Dipl.-Inf. Michael Sperber
infoteam Software GmbH
Am Bauhof 4
D-91088 Bubenreuth

Programming CANopen from an IEC1131 point of view

Abstract: With a typical CANopen module already having a processor, RAM and ROM to serve the network, the next evolutionary step is to add user-programmability to these nodes. The now well established, vendor-independent standard IEC1131 should be the logical choice. This article will explain how a CANopen bus is seen by IEC1131, and what the implication for CANopen is, and what the incompatiblities are. Then we will have a look at how CANopen and IEC1131 could be integrated, first at the tool side, then at the controller side. The last paragraph will outline how applications could use the combination of both standards.

1 The world, seen by an IEC1131 program

Suppose you were a PLC program, written with IEC1131. How would you see an attached CAN bus?

Directly represented variables, i.e. %I and %Q, are best known for accessing local I/O points using a hierarchical structure, e.g. %I2.3 accesses bit 3 of byte 2 of local inputs. Already defined in IEC1131-3 is how to extend this syntax to non-local I/O, e.g. %IW2.5.7.1 is explained as "... the first ‘channel’ (word) of the seventh ‘module’ in the fifth ‘rack’ of the second ‘I/O bus’...“. Clearly, not all these addresses have to be „local“, and a CAN or CANopen bus is a good communication path to link such remote I/O to the controller. But, as the bus is only used to hide the difference between local and remote I/O, it is difficult to exploit bus properties.

For directly represented variables, IEC1131 allows definition of new „**location prefixes**“ (in addition to I, Q, M). So, e.g. %N might perfectly be used to access „network variables“. If data on the network can be identified by numerical-only addresses, the hierarchical address is well suited for specification of address parameters. So, %N4.5.6 might specify bit 6 of byte 5 in PDO no. 4, or %N4.3.2 could represent the second bit of the third object sent by node 4. In low-end environments, you might well live without a CANopen configuration tool at all, given some sophisticated runtime support.

Part 5 of IEC1131 defines **standard communication function blocks**. This is a way to implement network-independent communication in IEC1131. For many purposes, it is very attractive to have a network independent communication facility that may easily be ported to any networking system. But hiding network details from the application means hiding desirable network properties as well. So usually this kind of communication is less used for exchanging process data, but rather for relatively slow diagnostic, supervisory data traffic. The runtime overhead of a FB-instance call will generally not be so much a problem for this type of data transmission than it would be for the more time critical process data.

IEC1131-3 defines „**access pathes**“ to allow external facilities to read from and write to variables defined anywhere within an IEC1131-3 application. The benefit is hiding the internal structure of the application and only allowing external access to clearly specified data points. Although this is not per se limited to IEC1131-3 applications, the external application should at least be IEC1131-aware. Less-intelligent (i.e. not programmable) CANopen modules generally are not IEC1131-aware, so access pathes are not suited to exchange data with these. So again, this is less likely used for process data but rather for visualisation and controlling purposes.

The most universal way to access the outside world are external variable definitions using the **VAR_EXTERNAL** keyword. With such definitions, only the type and name of a variable are declared within the IEC1131 application. Neither the storage location nor the scope of the variable are defined within IEC1131 but rather left to the programming or operating system. Of course, external variables might as well be mapped onto directly represented variables or be accessed by access pathes as well, so they are neatly combinable with most other means.

2 ... and the match to CANopen

How does this view match to CANopen? With CANopen, there is a clear distinction between process data (PDO) and service data (SDO). To exchange process data, you would preferably use PDOs of CANopen and either directly represented variables or external variables with IEC1131. To exchange service data, you would use SDOs with CANopen and communication FBs or access pathes in IEC1131:

CANopen	**IEC1131**
PDO	directly represented variables external variables
SDO	IEC1131-5 FBs access paths

fig. 1: PDO and SDO with IEC1131

Of course, this is just a obvious best-match. With any actual implementation, there might be good reason to take a different option.

3 CANopen and IEC1131 incompatibilities

IEC1131 and CANopen have been developed independently. So it is no surprise that there are minor problems when trying to combine both:

- data types match quite well for numerical data. There is a distinction between numerical and bitstring entities in IEC1131-3 that is not found in CANopen. The more sophisticated data types (real, time, etc.) could present a problem, as these are more strictly defined in CANopen than in IEC1131.
- naming syntax for variables is more restrictive in IEC1131 than in CANopen. This is to no surprise as IEC1131 does have to define (and compile) programming languages to use these variable names, whereas the corresponding names in CANopen are purely descriptive.
- unexpectedly, there is a difference in the understanding of what „read“ and „write“ mean. CANopen knows ‘read’ and ‘write’ permissions, and IEC1131 has read-only variables (CONSTANT). But, a value written to a CANopen module should be read by the IEC1131 program on that module, to make sense.

It should be made clear that although there are incompatibilities, CANopen and IEC1131 make a almost perfect match, making not only IEC1131-3 the best choice for CANopen’ers, but also CANopen a very good choice for IEC1131-3.

4 Tool-integration

From a user’s point of view, a fully integrated monolithic tool for network configuration and IEC1131 programming would provide the highest level of friendliness. But, only few companies are big enough to develop tools of their own for one or both of these areas. So most vendors will resort to third-party software-developers providing them with tools, and almost all software companies have clearly specialiced in one of these two areas.

And, besides this know-how-problem, CANopen is very open, allowing modules from different vendors to be combined in one network, being compatible to some extent at the network interface level. IEC1131-3 provides some level of compatibility among different vendors of programmable modules, but this is only at the source level, not at the (binary) network interface level. The idea of programming a vendor-B module with vendor-A software is far from PLC-world reality. So combining intelligent modules from different vendors in one network will be a problem to program with a fully integrated tool.

Therefore, distinct tools for both purposes are a clear advantage, making use of the well-defined EDS/DCF file format and IEC1131 variable declaration syntax as an interface definition. With the negligible drawback of slightly more uncomfortable tool integration, OEMs may combine the tools of their choice, compromising on price, features, platform etc.

Most CANopen modules present a fixed (though configurable) number of process data items to the network. With programmable modules, data presented to the network may originate within the application itself, hence a fixed interface would be a severe restriction for intelligent modules. Clearly, this is not a problem of CANopen, but means to manipulate the network interface (add, modify, remove network interface items) have to be integrated into CANopen tools.

When working with two distinct tools, both tools have to agree in a consistent view in the network interface of an intelligent module. A simple but powerful solution is to allow modification of the interface only from within the CANopen configuration tool (top-down approach).

5 Controller integration

There are no plug-and-play IEC1131-3 systems available. All systems require customization to the vendors’ hard- and firmware. So the modifications necessary to integrate CANopen access into the IEC1131-runtime environment should generally not present much of a problem.

To some extent, the IEC1131 application will need to access meta-data about the network, e.g. whether CANopen is operational or pre-operational. Such system-dependent information is usually made available via system-markers.

Besides access to process and meta-data, an IEC1131 runtime environment will need some supervisory interface to a controlling system, for programming and debugging purposes. With modules having a CAN-bus often the CAN-bus is used for this purpose as well. There is a CiA-SIG „Programmable Devices“ working on a standard DS302. But this is an interface only on the top-most level, dealing with module identification and program download, but not standardising more advanced debugging features.

6 Distributed programming

When integrating a network with a PLC appliction, there are three levels of integration:
a) using the network to reduce cabling to **remote I/O**
b) moving part of the application out of the central controller to **remote logic**
c) fully **distributing the application** onto the network

The most simple type of CANopen network, remote I/O, uses the remote nodes only as data-concentrators, reducing cabling cost for remote I/O points.

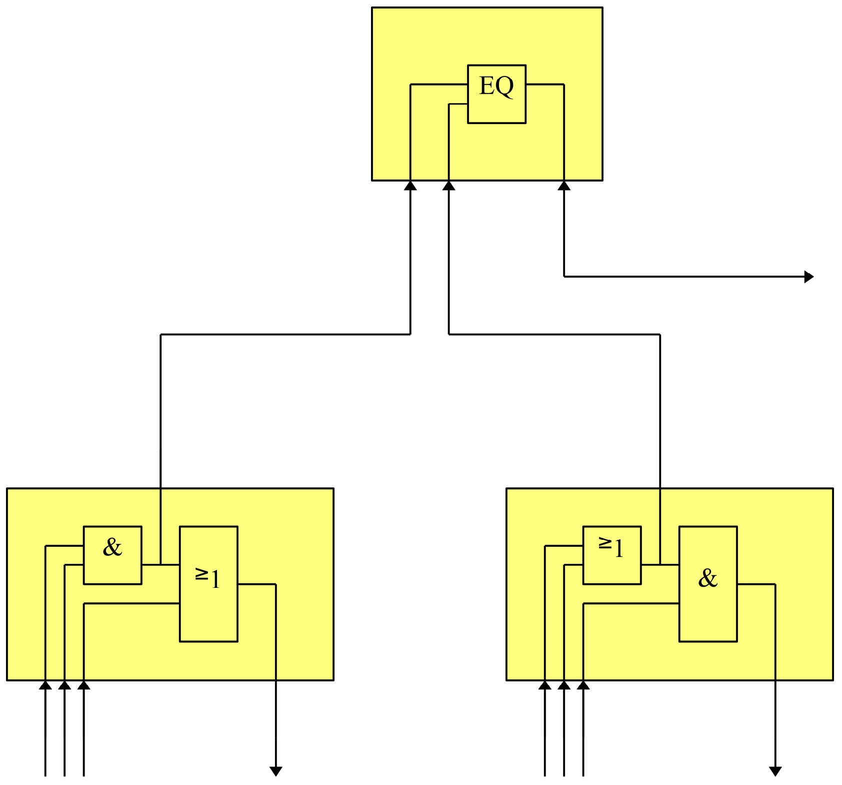

fig. 2: hierarchical logic

Remote logic is a more advanced kind of CANopen network (see fig. 2), using intelligent nodes. Relatively small parts of the application are moved to intelligent nodes. The „raw“ data available at the local I/O of these nodes in many cases does not need to be transmitted to the main PLC. Instead, the module does some local pre-calculation on this data and transmits only more abstracted information to the central controller:

- as routine tasks are partly been taken off the PLC, the PLC can be a less powerful and less expensive one
- Having only abstracted information arrive at the PLC yields a more simple application program, thus reducing engineering time to program and debug that application
- Having some pre-calculation done on the remote node may yield a simple form of fault tolerance, in that some local operations are still working while other parts of the application are down
- transmitting only digested information lowers bus load

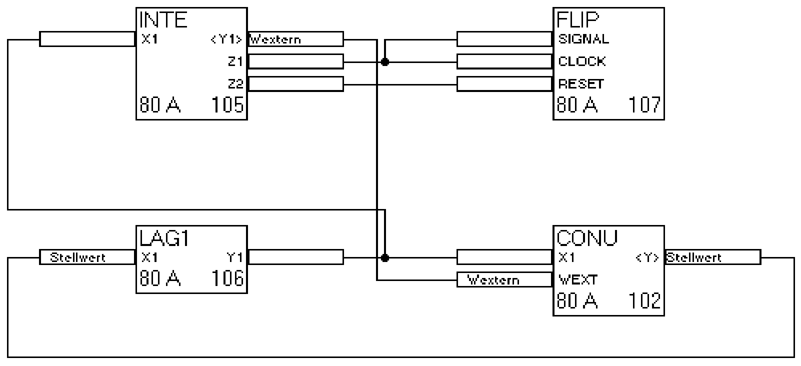

fig. 3: distributed application

Even a step further is an application distributed over multiple nodes in the network. As with remote logic, there are several small (or not so small) applications co-operating. But in contrast to remote logic, these chunks of application code are not only exchanging data, but also control information, i.e. they must synchronise in some way, get a consistent view on the status of the overall application and the peer nodes.

Unfortunately, IEC1131-3 does not have a language really suited to implement this kind of distributed application. The IEC1131-3 best fit would be Sequential Function Chart, probably with some multi-node support. But this would require one central SFC-master node with some sophisticated error handling.

More appropriate methods, languages and finally tools to solve these problems will be seen in some years, when IEC1499, in some respect a kind of IEC1131 follow-up, will hit the market. Until then, when you try to dsitribute applications like this, be assured that you are pioneering a new trend in industrial automation, and be prepared to work with tools that are not yet perfect.