Rapid prototyping for CANopen system development

Heinz-Jürgen Oertel, Rüdiger Härtel, Torsten Gedenk

port GmbH

The development of simple CANopen devices up to complex systems requires exact planning and preparation. Planning and concrete realization depend each other to the extent that the selection of communication services has an effect on the performance of the assigned hardware and vice versa. The use of a prototype system enables early recognition and avoidance of bottlenecks or errors in the system. This leads to shorter development times by avoiding erroneous development and undiscovered problems in the network design.

This article shows different approaches to construction, use and feasibility of prototype systems. Attention is drawn to to the scalability of the system and reuse of code from the prototypes for the target platform. In particular the codevelopment of Master/Slave applications are taken into consideration.

1 Introduction

The development of a CANopen system consisting various of CANopen devices communicating with each other is a complex task. Frequently the developer only knows the requirements for the overall system but the requirements for every single component or the type of component (Master/Slave and their functions in the system) are not clearly defined. In many cases the number of CANopen devices or the allocation of functions to the devices are not specified at the beginning of the development.

Additionally the target hardware is not available at the beginning of the development or its development is part of the whole project. This must be considered a critical issue especially for the development of master applications. By using prototypes for the slave devices the communication connections and network traffic in the network can be estimated. Furthermore it allows for the implementation of the corresponding functions on the master and for testing them without great effort. Thereby one discovers that changes at the slaves cause changes at the master and vice versa. This influences the communication services, which again have an effect on the bus load or on the required resources of the slaves. Out of this several scenarios arise that can be developed prototypically and of course be tested.

This paper addresses the mentioned problems and shows ways to solve them. In addition to presenting different approaches a framework for Rapid Prototyping of CANopen Systems is presented and explained. The topics elaborated on include

- rapid development,
- scalability of the system,
- re-use of code and
- documentation of the components and the overall system.

2 Prototypes in general

A prototype is a first outline of a system or part of a system. Prototypes are made to examine a function or a given behavior in detail. There are two types in software development:

- Throwaway prototypes
- reusable prototypes

can be distinguished.

The throwaway prototype is mostly used to collect system requirements. The reusable prototype can be subdivided as follows:

- Explorative prototyping
- Evolutionary Prototyping
- Experimental Prototyping

The methodology of the prototypes is already used in different parts of software development and is found also in programming paradigms like Extreme Programming (XP).

This article uses the term of prototyping in the sense of the evolutionary and experimental prototyping, i.e. for the gradual implementation of system functions, detection of problems and construction of an operating prototype as proof of the desired customer functionality. The prototype already contains the basic functionalities which are developed further into the end product.

A simulation is another attempt to get new knowledge of a system to be developed utilizing complex software like Matlab/Simulink. The simulation is carried out by means of a simulation model which only implements the main characteristics of the real system. The results of the simulation are strongly dependent on the predefined input quantities. Simulation tools use simulation languages of their own, such as Tcl/Tk, Python, CAPL and others.

With the prototypes a practical attempt for setting up CANopen systems and also individual CANopen nodes exists. The main differences in prototypes for the simulation are:

- use of the CANopen library that is being used for the "real" development
- incremental integration of the prototypes into the target platform and into the CANopen network
- running system in an early stage of development
- process simulation

3 Prototypes for CANopen devices

The use of Prototyping in the software development is supported by adequate tools and a software environment. Tools allow for the addition or removal of functionality in an easy and reproducible manner. The software environment permits compiling and testing of the prototype.

For a CANopen device the CANopen object directory and the CAN connection (CAN driver) are the central elements. The graphic tool, CANopen design tool, generates the object directory by means of databases. The object directory contains the CANopen communications profile (DS301, DS302) and device profiles (DS401, DS402 ...). The result is C code, an EDS file and a detailed documentation of the objects used (Fig. 1)

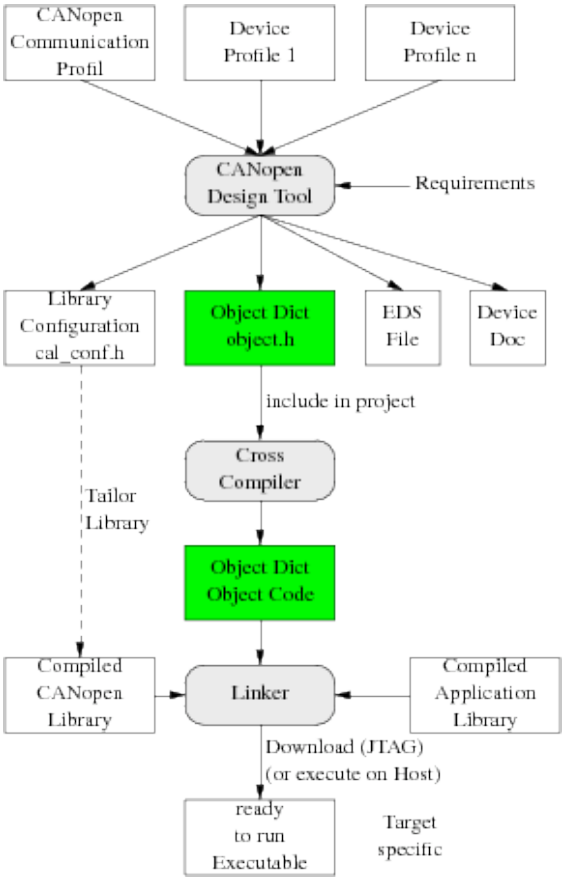

Figure 1: Principle of generating a CANopen

CANopen libraries are very extensivy scalable in their functionality by using compiler switches. The compiler switches are set depending on the objects used in the project which diminishes problems in the configuration of the CANopen library.

The software development starts before the target hardware is available. By using a hardware abstraction layer (HAL) the software can be developed on any arbitrary platform (Fig. 2).

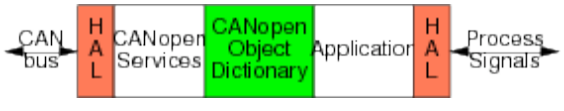

Figure 2: Hardware Abstraction Layer

he hardware abstraction layer is an application programming interface (API) which contains all the necessary functions of the target system. When changing to another platform the function contents are adapted to the circumstances of the other platform. Little effort is needed to change between platforms and this can be carried out at any time. A mixed network setup is possible with prototypes running on the target hardware and other devices of the network on the PC hardware.

4 CAN abstraction layer

The use of a standard PC permits simple creation of CANopen prototypes. The fault diagnosis can be carried out with the debugger without having to accept a limitation (number of break points) or waiting times for loading the program into flash or EEPROM. Different solutions for the hardware abstraction layer on the PC exist:

Virtual Network

A virtual network is created in the memory of the PC. The CANopen prototypes communicate via Shared Memory or other IPC mechanisms with each other.

One CAN-Interface per prototype

Prototypes access an existing CAN interface board (USB-CAN Dongle, PCI CAN card) directly. For each prototype a CAN interface is needed.

For the prototype an old PC is sufficient which is equipped with an AT or PCI CAN interface. The 1 diskettes Linux, fli4l project, equipped with the can4linux driver can be used. The CANopen software is then transmitted and started by FTP or telnet into the RAM of the PC.

5 Virtual network

The virtual network is created in the memory of the PC. TCP/IP was selected as an IPC mechanism. The TCP/IP server is a separate program and represents the CAN network. The CANopen prototypes are clients and connect to the virtual CAN network using of the TCP/IP server. As a result it becomes irrelevant which operating system is used. The prototypes can be distributed on many different PC's and started from there. If the broadcasting mechanism of the UDP protocol is utilized the server isn't needed at all.

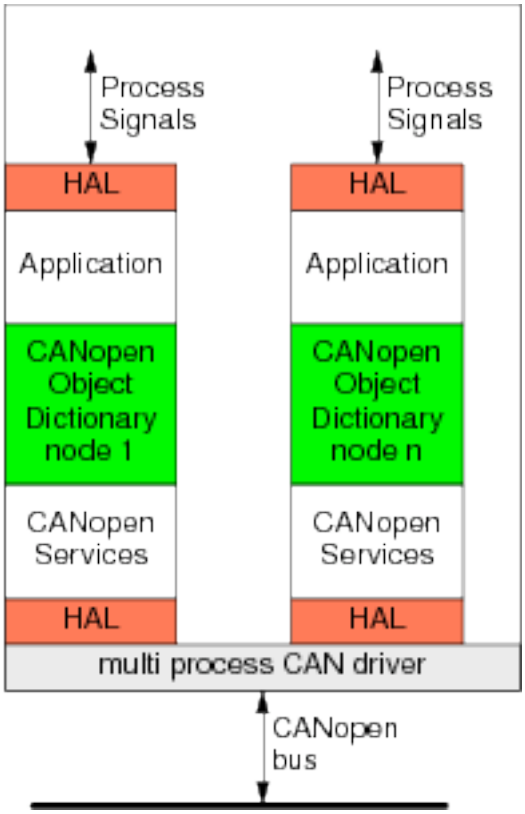

Figure 3: CAN HAL as multiprocess driver

Disturbing influences due to CAN bus physics can be ignored at first. If it is necessary to also test these cases they can be generated by the TCP/IP server. Moreover, the server can calculate the bus load which is reached at a desired bit rate. Conclusions can hereby be made on the distribution of the device functions in the network. It has to be taken into account that no bus arbitration takes place in the virtual network. The CAN telegrams in the virtual network are distributed and processed to all prototypes in chronological order through the TCP/IP server.

Diagnosis and configuration tools can be applied by using the same HAL. The virtual network can also be connected through the TCP/IP server directly to a CAN network.

Diagnosis and configuration tools then can be used in the network without any modification (Fig. 5). The prototypes in the virtual network appear as real devices.

6 One CAN interface per prototype

This approach is the 1:1 copy of a device as a prototype on a PC (Fig. 4). Large networks cannot be made on an individual PC. The number of prototypes which can be started at the same time on a PC is restricted by the number of CAN interfaces in the PC.

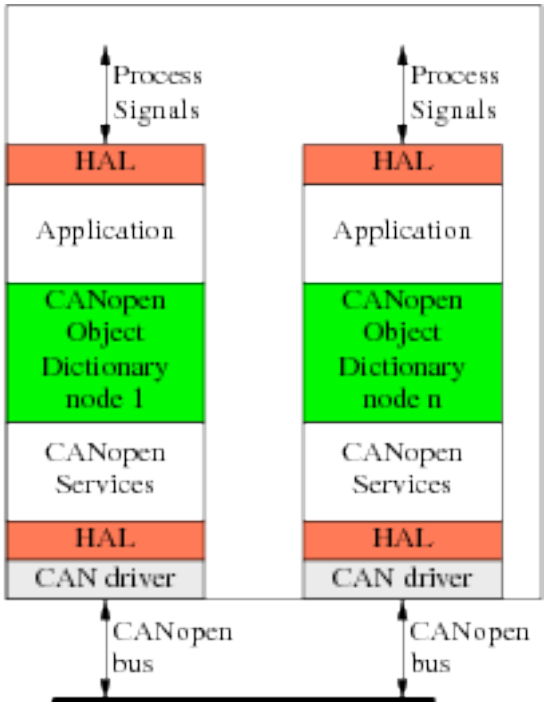

Figure 4: CAN HAL separate CAN interfaces for each prototype

The use of a PC for the prototype isn't mandatory. The prototype can therefore be created on every available platform. development boards are also a good choice for building prototypes. When small networks are to be developed all devices can be executed as prototypes on Development boards.

7 Incremental integration

During of the development, the prototypes can be exchanged for the target platform in the virtual network. Depending on the implementation of the hardware abstraction layer the exchange for the target platform occurs by setting other compiler switches or exchanging directories. A gradually comprehensible integration is the outcome of this procedure.

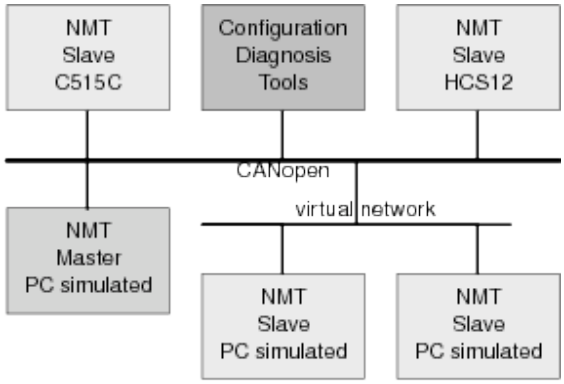

Figure 5: Network of prototypes

A test environment is created as a spin-off product where subsequent problems can be tested and solved. An individual device using the hardware abstraction layer can be started as a prototype on the PC while the remaining network consists of the real CANopen nodes.

8 Process abstraction layer

With a flexible implementation of the hardware abstraction in the process, the process quantities can be simulated by a separate module in the application (Fig. 6). Unlike open inputs with unpredictable values the process abstraction layer can produce values in predefined value ranges and predefined or recorded trajectory. This makes additional use of the simulation module as a test module in the later course of the system development possible. Therefore it is advisable to design the module to be configurable. A configuration possibility through CANopen is reasonable and easy to implement.

By the consistent use here of the programming language C the process simulation or the test module is operational both on PC platforms and on the target system.

The process quantities generated must be processed and, depending on the type of the process quantity, be transmitted by PDO. There are cyclical, synchronous or event-controlled PDO. Based on these PDOs the communication relationships between the devices are simulated. In the early development stage one can therefore receive a summary of the logical operations in the network (the PDO-Linking) without a message having to be transmitted physically over the CAN bus.

Using the calculated bus load, informatino about the PDO communication relationships or the distribution of the functions for the devices in the network can be derived. When an unfavorable network or function partitioning arises the prototypes can be adapted appropriately in this early stage of development.

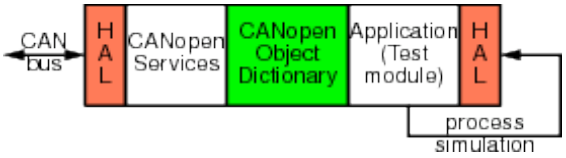

Figure 6: HAL and simulation of process signals

9 Summary

The use of a simulation software with a special programming language for the simulation seems attractive at first. The prototype approach has the advantage of the re-usability of the implementation in programming language C which is commonly used in embedded projects. A shorter training period necessary for learning another programming language and the operation of complex software accompanies the prototype approach.

The consistent use of a hardware abstraction layer enables the reuse of the prototype code on different targets. This simplifies a migration to a more powerful hardware if intial system tests reveal a shortage of computing power or memory resources for the application.

In conclusion one can say that this framework for Rapid Development of CANopen Systems is proven to be practiable and gives fast results. It has to be stressed that the double development of a simulation system of the device software can be omitted. The development times are considerably shortened by the prototype approach.

References

[1] CAN in Automation, Application Layer and Communication Profile DS301 (04 January 2006).
[2] CAN in Automation, Additional application layer functions DSP302 (18 April 2004).
[3] http://en.wikipedia.org, Software Prototyping.
[4] http://www.port.de/pdf/fli4l-can.pdf, TCP/IP - CAN/CANopen Server.

Glossary

HAL	Hardware Abstraction Layer
IPC	Interprocess Communication
TCP/IP	Transmission Control Protocol Internet Protocol