

# Testautomatisierung nach dem Schachspielerprinzip

---

Andreas Junghanns, Jakob Mauss, Mugur Tatar

## Abstract

The tight interaction among an ever increasing amount of software functions and hardware subsystems (mechanics, hydraulics, electronics, etc.) leads to a new kind of complexity that is difficult to manage during mechatronic design. System tests have to consider huge amounts of relevant test cases. Validation with limited resources (time and costs) is a challenge for the development teams. We present a new instrument that should help engineers in dealing with the complexity of test and validation. TestWeaver is based on a novel approach that aims at maximizing test coverage with minimal work load for the test engineer for specifying test cases. The method integrates simulation (MiL/SiL) with automatic test generation and evaluation, and has found successful applications in the automotive industry. We illustrate the approach using a 6-speed automatic transmission for passenger cars.

## Kurzfassung

Das Zusammenspiel von Funktionssoftware mit mechanischen, hydraulischen und elektronischen Bauteilen schafft eine Komplexität, die die Entwicklungsingenieure vor grosse Herausforderungen stellt. Fehler während der Produktentwicklung sind daher kaum zu verhindern. Entscheidend ist es dann, alle diese Fehler zu finden, und zwar rechtzeitig, also noch vor Produktionsanlauf. Dabei gilt: je früher im Entwicklungsprozess ein Konstruktions- oder Programmierfehler erkannt wird, desto schneller und billiger kann er behoben werden. Beim entwicklungsbegleitenden Testen von Mechatroniksystemen steht der Testingenieur allerdings heute vor einem Dilemma: Einerseits soll er eine möglichst große Testabdeckung erzielen, hat aber andererseits dafür nur begrenzte Mittel (Zeitfenster, qualifizierte Mitarbeiter) zur Verfügung. Wir stellen hier ein Verfahren vor, das in vielen Anwendungsfällen hilft, diesen Spagat zu meistern: Das Verfahren kombiniert klassische Simulation bzw. Co-Simulation (MiL, SiL) mit einer Methode zum automatischen Erzeugen, Durchführen und Auswerten von Tests. Damit lassen sich anhand virtueller Prototypen in kurzer Zeit zehntausende von Szenarien voll automatisiert testen, ohne das hierfür Test- oder Auswerteskripte zu schreiben sind. Das entsprechende Testwerkzeug TestWeaver wird zur Zeit in mehreren Serienentwicklungsprojekten der Automobilindustrie erfolgreich eingesetzt.

---

in: Clemens Gühmann (Hrsg.): Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik, Expert Verlag Renningen, ISBN -13: 978-3-8169-2818-8, 2008.

## 1. Einleitung

Die enge Verzahnung der ständig zunehmenden Zahl von Softwarefunktionen mit den mechanischen, hydraulischen und elektronischen Teilsystemen eines Fahrzeugs erzeugt eine neue Art von Komplexität. Ein umfassender Systemtest erfordert hier die Betrachtung einer sehr grossen Zahl relevanter Testfälle. Der entwicklungsbegleitende Systemtest mit begrenzten Ressourcen (Zeitfenster und Arbeitskosten) ist daher für die Entwicklungsteams mechatronischer Systeme eine grosse Herausforderung. Wir stellen auf den folgenden Seiten ein Werkzeug vor, das den Prozess des entwicklungsbegleitenden Testens weitgehend automatisiert (Bild 1). Dadurch kann die Testabdeckung dramatisch gesteigert werden ohne dabei gleichzeitig die Arbeitsbelastung für die Testingenieure zu erhöhen.



Bild 1: Testen als Spiel gegen das simulierte System

Der Artikel gliedert sich wie folgt. Abschnitt 2 beschreibt die Aufgabenstellung Systemtest am Beispiel eines Automatikgetriebes. Abschnitt 3 liefert eine kritischen Analyse der heute im Automobilbau eingesetzte Testmethoden. Abschnitt 4 beschreibt das neue Testverfahren. Der Artikel schließt mit einer Zusammenfassung der wichtigsten Erkenntnisse aus der industriellen Anwendung des Verfahrens.

## 2. Herausforderung Systemtest

Ziel des Systemtests ist es, alle verborgenen Fehler und Schwachstellen des Systems zu finden, und zwar so früh wie möglich im Entwicklungsprozess, auf jeden Fall aber, bevor das System produziert und an Kunden ausgeliefert wird.

Dazu reicht es normalerweise nicht aus, das System nur unter Laborbedingungen und nur für ein paar idealisierte Fälle zu testen. Um die Wahrscheinlichkeit zu erhöhen, wirklich alle Fehler und Schwachstellen zu finden, wird man stattdessen versuchen, das System in möglichst vielen relevanten Systemzuständen zu untersuchen. Man betrachte als Beispiel ein modernes Automatikgetriebe. In diesem Fall hat der Raum der möglichen Systemzustände mindestens die folgenden Dimensionen:

- *Wetter*: Außentemperaturen schwanken, z. B. von  $-40^{\circ}\text{C}$  bis  $40^{\circ}\text{C}$ , mit signifikanten Auswirkungen auf das Verhalten des hydraulischen Subsystems.
- *Straße*: verschiedene Straßenprofile, Bergfahrt, Talfahrt, Rad-Straße Kontakt
- *Fahrer*: Unterschiedliche Fahrerprofile, einschließlich unvorhergesehenen (merkwürdigem) Fahrerverhalten.

- *Spontane Bauteilfehler*: Einzelne Bauteile des Getriebes können während der Fahrt jederzeit ausfallen. Die Funktionssoftware muss solche Situationen korrekt erkennen und angemessen reagieren, z. B. durch selektives Abschalten einzelner Getriebefunktionen oder durch Schalten in den Notbetrieb.
- *Fertigungstoleranzen*: mechanische, elektrische und andere physikalische Bauteileigenschaften variieren in bestimmten Grenzen, abhängig vom Herstellungsprozess.
- *Alterung*: Parameterdrift für bestimmte Bauteil über die gesamte Lebenszeit des Getriebes.
- *Interaktion mit anderen Aggregaten*: Das Getriebe kommuniziert mit anderen Aggregaten (Motor, Bremssystem) über ein Netzwerk (CAN). Zum Beispiel bitet das Getriebe während einer Schaltung den Motor, das von diesem gelieferte Drehmoment kurzzeitig zu reduzieren, um die Schaltelemente des Getriebes zu schonen.

Diese Achsen spannen einen großen Raum möglicher Systemzustände auf. Die möglichen Zustände entlang jeder Achse multiplizieren sich dabei und bilden einen Raum (Kreuzprodukt) mit unendlich vielen Zuständen. Das ultimative Ziel des Systemtests besteht darin, zu zeigen, dass sich das System in jedem dieser Zustände genügend gut verhält. Es wäre natürlich ideal, wenn man Beweistechniken einsetzen könnte, um bestimmte Systemeigenschaften nachzuweisen, z. B. die physikalische Unmöglichkeit bestimmter unerwünschter Verhaltensweisen. Leider sind solche Beweistechniken (wie z. B. model checking, [1]) für die Analyse von komplexen Aggregaten wie dem hier betrachteten Automatikgetriebe heute noch nicht mächtig genug. In der Praxis muss daher das Ziel, alle Systemzustände zu untersuchen durch Auswahl und Analyse einer endlichen Menge von Testfällen aus diesem Raum approximiert werden.

### 3. Kritische Betrachtung heutiger Testmethoden

Der Test unterschiedlicher Integrationsstufen (Bauteil, Modul, System, Fahrzeug) in unterschiedlichen Testumgebungen (z. B. MiL, SiL, HiL, Prüfstand, Fahrversuch) ist heute integraler Bestandteil automobiler Entwicklungsprozesse. Je früher hierbei Probleme erkannt und behoben werden, desto besser. Allerdings

- lassen sich viele relevante Tests erst auf Systemebene formulieren, z. B. der Test der Systemreaktion beim Auftreten eines Bauteilfehlers
- werden Systemtests oft nur am HiL, Prüfstand oder im Fahrversuch durchgeführt.

Der Test am HiL, Prüfstand oder per Fahrversuch unterliegt aber folgenden Beschränkungen

- *Zeit, Kosten, Sicherheit*: HiL, Prüfstände und physikalische Prototypen sind vergleichsweise teuer und eine knappe Resource im Entwicklungsprozess; es können daher nicht sehr viele Tests durchgeführt werden, der Test der Systemreaktion auf Bauteilfehler ist aus Zeit- und Sicherheitsgründen nur für bestimmte Bauteilfehler praktikabel.
- *fehlende Agilität*: Verzögerung zwischen Änderung einer Softwarefunktion und ihrem Test im HiL, Prüfstand oder Fahrversuch, oft in der Größenordnung von Tagen.
- *Eingeschränkte Präzision oder Sichtbarkeit*: Wegen der Echtzeitanforderung an die Simulationsmodelle für HiL sind diese Modelle oft extrem simplifiziert

und darum unpräzise; Debugging und Inspektion von verborgenen Systemgrößen ist vergleichsweise schwierig wenn nicht unmöglich im HiL, Prüfstand oder Fahrversuch.

Die obigen Beschränkungen gelten dagegen nicht für MiL oder SiL. Die Bedeutung von HiL und physikalischen Prototypen für den Entwicklungsprozess sind unbestreitbar. Wegen der genannten Einschränkungen kann es aber nützlich sein, dies verstärkt um SiL und MiL basierte Testumgebungen zu ergänzen. Siehe auch [4] und [5]. Unabhängig von der eingesetzten Umgebung ist die größte Einschränkung beim Systemtest heute die begrenzte Testabdeckung, die mit vernünftigem Arbeitsaufwand erreichbar ist. In SiL/MiL und HiL Umgebungen basieren Ansätze zur Testautomatisierung meist auf handkodierte Testskripten, die bei Ausführung das zu testende System mit einer Sequenz von Eingabewerten stimulieren, einschließlich Programmcode zur Bewertung der gemessenen Systemantwort. Das Schreiben und Debuggen solcher Testskripte ist sehr zeitintensiv. Angesichts der verfügbaren Zeitfenster und des einsetzbaren Arbeitsaufwands können daher oft nur wenige (z. B. ein paar Dutzend) der insgesamt möglichen Testfälle bearbeitet werden. Für den Test am Prüfstand oder im Fahrversuch verschlimmert sich die Situation noch. So ist es z. B. praktisch unmöglich, die Fehlerreaktion eines Systems systematisch für jeden möglichen Bauteilfehler zu testen, wenn die Testumgebung dutzende von physikalischen (d. h. nicht simulierten) Bauteilen enthält.

Wenn man per Testskript das Auftreten oder Fehlen einer bestimmten Systemeigenschaft testet, dann wird die entsprechende Systemeigenschaft in der Regel nur für wenige, speziell dafür konstruierte Testsequenzen verifiziert, nicht aber für alle untersuchten Testfälle.

Beides zusammen bedeutet in der Praxis, dass viele relevante Testfälle während des Systemtests überhaupt nicht betrachtet werden, und das für die betrachteten Testfälle nur einige wenige der relevanten Systemeigenschaften überprüft werden. Dies erhöht die Gefahr, dass Fehler und Schwachstellen alle Systemtests unentdeckt überleben. Dieses Risiko soll durch die hier vorgestellte Testmethode verringert werden. Dadurch erhöht sich die Robustheit des Entwicklungsprozesses insgesamt.

#### **4. Systemanalyse mit TestWeaver**

TestWeaver ist ein Werkzeug für den systematischen, explorativen und weitgehend automatisierten Test komplexer Systeme. Das Verfahren kann zwar im Prinzip auch im HiL eingesetzt werden, wurde aber vor allem für den Einsatz in SiL/MiL Umgebungen konzipiert. Die Entwurfsziele von TestWeaver waren

- dramatische Steigerung der Testabdeckung bezüglich der erreichten Systemzustände
- niedrige Arbeitskosten für den Testingenieur

Um dies zu erreichen, musste vor allem die heute vorherrschende Abhängigkeit des Testprozesses von handgeschriebenen Testskripten reduziert werden, denn die exklusive Verwendung solcher Skripte behindert die gewünschte breite Testabdeckung.

## 4.1 Das Schachspielerprinzip

Die Schlüsselidee hierzu war: Das Testen eines 'Systems under Test' (SUT) ähnelt dem Schachspielen gegen dieses SUT, siehe Bild 1. Der Tester verfolgt dabei das Ziel, das SUT durch geeignete Spielzüge (Testinputs) in einen Zustand zu treiben, in dem das SUT seine Spezifikation verletzt. Hat der Tester eine Sequenz von Zügen gefunden, die das SUT in einen unerwünschten Zustand manövriert, dann hat er das Spiel gewonnen, und die Sequenz von Spielzügen repräsentiert einen fehlgeschlagenen Test.

Es gibt noch mehr Analogien: Um den nächsten Spielzug in einer gegebenen Spielsituation (Systemzustand) zu berechnen, untersuchen Schachcomputer rekursiv sehr viele alternative Spielzüge, die vom aktuellen Zustand aus nach den Regeln des Schachs erlaubt sind und prüfen so, welche Kette möglicher Züge am ehesten zu einem Zielzustand (Matt-Position des Gegners) führt. Dieser Suchprozess erzeugt einen riesigen Baum alternativer, sich verzweigender Spiele. In TestWeaver ist die automatisierte Suche nach Fehlern und Schwachstellen des SUT ganz ähnlich organisiert. Unsere Testmethode setzt voraus, dass das SUT als ausführbare Simulation (MiL/SiL) vorliegt und dass dieses Modell für den Test mit TestWeaver instrumentiert worden ist. Ein Instrument ist eine kleine Modellkomponente, die mit TestWeaver kommuniziert und dabei die Regeln definiert, nach denen TestWeaver gegen das SUT spielt. Die Instrumente repräsentieren zum einen Kontrolleingriffe (mögliche Spielzüge) von TestWeaver, zum anderen informieren sie TestWeaver über den internen Systemzustand des SUTs und ob dieser Zustand erwünscht ist, oder nicht (Mattposition). Jedes Instrument definiert eine Variable mit diskreten (endlich vielen) möglichen Werten. Die Menge aller Instrumente definiert den Raum der möglichen diskreten Systemzustände. Das "Spiel" findet in dem so definierten endlichen Zustandsraum statt.

## 4.2 Instrumente

Ein Instrument ist eine kleine Modellkomponente, die in das uninstrumentierte SUT Modell eingefügt wird. Instrumente sind in der selben Sprache geschrieben, wie das SUT Modell, also z. B. in Matlab/Simulink, Modelica oder C. Die Instrumente eines SUTs kommunizieren während des Tests mit TestWeaver und ermöglichen es TestWeaver dadurch, die SUT Eingänge zu steuern und den SUT Zustand zu überwachen, z. B. um zu entscheiden, ob ein unerwünschter Zustand erreicht wurde. TestWeaver unterstützt dabei zwei Arten von Instrumenten *state reporter* und *action chooser*.

1. *state reporter*: Dieses Instrument beobachtet eine diskrete oder kontinuierliche Variable (z. B. vom Typ double) des SUTs und ordnet jeden Wert der Variable einen Wert aus einer kleinen Menge diskreter Werte zu, z. B. 'niedrig', 'normal', 'zuHoch'. Während eines Test klassifiziert das Instrument also kontinuierlich die beobachtete Variable und meldet jede Änderung des klassifizierten Wertes an TestWeaver. TestWeaver nutzt diese Information, um über die erreichten diskreten Zustände Buch zu führen und so die Abdeckung des diskreten Zustandsraums zu maximieren. Die diskreten Werte werden bei der Instrumentdefinition vom Testingenieur außerdem als 'gut' oder 'verboten' markiert, wobei beliebig viele Zwischenstufen möglich sind. TestWeaver erhält so die Möglichkeit, die Qualität des Verhaltens des SUTs während aller Testläufe zu beurteilen und ins-

besondere das Erreichen von verbotenen Zuständen (Alarm, entspricht einer Mattpositionen) zu erkennen. Siehe Bild 4.

2. *action chooser*: Dieses Instrument ist einer diskreten oder kontinuierlichen Eingangsvariable (input) des SUTs zugeordnet. Im Fall einen Automatikgetriebes sind die Eingangsvariablen z. B. Fahrpedal, Bremspedal oder Wahlhebelstellung, siehe Bild 5. Je nach Art der Instrumentierung fordert das Instrument entweder periodisch und bei Eintreten bestimmter Modellbedingungen von TestWeaver einen neuen Eingabewert an, wobei das Instrument selber eine endliche Menge möglicher Werte zur Auswahl anbietet. Diese Mengen müssen bei der Instrumentdefinition vom Testingenieur vorgegeben werden. Für ein Fahrpedal können das z. B. die Werte 'kein Gas', 'Halbgas' und 'Vollgas' sein. Wie schon bei *state reporters* sind auch hier wieder alle möglichen Werte als 'nominal' oder 'nicht nominal' markiert. Nicht nominale Werte repräsentieren Bauteilfehler. Wählt TestWeaver einen nicht nominalen Wert aus, so injiziert er damit einen Bauteilfehler (component fault) in das SUT, z. B. um die Fehlerreaktion des SUTs zu testen.

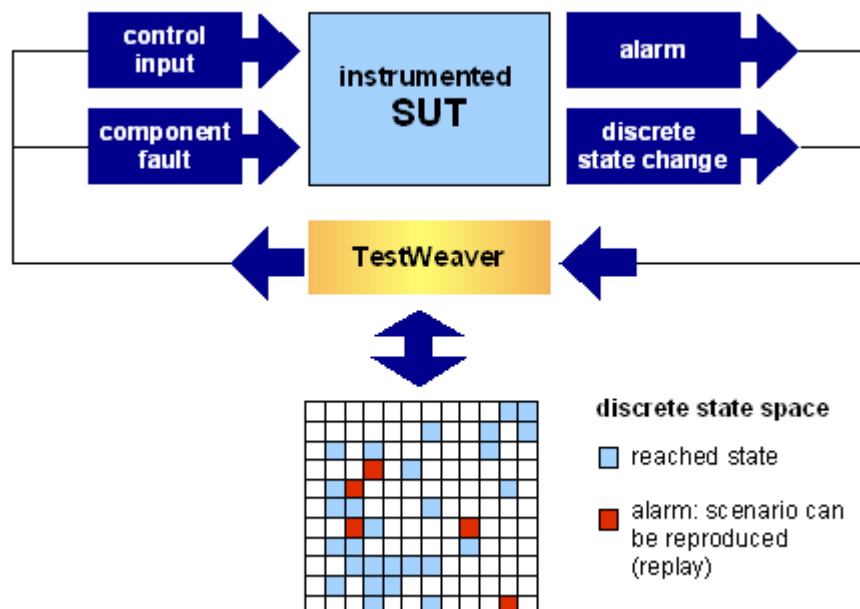


Bild 2: Instrumente verbinden das SUT mit TestWeaver.

TestWeaver wird mit Instrumenten ausgeliefert, um Simulink und Modelica Modelle und C Programme zu instrumentieren. Die Instrumente sind jeweils selber als Simulink Blöcke, Modelica Komponenten oder C Libraries implementiert. Das hat den Vorteil, dass der Testingenieur zur Instrumentierung seines Modells keine neue Modellierungssprache zu lernen braucht und in seiner gewohnten Modellierungs- oder Programmierumgebung arbeiten kann. Neben den *state reporters* eines SUTs überwacht TestWeaver auch den laufenden SUT Prozess und registriert dabei auftretende Probleme wie division-by-zero, Speicherzugriffsverletzungen oder time-outs.

Die Verwendung der TestWeaver Instrumente ist im folgenden am Beispiel Modelica (siehe [www.modelica.org](http://www.modelica.org)) illustriert. Bild 3 zeigt ein Fahrzeugmodell mit Automatikgetriebe, das für TestWeaver instrumentiert wurde. Bild 4 zeigt einen Reporter, der im Modell des Automatikgetriebes angebracht ist, eine Temperaturvariable *frictionHeat*

überwacht und periodisch alle 0.2 Sekunden an TestWeaver berichtet. Entsprechend zeigt Bild 5 zwei *Chooser*, die im Umwelt- und Fahrermodell *world* aus Bild 3 platziert sind und periodisch einmal je Sekunde neue Werte für Brems- und Gaspedalstellung von TestWeaver erfragen.

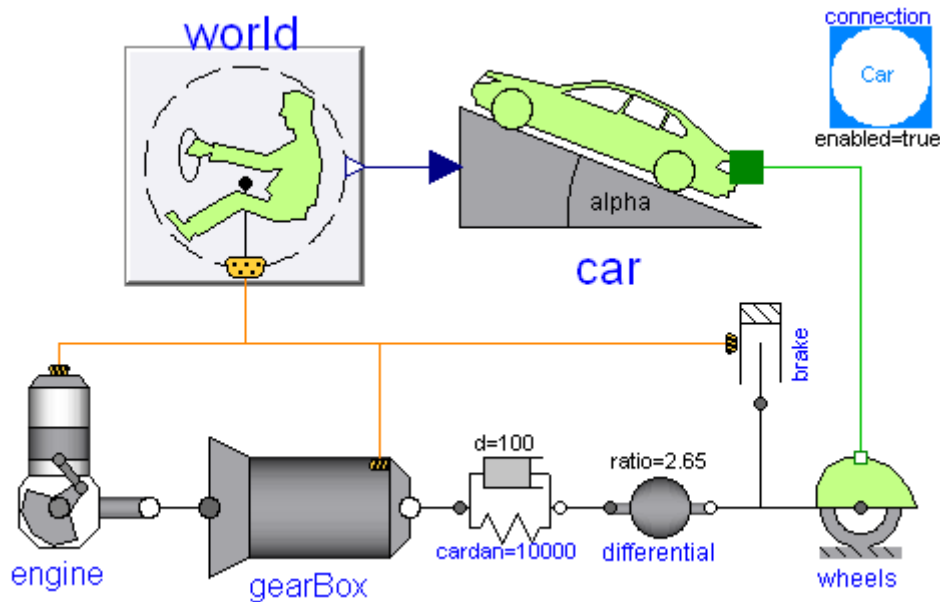


Bild 3: Ein instrumentiertes Fahrzeugmodell (Modelica)

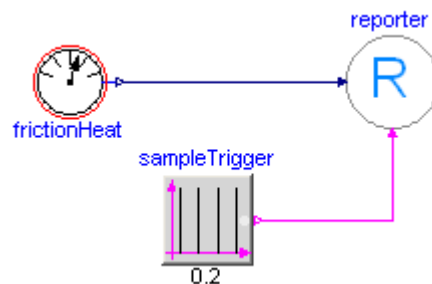


Bild 4: Reporter zum Berichten einer Temperatur

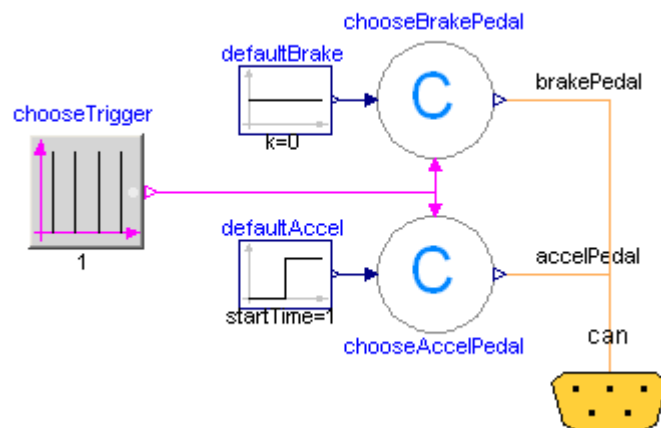


Bild 5: Chooser zum Steuern von Gas- und Bremspedal

### 4.3 Experimente, Szenarien, Reports

Ein TestWeaver *Experiment* ist die Untersuchung und Registrierung aller Systemzustände, die vom SUT in einer bestimmten Zeit erreicht wurden. Dieser Suchprozess berücksichtigt dabei zusätzlich spezifizierte Randbedingungen (z. B. 'nur Anfahrvorgänge') und Überdeckungsziele (z. B. 'mindestens vier Beispiele für jeden Gangwechsel'). Ein Experiment läuft typischerweise völlig autonom, also ohne Eingriffe des Testingenieurs und typischerweise für mehrere Stunden.

Zur Durchführung eines Experiment generiert TestWeaver viele unterschiedliche Szenarien indem er unterschiedliche Antworten auf die Anfragen der *action chooser* des SUTs liefert. Ein Szenario ist ein Protokoll eines Simulationslaufs des SUT im diskreten Zustandsraum. TestWeaver kombiniert verschiedene Suchstrategien um die Abdeckung des diskreten Zustandsraums zu maximieren und um die Wahrscheinlichkeit zu erhöhen, dabei auf Fehler (Alarm eines *state reporters*) zu stoßen. Die während eines Experiments erreichten Zustände werden in einer Szenario Datenbank abgelegt. Die Szenarios eines Experiments bilden einen Baum (eigentlich einen gerichteten Graph) wie in Bild 6 dargestellt. Jeder Pfad in diesem Graph ist ein Szenario.

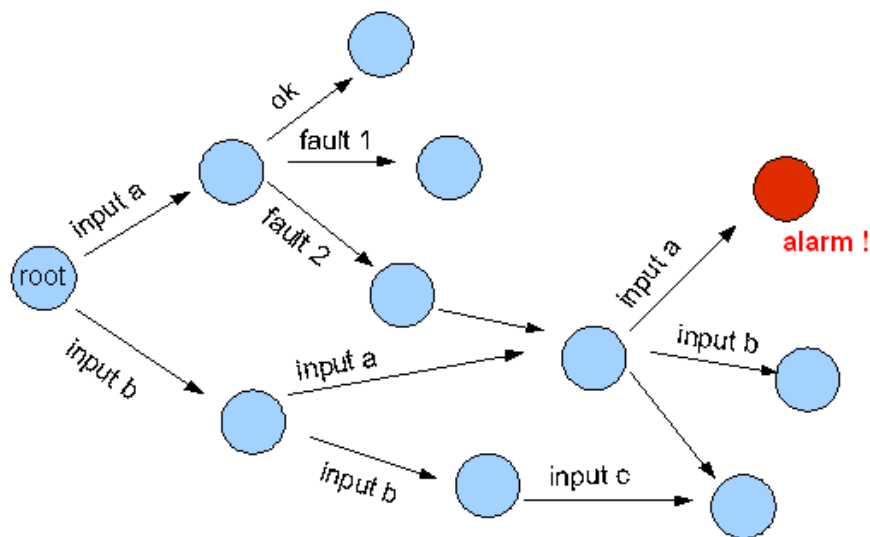


Bild 6: Szenarios, die während eines Experiments erzeugt wurden.

Der Testingenieur kann die erreichten Systemzustände untersuchen. Dazu steht in TestWeaver eine Abfragekommando ähnlich dem SQL (Structured Query Language) Select Statement zur Verfügung. Experimentierergebnisse werden als tabellarische Reports dargestellt. Struktur und Layout solcher Reports werden vom Testingenieur durch Reportvorlagen (Templates) mit eingebetteten Abfragekommandos festgelegt. Bild 7 zeigt einen solchen Report. TestWeaver unterscheidet zwei Arten von Reports: Überblicksreports listen alle untersuchten Szenarios mit einer gegebenen Eigenschaft (z. B. alle Szenarios, in denen ein division-by-zero Ereignis registriert wurde). Szenarioreports listen alle diskreten Zustände eines einzigen Szenarios in chronologischer Reihenfolge.

Der Testingenieur kann in TestWeaver ein Experiment spezifizieren, starten, unterbrechen und fortsetzen. Er kann sich den Inhalt der Datenbank mit Hilfe von Reports anzeigen lassen ohne dabei das laufende Experiment zu unterbrechen.

## 4.4 Experiment Focus

Die Instrumente eines SUT definieren die Achsen des diskreten Zustandsraums (eine Achse pro Instrument) und die Partitionierung der reellen Achsen in diskrete Teilbereiche. Ein instrumentiertes SUT kann in vielen Experimenten benutzt werden, von denen jedes seinen eigenen *Focus* definieren kann. Der Focus eines Experiments gibt an, welcher Teilbereich des Zustandsraums vorrangig im Experiment untersucht werden soll. Während eines Experiments versucht TestWeaver das SUT in den Bereich des Zustandsraums zu treiben, der durch den jeweiligen Experiment Focus beschrieben ist. Der Experiment Focus ist definiert durch:

- *Constraints*: Einschränkungen des betrachteten Zustandsraums. Man kann dadurch z. B. die maximale Dauer eines Szenarios beschränken oder bestimmte Kombinationen von Zustandsgrößen ausschließen. In Fall eines Automatikgetriebes kann man damit z. B. alle Szenarien ausschließen, in denen gleichzeitig gebremst und Gas gegeben wird. Für den Test der Fehlerreaktion kann man z. B. mittels Constraints ausschließen, dass mehr als zwei Bauteilfehler in ein und dasselbe Szenario injiziert werden.
- *Coverage*: Der Testingenieur kann in TestWeaver einen oder mehrere Überblicksreports als Coverage (Abdeckungsziel) definierenden Report markieren. Die Coverage-Reports definieren wichtige Zustandsraumklassifikationen. TestWeaver versucht, alle diese Zustandsklassen zu erreichen. Auf diese Weise kann der Testingenieur das Experiment leicht auf bestimmte Bereiche des Zustandsraums fokussieren, z. B. auf unterschiedliche Anfahrvorgänge oder auf unterschiedliche Gangschaltungen.

TestWeaver ermöglicht außerdem das Ausführen und anschließende Vergleichen von Experimenten mit unterschiedlichen SUT Versionen und Experiment Foci.

## 4.5 Experiment Auswertung und Debugging

Die während eines Experiments gefundenen Fehler und Schwachstellen werden von TestWeaver in Übersichtsreports berichtet. Für jedes gefundene Problem stehen dabei ein oder mehrere Beispielszenarios in der Szenario Datenbank zur Verfügung. Jedes dieser Szenarios ist auf Knopfdruck reproduzierbar (replay). Dazu startet TestWaver das SUT noch einmal mit der für die *action chooser* gespeicherten Eingabe-sequenz. Je nach Laufzeitumgebung des SUTs erlaubt dies das detaillierte Debuggen mit Sourcecode Stepn, Setzen von Breakpoints und Plotten von Signalverläufen.

Bild 7 zeigt einen solchen Übersichtsreport für den Test eines Automatikgetriebes. Die ersten beiden Spalten *currentGear* und *targetGear* zeigen alle während des Experiments erreichten Ist- und Zielgänge. Die Spalten *clutch A* und *clutch B* zeigen die bei diesen Gangwechseln registrierten Temperaturen an Kupplungen A und B. Die Spalte *scenarios* ganz rechts listet für jeden erreichten Systemzustand ein bis zwei Beispielszenarios, die in genau diesen Zustand führen. Bei mindestens einer Rück-schaltung 4-5 wurde z. B. eine Überhitzung an *clutch B* registriert (*heatB=damaged*), und zwar in Szenario *s10*. Klicken auf *s10* öffnet eine Detailansicht des Szenarios und ermöglicht unter anderem das Abspielen des Szenarios in der Stimulationsumgebung. Alle Übersichts- und Szenarioreports sind in TestWeaver über eine Abfragekommando (Query, ähnlich dem SQL Select Statement) konfigurierbar.

TestWeaver - TransmissionTests

File Experiment View Help

C:\users\workspace\et\demos\TransmissionTests\GearShifts B\Scenario.rtl | 0

currentGear	targetGear	clutch A	clutch B	scenarios
1	neutral	ok	ok	<a href="#">s7</a>
	1	ok	ok	<a href="#">s6, s2</a>
	2	ok	ok	<a href="#">s10, s12</a>
2	neutral	ok	ok	<a href="#">s14</a>
	1	ok	ok	<a href="#">s16</a>
	2	ok	ok	<a href="#">s10, s12</a>
3	3	ok	ok	<a href="#">s10, s12</a>
	neutral	ok	ok	<a href="#">s20</a>
	2	ok	ok	<a href="#">s16</a>
4	3	ok	ok	<a href="#">s16</a>
	4	ok	ok	<a href="#">s10, s12</a>
	5	ok	damaged	<a href="#">s10</a>
			ok	<a href="#">s10, s12</a>
5			hot	<a href="#">s10, s22</a>
		hot	hot	<a href="#">s10</a>
	5	ok	ok	<a href="#">s10, s12</a>
6	6	ok	ok	<a href="#">s10, s12</a>
	5	ok	ok	<a href="#">s18, s21</a>
6	6	ok	ok	<a href="#">s12, s17</a>

Console

GearShifts B Scenarios: 22 Simulated time: 0h5m38s Tested time: 0h4m53s Running tir

Bild 7: Ein Übersichtsreport in TestWeaver

## 6. Zusammenfassung und Ausblick

Der zunehmende Druck, Entwicklungszyklen für immer komplexer werdende Produkte noch weiter zu verkürzen und zu verbilligen erfordert den Einsatz neuer Teststrategien. Üblich sind im Automobilbau heute frühe Modultests und relativ späte Systemtests mit HiL, Prüfstand oder im Fahrversuch. Die Bedeutung früher Systemtests

nimmt dabei mit steigender Vernetzung der Module untereinander zu, denn Fehler auf Systemebene werden dadurch wahrscheinlicher, schwerer zu entdecken und teurer zu beheben. Systemtest ohne physikalische Prototypen, also durch Simulation (SiL, MiL), ist dabei eine der Voraussetzungen für frühen Systemtest.

Skriptbasiertes Testen ist eine gangbare Teststrategie solange das gewünschte Systemverhalten durch einfache Stimulus/Antwort Muster einfach beschreibbar ist. Mit zunehmender Systemkomplexität lässt sich aber mit diesem Ansatz die nötige Testabdeckung nicht mehr mit wirtschaftlich vertretbarem Aufwand (Arbeitskosten) erzielen. Das hier vorgestellte Testverfahren ermöglicht dagegen

- die systematische Untersuchung großer Zustandsräume mit geringen Arbeitskosten: nur die 'Spielregeln' müssen vom Testingenieur definiert werden, nicht die individuellen Testszenarien
- das Entdecken neuer, nie zuvor bedachter Szenarien. Dies ist ein wichtiger Unterschied zu skriptbasiertem Testen. TestWeaver erzeugt tausende qualitativ unterschiedliche Szenarien, einschließlich "exotischer" Szenarien, die wahrscheinlich ein Testingenieur so nicht definieren würde
- die Wahrscheinlichkeit zu steigern, dass alle Schwachstellen in einem Produkt rechtzeitig entdeckt werden.

TestWeaver wird derzeit in mehreren Serienentwicklungsprojekten im Automobilbereich verwendet. Beispielsweise wurde TestWeaver während der gesamten Entwicklung des *AMG SPEEDSHIFT MCT 7-Gang-Sportgetriebes* von Mercedes-Benz eingesetzt. Dabei wurde jeder neue Softwarestand der Getriebesteuerung auf mehreren PCs für jeweils 24 Stunden getestet. Neben dem Einsatz für Automatikgetriebe sind viele weitere Einsatzgebiete aussichtsreich, insbesondere dort wo eine komplexe Interaktion zwischen Regelsoftware und der physikalischen Welt auftritt, wie zum Beispiel:

- *Fahrerassistenzsysteme*: Systeme wie ABS und ESP führen zu einem komplexen Zusammenwirken von Steuergerätsoftware, Fahrer, Fahrdynamik und anderen Assistenzsystemen. Hier ergibt sich eine Unzahl relevanter Fahrerszenarien, die entwicklungsbegleitend untersucht werden sollten.
- *Prozesssteuerung*: Im Anlagenbau, zum Beispiel, in Kraftwerken und chemischen Produktionsprozessen findet man eine ähnlich komplexe Interaktion zwischen Steuersoftware, dem menschlichen Operateur und der physikalischen Anlage. Auch hier möchte man vor Inbetriebnahme einer Anlage sehr viele unterschiedliche Betriebssituationen systematisch untersuchen.

TestWeaver läuft unter Windows auf herkömmliche PCs. Es ist ein mächtiges und dennoch relativ einfach zu benutzendes Werkzeug. TestWeaver erfordert lediglich die Instrumentierung eines Modells des zu untersuchenden Systems. Dazu muss der Ingenieur keine neue Instrumentierungssprache erlernen, sondern kann die Sprache verwenden, in der er Teile des zu testende System implementiert hat, zum Beispiel Simulink, C oder Modelica.

## Literatur

- [1] Berard et. al.: Systems and Software Verification: Model-Checking Techniques and Tools, Springer Verlag, 2001.
- [2] M. Grochtmann, K. Grimm: Classification Trees for Partition Testing - In: Software Testing, Verification & Reliability, Volume 3, No 2, pp. 63-82, 1993.

- [3] Meyer, Bertrand: Applying "Design by Contract" - In: Computer (IEEE), 25, 10, October, pages 40-51, 1992.
- [4] S. Rebeschies., Th. Liebezeit, U. Bazarsuren, C. Gühmann: Automatisierter Closed-Loop-Testprozess für Steuergerätefunktionen - In: ATZ elektronik, 1/2007.
- [5] S. Thomke: Experimentation Matters: Unlocking the Potential of New Technologies, Harvard Business School Press, 2003.
- [6] A. Junghanns, J. Mauss, M. Tatar: TestWeaver - A Tool for Simulation-based Test of Mechatronic Designs -In: Proceedings of the 6th International Modelica Conference, 3. - 4.3.2008, Bielefeld.