

Development-accompanying Emulation of CANopen Control Units under OSEK/VDX

Systems of networked control units that communicate via CANopen are becoming ever more complicated. While the total development must generally occur under extreme time pressure, requirements for the functionality of such systems are increasing. In order to produce the total system cost-effectively and on time despite these challenges, a verification of the system at the earliest possible point in time is necessary - or at least sensible. Unfortunately, however, the control unit prototypes are only available at a very late date. Even if the application code for the control units is written parallel to their hardware development, it is only possible to undertake a system verification after the hardware becomes available. However, it is possible to emulate the whole control unit software together with the input and output peripherals and the connected networks. Thus in all development phases of the project, a verification of the total system is possible.

Network Simulation as Control Unit Emulator

CANoe/DENoe, a PC-based simulation tool for networks from Vector Informatik GmbH of Stuttgart, has been used for years in control unit development to support the development process of networked systems from planning to start-up. At the beginning of the development phase, network planners - as a rule system manufacturers - plan a functional model of the system-wide bus systems. Using this model, the communication behavior of the control units that participate in the communication is specified and the re-

quired parameters are defined. For example, the model contains all messages and signals used, together with their parameters such as CAN identifier and transmission type. On this purely virtual network, already-important factors such as bus load and latency times can be estimated. This is depicted in the first step in Figure 1.

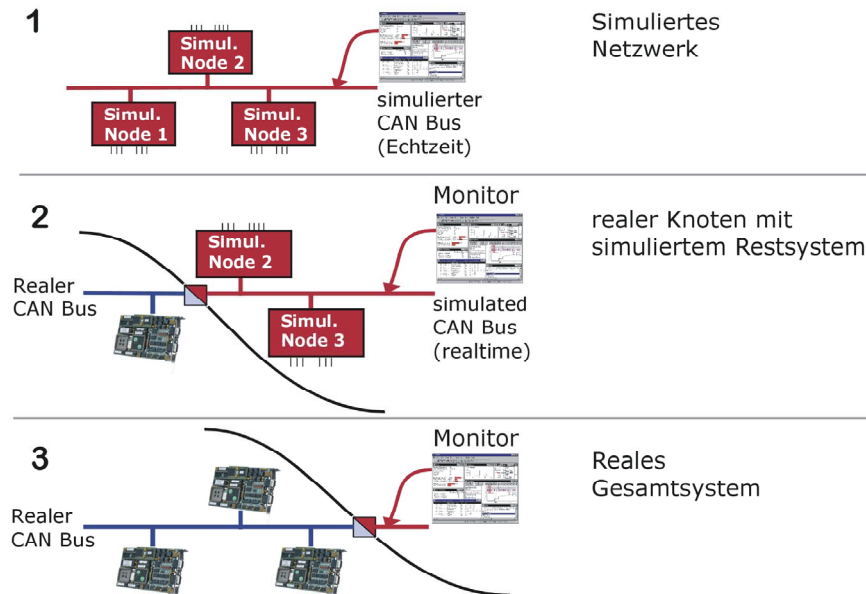


Figure 1: Development steps for the simulation of a network

A special property of the CANoe/DENoe tool is the coupling of a simulated network with the real bus. Thus, as soon as a control unit is available, it can be verified together with the simulated remaining bus. Figure 1 depicts the structure of the remaining bus simulation in the second step.

The emulation of the network nodes to be simulated for a remaining bus simulation occurs within CANoe/DENoe with the C-like programming language CAPL. Furthermore, it is possible to incorporate DLLs that contain higher protocol layers such as CANopen into the simulation.

The third step (Figure 1, at the bottom) in the development

process is the integration test at the system manufacturer. The originally purely-virtual nodes in the simulation model are replaced step by step with real hardware.

Integration of the Application Code into the Network Simulation

For a truly meaningful verification of the control unit under development, it is necessary to map it in as "real" a fashion as possible in the simulation environment. Ideally, exactly the same application code will be used for the simulation as for the real control unit. This task is handled by transforming the control unit software, which is usually written in C, into a DLL. In CANoe/DENoe, this DLL is assigned as the behavior model to the associated network nodes. To create the DLL, developers have available an osCAN-CANoe library. It contains the process flow environment for an OSEK/VDX-conforming operating system. In addition, a CAN driver layer is available that communicates via a CAN interface on the PC. The osCAN operating system - available as a real-time kernel for more than 30 microcontroller families - is a globally-popular process flow environment for control unit applications.

At the beginning of development, the application code for the control unit to be emulated must be designed. In most cases, this is the actual functionality of the control unit. For an engine, this might be the regulating algorithms and the activation of the engine power output stage, for example. Then the CANopen source code is configured so that the requirements made of the control unit are fulfilled. This configuration includes specifications about

the number of communication mechanisms made available and the protocols used. Furthermore, developers must also determine which application data the CANopen control unit makes available via the bus and in which CAN messages this data should be transmitted. Under CANopen, application data is available via a so-called object directory. This object directory assigns data and functions of the application a unique identification, which is formed using an index (16-bit value) and a sub-index (8-bit value). If data should be accessed via the CAN bus, this index is used as a "virtual" address.

During the development of the application code it is also specified whether individual tasks are hived off into separate operating system tasks and which operating system mechanisms are used. For the CANopen source code, a breakdown into individual tasks has already been undertaken. No additional intervention is necessary here.

Access to the CAN controller is also made available via the osCAN-CANoe library. The CANopen source code is already available for a large number of different hardware platforms. When used under the CANoe Option CANopen, the source code uses the same interface to the CAN controller as on these platforms. Only the driver calls are mapped to functions in the osCAN-CANoe library.

Once the application is created, the application code is compiled together with the CANopen source code via the Microsoft Visual C++ development environment into a DLL and incorporated into CANoe.CANopen. The osCAN-CANoe library is also linked to this DLL. This library contains a complete CAN driver as well as, with the exception of portions of

the interrupt handling, the complete osCAN operating system. Figure 2 shows the structure of control unit emulation using the original control unit code.

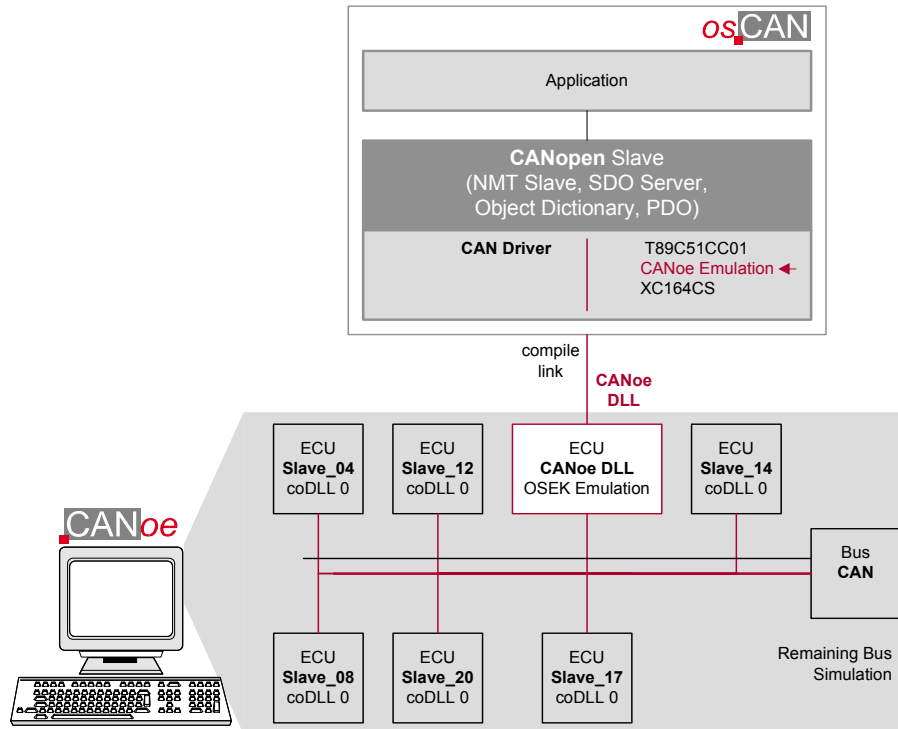


Figure 2: Integration of control unit code into the CANoe.CANopen simulation environment

Emulation of the Input/output Hardware

Since the code for the control unit cannot be used directly, the control unit-specific input/output hardware must be emulated. The hardware's behavior can be emulated best using the C-like programming language CAPL. Even more complex algorithms and state machines can be stored here, with help of which approximating the behavior of the real hardware is easily possible. In addition, it is possible to create panels that can emulate the switches and displays. Thus the simulation can also be designed in a visually-appealing manner.

Developers must transfer access to the hardware, e.g. to the interrupt controller or to I/O channels, to the CANoe.CANopen interface. Interrupt service routines can, however, be retained in the original control unit code and they can be activated manually by developers using a CAPL function or by a suitable CAPL algorithm. If in the control unit application code there is an interrupt service routine with the name "ISR_NewData," this interrupt can be triggered from the CAPL environment with the call `OSEK_Trap("ISR_NewData")`.

The connection between the emulated input/output hardware and the application code occurs via "environment variables," a process interface defined in CANoe.CANopen. These environment variables can hold integer values or strings that can then be accessed from the CAPL environment or via panels. For access from the application code, there are functions available that permit the reading or writing of the value of environment variables. Everywhere in the application code where hardware is accessed, this access is mapped to suitable environment variables. If a LED is to be switched on from the application code, this could be realized for real hardware with the following command:

```
"P3 |= 0x0001;"
```

Thus a port pin of Port 3 is set to "1" on the hardware.

In the emulation environment, by contrast, the function `CANoeAPI_PutValueInt("LED_1", 1);`

would be used. This call causes the environment variable "LED_1" to receive the value "1." This environment variable can, in turn, be displayed directly graphically on a panel.

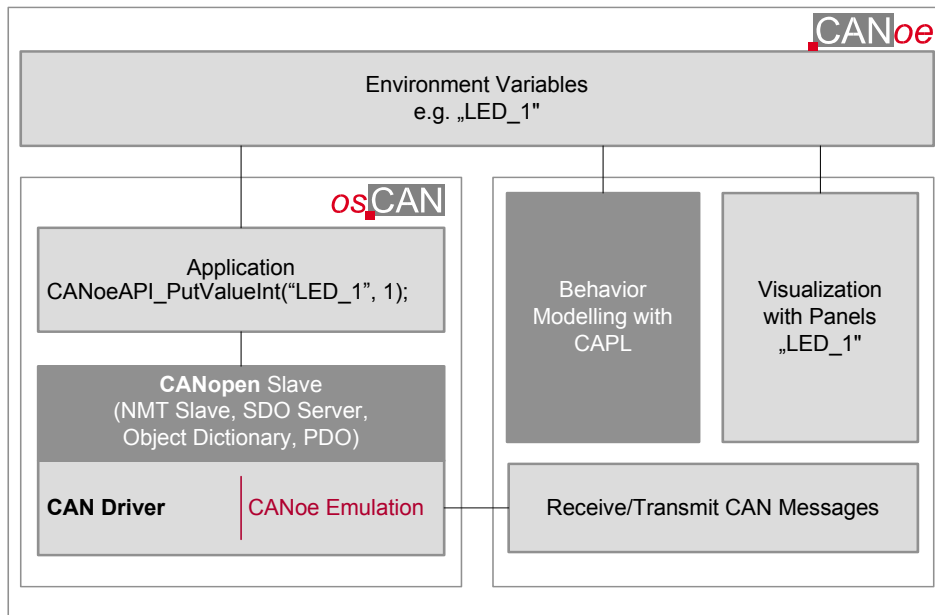


Figure 3: Mapping of real hardware to environment variables

Through a CAN driver specially tailored for CANoe.CANopen, there is an interface available for access to the CAN bus that can be used directly with Vector Informatik's CANopen source code API. Therefore, when changing over from the emulation to the target hardware, no changes are necessary for the CAN connection within the control unit application. Only the driver file for the selected target platform needs to be used here. The handling of the CAN interrupts also already occurs in this driver and requires developers to take no further measures. In principle, this procedure can be applied to all control units on the system to be emulated. Therefore, a separate DLL must be created for each individual control unit.

With this methodology, the CANoe.CANopen network simulation tool becomes an emulation tool. CANoe.CANopen thus enables not just an emulation of the network behavior of one or several control units, it also enables analysis of the process behavior of the control unit application and its functional behavior. Task-switching in the operating system can

be examined, just like the intertask communication and bus communication.

Generation of Simulation Nodes

Since in a complete system ever more control units must interact with one another, a corresponding emulation must be created for each individual control unit. Depending on the number of control units, this can mean quite a bit of effort for system developers. The simulation environment can be created more quickly if first of all a framework is generated for each control unit to be simulated. With the network management (NMT), the object directory, and the configured process data objects, such a framework already contains the basic functionality of a CANopen control unit. Technically speaking, behind this solution lies a generic node layer DLL that implements the CANopen behavior and a CAPL program that configures and controls this node layer DLL. Furthermore, there is also a panel available automatically for visualizing the control unit's process data via environment variables.

The CAPL programs and the panels are generated by the Pro-CANopen tool, which is included in the CANoe.CANopen scope of delivery. Of course in case of an automatic generation, system developers must intervene and determine how the control unit should look functionally as well. Using ProCANopen, system developers can specify on a graphic display which control units should later be contained in the total system. The properties of the individual control units with respect to CANopen are determined separately by an "electronic data sheet." This electronic data sheet specifies the structure of the object directory

and defines essential device properties. Furthermore, with the help of ProCANopen, system developers specify which data should be exchanged among the individual control units via CAN and whether this exchange occurs cyclically or event-oriented.

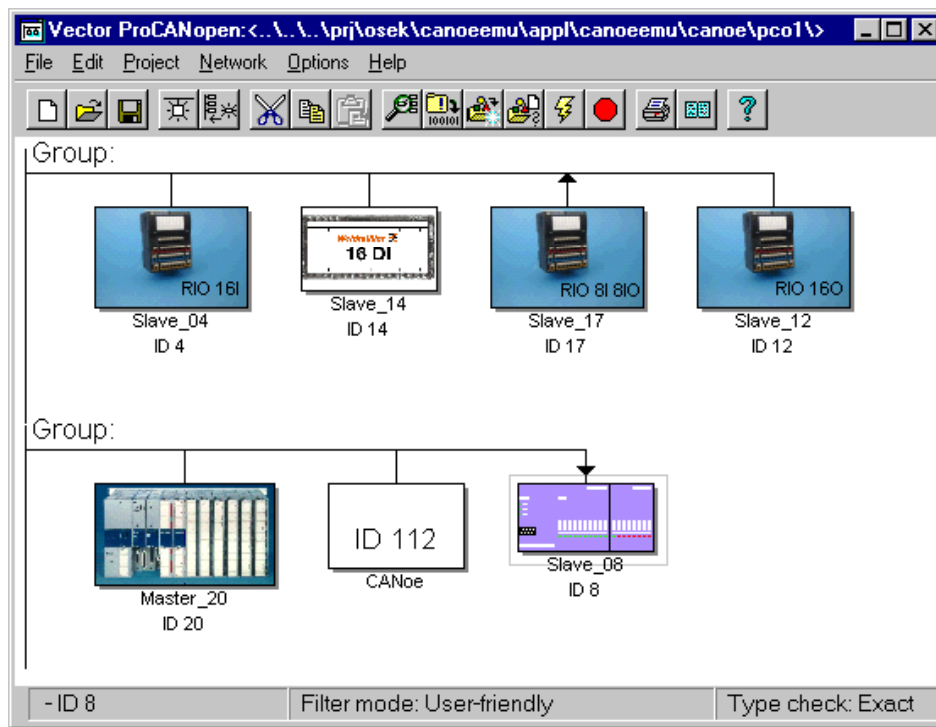


Figure 4: Generating the simulation environment using ProCANopen

If all these specifications have been made, a corresponding simulation environment is generated with the touch of a button. This simulation environment is then read into CANoe.CANopen.

Here it is possible to refine the behavior of the simulated control units by making small changes to the CAPL programs generated. Later on, simulated control units are replaced with the real application code according to the procedure described in the section "Integration of the Application Code into the Network Simulation." Thus step by step an ever more precise simulation is achieved.

Debugging and Testing

The methodology described above permits control unit developers to analyze bus communication and application behavior at a very early point in the development process without the presence of the corresponding hardware. This provides many advantages:

- Simple debugging of the application is possible. As usual, when the program counter hits the breakpoint, the detailed analysis of the system environment is possible since the complete system including bus communication is halted, that is, the "global time" of the emulation is stopped.
- The use of the wide-ranging CANoe.CANopen functionalities such as logging of data traffic on the bus delivers valuable information to the software validation process.
- Analysis of the application code is also possible without participation of bus communication. An example of this would be display control. Thus the focus of the validation is not on bus communication, but rather on the handling of the display I/Os.

For software validation, many tools including Microsoft Visual C++ are available to the development team.

Summary

Testing of control unit software is possible on a virtual or real CAN network even without the presence of hardware. The osCAN-CANoe library offers developers of control unit software the possibility to test their own applications in

advance with the CANoe.CANopen software tool. With the implementation of the OSEK/VDX-conforming operating system osCAN by Vector Informatik GmbH and a CAN driver based on CANoe.CANopen, developers can concentrate entirely on testing the self-created application code. Vector Informatik GmbH also offers integration of the higher network protocols such as network management so that developers need only activate the API of the corresponding protocol.

Status 02/2004

Number of words: 2,102

Number of characters: 13,605

Figure 1: Development steps for the simulation of a network

Figure 2: Integration of control unit code into the CANoe.CANopen simulation environment

Figure 3: Mapping of real hardware to environment variables

Figure 4: Generating the simulation environment using Pro-CANopen

All photos: Vector Informatik GmbH, Stuttgart

Author:

Mirko Tischer, Dipl.-Ing.

Tel. +49-711/80670-211, Fax +49-711/80670-111.

E-Mail: mirko.tischer@vector-informatik.de

Vector Informatik GmbH

Ingersheimer Str. 24

D-70499 Stuttgart

www.vector-informatik.com

Editorial contact person: Holger Heit

Tel. +49-711/80670-567, Fax +49-711/80670-555.

E-mail: holger.heit@vector-informatik.de