Automated CanOpen PDO mapping of IEC 61131-3 Directly Represented Variables

Edouard Tisserant, Laurent Bessard and Grégory Trélat, Lolitech

CiA DS-405 defines a way to publish variables of IEC 61131-3 programmables CANOpen nodes through their Object Dictionary, using Dynamic Index Assignment defined in DS-302. Correspondence between IEC 61131-3 variables and Object Dictionary entries is let to the responsibility of the PLC manufacturer.

IEC 61131-3 defines some Directly Represented Variables, specifying direction, size, and location of physical variables. In this representation, location is an arbitrary count of integers separated by dots. Again, correspondence between location of IEC 61131-3 and physical variables is manufacturer-specified.

As a consequence, and despite of the standardization efforts of PLCopen and CiA, there is still no real interchangeability of PLC nodes in a CANopen network.

This paper proposes a method for the PLC application writer to explicitly publish and subscribe to CanOpen remote variables with Directly Represented Variables location. As a complement to DS-405, it could suppress most network reconfiguration steps when moving PLC programs from one brand to another.

Presented concepts and algorithms are already implemented in the Beremiz and CanFestival open source projects, and are publicly available.

I. Introduction

Originally designed for carrying the increasing amount of data exchanged in modern cars, the CAN (Controller Area Network) protocol is today adopted in a wide range of automated manufacturing environments. CAN only covers physical and data link layer protocol, that's why several application layers like DeviceNet or CANopen have been developed in order to standardize communication between nodes from many manufacturers. CANopen is certainly the most popular embedded communication protocol and is found in many economic sectors such as medical, industrial machinery and military. Since 1992, CANopen is maintained by the Can in Automation (CiA) organization. Thanks to CAN and other low level protocols, CANopen reduces the amount of wires used to connect Programmable Logic Controller (PLC) to the devices it controls in a machine.

During decades, PLC manufacturers have provided proprietary tools and programming languages. Due to long learning times and incompatibility between PLC brands, users where locked to one PLC manufacturer and could generally not afford to acquire skills for other tools. This situation was globally negative for the automation market, as it was hard to find skilled control engineers for one specific PLC manufacturer.

For this reason, the IEC (International Electrotechnical Commission), an international standards body, has approved a collection of standards with the intention of creating a common user experience in programming industrial controllers. One of the components of this standard, namely the IEC 61131-3 [2], defines how the user may program PLCs, thanks to a programming framework and several programming languages. Later, TC6-XML work group of PLCopen [4] association have released XML based file format specification for IEC 61131-3, subsequently letting PLC user exchange programs across PLC brands.

Thanks to IEC 61131-3 and CANopen, it is possible to build a fully functional PLC. CiA DS-405 [3] profile has been written for that purpose. Among other definitions such as types and Service Data Object (SDO) access function blocks, it defines a

way to publish variables of IEC 61131-3 programmables CANOpen nodes through their Object Dictionary. It relies on CiA DS-302 [6] for dynamic assignment of variables in Object Dictionary and CiA DS-306 [7] for configuration storage file format (DCF, nodelist.cpj).

However, CANopen PLC user's primary concern isn't about the organization of variables inside the Object Dictionary. For such devices, configuration process generally consist in mapping IEC 61131-3 Directly Represented Variables to CANopen process variables. This task is generally long and subject to critical errors, particularly while reconfiguring an existing PLC. On user point of view, organization of these variables in PLC node's Object Dictionary is an intermediate result.

Even if individual nodes configuration can be exchanged between nodes from different CANopen vendors through DCF files and if PLC programs can be exchanged with PLCopen's TC6 XML files, there is no standardized scheme to explicitly map IEC 61131-3 Directly Represented variable to CANopen process variables.

As a consequence, user willing to replace a CANopen PLC with one from another manufacturer still have to manually reconfigure all the CANopen to IEC 61131-3 variables mappings.

These reconfiguration steps could be avoided by defining an IEC 61131-3 variable location to remote CANopen variables mapping scheme. Using this scheme, TC6-XML program file along with node list file (nodelist.cpj) and associated EDS and DCF files would be sufficient to fully describe a PLC an the CANopen network it controls. This files could theoretically be exchanged between CANopen PLC solutions independently of their manufacturer.

II. IEC 61131-3 Overview

The IEC 61131 standard [2] is a general framework, that tries to establish the rules to which all PLCs should adhere to, encompassing mechanical, electrical, and logical aspects. The third part, IEC 61131-3, deals with the programming aspect of the industrial controllers, defining logical programming blocks and programming languages.

There are three variations of top level programming blocks: functions, function blocks, and programs. Functions have similar semantics to those in traditional functional languages, and directly return a single output value.

Function block types are similar to classes in object oriented languages, with the limitation of having a single public member function. Function blocks are instantiated as variables, each with their own copy of the function block state. Since a function must be idem-potent, it can neither instantiate nor call a function block instance.

Program types are very similar to function blocks, with the exception that these may only be instantiated inside a configuration, and not inside other functions, function block types or program types.

A configuration is the program organization unit with the highest level. It does not contain executable code, but instantiates programs and/or function blocks, creates and configures tasks, and assigns the programs and/or function blocks to tasks. Tasks are similar to processes in operating systems, and may execute periodically or upon the occurrence of the rising edge of a specified Boolean variable.

These blocks may be programmed with textual languages such as IL (Instruction List) and ST (Structured Text), or with graphical languages such as LD (Ladder Diagram), FBD (Function Block Diagram), or SFC(Sequential Function Chart).

SFC specify state machines, and is mostly based on Grafcet. Since a state machine implies the maintenance of state, SFCs may not be used to program functions as they must be idem-potent.

III. Map CANopen as IEC 61131-3 locations

Directly represented variables is a category of IEC 61131-3 variables that allow specification of a physical or logical location in declaration syntax.

These variables can be used directly in programs or function blocks or declared as global variables in configurations and resources.

Directly represented variables syntax states that all identities must start with the percent character ('%') followed by one or two letters and a sequence of numeric fields separated by periods ('.').

The first and required letter defines the location prefix. Three values are possible:

- 'I' for input locations,
- 'Q' for output locations
- 'M' for internal memory locations.

The second and optional letter defines the size of memory block referenced by directly represented variable. If the second letter is omitted, memory block is considered to be a single bit. There are 5 values available for this letter:

- 'X' for a bit
- 'B' for a byte (8 bits)
- 'W' for a word (16 bits)
- 'D' for a double word (32 bits)
- 'L' for a long word (64 bites)

The interpretation of the last part is left to the manufacturer implementation. However, the standard specifies that the numeric fields shall be interpreted as a hierarchical address with the leftmost field representing the highest level of the hierarchy. '%IW10.1.21' could then represent a 16 bits input memory located on rack 10, module 1, channel 21.

Following this requirement, the CANOpen hierarchy can be represented as follows:

Leftmost field, named 'Protocol ID', is fixed to a value recommended for CanOpen network, such as '503254', in reference to EN 50325-4 standard.

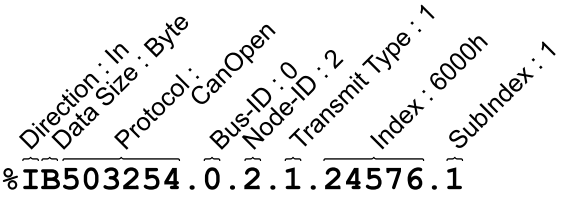
A 'Bus ID' field is required to select the bus among all CANopen buses connected to PLC.

In case of remote variable access, two fields are required to select the type of transmission through the network, and ID of remote node. Value should be respectively based on PDO Transmission Type and Node-ID values defined in DS 301.

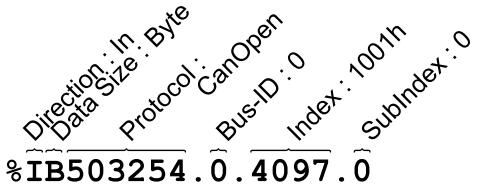
Location of the variable in the node object dictionary is represented with two fields, respectively index and subindex of the designated entry.

An optional rightmost field can be used to point individual bits, as soon as memory size letter is specified to 'X'.

As an example, the variable to access synchronously the first 8 bit read input (index 6000h, subindex 01h) of a DS-401 [5] node with ID 10h, connected on the first CANOpen bus, can be represented :



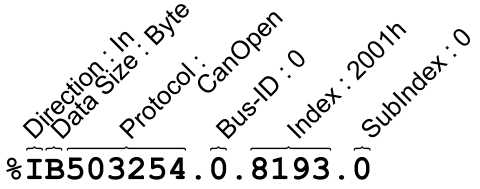
Another example, reading local CANopen Error Register (1001h,00h):



IV. PLC variables in local object dictionary

When representing local object dictionary variables, directly represented variables can also be used to implicitly declare new entries in object dictionary. In that case, with 'I' and 'Q' prefixes declares reactively Write Only and Read Only entries. 'M' prefixed variables declares Read/Write entries.

Here is an example of an arbitrary variable exported in manufacturer specific object dictionary section (2001h, 0h).



Using implicit declaration to populate object dictionary with PLC variables is useful to program NMT slave PLC nodes, that are not supposed to configure network on start-up. In that case, the EDS file included in PLC program compilation results is passed to the CANopen configuration tool.

V. Network configuration computation

When PLC node is NMT master, it must adapt the configuration of remote nodes according to PLC program needs.

This new network configuration can be deduced from the list of directly represented variables declared in PLC, the list of connected Node-ID and associated EDS files.

Hereafter is a description of a simplified algorithm used to generate network configuration out of these informations. Transmission type is ignored.

Before any computation, unused COB-IDs in the range of the standard PDO have to be deduced from EDS and Node-ID lists. COB-ID from this list will be affected to each new generated PDO, until no one left.

As described in Figure 1, directly represented variables are filtered keeping only those that are related to each CANopen network the PLC is connected to. At the same time, they are checked, keeping only those that refers to an existing node and where index and subindex are valid for this node.

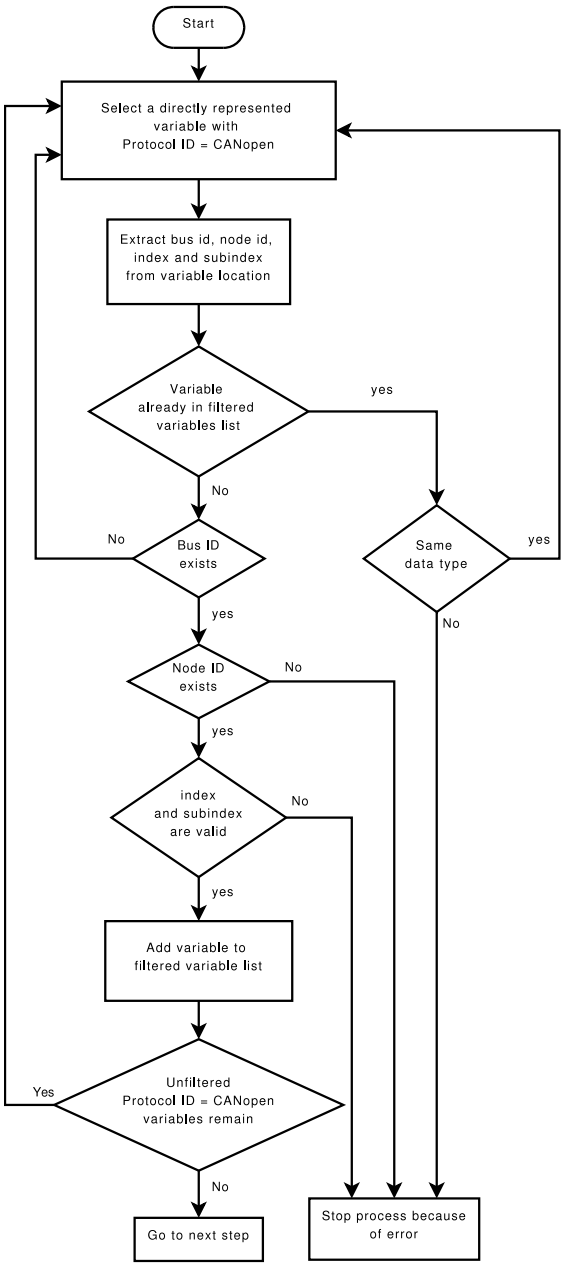


Figure 1: Variable filter diagram

Once variables filtered, we use EDS default values to distinguish those that are already mapped in some PDO and those that aren't mapped by default. Each transmitted or received default PDO that map a requested variable is kept as this, and marked as a PDO to be transmitted or received by the PLC. Each undesired object mapped in this PDO is mapped in a special 'Trash' section of local object dictionary.

Keeping as much as possible each node PDO configuration to default values is necessary in order to be able to deal with nodes that do not support PDO mapping changes. For optimisation purposes, this

feature should be optional. In that case, default PDO are systematically overwritten.

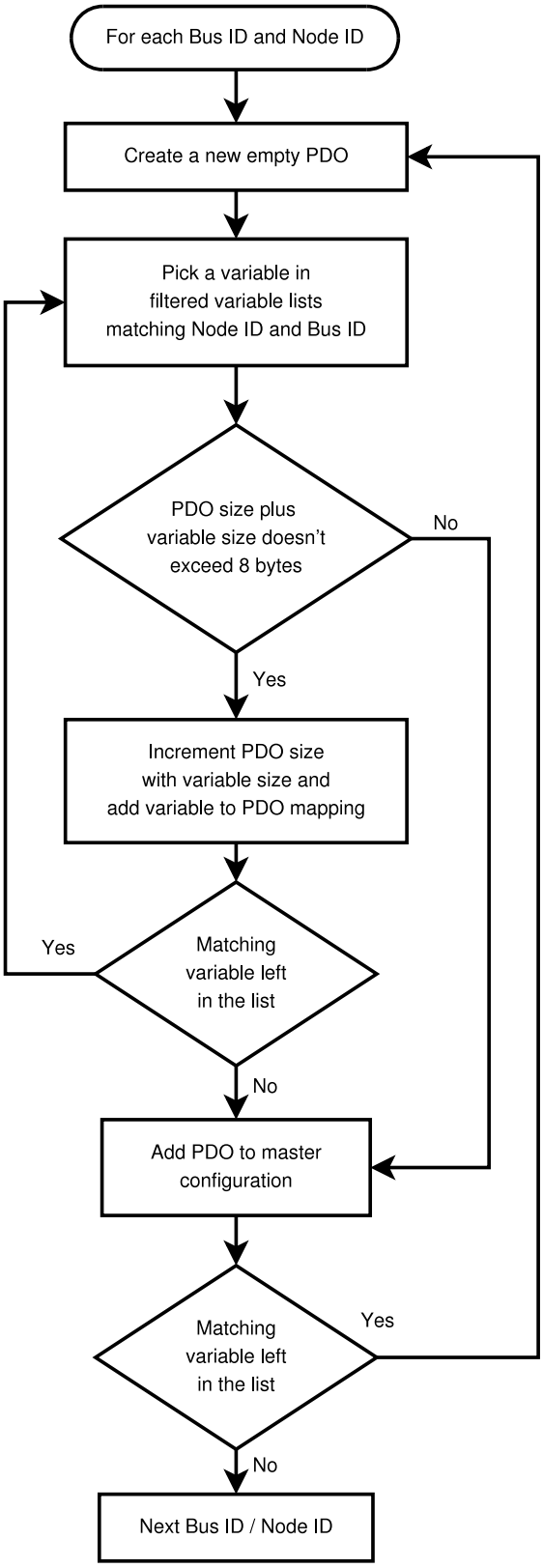


Figure 2: PDO mapping diagram

As shown in figure 2, remaining unmapped variables have to be brought together to form new PDO mappings. For each new PDO, a COB-ID is chosen among unused ones. These new PDO is then marked as a PDO to be transmitted or received by the PLC.

VI. Compatibility

As a consequence of PDO mappings, each directly represented variables aimed to access remote CANopen variables correspond to a new created variable in PLC Object Dictionary. CiA Draft Standard DS-405 should be applied to organize those variables in profile specific section.

Depending on respective PLC runtime and CANopen protocol stack implementations, directly represented variables and Object Dictionary entries may have to be periodically copied in one or the other direction or instantiated in shared memory.

Resulting network configuration may be stored in master node with respect to the DCF protocol defined in DS-302 [6]. For each configured node a DCF entry is generated and added to the network manager node object dictionary (object 1F20h or 1F22h for concise format)

VII. IEC-61131-3 workbench integration

Available IEC-61131-3 integrated development environments already provide tools to pick CANopen network variables and associate them to arbitrary directly represented variables. Integrating mapping scheme defined in this document may simply consist in replacing the incremental behaviour of variable location attribution used in most CANopen variable picker tool.

All occurrences of a variable have to be renamed when its CANopen location change. Similarly, some more advanced substitution may occur in directly represented variables location when the Node-ID of a CANopen device is changed in network description.

IEC-61131 defines that directly represented variables locations numeric

fields are expressed in decimal form instead of the hexadecimal form commonly used in CiA specifications. This can be disturbing for end users, who may not immediately see correlation between location and position in remote node Object Dictionary. For this purpose, it is preferable to let variable locations be generated by a CANopen variable selection tool, and eventually enhance programming environment so that PLC programmer could optionally edit those numerical fields in hexadecimal form. Many other ergonomic improvement can be imagined to face the lack of readability of decimal form.

VIII. Conclusion

Defining a scheme for direct representation of CANopen variables in IEC 61131-3 programs could bring real interchangeability of PLC in CANopen networks.

PLC programming workbenches that already provide PLCopen TC6-XML import and export filters, and also respect file formats described in DS-405 and DS-306 may adopt such recommendations easily.

As a DS-405 compatible complement, this scheme could eventually be used as basis for possible DS-405 enhancements.

For more technical details on possible implementation, please refer to the CanFestival [8] and Beremiz [9] open source projects, at the origin of those concepts.

Edouard TISSERANT
TBI SARL - Lolitech
24 rue Pierre Evrat
88100 Saint-Dié des Vosges
FRANCE
Phone: (+33) 3 29 52 95 67
Fax: (+33) 3 29 58 93 16
E-mail: edouard.tisserant@lolitech.fr
Website: www.lolitech.net

Laurent BESSARD
TBI SARL - Lolitech
24 rue Pierre Evrat
88100 Saint-Dié des Vosges
FRANCE
Phone: (+33) 3 29 52 95 67
Fax: (+33) 3 29 58 93 16
E-mail: laurent.bessard@lolitech.fr
Website: www.lolitech.net

Grégory TRELAT
TBI SARL - Lolitech
24 rue Pierre Evrat
88100 Saint-Dié des Vosges
FRANCE
Phone: (+33) 3 29 52 95 67
Fax: (+33) 3 29 58 93 16
E-mail: gregory.trelat@lolitech.fr
Website: www.lolitech.net

References

[1] CiA DS 301, CANopen application layer and communication profile
[2] IEC 61131-3, 2nd Ed. Programmable Controllers – Programming Languages
[3] CiA DS 405, Interface and Device Profile for IEC 61131-3 Programmable Devices
[4] PLCopen XML formats for IEC 61131-3. http://www.plcopen.org/pages/tc6_xml/
[5] CiA DS-401, Device Profile for Generic I/O Modules
[6] CiA DS-302, Framework for CANopen Managers and Programmable CANopen Devices
[7] CiA DS-306, Electronic data sheet specification for CANopen
[8] CanFestival, an OpenSource CANOpen framework. http://www.canfestival.org/
[9] Beremiz, an OpenSource framework for automation based on IEC 61131-3 and PLCopen. http://www.beremiz.org