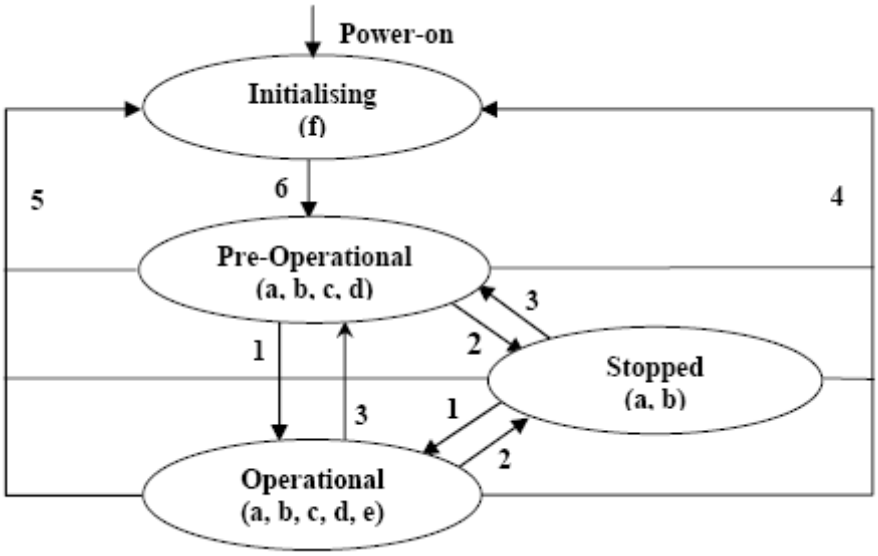
Figura 10: Estructura del mensaje de *Boot-up*

  Para permitir un reseteo parcial del nodo se subdivide en tres subestados:

  - *Reset-Application*: los parámetros específicos del fabricante y del perfil del dispositivo son fijados a su valor inicial. A continuación el nodo pasa al estado *Reset-Communication*.
  - *Reset-Communication*: los parámetros del perfil de comunicaciones son fijados a su valor inicial. A continuación el nodo pasa al estado *Initialising*.

- ***Pre-operational***: en este estado el dispositivo puede ser configurado mediante SDOs. Puede enviar y recibir SDOs, mensajes de emergencia, de sincronización, *time stamp* y mensajes NMT.
- ***Operational***: el dispositivo ya ha sido configurado y funciona normalmente. Todos los objetos de comunicación están activos así también puede enviar y recibir PDOs.
- ***Stopped***: todos los objetos de comunicación dejan de estar activos. No se pueden enviar ni recibir PDOs ni SDOs, sólo mensajes de NMT.

A continuación podemos ver en detalle el diagrama de los posibles estados de un nodo:



Las letras entre paréntesis indican qué objetos de comunicación están permitidos en cada estado:

a. NMT, b. Node Guard, c. SDO, d. Emergency, e. PDO, f. Boot-up

Transiciones entre estados (mensajes NMT):

1: Start_Remote_Node (command specifier 0x01)

2: Stop_Remote_Node (command specifier 0x02)

3: Enter_Pre-Operational_State (command specifier 0x80)

4: Reset_Node (command specifier 0x81)

5: Reset_Communication (command specifier 0x82)

6: Inicialización terminada, entra en Pre-Operational directamente al mandar el mensaje de Boot-up

Figura 11: Diagrama de transición de estados de un nodo en CANopen

Sólo el NMT maestro puede mandar mensajes del módulo de control NMT, pero todos los esclavos deben dan soporte a los servicios del módulo de control NMT.

No hay respuesta para un mensaje NMT. Sólo los envía el maestro y tienen el siguiente formato:

COB-ID	Byte 0	Byte 1
0x000	*CS*	Node-ID

Figura 12: Estructura de un mensaje NMT

Con el Node-ID (identificador de nodo) igual a cero, todos los NMT esclavos son diseccionados (*broadcast*).

El campo CS (*Command Specifier*) puede tener los siguientes valores:

Command Specifier	NMT Service
1	Start Remote Node
2	Stop Remote Node
128	Enter Pre-operational State
129	Reset Node
130	Reset Communication

Tabla 10: Valores del campo CS del mensaje NMT

3. Bibliografía y referencias

[1] **CiA (CAN in Automation).**
http://www.can-cia.org/ (Última vez visitado: 27/6/2007)

[2] **Bus CAN.**
http://www.can-cia.org/can/standardization/standards.html/
(Última vez visitado: 27/6/2007)

[3] **Protocolo CANopen.** "*CANopen: high level protocol for CAN-bus*".
http://www.nikhef.nl/pub/departments/ct/po/doc/CANopen30.pdf
(Última vez visitado: 27/6/2007)

[4] **CANopen solutions.**
http://www. canopensolutions.com/index.html (Última vez visitado: 27/6/2007)