Absicherung von Steuerungssoftware für Hybridsysteme

- Automatisierte Methode zur Testfallgenerierung

Dr.-Ing. **Martin Neumann**, Dipl.-Ing. Mario Nass, Dipl.-Ing. Carsten Paulus, Fa. ZF Friedrichshafen AG, Friedrichshafen;
Dr. Mugur Tatar, Fa. QTronic GmbH, Berlin

Kurzfassung

Konventionelle Prozesse für die Entwicklung von Fahrzeugsteuerungen stützen sich heute auf modellbasierte Funktionsentwicklung, Softwareabsicherung auf Hardware-in-the-Loop (HiL) Simulatoren mittels Testskripten und Systembewertung und Applikation im Fahrversuch. Durch die Hybridisierung des Antriebstrangs ist der Grad der funktionalen Vernetzung von Aggregaten sprunghaft angestiegen. Hybridspezifische Funktionen sind auf unterschiedlichen Steuergeräten verteilt und werden unabhängig voneinander entwickelt. Eine große Herausforderung stellt dabei die Integration dieser neuen Hybridfunktionen in die Softwarearchitektur vorhandener Steuergeräte, sowie in den Hybridverbund mit neuen Steuergeräten dar. Für den Entwickler stellen sich dabei folgende Fragen:

- Welche Entwicklungs- und Testmethoden sind notwendig, um diese stark gestiegene Komplexität beherrschbar zu machen und
- wie kann die notwendige Testabdeckung trotz wachsendem Funktionsumfang im gegebenen Zeitfenster sichergestellt werden?

Dieser Beitrag diskutiert diese Fragen im Kontext der Entwicklung von Steuerungen für Hybridantriebe. Zentrale Aspekte sind dabei:

- Verlagerung von Testumfängen von HiL-Simulator und Fahrversuch in eine PC gestützte Entwicklungsumgebung mit Hilfe einer vernetzten Software-in-the-Loop (VSiL) Simulation. Dies fördert eine verblockungsfreie, parallele Entwicklung in einer vergleichsweise schnell und preiswert duplizierbaren Testumgebung.
- Systematische Modellierung der Funktions- und Softwareanforderungen in Form von Systeminvarianten. Im Gegensatz zur Programmierung statischer Testskripte ermöglicht diese Art von Modellierung eine Überprüfung der Anforderungen im gesamten Zustandsraum des Systems.
- Automatisierte Methode zur Testfallgenerierung, -Ausführung und -Bewertung. Diese sichert eine gewünschte hohe Testabdeckung bei vergleichsweise geringem Arbeitsaufwand.

1. Einleitung

Durch zusätzliche Komponenten wie E-Maschine, Wechselrichter, Energiespeicher und elektrische Nebenaggregate nimmt die Komplexität der funktionalen Vernetzung im Hybrid-Antriebsstrang gegenüber dem konventionellen Antriebsstrang um ein Vielfaches zu. Dies liegt daran, dass einerseits die Funktionen der etablierten Steuergeräte erweitert werden und andererseits zusätzlich neue Funktionalitäten im Zusammenspiel mit weiteren Steuergeräten im Verbund realisiert werden. Durch letzteres ergeben sich zahlreiche Abhängigkeiten zwischen früher isolierten und abgegrenzten Steuergerätefunktionen, so dass bei der Hybridentwicklung ein komplexes Gesamtsystem mit verteilten Funktionen zu realisieren ist. Im Bild 1 ist die Systemvernetzung eines typischen Parallelhybrid-Antriebsstrangs abgebildet. Um diese Komplexität in Serie erfolgreich zu beherrschen, setzt ZF bei der Entwicklung von Steuerungen für Hybridantriebe seit einigen Jahren eine Plattform aus modellbasierter Funktionssoftware mit konventioneller Basissoftware im gesamten Softwareentwicklungsprozess ein. Die hybrid-spezifischen Funktionen werden in das Funktionsnetzwerk bestehender Basisantriebsstränge für Pkw- und Nfz-Systeme integriert. Je nach Fahrzeugvariante sind die hybrid-spezifischen Funktionen sowohl auf eigenständigen Steuergeräten, als auch auf den Steuergeräten mit der konventionellen Software untergebracht.

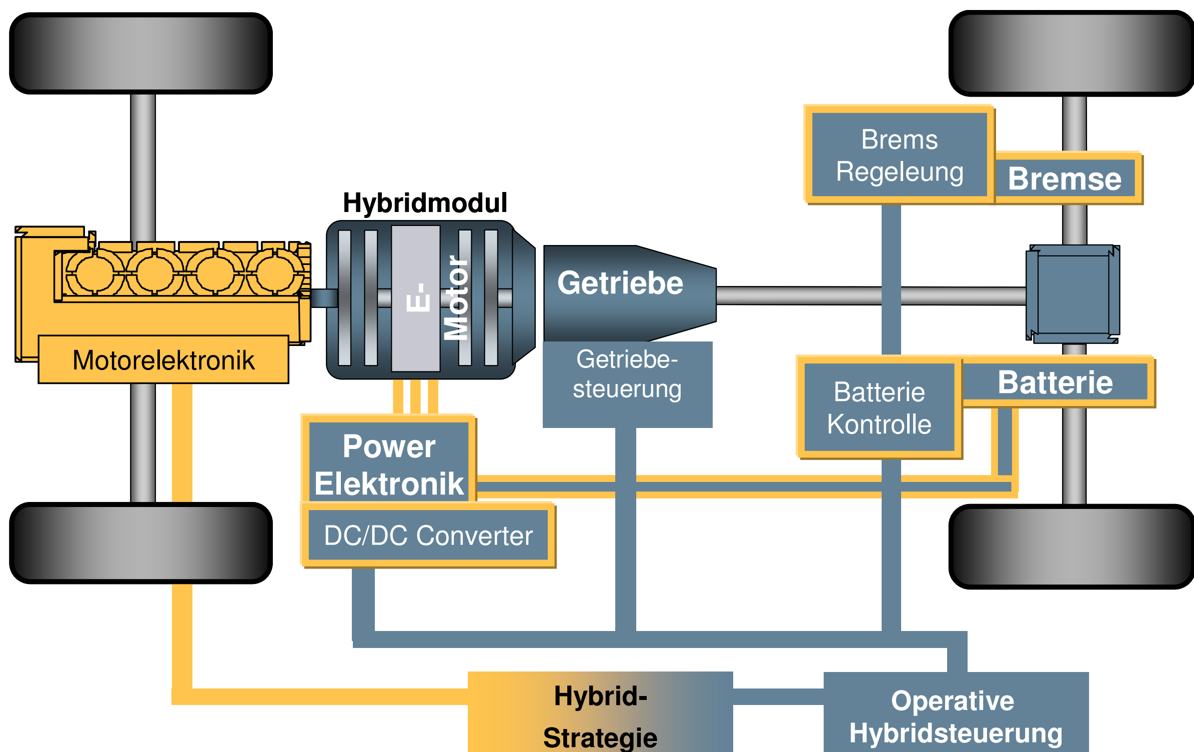

Bild 1: Antriebsstrang Parallelhybrid mit Funktions- und Steuergerätenetzwerk

2. Virtuelle Integration vernetzter Steuerungssoftware auf dem PC (VSiL)

Um Entwicklungszeiten zu verkürzen wird heute die Entwicklung mechanischer und elektronischer Komponenten parallelisiert. Dies gilt gleichermaßen für die Funktions- und Softwareentwicklung innerhalb der einzelnen Steuergeräte. Üblicherweise wird die Software eines Steuergerätes in separaten Teams entwickelt, getestet und anschließend integriert (Entwicklungszyklus klassisch). Die funktionale Integration mehrerer Steuergeräte findet i.d.R. erst im Fahrzeug, am Prüfstand, bzw. am VHiL auf der Zielhardware der realen Steuergeräte statt. Fehler, die dabei gefunden werden haben zeitaufwändige Rekursionsschleifen zur Folge. Mit der hier vorgestellten Methodik ist es nun möglich originale Steuergerätesoftware vernetzt auf einem PC zu testen, ohne dazu die realen Steuergeräte benutzen zu müssen. Die Steuergerätesoftware, wie z.B. Getriebe- und Hybridsteuerungen werden zusammen mit Verhaltensmodellen (Fahrer, Strecke, Fahrzeug, Soft-ECU's, Restbus) in einer „**V**ernetzten **S**oftware-**i**n-the-**L**oop Simulation" (VSiL) mit dem ZF eigenen Co-Simulationswerkzeug Softcar [1] simuliert. Je nach Entwicklungsschwerpunkt werden modellbasierte Teilfunktionalitäten, bzw. auch die Verhaltensmodelle, aus dem Systemverbund unter Softcar freigeschnitten und innerhalb Simulink/TargetLink Co-simuliert. Damit steht dem Entwickler, neben der vollständigen Analysierbarkeit des Quellcodes in Softcar, auch die vollständige Entwicklungsumgebung auf Modellebene in Simulink/TargetLink zur Verfügung. Dies ermöglicht eine komfortable und kostengünstige Simulationsplattform für Analyse, Debugging und Test des Systemverhaltens auf dem PC des Funktions-, Softwareentwicklers und Testingenieurs. Durch eine frühe Integration von Softwareständen in den Systemverbund („Frontloading") ist das Hybrid-Teilsystem bereits zu einem frühen Entwicklungszeitpunkt transparent verfügbar. Eine Basisabstimmung, sowie die Beurteilung des Zusammenspiels der verteilten Hybridfunktionalität werden damit im virtuellen Systemverbund auf dem PC möglich.

Im Bild 2 ist die Verbundsimulation VSiL am Beispiel einer Hybridanwendung dargestellt. Durch Umschalten von Compilerschaltern im Buildprozess werden die C-Quellen der Hybrid-Steuerungssoftware wahlweise als binary (*.hex) für die Zielprozessoren der Steuergeräte oder wie hier dargestellt als ausführbare Prozesse (*.exe) für die PC-Simulation mit Softcar übersetzt. Auf diese Weise kann der Softwareentwickler seine Änderungen sehr schnell im integrierten Gesamtsystem auf dem PC testen. Die Erstellung der Hybridsoftware (compile und build) dauert dabei nur wenige Minuten. Im Software-in-the-Loop Simulationsverbund stehen dann sämtliche Variablen der Getriebe- und Hybridsoftware, sowie die Variablen des Simulationsmodells zur Verfügung.

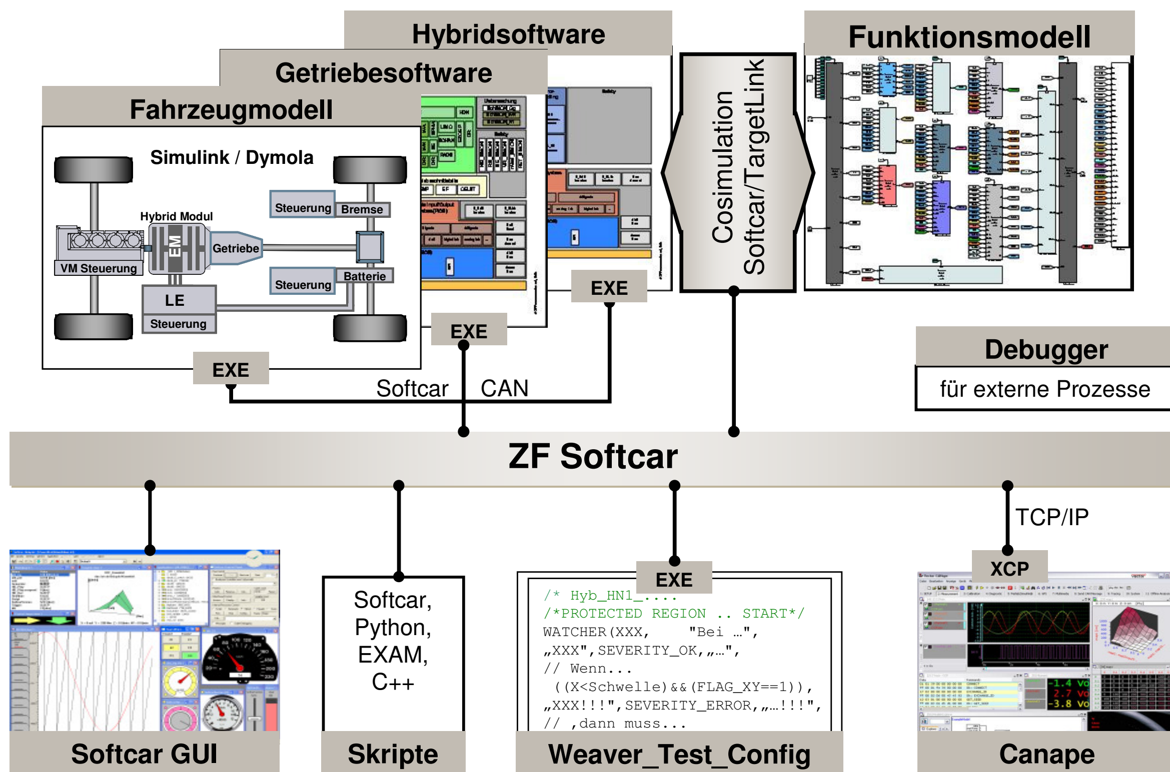

Bild 2: Systemintegration über „Vernetzte-Software-in-the-Loop Simulation" (VSiL)

Die Simulationen können zu jedem Zeitpunkt angehalten werden, um diese Variablen mit dem VisualStudio Debugger zu untersuchen, bzw. zu modifizieren. Die Software- und Simulationsmodellprozesse (.exe) sind über eine virtuelle CAN-Simulation miteinander verbunden und werden von einer Prozesssteuerung in Softcar koordiniert. Das ermöglicht eine Abstimmung der Schnittstellen und eine Validierung der CAN-Konfiguration. Damit ist sichergestellt, dass ein möglichst realitätsnahes Verhalten bereits auf der VSiL-Umgebung vorhanden ist. Zusätzlich können alle internen statischen Variablen der Softcarprozesse eingesehen und ggf. durch Applikation verändert werden.

Das Programm ZF-Softcar stellt ein graphisches User-Interface (GUI) zur Verfügung. Hierüber erfolgt die Interaktion der automatisierten Testumgebung bzw. des Benutzers bei manuellem Betrieb z.B. durch Verstellung von Zündung, Gaspedal, Bremse, etc. mit dem Simulationsverbund. Darüber hinaus können sämtliche Ein- und Ausgangssignale beteiligter Prozesse direkt in einer Fehlersimulation (Sensorausfälle, Gangspringer, Übertemperatur, Fehlerstrategien, Schutzfunktionen, etc.) getestet und manipuliert werden. ZF-Softcar verfügt bereits standardmäßig über umfangreiche Anzeige-, Applikations-, Mess- und Steuerungsfenster. Darüber hinaus sind weitere Schnittstellen zu kommerziellen Mess- und Applikationswerkzeugen (z.B. ETAS-INCA, Vector-CANape) mit dem XCP-Protokoll via TCP/IP über

Ethernet vorhanden. Damit ist das Messen wie am Prüfstand, HiL oder Fahrzeug mit identischer Messkonfiguration gegeben. Die Testfallerstellung und Testautomatisierung (z.B. Initialisierung Hybridsystem, Motorstart, Hybridabläufe, Schaltabläufe) erfolgt mit einer einfachen Softcar-Skriptsprache. Alternativ können Testfallbibliotheken in C/C++, Python und Visual Basic erstellt werden, die auch per EXAM generiert werden können. Zur Messung der Code Coverage kommt Testwell - CTC++ zum Einsatz. Die Schnittstelle zwischen Softcar und dem Testfallgenerator TestWeaver wird über einen externen Softcar-Prozess realisiert (weaver_test_config.exe).

3. Automatisierte Methode zur Testfallgenerierung

Zur Absicherung der Steuerungssoftware im VSiL wird TestWeaver, ein neuartiger Testgenerator von QTronic eingesetzt [2], [3]. TestWeaver steuert und beobachtet autonom das Verhalten eines Systems mittels Simulation und versucht selbstständig das System in viele verschiedene Betriebszustände zu fahren und dabei Bugs und Schwachstellen aufzudecken und zu protokollieren. Tausende Testszenarien mit unterschiedlichen Fahrmanövern werden von TestWeaver automatisch entwickelt und ausgeführt. Das dynamische Systemverhalten wird dabei kontinuierlich in einer Zustandsdatenbank registriert, anhand von Abdeckungszielen klassifiziert, und bezüglich Korrektheits- und Qualitätskriterien bewertet. Intelligente

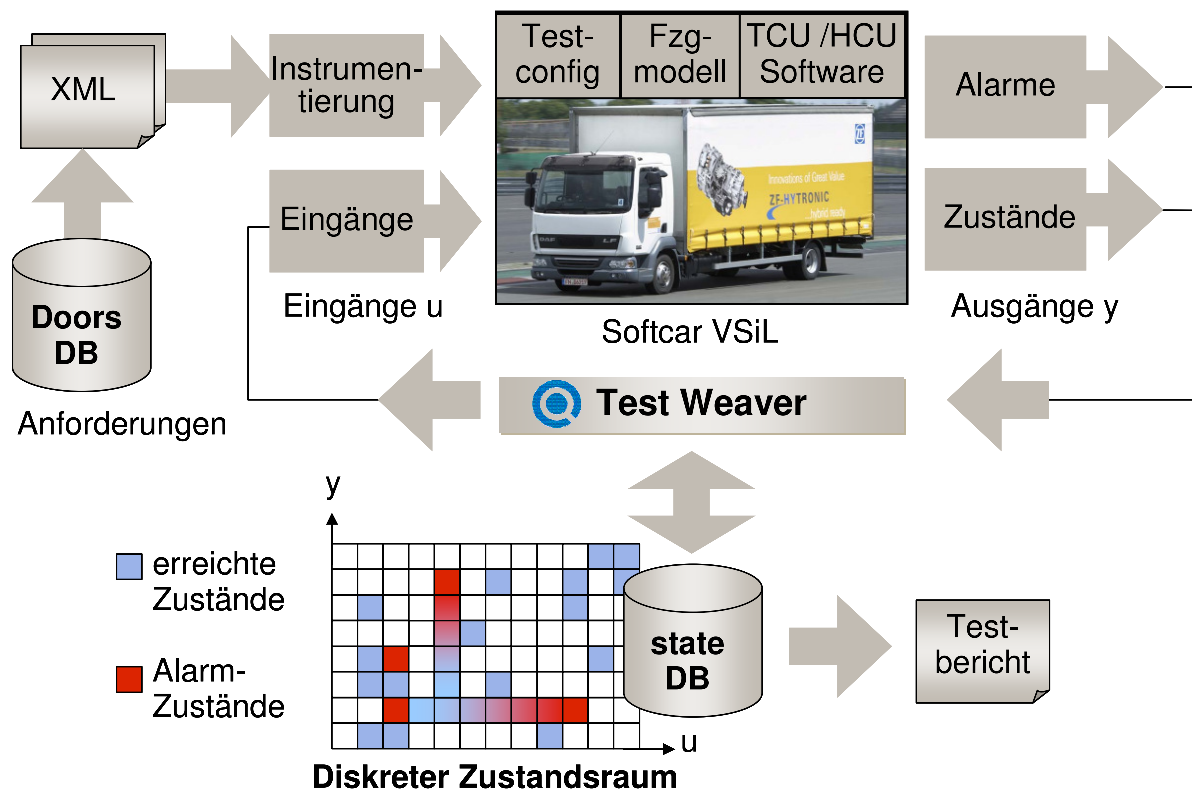

Bild 3: Konfiguration der Softcar – Testweaver Testumgebung

Suchalgorithmen ermöglichen TestWeaver autonom das Systemverhalten zu „erforschen" und ein abstraktes Zustandsraummodell zu bauen (vgl. Bild 3). Dieses wird von TestWeaver benutzt um reaktiv neue Szenarien zu konstruieren, die (a) das System in weitere neue Zustände führen, dabei die Zustandsabdeckung erhöhen und (b) Verletzungen der Bewertungskriterien aufdecken. Alle erreichten Systemzustände, ob korrekte oder fehlerhafte, können später in der Simulation wiederholt und im Detail analysiert werden (Replay-Funktion). Interessante Szenarien können für Regressionszwecke gespeichert werden. Die Korrektheits- und Qualitätskriterien sind oft als Beobachter in der Simulation realisiert, die die Simulation ständig überwachen und Verletzungen der Prüfbedingungen über Alarmsignale an TestWeaver melden. Vorteile dieser Absicherungsmethode sind:

- Die Korrektheits- und Qualitätskriterien werden ständig überwacht, d.h. nicht nur am Ende von vordefinierten Szenarien, sondern überall im Zustandsraum. Das ermöglicht eine fundierte Korrektheitsaussage und eine Qualitätsbewertung.
- Die Methode ist auch für sehr komplexe Systeme anwendbar, die mehrere Steuergeräte und detaillierte Fahrzeugmodelle einschließen. Die Quellen von Software und Simulationsmodellen können verborgen bleiben.
- Die Methode kann auch „verdeckte" Fehler und Schwachstellen finden, anhand von Szenarien an die kein Testingenieur vorher gedacht hat.
- Hohe Testabdeckung, hoher Automatisierungsgrad, vergleichbar geringer Spezifikationsaufwand.

Der Testraum wird anhand einer kompakten Konfiguration festgelegt:

1. Steuerbare Eingangsvariablen (Eingänge u) und Klassifikation der Eingangswerte auf einer gut-schlecht Skala, z.B. für die Analyse von Sensor- und Aktuatorfehlern (ein „schlechter" Inputwert aktiviert einen Komponentenfehler).
2. Beobachtbare Ausgangsvariablen (Ausgänge y) und Klassifikation in gute und schlechte Intervalle (ein „schlechter" Ausgangswert signalisiert unerwünschtes Verhalten).
3. Allgemeine Daten für die Konfiguration des Testgenerators, z.B. die Länge von Szenarien, Definition von Zwangsbedingungen (z.B. kein Gas und Bremse gleichzeitig), Abdeckungsziele in Form von „Coveragetabellen". Optional: Regressionsszenarien und hand-definierte Szenarien, die unabhängig von der autonomen Generierung betrachtet werden sollen.
4. Vorlagen zur Dokumentation der erreichten Testabdeckung im Zustandsraum, Codeabdeckung und weiterer Testergebnisse.

Typische Anwendungen für Systemtests definieren circa 20-30 steuerbare Eingangsvariablen, circa 20-30 Ausgangsvariablen, die für die Bewertung der Zustandsabdeckung wichtig sind, sowie 10-100 (oder mehr) Ausgangsvariablen, die Korrektheits- und Qualitätskriterien messen. Die Simulation läuft autonom, zum Beispiel über Nacht oder am Wochenende, auf einem oder mehreren Rechnern. Die Testergebnisse werden anschließend von einem Testingenieur bewertet, der gegebenenfalls Verbesserungsmaßnahmen einleitet. TestWeaver kann an verschiedene Simulationsplattformen angebunden werden, z.B.: MiL mit Simulink, SiL/VSiL mit Softcar oder Silver [2],[3], und HiL mit dSPACE-Systemen.

4. Konfiguration TestWeaver für Softcar VSiL

Die Schnittstelle zwischen TestWeaver und der Systemsimulation wird über einen zusätzlichen externen Prozess für Softcar realisiert und enthält folgende Funktionen:

- Verbindung zwischen Softcar und TestWeaver über TCP/IP herstellen.
- Referenzierung der Variablen (Simulationsmodell, Hybrid- und Getriebesoftware).
- Instrumentierung der Eingangsvariablen (Chooser-Instrumente).
- Instrumentierung der Ausgangsvariablen für ausgewählte Funktionen (Reporter-Instrumente).
- Implementierung der System- und Softwareanforderungen (Watcher-Instrumente).

Die projektspezifische Konfiguration der TestWeaver-Instrumente wird als C++ Code angebunden. Die Instrumentierung der Eingangsvariablen legt fest, welche Signale TestWeaver während eines Szenarios dynamisch variieren soll und welche Werte dafür zur Verfügung stehen, z.B.:

- Gaspedal, Bremspedal: 0%, 25%, 50%, 75%, 100%,
- Parkbremse: aus, ein,
- Fahrbahnsteigung: -5%, 0%, 5 % 10%,
- Fahrschalter: P, R, N, D,
- Starttemperatur Verbrennungsmotor: kalt, warm,
- Anfangsladezustand der Hybridbatterie: niedrig, mittel, voll,
- ggf. Hardwarefehler, wie z.B. defekte Reibbeläge, Ventil- oder Sensorfehler u.a..

Zusätzlich werden mit TestWeaver zyklische Softcar-Ereignisse dokumentiert. Dazu zählen

- Runtime exceptions: Division durch null, Zugriffsverletzungen, Überläufe, usw.,
- Code coverage: Die Codeabdeckung wird mit CTC++ von Testwell gemessen. Hierzu wird die Hybridsoftware mit einem speziellen Flag übersetzt. Die Darstellung der Codeabdeckung erfolgt durch einen TestWeaver-Report.

```
PARTITION(SoC)
            {{{0,20}, " 0..20%", SEVERITY_ERROR, ""},
            {{21,50}, "21..50%", SEVERITY_LOW, ""},
            {{51,80}, "51..80%", SEVERITY_OK, ""},
            {{81,90}, "81..90%", SEVERITY_HIGH, ""},
            {{91,100},"91..100%",SEVERITY_ERROR, ""}};

REPORT(SoC,SOC,"%","Ladezustand der Hybridbatterie");
```

Bild 4: Reporter-Implementierung – Auszug aus der Datei *weaver_test_config.cpp*.

Die Instrumentierung der Ausgangsvariablen erfolgt durch eine Zuordnung ausgewählter Systemvariablen (z.B. der Ladezustand der Hybridbatterie) zu Korrektheits- und Qualitätskriterien von OK bis Error (vgl. Bild 4). TestWeaver versucht durch gezielte Variation der Eingangsvariablen Fahrmanöver zu finden, in denen diese Korrektheitskriterien verletzt werden, z.B. das Hybridsystem in einen Zustand zu bringen, in dem die spezifizierten Grenzen der Batterieladung über- bzw. unterschritten werden.

5. Überprüfung der Software- und Systemanforderungen

Im vorliegenden Hybridprojekt werden die Software- und Systemanforderungen in DOORS verwaltet. Bei herkömmlichen Testmethoden wird zu jeder Anforderung ein passendes Testskript von einem Testingenieur entwickelt. Dies hat den Nachteil, dass die Anforderungen nur punktuell, in wenigen speziellen Fahrmanövern (Szenarien) geprüft werden, obwohl sie einen allgemeinen Charakter haben und in großen Teilen des Systemzustandsraums gültig sind. Eine weitere Schwäche des skriptbasierten Testens besteht bei der Formulierung negativer Testbedingungen (z.B.: „Der Ladezustand der Batterie darf nicht unter eine Schwelle fallen“). Solche Prüfbedingungen können nicht mit einzelnen Testszenarien assoziiert werden. Um den oben genannten Schwachstellen entgegenzuwirken, werden bei dem hier vorgestellten Ansatz die Anforderungen als Systeminvarianten modelliert und melden ständig (also für jedes Szenario) ihre Einhaltung oder Verletzung. Die Implementierung der Software- und Systemanforderungen erfolgt durch sogenannte Watcher-Instrumente (vgl. Bild 5).

```
/* Hyb_HN1_SYS_07_01-1556:  Bei einem niedrigen SOC …
   Hyb_HN1_SYS_07_01-1369:  Wenn die Energie im Speicher unterhalb …
   Hyb_HN1_HCU_FSW_09_01-497: Wenn der SOC unter einer apllizierbaren Schwelle …
/*PROTECTED REGION ID(CheckChargeStandstill) ENABLED START*/
WATCHER(CheckChargeStandstill, "Bei einem niedrigen SOC …","Wenn Energie < …",
"ChargeStandstill„, SEVERITY_OK,"EM lädt die Batterie nicht!!!",
// Wenn …
"ChargeStandstill!!!",SEVERITY _ERROR,"EM lädt die Batterie nicht!!!",
// dann muss …
 ((HCU_SiEMot_M_trq_filt<=0) && (HCU_HSML_M_ctrlStrategicMode_req==4)),0.5);
/*PROTECTED REGION END*/
```

Bild 5: Watcher-Implementierung – Auszug aus der Datei *weaver_test_config.cpp*.

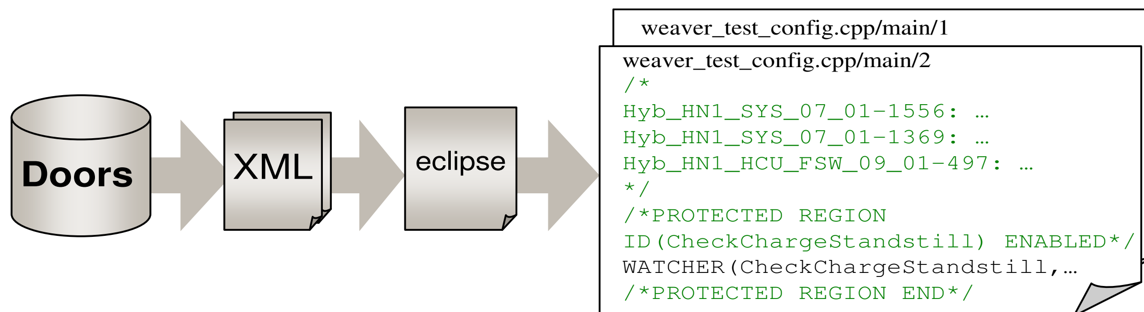

Bild 6: Werkzeugkette zur Herstellung der Traceability zwischen DOORS und TestWeaver.

Die Watcher-Instrumente beinhalten die konkrete Implementierung dieser DOORS-Anforderungen nach folgendem Schema:
"Immer wenn <Vorbedingung>, dann folgt <Nachbedingung> nach maximalem <Delay>".
Um die Vollständigkeit und die Konsistenz der Implementierung gegenüber der DOORS-Spezifikation zu sichern (Traceability), wird die in Bild 6 skizzierte Werkzeugkette verwendet. Zwischen den Watcher-Instrumenten und den Anforderungen besteht eine 1 zu n Beziehung. Dies bedeutet, dass ein Watcher-Instrument mehrere Anforderungen prüfen kann. Um diesem Umstand Rechnung zu tragen, wird in DOORS ein Text-Attribut eingeführt. Zu jeder Anforderungs-ID, die ein Watcher prüfen soll, wird der Watcher-Name in das Attribut eingetragen. Dadurch ist in DOORS gekennzeichnet, welche Anforderungen mit TestWeaver verlinkt sind. Diese Anforderungen werden in ein XML-Format exportiert, welches unter Verwendung der Modell-zu-Text-Transformations-Sprache XPAND aus dem Eclipse Modeling Project in die *weaver_test_config.cpp* transformiert wird. Die Transformation generiert vor jedes Weaver-Instrument ein Kommentarfeld mit zugehörigem DOORS-Anforderungstext. Die Implementierung der Watcherbedingung ist in einer nachfolgenden Protected-Region enthalten. Bei Änderungen in der Anforderungsspezifikation, muss auch der DOORS-Export neu ausgeführt werden. Die eigentlichen Implementierungen innerhalb der Protected-Regions werden bei erneutem Generatordurchlauf nicht mehr überschrieben. Mittels Diff-Tool im Versionsmanagement kann nun leicht geprüft werden, ob die textuelle Beschreibung der Anforderungen in der Datei *weaver_test_config.cpp* verändert wurde oder Anforderungen neu hinzugekommen, bzw. weggefallen sind. Neue Anforderungen werden als leere Protected-Region angelegt, in die der Testingenieur dann die Implementierung des Watchers vornehmen kann.

6. Testergebnisse

In diesem Abschnitt werden exemplarisch Softcar-Testweaver Testergebnisse diskutiert. Im Bild 7 sind die Antriebstrangzustände des Hybridsystems dargestellt. Ziel der Tests ist eine

Bild 7: Antriebstrangzustände des Hybridsystems

möglichst hohe Testabdeckung der Software zu erreichen, d.h. alle Hybridzustände und Zustandsübergänge zwischen den Zuständen zu durchlaufen und dabei Bugs und Schwachstellen der Software aufzudecken. Das Abdeckungsziel wird im TestWeaver in Form von Coverage-Bedingungen vorgegeben. In einer Coveragebedingung werden lediglich die Softwarevariablen inkl. einer Partition des Wertebereichs z.B. DrivelineMode_current (S1,…,S8), DrivelineMode_target (S1,…,S8), OperatingMode (Op1,…OpX) definiert. Testweaver ver-

Driveline Mode		Operating Mode	OUTPUT
current	requested	requested	scenarios
S0_WeaverDefault	S0_WeaverDefault	OpDefault	[s0, s49]
S1_Coupled	S1_Coupled	OpCharge	[s966, s993]
	S2_Decoupled	OpNeutralDecouple	[s966, s993]
	S8_HybridDrive	OpCharge	[s0, s1]
S2_Decoupled	S2_Decoupled	OpTransmissionCommand	[s361, s571]
		OpDrivelineOpen	[s990, s1000]
	S5_Charge	OpImpulseStart	[s974, s966]
	S6_DecoupledE	OpNeutralDecouple	[s974, s966]
. . .			. . .
		OpHybridDrive	[s361, s217]
S5_Charge	S6_DecoupledE	OpNeutralDecouple	[s0, s1]
	S7_EDriveEngOn	OpDrivelineOpen	[s0, s1]
	S8_HybridDrive	OpTransmissionCommand	[s11, s12]
		OpCharge	[s0, s1]
S6_DecoupledE	S6_DecoupledE	OpDrivelineOpen	[s0, s1]
	S8_HybridDrive	OpTransmissionCommand	[s11, s0]
		OpDrivelineOpen	[s0, s1]
S7_EDriveEngOn	S7_EDriveEngOn	OpElectricDrive	[s677, s673]
	S8_HybridDrive	OpEngCouple	[s992, s1070]
		OpTransmissionCommand	[s0, s1]
		OpElectricDrive	[s0, s1]
S8_HybridDrive	S8_HybridDrive	OpTransmissionCommand	[s0, s3]
		OpHybridDrive	[s4, s0]

Bild 8: Codeabdeckung für Antriebstrangzustände und Operative Funktionen

sucht nun durch intelligente Generierung neuer Szenarien (Variation der Eingangssignale) möglichst alle Kombinationen zwischen Ist- und Zielzuständen mit allen Operativen Funktionen (Zustandsübergänge) zu erreichen. Während der Durchführung der Testläufe wird das Ergebnis der erreichten Abdeckung automatisch von TestWeaver in einem Coverage-Report dokumentiert (vgl. Bild 8). Der Testingenieur kann so leicht überprüfen, ob alle spezifizierten Zustände und Operativen Funktionen für die jeweilige Fahrzeugvariante erreicht wurden.

Die Anforderungen aus der Softwarespezifikation, die als Watcher-Instrumente implementiert sind, werden zyklisch jede 1ms ausgewertet. Folgende Arten der Überwachung (ca. 100 Weaver-Instrumente) wurden implementiert:

- Überwachung des Fehlerspeichers (Funktions- und Safety-Layer),
- Überwachung der Software- und Systemanforderungen,
- Überwachung von Wertebereichsüberschreitungen, Assertions, Resets, Prozessabstürze,
- Überwachung der Dynamik von Zustandsänderungen (z.B. Ankoppeln des Verbrennungsmotors dauert zu lange),
- Überwachung von „toggelnden“ von Zustandsvariablen (Bits, Modi), etc..

Desweiteren können Spezifikationslücken aufgedeckt werden, z.B. wenn durch TestWeaver ungewöhnliche Fahrsituationen angefahren werden, die bisher nicht spezifiziert sind. Als

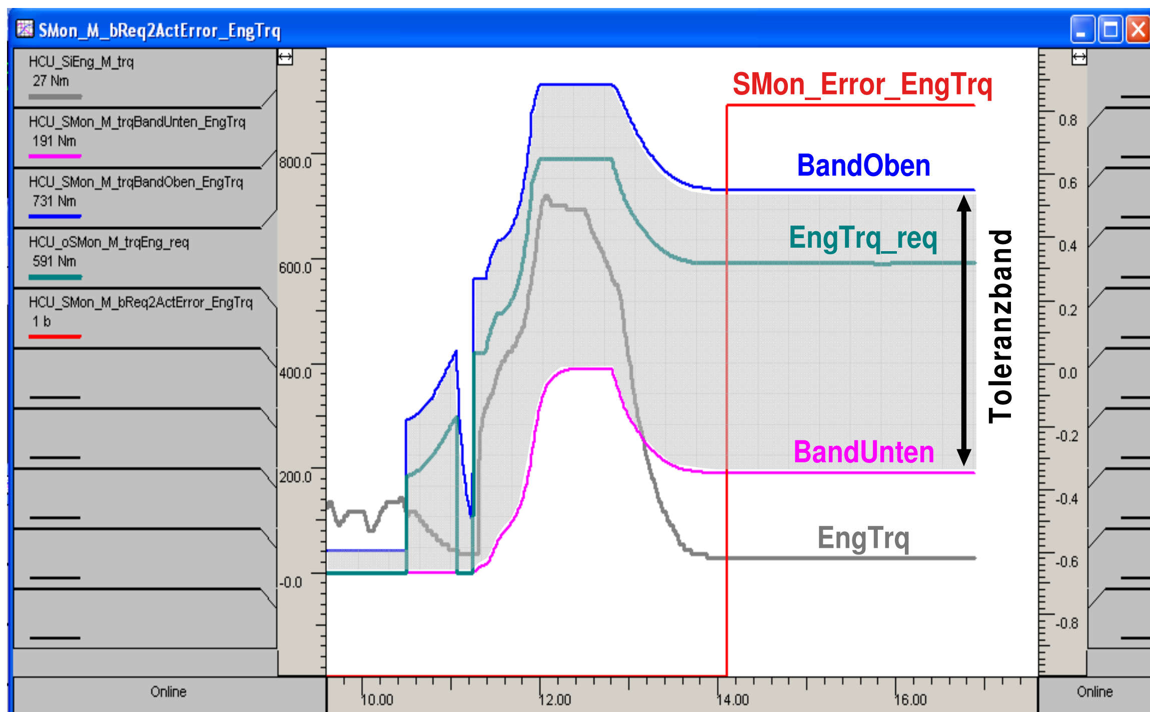

Bild 9: Safety-Monitor – Abweichung Soll-Istmoment des Verbrennungsmotors

Beispiel sei hier eine fehlerhaft applizierte Sicherheitsfunktion genannt. Dort ist ein Toleranzband für das Verbrennungsmotormoment hinterlegt (vgl. Bild 9). Bei stationärer Rückwärtsfahrt mit Vollgas, geht der Verbrennungsmotor bei maximaler Drehzahl in den Abregelmodus und kann den Fahrerwunsch (EngTrq_req) nicht mehr realisieren. Dabei läuft das Verbrennungsmotormoment (EngTrq) aus dem Toleranzband, was fälschlicherweise zu einem Fehlereintrag (SMon_Error_EngTrq) führt, obwohl das System korrekt funktioniert.

7. Zusammenfassung

Der Innovationsgrad der vorgelegten Arbeit ist sehr hoch. Der große Vorteil des Tests mit TestWeaver liegt in der hohen und messbaren Testabdeckung bei gleichzeitig geringem Arbeitsaufwand für die Erstellung der Testspezifikation. Eine vergleichbar hohe Testabdeckung ist mit dem herkömmlichen Ansatz über handgeschriebene Testskripte wegen des hohen Arbeitsaufwandes praktisch nicht erreichbar.

Literatur

[1] Bieber, Gillich, Neumann, Paulus, Welt: *Systematische Absicherung von Steuerungssoftware für Hybridsysteme bei ZF,* 4. IAV Tagung Simulation und Test für die Automobilelektronik, 30.05. – 01.06.2010, Berlin, Germany

[2] Tatar, Schaich, Breitinger: *Automated test of the AMG Speedshift DCT control software*, 9th International CTI Symposium Innovative Automotive Transmissions, Berlin, 30.11. - 01.12.2010, Berlin, Germany

[3] Hilf, Matheis, Mauss, Rauh: Automated simulation of scenarios to guide the development of a crosswind stabilization function. IFAC Symposium Advances in Automotive Control 2010, 12. -14.07.2010, Munich, Germany.