

Test requirements in networked systems

By Jürgen Klüser (Vector Informatik)

This article discusses concepts of combining system prototyping with test case generation along the V-model. The use of CAN with J1939 or CANopen-based higher-layer protocols leads to cost-efficient and flexible solutions, but also to a high increase in the electronics' complexity. An additional complication is the typical approach of distributed development between OEMs and several suppliers. The consequence has to be a systematic improvement of the development process. Project risks are to be reduced by taking testability as a design requirement and by performing the appropriate tests in the very early project phases.

The development process of a networked system can be described by the V-model. In contrast to pure theory the reality shows a delay of the implementation phase. Testing activities, even if planned in the beginning, are specified just before they have to be performed. A typical argument is that the implementation phase will bring expertise and even changes in the original design and maybe even a more detailed view on the requirements. The consequence of late test specification is a high risk. Due to the reduced time remaining, this most often will delay the overall project – or testing will not be performed sufficiently. Problems in the implementation or in the design, found at this late stage cannot be solved in time and tested again.

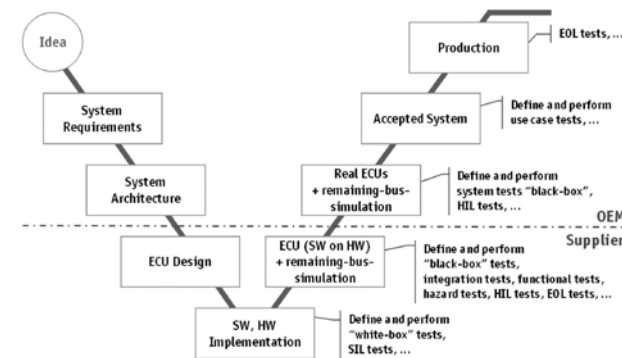


Fig. 1: Today's testing activities in the product's life cycle

Testability as a system requirement

The V-model includes two basic patterns. The first says that every phase is identified by results that have to pass a quality gate. The second demands that the phases and their quality gates on the right side are a directly traceable consequence of the corresponding phase on the left. The system requirements mainly define "use cases". This clearly should include topics such as testability as an input for the architecture, requirements for end of line tests, and diagnostics requirements. Applying the two basic patterns

to that approach of testability as requirement forces the system architect to provide specifications for tests in the early phases. Experience with that approach in the diagnostic domain as presented by [1] showed a significant increase in quality and process stability. This can be applied to the testing domain in the very same manner.

System architecture

For years it has been state of the art to describe the system architecture with simulation tools as a simulated model. Besides manual review technology this allows to systematically verify many aspects of the design.

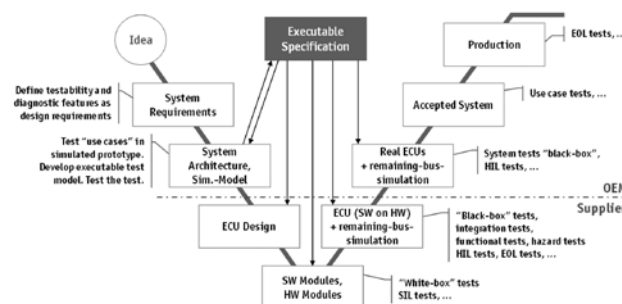


Fig. 2: Approach with an executable test specification

Consequent application of this approach leads us to the concept of the executable specification. This is true also for the testing domain [2]. Specifying tests in an executable form leads to a first implementation level of these tests. This immediately brings a set of benefits:

The system architecture model can be tested systematically. The real world shows that problems detected in later phases may require slight modifications in the design. The tests can then be applied as regression tests. This means passing the Quality Gate Architecture can be achieved with less effort on a higher maturity level.

Even the test concept itself can be tested in an early phase. A valid argument against writing test specifications before the verification and validation phases had been that the implementation almost always brings modifications and changes. These changes invalidated test specifications and therefore resulted in wasted effort. An executable test spec can be changed without the need to specify everything from scratch again.

From the communication's perspective the executable specification has further advantages: After definition of the relations between ECUs, a big part of the communication description can simply be generated. This is even more true in those cases where a higher-layer protocol such as J1939 or CANopen is used. ▶

These standards give much information about how relations have to be realized. The object model of CANopen allows generating the complete communication behavior after selection of the object relations. Tools with built-in knowledge of these protocols not only generate complete communication system simulations, they additionally generate many test scenarios that are ready-to-use. From pure "use-case tests" as the highest layer this can go downwards to "protocol tests" and "communication tests".

Refining the system model by a basic behavior description specifies the requirements as an input for the ECU design. The test specifications are refined in parallel. These are to be passed to the ECU suppliers. Today they are faced with the problem of insufficient of inconsistent specifications. Improved by the concept of passing simulations, the full strength develops with having efficient test cases available that are part of the development contract. Regardless of whether the behavior description is model-driven or conventional, regardless of the abstraction level - the simulation tool must support this by a built-in description language or by an interface to external behavior modeling – or the best of both. Whether this is UML-based, XML-based, proprietary or whatever goes beyond the scope of this article.

ECU development

With a written specification, designing, implementing and testing it on that basis only, is pure theory. In reality the developers need an environment with which they can implement and test stepwise. Like a debugger or emulator for the software environment the remaining bus simulation part most often is the only

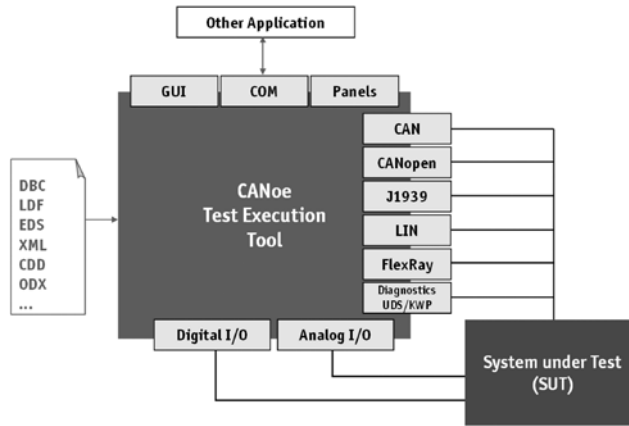


Fig. 3: Testing domain

chance to provide a sufficient environment. For example, implementing and testing operation modes of an ECU is not possible without correctly working network management. A typical problem in these small development steps is the availability of the counterpart of the communication modules. It costs much effort and time to set-up or implement the tests. For many cases standard analysis tools are a good choice. Much more efficient is the support by a dedicated test tool. It should provide the generated tests not only as a set but should allow for the selection, grouping, and execution of single tests easily. To accomplish this each test needs clear pre-conditions and constraints. Specifically the protocol tests and communication tests help to detect problems very early and therefore reduce the number of iterations be-

tween implementation and verification phases.

Re-use and refinement

The final phase of the ECU development is the ECU verification. Potential modifications of the original specifications caused by implementation experience lead only to modifications of the executable prototype. These immediately can be applied in the form of the "remaining bus simulation" concept. And the same is true for the modeled tests.

For many higher layer protocols user organizations provide conformance tests, for example the CANopen conformance test tool of CiA, or the Isobus test of DLG. It is strictly recommended to use such support. It guarantees that an ECU fulfills a certain level of conformance and interoperability. Anyhow, an intend-

ed drawback is that if a test fails, the tools do not really help in finding the reasons. This is a gap that has to be filled by the dedicated development test tools. So, once again the ability to perform single tests with meaningful tracing and problem tracking – and this with fast turnaround times – is indispensable.

As an example take CANopen: If the SDO (transport protocol) implementation has a problem accessing an object, the conformance test just will tell you "SDO failed". Repeating this in order to find details will require a long turnaround time again. A development tool with a test feature set and its generated "protocol test" will quickly tell, what messages have been wrong and can support in finding the problem source even down to the bit level. As already demanded for the ECU design, the ability of a test description language is essential here as well. In contrast to the pure conformance test tools the test procedure needs to be extendable by ECU-specific or application-specific features.

A strong argument for remaining bus simulation is to make the system integrator capable of starting the integration of an ECU before all other ECUs are available. Features such as simulated network management let the ECU think it was in a real environment. This is also a precondition for starting the first integration tests. This will save time for example if delivery delays of single ECUs occur. The automatic tests can be applied. Final integration will then only require regression tests. The focus here will be on the level of the communication tests, while the use-case tests are more a subject of the validation and acceptance phases.

The behavior of an ECU depends on its whole environment. Essentially known from the HIL concepts ([3] ▶

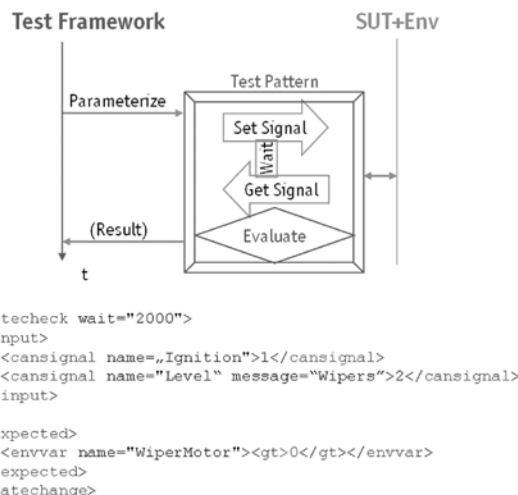


Fig. 4: Test pattern approach

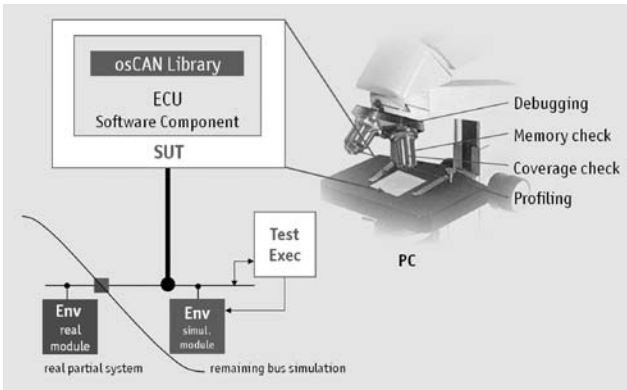


Fig. 4: White Box Test support

and [4]) any test environment needs to fulfill that aspect. As shown in Fig. 3 this requires the ability to treat all relevant bus systems, but also to observe and control digital and analog I/O. This includes the treatment of standard and OEM-specific system and device descriptions such as DBC (CAN), LDF (LIN), EDS (CANopen), CDD/ODX (Diagnostics) and many more. Besides a GUI it shall be able to define and use user-defined panels to application-specific software. For White Box Tests it is desirable to have an interface to debugging tools. For example in car ECUs the real-time operating system OSEK is used quite often. Here for example debugging and control of task states is essential. As mentioned above, for Black Box Tests it is important to have various possibilities to efficiently describe the test activity, sequence, and timing. This should be selectable from scripting or with a more abstract test pattern approach. In both cases the support has to include basic bus and I/O operations like output bus messages (CAN frames, MOST frames, J1939 PNGs, etc.) but also signal operations. For easy treatment of timing conditions the system has to support event conditions with Wait constructs for messages, environmental conditions, and signal conditions in any combination.

There are many highly sophisticated tools for editing, checking and format-

ting XML. The user does not need knowledge about the XML structure itself or about complex scripting lan-

guages. The open structure for tools can be utilized for test generation tools, too. Typically domain know-how exists in various forms. Examples for test-case generations are:

- ◆ Doors: Structuring requirements from which test-case can be organized.
- ◆ CANdb++: Monitoring of communication parameters such as cycle times, message DLCs, mapping of signals.
- ◆ CANdela Studio and CANoe Option DiVa: Specify diagnostic ser-

vices and generate the necessary tests.

- ◆ DaVinci: Test mapping of all input and output signals to ECUs.
- ◆ ProCANopen: Generate complete CANopen protocol tests.

Last but not the least – all test support is useless without a thorough reporting of the results. A modern test tool will provide automatic reporting. By using XML for the test patterns and specifications, groups of tests, test cases and sub-test-cases including their respective invariants the ►

test tool will control the test execution. At the same time it will create the report, formatted by a style sheet. It contains the test structure, the test cases, the overall result, and the test passes and fails with their respective environmental conditions. The impact of consequential test concepts is very often neglected in the early phases of a project. For designing, specifying, implementing, and executing tests near the end of the project, the typical lack of time in that phase is a high risk for the project delivery date, the costs, and in particular for the quality of the product. Applying concepts of model-based development, rapid prototyping, and code generation leads

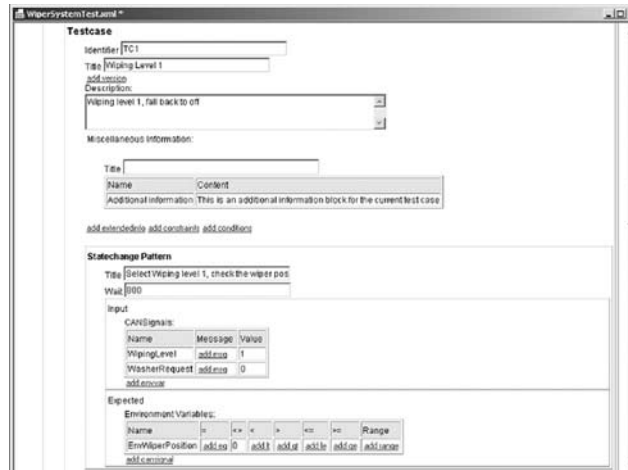


Fig. 6: Test description in Altova AuthenticDesk (XMLSpy)

to an executable test specification. This shifts much of the work to earlier project phases, provides a testing environment already before the implementation and

at the same time avoiding double-work for test specification changes due to design changes. The quality of all phases is significantly enhanced, the efficiency of

the implementation phases is increased, and the project risk and cost are reduced.

References:

- [1] Schlingmann, N.: Challenges and methods in the development process of CAN based diagnostic systems, Vector Symposium CAN in commercial vehicles 2005.
- [2] Tischer, M.: Prototyping and testing CANopen systems, CAN Newsletter March 2006.
- [3] Beeh, S.: Testing with CANoe, Vector Congress 2004.
- [4] Bardelang, T.: CAN in the HIL for the new Actros - Experiences in the test of networking mechatronic systems, Vector Symposium CAN in commercial vehicles 2004.

info@vector-informatik.de

Tool simplifies development, test and service of 16-bit and 32-bit μ Cs

PLS (www.pls-mc.com) introduced the UDE tool version 2.0, a tool for the development, test and service of micro-controller applications. The tool, together with the equipment family UAD 2+, supports popular 16-bit and 32-bit micro-controller families including the Freescale (MAC71xx), Infineon (TC116x, TC176x, TC1792, TC1766, TC1766ED and TC1796ED), Marvell/Intel (PXA255 and PXA27x), Philips (LPC3180) and ST-Microelectronics (STR910). The tool comes with a HTML-based profiling page, which simplifies the evaluation of trace data since it allows the tracking of how its run-time is shared by the individual functions of an application. Also accepted as data sources are on-chip debug support (OCDS) Level 2, embedded trace macrocells (ETMs), on-chip emulators such as Infineon's Multi Core Debug System (MCDS), simple 'Instruction Pointer Snooping' (XC16x and TriCore micro-controllers) or also simulators. Implemented with the



tool is an OCDS Level 2 option, which supports program tracing both for the TriCore versions 1.2 and 1.3 up to 180 MHz as well as for the peripheral control processors PCP and PCP2. Additional features include activity tracing for the direct memory access (DMA) unit in TriCore derivatives. Based on the ETM, a follow-up of the program and data sequences, also for ARM-based micro-controllers, is guaranteed.

The UAD 2+ is equipped with isolated interfaces and JTAG extender technology. By means of common JTAG

debug interfaces, CAN or other serial interfaces, it enables access of the UDE 2.0 to the target flexibly. Tools can be also integrated in a 100-Mbit/s Ethernet network for fast access. The assignment of the necessary IP address is flexible via DHC protocol, but also possible manually. The support for CAN, within the test environment, has also been expanded further. The integrated CAN recorder now allows the clear-text display of CANopen messages. Particularly for the debugging of TriCore and XC16x micro-controllers, the tool

additionally provides users with target monitor software, which uses the respective on-chip debug system. Debugging in the Flash memory and the use of data breakpoints – also via CAN – is possible.

With CMX from CMX Systems and μ C/OS-II from Micrium Technologies, the tool supports two real-time operating systems. The HTML-based pages for these enable the representation of operating system states, in table and graphic form, dependent on the run-time of the application.

The customizable user interface features a workbook mode, a symbol browser, customizable toolbars, property pages, context-related menus, adjustable fonts and colors, a user-adaptable appearance of the standard windows as well as HTML as description language for user-specific windows. Available are the docking containers, which, by means of addressed tabs, allow any of the windows to be arranged and exchanged. (mm)