

Prototyping and testing CANopen systems

Mirko Tischer (Vector Informatik)

The majority of the development tasks within the framework of the V-Model can be ascribed to the areas of verification and validation. Comprehensive testing conducted as early as possible allows the developer to nearly exclude errors that might otherwise be discovered too late. The bandwidth of tasks involved in developing a CANopen system ranges from the development of a single ECU to configuration and start-up of the total system. The use of proven tools is advisable so that CANopen's flexibility can be fully exploited. Also, one must not overlook the protocol functionality that is needed to implement a single ECU. Each stage in developing the total system must always be accompanied by appropriate tests. In practice, initial testing is not performed in a real system at the first-tier customer. Instead a test bench is used which contains all system components of what will later become the total system. It also includes special measuring and diagnostic instruments for testing as well as actuators used to make the system environment as realistic as possible. Depending on the system size, such a test bench might be very difficult to construct and is therefore also cost-intensive. Generally only a single test bench is available, which almost guarantees that there will be bottlenecks in the testing phase.

A way out of this dilemma is a mature tool that makes it easy to prototype

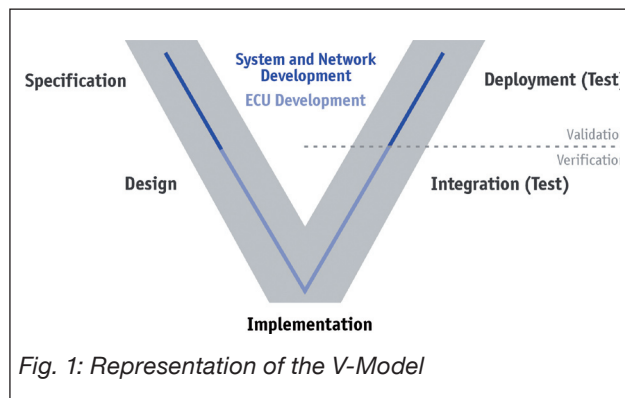


Fig. 1: Representation of the V-Model

the future total system. In an optimal solution this tool would also provide test functionality.

Prototype environment

First and foremost, the prototype of a total system networked by CAN should support testing and validation activities. Moreover, it should be available at a very early phase of the development project. Therefore it is important that individual components of the overall system lend themselves to representation by real or simulated ECUs. This makes it relatively easy to test the functional integrity of real ECUs over the course of system development. Consequently, the functionality of a prototype environment must extend far beyond that of a pure simulation.

Prototyping of a complex total system always incurs high cost and effort. Suitable tools simplify this task considerably. CANoe. CANopen from Vector Informatik (www.vector-informatik.com) supports the user

primarily in setting up communications for the prototype system. With just a few configuration steps it is possible to create a prototype system whose communication properties match those of the later real system. First, description files (EDS – Electronic Data Sheet) are selected for the CANopen ECUs that are to communicate with one another in the system. If no description file exists for a device, because its development has not yet been completed, an empty template is used as a placeholder instead. In the next

step, the application data to be exchanged later over the bus are interrelated, e.g. the input "PressureValve" of the device with Address 5 (pressure sensor) is linked to the variable "GasPressure" in the device with Address 10 (Control). This is how all PDO (Process Data Object) connections are defined for the prototype system. The resulting mapping is automatically calculated, and it may be modified afterwards. After this step all of the prototype system's configuration information is available in several configuration files (DCF – Device Configuration File). In a later step the user utilizes these files to create the prototype environment. That is, for each ECU in the real system a counterpart is generated in CANoe which possesses the same communication properties. The tool is started, the communication portion of the prototype environment is already available. The object directories of the (emulated) ECUs are ▶

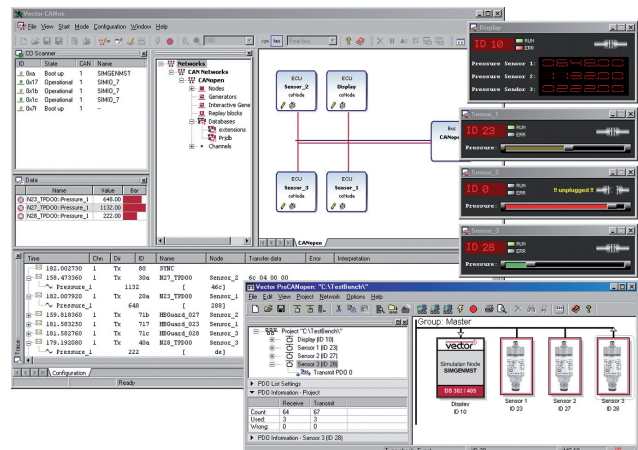


Fig. 2: CANoe.CANopen sample configuration for a simulated CANopen network

accessed by SDOs (Service Data Objects); it is possible to make additional changes within these directories.

Application behavior

Also of interest in prototyping is the application behavior of the individual ECUs in the system. This behavior cannot be derived from the EDS files, because only the framework of an object directory is represented there. Generally, the formulation of application behavior is associated with additional programming effort. The software tool with its integrated CAPL programming language describes ECU behavior very simply. An alternative would be to describe ECU behavior in a DLL. Such a DLL is written in C or C++ and is then linked to the prototype environment under the tool. It is also possible to integrate Matlab/Simulink models trouble-free. Depending on the required level of detail, the prototype can always be refined afterwards. Upon completion of the prototype system there follow the necessary tests of the total system. In this context the tool offers support, both in the creating the tests and in evaluating and logging them. The test functionality required of a CANopen system encompasses several levels: Protocol level:

An example of this is testing of the SDO protocol as defined by the conformity test of the organization CiA e.V. This involves sending a request to the DUT (device under test), then evaluating the received response. This is a way to test whether CANopen-specific communication protocols have been implemented correctly in an individual device within a total system. Communication level:

The correctness of the protocol is not tested here, rather a check is made to verify proper logical flow of (independent) protocol sequences, e.g. in the configuration of PDOs. In the case of PDOs, the PDO-relevant entries in the object directory must be written in a specific order. Observance of this order can be tested (Good Cases), as can the reaction of the DUT to a faulty order (Bad Cases). Creating this test requires detailed CANopen know-how. Primarily this task involves working with the interdependencies between the different communication mechanisms that are used.

Application level: Application tests check the interrelationships between process variables. To even be capable of determining these interrelationships the following prerequisites must be satisfied: The process variables must be exchanged by PDOs and the

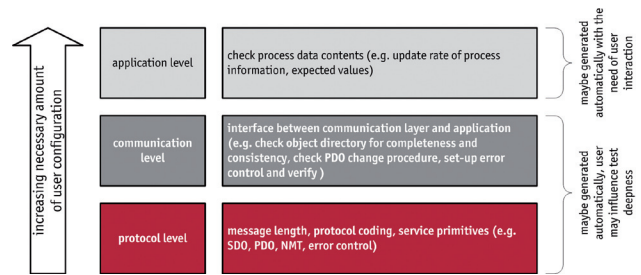


Fig. 4: CANopen test levels

system must be fully configured. For example, the state of a valve might be tested as a function of temperature or pressure. Such examples suggest that it is clearly the user who must formulate the test.

Test procedures

Under the tool, test procedures are formulated with the help of the integrated CAPL programming language. The developer uses this language to formulate extremely flexible tests for complex communication systems.

Each CAPL test module is a separate test consisting of many individual test cases. In turn, each of these test cases contains a number of test steps. During test execution the tool runs through the individual test cases sequentially. Suitable test flow control could enable skipping or repeating of specific tests. A dynamic aspect is introduced to test functionality in this way. The process of building test cases is greatly simplified by predefined CAPL functions. A typical test sequence might be structured as follows: After stimulation of the DUT, the tester waits for the response which is then evaluated. There are functions for synchronizing the test procedure to events such as receipt of a specific message or a changed environment variable value (which might also be modified via COM). At the same time fulfillment of other conditions and constraints can be monitored in quasi background. This approach makes sense if,

while waiting for a specific message, the user wishes to check whether a different message is being sent periodically on the bus. Especially when setting up automatically executable tests, it is important to keep a detailed record of the results of individual test steps. Other CAPL functions may be used to write test results to an XML file for post-processing. The results are also available as an HTML file for immediate evaluation. The tool's test procedures may also be specified under XML. This is preferable if many similar test procedures are being generated with a tool. The tool utilizes a large number of test patterns specified in XML and can process them suitably.

Summary

Prototyping in CANopen-networked systems is always associated with significant effort. Nevertheless, prototyping is often essential in order to arrive at conclusions regarding functionality and system performance without having to wait until a very late project phase. The user gets support in creating a prototype from special tools. In particular, coverage of technical communication requirements is very easy to implement. The tool's test functionality lets the system developer perform verification work in every phase of a project, right up to the finished system. ■

mirko.tischer@vector-informatik.de

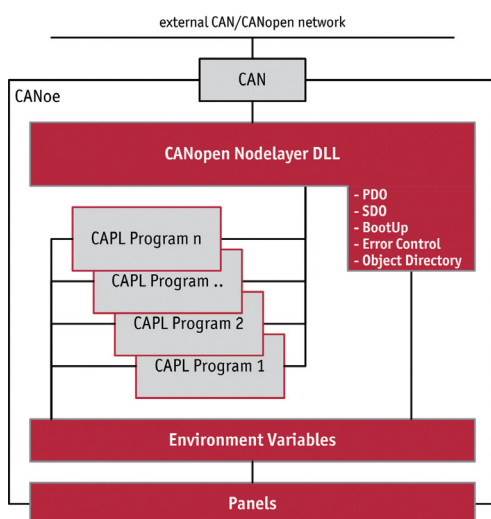


Fig. 3: Overview of the "CANoe. CANopen" Environment