Dr. Joachim Wegener

[E-Mail: joachim.wegener@berner-mattner.com]
studierte Informatik an der Technischen Universität Berlin und promovierte über den evolutionären Test von Echtzeit-Systemen an der Humboldt-Universität. Für diese Arbeit erhielt er den Software Engineering Preis der Ernst-Denert-Stiftung und der Gesellschaft für Informatik. Nach verschiedenen Teamleitungen bei der Daimler AG wechselte er 2007 zur Berner & Mattner Systemtechnik GmbH. Dort leitet er die Berliner Niederlassung. Für die Daimler AG leitete Dr. Wegener die Entwicklung des ersten industriellen Testsystems zur Automatisierung evolutionärer Strukturtests und die Entwicklung des Klassifikationsbaum-Editors CTE XL. Zudem war er an der Entwicklung des Testsystems Tessy und dem Time-Partition-Testing beteiligt.

Peter M. Kruse

[E-Mail: peter.kruse@berner-mattner.com]
ist Diplom-Informatiker und hat an der Otto-von-Guericke-Universität in Magdeburg studiert. Er arbeitet bei Berner&Mattner im Bereich des Evolutionären Tests und an der Weiterentwicklung des Klassifikationsbaum-Editors CTE XL. Außerdem erforscht und entwickelt er derzeit Erweiterungen für die Klassifikationsbaum-Methode. Peter Kruse hat bereits an einer Vielzahl von Veröffentlichungen und Konferenzbeiträgen mitgewirkt.

Hamilton Gross

[E-Mail: hamilton.gross@berner-mattner.com]
studierte Electrical und Electronic Engineering an der Universität Bristol. Während des Studiums wurde er mit dem angesehenen Engineering Leedership Award der Royal Academy of Engineering ausgezeichnet. Nach dem Studium arbeitete Hamilton Gross für verschiedene Firmen in Großbrittanien und Deutschland an der Entwicklung von eingebetteten Systemen. Seit 2008 ist er für Berner & Mattner tätig und beschäftigt sich mit dem evolutionären Test.

Evolutionäre Funktionstests für MiL-, SiL- und HiL-Umgebungen

Durch die kontinuierlich wachsende Komplexität von softwarebasierten Systemen und die sich weiter verkürzenden Entwicklungszyklen, steigt der Bedarf nach effizienten Testverfahren. Für den Test von eingebetteten Systemen ist die Verwendung von Simulationsumgebungen üblich, mit denen das zu testende System in einer kontrollierten Umgebung getestet werden kann. In der Regel erfolgen Testfallermittlung und Testimplementierung dabei manuell und sind entsprechend aufwendig. Obwohl eine Reihe akademischer Arbeiten das Potenzial evolutionärer Tests gezeigt hat, den Test in Hinblick auf verschiedene, in der Praxis gängige Testziele vollständig zu automatisieren, haben evolutionäre Tests bisher kaum Eingang in die industrielle Praxis gefunden. Mit diesem Beitrag stellen wir die Integration von evolutionären Tests in ein kommerzielles Testwerkzeug zur Automatisierung von Model-in-the-Loop, Software-in-the-Loop und Hardware-in-the-Loop Tests vor und demonstrieren die Verwendung evolutionärer Tests für den Funktionstest eines Antiblockiersystems.

Einleitung und Motivation

Durch den wachsenden Umfang und die steigende Komplexität softwarebasierter Systeme bei gleichzeitig zunehmenden Qualitäts- und Sicherheitsanforderungen wächst der Bedarf nach effizienten Testverfahren. Nur durch eine umfassende Testautomatisierung lassen sich kostenintensive manuelle Tätigkeiten während des Tests vermeiden. Entsprechend viele Werkzeuge sind für die Testautomatisierung erhältlich. Diese konzentrieren sich zumeist auf die Automatisierung der Testausführung, der Ablaufüberwachung während des Tests und die Testdokumentation. Nur wenige Werkzeuge bieten Möglichkeiten zur Automatisierung der Testfallermittlung. Gründe hierfür sind im Wesentlichen die geringe Verbreitung formaler Spezifikationen in der industriellen Praxis als notwendige Basis für die Testgenerierung und die Wichtigkeit des Erfahrungswissens und der Kreativität des Testers, auf die bei der Testfallermittlung nicht verzichtet werden soll.

In den letzten Jahren wurde von verschiedenen Autoren die Verwendung von evolutionären Tests für die Automatisierung der Testfallermittlung untersucht. Die Idee evolutionärer Tests besteht darin, typische Aufgabenstellungen beim Test softwarebasierter Systeme in ein Optimierungsproblem zu überführen, das dann mithilfe von heuristischen Suchverfahren gelöst wird. Der Eingabedatenraum des zu testenden Systems bildet dabei den Suchraum, in dem nach interessanten und fehlersensitiven Testdaten gesucht wird. Die Generierung der Testdaten erfolgt iterativ, d. h. auf Basis des im Test gezeigten Verhaltens des Testobjekts wird gelernt, wie sukzessive bessere Testdaten generiert werden können. Jedes generierte Testdatum wird hinsichtlich der Erreichung des gestellten Testziels bewertet (Fitnessbewertung), um zu entscheiden, ob es als Basis für die Generierung weiterer Testdaten herangezogen werden soll. Auf diese Art und Weise wird die Suche des Testers nach möglichst fehlersensitiven Eingabesituationen automatisiert.

Am weitesten verbreitet ist die Automatisierung von Strukturtests, bei denen eine Menge von Testdaten generiert wird, die eine weitgehend vollständige Ausführung des Kontrollflusses eines zu testenden Sys-

tems gewährleisten soll [Weg01]. Obwohl die Durchführung von Strukturtests in vielen Standards gefordert wird und die manuelle Ermittlung strukturorientierter Testfälle sehr aufwendig ist, haben evolutionäre Strukturtests noch keinen Eingang in die industrielle Praxis gefunden, da es bislang an leistungsfähigen Werkzeugen zu ihrer Unterstützung fehlt.

Eine noch größere Bedeutung als der Strukturtest hat in der Praxis der Funktionstest und damit die Verifikation des zu testenden Systems gegen seine Spezifikation. Üblicherweise werden entsprechende Tests als reine Black-Box-Tests durchgeführt: Testfälle werden manuell aus der Spezifikation abgeleitet und Interna der Implementierung werden nicht betrachtet. Trotz der großen Bedeutung der Funktionstests gibt es nur wenige Veröffentlichungen, die sich mit dem evolutionären Funktionstest beschäftigen. Bühler und Wegener [Bue08] haben in ihren Arbeiten beispielsweise gezeigt, dass evolutionäre Tests gut für die Automatisierung des Funktionstests von Fahrerassistenzsystemen geeignet sind und vollautomatisch Fehler in den Systemen gefunden werden können. Allerdings wurden die Tests stets nur in Software-in-the-Loop Testumgebungen durchgeführt.

Mit unseren Arbeiten haben wir eine Testinfrastruktur entwickelt, die eine einfache und durchgängige Verwendung von evolutionären Funktionstests für Model-in-the-Loop (MiL), Software-in-the-Loop (SiL) und Hardware-in-the-Loop (HiL) Tests ermöglicht. Unsere Testlösung haben wir mit dem Test eines im Serieneinsatz befindlichen Antiblockiersystems (ABS) evaluiert.

Testumgebung zur Automatisierung evolutionärer Funktionstests

Softwarebasierte Systeme finden in fast allen industriellen Einsatzgebieten ihre Anwendung, um komplexe Sachverhalte zu steuern, z. B. in der Luft- und Raumfahrtindustrie, der Medizin- und Umwelttechnik sowie in der Automobilindustrie.

Ziel des Funktionstests ist es, Fehler im funktionalen Verhalten des Systems aufzudecken. In der Regel werden Funktionstests für softwarebasierte Systeme auf verschiedenen Integrationsstufen durchgeführt. Besonders verbreitet sind Unit-Tests, Integrationstests und Systemtests. Ergänzend kommen Modelltests zum Einsatz, wenn die Systeme modellbasiert entwickelt werden und die Funktionsabsicherung bereits frühzeitig im Entwicklungszyklus gestartet wird. Gängige Teststufen sind entsprechend

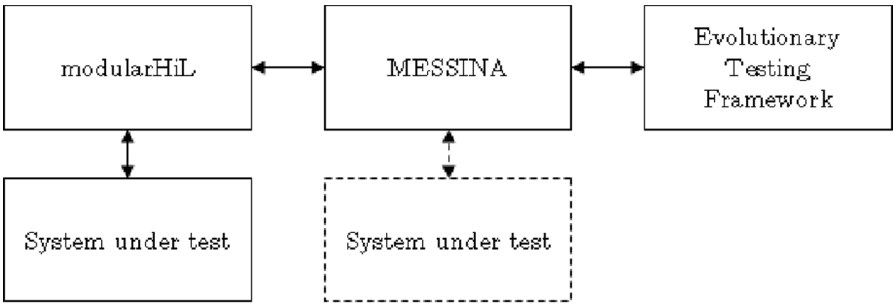

Abbildung 1: Bestandteile der Testumgebung für evolutionäre Funktionstests (für MiL und SiL Tests wird kein HiL Testsystem benötigt, die Testausführung erfolgt dann direkt durch MESSINA)

- Model-in-the-Loop Tests, mit denen Modelle oder Sub-Modelle in einer Simulationsumgebung getestet werden,
- Software-in-the-Loop Tests zur Überprüfung der entwickelten Software und
- Hardware-in-the-Loop Tests, mit denen die integrierte Software auf dem Zielsystem in einer Echtzeitsimulation getestet wird.

Ebenfalls häufig zu finden sind Processor-in-the-Loop (PiL) Tests, bei denen der Test der Software auf einem mit dem Zielprozessor bestückten Evaluation-Board durchgeführt wird.

Um eine Testumgebung bereitzustellen, die die Durchführung von evolutionären Tests in verschiedenen Teststufen unterstützt, wurden von uns verschiedene Hardware- und Software-Werkzeuge erweitert und miteinander integriert (**Abbildung 1**). Dabei handelt es sich um das HiL-Testsystem modularHiL, die Testausführungsumgebung MESSINA und das Automated Evolutionary Testing Framework.

modularHiL

Bei modularHiL [HiL] handelt es sich um ein universelles HiL-Testsystem der Berner & Mattner Systemtechnik GmbH, mit dem sich softwarebasierte Systeme und Steuergeräte-Verbünde in Echtzeit testen lassen. Dazu werden eingebettete Systeme in den Prüfstand eingespannt und die Eingangssignale vom HiL-Testsystem mit Testdaten versorgt. Die Umgebung des Testobjekts wird so simuliert, dass das Testobjekt keinen Unterschied zum Einsatz in der realen Einsatzumgebung feststellt. Vom Testobjekt berechnete Ausgangssignale fließen dazu wieder in die Simulation ein, um eine plausible Reaktion der Systemumgebung auf das Verhalten des Prüflings sicherzustellen.

In der Basiskonfiguration dient der modularHiL dem Test einzelner softwarebasierter Systeme. Durch die Verbindung mehrerer modularHiL über einen optischen Bus, ist es einfach möglich, eine leistungsfähige Testumgebung für den Integrationstest mehrerer im Verbund miteinander interagierender Testobjekte herzustellen. modularHiL-Testsysteme basieren auf industriellen Standardkomponenten, sodass eine hohe Langzeitverfügbarkeit der Komponenten bei attraktiver Preisgestaltung sichergestellt ist. Durch den modularen Aufbau der Systeme ist eine gute Skalierbarkeit der Testumgebungen gegeben: Es kann stets die notwendige Rechenleistung für die Ausführung der Tests in Echtzeit bereitgestellt werden.

MESSINA

MESSINA [MES] ist ein Testwerkzeug für die Implementierung und Ausführung von hardware- und softwareunabhängigen Testfällen. Die Modellierung der Testfälle kann dabei in unterschiedlichen Notationen erfolgen, z. B. UML, Java oder TPT [TPT]. Die Stärke von MESSINA liegt in der virtuellen Integration mehrerer miteinander vernetzter Komponenten. Weitere Leistungsmerkmale sind die Möglichkeit Templates für sich wiederholende Testumfänge anzulegen und diese in Testfälle zu integrieren, die Definition generischer Testfälle und die enthaltene Parametrierung und Variantenbildung für Testfälle.

Durch zwei Abstraktionsschichten wird die Ausführbarkeit der in MESSINA beschriebenen Testfälle auf unterschiedlichen Testumgebungen und in verschiedenen Teststufen gewährleistet. Die erste Abstraktion beruht auf der Einführung eines Signalpools, der alle Signale der angeschlossenen Systeme und Software-Komponenten enthält. Über den Signalpool sind einheitliche

Schreib- und Lesezugriffe auf die Ein- und Ausgänge der Testobjekte und der Simulationsumgebung realisierbar. Die Umsetzung auf die konkreten Signale und Protokolle der Testobjekte wird von MESSINA automatisch vorgenommen. Die zweite Abstraktionsschicht besteht in der Bereitstellung von Ausführungsumgebungen für verschiedene MiL-, SiL- und HiL-Testumgebungen, wie Matlab oder modularHiL. Einmal definierte Tests können damit unverändert auf unterschiedlichen Testumgebungen (MiL, SiL, HiL) und für unterschiedliche Teststufen ausgeführt werden. Der einzige Unterschied liegt in der Auswahl der jeweils gewünschten Testumgebung für die Testausführung. Der MiL- und SiL-Test einer ABS-Steuerung erfolgt mit dem ABS-Modell bzw. der ABS-Software in einer Software-Simulation. Für den Test des ABS-Steuergeräts wird das Steuergerät hingegen in den HiL-Prüfstand eingespannt und in MESSINA der HiL für die Testdurchführung ausgewählt. Eine Anpassung der Tests ist nicht erforderlich.

Derzeit stellt MESSINA Ausführungsumgebungen für die weit verbreiteten Simulationsplattformen MATLAB/Simulink und ASCET zur Verfügung. Darüber hinaus besteht die Möglichkeit, AUTOSAR [AUT] Software-Komponenten mit MESSINA zu testen. Grundsätzlich können beliebig viele Software-Modelle und Software-Komponenten unabhängig von der eingesetzten Modellierungstechnik gemeinsam miteinander in MESSINA ausgeführt werden, um eine virtuelle Systemintegration und die entsprechenden Integrationstests durchzuführen.

Als Hardware-in-the-Loop Testsysteme werden modularHiL- und dSPACE-Prüfstände unterstützt. Die in MESSINA definierten Tests werden hierbei für die Testdurchführung auf den HiL hinuntergeladen, um dort in Echtzeit ausgeführt zu werden.

MESSINA bildet damit eine hervorragende Basis, um evolutionäre Funktionstests für möglichst viele Testumgebungen und Teststufen anzubieten.

Automated Evolutionary Testing Framework

Den Kern der Testumgebung zur Automatisierung von evolutionären Funktionstests bildet das im EU-geförderten Projekt Evo-Test [Evo] entstandene Automated Evolutionary Testing Framework. Dieses Framework stellt Komponenten und Schnittstellen für die automatisierte Testdatengenerierung mit heuristischen Suchverfahren, für die Testdurchführung und Ablaufüberwachung sowie für die Auswertung durchgeführter Test und die Fitnessbewertung der erzeugten Testdaten bereit.

Dem Framework wird die Schnittstelle des zu testenden Systems als Problembeschreibung übergeben. Die Schnittstelle wird von der im Framework integrierten Komponente GUIDE [Gui], [DaC07] analysiert, um dann einen auf das Testobjekt zugeschnittenen evolutionären Algorithmus zu generieren. Der erzeugte evolutionäre Algorithmus übernimmt die Implementierung des Suchverfahrens und die Generierung der Testdaten. Die Testdaten werden über das Framework für die Testausführung bereitgestellt. Zu jedem Testdatum erwartet das Framework und der evolutionäre Algorithmus einen Fitnesswert zurück, der die Eignung des Testdatums für die Erfüllung des gestellten Testziels widerspiegelt. Auf Basis der Fitnesswerte werden die Testdaten der nächsten Iteration generiert. Der Test endet, wenn das Testziel erreicht worden ist oder eine vorher definierte Anzahl von Testdaten erzeugt und ausgeführt wurde. Eine umfassende Darstellung des Automated Evolutionary Testing Framework geben Dimitar et al. [Dim08].

Die Fitnessbewertung der Testdaten muss beim Funktionstest außerhalb des Frameworks vorgenommen werden, d. h. im Rahmen der Ablaufüberwachung und Testauswertung muss entschieden werden, ob das jeweils ausgeführte Testdatum einer Fehleraufdeckung nah kommt oder kein Potenzial hinsichtlich einer Fehleraufdeckung besitzt. Wichtig bei der Gestaltung der Fitnessberechnung ist, dass stets sichergestellt ist, dass ein interessantes Testdatum einen höheren Fitnesswert erhält als ein weniger interessantes.

Integration Komponenten

Für die Entwicklung einer Testumgebung zur Automatisierung von evolutionären Funktionstests müssen die beschriebenen Bausteine modularHiL, MESSINA und Automated Evolutionary Testing Framework miteinander integriert werden. Da MESSINA bereits standardmäßig über eine gute Integration mit modularHiL verfügt, waren hier lediglich Erweiterungen erforderlich, die es ermöglichen, eine Menge generierter Testdaten über generische Testfälle auf dem HiL zur Ausführung zu bringen. Für die Integration von MESSINA mit dem Automated Evolutionary Testing Framework wurde MESSINA um ein Plugin für die Konfigurierung und Durchführung evolutionärer Tests erweitert.

Das Plugin implementiert die Kommunikation zwischen MESSINA und dem Framework, liefert dem Framework die Schnittstellenbeschreibung des Testobjekts und die vom Anwender gewünschten Randbedingungen für die Konfiguration des Suchalgorithmus, steuert die Testausführung der generierten Testdaten, berechnet die Fitnesswerte zu den Testdaten auf Basis der Ergebnisse aus der Testdurchführung und liefert diese wieder an das Framework.

Für die Schnittstellenbeschreibung des Testobjekts wird der MESSINA-Signalpool ausgelesen, der alle relevanten Daten für die Spezifikation der Signale enthält, beispielsweise den Datentyp der Signale und zulässige Wertebereiche. Dabei können auch verschiedene Datentypen für unterschiedliche Signale miteinander kombiniert werden. Auf Basis der übergebenen Schnittstellenbeschreibung wird von der GUIDE-Komponente des Frameworks ein evolutionärer Algorithmus für den durchzuführenden Funktionstest generiert.

Anschließend wird der Testlauf von MESSINA gestartet. GUIDE beginnt mit der Generierung einer Menge von Testdaten für die erste Iteration. Die Testdaten werden von MESSINA entgegengenommen und ausgeführt. Hierfür werden im MESSINA erzeugte generische Testfälle mit den Testdaten parametrisiert. In vielen Fällen können die generierten Testdaten direkt ausgeführt werden. Müssen komplexere Eingangsdatenverläufe abgebildet werden, erfolgt im generischen Testfall eine Transformation der generierten Daten in die gewünschten Eingabedatensequenzen.

Für MiL- und SiL-Tests wird das Testobjekt direkt von MESSINA ausgeführt. Für den HiL-Test werden die Tests auf den modularHiL hinuntergeladen und dann die Testdurchführung auf dem HiL angestoßen. Der Test des softwarebasierten Systems erfolgt in Echtzeit auf der Zielhardware. Das Laufzeitsystem von MESSINA ermöglicht hierbei eine einfache Ausführung der Tests auf dem Zielsystem und stellt die Mechanismen für die Ablaufüberwachung zur Verfügung. Derselbe Testfall kann dadurch sowohl für die Ausführung der MiL- und SiL-Tests als auch für die HiL-Tests verwendet werden.

Für die Fitnessbewertung der Testdaten muss das Verhalten des Testobjekts zu jedem ausgeführten Test analysiert werden. MESSINA zeichnet hierfür das Istverhalten des Testobjekts über Monitoring-Schnittstellen auf. Das gezeigte Verhalten wird im

generischen Testfall für die Fitnessberechnung ausgewertet. Jedem Testdatum wird ein Fitnesswert zugeordnet, der an das Automated Evolutionary Testing Framework weitergegeben wird. Unter Berücksichtigung der Fitnesswerte erfolgt durch GUIDE dann die Testdatengenerierung für die nächste Iteration.

Mit der entwickelten Testumgebung können evolutionäre Funktionstests durchgängig vom Modelltest bis zum HiL-Test des Zielsystems automatisiert werden. Durch die hardware-unabhängige Definition der Testfälle in MESSINA ist die Portabilität der Tests über die verschiedenen Integrationsstufen gewährleistet. Der einzige Unterschied zwischen MiL-, SiL- und HiL-Tests liegt in der Auswahl unterschiedlicher Ausführumgebungen.

Evaluierung

Um die entwickelte Testumgebung zur Automatisierung evolutionärer Funktionstests zu evaluieren, wurde ein evolutionärer Funktionstest für die Prüfung eines Antiblockiersystems (ABS) aufgesetzt. Das getestete ABS befindet sich bereits im Serieneinsatz.

Antiblockiersystem (ABS)

Ein Antiblockiersystem verhindert das Blockieren der Räder eines Fahrzeugs während eines Bremsmanövers und erlaubt dem Fahrer, das Fahrzeug auch bei einer Vollbremsung weiter steuern und lenken zu können. Bei widrigen Witterungsbedingungen wie Eis und Schnee verkürzt ein Antiblockiersystem zudem den Bremsweg.

Ein ABS baut auf der physikalischen Erkenntnis auf, dass eine maximale Bremsverzögerung je nach Fahrbahnzustand bei etwa 10 bis 30 % Radschlupf erreicht wird. 20 % Bremsschlupf bedeuten, dass im selben Zeitraum, in dem das Fahrzeug einen Weg von einem Meter zurücklegt, die Räder nur 0,8 Meter abrollen. Wird die maximal übertragbare Bremskraft überschritten, wächst der Bremsschlupf, bis das Rad schließlich blockiert (= 100 % Bremsschlupf). Im blockierten Zustand wird nur über Gleitreibung abgebremst, die typischerweise um 15 bis 20 % unter der Haftreibung liegt. Auch ist ein Fahrzeug mit blockierten Reifen kaum mehr steuerbar. Das ABS reguliert die Bremskraft an jedem Rad so, dass der Schlupf während des Bremsvorganges jederzeit möglichst nahe an der optimalen Grenze bleibt [ABS].

Wenn das Bremspedal eines mit ABS ausgestatteten Fahrzeugs kräftig betätigt wird, wie beispielsweise bei einer Vollbremsung, dann sorgt das ABS dafür, dass die Bremse mehrmals pro Sekunde gepumpt wird. Dabei messen Sensoren die Radumdrehungen und verringern hydraulisch den Bremsdruck, wenn die Räder zu langsam werden, also die Gefahr des Blockierens besteht. Das ABS versucht mehrmals pro Sekunde, den Bremsdruck knapp unterhalb des Blockier-Grenzwerts einzuregeln. Ziel ist es, den Radschlupf in einem Bereich zu halten, der trotz möglichst hoher Fahrzeugverzögerung eine gute Steuerbarkeit des Fahrzeugs gewährleistet.

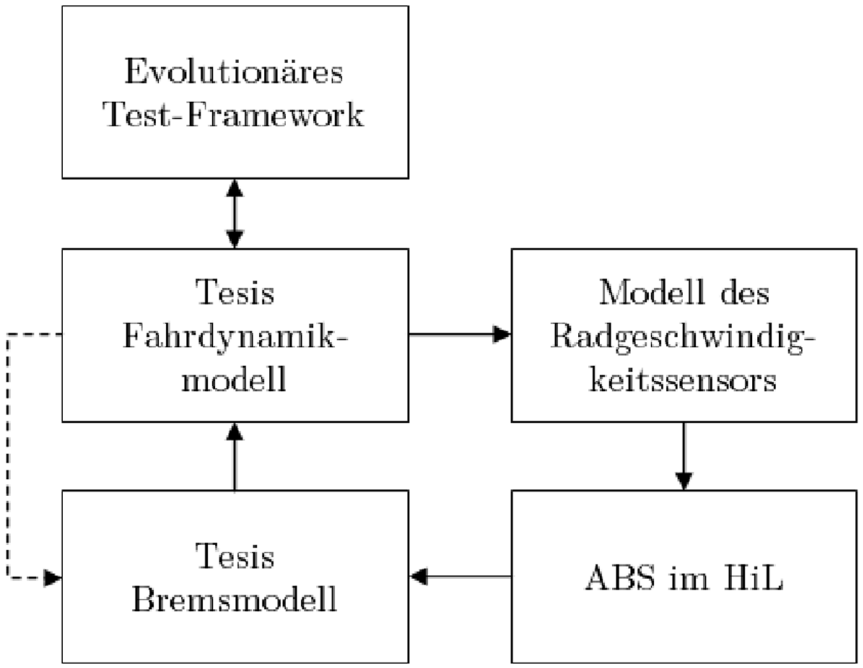

Abbildung 2: Aufbau der Umgebungssimulation für den Test des Antiblockiersystems

Evolutionärer Funktionstest des ABS

Für den evolutionären Funktionstest des ABS wurde ein HiL-Test mit der entwickelten Testumgebung aufgesetzt. Darüber hinaus wurde ein Umgebungsmodell für das ABS erstellt, um die Systemumgebung des ABS im Test zu simulieren. Das ABS-Steuergerät würde auf ein nicht realitätsnahes Verhalten der Systemumgebung mit einer Fehlerabschaltung reagieren und den Funktionstest damit verhindern. Bestandteile der Umgebungssimulation sind ein Bremsmodell und ein Fahrdynamikmodell von Tesis [Tes] und ein Modell des Radgeschwindigkeitssensors. Abbildung 2 zeigt das Zusammenspiel der beteiligten Komponenten.

Fitnessfunktion

Ziel des Tests ist es, dass ABS in kritische Bereiche zu bewegen, in denen der Radschlupf schlecht eingeregelt wird, sodass nur eine eingeschränkte Fahrzeugverzögerung erreicht wird. Als Testdaten werden unterschiedliche Reibwertprofile generiert, d. h. Profile wechselnder Griffigkeit für den Straßenbelag auf dem das Bremsmanöver stattfindet. Alle übrigen Fahrzeugparameter, wie Fahrzeuggeschwindigkeit, Bremsdruck oder Lenkwinkel, bleiben für alle Tests konstant. Ziel ist es folglich, ein Reibwertprofil zu generieren, in dem es dem ABS nicht gelingt, einen optimalen Radschlupf einzuregeln.

Die Fitnessfunktion basiert auf der Berechnung des vom ABS erreichten durchschnittlichen Radschlupfs S während eines Bremsvorgangs:

$$S = \sum_{k=1}^{n} \frac{s_k}{n}$$

Je größer der durchschnittliche Radschlupf während der Bremsung ist, desto geringer ist die abgesetzte Fahrzeugverzögerung. Folglich muss der durchschnittliche Radschlupf während der Suche maximiert werden, um kritische Szenarien für das ABS zu ermitteln. Da Teile des Automated Evolutionary Testing Frameworks nur zur Lösung von Minimierungsproblemen geeignet sind, ergibt sich als Fitnessfunktion schließlich:

$$F = \min(1 - S)$$

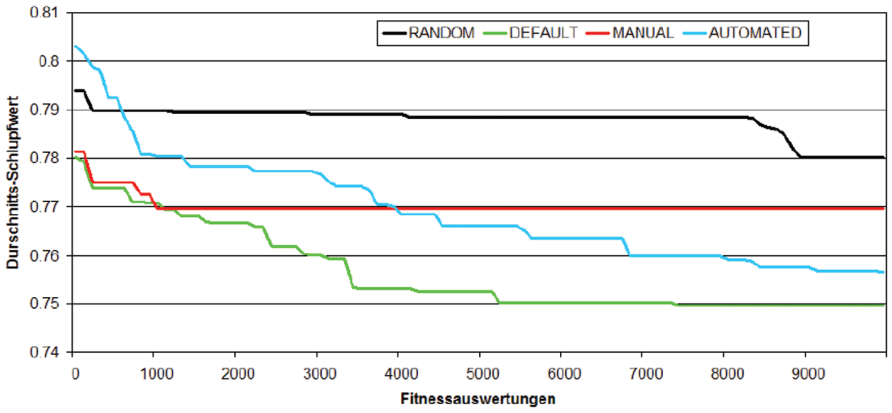

Abbildung 3: Beste Fitnesswerte in der jeweiligen Iteration über alle Versuche

Durchgeführte Experimente

Mit der beschriebenen Testumgebung zur Automatisierung von evolutionären Funktionstests wurden in den Experimenten stets Testläufe über 100 Iterationen mit jeweils 100 generierten Testdaten ausgeführt. Für den evolutionären Test wurden dabei drei unterschiedliche Suchkonfigurationen des evolutionären Algorithmus festgelegt und durchgeführt:

- DEFAULT: Die mit dem Automated Evolutionary Testing Framework ausgelieferten Standardeinstellungen für die Suche
- MANUAL: manuelle Konfiguration der Suche
- AUTOMATED: Dynamische Adaption der Suche während des Tests

Eine detaillierte Erläuterung der möglichen Einstellungen und Parameter für die Konfiguration der Suche findet sich in der Beschreibung zu GUIDE [DaC].

Zum Vergleich der evolutionären Funktionstests mit anderen Testverfahren wurden zudem Zufallstests mit jeweils 10.000 Testdaten durchgeführt (RANDOM). Alle Testläufe wurden dreißigmal wiederholt, um die Verlässlichkeit der Experimente zu steigern.

Testergebnisse

Die in den 30 Versuchen erzielten Testergebnisse werden in **Abbildung 3** und **Abbildung 4** dargestellt. **Abbildung 3** zeigt über alle 10.000 ausgeführten Testdaten, den jeweils bis zu diesem Testdatum gefundenen besten Fitnesswert an. **Abbildung 4** zeigt den durchschnittlichen Fitnesswert aller Individuen bis zu dem jeweiligen Testdatum an.

Deutlich zu erkennen ist, dass der Zufallstest (*RANDOM*) am schlechtesten abschneidet, sowohl was die absoluten Ergebnisse angeht als auch die erreichten Durchschnittswerte. Dies liegt im Wesentlichen daran, dass Zufallstests die Ergebnisse aus vorherigen Tests nicht in die Erstellung neuer Testfälle einbeziehen können und somit keine zielgerichtete Suche erfolgt.

Mit der Suchstrategie *DEFAULT* wurden sehr gute Ergebnisse erzielt. Über alle Experimente betrachtet lieferte diese Strategie das beste Einzelergebnis. In der Betrachtung der Durchschnitte ist die Strategie *DEFAULT* allerdings der Strategie *AUTOMATED* unterlegen, d. h. die Varianz der Ergebnisse ist hier höher, Zuverlässigkeit und Robustheit der Suche geringer. Es ist davon auszugehen, dass bei einer weiteren Fortsetzung der Suche, insbesondere mit der Strategie *AUTOMATED*, noch bessere Ergebnisse erzielt würden, da die dynamische Adaption der Suche selbst eine Reihe von Generationen benötigt und die optimierte Suche möglicherweise in den 100 Generationen noch nicht ihre volle Wirkung erreichen konnte. Dieses legen beide Abbildungen durch den steileren Gradienten der Ergebnisverbesserung für die Strategie *AUTOMATED* gegenüber den anderen Verfahren in den letzten Generationen nahe. Während die Suchstrategien *DEFAULT* und *MANUAL* nach ca. 50 Iterationen bzw. 5.000 Testdaten nur noch geringe Steigerungen in den Fitnesswerten erzielen, ergeben sich für die Strategie *AUTOMATED* kontinuierliche Verbesserungen.

Die manuell konfigurierte Suche (*MANUAL*) ist zwar deutlich besser als der Zufallstest, fällt aber gegenüber den beiden Suchstrategien *AUTOMATED* und *DEFAULT* zurück. Die Konfiguration der Suche ist offensichtlich nicht optimal gewählt.

Bewertung

Mit dem evolutionären Funktionstest des Antiblockiersystems konnten sehr gute Ergebnisse erzielt werden. Alle angewendeten Suchstrategien waren gegenüber dem Zufallstest überlegen. Gerade mit den erfolgreichen Strategien *DEFAULT* und *AUTOMATED* wurden Reibwertprofile gefunden, in denen das Antiblockiersystem keine gute Bremsung einregeln konnte. Durch die automatische Adaption der Suche mit der Strategie *AUTOMATED* verläuft die Suche zunächst langsamer als bei den anderen Strategien. Mit fortschreitender Anzahl von Iterationen entwickeln sich die Fitnesswerte aber besser als für die anderen Strategien.

Mit dem evolutionären Funktionstest erfolgt ein zielgerichteter, vollständig automatisierter Test des zu testenden Systems mit einer Vielzahl von Testdaten. Dadurch können in vielen Fällen kritische Situati-

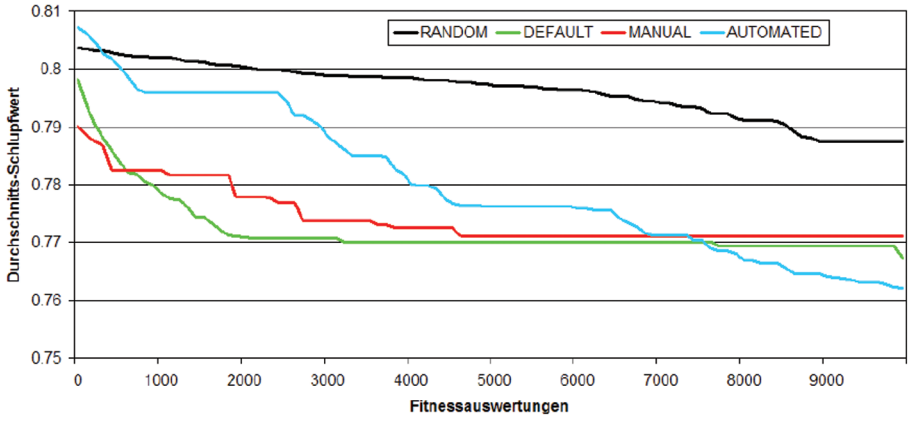

Abbildung 4: Durchschnitt der besten Fitnesswerte in er jeweiligen Iteration über alle Versuche

onen gefunden werden, die bei einer manuellen Testfallermittlung möglicherweise übersehen oder als nicht kritisch eingestuft werden.

Für die Nutzung der entwickelten Testumgebung zur Automatisierung evolutionärer Funktionstests sind Kenntnisse des im EvoTest-Projekt entwickelten Automated Evolutionary Testing Framework erforderlich. Insbesondere die hohe Anzahl einstellbarer Parameter für die Konfiguration der Suche erschwert neuen Anwendern die Anwendung. Aus diesem Grund wurden Strategien für die automatische Adaption der Suche entwickelt und in das Framework integriert. Die durchgeführten Experimente haben gezeigt, dass eine automatische Anpassung der Suche unter Berücksichtigung der durchgeführten Testläufe gute Ergebnisse erzielen kann.

Eine weitere Schwierigkeit beim industriellen Einsatz des evolutionären Funktionstests liegt in der Definition der Fitness-Funktion. In der Praxis sollte diese iterativ oder in Zusammenarbeit mit Experten entwickelt werden, um Erfahrungen aus ersten Testläufen in die Definition mit einfließen zu lassen.

Zusammenfassung und Ausblick

Zur Verbesserung der industriellen Anwendbarkeit evolutionärer Funktionstests wurde von uns eine Testumgebung unter Verwendung kommerzieller Testwerkzeuge (modularHiL und MESSINA) und eines prototypischen Frameworks zur Automatisierung von evolutionären Tests aufgebaut. Die entwickelte Testumgebung ermöglicht die vollständige Automatisierung von Funktionstests für MiL- und SiL-Tests unter Verwendung von evolutionären Algorithmen. Dadurch lassen sich gängige Teststufen wie der Modelltest und der Softwaretest auf unterschiedlichen Integrationsstufen abdecken. Durch die Unterstützung von HiL-Testsystemen wird zudem der Systemtest auf der realen Zielhardware in einer Echtzeitsimulation ermöglicht.

Evolutionäre Funktionstests bilden eine gute Ergänzung zu konventionellen Tests und reduzieren das Risiko, wichtige Tests zu übersehen, die vom Tester nicht berücksichtigt worden sind.

Für die Evaluierung der Testumgebung zur Automatisierung von evolutionären Funktionstests wurde ein ABS getestet. Testfälle wurden dabei mit unterschiedlichen Suchstrategien generiert und mit einem Zufallstest verglichen. Dabei zeigten sich alle evolutionären Tests gegenüber dem Zufallstest überlegen. Die Tests konnten problemlos in SiL- und HiL-Testumgebungen ausgeführt werden. Die Suche nach interessanten Testdaten war erfolgreich. Es wurden automatisch Reibwertprofile gefunden, die zu einem zu langen Bremsweg führten.

Künftige Verbesserungen der Testumgebung sollen eine Visualisierung des Test- und Suchfortschritts während der Testdurchführung ermöglichen, um schneller auf ungeeignete Konfigurationen reagieren zu können. Zudem sollen verlässliche Abbruchkriterien für den Test entwickelt werden. In den durchgeführten Experimenten wurde der Test nach einer im Vorhinein definierten Anzahl von Iterationen abgebrochen. Wenigstens ein Suchverfahren (*AUTOMATED*) hätte aber bei einer Fortsetzung der Suche mit hoher Wahrscheinlichkeit noch bessere Ergebnisse erzielt. Auch ein evolutionärer Test, der in einem Testlauf mehrere Testziele gleichzeitig verfolgt wird mit der aktuellen Lösung noch nicht unterstützt. Beispiele sind eine Kombination von Funktionstests und Zeitverhaltenstests oder von Funktionstests und Strukturtests.

Da die Testdurchführung in Echtzeit am HiL zeitaufwendig ist, sollen künftig zudem Strategien entwickelt werden, die die Anzahl auszuführender Testdaten reduziert. Hierfür soll das Potenzial von Testdaten bereits vor der Ausführung beurteilt werden, in dem eine Potenzialbewertung aus den bereits durchgeführten Tests gelernt wird. Testdaten, die bereits durchgeführten Tests zu ähnlich sind, werden vom Test ausgenommen.

Zusätzliche Erweiterungen betreffen die Integration eines Signalgenerators für die Generierung komplexer Eingangssignalverläufe [Win08] und die Verwendung von Techniken zur Reduzierung der Suchraumgröße [Har07].

Acknowledgements

Die beschriebenen Arbeiten wurden durch die EU-Förderung IST-33472 (EvoTest) ermöglicht.

Referenzen

[Weg01] Wegener, J.; Baresel, A.; Sthamer, H.: Evolutionary test environment for automatic structural testing. Information and Software Technology, 43(1):841-854, 2001.

[Bue08] Buehler, O.; Wegener, J.: Evolutionary functional testing. Comput. Operations Research. 35 (10), 3144-3160, 2008.

[HiL] modularHiL: http://berner-mattner.com/automotive-modul-hil.php

[MES] MESSINA: http://berner-mattner.com/automotive-messina.php

[TPT] TPT: http://piketec.com/products/tpt.php?lang=en

[Evo] EvoTest: http://www.evotest.eu/

[Dim08] Dimitar, M.; Dimitrov, I. M.; Spasov, I.: Evotest - Framework for customizable implementation of Evolutionary Testing. International Workshop on Software and Services, October 2008, Sofia, Bulgaria.

[Gui] GUIDE. http://guide.gforge.inria.fr/

[DaC07] Da Costa, L.; Schoenauer, M.: GUIDE, a Graphical User Interface for Evolutionary Algorithms. GECCO Workshop on Open-Source Software for Applied Genetic and Evolutionary Computation (SoftGEC), 2007.

[Tes] Tesis veDYNA. http://www.tesis.de/en/index.php?page=544

[Win08] Windisch, A.: Search-Based Testing of Complex Simulink Models containing Stateflow Diagrams. Proceedings of the 1st International Workshop on Search-Based Software Testing, Lillehammer, Norway, 2008.

[DaC] Luis Da Costa and Marc Schoenauer: Description of Evolution Engine Parameters, http://guide.gforge.inria.fr/eeparams/EEngineParameters.html

[AUT] AUTOSAR: http://www.autosar.org

[Har07] Harman, M.; Hassoun, Y.; Lakhotia, K.; McMinn, P.; Wegener, J.: The Impact of Input Domain Reduction on Search-based Test Data Generation. Proceedings of the 6th European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, pp. 155-164, 2007.

[ABS] ABS-Funktionsweise: http://de.wikipedia.org/wiki/Antiblockiersystem