

Automatisierter Test von verteilten Systemen basierend auf CAN oder MOST

Umfassende Tests in kürzerer Zeit bringen reproduzierbare Resultate

Die Vielfalt der heutigen Telematik- und Multimediaumgebung in Kraftfahrzeugen kann nur durch verteilte Systeme implementiert werden, die über einen Bus miteinander kommunizieren. Derzeit ist der MOST-Bus das am weitesten verbreitete Konzept für eine schnelle ringförmige Vernetzung von Multimedia-Endgeräten. MOST steht für ‚Media Oriented Systems Transport‘ und ist heute die von vielen Herstellern favorisierte Lösung für die Vernetzung von Audio- und Telematikkomponenten. Das MOST-Bussystem zeichnet sich aus durch hohe Bandbreite für synchrone Übertragung, relative niedrige Produktionskosten und geringen Protokoll-Overhead.

Im Bereich der ‚Body Electronics‘ gibt es ebenfalls eine Vielzahl von Einzelanwendungen, die zum Teil unabhängig voneinander sind, häufig aber auch interagieren, wie z.B. Hochfahren der Fensterscheiben und Schließen des Schiebedachs bei Betätigung der Zentralverriegelung. In diesem Bereich hat sich der CAN-Bus (Controller Area Network) als Vernetzungsmedium für die Steuergeräte etabliert.

Sowohl die Infotainment- als auch die Body-Electronics-Funktionen werden durch eine Vielzahl von Steuergeräten realisiert. Die einzelnen Komponenten interagieren miteinander: Befehle, Meldungen und Daten werden ausgetauscht.

Die Kommunikation der Steuergeräte über einen Bus erfordert die Implementierung der

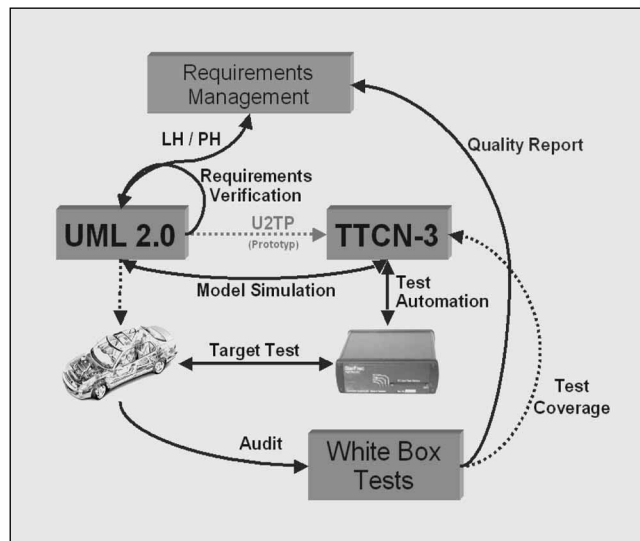


Abb. 1: Ein Framework für Design, Implementierung und Test

entsprechenden Protokolle. Der Modultest eines einzelnen Steuergerätes als Subsystem auf dem Entwicklungsrechner kann in der Regel nicht feststellen, ob die Protokollimplementierung korrekt ist, und ob alle Realzeitbedingungen sicher eingehalten werden. Gerade Fehler, die durch Echtzeitabhängigkeiten zwischen zwei Bussystemen entstehen (z.B. MOST- und CAN-Interaktionen bei MOST- Gateways), sind ohne Echtzeittest sehr schwierig zu lokalisieren. Solche Fehler werden erst sehr spät im Entwicklungsprozess festgestellt, wenn die Korrektur sehr teuer ist.

Der traditionelle Ansatz

Man versucht, durch den Einsatz geeigneter Testwerkzeuge auf dem Entwicklungsrechner die Anforderungen an Logik und Echtzeitverhalten des Kommunikationsprotokolls und der Anwendung auf dem Steuergerät zu simulieren und deren Einhaltung zu überprüfen. So erreicht man ein gewisses Maß an Sicherheit, dass sich das Steuergerät im Verbund mit den anderen richtig verhält. Große Probleme treten jedoch häufig bei der Integration der Komponenten auf, wenn alle Subsysteme erstmalig interagieren müssen. Grundsätzliche

Autoren

RENATE STÜCKA ist Director of Marketing, Segment Automotive Telelogic Deutschland;
Otto-Brenner-Straße 247, D-33604 Bielefeld
Fon: 0521/14503-254, Fax: 0521/14503-50
E-Mail: renaete.stuecka@telelogic.de

FREDRIK MATTSON ist Business Area Manager Combitech Systems AB;
Graftiv 23C, SE-461 38 Trollhättan, Schweden
Fon: +46/708 895 086, Fax: +46/520 364 70
E-Mail: fredrik.mattson@combitechsystems.com

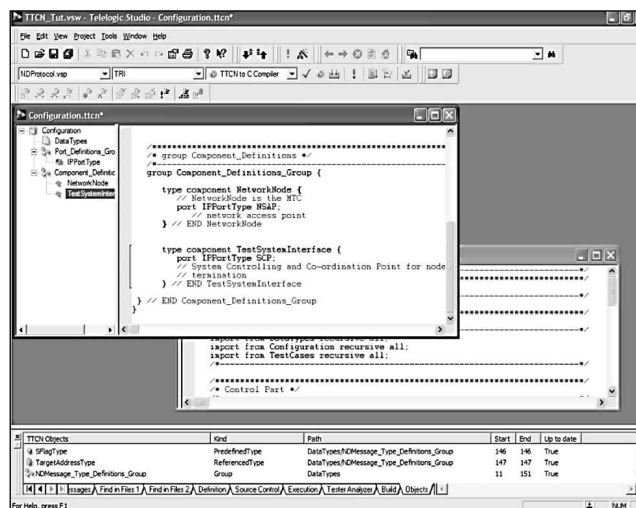


Abb. 2: Die Syntax der TTCN-3 ist vergleichbar einer modernen Programmiersprache

konzeptionelle Mängel in der Systemarchitektur werden oft erst bei der Integration erkannt.

Bereits existierende HIL-Testkonzepte (hardware-in-the-loop), welche helfen können, schwerwiegende Fehler früh zu entdecken, sind oft nicht wirklich echtzeitfähig und nicht in der Lage, systematische Protokolltests durchzuführen.

Zudem müssen die Tests immer wieder neu individuell für jedes Steuergerät entwickelt werden, der Grad der Wiederverwendung ist sehr gering.

Ein Vorschlag für eine neue Lösung

Die Aufwendungen für Test können sich auf 50 Prozent oder gar mehr der gesamten Projektkosten summieren. Vor diesem Hintergrund erscheint es einleuchtend, diesen quantitativ erheblichen Anteil der Aufwendungen nicht als separate, angehängte Folgeaktivität der Entwicklung aufzufassen, sondern als integrierten Bestandteil eines Entwicklungsprozesses. Mit den neuen Standards UML 2.0 (Unified Modeling Language, ein Standard der Object Management Group OMG) und TTCN-3 (Testing and Test Control Notation, ETSI) ist es möglich, ein durchgängiges Framework für Design, Implementierung und Test von MOST- oder CAN-Applikationen zu realisieren (Abb. 1).

Der Prozess beginnt mit dem Anforderungsmanagement, der Sammlung, Bewertung und Auswahl der Anforderungen, die das zu entwickelnde System erfüllen soll. Strukturiertes und konsequentes Anforderungsmanagement mit durchgängiger Verfolgung jeder einzelnen Anforderung bis hin zur Detailimplementierung und zum Test ist eine der wichtigsten Voraussetzungen für den Erfolg von Entwicklungsprojekten. Hierfür empfiehlt sich der Einsatz eines leistungsfähigen Werkzeuges wie z.B. ‚Telelogic Doors‘. Als Ergebnis der Sammlung und Auswahl von Anforderungen entstehen in der Regel Pflichten- und Lastenhefte, die den angestrebten Leistungsumfang verbal und mit Hilfe ergänzender graphischer Darstellungen beschreiben.

Weitere Möglichkeiten der Beschreibung eröffnet an dieser Stelle UML 2.0, eine formale Sprache, die gegenüber den Vorgängerversionen vielfältige Erweiterungen bietet, z.B. bei der Beschreibung von Verhalten, parallelen Abläufen oder Systemarchitekturen. Mit Hilfe dieser Sprache können die verbal beschriebenen Funktionen modelliert werden. Diese Modelle stellen eine eindeutige Beschreibung dar, die von jedem in gleicher Weise verstanden wird. Missverständnisse durch

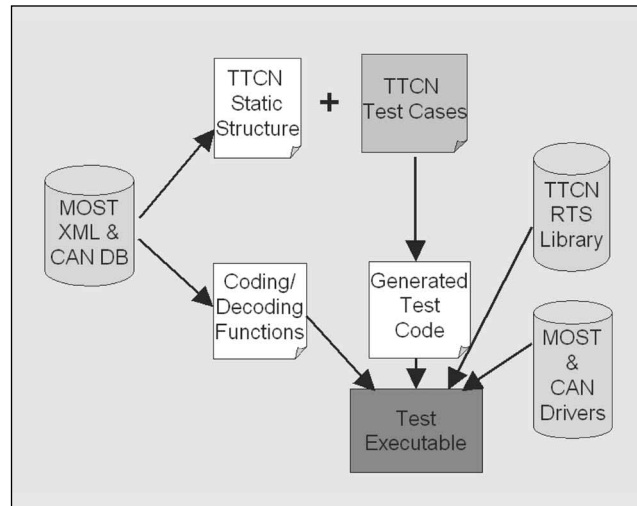


Abb. 3:
Der Weg zur ausführbaren Beschreibung der Testsequenzen

unterschiedliche Auslegung von Anforderungen werden ausgeschlossen. Darüber hinaus sind diese Modelle ausführbar, das Verhalten der modellierten Funktionen kann simuliert werden. So ist eine frühzeitige Verifikation und Validierung der Anforderungen möglich. Widersprüchliche Beschreibungen, Lücken oder nicht sinnvolle Anforderungen werden erkannt und können korrigiert werden, bevor sie sich durch mehrere Phasen des Entwicklungsprozesses hindurch manifestiert haben.

Das zu entwickelnde System kann mit UML 2.0 nahezu vollständig beschrieben und simuliert werden. Das Modell ist nicht nur eine formale Beschreibung des Systems, es ist darüber hinaus auch ein virtueller Prototyp, der z.B. für Akzeptanztests oder Integrations-tests eingesetzt werden kann. Nachdem das Modell fertig ist, kann daraus der Code für die Zielplattform automatisch erzeugt werden.

Die UML 2.0 bietet eine weitere signifikante Neuerung: das Testing Profile, das auf der TTCN-3 basiert. Dieser Standard für Testbeschreibungssprachen wurde vor knapp zwei Jahren verabschiedet. Die Vorgängerversion TTCN wurde ursprünglich entwickelt, um Testsequenzen zur Überprüfung der Implementierung von Protokollen im Rahmen des OSI-Referenzmodells eindeutig und ohne Bezug auf die interne Realisierung zu beschreiben. TTCN ist standardisiert (ISO 9646) und damit herstellerunabhängig.

TTCN-3 ist nicht einfach eine neue Version des Standards, es handelt sich in weiten Teilen um eine neue Sprache, die von der ETSI (ES 201873-1) entwickelt wurde und inzwischen den Status einer ITU Empfehlung hat (Z.140). Die Motivation zur Entwicklung von TTCN-3 und die ihr zugrundeliegende Philosophie ist, die in Telekommunikationsanwendungen bewährten Stärken der TTCN-2 zu erhalten und mit Hilfe einer neuen erweiterten Notation auch für die vielfältigen Anwendungen anderer Industrien nutzbar zu ma-

chen. Mit TTCN-3 steht heute eine flexible Sprache zur Verfügung für die Spezifikation vieler Arten von Systemtests für ein breites Spektrum von Anwendungen. Die Syntax der TTCN-3 ist vergleichbar der einer konventionellen Programmiersprache und beinhaltet unter anderem auch synchrone und asynchrone Kommunikationsmechanismen sowie die Möglichkeit, auch dynamische parallele Tests zu definieren (Abb. 2).

Das Testing Profile erlaubt die automatische Generierung von TTCN-3-Testbeschreibungen aus dem UML-Modell. Diese können je nach Anwendungsfall durch manuell erstellte TTCN-3-Testsequenzen ergänzt werden. Für die Erstellung, Bearbeitung und Ausführung von TTCN-3-Testfällen ist ein entsprechendes Werkzeug erforderlich, welches das Potential dieser Beschreibungssprache nutzbar macht, wie z.B. Telelogic Tau/Tester. Aus den TTCN-3-Testbeschreibungen wird anschließend der C-Code erzeugt, der mit Hilfe eines entsprechenden Cross Compilers für das Zielsystem übersetzt wird. Dabei handelt es sich um Prüfsystem, welches in die MOST- oder CAN-Umgebung eingefügt wird und das zu testende Subsystem automatisch allen vorher definierten Tests unterzieht. Ein Prüfsystem, das die hier beschriebene Funktionalität bietet, ist StarTest von Combitech Systems.

Ein besonderer Vorteil dieser Lösung ist die Echtzeitfähigkeit: Diese HIL-Testumgebung erlaubt die Prüfung von Echtzeitanforderungen mit einer Genauigkeit von bis zu 100 µs und ist damit präziser als typische Simulationsverfahren. Die Prüfhardware kann sowohl mit dem PC als auch Standalone eingesetzt werden, so dass alle Testsequenzen unter realen Bedingungen im laufenden Betrieb im Fahrzeug oder im Rahmen von Produktionstests ausgeführt werden können.

Die Testergebnisse werden vollständig aufgezeichnet und können mit verschiedenen

gängigen Werkzeugen, wie z.B. MSC-Editor, OptoLyzer oder CANalyzer visualisiert und ausgewertet werden.

Die StarTest-Hardware-Plattform besteht aus einem PowerPC-Prozessor zusammen mit einer MOST- und zwei CAN-Hardware-schnittstellen. Für Download- und Tracing-Aufgaben steht eine Ethernet-Schnittstelle zur Verfügung. Im eCOS-Echtzeitbetriebssystem auf dem Prozessor sind MOST- und CAN-Treiber integriert, sowie ein PC-Card-Dateisystem für die Speicherung der Testergebnisse.

StarTest wird wie folgt eingesetzt (Abb. 3) :

- ▶ TTCN-Signal und Datentyp-Definitionen werden aus bestehenden MOST- und CAN-Signal-Datenbanken erzeugt. Die Verwendung der existierenden Datenbanken reduziert den Aufwand und insbesondere die Fehleranfälligkeit manueller Definitionen. Die statischen TTCN-Definitionen sind auf jeden Fall konsistent mit den Busspezifikationen.
- ▶ Funktionen zur Codierung und Decodierung der abstrakten TTCN-Datentypen werden generiert. Aufgabe dieser Funktionen ist die Umsetzung von abstrakten Datentypen (z.B. String) in die Bit- oder Byte-Sequenz, die tatsächlich über den Bus gesendet werden soll, sowie die Transformation in umgekehrter Richtung.
- ▶ TTCN-Testfälle werden nach C übersetzt. Der TTCN-Compiler des Tau/Tester erzeugt aus den abstrakten Testfallbeschreibungen C-Funktionen, die auf einer Bibliothek von Support-Funktionen aufsetzen (z.B. Threading, Message-Boxes,

etc.). Diese Bibliothek (TTCN Runtime-System oder RTS) muss auf die Hardwareplattform portiert werden und gegebenenfalls mit dem dort vorhandenen Echtzeitbetriebssystem integriert werden.

- ▶ Ein ausführbare Datei wird erzeugt. Dazu wird der aus den Testfallbeschreibungen generierte und compilierte C-Code zusammen mit der TTCN-RTS-Bibliothek und den MOST-, bzw. CAN-Treibern zu einem Executable gelinkt.
- ▶ Die ausführbaren Testfälle werden per Ethernet auf den StarTest heruntergeladen und dort ausgeführt. Die Testergebnisse werden als Log-Dateien (textuell oder als Sequenz-Diagramme) auf der PC-Card gespeichert.

Ergänzende ‚White Box‘-Tests geben Aufschluss über weitere qualitätsrelevante Merkmale. Diese Tests, die z.B. von ‚Telelogic Tau Logiscope‘ durchgeführt werden, sind insbesondere dann zu empfehlen, wenn die individuelle Aufgabenstellung und die Rahmenbedingungen des Projektes neben automatisch generiertem Quellcode und Testsequenzen auch die Eingliederung manuell erstellter Module erfordern.

Bei der ergänzenden Nutzung manuell erstellter Testsequenzen ist die Überprüfung der Testabdeckung ein wichtiges Mittel um festzustellen, ob mit den angelegten Testfällen tatsächlich jeder Pfad und jedes Statement im Source Code mindestens einmal ausgeführt werden. Anhand der ermittelten Werte der Testabdeckung lassen sich die Testfälle ergänzen und optimieren, bis eine 100-prozentige Testabdeckung erreicht ist.

Zur Überprüfung von manuell programmiertem Quellcode sind Softwaremetriken besonders geeignet, mit deren Hilfe besonders fehleranfällige Module, unzulässige Sprünge oder rekursive Aufrufe oder auch sogenannter ‚dead code‘ schnell und zuverlässig identifiziert werden können. Die ermittelten Werte der Metriken werden mit Hilfe eines vorgegebenen oder individuell festgelegten Qualitätsmodell bewertet und gehen als Qualitätsbericht wieder ein in das Anforderungsmanagement. Damit ist der Kreis geschlossen, und die Basis für eine kontinuierliche und objektiv überprüfbare Verbesserung der Qualität für die weiteren iterativen Entwicklungen ist geschaffen.

Literatur

- [1] The Evolution of TTCN, ITU 2001, <http://www.itu.int/ITU-T/studygroups/com17/ttcn.html>
- [2] ITU Recommendation Z.140, Tree and Tabular Combined Notation Version 3, 2001
- [3] Johan Nordin, Telelogic: Meeting the Demands of Software Testing, 2002
- [4] Telelogic White Paper: TTCN-3 The Ultimate Test
- [5] Renate Stuecka, Bridging the Gap is not Enough - Lifecycle Management for Automotive Electronics and Software, in Business Briefing: Global Automotive Manufacturing & Technology 2003
- [6] Renate Stuecka, Automatisierter Test in der Softwareentwicklung, TEST KOMPENDIUM 2003

Beitrag als PDF im Internet:

www.duv24.net

more @ click TK4B0702



LESETIPP

? **Sie interessieren sich für Informationen in Anzeigenmotiven?**

**Die Firmenreferenzliste (Griffmarke D.05)
listet alle in dieser Ausgabe vertretenen Inserenten mit
entsprechenden Seitenverweisen auf**

KOMPENDIUM
Messen • Prüfen • Verifizieren

publish industry
TECHNIK KOMMUNIZIEREN

Gollierstraße 23 · 80339 München, Germany · Fon +49/89/500383-0 · Fax +49/89/500383-10 · info@publish-industry.net · www.publish-industry.net