Improving development efficiency and quality of distributed IEC 61131-3 applications with CANopen system design

Heikki Saha

Traditional way of working with distributed systems has focused only to application software development of each node independently of each other. Typically signal connections are described in manually maintained documentation, which rarely fully conform CANopen. Result is potentially faulty documents, which need to be checked during build process. The main problem is a lack of system design – faulty or inconsistent signal connections, parameter access paths and values can not be detected as long as file formats are not violated. Further problems are met in system assembly and service, where faulty configuration files potentially lead to invalid system behavior. This paper presents the main methods to help managing the signal and parameter transfers in system level. Because CANopen is a system integration framework, all necessary services already exist – they just need to be used. First half of the paper describes how CANopen supports consistent signal validity monitoring and plausibility checking. Another half of the paper describes how parameter accesses and parameter attributes can be managed by CANopen mechanisms. Main result is that CANopen intrinsically supports the system design and required parameter and signal abstractions can be transferred from CANopen system designs to IEC 61131-3 application projects in a standardized way.

Introduction

Traditional development approach of distributed control systems has been application software oriented. One method to handle system level information is to declare signals into applications and then use the exported EDS-files in network design. Second approach is to use any kind of written documents or spreadsheets to manage the system integration information. Third approach is that communication details are included into application software /9/.

Main problems of the current approaches are too close dependencies between applications and system communication infrastructure – any change in system communication infrastructure introduces massive software updates. Another problem is information storage in non-standard files – automated information transfer e.g. from design to system assembly and service is impossible, which degrades the overall quality during the design process.

There are also problems with IEC 61131-3 development tools. They do not support export of well-formed EDS-files. It is obvious, because there do not exist all required information in the application software. Better way would be an ability to import interface description from CANopen system design, which contains all the necessary information for CANopen abstraction exports. But instead, tool vendors support their own, very constrained system design components to their tools.

CANopen defines well the system design process. The core of the process is presented in /2/, where the main use cases for the design files are presented, too. The process description is extended by means of EDS design and testing in /3/ and automated design tool integration issues /4/. Furthermore, interfacing CANopen in IEC 61131-3 programming environments are well defined /6/ /7/.

The overall process /10/ and process performance improvements /11/ are already presented in the literature, but this paper focuses on the detailed signal and parameter management concepts and how to automatically create signal and parameter abstractions to an example IEC

61131-3 programming environment. Signal management is presented first and then parameter management. Both sections follow the same basic structure – definition, connection management, getting information into use and using it. Local and remote parameter exports are covered in their own sections.

Signal definitions

Signals connect system components or subsystems together and are defined based on system specification. Because signals are integrating components to system, they shall be systematically managed. Unmanaged system integration interfaces lead to serious inconsistency problems.

```
[A640sub7]
ParameterName=SysPressure
ObjectType=0x7
DataType=0x0004
AccessType=rww
DefaultValue=0
LowLimit=0
HighLimit=2500
PDOMapping=1
```

Figure 1: Declaration of a signal in a consumer EDS- or DCF-file

```
[9130sub1]
ParameterName=Pressure Value 32-bit
ObjectType=0x7
DataType=0x0004
AccessType=rwr
DefaultValue=0
PDOMapping=1
Denotation=SysPressure
```

Figure 2: Declaration of a signal in a producer EDS- or DCF-file

Focus shall be in the EDS-files of the application processing platforms, where application names can be derived to the DCF-files of sensors, actuators and I/O-devices. In addition to the signal name and type, additional attributes – such as access type, minimum, maximum and default value – are needed. Signals shall always be mappable to/from PDOs. Signal definitions included in EDS- and DCF-files, which are used for system integration management. An example of consumer side signal description for PLC is presented in Figure 1. Producer side description of standard CANopen pressure transmitter is presented in Figure 2.

Denotation always overrides *ParameterName*, which enables flexible naming /2/. Most system design tools in the market just modify *ParameterName* of signal object instead of utilizing *Denotation*, which is used in the example to clarify the difference between default and application specific signal names. It is important to use the application specific names in the producer side, because producer side names are used in the communication databases.

Signal connections

Signal connections in CANopen networks are defined via PDO mapping and communication parameters /1/. Because multiple devices can consume signals produced by a single node, emphasis shall be in the producer side to avoid system inconsistency. Connection information is inserted during network design and only invalid connection definitions can be automatically detected, not wrong or missing connections. Example of valid signal connection is presented by Figure 3 and Figure 4.

```
[1800]
ParameterName=TPDO Communication Parameter
[1800sub1]
ParameterValue=0x191

[1A00]
ParameterName=TPDO Mapping Parameter
[1A00sub0]
ParameterValue=0x1
[1A00sub1]
ParameterValue=0x91300120
```

Figure 3: Simplified TPDO parameters in the producer's DCF-file

```
[1400]
ParameterName=RPDO Communication Parameter
[1400sub1]
ParameterValue=0x191

[1600]
ParameterName=RPDO Mapping Parameter
[1600sub0]
ParameterValue=0x1
[1600sub1]
ParameterValue=0xA6400720
```

Figure 4: Simplified RPDO parameters in the consumer's DCF-file

CANopen signal transfers include intrinsic message length check provided by RPDO-mapping mechanism and an optional message cycle time monitoring can also be supported /1/. Additionally, heartbeat consumer can be used for coarser validity monitoring of incoming signals.

Signal abstraction export

It is most logical to manage the signals in CANopen system design, because it is the system integration framework. Signal variable declarations and attribute definitions can be exported to the software project. Variables and attribute definitions need to be defined separately, because absolute addressing can not be used for structure members. Signals with attributes are read from the DCF-file of the PLC. An example export of the pressure signal is presented in Figure 5.

```
VAR_GLOBAL
  SysPressure AT %MD262: DINT := 0;
  dSysPressure: dtDINT := (
    ParName := 'SysPressure', Unit := '',
    DefVal := 0, MinVal := 0,
    MaxVal := 2500, Status := 0);
END_VAR
```

Figure 5: Declaration of a signal in PLC-application

Signal monitoring information can be collected from RPDO timeout monitoring – if supported – and heartbeat consumer of the signal producer. In the example in Figure 6 the producer node-ID is 17.

```
FUNCTION PdoMon : BOOL
VAR_INPUT
  Dummy: BYTE;
END_VAR
  dSysPressure.Status:=
    UpdPdoSt(dSysPressure.Status,
    NmtSts[17]);
  PdoMon := TRUE;
END_FUNCTION
```

Figure 6: Automatically generated input signal validity monitoring function supporting heartbeat consumer only

To be able to automatically retrieve system level information, there shall be a standardized method to store the system structure. Node list file is used as a project table of contents, listing the DCF-files of the system components. Example standard node list file is presented in Figure 7. Due to the tools used in the evaluation, comparable tool vendor specific format presented in Figure 8 had to be used. The information content is fortunately equal.

```
[Topology]
Node2DCFName=D002.DCF
:
Node17DCFName=D0017.DCF
```

Figure 7: Example of CANopen-conformant *nodelist.cpj* file /4/

```
[Nodes]
2=C:\Documents and Settings\...\D002.DCF
:
17=C:\Documents and Settings\...\D017.DCF
```

Figure 8: Example of proprietary *nodelist.pco* file used in the experiments

Using signals with attributes

Signal description with attributes can be exported from CANopen system design to the SW project, but where do we need such information? The main goal is to enable re-usable SW by excluding all system dependent information from the application behavior and offer it separately. Signal attributes may be used for various purposes, but an example of one common purpose is presented in Figure 9. Normal application processing takes place only if the input signal is valid and signal value within the defined range. Default signal value is used if up-to-date value is not available.

```
IF dSysPressure.Status = 0 THEN
  IF (SysPressure > dSysPressure.MaxVal) OR
     (SysPressure < dSysPressure.MinVal)
  THEN
    (* Out-of-range processing *)
    :
  ELSE
    (* Normal processing *)
    :
  END_IF;
ELSE
  (* Not-up-to-date processing *)
  SysPressure := dSysPressure.DefVal;
  :
END_IF;
```

Figure 9: An example program utilizing signal attributes in PLC

Parameter definitions

Parameters are defined locally for each node in their EDS-files. Each parameter has same set of attributes than signals, defined also in the EDS-file. Target position specific values are assigned in the system design phase and they are available only in the DCF-files. Example parameter definitions are presented in Figure 10, where several details need special attention.

If a parameter value can be modified, its access type shall be of course read-write (RW). All parameters, which can be modified, shall be systematically managed and derived from the system requirements. But, if the parameter is used for indication only, it shall be defined as

read-only (RO). More freedom can be allowed for status indication parameters. Special attention shall be put on minimum, maximum and default value attributes, because they are used also by clients. PDO-mapping attribute is used for determining parameter kind. Parameters are set to volatile by setting attribute *Mappable* to 1 and remanent by setting attribute *Mappable* to 0. The use this attribute is safe, because it does not violate CANopen and it is supported by existing CANopen system design tools.

```
[2100]
ParameterName=AppParam0
ObjectType=0x7
DataType=0x0005
AccessType=rw
DefaultValue=10
LowLimit=5
HighLimit=200
PDOMapping=0

[2200]
ParameterName=StatusOut0
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=0
LowLimit=0
HighLimit=255
PDOMapping=1
```

Figure 10: Declaration of a remanent and volatile parameter in EDS- or DCF-file

Publishing parameters to clients

Remote parameter access path definitions are not directly defined by CANopen, but additional attributes can be added for each object in the EDS- and DCF-files /2/. According to the example in Figure 11, access path is defined by attribute *ClientX*. To enable multiple client definitions, an index *X* starting from 0 is included to the attribute name. The attribute value is directly the client node-ID.

Names of published parameters are combination of *NodeName* and *ParameterName* to prevent conflicts with local names. The naming convention introduces one major challenge – how to modify the names so, that they do not contain invalid characters and too many characters. Typical constraint of IEC 61131-3 variable names is maximum of 32 significant characters. Standard attribute *Denotation* can again be used for holding a shortened name for parameter. System designer has to give the short name as part of the parameter publishing action. If *Denotation* is not defined, *ParameterName* can be used instead.

```
[2100]
ParameterName=ApplicationParameter0
ObjectType=0x7
DataType=0x0005
AccessType=rw
DefaultValue=10
LowLimit=5
HighLimit=200
PDOMapping=0
Client0=2
Denotation=AppParam0

[2200]
ParameterName=StatusOut0
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=0
LowLimit=0
HighLimit=255
PDOMapping=1
Client0=2
Denotation=StatOut0
```

Figure 11: An example of parameters published for client with node-ID 2

As defined earlier, node list file is used as a project table of contents, listing the DCF files of the system components. Remote parameters are collected from all the DCF-files, except the target device's own DCF.

Local parameter abstraction export

Local parameter variable definitions can be exported directly from the local EDS- or DCF-file.

```
VAR_GLOBAL RETAIN
  AppParam0: BYTE := 0;
END_VAR
VAR_GLOBAL
  StatusOut0: BYTE := 0;
  dAppParam0: dtBYTE := (
    ParName := 'AppParam0', Unit := '',
    DefVal := 10, MinVal := 5,
    MaxVal := 200, Status := 0);
  dStatusOut0: dtBYTE := (
    ParName := 'StatusOut0', Unit := '',
    DefVal := 0, MinVal := 0,
    MaxVal := 255, Status := 0);
END_VAR
```

Figure 12: Declaration of remanent and volatile parameters in PLC-application

Local parameter variables are defined equally with the signal variables, except there is no absolute address defined. Also attributes are defined equally, as presented in Figure 12.

Remote parameter abstraction export

Local parameter objects are handled as signals – they are variables linked into the object dictionary. According to the

example in Figure 13, remote variables are represented as arrays of structures, containing access- and other attributes and placeholder for value.
Each data type has its own array to enable efficient reading and writing in sequences. *AccTyp* attribute supports accessing in sequences, because value 0 indicates that the parameter can not be written. Application names are defined as separate constants, by which the arrays can be indexed.

```
VAR_GLOBAL
  (* Remote byte definitions *)
  ReBytes: ARRAY[0..1] OF ReByte :=
   (NetId := 0, SdoNum := 3,
    OdIdx := 16#2100, OdSub := 16#0,
    ParName := 'APP_PLC_AppParam0',
    Unit := '-', DefVal := 10,
    MinVal := 5, MaxVal := 200,
    Value := 0, AccTyp := 1 ),
   (NetId := 0, SdoNum := 3,
    OdIdx := 16#2200, OdSub := 16#0,
    ParName := 'APP_PLC_StatusOut0',
    Unit := '-', DefVal := 0,
    MinVal := 0, MaxVal := 255,
    Value := 0, AccTyp := 0 );
  (* Remote byte enumerations *)
  APP_PLC_AppParam0: BYTE := 0;
  APP_PLC_StatusOut0: BYTE := 1;
END_VAR
```

Figure 13: An example of published parameter in the client PLC application

Accessing remote parameters

IEC 61131-3 function blocks for SDO up- and downloads are standardized /6/. While there are some deviations in the CANopen libraries, minor deviations do not matter, because same information is required for the accesses. An example of parameter value read is presented in Figure 14 and of parameter write in Figure 16.
Remote parameter value reads and writes shall always be triggered by application to prevent risk to break the communication schedule by unintentional accesses. A prerequisite is of course thoroughly made scheduling of the communication. Automatic background accesses may sound attractive, but the use of such prevents the applications to ensure the safe operation during the remote parameter accesses.

```
SDOinst(
  Handle :=
    ReBytes[APP_PLC_AppParam0].SdoNum,
  Index :=
ReBytes[APP_PLC_AppParam0].OdIdx,
  SubIndex :=
    ReBytes[APP_PLC_AppParam0].OdSub,
  Length := 4,
  DataPtr :=
    ADR(ReBytes[APP_PLC_AppParam0].Value),
  Start := WORD_TO_BYTE(SDO_READ));
```

Figure 14: An example program for parameter value read in client PLC

```
SDOinst(
  Handle :=
    ReBytes[APP_PLC_AppParam0].SdoNum,
  Index :=
ReBytes[APP_PLC_AppParam0].OdIdx,
  SubIndex :=
    ReBytes[APP_PLC_AppParam0].OdSub,
  Length := 4,
  DataPtr :=
    ADR(ReBytes[APP_PLC_AppParam0].Value),
  Start := WORD_TO_BYTE(SDO_WRITE));
```

Figure 15: An example program for parameter value write in client PLC

Using remote parameters with attributes

Local parameters are just global variables with separate attribute structures. CANopen stacks of PLCs do not necessarily support range checking for parameter object values. Parameter values can be limited within the defined ranges e.g. by application according to the example in Figure 16.

```
IF (AppParam0 > dAppParam0.MaxVal) THEN
  AppParam0 := dAppParam0.MaxVal;
END_IF;
IF (AppParam0 < dAppParam0.MinVal)
  AppParam0 := dAppParam0.MinVal;
END_IF;
(* Normal processing *)
:
```

Figure 16: An example PLC program using local parameter with its attributes

```
ReBytes[APP_PLC_AppParam0.Value :=
  ReBytes[APP_PLC_AppParam0].DefVal;
```

Figure 17: An example program setting remote parameter value to its default in client PLC

Remote parameter values can be used similarly. After intentional read, parameter values can be accessed locally in the client and finally intentionally written back if required. An example how client can restore default value for a single parameter is presented in Figure 17. Figure 18 presents range-sensitive parameter value increment as another example.

```
IF ParamIncrease = 1 THEN
  IF (ReBytes[APP_PLC_AppParam0.Value] <
     ReBytes[APP_PLC_AppParam0].MaxVal)
  THEN
    ReBytes[APP_PLC_AppParam0.Value :=
      ReBytes[APP_PLC_AppParam0].Value + 1;
  ELSE
    ReBytes[APP_PLC_AppParam0.Value :=
      ReBytes[APP_PLC_AppParam0].MinVal;
```

```
  END_IF;
END_IF;
```

Figure 18: An example program modifying value of a remote parameter by utilizing its attributes in client PLC

Putting all together

NMT-master and membership monitoring are proven mechanisms, which are presented earlier in literature /10/. Even if NMT-master is implemented as a part of the application, configuration can be automatically exported from CANopen system design. Exported node names enable e.g. the use of generic membership monitoring GUI components.

Signal management presented in this paper follows CANopen and is simple but efficient. Moreover, heartbeat consumer based signal validation was implemented, for which configuration was automatically exported from system design. If PDO monitoring were supported, it can be used as a supporting signal validation. Validation mechanisms provide increased dependability and improve the overall safety without additional safety services.

Parameter management presented in this paper provides extremely efficient mechanisms for parameter management in distributed systems. Main improvement comes from significantly reduced number of errors provided by automated remote parameter attribute management.

The only missing feature is management of emergency error codes (EEC). The reason is simple – there is not such a mechanism defined for EDS- and DCF-files /2/. EEC decoding details will be included into XDD- and XCD-files /5/ and the corresponding management is presented in the future.

In addition to the improved design process performance, proposed mechanisms improve overall dependability. Table 1 lists commonly accepted failure categories /12/ and protection mechanisms provided by CANopen. Mechanisms included the work presented by this paper have positive influence on the underlined categories.

Table 1: Typical communication failure categories and protection mechanisms supported by presented CANopen cocepts

Threat	CANopen protection mechanism(s)
Repetition	Managed communication parameters, such as event- and inhibit-times
Deletion	Heartbeat producer and consumer
	Signal monitoring with RPDO timeout monitoring
	Synchronous PDO transmission monitoring
	SDO timeout monitoring
Insertion	Detection of reserved CAN-IDs of each device
	Request–reply protocols, such as SDO and LSS
	NMT state-machine
	Device state-machines
Incorrect sequence	Request–reply protocols, such as SDO and LSS
	NMT state-machine
	Device state-machines
Corruption	CAN error detection mechanisms
	Request–reply protocols
	NMT state-machine
	Device state-machines
	Signal plausibility checking
Timing error	See *repetition*, *deletion* and *insertion*
Masquerade	CAN error detection mechanisms
	DLC monitoring of protocols
Inconsistency	NMT state-machine
	Device state-machines

Concluding remarks

Main result was that CANopen significantly boosts the development of distributed systems and deviations from any related standards are not required, as commonly stated in the industry.

It was positive surprise, how small impact was caused by proprietary SDO function blocks and proprietary node list file. Main reason for good usability was that the function blocks used mainly same information in and out and node list file contained paths to the DCF-files, independently of the format.

All object attributes supported by current EDS- and DCF-files are required in the system development. Some more attributes – such as unit, scaling, signal enumeration, boolean signal groups – will be needed in the future together with EEC decoding details to improve the system design further.

Current EDS- and DCF-files will be superseded by XML-format XDD- and XDC-files in the near future. The transition is important, because previously

mentioned important features are missing from the current files.

Moreover, exporting will be more efficient in the future with standardized PLCopen XML file format /8/. That reduces the target dependency of the presented concepts in the future. However, there will be some target-PLC specific details left, e.g. publishing parameter objects and signal object access method.

References

/1/ CANopen application layer and communication profile, CiA-301, CiA
/2/ Electronic device description Part 1: Electronic Data Sheet and Device Configuration File, CiA-306-1, CiA
/3/ Electronic device description Part 2: Profile database specification, CiA-306-2, CiA
/4/ Electronic device description Part 3: Network variable handling and tool integration, CiA-306-3, CiA
/5/ Device description, XML schema definition, CiA-311, CiA
/6/ Accessing CANopen services in devices programmable in IEC 61131-3 languages, CiA-314, CiA
/7/ IEC 61131-3 programmable devices, Implementation and user guideline, CiA-809, CiA
/8/ Technical Paper PLCopen Technical Committee 6, XML Formats for IEC 61131-3, Version 2.01 – Official Release, PLCopen, 2009, 80 p.
/9/ Tisserant E., Bessard L., Trélat G., Automated CANopen PDO Mapping of IEC 61131-3 Directly Represented Variables, Proceedings of 12:th iCC, CiA, 2008, pp. 06-08..06-13
/10/ Saha H., Wikman M., Nylund P., CANopen network design and IEC 61131-3 software design, CAN-Newsletter 3/2009, CiA, 2009, pp. 52–58
/11/ Saha H., Benefits of intelligent sensors and actuators throughout the systems life cycle, The Twelfth Scandinavian International Conference on Fluid Power, May 18-20, 2011, Tampere, Finland, ISBN-978-952-15-2517-9, pp. 169–181
/12/ Alanen J., Hietikko M., Malm T., Safety of Digital Communications in Machines, VTT Industrial Systems, 2004, 95 p.

Dr. Heikki Saha
Research Engineer
Sandvik Mining and Construction Oy
Applied Research
P.O. BOX 100, FIN-33311 TAMPERE, Finland
Phone: +358 (0)400 346 537
Fax: +358 (0)205 44 121
Email: heikki.saha@sandvik.com
www.miningandconstruction.sandvik.com
