CANopen on general serial networks

Olaf Pfeiffer, Christian Keydel, Andrew Ayre, Embedded Systems Academy

One of the challenges in today's communication applications is that more and more different network technologies need to be interfaced to another. A machine might use an RS-485 or other general serial network and also requires access to a higher control level via networks such as CANopen. The interfacing between those networks is especially challenging if not only different communication technologies but also completely different network protocols are used. Gateways interfacing such different networks need to process data through all protocol layers – in each direction in and out of the gateway – making this a higher-end application in terms of MCU performance, memory requirements and software development.

Interfacing between networks would greatly be simplified if the same network protocol would be used on the various communication technologies as illustrated by figure 1. Instead of complex gateways, simpler bridges could be used requiring far less resources and development work.

One of the few network protocols truly open to multiple communication technologies and applications is CANopen. It was originally developed to operate on CAN, however it was always kept open enough to be used on multiple communication technologies and with any application. This paper and class examines the core features of CANopen and how they can be adopted to other communication technologies .

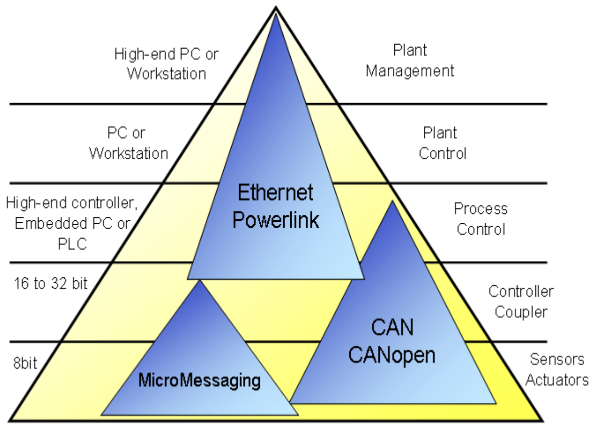

Figure 1: Communication in the automation pyramid

1. About Repeaters, Bridges and Gateways

As soon as interfaces between different network architectures are required, a variety of methods can be used to inter-connect these networks.

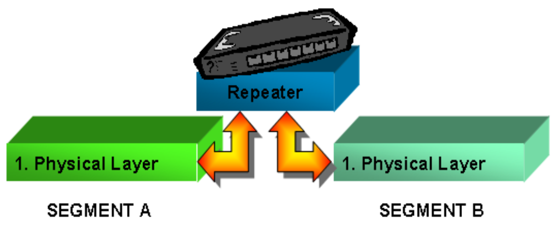

Figure 2: Repeaters act on the physical layer

1.1 Repeater

As shown in figure 2, repeaters only act on the physical layer of a network. They are independent of the protocols running on higher layers as they primarily build an interface on the signal level. Interfacing happens on the signal or bit level.

1.2. Bridge

Bridges act on the lower protocol levels as illustrated by figure 3. They receive messages entirely before forwarding them to another network segment. In order to use bridges, networks need to use the same messaging system on the lowest levels of the network protocol used. Where regular bridges simply forward all messages between network segments, so-called smart bridges have a basic understanding of some of the higher-layers in a network protocol and only forward those messages that require forwarding. At this level a smart bridge would need to be able to know which nodes are on which network segment and

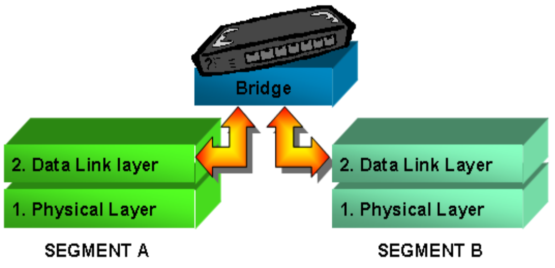

Figure 3: Bridges act on the data link layer

also know the destination for messages received so that it can make a decision to which segment this message needs to be forwarded.

1.3. Gateway

Gateways as shown in figure 4 act on the highest protocol levels and have detailed knowledge about the network protocols used. In large networks they also have more knowledge about the entire network structure, supporting optimized long-distance routing of messages. Gateways could also operate on different network protocol stacks allowing data exchange between completely different network technologies such as CAN with CANopen and Ethernet with TCP/IP.

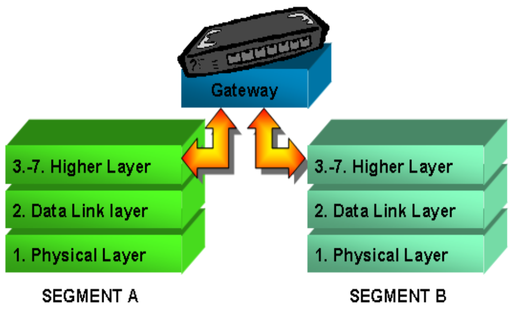

Figure 4: Gateways act on higher layers

1.4. Comparison of Complexity

In terms of hardware and software requirements, including the development and configuration the differences between repeaters, bridges and gateways are immense. A repeater can be developed without microcontroller and software. A bridge requires a medium performance microcontroller/microprocessor with software that is aware of selected network protocol features. A gateway however, requires a high-end microcontroller/microprocessor with an extensive software package. It needs to have a full understanding of ALL network protocols connected to it and requires extensive configuration information about ALL transferred data. Only with that information can a gateway make decisions about which data is forwarded how to where.

2. Drastically reduce Complexity

In applications where multiple communication technologies are used (for example any combination of RS-485, I2C, CAN and Ethernet), the overall time used for development, test, integration and maintenance can be reduced drastically if a common network protocol is used across all the communication technologies.

Not only does one reduce the overhead of developing, testing and maintaining a variety of network protocols. The interfaces between the network architectures do not need to be gateways anymore. In most cases a bridge or smart bridge would be sufficient to interface between the different communication technologies, if they use the same network protocol.

By using a common network protocol such as CANopen on two different communication technologies, the overall project complexity can typically be reduced by far more than 50%. The entire overhead for developing, testing and maintaining that second network protocol is removed, including the complex gateways required to handle the interfacing between the two.

3. Choosing CANopen as a Common Protocol

CANopen variations have already successfully been used on several serial busses (MicroMessaging) and on Ethernet

(Ethernet-Powerlink). Other arguments that make CANopen a first-choice candidate for a common network protocol are:

- Its openness in regards to availability of the standard (specification available for free from the CiA – CAN in Automation User's and Manufacturers Group)
- Its openness in regards to its ability to be customized for deeply embedded applications (can be optimized towards application requirements)
- Its minimal form factor, minimal implementations fit into some of the smallest 8-bit microcontrollers
- Its application independence (not designed for a specific, limited usage)
- Large number of suppliers for development tools such as configuration tools, monitors, analyzers, libraries, source codes and off-the-shelf products.

In summary, CANopen is very suitable to be used on multiple communication technologies, as long as those technologies provide some basic, common requirements.

3.1. Basic Requirement: Message Oriented

A communication network must fulfill the following basic requirement to be usable for CANopen: It must provide a message-oriented communication system that allows giving messages transmitted a message identifier. In case of very simple, byte-oriented serial communications a message system would need to be defined that uses a message identifier (preferably in the range of 8 to 16 bits) followed by a specified number of data bytes (at least a message size of 8 bytes must be supported).

In regards to bus arbitration (how is it decided which node may when access the bus and write something) a multi-master access system is preferred, but not required. In case of a multi-master system multiple nodes may write messages at the same time and collisions get resolved by the communication technology.

In case multi-master access is not available, a polling/synchronization mechanism is used where nodes are polled individually.

3.2. General CANopen Methodologies

When looking at CANopen a little closer, one realizes that many techniques used are completely independent in regards to the specific messaging or networking system used.

3.2.1. The Object Dictionary Concept

The CANopen Object Dictionary concept of identifying configuration parameters as well as process data used by an index and subindex is independent of the physical layers used and can be directly used with other data link layers.

For operation on lower-performance microcontrollers it may be considered to limit the size of Object Dictionary entries to a total of 32-bit. If all entries are of 32bit or less, segmented SDO transfers do not need to be implemented and only the simpler expedited SDO transfer is used.

3.2.2. Access to the Object Dictionary with Service Data Objects

The message contents used during SDO data transfer shall directly be adopted when using CANopen on other serial networks. This requires that single messages on these serial networks can store at least 8 data bytes.

The actual length of an SDO message may be optimized. In traditional CANopen SDO messages always contain 8 data bytes, even if some of them are unused. When using CANopen on other general serial networks, unused bytes may be omitted.

In order to provide this standardized access to the OD of a node, each node must implement at least one SDO server that uses a default set of message identifiers for the SDO request sent to it and the SDO response it uses to reply to such requests. The specific message assignment is part of the pre-defined connection set explained further on.

3.2.3. Process Data Object Mapping

Although the SDO accesses theoretically allow access to all configuration and process data of all nodes, the SDO communication method is not suitable for real-time process data. Using a node to node communication method involving a polling mechanism makes this a very inefficient communication method needing at least one pair of messages must to get a single variable transmitted from one node to the other.

The Process Data Object (PDO) is a method of allowing multiple OD entries to be transmitted within one message. The transmission is always a broadcast, so any nodes connected to the network can consume this data. In addition there are a variety of triggering mechanism depending on the communication system used.

When using CANopen on general serial networks, the existing PDO mapping mechanism is directly used. This again requires that the data link layer used supports message sizes with up to 8 data bytes. The PDO communication parameters are interpreted slightly

3.2.4. Default Connection Set

Any CANopen like network system shall have a default connection set. This is the default assignment for the usage of the message identifiers. Although some identifiers might be re-configured and used differently after initialization, there are some that must not be changed to ensure that the basic communication channels always work. The message identifiers that may not change during operation are those used for the default SDO accesses (one message each for request and response), the NMT Master message (broadcast from the Network Management Master to all nodes) and the NMT control messages (boot-up and heartbeat messages from the nodes).

In traditional CANopen the message identifier is divided into a function code of 4 bits and the node ID field of 7 bits.

For common serial networks, different default connection sets are defined depending on the number of bits provided by the message identifier field.

Lowering the number of bits in the message identifier field to 8, 7 or 6 bits also results in smaller function code fields and in a smaller amount of node IDs supported. The following table summarizes the different pre-defined connection sets used.

Bits in Msg ID	Bits in Fct. Code	Bits in Node ID	Maximum number of Nodes per segment
11	4	7	127
8	3	5	31
7	3	4	15
6	3	3	7

The usage of a 3-bit Function Code is illustrated by the following table

Function Code	Node ID field	Function Code assignment
0	0	NMT Master message
0	> 0	Transmit Heartbeat
1	0	SYNC message
1	> 0	EMCY message
2	0	TIME message
2	> 0	Transmit PDO 1
3	0	Reserved
3	> 0	Receive PDO 1
4	0	Reserved
4	> 0	Transmit PDO 2
5	0	Reserved
5	> 0	Receive PDO 2
6	0	Reserved
6	> 0	Receive SDO
7	0	Reserved
7	> 0	Transmit SDO

3.2.5. Bus Arbitration and Message Triggering

The recently introduced, enhanced synchronization method of CANopen can easily used for an individual polling of nodes, independent of the communication technology used. This method enhanced the SYNC signal by one byte, the SYNC counter. Depending on configuration, this signal can now also be used to poll individual nodes.

Such a system assumes that there is one node responsible for assigning the bus to a node ensuring that only one node at the time has the permission to speak. In general two methods can be implemented:

- Single SYNC producer: a polling mechanism where one node is in charge of cyclically polling all nodes connected
- Multiple SYNC producer: A token passing mechanism where one node is in charge of starting and monitoring a token that is passed on from node to node

Both mechanisms allow the reduction of the amount of messages produced by each node when being polled (lower priority messages might need to wait until the next cycle) making this communication method very deterministic and suitable for real-time applications.

It should be noted that PDO data transmission can still be combined with the triggering methods event driven (COS: change-of-state) or time driven. Meaning that only if the data changed and/or a timer expired does data actually get transmitted in the next poll cycle.

3.2.6. Single SYNC producer

In this communication model a single node is responsible for polling all nodes connected. It does this by producing SYNC messages with a one-byte counter used as a node identifier. Nodes receiving a SYNC message with its own node ID in the counter now have an assigned time window to make their transmissions. In case the window is not big enough for all transmissions queued, only the higher priority transmissions (those with the lowest message identifier) are made, others will have to wait until the next cycle. In this communication mode defining the length of the window is a crucial parameter. If it is too long, bandwidth is wasted. If it is too short, even important data messages might get delayed.

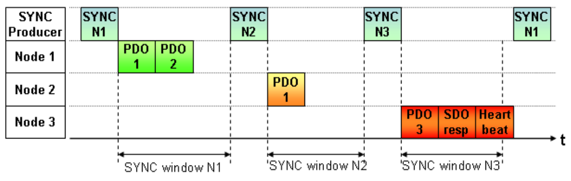

Figure 5: One SYNC producer polls all CANopen nodes

3.2.7. Multiple SYNC producer

In this optimized communication model each node automatically produces the SYNC signal for the next node upon completion of its transmissions. This method makes better use of the available total bandwidth as individual nodes are not assigned fixed time windows for their transmissions. These windows still exists, but only as a maximum. If a node has nothing to transmit, the time window is automatically shortened and the next node gets polled. In this communication mode the length of the window can be kept larger as in the single SYNC producer mode as a window gets "terminated" if no further transmissions are pending.

It should be noted that one node would still be responsible for creating the initial SYNC signal and that needs to monitor if the SYNC gets lost or needs to be replaced because of gaps in the chain (maybe node 3 polls node 4, but there is no node 4 in the network).

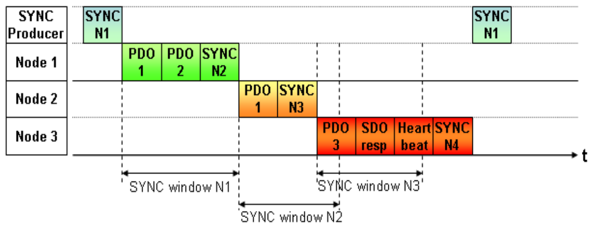

Figure 5: SYNC "token" passing between multiple CANopen nodes

3.2.8. Simplified Bridges

The method introduced in this paper also drastically reduces the complexity of bridges between multiple network segments, even if they use different data link layers. As long as the default pre-defined connection set is used a smart bridge looks at the message identifier and divides it into the function code and the node ID field. Depending on the segment a message is forwarded to, the function code gets translated using a simple lookup table and the node ID might be expanded adding an offset. Figure 7 shows a simple example how multiple networks with an 8-bit message ID can be bridged to a standard system with 11-bit message IDs.

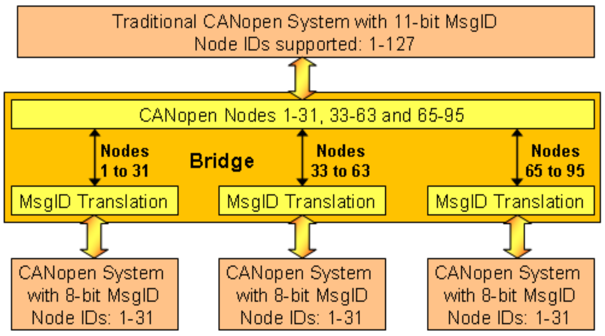
Traditional CANopen System with 11-bit MsgID
Node IDs supported: 1-127
CANopen Nodes 1-31, 33-63 and 65-95
Nodes 1 to 31
Bridge
Nodes 33 to 63
Nodes 65 to 95
MsgID Translation
MsgID Translation
MsgID Translation
CANopen System with 8-bit MsgID Node IDs: 1-31
CANopen System with 8-bit MsgID Node IDs: 1-31
CANopen System with 8-bit MsgID Node IDs: 1-31

Figure 7: A CANopen bridge

4. Summary and Outlook

Minimizing the complexity of today's embedded systems is a general goal. If multiple embedded networks are used, a common network protocol used across different communication technologies greatly simplifies the overall implementation and maintenance effort. For existing systems abandoning currently used protocols might not always be possible and sometimes customer demand might require the implementation of specific network protocols. However, for new developments and implementations a common network protocol must be seriously considered to keep development times and system complexity in check.

Olaf Pfeiffer
Embedded Systems Academy
Giesener Str. 14
31157 Sarstedt
Germany

Christian Keydel
Embedded Systems Academy
50 Aiport Parkway
SAN Jose, 95110
USA

Andrew Ayre
Embedded Systems Academy
50 Airport Parkway
San Jose, 95110
USA