Thilo **SCHUMANN**
Holger **ZELTWANGER**

STANDARDIZED EMBEDDED NETWORK BASED ON CAN

The Controller Area Network (CAN) serial bus system originally developed for in-vehicle networking in passenger cars, has been used in embedded machine-control systems since the beginning of the 1990s. Most of those early users developed their own application-layer protocols. However, the days of data-link layer solutions are over: CAN and Ethernet are the most successful data-link layer protocols. The next challenge is to select a higher-layer protocol.

Around the world, there are several organizations promoting Ethernet-based application layers for industrial applications. None of them has succeeded yet. However, in the CAN world, the battle has been concluded. In factory automation, DeviceNet is the clear winner in its battle with Smart Distributed Systems (SDS), and in embedded machine control, CANopen is the most widely accepted application layer.

With DeviceNet and CANopen, two standardized (EN 50325) application layers are now available, addressing different markets. DeviceNet is optimized for factory automation and CANopen is especially well-suited for embedded networks in all kinds of machine controls. This has made proprietary application layers obsolete; the need to define application-specific application layers is history, except, perhaps, for some specialized high-volume embedded systems.

Since 1992, the CAN in Automation (CiA) international users and manufacturers group has supported different industries in the development of standardized higher-layer protocols for CAN-based networks. The solution for embedded machine networking has been developed in cooperation with the European Community. The outcome is the CANopen application layer, which has been accepted as an European standard (EN 50325-4). In addition to the CANopen application layer, CiA CANopen Special Interest Groups have specified device, interface, and application profiles. The first CANopen users in U.S. include manufacturers of industrial trucks, medical devices (e.g. GE Medical Systems), and several machine builders (e.g. Bell & Howell). In Europe, CANopen is widely accepted as the higher-layer protocol for embedded machine control in any kind of machinery, including textile machines, injection-molding machines, railways, truck-based superstructures, and even in professional coffee machines. CiA is working very closely with other user organization in standardizing device profiles, e.g., the Europmap injection-molding machine manufacturers group, the VAK German association of superstructure manufacturers, and VDA German association of public transportation. In addition, the first CANopen practice recommendations has been published by the U.S.-based Industrial Truck Association (ITA), and the U.S.-based EPRI user organization has developed the CANopen device profile for battery devices in cooperation with CiA.

I believe there are enough standardized higher-layer protocols for CAN-based networks available to suit any application. For embedded networks using established components (with annual volumes ranging from several hundred to tens of thousands), CANopen seems to be the best solution, in particular, if you want to buy off-the-shelf devices, tools, and protocol stacks.

For system designers, it is very important to reuse application software. This requires not only communication compatibility, but also interoperability and interchangeability of

Thilo **SCHUMANN**, Holger **ZELTWANGER** – CAN in Automation GmbH, Erlangen, Niemcy

devices. Therefore, CiA supports the efforts of device manufacturers, system designers, and end users to define CANopen device, interface, and application profiles.

In order to provide company and product-independent training and education services, the users and manufacturers group has its own training staff teaching CANopen technology. In addition, CiA provides worldwide consulting services in order to help users to get their CANopen networks up and running.

To make life easier for system designers, there is a CANopen conformance-test tool available. This tool is also used to certify CANopen devices. Even if the tool provides only a static test, up to 80% of failures can be detected. From my experience, I strongly recommend using only tested devices, regardless of whether they are officially certified or have successfully passed a self-test.

Introduction into CANopen application layer and communication profile

CANopen is family of communication profiles for CAN-based networks using high-speed transceivers compliant with ISO 11898-2. The CANopen application layer and communication profile is specified in CiA DS 301 (version 4.02). In each decentralized control application there are different communication objects required. In CANopen all these communication objects are standardized and well described in the Object Dictionary. The CANopen Object Dictionary is accessible by a 16-bit index and in the case of Arrays and Records additionally by an 8-bit sub-index. This dictionary also describes all application objects of the device. Application objects may be specified by a CANopen Device Profile or by an Application Profile. In addition, a device manufacturer may define non-standardized application objects, but then this device will not be interchangeable with one of the same class from another company.

There are four classes of communication objects standardized in CANopen. Process Data Objects (PDO) are mapped to a single CAN frame using all 8 bytes of the data field to transmit application objects. Each PDO has a unique identifier and may be transmitted by only one node, but it can be received by more than one (producer/consumer communication). PDOs may be transmitted in different modes that is driven by an internal event, by an internal timer, by remote requests and by a sync message received from a specific node. The default mapping of application objects as well as the supported transmission mode is also described for each PDO in the Object Dictionary. CAN identifiers for PDOs have by default high priorities to guarantee good real-time performance. The system designer can configure an inhibit-time for each PDO. The inhibit-time forbids this object to be transmitted within a specific time.

Which application objects are transmitted within a PDO is defined in the PDO Mapping Object. It describes the sequence and length of the mapped application objects. A device that supports dynamic mapping of PDOs must support this during the Pre-Operational State. If dynamic mapping during Operational State is supported, the SDO Client is responsible for data consistency.

The second class of communication objects are Service Data Objects (SDO) transmitting configuration data, which is sometimes longer than 8 bytes. The SDO transport protocol allows one to transmit objects of any size. The first byte of the first segment contains the necessary flow control information including a toggle bit to overcome the well-known problem of double received CAN frames. The next three bytes of the first segment contain index and sub-index of the Object Dictionary entry to read or write. The last four bytes of the first segment are available for configuration data. The second and the following segments using the same CAN identifier contain the control byte and up to seven bytes of configuration data. The receiver confirms each segment, so that there is a peer-to-peer communication (client/server). In the future CANopen will allow also a fast SDO transfer confirming not only each segment but also the complete object.

The third class of communication objects belongs to network management: the Error Control Object and the NMT Object. The Error Control Object is a CAN frame with 1 byte remotely requested by the NMT master node (Node guarding) or transmitted periodically by the node itself (Heartbeat). CiA recommends using Heartbeat for new designs. The Error Control Objects contains 7 bits indicating the state of a node. The heartbeat producer and consumer time as well as the guarding time and life guarding time are referenced in the Object Dictionary and can be configured by SDOs.

The NMT Object is mapped to a single CAN frame with a data length of 2 bytes. It has the identifier 0. The first byte contains the command specifier and the second byte contains the node ID of the device that shall perform the command (in the case of node ID is 0, all nodes have to perform the command). The NMT object transmitted by the NMT master forces the nodes to transit to another state. The CANopen state machine specifies Initialization State, Pre-operational State, Operational State and Stopped State. After power on each CANopen node is in the Initialization State and transits automatically to the Pre-operational State. In this state Sync objects and node guarding is provided, also the transmission of SDOs is allowed. If the NMT master has set one or more nodes to the operational state they are allowed to transmit and receive PDOs. In the stopped state no communication is allowed except NMT objects.

The Initialization State is divided in three sub-states in order to enable a complete or partial reset of a node. In the Reset_Application sub-state the parameters of the manufacturer-specific profile area and the standardized device profile area are set to their default values. In the Reset_Communication sub-state the parameters of the communication profile area are set to their power-on values. The third sub-state is the Initialization State, which a node enters automatically after power on or after reset communication or reset application. Power-on values are the last stored parameters.

CANopen defines also three specific objects for synchronization, emergency indication, and time stamp transmission. The Sync Object is broadcast periodically by the Sync Producer. This object provides the basic network clock. The time period between Sync messages is defined by the Communication Cycle Period Object, which may be written by a configuration tool to the application devices during the boot-up process. There can be a time jitter in transmission by the Sync Producer due to some other objects with higher priority identifiers or by one frame being transmitted just before the Sync Object. The Sync Object is mapped to a single CAN frame with the identifier 128. By default, the Sync Object does not carry any data, but it can have up to 8 bytes of user-specific data.

Emergency Objects are triggered by the occurrence of a device internal fatal error situation and are transmitted from an emergency client on the concerned application device. This makes them suitable for interrupt type error alerts. An Emergency Object may be transmitted only once per ‘error event’. As long as no new errors occur on a device, no further Emergency Object must be transmitted. Zero or more Emergency consumers may receive these objects. The reaction on the emergency consumers is not specified. CANopen defines several emergency error codes to be transmitted in the Emergency Object, which is a single CAN frame with 8 data bytes.

By means of the Time Stamp Object a common time frame reference is provided to application devices. It contains a value of the type Time-of-Day. This object transmission follows the producer/consumer push model. The associated CAN frame has the identifier 256 and a data field of 6 bytes length.

Optimization of PDO Communication in CANopen Networks

CANopen has become silently without much marketing effort the most important application layer for CAN-based embedded networks in industrial and general-purpose control systems. One of the most unique features is the high flexibility and the configurability of communication objects. In particular, the optimization of process data transmission gives the system designer the capability to save bandwidth and to achieve the required response times. The pre-defined PDO communication channels and their pre-defined scheduling modes as well as the pre-defined mapping of application objects are sufficient for simple networks. However, in more sophisticated network applications PDO optimization may be required. Even simple applications may be optimized in order to achieve a lower baudrate. The lower the baudrate the better is the EMC performance. This may allow using less expensive cables, connectors and avoiding eventually shielding.

CANopen defines a default assignment of CAN identifiers for up to four Process Data Objects (PDO) to be transmitted and up to for PDOs to be received – so-called pre-defined master/slave connection set. PDOs are implemented as a single CAN data frame or CAN remote frame. One can use the entire 8-byte data field for process data (application objects) transmission. In addition, the CANopen protocol specifies that all PDOs by default are connected to the application master device, which is also the network management master (NMT Master). Direct PDO communication between NMT slaves is not possible by default. This is because only the system designer knows who should communicate to whom and has therefore to be configured. However, the pre-defined master/slave connection set allows PDO communication without configuration. Even a slave-to-slave communication is possible by two PDO channels provided by the master. Of course, this requires two CAN data frames.

If it is direct communication between slaves demanded, the system designer has to configure the CAN communication object identifiers (COB-ID) in the PDO Communication Parameter Object. This configuration is performed by means of Service Data Object (SDO) communication, which are also implemented as CAN data frames. This PDO Linking supports also broadcast/multicast PDO communication. This may cause a dramatically reduction of busload.

The standardized CANopen device profiles specify pre-defined PDO with default communication parameters. There are different scheduling modes defined. PDOs can be transmitted event-driven if a device-specific or manufacturer-specific event occurs. This is scheduling mode requiring a minimum bandwidth. There is only bus traffic when an application object has been changed. Using only event-driven PDOs it is not that easy to predict response times in case of high busload. Especially, if all PDOs try to access the bus simultaneously. Therefore the average busload should be less than 30 percent.

In order to avoid a frequent transmission of a prior PDO one can assign inhibit-times, which prohibits transmission of this PDO before the inhibit-timer is elapsed. The inhibit-time is also a communication parameter configurable by an SDO operation. The inhibit-time not only avoids bubbling idiots but also allows designing deterministic transmission of PDOs. If the highest prior PDO is not allowed to be transmitted the second highest prior PDO is in this time period the highest prior PDO and so on. However, to calculate the inhibit-times is not that easy, because error frames and retransmission and stuffbits have to be considered. In general, the inhibit-time should not be longer than the frequency of process data changes. Otherwise the value change of an application object can not be lost. Unfortunately, there is no generic tool for this calculation available.

The event causing the transmission of an asynchronous PDO also may be a local event-timer. The PDO is transmitted periodically until the specified device-internal event occurs. In this case, the PDO will be sent immediately, independent if the event-timer has elapsed. In addition, the event-timer is reset. The event-timer is also a communication attribute and is configurable via SDO. Combining event-timer and inhibit-timer allows specifying virtual transmission windows. That is a PDO transmission is only possible after the inhibit-timer has been elapsed and before the event-timer has been elapsed.

PDOs are remotely requested by means of transmitting a CAN remote frame. The device supporting the corresponding CAN data frame responds this remote frame. Due to the fact that the remote frame behavior of CAN controller is different, this transmission type should not be used for standard PDO operations. In addition, remote frames require a higher bandwidth as asynchronous PDOs. However, remotely requested PDOs may be used by temporarily connected service tools and to provide hot swapping capability in systems with asynchronous PDOs. In these applications, the added devices have no knowledge of the communication history and can request in this way all necessary data.

In motion control applications, there is a high demand on synchronous transmission of process data. In order to provide synchronous operation in an asynchronous network, CANopen defines the Sync message, which is transmitted periodically. The sync period time is a configurable communication object. Another configurable time is the synchronous window length. If this is elapsed before the synchronous PDO is transmitted this may cause an internal event indicating a synchronization failure. The reception of the Sync messages forces the device to sample the inputs and to transmit the related PDOs. The reception of a synchronous PDO will not cause any action in the device before the next Sync message is received. Now the outputs will be set simultaneously in all devices that have received synchronous process data. In order to reduce busload, PDOs with low-frequent process data may be transmitted not on each Sync message reception. This is configurable from 1 to 240. The system designer has to take care that the process data frequency is higher than the PDP transmission frequency in order to avoid process data lost.

Because synchronous Receive-PDOs are performed after the next Sync message reception the process data may be not more valid. If the system designer configures these PDOs as asynchronous, the data will be processed immediately after reception. However, this combination of transmission types does not allow synchronous output operations.

Synchronous PDO transmission avoids transmission bursts and can be regarded as worst-case. Therefore the busload may be up to 80 percent. Of course, SDO and error control messages have to be considered. In order to reduce synchronous PDO communication, the acyclic PDO transmission type was introduced. These PDOs are only transmitted if a Sync message was received and an internal event has occurred. Another transmission type is remotely requested synchronous PDO, which samples only data at reception of the Sync message. The PDO transmission must be requested by another device sending a CAN remote frame. Of course, mixing of PDOs with different transmission types is possible. For example, synchronous PDOs may be used for motion control commands and asynchronous PDOs contains the limit switch data. To optimize the PDO scheduling saves not only bandwidth, but also may increase response times of process data.

If using synchronous PDOs, one has to consider the transmission jitter of the Sync message. Even if the pre-defined Sync message identifier is very high, it may happen that a lower prior message is already on the bus when the Sync transmission request occurs. If higher accuracy is required, the high-resolution methods can be used. In this case, the time when the Sync transmission request occurs is transmitted within the pre-defined Time-stamp message. All devices can re-calculate the Sync message reception and with the global time it is possible to achieve a resolution of plus-minus one bittime.

In order to optimize PDO transmission within given response time, one can assign higher prior identifier to time-critical PDOs. The device-independent assignment of PDO priorities allows transmitting high and low PDOs by the very same device. Of course, the system designer has to verify that all PDOs will be transmitted with respect to the required response times. Even in worst-case scenarios these response times should be met.

CANopen device profiles specify the default mapping of process data into PDOs. The application objects to be mapped are described in the Mapping Parameter object. This object contains for each mapped process data the 32-bit address (24-bit index and 8-bit subindex) of the CANopen Object Dictionary and the length of the process data. One PDO may contain up to 64 1-bit objects. The PDO mapping is subject of configuration. This variable or dynamic mapping allows optimized packaging of PDOs.

The system designer will map as much process data as possible in one PDO in order to minimize busload. Regardless the length of the data field each CAN telegram has to provide the same protocol-overhead (SOF, Arbitration field, Control field, CRC, Ack field, and SOF). On the other hand, it does not make sense to map process data that is high-frequently transmitted with application objects that changes very seldom. This will cause a redundant transmission of the low-frequent data. Sometimes, it is better to support two or even more PDOs.

Each CANopen device can support „only“ 512 Transmit-PDOs and 512 Receive-PDOs. If this number is not sufficient or no more CAN identifiers are available, the Framework for Programmable CANopen Devices specifies Multiplex PDOs. These PDOs contains in the first 4 byte the 24-bit index, the 8-bit index and the node-ID of the transmitting device. Only 4 byte are available to transport process data. This overhead should only be paid if your are running out of identifiers.

The configuration of PDO communication is multi-dimensional tasks requiring deep knowledge of the application objects. Due to the multiple options CANopen is providing, this embedded network can be used in very wide range of applications such as machine control, maritime electronics, avionics, medical equipment, building automation, off-road vehicles, domestic appliances, office machines, data acquisition, and public transportation. CANopen is well suited for low and intermediate volume applications, because the use of generic device in different markets increases the sales and reduces prices. However, higher volume and lower costs is only one advantage. Some manufacturers have developed generic CANopen tools for the entire life cycle of CANopen networks. Regarding configuration tools, there are several PC-based software packages available. The user of these tools does not need to have a very deep knowledge. At the end the system integrator may only link input to output variables. The configuration tool performs the optimization of the PDO communication. In addition, the user may tune the PDO communication, but therefore is CANopen know-how necessary. An overview on available devices and tools is given in the semi-annually published CANopen Product Guide.

CANopen Device Profile for Generic I/O Modules

The CiA DS-401 device profile version specifies the communication behavior of process data objects (PDO) and defines the default PDO mapping as well. Devices compliant to this specification are partly interchangeable. Sophisticated configuration options allow adaptability to very different applications. Input/output devices may have analog and digital ports for sensors and actuators. They are available in different form factors and housings. Some of them are modular expandable. In order to achieve in minimum a partly interchangeability, member s of the CAN in Automation (CiA) international users and manufacturers group have developed the device profile for generic I/O modules. Version 2.0 of this profile is based on the CANopen application layer specification submitted for European standardization. The device profile defines not only the communication, configuration, and application objects but also the default behavior after power-on and application reset.

In case of serious communication (e.g. message lost) or device failure (e.g. short circuit at output) the I/O device switches by default into ‘pre-operational’ state and transmits an emergency message. In the state ‘pre-operational’, the PDOs are inactive. However, the device may be optionally configured that it switches to the ‘stopped’ state or remains in the current state in case of failure.

CANopen devices compliant to the CiA DS-401 version 2.0 device profile specification use default communication objects as well as default application objects. The implementation of optional functions (e.g. signal conditioning and filters) and manufacturer-specific functions are not standardized. Therefore, interchangeability is only achievable regarding communication and configuration, so that the system designer can use generic configuration tools. First products according to this device profile will be presented at Hanover Fair Industry 2000.

CANopen devices compliant to the I/O device profile indicates in the ‘Device Type’ object, which I/O functionality is supported. In the annex of the CiA DS-401 specification there are implementations hints for specific I/O modules such as joy sticks. By default, in the first pre-defined Transmit-PDO there are mapped in maximum 8 x 8 digital inputs. The first Receive-PDO contains 8 x 8 digital output values in maximum. TPDO 2 to 4 send each up to four 16-bit analog inputs and in RPDO 2 to 4 there are up to four 16-bit output values. All these pre-defined PDOs are asynchronously transmitted and received. Optionally devices may support other scheduling modes: in particular, analog inputs may be transmitted synchronously. In order to save bandwidth you may map low-frequent signals into PDOs, which are transmitted with any n-th (1 to 244) Sync message. Some devices also support the bandwidth-saving acyclic PDO transmission. This PDO is only send, if the Sync message is received and the value of the input has been changed.

By default, the optional inhibit as well as event timers are zero, and the PDO identifier values are set accordingly to the ‚Pre-defined Connection Set‘ specified in the CANopen standard. When a device is switched into the ‘operational’ state by the NMT master device, it initiates transmission of all asynchronous default PDOs. Additional PDOs (in total 512 Receive-PDO and 512 Transmit-PDO) may be implemented device-specific. These have no COB-IDs (CAN identifier) assigned and therefore they are not existing after the first power-on. Only the entries in the object dictionary are available.

The default mapping of input and output variables into PDOs is optional configurable (vriable and dynamical mapping). For modules with digital I/O functionality also bit-wise, 16-bit and 32-bit access may be configured. Besides the 16-bit default access, the I/O profile allows also 8-bit, 32-bit, floating point and manufacturer-specific access. In addition, the system designer may map any other object in PDO, if this object is mappable (object dictionary attribute).

Devices providing digital inputs transmit the default PDO if one of the mapped inputs (object ‘read input’) is switched. The system designer can enable optionally input filter and polarity change function. In addition, interrupt masks (low-to-high or high-to-low) may be enabled in order to reduce PDO traffic. In order to avoid PDO transmission in ‘Operational’ state, the object ‘interrupt enable’ may be switched off.

Devices implementing digital outputs receive in PDOs current output values, which are stored in the correspondent ‘write output’ objects. The system designer can enable optionally priority change function and block filter. Using the ‘error mode’ and ‘error value’ objects you can configure the output behavior in case of device internal failures. The output can be set to a pre-defined value or can remain the last value.

After A/D conversion the analog input value is stored in the ‘read analogue input’ object. The analog input value can be conditioned if the optional offset and scaling objects are enabled. The corresponding Transmit-PDO is also in ‘Operational’ mode disabled by default. Each PDO has to be activated by setting the ‘interrupt enable’ object. In addition, the scheduling conditions have to be configured (‘upper limit’, ‘lower limit’, ‘delta’, ‘negative delta’, and ‘positive delta’ objects). With these objects the system designer can optimize PDO communication. The scheduling conditions can be configured in Integer or Floating point formats. Each analog value can be assigned with SI physical unit and prefix.

Analog output values are received in PDOs and are stored in the corresponding ‘write analogue output’ objects. Depending on the enabling of signal conditioning objects (‘offset’ and ‘scaling’) the analog values will be converted. If device internal failures occur, the optional ‘error mode’ and ‘error value’ objects define the output behavior. The output value can be set to pre-defined value or remain the last value. Each analog value can be assigned with SI physical unit and prefix.

References

[1] CANopen Application Layer and Communication Profile (CiA Draft Standard 301, Version 4.02). CAN in Automation, Erlangen, 2002.
[2] Framework for Programmable CANopen Devices (CiA Draft Standard Proposal 302, Version 3.2). CAN in Automation, Erlangen, 2003.
[3] CANopen Product Guide. CAN in Automation, Erlangen, 2003.

STANDARYZACJA WBUDOWANYCH SIECI OPARTYCH O CAN

Streszczenie: Artykuł prezentuje sieć CAN jako rozwiązanie nie tylko dla przemysłu samochodowego. Opisany został protokół komunikacyjny CANopen wraz z usługami PDO i SDO oraz optymalizacja komunikacji w sieci CANopen. Przedstawiono również profil DS.-401 urządzeń dla modułów IO.

Recenzent: dr hab. inż. Zdzisław FILUS, prof. Politechniki Śląskiej