Prof. Dr.-Ing. K. Etschberger

IXXAT Automation GmbH
Leibnizstr. 15, 88250 Weingarten, Germany
Phone: +49-(0)7 51 / 5 61 46-0, Fax: +49-(0)7 51 / 5 61 46-29

e-Mail: info@ixxat.de
http://www.ixxat.de

CAN-based Higher Layer Protocols and Profiles

1 Introduction

The CAN protocol has gained worldwide acceptance as a very versatile, efficient, reliable and economic platform for almost any type of data communication in mobile systems, machines, technical equipment and industrial automation. Based on sophisticated standardised higher layer protocols and profiles, CAN-based open automation technology successfully competes on the market of distributed automation systems. One of the main reasons for the enormous success of CAN-based systems obviously are the special features of the CAN-protocol, especially its producer-consumer- oriented principle of data transmission and its multimaster capability. With these properties, the CAN-protocol also from the technical point of view is very attractive for the usage in distributed systems applications.

When referring to the "CAN standard" or "CAN protocol" we understand the functionality which is standardised in ISO 11898 [1] respectively [2]. This standard comprises the Physical (Layer 1) and Data Link Layer (Layer 2) according to the OSI-reference model. Whereas Layer 1 is responsible for functions like physical signalling, encoding, bit timing and bit synchronisation, Layer 2 performs functions like bus arbitration, message framing and data security, message validation, error detection and signalling and fault confinement. The CAN standard does not specify the medium attachment unit and the medium upon which it resides, nor an Application Layer.

The Layer 2 of the CAN protocol offers two types of connectionless transmission services to the user:

- Unacknowledged transfer of a CAN-message and
- Unacknowledged remote request of a CAN-message
-

Connectionless transmission means that no data link connection has to be established before performing a message transfer or request. Reception of messages is supported by the CAN chips in form of different type object filtering and object buffering methods. A Layer 2 CAN data message according to the CAN Specification V 2.0 is determined by the message identifier, standard/ extended format indication, data length and the data to be transmitted.

Since the CAN-Protocol specifies no rules for the assignment of message-identifiers, a variety of different, application-specific usages is possible. Assignment of the CAN message identifiers therefore

is one of the most important decisions when designing a CAN-based communication system. Assignment and allocation of message identifiers also is one of the main items of a higher Layer approach.

The Requirement of Higher Layers

In practice the implementation even of very simple distributed CAN-based systems shows, that beside of the basic Layer 2 services further functionality is required or desirable e.g. for the transmission of data blocks longer than 8 bytes, acknowledged or confirmed data transfer, identifier distribution, network start-up or the supervision of nodes. Since this additional functionality directly supports the application process, it is understood as "Application Layer"[1]. If implemented properly, the introduction of an Application Layer in addition with an appropriate Application Layer Interface provides a clearly defined separation of communication and application processes.

Since the CAN protocol provides very unique features, most of the known higher layer protocols conserve this features for the user of the Application Layer by providing direct access to the services of the Data Link Layer (no additional protocol overhead for basic functions).

Especially for industrial automation applications, the need for open, standardised higher layer protocols was raised which support interoperability and exchangeability of devices of different manufacturers. Supplementary to a standardised Application Layer therefore the specification of standard device models, "standard devices" and "standard applications" of basic functionality is required.

In the following at first a short overview of the main higher layer protocol solutions will be given. Then the main items of the different solutions will be presented. Within the scope of this paper it is only possible to evaluate some of the main aspects. The main focus will be given to higher layer protocols for industrial automation.

2 Survey of CAN-based Higher Layer protocols

According to the widespread usage of CAN networks with different objectives and requirements beside of many special solutions several main standards of CAN-based Higher Layer protocols are available today. According to the different requirements these solutions differ significantly with respect to their scope and performance.

Representatives of widely accepted Application Layer standards for direct usage by the application are CAL[3] and OSEK [4]. Whereas CAL may be considered as application-independent application layer applicable in any kind of CAN-based application which directly uses the services of the application layer, the OSEK-Com/Net specification represents an application layer[2] and network management functionality whose application is primarily intended for networking in cars.

CAL (CAN Application Layer) was specified as one of the first work items of CAN-in-Automation (CiA) and was published in 1993. CAL offers an application-independent, object-oriented environment for the implementation of CAN-based distributed systems [6]. It provides objects and services for communication, identifier distribution, network and layer management. Main application areas of CAL are CAN-based distributed systems which do not require configurability and standardised device

[1] Since for field bus applications normally no networking and transport via serveral networks is involved, the seven layer OSI reference model may be reduced to only three layers (physical, data link, application).

[2] In OSEK actually this is called "Transport Layer"

modelling. A subset of CAL is used as Application Layer of CANopen[3]. Therefore CANopen devices may be used within application-specific CAL systems.

OSEK/VDX is a joint project of the automotive industry with the objective to provide an industry standard for an open architecture for distributed control in vehicles. This standard comprises the definition of a real-time operating system, software interfaces as well as a communication and network management system. The OSEK operating system provides services for management and synchronisation of tasks, interrupt management, alarms and error handling. One major goal of the approach was the provision of a common platform to integrate software modules from various manufacturers. As the operating system is intended for use in any type of control units, it must support time-critical applications on a wide range of hardware. The OSEK communication specification [6] defines a hardware and bus system independent application interface. Communication between local and remote tasks is performed by the operation system via "Message Objects". Two types of messages are distinguished: "State Messages" and "Event Messages". State Messages always represent the most actual state of a system variable (not buffered), by means of Event Messages events are reported. Thereby each message has to be processed by the consumer. Both types of messages can be used in a peer-to-peer and multicast fashion with transfer modes periodically, event-driven and periodically/eventdriven. Transport Layer services provide acknowledged and fragmented transfer of data in addition to the unacknowledged and unfragmented services of the Data Link Layer.

Based on the very high requirements of in-vehicle communication a sophisticated Network Management System is specified [7] which has to ensure the safety and reliability of the communication network. By means of "Node Monitoring" every node is actively monitored by every other node in the network (Direct Monitoring). For this purpose the monitored node sends a NM message according to a dedicated and uniform algorithm. Direct node monitoring requires a network-wide synchronisation of NM messages. For this purpose a logical ring is used. Any node must be able to send NM messages to all other nodes and receive messages from them.

If direct monitoring is too complex for a device the principle of "Indirect Monitoring" may be applied. It is based on the observation of application messages and limited to nodes which periodically send messages. A node of that type may be monitored by one or more other nodes.

A quite different kind of an open system solution is provided with the **SAE J1939** [11] standard. This standard was defined by the Society of Automotive Engineers Heavy Truck and Bus Division to provide an open interconnect system for electronic systems. Main applications are light, medium and heavy duty vehicles used on or off roads as well as appropriate stationary applications which use vehicle derived components. Vehicles include highway trucks and their trailers, construction equipment, agricultural equipment and ship instrumentation. J1939 is based on the usage of a 29 bit message identifier. The standardised usage of the message identifier results in the distinction of 8 priority classes, predefined message types, destination specific communication and broadcasting. In the J1939/7x standard in-vehicle and diagnostic messages are defined. Hereby data type, range, repetition rate etc. together with the corresponding Parameter Group Number determine the respective message identifier. Also the mapping of the messages into CAN data field of a parameter group is defined.

Main representatives of open distributed system standards for industrial applications are CANopen [8], DeviceNet [9] and SDS [10].

The industrial application of open distributed systems standards comprises low-level networking of industrial devices (sensors, actors, controllers, men-machine-interfaces) in industrial automation. The main requirements of this type of application are configurability, flexibility and extendibility. To provide manufacturer-independence the definition of device functionality has to be specified in form of "Devices Profiles". Accordingly, communication systems solution of that type provide a complete

[3] CANopen modules which support "extended bootup" may be used within CAL-based systems

framework of communication and systems services, device modelling, facilities for system configuration and device parametrising.

The **CANopen** standard was specified by a group of CiA (CAN-in Automation) members based on the results of an EU-research program. CANopen uses a subset of CAL (CAN Application Layer) for communication and system services as well as network management. Device modelling is based on a description of the device functionality by means of an object directory. This approach widely corresponds to the form of device description used by other field busses (Interbus-S, Profibus). Standard devices are specified in form of "Device Profiles". The CANopen-standard is supported by the interest group CiA, device profiles are specified by special interest groups within the CiA

DeviceNet™ is a very sophisticated open network developed by Allen-Bradley. It is defined in terms of an abstract object model which represents the available communication services and the external visible behaviour of a DeviceNet node. The DeviceNet standard is managed by an independent supplier organisation (ODVA, Open DeviceNet Vendor Association) which also supports the worldwide marketing of DeviceNet. The behaviour of alike devices is specified in corresponding device profiles

SDS™ (Smart Distribution Systems) is an open network standard developed by Honeywell Micro Switch. Based on a specific Application Layer Protocol an object-oriented hierarchical device model is defined to make interoperablity between SDS devices possible. SDS is especially designed for distributed binary sensors and actors.

3 Main Items of CAN-based Higher Layer Protocols

In the following the main solutions for industrial automation CAL/CANopen, DeviceNet and SDS will be evaluated closer. This will be done by considering the main items of CAN-based higher layer protocols. These are the

- message identifier assignment system,
- method of exchanging Process Data,
- peer-to-peer communication,
- method of establishing Process Data connections,
- network management,
- principle of device modelling and device profiles

3.1 Message Identifier Assignment System

The method of message identifier assignment may be regarded as the major architectural element of CAN-based systems, since the identifier of a CAN-message determines the relative priority of the message and such the message latency time. It also has influence on the applicability of message filtering, possible communication structures and the efficiency of identifier usage.

Concerning identifier assignment quite different philosophies have been chosen in the considered system solutions. Whereas CAL and CANopen, apart from reserving some identifiers for management purposes[4] do not apply a predefinition of identifiers for general system structures[5], DeviceNet and SDS do.

---

[4] In the so-called minimum-device configuration CANopen also provides a simple identifier assigment rule for systems with 1:n communication structure

[5] CANopen defines a "predefined connection set" for 1:n system structures

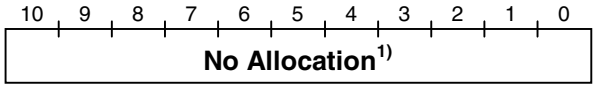
CAL/CANopen provide a common pool of identifiers, available to all devices and a central instance which automatically or manually allocates identifiers according to the requirements of the devices. With this approach, identifier usage and such the real-time-behaviour of the data communication system may be completely determined through the system designer or integrator. Also maximum usage of the available identifiers is possible since almost the complete set of message identifiers is available for distribution.

In Fig. 3.1-1a and 3.1-2a the identifier assignment scheme of general CAL/CANopen systems is shown, in Fig. 3.1-1b and 3.1-2b the identifier assignment of a minimum configuration CANopen system is shown with the predefined set of messages. 1760 message identifiers are available for general usage. Since in a CAL-based system up to 256 nodes may be addressed, 256 messages are reserved for node guarding, in CANopen 128 nodes may be addressed, only 128 messages are reserved for node guarding.

In the minimum system configuration, CANopen specifies a device-oriented identifier allocation scheme by which default connections between up to 127 devices to a master device are provided. By means of the 4-bit function code 16 basic functions are distinguished for the reception and transmission of two process data channels, one peer-to-peer channel, node state control, node guarding, emergency notification and the reception of the synchronisation and time stamp message. Since the priority of a message should be determined by its function, the function code is located in the most significant bits of the message identifier (Fig. 3.1-2).

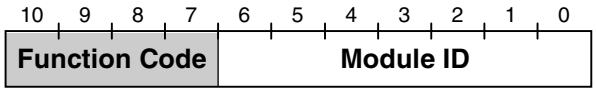
With the CANopen minimum system configuration a 1:n communication structure is supported per default. By means of identifiers not used by the predefined connection set also direct connection between devices may be established.

a CAL/CANopen

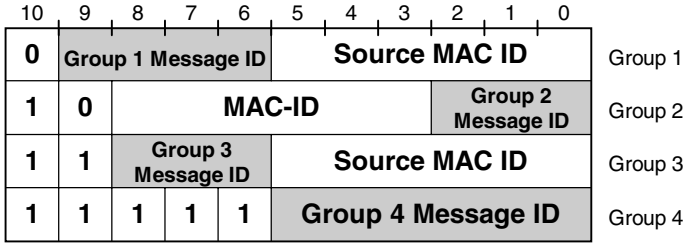


1) Some identifiers reserved for management

b CANopen - Minimum Device

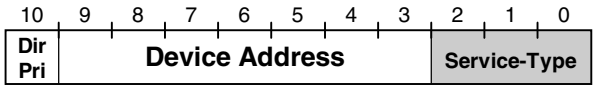


c Device Net



MAC-ID: Node Identification Number

d SDS



Dir/Pri: Direction Bit, determines if Device Address is destination or source address resp. Priority Class

Service-Type: specifies type of service

Figure 3.1-1: Identifier Usage in CAL/CANopen, DeviceNet and SDS

a CAL/CANopen

Identifier	Usage
0	NMT Start/Stop
1	CMS Objects (8 Priority Groups)
1760	
1761	Node Guarding
2015	
2016-2031	NMT-,LMT-,DBT-Services

b CANopen Minimum Configuration

Identifier	Usage
0	NMT Start/Stop
128	Sync-Message (Rx)
129	Emergency Messages
255	
256	Time-Stamp-Message
385	Process Data Objects (Tx)
511	
513	Process Data Objects (Rx)
639	
641	Process Data Objects (Tx)
767	
769	Process Data Objects (Rx)
895	
1409	Service Data Objects (Client)
1535	
1537	Service Data Objects (Server)
1663	
1793	Node Guarding
1919	
2016-2031	NMT-,LMT-,DBT-Services

c DeviceNet

Identifier	Usage
0-63	Device n
	Device n
	Device n
	Device n
	Device n
	16 Group 1 Messages per Device (64 Devices)
-1023	Device n
1024-1031	Device 0
1032-1039	Device 1
	Group 2 Messages
1528-1535	Device 63
1536	Device n
	7 Group 3 Messages per Device
1983	Device n
1984-2031	Group 4 Messages

d SDS Long Form APDU

Identifier	Usage	
0	SourceAddr 0	High-Prio Channel Serv
1	DestAddr 0	Connect Serv Req
2-3	reserved	
4-7	DestAddr 0	Read/Write/Action/Event Serv Req
1000	SourceAddr 125	High-Prio Channel Serv
1001	DestAddr 125	Connect Serv Req
1002-1003	reserved	
1004-1007	DestAddr 125	Read/Write/Action/Event Serv Req
1008-1023	Not specified	
1024	SourceAddr 0	Low-Prio Channel Serv
1025	SourceAddr 0	Connect Serv Resp
1026-1027	reserved	
1028-1031	SourceAddr 0	Read/Write/Action/Event Serv Resp
2024	SourceAddr 125	Low-Prio Channel Serv
2025	SourceAddr 125	Connect Serv Resp
2026-2027	reserved	
2028-2031	SourceAddr 125	Read/Write/Action/Event Serv Resp

Figure 3.1-2: Usage of Identifiers in CAL/CANopen, DeviceNet and SDS

One of the basics of the **DeviceNet** identifier assignment scheme is the node-oriented ownership of message identifiers[6]. Each of the maximum 64 nodes of a DeviceNet system owns a set of identifiers out of 3 message groups (Fig. 3.1-1c). Message group 1 provides a high priority message pool of 16 messages per device, message group 3 5 low priority identifiers per device[7] . The identifiers/priorities of that groups are distributed evenly among all the devices on the network (Fig. 3.1-3a). The reservation of message identifiers for the maximum number of devices implies, that for networks of less than 64 nodes, the identifiers of the unused nodes are not available for the system.

With Message Group 2 it was intended to support devices with limited message filtering capabilities due to Basic-CAN type controllers[8]. Therefore, a filtering according to the node number (MAC-ID) was chosen. This means that the priority of messages of that group is primarily determined by the node number. Two messages of that group are reserved for management tasks (allocation of predefined connection set, Duplicate MAC-ID check). The MAC-ID of Group 2 messages may be destination or source address.

DeviceNet specifies a so-called "Predefined Master/Slave Connection Set" to facilitate the communication observed in a Master-Slave system configuration. Fig. 3.1-3 shows the identifier assignment of this set. The following channel functions are supported for exchanging of I/O and Explicit messages between a Slave and a Master device based on the predefined connection set:

- Explicit Message channel
- Master Poll/Change of State/Cyclic channel
- Slave I/O Change of State/Cyclic channel
- Bit Strobe channel

The Explicit Message Channel mainly serves for configuration of a device. With the Master Poll/Change of State channel the master can request I/O data from the device and send Output data to the Slave device. With the Slave I/O Change of State/Cyclic a Slave device can transmit Input data to the Master,

IDENTIFIER BITS											**IDENTITFIER USAGE**
10	9	8	7	6	5	4	3	2	1	0	
0	**Group 1 Message ID**				**Source MAC-ID**						**Group 1 Messages**
0	1	1	0	1	Source MAC-ID						Slave's I/O Change of State or Cyclic Message
0	1	1	1	0	Source MAC-ID						Slave's I/O Bit-Strobe Response Message
0	1	1	1	1	Source MAC-ID						Slave's I/O Poll Response or Change of State / Cyclic Acknowledge Message
1	0	**MAC-ID**						**Group 2 Message ID**			**Group 2 Messages**
1	0	Source MAC-ID						0	0	0	Master's I/O Bit-Strobe Command Message
1	0	Source MAC-ID						0	0	1	Reserved for Master's Use - Use is TBD
1	0	Destination MAC-ID						0	1	0	Master's Change of State or Cyclic Acknowledge Message
1	0	Source MAC-ID						0	1	1	Slave's Explicit Response Messages
1	0	Destination MAC-ID						1	0	0	Master's Explicit Request Messages
1	0	Destination MAC-ID						1	0	1	Master's I/O Poll / Change of State / Cyclic Message
1	0	Destination MAC-ID						1	1	0	Group 2 Only Unconnected Explicit Request Messages
1	0	Destination MAC-ID						1	1	1	Duplicate MAC-ID Check Messages

Figure 3.1-3: Device Net Predefined Master/Slave Connection Set Identifier Assignment

[6] for transmitting purposes. Principally a node may receive all of the messages of the system

[7] 3 identifiers of message group 3 are reserved for management tasks

triggered by change of state, cyclically or by the Slaves application. By means of a Bit Strobe command the Master can request input data from any of up to 64 Slave devices with only one message. Since all of these messages are acknowledged 8 message identifiers are allocated for these functionalities. If Bit Strobe requested data acquisition is not used a very effective identifier filtering on the Slave devices is possible by means of the destination address field.

In Fig. 3.1-1d the identifier assignment scheme of **SDS** is shown. This is very similar to the DeviceNet group 2 messages and provides a completely predefined set of messages for different purposes. SDS distinguishes two forms of messages[9]: short form and long form. Long form APDUs can be non-fragmented or fragmented. Short form messages do not contain any data byte and are used for a very efficient interfacing with single binary I/O devices.

The meaning of a SDS message is determined by three elements:

- Dir/Pri bit
- Logical address (7 bit)
- Service Type (3 bit)

The meaning of the Dir/Pri bit depends on the message type. For a short form message[10] this bit indicates the direction of the message (Dir =0: Logical Address = Destination Address, Dir =1: Logical Address = Source Address).

For long form messages the meaning of the Dir/Pri bit is defined by the Service Type field. For service types Read, Write, Action, Event and Connection, this bit determines the direction with respect to the content of the logical address field like in the short form ADPU. To support simple identifier filtering by simple CAN-controllers message priorities are determined by the device number. For these types of services all destination addressed messages therefore have higher priority than source addressed messages.

For service type "Channel" (Multicasting) the logical address is always a source address and the Dir/Pri bit specifies two priority classes (high/low).

The Logical Address field permits Logical Addresses from 0 to 125. For the Channel service the logical address always contains the source address.
Because the Address field is located in the upper part of the message identifier, lower address devices always have higher priority than higher address devices.

The 3-bit Service Type Field distinguishes between 8 different services each for Short Form messages and Long Form messages. Long Form services include: Channel, Connection, Write, Read, Action and Event. The Read, Write, Event and Action services are used to read or modify an attribute of an Embedded Object, report an occurrence of an event specified for an Embedded Object or to execute the actions specified for an Embedded Object. Short Form services are specialised Event and Write services and are used for reporting or commanding of 1-bit event/action like "Change of State Off" or "Write ON State" in a very efficient way. With V2.0 of the SDS specification also the Connection and Channel services are provided. The Connection service is used to establish a Device Connection between an Initiating Device and a Responding Device. The Channel Service provides Multicast and bi-directional Peer-to-Peer communication channels for the requester.

Since SDS uses the identifier for the specification of device addresses and services and not as unique data tag, additional information is necessary to uniquely identify the transmitted data. This reduces the data transfer efficiency of the protocol. The complete data tag is given in SDS by the Logical Device address, Embedded Object ID and Attribute/Event or Action ID. Since all services but the

---

[8] These types of CAN controllers only allow a message filtering of the 8 most significant bits of the identifier

[9] In SDS terminology: APDU (Application Layer Protocol Data Unit)

[10] A Long Form APDU is distinguished form a Short Form APDU in that the long form has a non-zero Data Length Code

Channel service are confirmed by the application also a Service Specifier information field is necessary to specify whether the message is a request, success, error or wait response. This additional information is transmitted in the first two bytes of the CAN data field (Fig. 3.2-3).

3.2 Exchange of Process Data

The transmission of process data between the devices of a distributed automation system is the purpose of a CAN-based communication system. This should be accomplished in the most efficient way. Therefore transmission of application specific data (process data, I/O data) should be performed according to the producer-consumer model, with the meaning of the transferred data implied by the associated message ID. Producer and consumer of a message in that case are assumed to have knowledge of the intended use or meaning of the transmitted data.

In the following, the main characteristics of the different solutions for exchanging of process data will be outlined for CAL, CANopen, DeviceNet and SDS.

CAL is intended to provide standard, application-independent communication facilities for the implementation of distributed systems. It provides communication objects (CMS objects) in terms of "Variables", "Events" and "Domains". CMS objects are specified by a set of attributes and are identified by symbolic names. Objects and services of CAL are directly accessible by the user application. Variables may be of type "Basic" or "Multiplexed". With Basic Variables and Events, the multicast transmission of up to 8 bytes of data is provided, without any protocol overhead. Multiplexed Variables contain a multiplexer within the first data byte to distinguish between 128 "Multiplex-Variablen" per message identifier and allow the transmission of 7 bytes of data. Basic-Variables also may have different access type (read-only, write-only, read-write). With a read-write Variable, an acknowledged transfer of data between two devices is supported. The transmission of a Variable is initiated by a client, the transmission of an Event is initiated by a server[11] of the corresponding object. Fig. 3.2-3a shows the "Store-And-Immediately-Notify-Event"-protocol, Fig. 3.2-3b the "Read-Event"-protocol, by which a client also can read previously stored data.

Basic and Multiplexed Domains support the acknowledged transmission of data of more than 8 bytes by means of a flow-controlled fragmented protocol. Since each data segment is acknowledged a receiver-controlled flow control is implicitly provided. With the 3 byte Domain-Multiplexer a variety of different domains may be identified per message identifier. The transfer of a data block is initiated by an "Initiate"-request/response sequence, following data segments are transmitted by means of a Data Segment request/response sequence. Fig. 3.2-1 shows the principal structure of the data field of a "Initiate-Multiplexed-Domain" request and a "Download-Segment" request. In the Control byte the message type (Initiate/Download Segment/Upload Segment), the transfer type (expedited/non-expedited), toggle-bit and number of bytes with no data is indicated. With

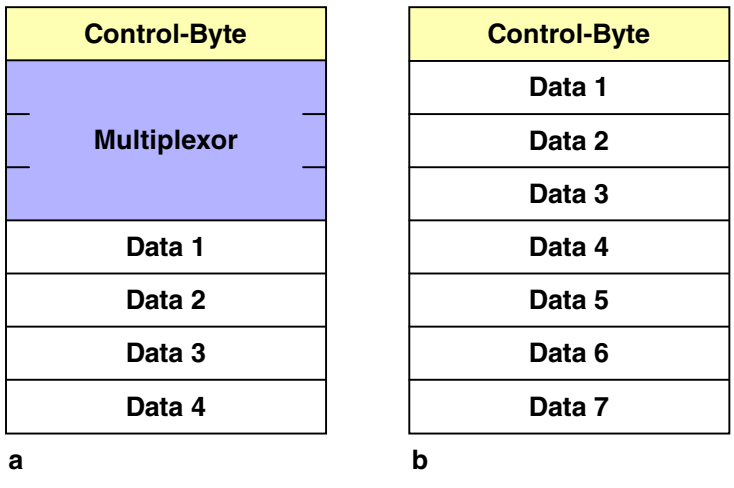


Figure 3.2-1: Structure of the Data Field of an Initiate_Multiplexed_Domain.request (a) and a Download_Segment.request (b)

[11] A Server in term of CAL is understood as the owner of the physical object, a Client uses the services of a Server

the expedited (non-fragmented) protocol the transmission of up to 4 bytes of data is possible.

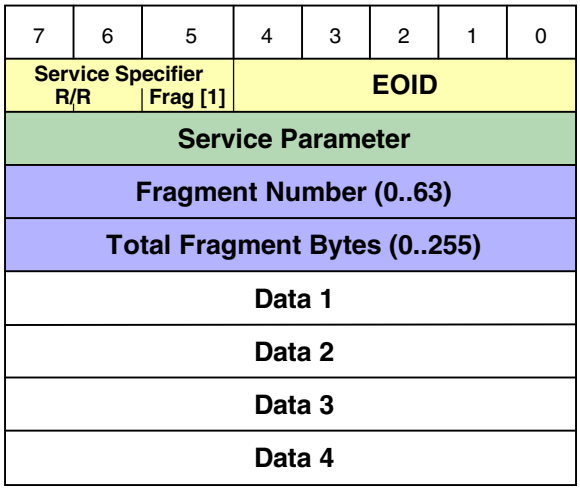
CANopen and **DeviceNet** at a first glance provide quite similar communication mechanisms for transferring process and service/configuration data. With CANopen, the transmission of process data occurs by means of so-called "Process Data Objects (PDOs)", with DeviceNet by means of "I/O-messages".

In Table 3.2-1 the main characteristics of process data exchange are summarised for CANopen, DeviceNet and SDS. One of the main differences is the provision of an unacknowledged fragmentation protocol in DeviceNet and SDS, which makes it possible to transmit also process data with a length more than 8 bytes. Also three different protocols with respect to acknowledgement ("Transport Classes") are supported (Fig. 3.2-4) and determined of the "Transport Class" of a Connection end point. Transport class 2 or 3 may be used for example for efficient "polling" of devices. For that purpose a master device implements the communication resources (connection objects) associated with each Poll Command as a Client Transport Class 2 or 3. Each slave implements a Server Transport Class 2 or 3 Connection Object to receive the Poll Command and to transmit the associated response data.

SDS originally was designed for providing an optimised solution for distributed systems with single binary devices. Therefore a short form message is used, which uses a 3-bit part of the message identifier (Service Type) to report or modify the status of a single bit device.

If more information is to be transmitted, the long form message is used. Read and Write services operate on defined object attributes of a particular device. An Action indicates that the device is to perform a specified function. Events provide for notification of spontaneously generated events in a device or events sensed by the device. Each of these services expects an acknowledge/response by the application of the remote device.

Fig. 3.2-2 shows the structure of the data field of a long form, fragmented SDS APDU. In the first data byte the Service Specifier[12] determines if the message is a request, positive or an error response, the Fragment Indicator bit indicates if the message is fragmented and the Embedded Object ID field indicates which of the possible 32 Embedded Objects of the logical device is addressed. In the next data byte the number of the attribute, action or event of the addressed Embedded Object is specified. The remaining 4 bytes are free for transferring data. In a non-fragmented message 6 byte of data per message can be transferred.



Service Specifier
R/R Request / Response / Error / Wait
Frag Fragment Indicator

EOID Embedded Object ID

Service Parameter
Attribute- / Action- / Event-Number

Figure 3.2-2: Data Field of a SDS fragmented CAN frame

With version 2.0 of the SDS specification also multicasting of process data is introduced by SDS by means of so-called "Channel APDUs". Channel APDUs always are transmitted with the address of the initiator (source address) and Service Type = Channel in the message identifier. Channel APDUs may be transmitted unfragmented and fragmented. Unfragmented APDUs are unacknowledged and contain a Channel Number (0-31) and Channel Code (Multicast, Peer-to-Peer) in the second data byte. Channel Numbers are typically established during configuration. The remaining 6 byte of the CAN data field are available for the transmission of process data.

[12] The service type (Read, Write, Action, Event, Connect, Channel) is indicated in the message identifier

The structure of the fragmented Multicast APDU is the same as for Read, Write, Action and Event (Fig. 3.2-2). Since the fragments are transmitted back-to-back and only the transmission of the complete data block is acknowledged no flow control is implied. Therefore, and to allow bus access by other devices, an appropriate delay between succeeding fragments is recommended.

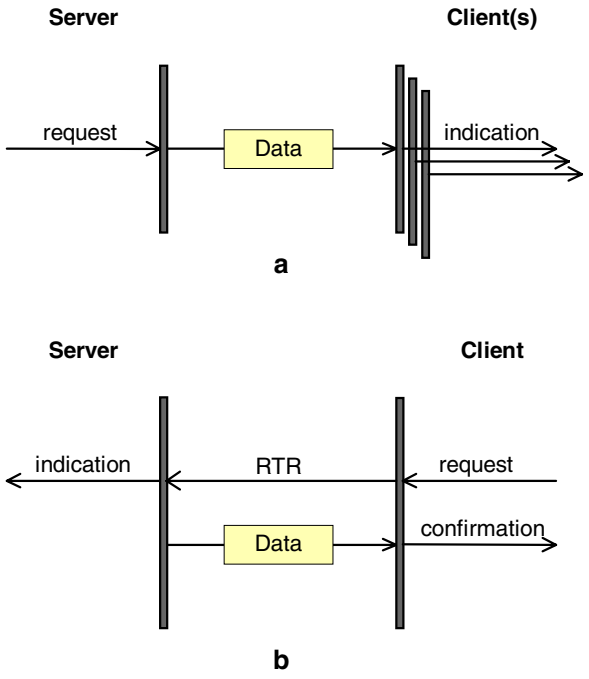


Figure 3.2-3: Store-and-Immediately-Notify-Event-Protocol (a), Read-Event-Protocol (b) (CAL/CANopen)

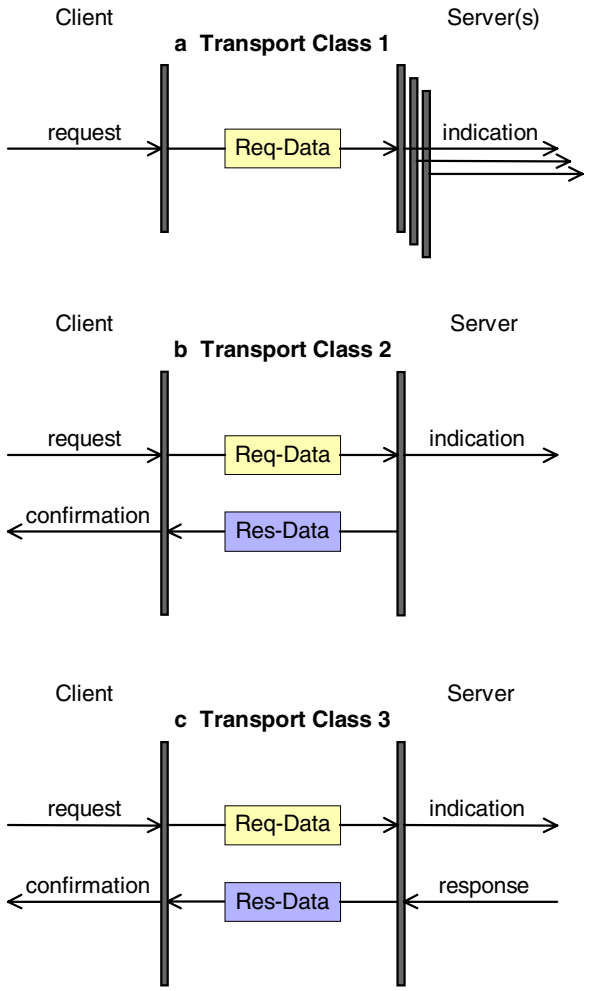


Figure 3.2-4: Device Net Transport Classes

Message Triggering

All of the regarded protocols provide alternative modes of message triggering supported by the Application Layers.

DeviceNet supports the triggering modes Cyclic, Change-of-State and Application Object Triggered. With Cyclic Triggering the expiration of a message-specific Transmission Trigger Timer starts the transmission of a message. With Change-of-State the transmission occurs when a change-of-state of an Application Object is detected. A message is also transmitted when a specified time interval has elapsed without a transmission. With Application Object Triggering the Application Object decides when to trigger the transmission. A message is also transmitted when a specified time interval has elapsed without a transmission

CANopen distinguishes between triggering On Event, Application Request or after Reception of a predefined Synchronisation Message[13].

Triggering On Event may occur on an profile- or application specific event ("Asynchronous PDO"). The transmission of a PDO may also be triggered by the reception of a remote request message (Remotely Requested). "Synchronous PDOs" are cyclically triggered by the reception of a specified number of Synchronisation Messages.

The Synchronisation Message also may be used for network wide synchronisation of data acquisition and the strobing of output data.

Also in **SDS** different triggering modes (e.g. cyclic, state-of-change/value) are provided by configuring the corresponding attributes of the Embedded Objects.

Mapping of Application Objects

Network devices normally will produce and/or consume more than one Application Object and assembling of more than only one Application Object within one PDO respectively I/O-Message will be appropriate.

In **CAL**-based applications, the mapping of application data is done by the programmer when defining communication objects (e.g. CMS Variables or Events).

SDS describes the mapping of the I/O data of an Embedded Object into the data field of an APDU by means of the so-called "Network Data Descriptor". This information can be read by other devices.

CANopen and DeviceNet provide very sophisticated means for a flexible mapping of application data into communication objects.

CANopen specifies the mapping of Application Objects into a PDO by means of a data structure called "PDO-Mapping Record". This structure specifies the mapped application object data in form of a list of object identifications (Object Directory index/subindex) and data length. Since the PDO Mapping is accessible by means of SDOs, PDO mappings are configurable by means of a configuration tool.

In **DeviceNet** the grouping of Application data is specified by means of instances of the "Assembly" object, which defines the format of the transmitted application object data. A device may contain more than one I/O assembly and the selection of the appropriate assembly (consumed/ produced_connection_path) may be a configurable device option.

	CANopen	DeviceNet	SDS (V2.0)[14]
Name of Communication Object	Process Data Object	I/O-Message	Multicast Channel APDU
Maximal Number of Communication Objects per Device	512 Transmit PDOs 512 Receive PDOs	27 I/O- Transmit Messages[15] 1701 I/O Receive Messages per device	32 Multicast Channels for each of up to 32 Embedded Objects per device

[13] This implies that in the network a "Synchronisation Client" instance is active.

[14] SDS V1.8: Only Acknowledged point-to-point transfer of process data by means of Write, Read, Event and Action services.

[15] Since a device can transmit messages from other devices message pools via message Group 2, the actual amount of transmit messages can exceed 27.

	CANopen	DeviceNet	SDS
Maximal length of Data Field	8 bytes	8 bytes fragmented: Arbitrary length	6 bytes fragmented: 64 * 4 bytes
Protocol	Unfragmented: No overhead, Notify/Read "Stored-Event"-protocol (CAL/CMS) Unacknowledged	Unfragmented: No overhead, three "Transport Classes" supported: • Unacknowledged, • Acknowledged by Server Connection Object, • Acknowledged by Application Fragmented: Unacknowledged fragmented protocol 1 byte protocol overhead per frame	Unfragmented: 2 byte protocol overhead, Unacknowledged Fragmented: Acknowledged fragmented protocol with Acknowledge after reception of complete block 4 bytes protocol overhead per fragment
Message Production Triggering Modes	- On Request of local or remote application - Cyclic/acyclic synchron	- Cyclic - Change-of-State - Application specific	Specified by object model
Mapping of Application Objects	Maximum number of mappable application objects/PDO dependent on data size of objects (1-bit objects: 64 application objects mappable) Definition of Application objects by means of "Mapping Parameter Record" (configurable) Dynamic mapping supported	Arbitrary number of Application objects mappable with fragmented protocol Definition of Application objects by means of Assembly Object (several Assembly Objects possible) Dynamic mapping supported	Network Data Descriptor defines size, granularity and data type of I/O data of Embedded Object (V1.8)

Table 3.2-1: Exchange of Process Data in CANopen, DeviceNet and SDS (Multicasting)

3.3 Peer-to-Peer Communication Channels

For configuration of devices by means of a configuration tool, specific device functions or program loading multi-purpose communication channels are required. These non-time-critical communication channels always exist between two devices, e.g. between a configuration tool and the device to be configured. The transfer of data has to be performed by means of an acknowledged fragmentation protocol. Any of the different higher layer protocols which support some kind of device configuration provide this kind of peer-to-peer communication facility.

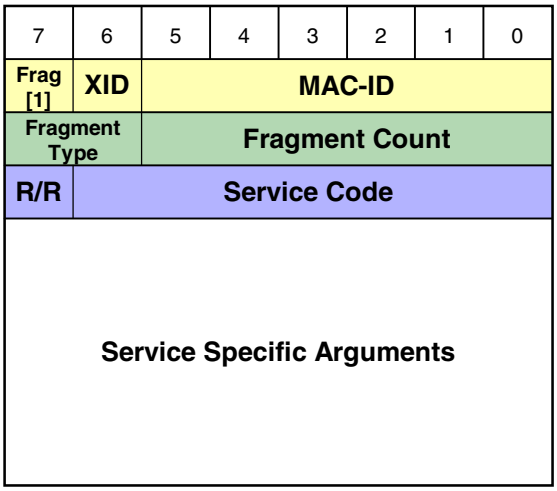
CAL for that purpose provides "configuration services" across predefined management channels to each device as part of the CAL Network Management service element. For that purpose two identifiers are reserved, the addressed device is specified in the data field of the first fragment of a message by its node ID

CANopen provides so-called "Service Channels" across which "Service Data Objects" (SDOs) may be exchanged between any two devices according to the CAL Multiplexed Domain protocol. This protocol provides the acknowledgement of any frame transmitted. Within the first three bytes of the data field of the Initiate-Domain-Request the address of the Object Directory entry is specified by means of an 16-bit index and 8-bit subindex. With the index/subindex of an Object Directory entry the function to be performed is specified implicitly.

Data of less than 5 bytes may be transferred with only the Initiate-Domain Request Frame ("expedited protocol"), if more than 4 bytes of data have to be transmitted, the acknowledged fragmented protocol has to be applied, with 7 bytes of data per fragment. Each CANopen device has to provide a default server SDO-connection with two predefined message identifiers according to the predefined connection set. Across this default server SDO connection a device may be accessed by a configuration tool.

For applications which require a dynamic establishment of SDO connections (e.g. between test tools and devices) the "SDO-Manager" instance is introduced [10]. The SDO Manager is the owner of the predefined set of SDO connections and therefore has access to any device on the network. A SDO-connection requesting device first has to address the SDO Manager and to ask for establishing the requested connection.

DeviceNet provides multi-purpose device-oriented channels and services. Writing and Reading of object attributes, control of objects (reset, start, stop etc.), storing/restoring of classes/objects attributes etc. is performed by means of "Explicit Messages". These are exchanged across "Explicit Messaging Connections". The meaning/intended use of an Explicit Message is stated in the CAN data field. In Fig. 3.3-1 the data field format of a fragmented Explicit Message is shown. For unfragmented



Frag	Fragment Bit
XID	Transaction ID
MAC-ID	Source or Destination MAC-ID
Fragment Type	Indicates first, middle or last fragment
Fragment Count	
R/R	Request /Response
Service Code	Type of Service

Figure 3.3-1: DeviceNet Fragmented Explicit Message Data Field Format (Request/Response)

transfer the "Fragment Byte" is not transmitted. For a service request normally the access path (class number, instance number, attribute number) of the addressed object attribute is specified (Service specific arguments).

An Explicit Message Connection has to be established by means of the "Unconnected Message Manager (UCMM)". The UCCM provides two services for opening and closing of an Explicit Message Connection. Each device supporting an UCMM reserves message identifiers for transmitting UCMM request and response message. For "Group 2 Only" devices (devices not supporting an UCMM port) a master device first has to allocate the Explicit Messaging Connection of the devices' Predefined Connection Set. The request to allocate a Group 2 Only device is transmitted as a Group 2 Only Unconnected Explicit Request with a reserved message identifier.

SDS Channel services provides direct peer-to-peer communication channels between any Embedded Objects on different Logical Devices. This allows to perform Read, Write, Action, Event messaging between any two peer-to-peer connected devices. For each Embedded Object 4 channels may be specified for general purposes and configuration. In a unfragmented peer-to-peer channel APDU 5 bytes of data may be transmitted, in a fragmented APDU only 3 bytes.

A connection must be established with the responding device by means of the Connection service before a Peer-to-Peer channel may be opened and successfully used for communication. For establishing connections between devices a Connection Manager is necessary in the network. An initiating device sends a Connection Request message to initiate the connection establishment process. All Connection Requests are serviced by the Connection Manager. The Connection Manager relays the request, as a relayed Connection Request to the responding device. The responding device acknowledges the request with a response message that is also serviced by the Connection Manager. Finally, the Connection Manager forwards a relayed Connection Response message to the initiating device, indicating that the responding device has established the connection.

In Table 3.3-1 the main characteristics of peer-to-peer communication channels of CANopen DeviceNet and SDS are summarised.

	CANopen	DeviceNet	SDS (V2.0)
Name	Service Data Channel	Explicit Message	Peer-to-peer Channel
Maximum number of channels	128 Client SDOs, 128 Server SDOs per device	27 Explicit Transmit Messages[16] 1701 Explicit Receive messages per device	4 channels per Embedded Object. 32 Embedded Objects per Logical Device
Protocol	< 5 byte: Acknowledged unfragmented > 4 byte: Fragmented transmission (7 bytes per fragment) Each frame acknowledged Any length (CAL Multiplexed Domain protocol)	< 7 byte: Acknowledged unfragmented > 6 byte: Fragmented transmission. (6 bytes per fragment) Each frame acknowledged Any length	< 6 bytes Acknowledged unfragmented > 5 byte Fragmented transmission (3 bytes per fragment) Acknowledgement of complete data block. Max. 255 byte

[16] Since a device can transmit messages from other devices message pools via message Group 2, the actual amount of transmit messages can exceed 27.

	CANopen	DeviceNet	SDS
Establishing of Connections	- Dynamic establishment by means of SDO Manager - Default predefined connections	- Dynamic establishment by means of Unconnected Message Manager - Group 2 Only devices: Allocation of Explicit Message from Predefined Connection Set	- Dynamic establishing by means of Connection Manager - Master/Slave Set of Connections Set
Connection Services and Arguments	Initiate, Abort Upload/Download Segment/Domain Index and Subindex of addressed Object Directory Entry	Open/Close Creation, Configuration, Start, Stop, Reset etc. of Objects Object attribute access path, Service arguments	Open/Close Read, Write, Event, Action Channel Number, Attribute/Action/Event Identifier

Table 3.3-1: Main Characteristics of Peer-to-Peer Communication Channels

3.4 Establishment of Process Data Message Connections

Allocation of identifiers to the transmit messages of the message producers respectively receive messages of the message consumers establishes the communication paths in a CAN network. Establishing message connection is possible through usage of predefined messages with already allocated message identifiers or through a variable allocation of identifiers to messages.

In a system with predefined messages "functions" and identifiers of messages are already defined. For example, if a **SDS** Master device wants to read the attribute value of an Embedded Object in a Slave Device it uses a message identifier with direction bit equal to 0, with the Slave device`s address in the logical address field and the Read Service code in the service type field. The specification of the Object and attribute ID has to be done in the data field of the message. SDS Master-Slave mode therefore does not need any means for allocation of identifiers. With the extension of SDS by the Channel service each device in addition owns 1 high and 1 low priority multicast message. Also these messages are predefined but their usage is more flexible since arbitrary multicast connections may be built by means of the Connection service.

DeviceNet and CANopen also make use of a predefined connection set approach for 1:n system structures. A DeviceNet Master for example which has allocated a Slave device`s predefined poll-connection already "knows" the message IDs for transmitting the poll request and expecting the poll response message since they are derived from the Slaves MAC-ID, according to the predefined set. Similarly in CANopen the default predefined connection set, besides of other predefined messages, provides two predefined Receive and Transmit PDOs. The usage/meaning of the Default-PDOs is determined by the device type.

The main advantages of a non-predefined identifier allocation is the possibility of establishing any type of communication structure, the availability of the maximum number of message identifiers and the design-controlled allocation of message identifiers according to the requirements of the application.

Whereas **CAL/CANopen** are based on a variable identifier allocation scheme based on a central message identifier pool, DeviceNet distributes the available identifiers across the maximum 64 devices of a DeviceNet system.

The allocation of identifiers with a common identifier pool is controlled by a specific network instance (the Distributor in CAL) or by means of a configuration tool (CANopen[17]) which supports the building of message connections by the system administrator.

CAL and CANopen are based on a variable identifier allocation with a common identifier pool. Identifier allocation in CAL-based systems may be performed by the "Distributor" service element (Fig. 3.4-1). The Distributor (DBT) master instance allocates message-IDs from a central pool of message-identifiers according to the requests (priority group, name and type (transmitter/receiver)) of all of the communication objects of the devices. By linking of requests according to object name and type client(s) and server (s) of a message are connected. The distribution process is controlled by a Network-Master application across network management connections to the devices.

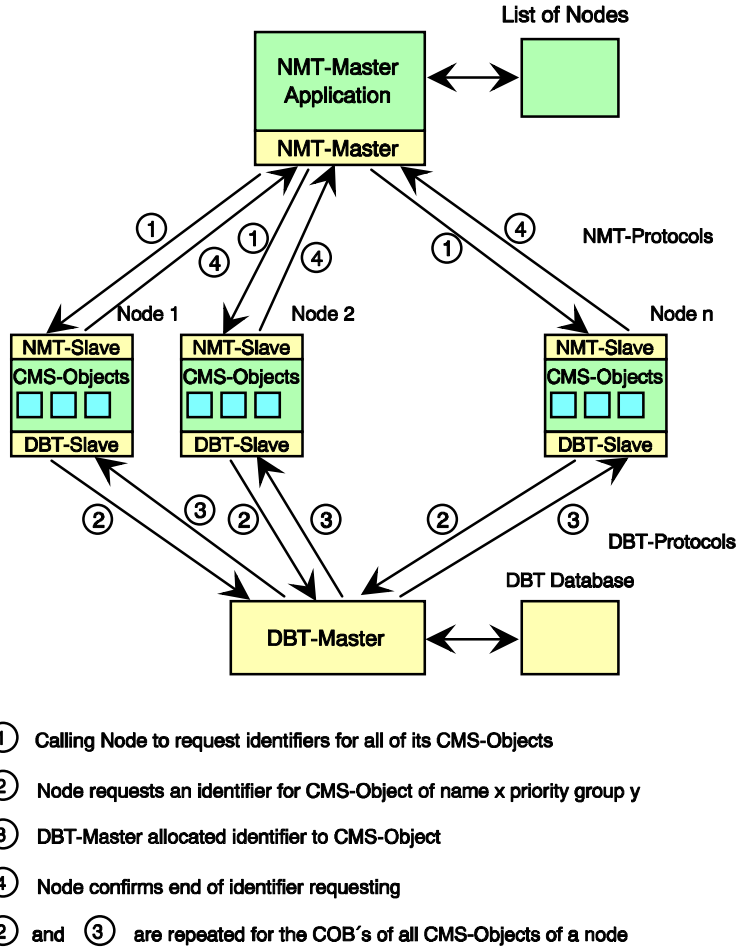


Figure 3.4-1: Identifier Allocation Process with CAL-Distributor

If no CAL-Distributor is used, configuration of message identifiers in CANopen based systems may be performed by setting the corresponding PDO identifiers in the Object Directory of the devices[18] via a SDO channel.

The generic identifier allocation method of **DeviceNet** is determined by the fact that here the devices are owner of message identifier pools. Therefore the connection of I/O messages first requires the allocation of an identifier out of the identifier pool of the message transmitting device. This identifier then has to be assigned to the consuming device(s).

In Figure 3.4-2 the process of establishing a I/O Message Connection between two devices by means of a configuration tool is illustrated. I/O connections are established by addressing the Connection Class across an already established Explicit Messaging Connection. This involves creation of an I/O Connection object and configuring the Connection instance at the end point of the connection[19]. During that process a message producing module allocates a free message-ID from the pool of its message-IDs and combines this with its Source MAC ID to generate a so-called "Connection ID". The "automatic" allocation of an identifier out of the message group may be overridden by a direct modification of the Connection ID attributes.

[17] CANopen also may apply a CAL Distributor entitity for dynamic identifier distribution

[18] Alternatively a configuration tool may apply a specific rule of automatic identifier allocation

[19] I/O connections can be either point-to-point or multicast

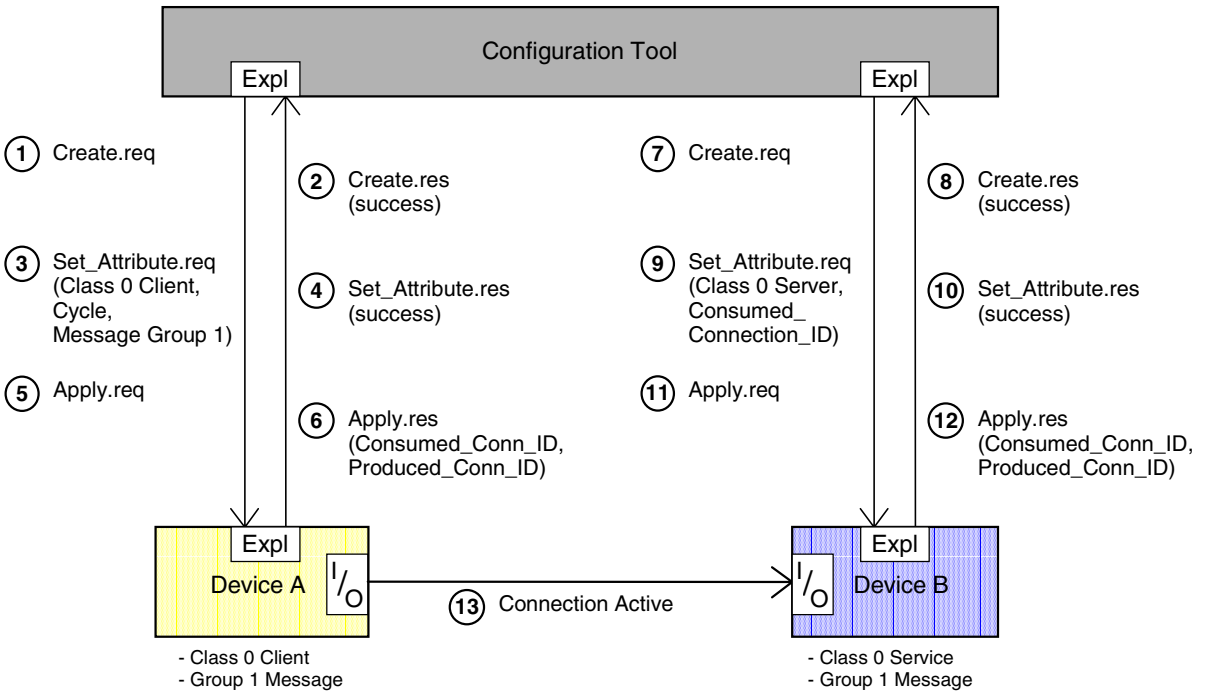


Figure 3.4-2: Creation, Configuration of I/O Connection Instances and Establishing of an I/O Message Connection (simplified) Explicit Message Connections already established

3.5 Network Management

Due to the fact that an application is distributed, certain events have to be handled (e.g. failure of parts of an application or failure of a node) which would not occur if the same application had not been distributed. Main tasks of an appropriate network management therefore are the detection and indication of failures in the network and services which allow to control the communication status of the distributed nodes in a co-ordinated manner. Depending on the system solution, network management functionality is provided by means of an explicit Network Management facility or implicitly by means of other measures.

CANopen network management is based on the CAL NMT service element which applies the principle of "Node Guarding" for the detection of node failures. For this purpose, a NMT master application cyclically transmits a guard request to each node (NMT slave) of the network by means of a Remote-Request Frame. The addressed slave responds to each request with its actual communication state. If the NMT master detects a change in the node state or no response from the addressed node, a guard error is indicated to the NMT master application. Node guarding starts, when a node is connected to the network. Each node also supervises the arrival of its guard request message. If there is no further guard request after expiration of the nodes "life time" a network error is signalled to the nodes application.

Co-ordination of the communication status of the nodes is also supported by the NMT master instance. Fig. 3.5-1 shows the node state transitions diagram of a CANopen node[20]. After power on a node initialises and transits to the "Preoperational State". In this state communication across SDO channels is possible for node configuration[21], but not yet across PDOs. With the NMT message "Start Remote Node" a selected or any nodes on the network can be set into the "Operational State". In this

[20] Simplified Boot-up shown. If the DBT facilitiy is implied, a device has to support an extended boot-up sequence.

[21] This also includes configuration of communication objects and linking if necessary

state also the exchange of data by means of PDOs is possible. With enabling the operation of all nodes of a network at the same time a co-ordinated operation of the communicating system is secured.

According to its connection-oriented design, in **DeviceNet** each connection is supervised. Therefore each receiving connection end point owns an "Inactivity/Watchdog-Timer" to supervise the arrival of a message according to the configured "expected packet rate". If the timer expires the connection performs the specified "Timeout Action". Fig. 3.5-2 shows the state transitions diagram for a I/O connection object. After reception of a Create Service (Explicit Message) the connection is configured by applying the appropriate sequence of Explicit Message services and enabled after the complete connection has been configured.

Prior to getting access to the network every DeviceNet node has to perform the so-called "Duplicate MAC ID Check". With this specific protocol sequence the uniqueness of the MAC ID of a device is secured. All DeviceNet modules are required to participate in this MAC ID detection algorithm.

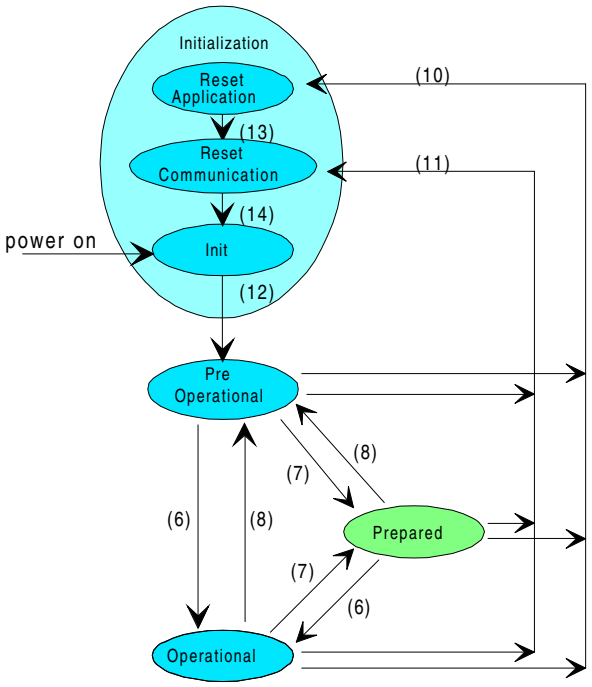


Figure 3.5-1: CANopen Node State Diagram

An optional means for the supervision of devices is provided by means of a "Heartbeat-Message" which may be broadcasted by the devices by means of the UCMM in form of an Unconnected Response Message or by Group 2 only devices by means of an Unconnected Response Message. In the data field of this message the device state is transmitted. The Heartbeat Message is triggered by the Idendity Object. A node may optionally

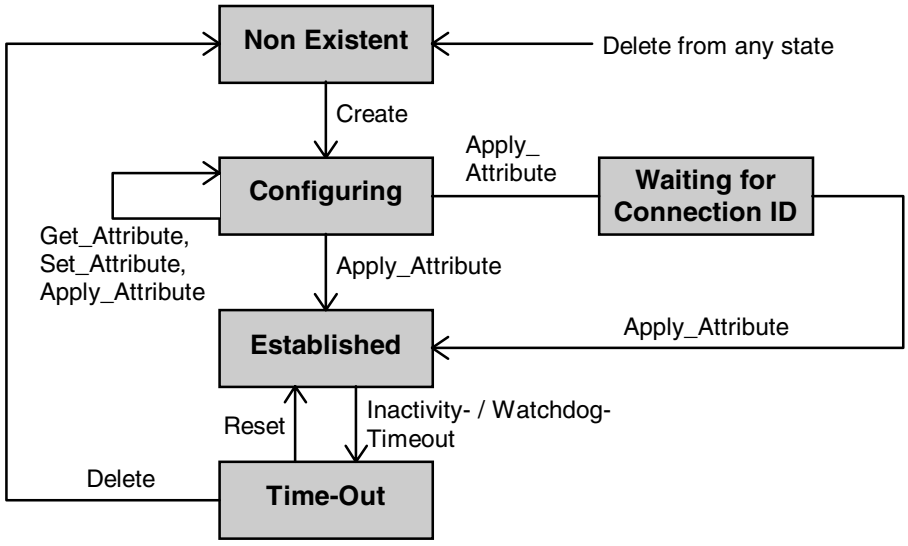


Figure 3.5-2: Device Net I/O Connection Object State Transition Diagram

broadcast fault information before going offlinge.

With **SDS** (from Version 2.2) a connection-watchdog for each output of each Embedded Object may be provided, for non-channel capable devices one watchdog per logical device. By means of a so-called „Heartbeat-Algorithm" a cyclical request (in form of a No-Operaton Action) of devices by a master device is performed. The detection of duplicate device addresses is supported by a „Enroll - Logical -Device" Action.

In any of the open system approches the coordination of the system-setup takes place under control of a master instance by scanning the network after power up of the system.

3.6 Device Modeling and Device Profiles

For open automation systems, besides of standard communication, in addition interoperability and interchangeability of alike device is demanded. Therefore open systems higher layer protocols like DeviceNet, SDS and CANopen describe the functionality of devices as seen from the network in form of a "Device Model". To promote the interchangeability of alike devices "Device Profile" of main device classes of industrial automation have to be specified which secures the same basic ("standard") behaviour of devices of different manufacturers.

Beside of a description of the functionality of the device the device model must also provide a description of the device`s identity, version number, status, diagnostic information, communication facilities and configuration parameters.

In Fig. 3.6-1 the model of a **DeviceNet** node is shown. This includes several objects, some required by DeviceNet, and others required by the product`s application function. An object provides an abstract representation of a particular component within a device and represent the related data (attributes) and procedures (services) on that data. In Table 3.6-1 the main function of the objects of a DeviceNet example node are summarised.

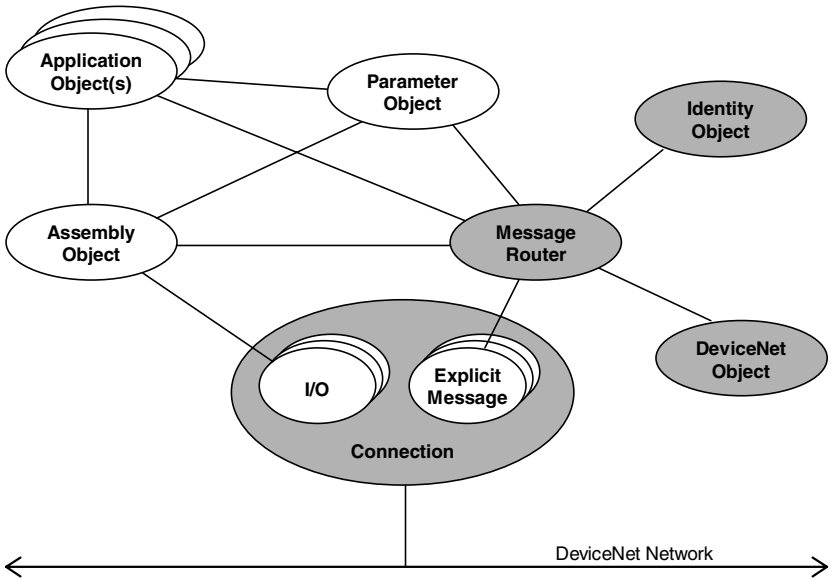


Figure 3.6-1: DeviceNet Object Model

Object addressing in DeviceNet is based on a hierarchical addressing scheme and consists of the MAC-ID (Medium Access Control Identifier), which distinguishes a node among all other nodes on the same link, the class identifier (Class ID), which identifies the object class, the instance identifier (Instance ID), which identifies an instance among all instances of the same class and the attribute identifier which identifies a attribute within a class or instance.

A DeviceNet device profile must contain the following information:

- an object model for the device type
- the I/O data format for the device type
- configuration d
- ata and the public interfaces to that data

Object	Function
Connection	Instantiates connections (I/O or Explicit Messaging)
DeviceNet	Maintains configuration and status of physical attachments to DeviceNet.
Message Router	Routes received Explicit Messages to appropriate target objects
Assembly	Groups attributes of multiple objects into a single block of data, which can be sent and received over a single connection
Parameter	Provides a standard means for device configuration and attribute access
Identity	Provides general information about the identity of a device
Application	Supplies application-specific behaviour and data

Table 3.6-1: Objects of a DeviceNet node

In **SDS** a Physical Component may contain up to 126 Logical Devices, a Logical device up to 32 Embedded Objects. SDS describes the functionality of a device in form of a hierarchical organized Component Model (Fig. 3.6-2). A SDS Component Model represents the visible structure and behaviour of a component. The topmost level of the hierarchy defines the structure and behaviour of the Physical Component and Logical Device in the SDS Model. Each SDS product must include this top level. The levels of functionality are documented by the SDS hierarchy and enumerated in the Object Type attribute. The hierarchy has a defined set of objects at the highest level of the hierarchy, which includes I/O Devices, SDS Interfaces, IEC 1131-3 defined Functions, and Function Blocks. The primary operating principle of the SDS Object hierarchy is inheritance. All models inherit the attributes, actions and events of the model from which they are derived. The Common Structure and Behaviour level, which is inherited by all SDS models, defines the minimum set of attributes, actions, and events an SDS component must possess.
A Device may contain up to 32 Embedded Objects and is the bus addressable entity (Fig. 3.6-3). According to the level of an Embedded Object, Attributes (e.g. Network Data Descriptor, Baud Rate, Object Type, ..., Software Release or Serial Number, Input Data, Output Data), Actions (e.g. Change Address, Self Test, Force State or Clear Errors) and Events (e.g. Diagnostic Error, End of timer, Change of value) are specified.

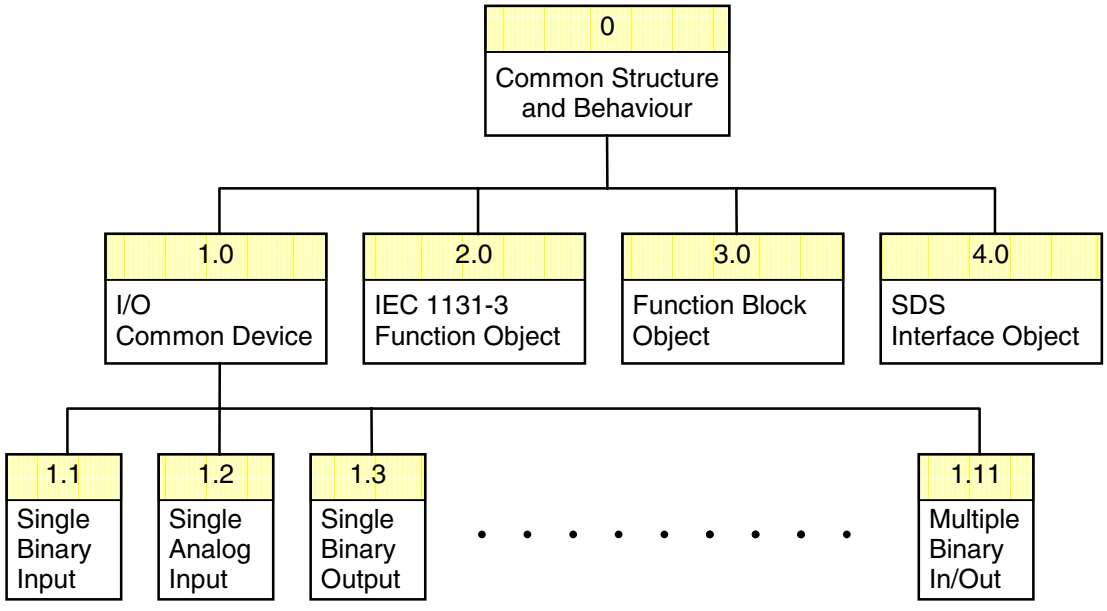
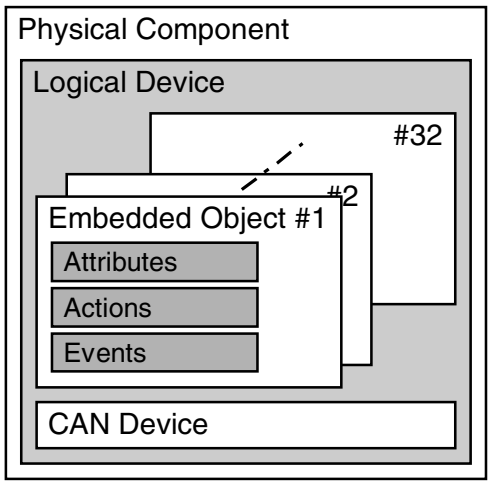


Figure 3.6-2: SDS Object Hierarchy

The **CANopen** approach is based on the description of a device`s functionality by means of an "Object Directory". Entries of the Object Directory are identified by a 16-bit index and a 8-bit subindex number with the function of an entry (data, parameter or function) implicitly specified. Beside of a section used for the definition of data types, three main sections are distinguished (Fig. 3.6-4): The Communication Profile Section, Standardised Device Profile Section and Manufacturer Specific Section.

The Communication Profile Section Information is identical for any CANopen device type and contains device related information, parameters and functions which are related for device identification, error management and the definition of the device`s communication channels including the mapping of application objects into Process Data Objects (Fig. 3.6-5). Related to DeviceNet the CANopen Communication Profile Section may be compared with the functionality of the DeviceNet, Identity, Connection and Assembly Objects.



- A Physical Component may contain 1-125 Logical Devices (Addressable Devices)
- A Logical Device may contain up to 32 Embedded Objects

Figure 3.6-3: SDS Component Model

The CANopen Device Profile Section provides the interface to the functionality of a basic ("standard") device of a specific class. Some of these entries are mandatory, some are optional. The mandatory, common entries shall ensure, that a device behaves in a defined basic manner. Different Device Profiles for main industrial devices like I/O modules [11] or drives [12] are specified to promote interchangeability of devices.

Manufacturer specific or non-standardised device functionality may be provided by means of the Manufacturer Specific Profile Section.

Index (hex)	Object
0000	Not used
0001-009F	Definition of static, complex, manufacturer-specific and device profile-specific data types
00A0-0FFF	Reserved
1000-1FFF	Communication Profile Area
2000-5FFF	Manufacturer Specific Profile Area
6000-9FFF	Standardised Device Profile Area
A000-FFFF	Reserved

Figure 3.6-4: CANopen Object Directory Structure

Index	Object Class	Object
1000h	VAR	Device Type
1001h	VAR	Error Register Device Specification Data Device Global Communication Parameters Number of supported PDOs and SDOs
1200h	Record	1st Server SDO Parameter ↓ 128th Server SDO Parameter
1280h	Record	1st Client SDO Parameter ↓ 128th Client SDO Parameter
1400h	Record	1st Receive PDO Parameter ↓ 512th Receive PDO Parameter
1600h	Record	1st Receive PDO Mapping ↓ 512th Receive PDO Mapping
1800h	Record	1st Transmit PDO Parameter ↓ 512th Transmit PDO Parameter
1A00h	Record	1st Transmit PDO Mapping ↓ 512th Transmit PDO Mapping

Figure 3.6-5: CANopen Object Directory Communication Profile Section

4 Conclusion

Due to the widespread use of the CAN protocol from automotive to any kind of industrial applications, several classes of higher layer protocols have been developed. In this paper only "open" solutions have been mentioned which may be regarded as industry standards. The focus of this paper were open system solutions for industrial automation. It was not intended to provide a guideline for choosing the best suited protocol, but primarily to provide a better functional understanding of higher layer protocols.

For heavy bus and truck and related applications the SAE J1939 standard has been widely accepted in the USA since it continues where the older standards J1708, J1587 and J1922 left off.

The OSEK/VDX specification may be considered as a promising effort for a common software platform for distributed control units in cars.

Caused by the immense distribution of CAN in the non-automotive automation several higher layer protocols and open systems approaches have been originated almost simultaneously.

With CAL, a widely accepted and approved application layer standard is available for use in any application which has to fulfil specific requirements in a fixed configuration environment. Due to the compatibility with CANopen, CANopen modules may be used within CAL systems.

With DeviceNet, CANopen and SDS three CAN-based sophisticated open systems standards are available today. On a first glance all of the three solutions provide about the same functionality if SDS Version 2.0 already is considered, although the approaches which are taken are quite different. Most significantly this concerns the usage of message identifiers which in SDS is based on a completely predefined manner, in CANopen completely left open to the system designer or integrator. DeviceNet also provides a free usage of message identifiers but with some limitations. DeviceNet and SDS are based on a connection-oriented view, CANopen is based on a message-oriented view. Each of the system uses different data transport protocols with DeviceNet providing the most variety. Quite different solutions are provided for device modelling with a very consistent object-oriented model used by DeviceNet.

But, besides of the functional features of a system solution there are further, very important aspects which determine the acceptance of a solution by the market. Main items of that kind are the support(documentation, training) available, the products (controllers, devices, interfaces, software, tools) already available and last but not least who is the originator/provider of the system solution. The last item explains why the application of DeviceNet and SDS mainly is concentrated on the US, the application of CANopen mainly is concentrated on Europe. For a European device manufacturer this implies that he is also forced to provide an interface solution for his product according to one or even two of the American Standards if he wants to participate on the worldwide market. Fortunately all of the solutions are based on the same Physical and Data Link Layer. That means that only Software is involved when applying appropriate bus interfacing.

5 References

1 ISO-IS 11898, Road vehicles - Interchange of digital information - Controller Area Network (CAN) for high speed communication, 1993

2 Bosch, CAN specification, Version 2.0, 1991, Robert Bosch GmbH

3 CAN Application Layer for Industrial Applications, CiA DS 201-207, Version 1.1, 1996

4 OSEK/VDX Operating System, Version 2.0, 1997

5 Etschberger, Modelling Distributed Application Processes with CAL,

Proceedings of 1. ICC, 1994

6 OSEK Communication Specification 1.2

7 OSEK Network Management, Version 2.0, Draft 1.1, 1997

8 CANopen, Communication Profile for Industrial Systems based on CAL

CiA Draft Standard 301, Version 3.0, 96

9 DeviceNet Specifications, Release 2.0 1997, Vol. I: Communication Model and Protocol, Vol. II: Device Profiles and Object Library

10 Micro Switch Specification: Application Layer Protocol Specification Version 2.0, 1996, SDS

Component Modelling Specification, 1995

11 SAE: Recommended Practice for a Serial Control and Communication Vehicle Network, J1939 Committee Draft, 1996

12 CiA Draft Standard Proposal DSP 302, Framework for Programmable Devices, 1997

13 CiA Draft Standard Proposal DSP 401, Version 1.4, Device Profile for I/O Modules, 1996

14 CiA Draft Standard Proposal DSP 402, Version 1.0, Device Profiles Drives and Motion Control, 1997