Design and Implementation of CANopen devices

Rüdiger Härtel, Torsten Gedenk

port GmbH

Abstract

The CANopen application protocol is widely used in industrial machines. There are several companies providing source code that implements the protocol according to standards of the CiA and ISO EN 50325-4. The first steps into a new technology are very time consuming, especially when developing directly on the target hardware.
Graphical tools can support development engineers during the creation of the object dictionary and produce the source definition in a programming language for the CANopen library as well as the EDS file and an HTML documentation.
Starting implementation on a desktop operating system like Windows or Linux, instead of starting directly on the target system is efficient because there is no need for flashing or downloading the firmware and debugging is easier. Porting the application from the OS to the target hardware is done by only exchanging drivers and new compilation.
The article shows that using a tool for defining the communication behaviour and object dictionary content at a high description level leads to dramatically reduced implementation effort for CANopen devices.

Introduction

Standardized higher layer protocols (HLP) offer several advantages when implementing a communication interface, such as fully thought through access to variables, separation of communication into services, ready to use protocol stacks. On the other hand understanding the complete matter of the protocol, the implementation of the stack and usage in the first application is accompanied with long times of orientation and training.
This article focuses on the development process of a tool for design, implementation and documentation of a device that uses CANopen as a higher layer protocol. It shows how a developer can be supported by a tool throughout the process of designing and implementing devices with CANopen as the higher layer protocol. The article will also take a look at the information that has to be provided by either the tool and/or the user of it.

The requirements that must be met by such a tool are:

- read user input
- handle project files
- import/export EDS file
- support user with profile databases
- support user with hardware databases
- generate source code, EDS-file and documentation

For the specification and a later implementation there were different aspects that had to be looked upon:

- provide an intuitive graphical user interface (GUI)
- data model

- storage format of databases
- source code generation

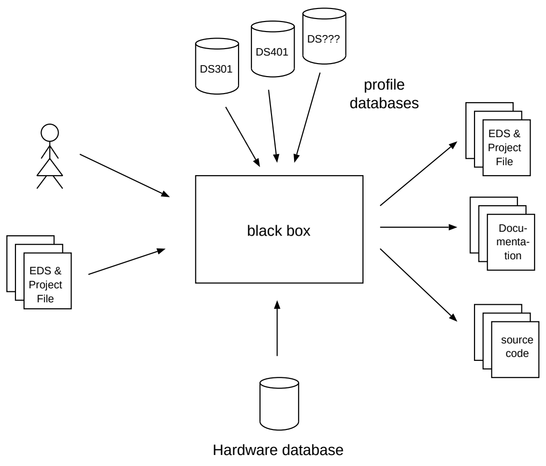


Figure 1: principle function

GUI

The graphical interface must be able to provide a good overview and organise all necessary data in a functional way. The data which is displayed consists of different parts. It is composed of:

- EDS data, documentation
- object dictionary
- activation of services
- optimisation of the library
- hardware information

CANopen uses an object oriented approach for storing communication and process variables. Access to an object is provided by means of an index and subindex. All variables are placed in the object dictionary. The structure of the object dictionary can be represented as a tree where each index is a node and each subindex is a leaf. For a better overview the indices can be grouped to their meaning.

Further nodes for hardware confi guration and EDS data can be added easily.

With the means of objects all kind of data, simple data types, records and arrays, can be represented. In addition to the plain value each object possesses

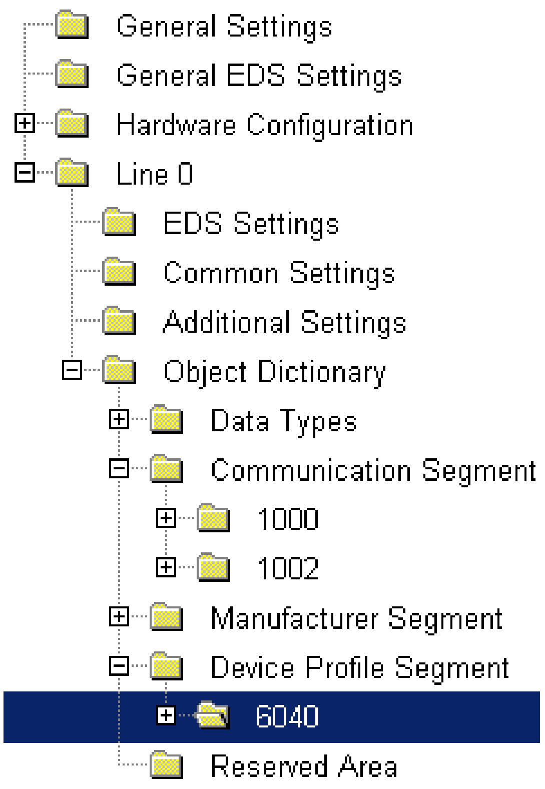


Figure 2: object dictionary as tree view

attributes like access rights, high/low limits.
This data can be displayed in an editor. On one side access to all values for the object should be given, i.e. edit fi elds for data needed to describe the EDS. This can be called structure view of an index/subindex. On the other side there should be access based on the meaning of the index. This can be called mask view. The mask view depends on the very index but is not needed for every index. The communication profi le DS301 can provide mask views for the indices:

Index	Description
1000h	Device type
1006h	Communication cycle period
1007h	Synchronous window length
100Ch	Guard time
1016h	Consumer heartbeat list
1017h	Heartbeat producer time
1028h	Emergency consumer list
1029h	Error behaviour object
1200-12FFh	SDO comm. param.
1300-1340h	SRDO comm. param.
1340-13CFh	SRDO mapping
1400-15FFh 1800-19FFh	PDO comm. param.
1600-17FFh 1A00-1CFFh	PDO mapping
1FA0-1FCFh	Object scanner list
1FD9-1FFFh	Object dispatcher list

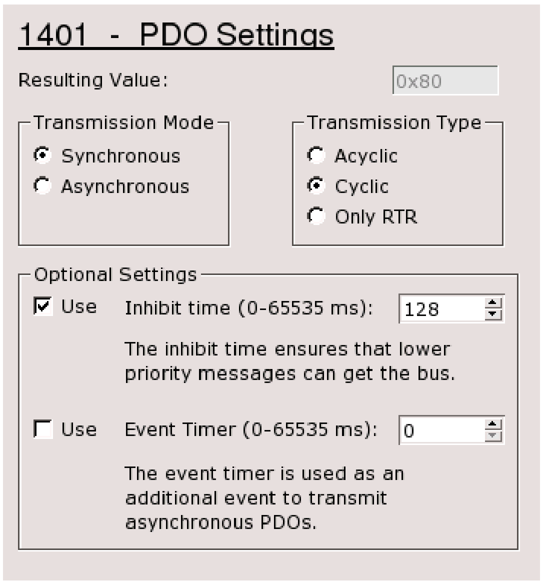
Table 1: indices with mask views

Mask views can also be provided for device profi le indices. For the future expansibility it is required that mask views are not hard coded within the tool but are provided by the databases.

As a third editor there is the need to offer optimisation for an index. The optimisation relates directly to the underlying CANopen library. Optimisation options refer to:

- the index variable and
- associated CANopen service [1]

It is possible to use an already defi ned variable of an existing application and to influence the placement of variable in memory (RAM, ROM).
Using an already defi ned variable allows to exchange only the communication layer of an already existing application that uses an other communication hardware/protocol than CAN/CANopen. This lowers implementation time and reduces


1401 - PDO Settings

Resulting Value: 0x80

Transmission Mode: [✓ Synchronous] Asynchronous

Transmission Type: Acyclic [✓ Cyclic] Only RTR

Optional Settings

[✓ Use] Inhibit time (0-65535 ms): 128

The inhibit time ensures that lower priority messages can get the bus.

Use Event Timer (0-65535 ms): 0

The event timer is used as an additional event to transmit asynchronous PDOs.

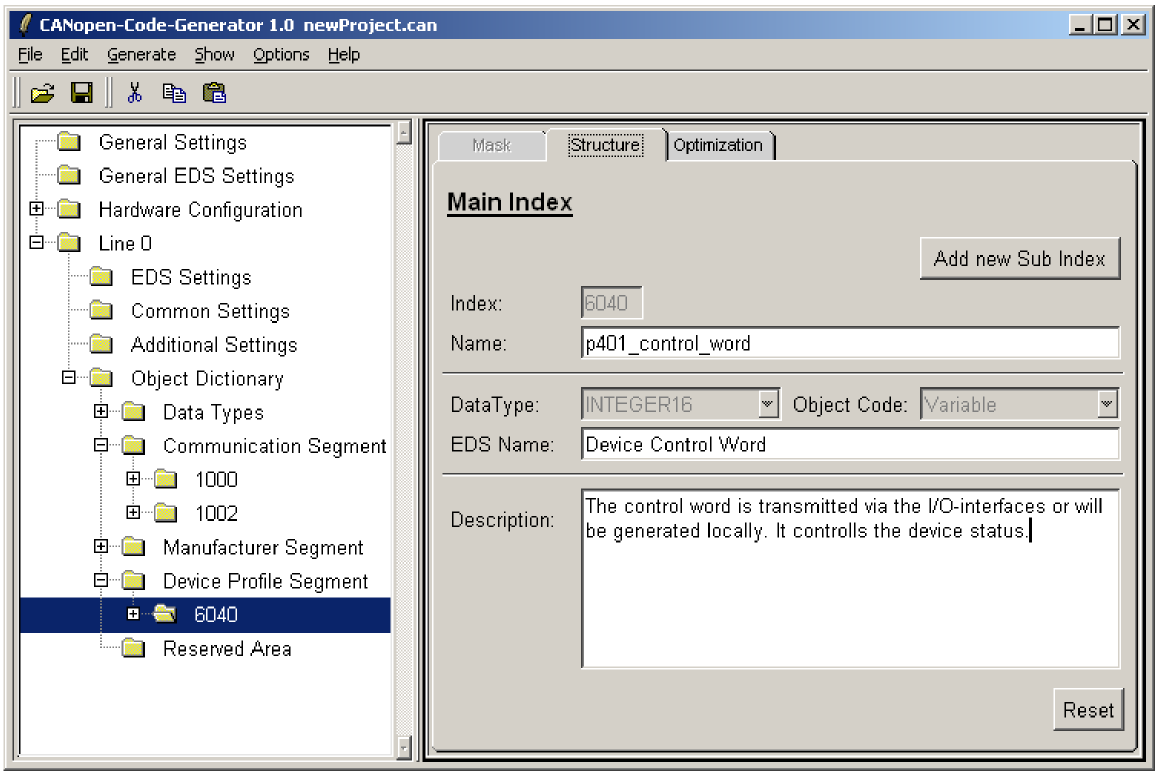
Figure 3: mask view for index 1401

costs.

Optimisation of CANopen services always affects code and RAM size of the CANopen library. The SDO service provides options for the available transfer modes (expedited, segmented, block). The PDO service has dynamic mapping and RTR optimisations available. These are two examples of optimisation. Each communication object has its own optimisation options.

An apropriate GUI element, had to be chosen in order to switch easily between the three editors. In this case tabs were chosen. They have the ability to group all elements in one view and switching between the different views. Instead of using a separated dialogue window that opens on activating an index the views are placed next to the tree.

[1] only applicable for communication parameter



Main Index

Index: 6040

Name: p401_control_word

DataType: INTEGER16 Object Code: Variable

EDS Name: Device Control Word

Description: The control word is transmitted via the I/O-interfaces or will be generated locally. It controlls the device status.

Figure 4: complete GUI layout

Data model

Databases provide the information used by the tool. They contain profile definitions as specified in the various profiles of the CiA, like DS301 and DS401, and hardware definitions that are used by the CAN driver of the CANopen library and that reflect the capabilities of the used CAN controller.

The profile databases contain the object description of every index. Additionally the database has to provide information about optimisation settings and optional mask views. Except for the mask view all data fields have the datatype VARCHAR[2]. The mask-view-data field uses the datatype BLOB[3]. In the case of interpreted programming languages the mask view can be stored as normal function definition. Table 2 lists the data that is needed in addition to the data required by the EDS.

[2] string of variable length

[3] binary large object

Name
C variable name
Object description
DefineIfExists
Opt_CreateVariable
Opt_CreateTypeDef
Opt_Memory_Specifier
Opt_SameStructure
Opt_UseDefinedVar
Opt_DefinedVarName
Mask_View

Figure 2: additional data to EDS for main index

A HLP doesn´t know anything of the underlying hardware and shouldn´t be restricted to a specific hardware. This is achieved by a well defined driver API[4]. A constant driver API makes it possible provide hardware configuration within a tool that provides configuration of a HLP.

[4] application programming interface

The hardware database unlike the profile database is subject to frequent changes due to announcement of new standalone CAN controller and new microcontroller with a CAN interface. These changes may not only affect the content of the database but also the structure and layout of the data model respectively. The data model of the hardware database has to take this into account. Changes in the hardware database have to be reflected within the input masks for the hardware settings. Therefore it was necessary that the hardware database stores the definition of the graphical user interface of the hardware settings, too. With this approach it was possible to provide access to hardware specific features that are available now and will be developed in future hardware.

The internal data model of the tool has to represent a CANopen device and must contain all information for source code and documentation generation.

In contrast to the data model of the profile database the internal structure has to be capable of handling multiple lines in order to support applications that have two or more CAN controller, i.e. CAN lines, which communicate via CANopen.

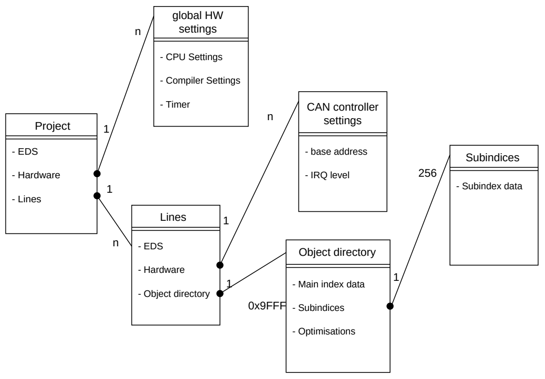


Figure 5: internal datamodell

Storage format of the databases

For the choice of the storage format the CODB-databases, EDS-files, a commercial database extension and eXtensible Markup Language (XML) were evaluated. The storage format had to match the requirements:

- extensible
- human readable/modifiable
- OS independent

The CiA defined a database (CODB) for use with its EDS checker. The storage format used is the comma separated value format where each index is written on one line and is not allowed to span over more lines. The database contains only the data needed for the purpose of checking an EDS file.

EDS files have a flat structure and are separated into sections. Besides the description of each index it contains also sections for file information, mandatory and optional indices. EDS files provide the possibility of embedding comments. Like the CODB it doesn´t allow spanning of lines.

A commercial database extension provides all required functions and even an SQL interface. From experience with other projects the installation is almost always a problem and makes support difficult. Especially when the underlying operating system changes dramatically as seen with Windows.

XML fulfils all requirements. Data is stored as text or can be encoded if needed. For processing XML data a parser is needed. Parser are available for all programming languages. Support for editing XML data is provided by many commercial and free tools.

As a result XML was chosen for the storage format for the profile and hardware databases. XML also eases the implementation of third party add-ons.

Source code generation

The tool generates all source code files required for a CANopen project. The requirement was that the generated code is ready to run directly after compilation without further modification. It makes it possible to give a first estimation of the memory size needed.

The application source code of a CANopen project consists of three files:

configuration header file
: contains the C defines for tailoring the CANopen library i.e. activating/deactivating functionality and configuring the hardware i.e. setting IRQ-level (Timer/CAN), base address of CAN controller.

object dictionary definition
: contains the structure definition in C code as specified by the CANopen library

CANopen service initialisation
: a function that does the initialisation of the different services like PDO, SDO, EMCY, TIME.

In addition to source code the tool generates the EDS-File, documentation and files for an other tool that is used for configuration of CANopen devices. This guarantees that the documentation always matches the device implementation.

Summary

A tool that abstracts from the detail implementation of a HLP and provides source code and documentation output releases developers from error prone tasks and leads to simplification and gain of time in the development process.

The source code generation simplifies the development to a great amount. When adding new objects or changing attributes of objects the developer does not have to insert/modify elements of a C-structure or C-array. All this is done with the graphical user interface.

The same applies to adding a new CANopen service. When the approriate object, e.g. 0x1282 - 2nd SDO Client, is added the service will be initialised by the generated initilisation routine.

With the integration of hardware settings into the tool switching between hardware configurations is simplified, too. This is useful when the development cannot be carried out on the target hardware but on another hardware or an operating system.

References

1. CAN in Automation, *Application Layer and Communication Profile DS301* (1 June 2000).
2. CAN in Automation, *Electronic Data Sheet Specification for CANopen DS306* (4 December 2002).
3. W3C, *Extensible Markup Language (XML) 1.0 (Second Edition)* (6 October 2000).