Multilevel CANopen networks

Heikki Saha, Sandvik Tamrock Oy

Single-bus CANopen systems have not been enough for large systems, where CANopen allows use of existing tools and good selection of standardized components. Still CANopen documentation is single-bus oriented, but does not limit the systems into single bus level. Many things, like signal oriented communications model, support more complex systems. Typically there are more than enough maximum supported amount of nodes and COB-Ids but not enough transmission bandwidth to carry all signals of the system in the desired time windows.

This paper presents one solution, where every bus is still 100% CANopen conformant. Dividing a large system into smaller subsystems makes it easier to schedule signals into PDOs, to monitor buses by keeping subsystem specific signals local to appropriate subsystem and to decrease the coverage of fatal bus errors. Special gateways are introduced for signals, SDO- and EMCY-protocol. Also protocols and services restricted into one hierarchy level and reasons for the restriction are explained. The biggest challenges are found in EMCY-protocol and signal scheduling over multiple independent bus segments. In those issues there are also the biggest risk for application dependencies. Hopefully this paper is a trigger for official documentation and advanced development of multilevel CANopen networking.

Introduction

Thanks to the development of the CANopen standardization and increasing selection of CANopen conformant products, CANopen has become the leading CAN higher level protocol in the mobile machinery. At the same time bigger machines with more features are needed. Bigger systems must be divided into smaller subsystems, which are easier to design, diagnose and maintain. Therefore something beyond single-bus CANopen standardization will be needed to enable building of those systems. Fortunately CANopen is really open, also for this kind of future developments. Only few additional object dictionary entries with some application-level software are needed.

In practice, bus bandwidth of even 1Mbps CANopen bus will be the most limiting feature. That's why two or more hierarchy levels are absolutely mandatory to allow division of low-level control-loops and system-level information sharing. Also coverage of fatal bus errors can be reduced significantly by isolating critical signal groups from each other when possible.

Typically there are multiple instances of certain subsystems in bigger machines, which may need communication local to subsystems. It should be possible to support all structures simplifying the systems design. Possibility to controlled system boot and centralized system administration are the most important system level basic features.

Before starting the protocol by protocol analysis of CANopen, following generic terms must be defined:

Gateway node: A node connected to both upstream and downstream buses and forwarding data between them.

Gateway process: Software in the gateway node forwarding data between upstream and downstream buses.

Backbone: The topmost bus in the system containing one or more bus hierarchy levels beneath.

System master: A node acting as NMT master of the topmost CANopen bus (backbone) in the system. It may also contain several centralized system-level functions, like event logging, onboard system configurator, user interface, host communication interface, etc.

Processing platform: Processing platform is an application platform consisting of electronics HW and one or more finite state machines or software layers at top of hardware. The main purpose of software layers between applications and hardware is to provide an abstract, hardware-independent API to applications and hide the complexity of the underlying system from application programmer(s).

System boot-up and NMT

The system boot must be performed from bottom to up to ensure that the system master node automatically gets the correct boot status by using standardized CANopen boot-up routine /2/ including LSS /3/. One must notice, that in a system with multiple hierarchical buses, boot time of upstream bus must contain also the net boot time of the downstream network structure to enable error free boot of the undamaged system.

Every bus has its own NMT-master containing the description of the underlying bus structure in the objects at indexes 0x1F81, 0x1F84 - 0x1F89 /2/. That will enable full access from the backbone bus to the whole system structure with standard SDO accesses. The access is needed for both system master controller and external configuration tool.

For diagnostic purposes, it is absolutely necessary to keep buses independent, because standard NMT-commands can not address more than 127 nodes. There is object Request NMT at index 0x1F82, which can be accessed with SDO to request the appropriate NMT master send an NMT command /2/. NMT requests are typically used to turn the single bus into pre-operational state before starting the configuring and to resetting the bus after completing the configuring /2/.

Mandatory downstream bus: If mandatory slaves don’t exist or they fail during boot, the upstream CANopen stack will go to the stopped mode via operational mode, when NMT-master tries to turn the upstream CANopen stack into operational state and upstream bus boot will also fail. The mandatory gateway node is in operational state only during transmission of the EMCY telegram in case of failed boot of downstream bus.

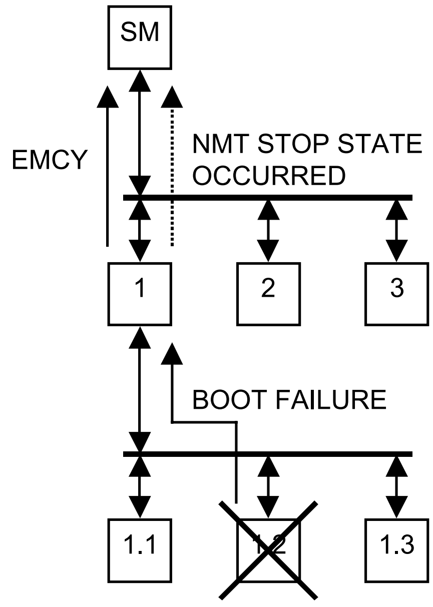

Figure 1: Boot failure of mandatory sub-bus

Optional downstream bus: After the upstream CANopen stack is turned into operational state, EMCY telegram is sent to upstream bus to inform the incomplete boot of the downstream bus.

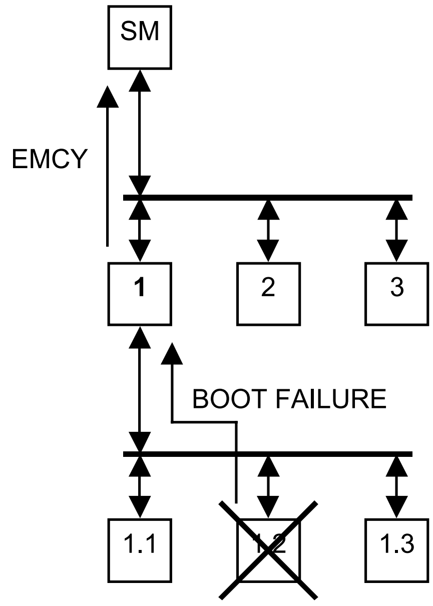

Figure 2: Boot failure of optional sub-bus

Additional object into upstream object dictionary is needed for selection of downstream bus type.

Another boot-related object needed, especially in large systems is hardware version number and HW version check for specified nodes. Currently the hardware version object at index 0x1009 is visible string type and it can not be checked during boot flow according to the DSP 302 /2/. The location for numeric format HW version could be i.e. at index 0x1018 sub-index 0x05 in the slave nodes and as array type at index 0x1F90 in the master nodes.

Heartbeat

According to the development of CANopen standardization, only heartbeat has included as a node monitoring protocol. Thanks to its multicast nature, it is extremely powerful protocol for node existence and NMT state monitoring /1/.

0	1	2	3	4	5	6	7
NMT STATE							

Figure 3: Single data byte heartbeat telegram

In a system with multiple independent networks, every bus must have its own node-monitoring domain to keep the whole system fully CANopen conformant and heartbeat transmission overhead in minimum. Depending on the application, heartbeat status of the downstream bus may affect on the NMT-state of the upstream bus in conjunction with Error behavior object at index 0x1029, /4,5,9/. The only mandatory action in case of downstream heartbeat exception is transmission of EMCY telegram into upstream bus to inform the system master node about the communication problem. This feature translating cyclical HB transmissions into event-based EMCY transmissions can be mentioned as heartbeat gateway.

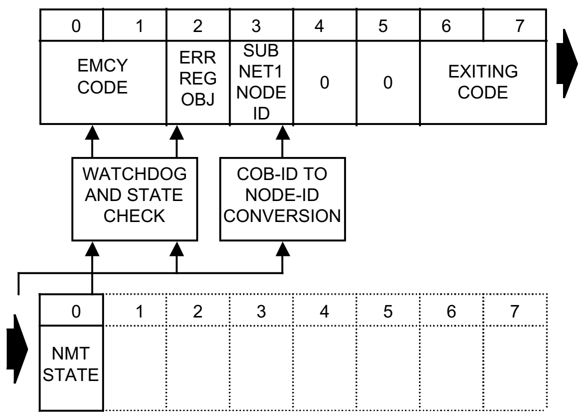

Figure 4: Heartbeat gateway operation

It is recommended that one sub-index between 0x02 and 0xFE of object 0x1029 will be standardized for configuration of sub-bus heartbeat error control.

EMCY gateway

Emergency telegrams are needed globally for event logging and monitoring purposes. Fortunately there are 5 bytes reserved for application specific use in the CANopen EMCY telegram /1/.

0	1	2	3	4	5	6	7
EMCY CODE		ERR REG OBJ	0	0	0	0	0

Figure 5: Plain EMCY telegram

The predefined fields indicate the pending error status, but some important information must absolutely be added:

EMCY code of exiting failure: If more errors are pending simultaneously and one of them exits, there must be a way to signal that transition into event log. After this 2 bytes addendum there are only 3 bytes left.

Source sub-bus information: EMCY cob-id identifies only the node-id of the topmost bus. In every EMCY gateway process, downstream node-id must be stored into one free byte of the telegram.

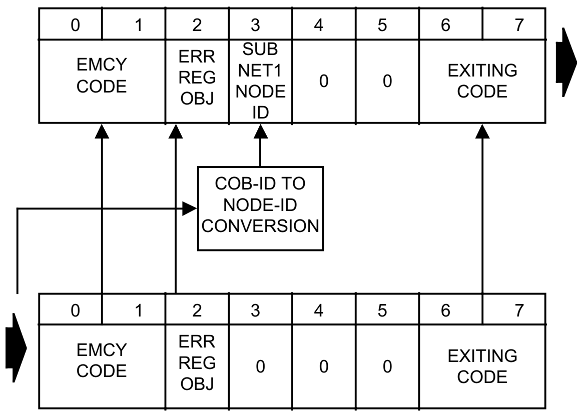

Figure 6: EMCY gateway operation

Based on the node-id information the EMCY consumers can decode the telegram producer unambiguously as long as default connection set is used. Otherwise a special conversion table between cob-ids and node-ids is needed.

If all optional bytes of the EMCY-telegram were used for producer addressing, maximum 6 hierarchical bus levels and 4195872914689 nodes total could exist.

0	1	2	3	4	5	6	7
EMCY CODE		ERR REG OBJ	SUB NET1 NODE ID	SUB NET2 NODE ID	SUB NET3 NODE ID	SUB NET4 NODE ID	SUB NET5 NODE ID

Figure 7: EMCY telegram with source information only

With mandatory and exit information there are 3 bytes left for producer information. That means maximum 4 hierarchy levels and 260144641 nodes total.

0	1	2	3	4	5	6	7
EMCY CODE		ERR REG OBJ	SUB NET1 NODE ID	SUB NET2 NODE ID	SUB NET3 NODE ID	EXITING CODE	

Figure 8: EMCY telegram with source and exiting information

In case of I/O-slave node, 1-2 more bytes could be used to identify source port and signal information to be accurate enough. That will limit hierarchy levels into 2 and total node count to 16129.

0	1	2	3	4	5	6	7
EMCY CODE		ERR REG OBJ	SUB NET1 NODE ID	SRC PORT ID	SRC SGN ID	EXITING CODE	

Figure 9: EMCY telegram with complete source information

EMCY code decoding is more painful and multiple string decoding tables and device profile indexing are needed because of the overlapping codes defined in the device profiles /4,5,6,7,8,9/. Figure 10 presents a mechanism, by which both event producer and type can unambiguously be decoded.

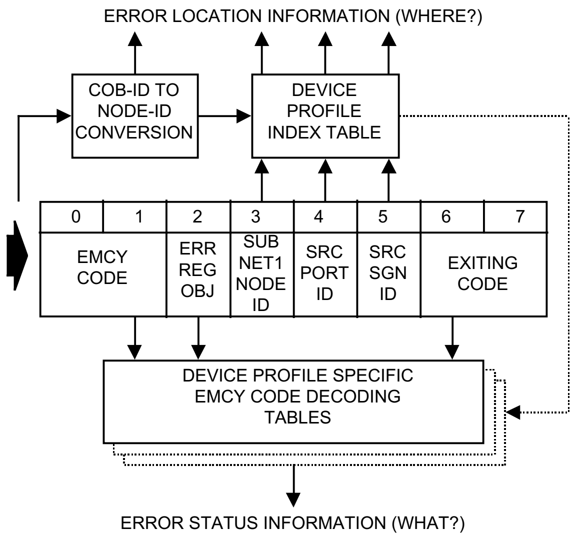
ERROR LOCATION INFORMATION (WHERE?)
COB-ID TO NODE-ID CONVERSION
DEVICE PROFILE INDEX TABLE
0
1
2
3
4
5
6
7
EMCY CODE
ERR REG OBJ
SUB NET1 NODE ID
SRC PORT ID
SRC SGN ID
EXITING CODE
DEVICE PROFILE SPECIFIC EMCY CODE DECODING TABLES
ERROR STATUS INFORMATION (WHAT?)

Figure 10: Decoding EMCY telegram

SDO gateway

In CANopen SDO protocol is used for all node configuration and other parameter accesses /1/. Those actions must cover the whole system, which is not included into SDO protocol. In multilevel CANopen networks, a special SDO gateway process is needed to forward SDO request into and reply from downstream bus.

An SDO access needs target node-id, command specifier, multiplexer and data. The data is needed only in write operations. In the SDO gateway process, those values must be written to upstream object dictionary with 3 consecutive SDO writes. Specifying command specifier and multiplexer object to trigger the downstream SDO will significantly save bandwidth during massive accesses.

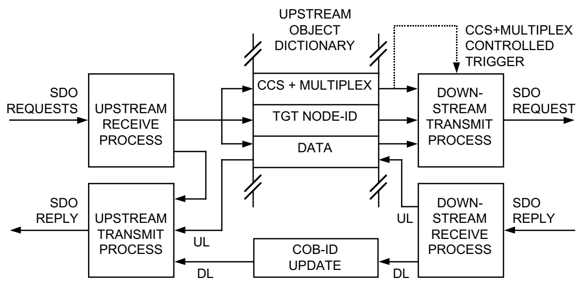
UPSTREAM OBJECT DICTIONARY
CCS+MULTIPLEX CONTROLLED TRIGGER
SDO REQUESTS
UPSTREAM RECEIVE PROCESS
CCS + MULTIPLEX
TGT NODE-ID
DATA
DOWN-STREAM TRANSMIT PROCESS
SDO REQUEST
SDO REPLY
UPSTREAM TRANSMIT PROCESS
UL
DL
COB-ID UPDATE
UL
DL
DOWN-STREAM RECEIVE PROCESS
SDO REPLY

Figure 11: SDO GW

The first downstream SDO download needs 3 upstream SDO downloads to perform a download operation.

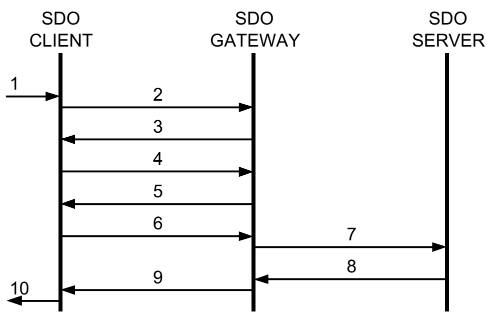

Figure 12: SDO download

1. SDO download call from the application to the SDO client service.
2. SDO download request of the downstream node-id.
3. SDO download reply of the downstream node-id
4. SDO download request of the data bytes
5. SDO download reply of the data bytes

6. SDO download request of the command specifier + multiplexer, which triggers the downstream download request.
7. Downstream SDO download request
8. Downstream SDO download reply, which triggers the upstream SDO download reply.
9. SDO download reply, which is equal to the downstream reply, except cob-id.
10. The application in the upstream SDO client gets the original reply or abort code with minimum amount of standard SDO transactions.

Next downstream SDO downloads to the same node need only 2 upstream SDO downloads for data and command specifier + multiplexer.

SDO uploads also need three upstream transactions or otherwise the abort codes may not be according to the client command specifier.

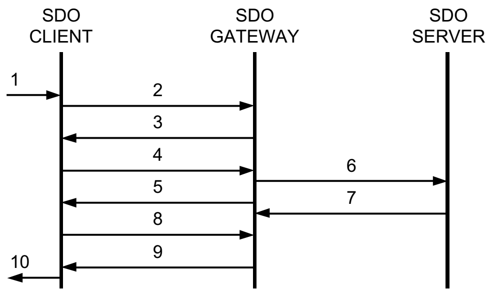

Figure 13: SDO upload

1. SDO upload query from the application to the SDO client service
2. SDO download request of the downstream node-id
3. SDO download reply of the downstream node-id
4. SDO download request of the client command specifier + multiplexer, which triggers the downstream SDO upload
5. SDO download reply of the client command specifier + multiplexer
6. Downstream SDO upload request
7. Downstream SDO upload reply
8. SDO upload request
9. SDO upload reply. Like in the SDO download, also downstream SDO upload reply will be forwarded as is, except the cob-id of the telegram.
10. Uploaded data or abort code to the application.

After first access to the same node, only one upstream SDO upload and download are needed for command specifier + multiplexer and data.

In multilevel CANopen system, presented SDO gateway process must run in all nodes acting as a gateway. The only drawbacks coming from the increased hierarchy levels are:

- Increased amount of needed SDO-transactions per one object access
- Need for 3 additional object entries into upstream object dictionary
- Additional SDO client and gateway software modules

On the other hand, following benefits are found:

- 100% CANopen conformant SDO accesses in all buses
- No limitations for number of hierarchy levels

Signal gateway

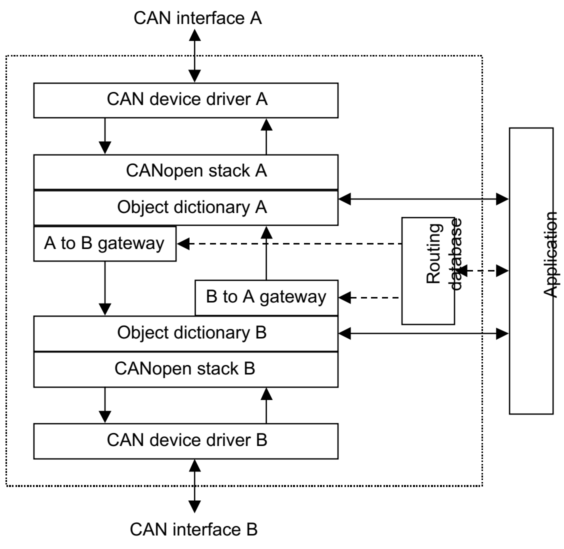

Figure 14: Signal gateway block diagram

Because of the signal-based communications model of CANopen, PDOs, MPDOs and SRDOs can not be forwarded as is. They must disassembled into signals and only wanted signals must be forwarded between object dictionaries and respective data objects transmitted according to the pending transmission types. All signals are defined by the application and therefore application must have full control over forwarding, which can also be modal at runtime. Still mapping configuration of the signals must be independent of application.

Routing database is an abstract name indicating well-organized data describing the forwarding conditions. It's main purpose is to isolate gateway engine and application from each other, because gateway engine may be executed in kernel or user-space, depending on the performance requirements.

Safety is the most important issue - forwarding is allowed only after complete start-up of the system and if application level safety conditions are fulfilled.

Net delay caused by the signal gateway process, CANopen stacks and CAN device drivers is the second most critical issue in signal forwarding. The maximum allowed forwarding delay is defined by the application and inside the application there will be different requirements for different signals. Typically, due to a tight delay budget of only few signals forces to the design of high-performance signal gateway process. In most machine control system applications the critical forwarding is unidirectional - manual drive commands from user interface to subsystems.

Reliability. In case of signal gateway, reliability mostly deals with proper scheduling of the gateway process. The gateway process must have high enough priority to be able to provide 100% forwards in specified time. On the other hand, the processing platform must be able to run all software, including operating system, fast enough and have some spare resources, just in case.

Conclusions

Single-level basic CANopen networks can easily be combined to form complex multilevel structures without breaching any CANopen standard. NMT- and boot protocols are local to every network and don't need any gateway processes. SDO-protocol requires gateway process to allow centralized access to every node in the system. EMCY-protocol gateway enables global visibility of EMCY-telegrams. SDO-gateway concept allows infinite amount of hierarchy levels, but EMCY gateway limits hierarchy levels to 2 to 6, depending on the detailed requirements. Signal gateway is the only gateway process, which may require real-time performance, depending on the delay budget of application signals.

In addition to the some software processes, multilevel support will require some new standardized object entries into NMT master nodes:

- Expected HW versions in array format for enhancement of boot-up checks
- Subnet mandatory/optional selection for subnet boot-up process
- Sub-index of Error control object for subnet heartbeat failures

And NMT slave nodes:

- Hardware version in numeric format
- 3 entries for SDO gateway

References

/1/ Application Layer and Communication profile, CiA DS 301 rev. 4.02, 13.02.2002

/2/ Framework for Programmable CANopen Devices, CiA DSP 302 rev. 3.0, 29.06.2000

/3/ Layer Setting Services and Protocol (LSS), CiA DSP 305 rev. 1.1, 09.02.2002

/4/ Device Profile for Generic I/O Modules, CiA DSP 401 rev. 2.1, 17.05.2002

/5/ Device Profile Measuring Devices and Closed-Loop Controllers, CiA DSP 404 rev.1.2, 15.05.2002

/6/ Interface and Device Profile for IEC 61131-3 Programmable Devices, CiA DSP 405 rev. 2.0, 21.05.2002

/7/ Device Profile for Encoders, CiA DSP 406 rev. 2.0, 11.05.1998

/8/ Device Profile Proportional Valves and Hydrostatic Transmissions, CiA DSP 408 rev. 1.5, 23.10.2001

/9/ Device Profile for Inclinometer, CiA DSP 410 rev. 1.0, 01.09.2000

Heikki Saha
UG Automation
Sandvik Tamrock Oy
P.O. BOX 100, 33311 TAMPERE, Finland
Tel: +358 (0)205 444 592
Fax: +358 (0)205 44 120
Email: heikki.saha@sandvik.com
http://smc.sandvik.com/