Automatic CANopen Test Generation

Kai Schmidt, Vector Informatik GmbH

The growing complexity of today's system architectures is associated with an increase in the effort that must be invested in test specification, test creation and test execution during the development of such systems and system components. Test specifications should be available in early phases of the development process, e.g. after the system architecture has been created or during component design. This makes it possible to detect errors early and correct them cost-effectively.
Device descriptions can be prepared for CANopen systems as early as after definition of the component architecture. Together with the system definition, the device descriptions form the basis for creating test specifications. They can be used to derive executable test sequences, which in turn can be executed in a suitable runtime environment.

Definition of the overall system

The development process for a CANopen system can be described based on the V-model (see Figure 1). In the first phase, system requirements are defined, which for the most part contain the definitions of individual "use cases" [1]. This information represents the input for the next step, where initial assessments can already be made of the system architecture. Functions are assigned to the individual ECUs, and device descriptions can be created for all devices in the form of EDS files (the format of the EDS files was standardized by CiA – CAN in Automation – and is being further developed by this organization in cooperation with industry). In addition, communication relationships between the ECUs can be configured, as network management and error detection mechanisms. These definitions form the foundation for:

1. Simulation of the overall system
2. Creation of test specifications
3. Requirements specification for suppliers

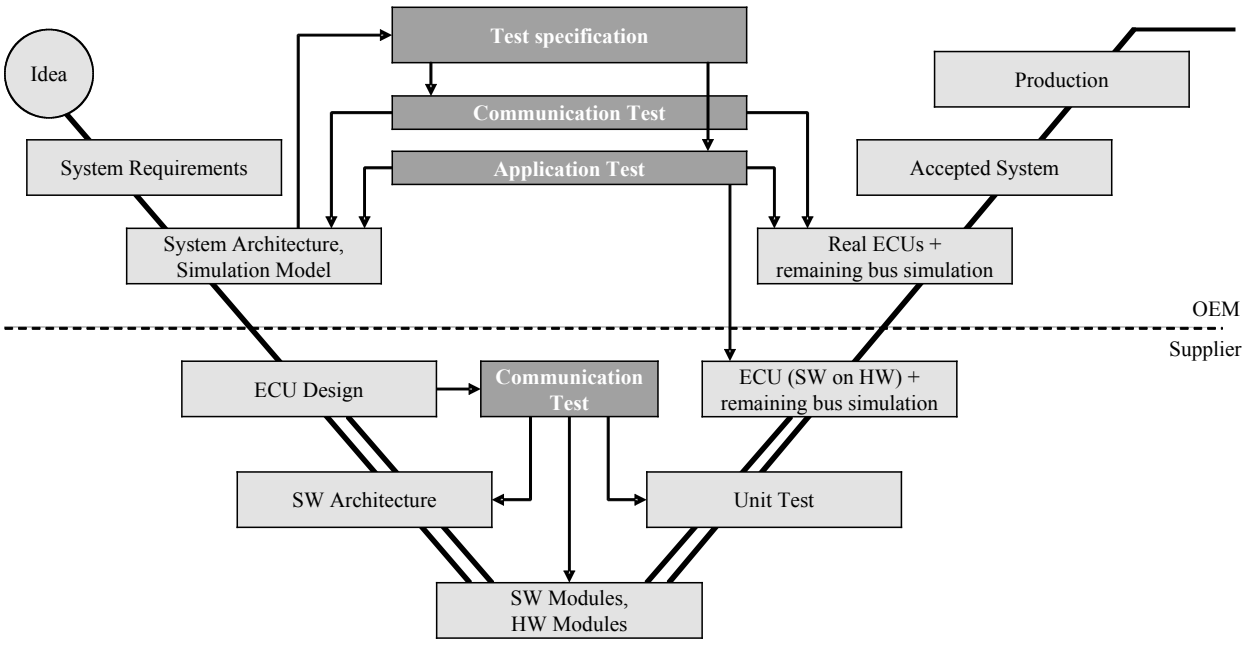

Figure 1: Development process of a CANopen system

A first test specification

EDS files describe significant parts of the functional scope of a CANopen device. These device descriptions form the foundation for executing the simulation and creating test specifications. Communication-specific tests can be derived directly from the device descriptions. An example of this would be a test that checks all objects in the object dictionary by SDO accesses and records the results. Besides communication-specific tests, application-related tests can also be specified. An example of such a test would be to stimulate the transmission of the digital input of an I/O device. Afterwards, a check is made to verify that the signal value exists at the output. Both tests could be used early in the simulated overall system. As soon as the stability of the overall system has been achieved, development of the individual components can be subcontracted. The EDS files can – with the exception of application-related behavior – be considered as a requirements specification for the supplier. Parallel development of the ECUs at the suppliers is accompanied by the simulated overall system. Application-related tests can also be utilized at the supplier to test the behavior of the device to be developed within the overall system. This can significantly reduce the number of cost-intensive changes desired by the OEMs – which generally occur within the integration phase. Communication-specific tests can be created at the supplier in a similar way as at the OEM.

Integration of the components

After completion and acceptance of the components, they are successively integrated into the simulated overall system. The previously created communication and application-related tests can now be applied to the system, consisting of the physical components and the rest-of-bus simulation. As soon as all of the components have been delivered the concluding test of the real overall system follows.

EDS files as the basis for generation

The development process should include the creation of an EDS file appropriate to the device. Unfortunately, practice shows that device producers often neglect this work step. Faulty or incomplete EDS files are the result; in the worst case there is no EDS file at all for a device. The development process described above shows that it is not just device producers who need to be concerned with creation of EDS files, but system designers too.

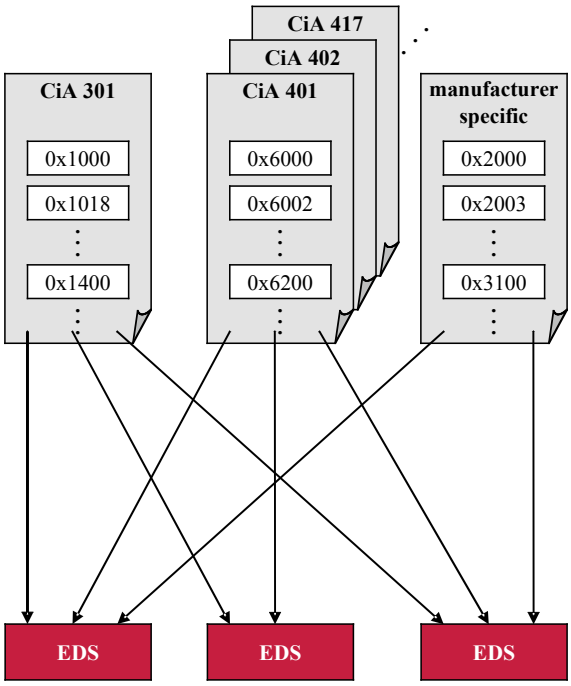

Figure 2: Arrangement of device functionality

The task of the system designer here is to distribute functionalities to the individual components. These could be standardized functionalities such as mechanisms for process data communication, but they might also be manufacturer-specific functionalities. Both of these are mapped via objects in the object dictionary. Standardized functionalities are described by CiA in standardization documents. Both the objects described in these documents and manufacturer-specific objects can be stored in a database format that is also standardized. The necessary objects can be selected from the object pool that is

created in this way, and be assembled into an object dictionary (see Figure 2).

Flexibility of the simulation

The device descriptions contain all of the information necessary for simulation of the CANopen device. The overall system, consisting of the individual device descriptions, is parameterized utilizing a suitable configuration tool, and an initial system description is obtained in the form of device configuration files (DCF), whose format has also been standardized by CiA. Based on this configuration, simulation models can be generated and executed in a suitable runtime environment. At an early point in the project, this already enables conclusions about the time behavior of the overall system. If excessive bus loads occur, for example, actions can immediately be initiated to correct the problem, since suppliers have not been involved in the development process yet. Accordingly, the simulated overall system offers a high degree of flexibility. It can be refined iteratively until it satisfies the defined requirements. Changes to the simulated system can be implemented cost-effectively and be checked immediately.

Derivation of test sequences

Besides the simulation, it is also possible to derive initial tests on the protocol and communication levels from the device descriptions. The protocol test includes checking of the SDO protocol, for example. The communication tests do not check for correctness of the protocol, but instead for correct flow of message sequences. For example, it is possible to test whether the configuration sequence for process data objects conforms to the sequence specified in CiA 301 [2]. The following test templates with general application can be defined for a CANopen device:

- SDO Download Test
- SDO Upload Test
- Heartbeat Producer Test
- Heartbeat Consumer Test
- Transmit PDO Test
- EMCY Test

Simple generation templates can be used to create device-specific test sequences. Test functions are created for each object contained in the object dictionary here. The test functions are parameterized based on the data contained in the configuration files for the devices. Among other things, test sequences can be generated to check the:

- PDO configuration
- Default values
- Object dictionary
- NMT state machine, and
- SDO protocol

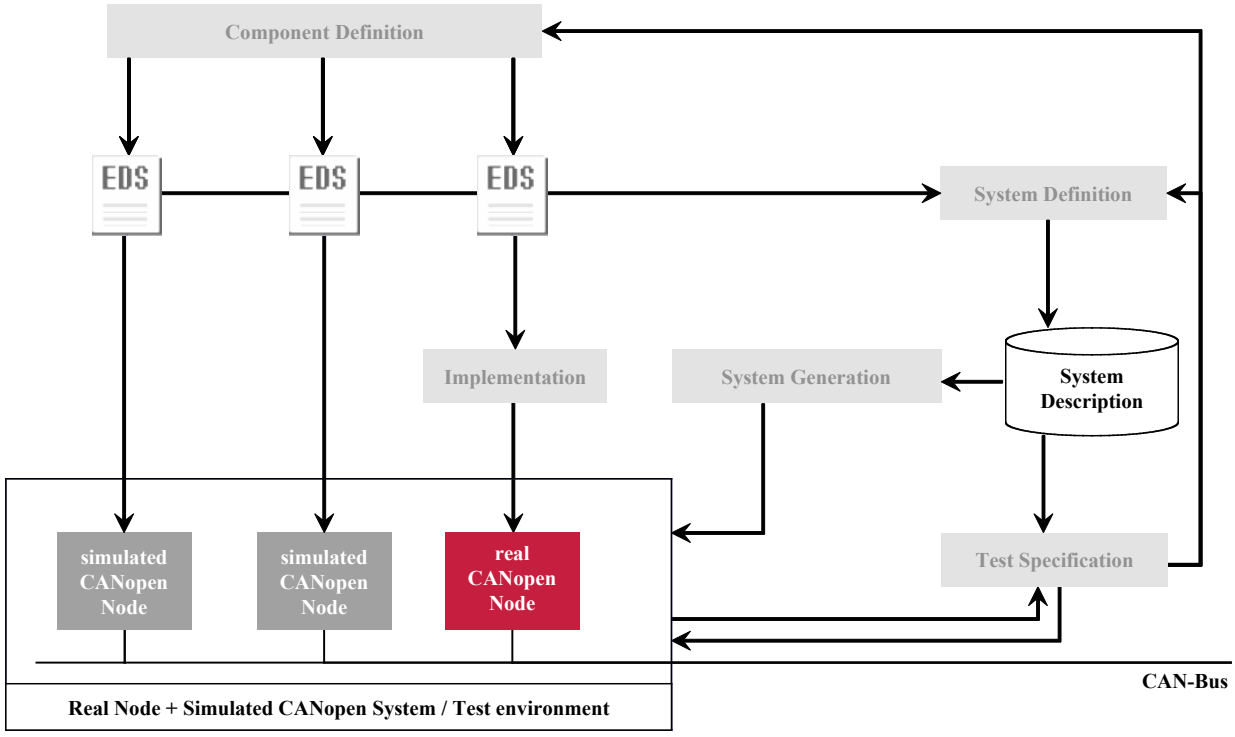

Figure 3: Development process of a CANopen system

The generated tests may be executed right away in a suitable runtime environment. In the framework of integration work, it is precisely such tests that are used to check the delivered components. In turn, suppliers can generate similar test sequences to assist in development. They can immediately be applied to the prototypes. Essentially, this is a way to generate test sequences of the conformance test (CiA 310) node-specifically. However, the goal of the system should not be to replace the CiA conformance test altogether. The system should accompany the development and give developers a way to test devices in advance of the actual tests. The final certification is only performed by CiA.

Generation templates

Generation templates must be created for each test, but they are applied to each device to be tested. A generation template that describes the creation of a test for checking the object dictionary would appear as follows:

```
for all objects
{
  get access type
  if(access == read only){
    add test function SDO Upload
    to test sequence
  } // if
  else if (access == read write){
    add test function SDO Upload
    to test sequence

    add test function SDO Download
    to test sequence
  } // else if
  .
  .
}
```

The generated test sequence created based on this test template contains a number of parameterized (by entry of object index, etc.) write and read routines. They are processed sequentially in test execution.

Iterative development process

Since iterative processes are applied throughout a device's development, the process for generating test sequences must be repeatable as often as needed. Changes to the device design can affect the device descriptions. The test that was originally generated would then likely fail. Nonetheless, it is still necessary to be able to manually extend test sequences after generation, e.g. to incorporate application-

specific supplements. These extensions must be read back when the sequence is regenerated. Extensions must be implemented in interface functions whose call and function definitions have already been generated into the test sequence:

```
Test start

// check object 0x1000
PreSdoUpload_1();
SDOUpload_1(0x1000,…);
PostSDOUpload_1();

// check object 0x1017
PreSdoUpload_2();
SDOUpload_2(0x1017,…);
PostSDOUpload_2();
PreSdoDownload_2;
SDODownload_2(0x1017,…);
PostSDODownload_2();
.
.
// check object k
PreSdoDownload_n();
SDODownload_n(0x1017,…);
PostSDODownload_n();

Test stop
```

Application-related tests

The application-related behavior of the devices can not be represented in the device descriptions. Furthermore, the tester does not want to have to deal with the CANopen-specific conceptual world and its definitions on the application level. On this level, it is entirely unimportant to know which signal is mapped to which object at which position. More important is information about which signals exist and which devices receive or send them. Among other things, precisely these aspects must be tested. This subject matter can be explained by the example of the CiA 447 application profile (application profile for special-purpose car add-on devices):

The standard defines an object “GPS date”. Mapped to this object are the signals “GPS date year”, “GPS date month” and “GPS date day”.

Attribute	Value
Index	$60B3_h$
Name	GPS date
Object code	Variable
Data type	Unsigned32
Category	See /CiA447-2/

31 26	25 20	19 16	15 0
r	*GPS date day*	*GPS date month*	*GPS date year*

Fields	Value	Definition	Unit
GPS date year	0000_h $FFFD_h$ $FFFE_h$ $FFFF_h$	Minimum value Maximum value Failure Signal not available	Years
GPS date month	00_h 01_h $0C_h$ $0D_h$ $0E_h$ $0F_h$	Reserved Minimum value (January) Maximum value (December) Reserved Failure Signal not available	Months
GPS date day	00_h 01_h $1F_h$ 20_h to $3D_h$ $3E_h$ $3F_h$	Reserved Minimum value (1^{st}) Maximum value (31^{st}) Reserved Failure Signal not available	Days
r	$11\ 1111_b$	Reserved	

Figure 4: “GPS date” object description

The CiA 447 profile, besides defining signal allocations in the objects, also defines the transmission type. The standard specifies that the object value “GPS data” is transmitted by SDO protocol. The following information is needed to transmit a signal as part of a test:

- Index + Sub-Index of the object
- Signal length
- Start position of signal within the object

The format of today’s EDS files is unable to describe the signal allocations of object values. Accordingly, information such as the signal length and start position of the signal is also unsupported. Even if these requirements could be implemented, it is not possible to automate generation of application-related test sequences, since the behavior of the system is not described.

A generated test environment

Nonetheless, the developer can be supported by generation of an “interaction layer” in test creation. If this extension can be integrated in the simulated overall

system, then it is easy to create application-related test cases.

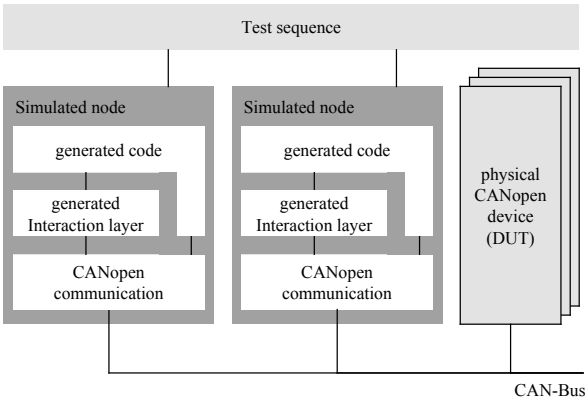

Figure 5: Generated test environment

The test system consists of the simulated nodes that are extended to include an "interaction layer". One or more physical devices are tested. The simulated devices are stimulated via generated interface functions. Signal values are mapped to object values and the CAN messages are sent. In the example depicted, the signal value "GPS date month" would be mapped to the relevant position in the object value (startbit 16, length 4 bit). Without this mapping, the test step for sending the signal "GPS data month" = 12 (December) would appear as follows:

```
// write signal "GPS data month"

SDO Download(src,
             dst,
             0x60B3,
             0xC0000);
```

Parameters one and two define the send and receive nodes. The relevant object is given in parameter three. Parameter four describes the object value to which the signal (signal value = 12) was mapped. Parameterization of the test functions assumes that the positions and length of the signals are known. Moreover, the transmission type must be considered. This information is described exclusively in the standard and must be considered in test creation. Use of an "interaction layer" enables signal-oriented test creation. The following test function is used to send the signal "GPS data month":

```
Send_GPS_month(src,dst,12);
```

It will be possible to define the function "Send_GPS_month" and generate its implementation based on the CiA 447 specification, if it exists in XML format in the future. Today's format of the specification requires converting the specification to a readable format (XML or Excel). This conversion task can be assumed by a generator. The generated functions contain a mapping of the signal to the object value and a routine for sending the CAN messages. During test creation, the test engineer need not be concerned about signal positions, indices or transmission types. All the test engineer is interested in are the signal name, sender, receiver and signal value.

Summary

Continually shorter development cycles necessitate a development process that is continually becoming more effective. Serious errors that are not localized during the integration phase drive project costs to enormous heights. Test scenarios that assist in development can make a contribution to detecting errors as early as possible and correcting them cost-effectively. Based on EDS files, it is possible to automatically generate test scenarios that check communication behavior device-specifically. Generation of application-related tests is not possible, since normally there is no standardized syntax for describing the behavior of the application. Nonetheless, generation of an "interaction layer" in test creation still offers test engineers support. The basis for this is formed by the application profiles standardized by CiA. The use of an "interaction layer" enables simple, quick and error-free creation of test scenarios of any desired complexity.

Kai Schmidt
Vector Informatik GmbH
Ingersheimer Str. 24, D-70499 Stuttgart
+49 711 / 80670-0
+49 711 / 80670-249
kai.schmidt@vector-informatik.de
http://www.vector-informatik.com

References

[1] Jürgen Klüser, Vector Informatik GmbH: Test Requirements in Networked Systems

[2] Mirko Tischer, Vector Informatik GmbH: Prototyping and testing CANopen systems

[3] CiA 447, Application profile for special-purpose car add-on devices