

Test von adaptiven Softwaremechanismen zur Fehlerkompensation

Olaf Maibaum, Sven Rebeschies

Der Beitrag stellt den im Verbundvorhaben SiLEST* (Software in the Loop for Embedded Software Test) verfolgten Ansatz des durch eine Umgebungssimulation unterstützten Softwaretests vor. Besonderes Ziel des Testansatzes ist es, hinsichtlich der Umgebungssimulation einen hohen Grad der Wiederverwendbarkeit von Simulationsmodulen und eine weitgehende Testautomatisierung zu gewährleisten. Von besonderem Interesse bei dem Testansatz ist dabei der Test des Softwareverhaltens unter gestörten Betriebsbedingungen, wie es beispielsweise in einem gealterten System oder bei beschädigter Sensorik bzw. Aktuatorik der Fall ist.

1. Einleitung

Die Umweltbedingungen im Automobil sind für die eingesetzte Fahrzeugelektronik nicht optimal. Hitze- und Kältezyklen, Nässe und Korrosion, Erschütterungen und Verschmutzung sowie mechanische Abnutzung führen zur Alterung der verwendeten Sensorik und Aktuatorik. Die in Software ausgelegte Regelung der Fahrzeugkomponenten muss diesem Umstand Rechnung tragen, indem sie adaptive Mechanismen einsetzt, um sich den im Laufe der Zeit veränderten Randbedingungen anzupassen. Solche adaptiven Mechanismen können beispielsweise die Anpassung von Filtern für Sensordaten oder die Rekonfiguration von redundanten Sensornetzen sein, wie sie im Hinblick auf zukünftige X-by-Wire-Systeme unerlässlich sein werden und für On-board-Systeme in der Raumfahrt zur Verlängerung der Lebenszeit üblich sind. Ziel dieser adaptiven Softwaremechanismen sind die Gewährleistung des sicheren Betriebs von älteren Fahrzeugen oder ein Notfallbetrieb des Fahrzeugs bei Ausfall von sicherheitskritischen Systemen.

Der Test dieser adaptiven Softwaremechanismen stellt aufgrund der Vielzahl von Alterungs- und Ausfallerscheinungen von Sensorik bzw. Aktuatorik im Fahrzeug und der hohen Komplexität der Sensornetze eine große Herausforderung dar. Im Rahmen des durch das BMBF geförderten Verbundvorhabens SiLEST soll unter anderem mit Hilfe einer „Software in the Loop“-Simulation (SiL-Simulation) das Softwareverhalten in einem gealterten System für die Anwendungsdomänen Automobil und Raumfahrt getestet werden. Wir definieren den Begriff „Software in the Loop“-Test entsprechend der Definition in [1], dass der Test der realen Software einschließlich aller Restriktionen hinsichtlich der Ressourcen in einer simulierten Umgebung oder mit experimenteller Hardware erfolgt. Dieser Testansatz wird teilweise auch als „Processor in the Loop“ bezeichnet.

In dem folgenden Abschnitt wird zunächst der Begriff der adaptiven Softwaremechanismen zur Fehlerkompensation beschrieben. Im dritten Abschnitt wird der Schich-

* Das diesem Begriff zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01ISC12A gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Autor.

tenaufbau der Simulationsumgebung und der Aufbau der Simulationsmodule mit Hilfe der Unified Modeling Language (UML) vorgestellt. Der vierte Abschnitt wird die Durchführung der Tests der Regelsoftware im Regelkreis vorstellen. Dieser gliedert sich in die Auswahl der Testfälle anhand der FMEA und der Beschreibung von Ausfallszenarien mit einem Ereignisbaum, dem Aufbau der Testumgebung und der Durchführung der Testfälle. Im letzten Abschnitt werden wir noch einmal eine Zusammenfassung und einen Ausblick auf die laufenden Arbeiten im Verbundvorhaben SiLEST geben.

2. Adaptive Softwaremechanismen zur Fehlerkompensation

Software eingebetteter sicherheitskritischer Systeme muss autonom und adäquat auf ein mögliches Fehlverhalten der zu regelnden Hardware reagieren können. Hierzu sind adaptive Softwaremechanismen notwendig, welche ein Fehlverhalten oder eine Störung erkennen und behandeln können, um den weiteren Betrieb des Systems zu gewährleisten, gegebenenfalls auch mit einem suboptimalen Verhalten.

Solche Mechanismen sind Standard in der Raumfahrt, da in der Regel direkte menschliche Eingriffe zur Behebung der Fehlerursache nicht möglich sind. Der Schlüssel hierzu ist die Schaffung von Redundanzen. Dabei ist zu unterscheiden zwischen kalter und warmer Redundanz. Unter kalter Redundanz ist Hardware zu verstehen, welche mitgeführt aber nur in Betrieb genommen wird, wenn die primäre Hardware versagt. Der Zweck der kalten Redundanz liegt in der Lebenszeitverlängerung von Sonden und Satelliten oder der Sicherstellung eines Notfallbetriebs. Warm redundante Hardware hingegen ist ebenfalls im Betrieb und dient der Überwachung der primären Systeme und der schnellen Übernahme eines Notfallbetriebs bei einem Fehlverhalten der primären Hardware.

Für sicherheitskritische Systeme sind im Automobilbereich die warm redundanten Mechanismen von Interesse. Kalte Redundanz wird durch den jederzeit möglichen Austausch von Bauteilen im Automobil nicht benötigt. Die Mechanismen zur Ansteuerung der warm redundanten Hardware werden immer häufiger in Software implementiert.

In der Regel kommen zur Umsetzung der warmen Redundanz von Sensorik oder Aktuatorik Voter zum Einsatz, welche eine Mehrheitsentscheidung aus mehreren zur Verfügung stehenden Sensorwerten treffen. Für eine Mehrheitsentscheidung sind immer mindestens drei Sensorwerte notwendig. Falls ein Sensor nur doppelt ausgelegt wurde, so muss die Entscheidung auf Basis einer Plausibilitätsüberprüfung getroffen werden. Hierzu wird in der Regel eine Schätzung der zu erwartenden Sensorwerte verwendet, welche sich aus dem vorherigen Zustand zuzüglich der erwarteten Änderung ergibt. Die zu erwartende Änderung setzt sich zusammen aus der Änderung ohne Regelaktivität und dem Effekt der letzten Regelaktivität.

Mit Hilfe der Zustandsschätzung lässt sich auch ein Aktuatordefekt erkennen. In diesem Fall fehlt allen gemessenen Sensorwerten der Effekt des defekten Aktuators. Bei einem System mit mehreren Aktuatoren fällt es allerdings schwer, den defekten Aktuator zu bestimmen, um ihn für den weiteren Betrieb aus der Regelung heraus zu nehmen. Hierzu müsste im laufenden Betrieb der Anteil der Aktuatoren an der Regelung variiert werden, um aus der anteiligen zu messenden Störung Rückschlüsse auf den defekten Aktuator ziehen zu können.

Für alle Maßnahmen bei einem erkannten Defekt gilt, dass die Software das System in einen sicheren Zustand bringen muss. Je nach Defekt kann dies der weitere Be-

trieb mit einer eingeschränkten Funktionalität sein oder aber auch die Abschaltung des Systems, sofern alle anderen Maßnahmen ein nicht tragbares Sicherheitsrisiko darstellen.

Neben Maßnahmen zur Erkennung von plötzlichen Ausfällen gibt es auch adaptive Verfahren zur Kompensation von Alterungserscheinungen, wie ein schlechteres Signal-Rausch-Verhältnis, die Abnahme der Sensorensitivität oder eine verringerte Wirkung oder Genauigkeit von Aktuatoren. Um diesen schleichenden Veränderungen zu begegnen, lassen sich die Filter den Veränderungen der Hardware anpassen. Das System würde dann kein optimales Verhalten mehr zeigen, dennoch ließe es sich bis zur nächsten Wartung störungsfrei betreiben.

3. Aufbau der Simulationsumgebung

Der Aufbau der Simulation erfolgt gemäß dem in Bild 1 gegebenen allgemein gültigen Schichtenmodell. Das Schichtenmodell besteht aus insgesamt drei Ebenen. Die unterste Ebene ist das eingebettete System. Sie besteht aus der zu testenden Regelungssoftware, dem Betriebssystem (OS) bzw. der Hardware-Abstraktion (HAL) sowie der elektrischen Anbindung der ECU. Die nächste Ebene ist die der Sensorik und Aktuatorik, welche ebenfalls in drei Schichten aufgesplittet ist. Dies sind die elektrische Anbindung und das Kommunikationsprotokoll zur ECU hin, die interne Sensor- oder Aktuator-Prozessierung sowie eine Schicht zur Simulation der physikalischen Sensormessung bzw. der physikalischen Aktuatoraktivität. Die oberste Ebene besteht aus den Schichten Einbaulage, der Speicherung des physikalischen Einflusses oder Wirkung sowie der eigentlichen Simulation der physikalischen Umwelt wie beispielsweise der Simulation der Fahrzeug-Längsdynamik, des Fahrers und der Umgebung. Die Schicht der Einbaulage dient der Konvertierung von Daten aus dem lokalen Bezugssystem eines Sensors oder Aktuators in ein globales Bezugssystem, in dem die Simulation der physikalischen Umwelt erfolgt. Die Schicht „physikalischer Einfluss/Wirkung“ dient der Datenhaltung und dem Austausch von physikalischen Daten zwischen den einzelnen Simulationsmodulen.

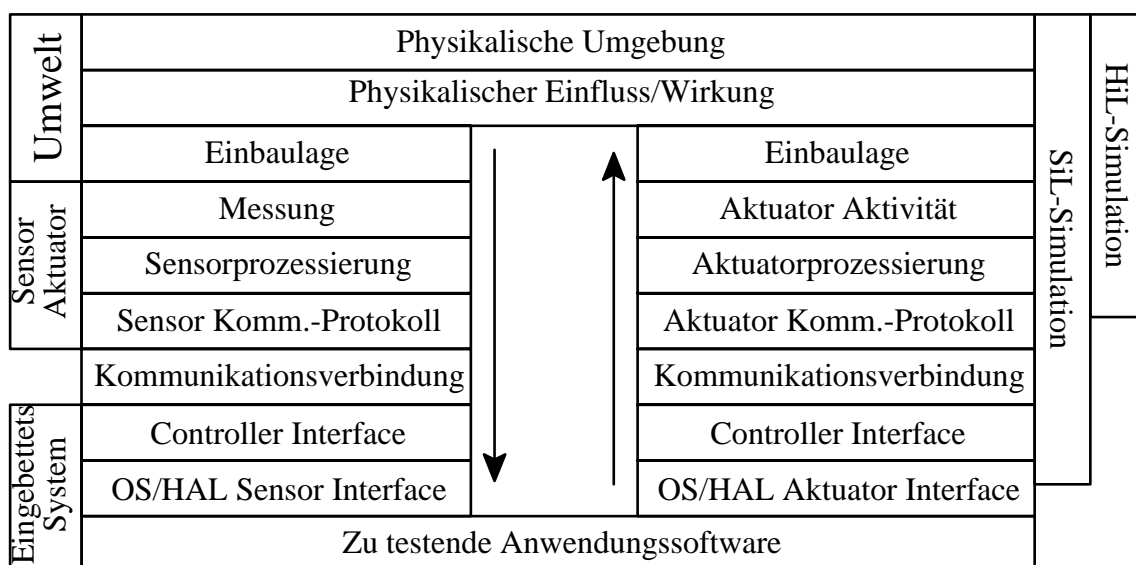


Bild 1: Schichtenstruktur eines eingebetteten Systems

Bild 1 zeigt am rechten Rand angetragen den Anteil der Simulation für die HiL- und SiL-Simulation. Beim Test mit einer HiL-Simulation erfolgt die Kopplung zwischen Umgebungssimulation und dem eingebetteten System durch die elektrischen Schnittstellen auf der Schicht der Sensor- und Aktuator-Kommunikationsprotokolle. Bei der SiL-Simulation dagegen erfolgt die Kopplung durch eine angepasste OS/HAL-Schnittstelle. Durch das Schichtenmodell kann so für den HiL- und SiL-Test die gleiche Simulation verwendet werden. Die einzigen Unterschiede zwischen beiden Testansätzen bestehen lediglich in der Ankopplung der Simulation an die zu testende Regelungssoftware.

Allerdings würde durch die Ankopplung der Simulation auf Betriebssystemebene bei einem SiL-Test ohne eine zusätzliche Synchronisierung zwischen Simulation und der zu testenden Software das Laufzeitverhalten der zu testenden Regelung geändert werden. Die Folge wäre eine mögliche Maskierung von Fehlern bei der Testdurchführung, was vermieden werden sollte. Zu diesem Zweck wird die Regelungssoftware in einem künstlichen Zeitrahmen ausgeführt. Dies heißt, alle Zugriffe der zu testenden Regelungssoftware auf Schnittstellen zur Kommunikation mit der Umwelt und zum Ermitteln der Uhrzeit werden durch Schnittstellen auf der Betriebssystemebene ersetzt, damit die zu testende Regelungssoftware nur einen künstlichen Zeitrahmen sehen kann.

Durch diese Art der Kopplung beim SiL-Test wird es möglich, die Regelungssoftware und die Simulation in einem konsistenten Zustand zu stoppen. Hierdurch werden sowohl ein Debugging als auch die Sicherung eines konsistenten Betriebszustands als Initialwert für spätere Tests möglich.

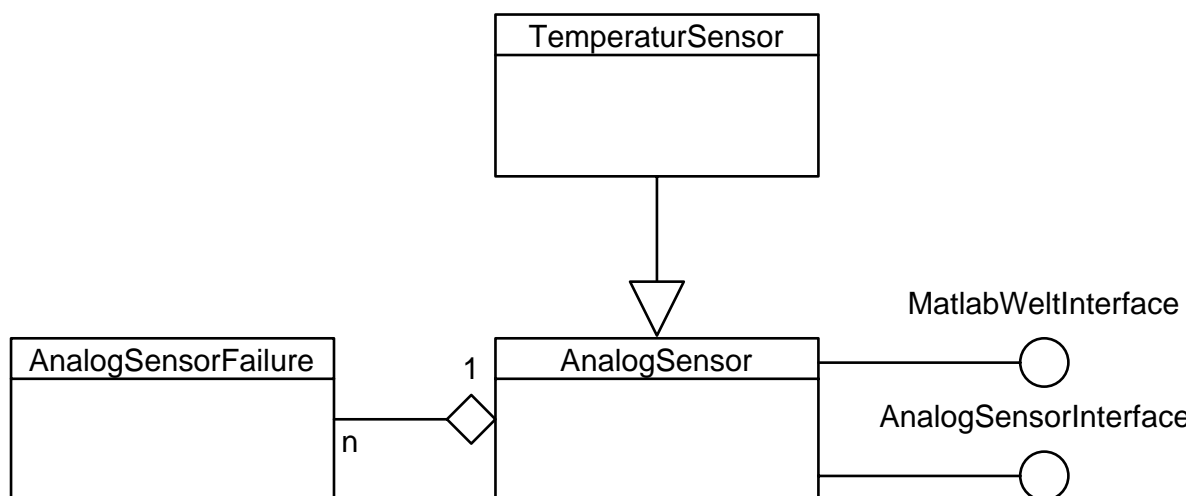


Bild 2: UML-Klassendiagramm eines Sensors in der Umgebungssimulation

Der Aufbau der Umgebungssimulation kann entweder mittels Simulink oder durch die UML [2,3] beschrieben werden. Bild 2 zeigt den prinzipiellen objektorientierten Aufbau eines Temperatursensors in einem UML-Diagramm. Die Klasse eines Temperatursensors ist eine Spezialisierung eines Anlogsensors und ererbt dadurch auch alle Fehlerfälle, welche ein Anlogsensor haben kann. Ein Fehlerfall eines Anlogsensors kann z. B. ein Rauschen auf der Signalleitung oder eine lose Steckverbindung zwischen Sensor und ECU sein. Fehlerfälle würden dann als Spezialisierung eines AnalogSensorFailures implementiert werden. Der AnalogSensor selber besitzt Zugang zu dem AnalogSensorInterface, über das die zu testende Regelungssoftware mit der

Simulation gekoppelt wird. Für einen HiL-Test ist dies die Schnittstelle zu einem dSPACE-Interface oder für einen SiL-Test die Kopplungskomponente mit dem Betriebssystem der Regelungssoftware. Als weitere Schnittstelle besitzt jede Simulationskomponente noch ein MatlabWeltInterface, mit dem die Simulationskomponenten untereinander verbunden werden. Das MatlabWeltInterface ist im Schichtenmodell der Simulation der Einbaulage-Schicht zugeordnet. Bei der Instantiierung eines Temperatursensors hat dieser dem MatlabWeltInterface die Einbaulage des Sensors selber und den globalen Variablennamen des Temperaturwerts in der Umgebungssimulation mitzuteilen, um den Zugriff auf die entsprechende Variable zu ermöglichen. Die Klassendiagramme eines einzelnen Sensors oder Aktuators werden durch eine Komponente gekapselt. Die einzelnen Komponenten einer Umgebungssimulation werden dann durch ein UML-Komponentendiagramm zusammengeführt.

4. Durchführung der Tests

4.1 Auswahl der Testfälle

Neben dem Verhalten im störungsfreien Betrieb soll die Regelungssoftware auch im gestörten Betrieb getestet werden. Unter einem gestörten Betrieb sind Störungen an oder der Ausfall der das eingebettete System umgebenden Hardware gemeint. Ein Testfall setzt sich zusammen aus einem Betriebsszenario und einem Versagensszenario.

Ein Betriebsszenario beschreibt alle externen Stimuli, die auf das zu testende Gesamtsystem wirken. Dies entspricht zum Beispiel dem Profil der Gaspedalstellung oder der zeitlichen Änderung der Last durch Änderung der Fahrbahnbeschaffenheit. Ein Versagensszenario beschreibt den zeitlichen Ablauf einer Störung in der umgebenden Hardware. Dies kann das langsame Auftreten eines Rauschens sein oder aber die plötzliche Unterbrechung einer Steckerverbindung.

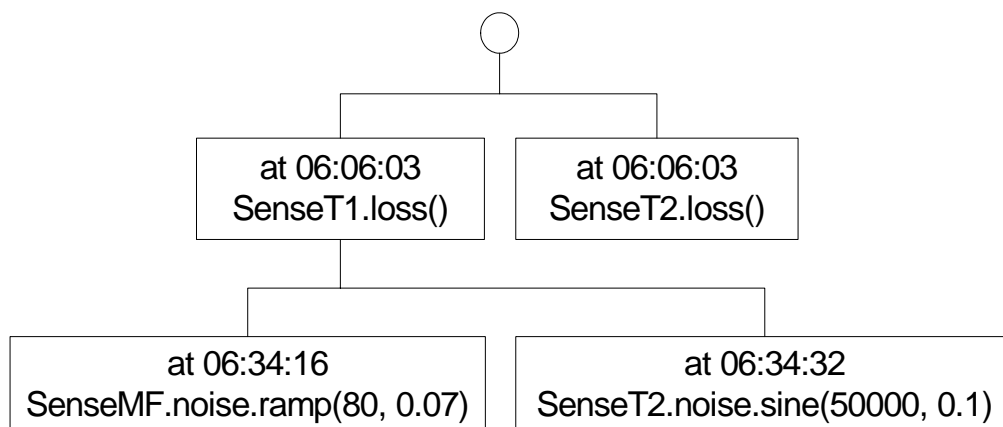


Bild 3: Ereignisbaum

Als Grundlage für die Auswahl der Versagensszenarien dienen die Ergebnisse der für das System durchgeführten FMEA [1]. Entsprechend der geforderten Testtiefe lassen sich aus der FMEA alle Sensoren und Aktuatoren bestimmen, von deren Fehlverhalten eine Gefährdung ausgeht. Als Maß für die von einem Sensor oder Ak-

tuator ausgehende Gefährdung wird das Risiko verwendet. Das Risiko wird berechnet aus der Eintrittswahrscheinlichkeit eines Defekts, einer Kostenfunktion für die Folgen eines Defekts und der Wahrscheinlichkeit für die Erkennung des Defekts.

Mindestens für die Sensoren und Aktuatoren mit einem hohen oder nicht vertretbaren Risiko sind mit Hilfe des für sie bekannten Ausfall- oder Störverhaltens verschiedene Versagensszenarien zu entwerfen. Die Versagensszenarien beinhalten dabei das Ausfall- und Störverhalten einzeln oder in Kombination.

Die zeitliche Abfolge von unterschiedlichen Störereignissen lässt sich mit Hilfe eines Ereignisbaumes beschreiben. Bild 3 zeigt einen Ereignisbaum mit insgesamt drei verschiedenen Abfolgen von Ausfällen. Dies ist der Ausfall von Sensor T1 mit einem später langsam ansteigenden Rauschen im Signal von Sensor MF oder einem späteren Rauschen auf Sensor T2, dessen Amplitude sinusförmig schwingt, oder der Ausfall des Sensors T2 alleine.

Die durch einen Ereignisbaum beschriebenen Abfolgen von Störereignissen sind mit den Betriebsszenarien zu verknüpfen. Zur Komplettierung der Testfälle sind die Grenzen eines akzeptablen Verhaltens des Systems bzw. das Systemverhalten für den Testfall zu definieren. Die Grenzen eines akzeptablen Verhaltens sind die Vorgaben, in denen sich das System laut Spezifikation bewegen darf, also beispielsweise maximale Drehzahl- oder Beschleunigungswerte. Das Systemverhalten ist die laut Spezifikation zu erwartende Reaktion des Testobjektes im Betriebsszenario sowie im Fehlerfall.

Für die automatische Testdurchführung haben die Betriebsszenarien, die Störereignisse und die Spezifikation des Verhaltens in einer maschinenlesbaren Form vorzuliegen, welche durch die Testumgebung verarbeitet werden kann.

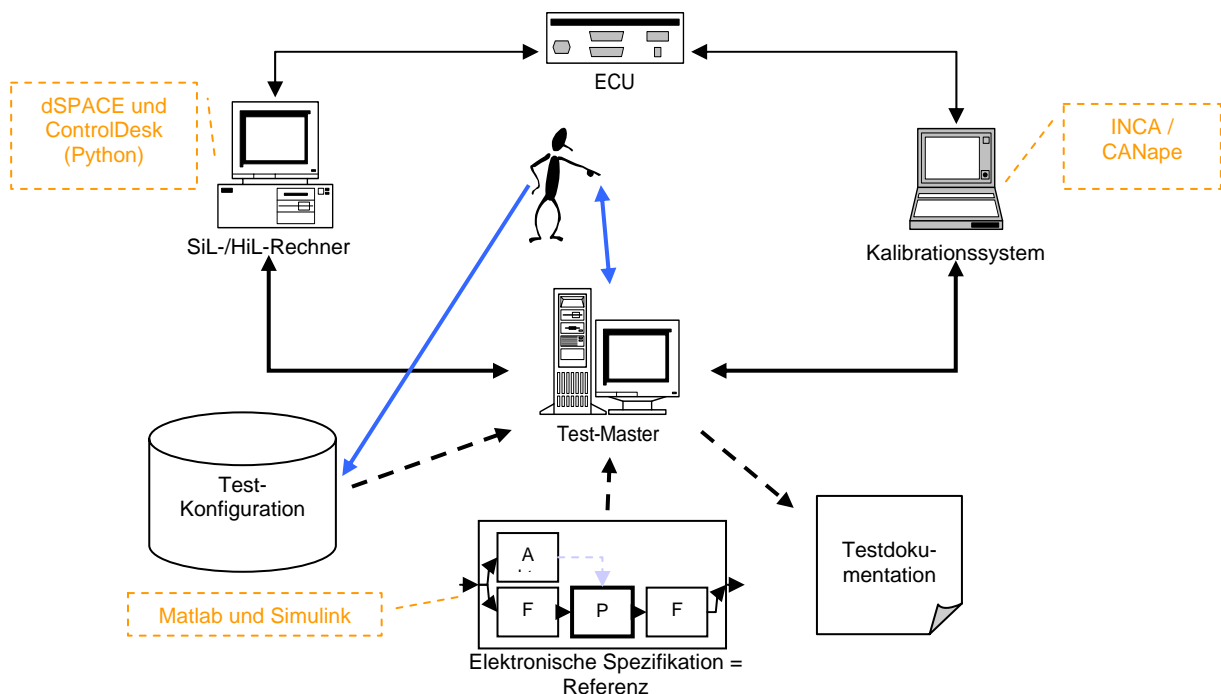


Bild 4: Rechnerumgebung für die Durchführung der Softwaretests

4.2 Aufbau der Testumgebung

Der Hardwareaufbau für die Durchführung der Softwaretests ist in Bild 4 dargestellt. Er setzt sich aus einer ECU als Plattform für die zu testende Software, einem SiL-/HiL-Rechner (z.B. dSPACE-Hardware mit Bedienung über ControlDesk), einem Kalibrationssystem (z.B. INCA oder CANape) und einem Test-Master zusammen.

Auf dem SiL-/HiL-Rechner läuft das Modell der physikalischen Umgebung (Motor, Getriebe, Fahrzeug, Fahrer, Umgebung). Bei HiL-Tests erfolgt hier zudem die Abfrage der Sensoren sowie die Ansteuerung der Aktuatoren bzw. ihrer Ersatzlasten. Bei SiL-Tests werden von diesem Rechner auch die Sensor- und die Aktuator-Schichten simuliert. Zudem kommuniziert er dann über die SiL-Kommunikationsschnittstelle mit der ECU.

Der Test-Master verwaltet die automatische Durchführung der aus der Test-Konfigurationsdatenbank ausgewählten Testfälle. Er speist die den Testfällen zugeordneten Kalibrationsgrößen und ECU-Inputs über das Kalibrationssystem ein. Zudem setzt er die dazugehörigen Parameter der Umgebungs- und Sensor-/Aktuator-simulation.

Nach der Testdurchführung erfolgt ein Soll-/Istvergleich zwischen der Spezifikation, die als Referenz dient, und dem tatsächlichen Systemverhalten. Die Resultate werden automatisch dokumentiert und abgelegt.

4.3 Durchführung der Testfälle

Bevor die Testfälle durchgeführt werden können, sind ausführbare Testskripte für die Testumgebung zu generieren, welche aus den bei der Auswahl der Testfälle definierten Betriebsszenarien und Störfällen abgeleitet werden. Die Testskripte konfigurieren die Testumgebung, simulieren einen Benutzer und rekonfigurieren die Simulation der Sensoren und Aktuatoren beim Auftreten eines simulierten Störfalls. Die Rekonfiguration der Sensor- und Aktuator-Simulation besteht aus dem Umschalten von Datenströmen in der Umgebungssimulation zwischen Simulationsmodulen für den fehlerfreien Betrieb und denen für einen fehlerbehafteten Betrieb eines Sensors oder Aktuators.

Für die Durchführung der Testfälle mit einem HiL-Test werden alle durch den Ereignisbaum beschriebenen Ausfallszenarien mit Mehrfachfehlern einzeln abgefahren. Für den Fall eines SiL-Tests lassen sich durch die virtuelle Zeitbasis, in der der Test durchgeführt wird, konsistente Zustände von zu testender Regelungssoftware und Umgebungssimulation abspeichern. Dies ermöglicht das Aufsetzen von Testabläufen auf einem gespeicherten Zustand.

Zum Speichern eines konsistenten Zustands von Simulation und zu testender Regelungssoftware werden die Simulation und die Regelungssoftware zu einem Zeitpunkt eingefroren, und es wird ein kompletter Datendump der Umgebungssimulation und des Testobjekts angelegt. Um einen weiteren Testablauf auf dem konsistenten Zustand aufzusetzen, werden der Datendump in die Simulation und die zu testende Regelungssoftware zurückgespielt und Simulation und Regelungssoftware gestartet.

Auf diese Weise brauchen verschiedene Abfolgen von Störungen oder Betriebsszenarien nicht komplett neu simuliert werden, sofern sie sich in der anfänglichen Abfolge nicht unterscheiden. Gemäß dem Ereignisbaum lässt sich an jeder seiner Gabelungen ein konsistenter Zustand anlegen, von dem aus die weiteren Tests starten. Der Zeitpunkt zur Sicherung des Zustands ist der Zeitpunkt des frühesten Ereignis-

ses nach einer Gabelung. Hierdurch lässt sich der nötige Aufwand für den Test von Mehrfachfehlern reduzieren.

Während der Durchführung der Tests werden die im Testfall vorgegebenen Grenzen für ein akzeptables Verhalten als auch das erwartete Systemverhalten überwacht. Werden die Grenzen für ein akzeptables Verhalten entsprechend der Spezifikation verletzt, so wird der Testfall mit dem Testergebnis FAILED abgebrochen. Für Testfälle mit einem nachfolgenden Störfall im Ereignisbaum bedeutet dies ebenfalls das Testergebnis FAILED. Sollte sich das System bis zum Ende des Testfalls in dem Bereich des erwarteten Systemverhaltens bewegen, so wird der Testfall mit dem Testergebnis PASSED beendet.

5. Zusammenfassung und Ausblick

Es wurde eine Vorgehensweise für den Softwaretest vorgestellt, mit der die in Software implementierten adaptiven Mechanismen zur Fehlererkennung und -kompensation in eingebetteten Systemen getestet werden können. Zukünftige X-by-Wire-Anwendungen mit ihren stark wachsenden Sicherheitsanforderungen an die Software sicherheitskritischer Systeme im Fahrzeug machen einen Testansatz für die Software erforderlich, welcher die Systeme in ihrem nominalen Verhalten als auch in Grenzsituationen und dem Ausfall von sicherheitskritischen Komponenten überprüft.

Die beschriebene Vorgehensweise wird zurzeit im Verbundvorhaben SiLEST untersucht und versuchsweise angewandt. Von besonderem Interesse ist dabei, in wie weit sich die Testfälle in einer SiL-Testumgebung automatisiert durchführen lassen und ob der SiL-Test Vorteile gegenüber den heute zumeist durchgeführten HiL-Tests besitzt. Hierzu werden im Rahmen des Projektes der SiL-Test und der HiL-Test hinsichtlich ihres Aufwands verglichen und die Grenzen beider Ansätze hinsichtlich der Aufdeckung von Fehlern und der Anwendbarkeit untersucht.

Abkürzungen

ECU	Engine Control Unit, Motorsteuergerät
FMEA	Failure Modes and Effect Analysis
HAL	Hardware Abstraction Layer
HiL	Hardware in the Loop
OS	Operating System
SiL	Software in the Loop
SiLEST	Software in the Loop for Embedded Software Test
UML	Unified Modeling Language

Literatur

- [1] B. Broekman, E. Notenboom. *Testing Embedded Software*. Addison-Wesley, 2003.
- [2] G. Booch, J. Rumbaugh, I. Jakobson. *The Unified Modeling Language: User Guide*. Addison-Wesley, 1999.
- [3] J. Rumbaugh, I. Jakobson, G. Booch. *The Unified Modeling Language: Reference Manual*. Addison-Wesley, 1999.

Dr. Olaf Maibaum
Deutsches Zentrum für Luft- und Raumfahrt e.V.
Simulations- und Softwaretechnik
Lilienthalplatz 7
38108 Braunschweig

Dipl.-Ing. Sven Rebeschies
IAV GmbH
Powertrain Mechatronics
Carnotstraße 1
10587 Berlin