AutoTest 2008 - Test und Diagnose von Hard- und Software in der Automobilentwicklung, 22. - 23. Oktober 2008.

TestWeaver
Funktionstest nach dem Schachspielerprinzip

Andreas Junghanns, Jakob Mauss, Mugur Tatar

QTronic GmbH
Alt-Moabit 91d, 10559 Berlin
{andreas.junghanns, jakob.mauss, mugur.tatar}@qtronic.de

Abstract: Durch das komplexe Zusammenspiel von Funktionssoftware mit mechanischen, hydraulischen und elektronischen Bauteilen entsteht eine neuartige, für Entwicklungsingenieure schwer beherrschbare Komplexität. 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 korrigiert 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 in diesem Beitrag ein Verfahren vor, das in vielen Anwendungsfällen hilft, diesen Spagat zu schaffen: Das Verfahren kombiniert klassische Simulation bzw. Co-Simulation (MiL, SiL) mit einer Methode zum automatischen Erzeugen, Durchführen und Auswerten von Tests. Damit lassen sich in kurzer Zeit zehntausende von Szenarien vollautomatisiert testen, ohne das hierfür Test- oder Auswerteskripte zu schreiben sind. Das entsprechende Testwerkzeug TestWeaver wird zur Zeit in mehreren Serienentwicklungsprojekten der Automobilindustrie erfolgreich eingesetzt.

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 entwicklungsbegeleitende Systemtest mit begrenzten Resourcen (Zeitfenster und Arbeitskosten) ist daher für die Entwicklungsteams mechatronischer Systeme eine große Herausforderung.

Wir stellen in diesem Beitrag ein Werkzeug vor, das den Prozess des entwicklungsbegleitenden Testens weitgehend automatisiert (Abb. 1). Dadurch kann die Testabdeckung dramatisch gesteigert werden ohne dabei gleichzeitig die Arbeitsbelastung für die Testingenieure zu erhöhen.

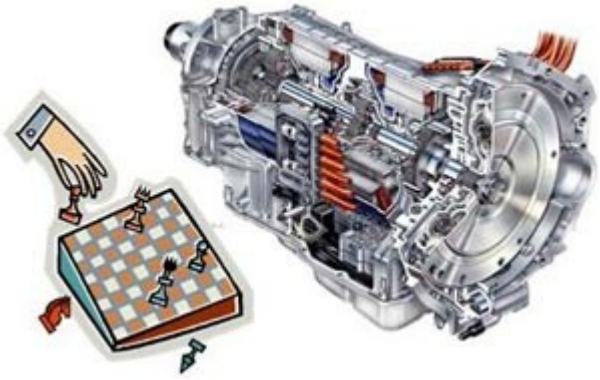


Abbildung 1: Testen als Spiel gegen das simulierte System

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 und Abschnitt 5 mögliche Realisierungen des benötigten Simulationsmodells. 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°C bis 40°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 bittet 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, [MC01]) 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 (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 im HiL, am Prüfstand oder im Fahrversuch vergleichsweise schwierig wenn nicht unmöglich,.

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 [SiL07], [Silver] und [EM03].

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 handkodierten 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 zeit intensiv. 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 Systemantwort 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, das 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 hinter TestWeaver ist: Das Testen eines Systems ähnelt einem Schachspiel gegen dieses System (Abb. 1). Ziel des Testers ist es, das zu testende System durch eine passend gewählte Sequenz von Spielzügen in einen Zustand zu treiben, in dem es seine Spezifikation verletzt.

Ein Spielzug des Testers ist dabei jeweils ein diskreter Eingriff in eine steuerbare Größe des Systems, im Fahrzeug z. B. Fahrpedal, Bremspedal, Wahlhebel. Das System antwortet auf einen solchen Spielzug deterministisch, wobei die Antwort durch Simulation ermittelt wird.

Auf diese Weise generiert TestWeaver (zum Beispiel über Nacht) zehntausende von qualitativ verschiedenen Simulationsläufen, so genannte Szenarien. Es werden also keine handgeschrieben Tests benötigt. Stattdessen wird das Simulationsmodell des Systems für den Test mit TestWeaver instrumentiert. Dazu müssen kleine Modellkomponenten definiert werden, siehe Abb. 2. Diese so genannten Instrumente ermöglichen TestWeaver die Steuerung des Modells (*action chooser*, links in Abb. 2 und abb. 4) und die Überwachung der Testabdeckung (*state reporter*, rechts in Abb. 2 und Abb. 5).

TestWeaver generiert nicht nur selbstständig Szenarien und führt diese aus, sondern bewertet auch das resultierende Systemverhalten. Dazu dienen so genannte Alarminstrumente. Dies sind spezielle state reporter, die das Systemverhalten kontinuierlich klassifizieren und bei der Verletzung (oder beinahe Verletzung) von Qualitäts- und anderen Kriterien Alarm auslösen, z. B. bei Überhitzung von Bauteilen oder Division durch Null in der Funktionssoftware.

TestWeaver generiert die Szenarien nicht nach dem Monte Carlo Prinzip - dies ist aus kombinatorischen Gründen nicht praktikabel - sondern reaktiv, in informierter, zielgerichteter Weise. TestWeaver analysiert dazu das Systemverhalten für alle bereits generierten Szenarien.

TestWeaver verfolgt dabei zwei Ziele

- Szenarien zu konstruieren, die einen Alarm auslösen, z. B. durch systematisches variieren von bereits suboptimalen Szenarien
- die Maximierung der Testabdeckung.

Testabdeckung ist dabei wie folgt definiert: Jedes Instrument (action chooser und state reporter) zerlegt den Wertebereich der von ihm kontrollierten oder beobachteten Variable in eine kleine Menge von Intervallen bzw. diskrete Wertebereiche, so genannte *Partitionen*. Die Instrumente eines Systemmodells spannen so einen n-dimensionalen diskreten (also endlichen) Zustandsraum auf. Abb 2 zeigt unten ein Beispiel für den Fall $n = 2$. Das Überdeckungsziel von TestWeaver ist es, jeden erreichbaren diskreten Zustand in diesem Raum mindestens einmal zu erreichen.

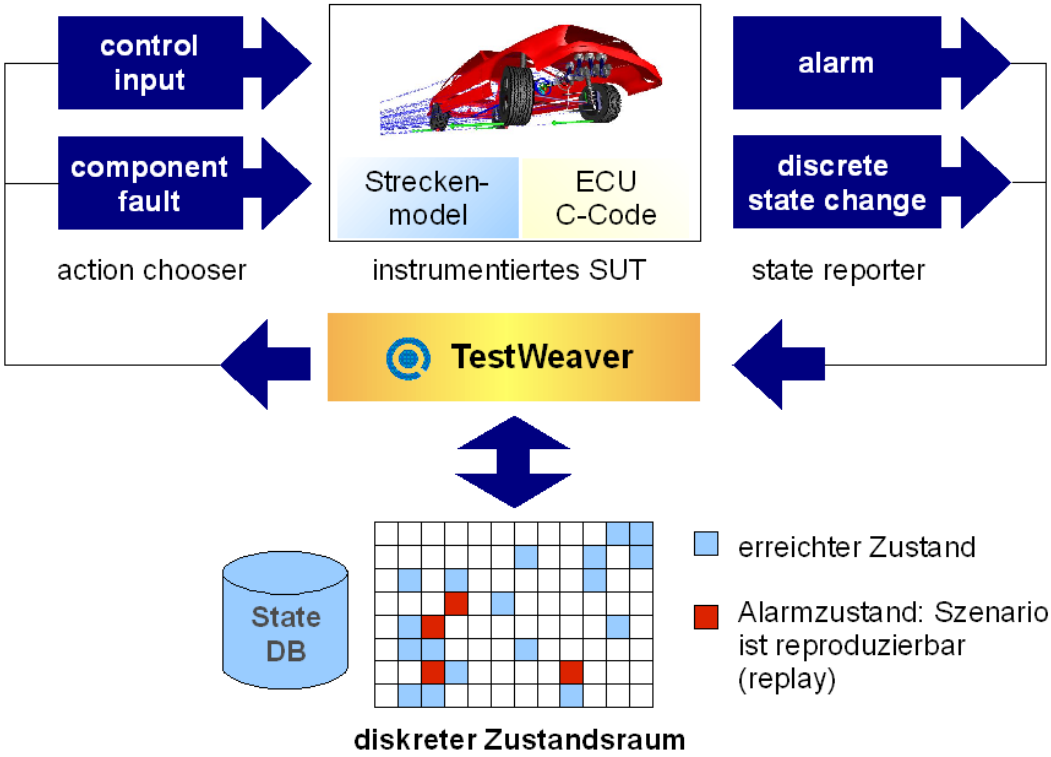


Abbildung 2: Instrumente verbinden das SUT mit TestWeaver.

TestWeaver wird mit Instrumenten ausgeliefert, um Simulink und Modelica Modelle sowie Python und C Programme zu instrumentieren. Die Instrumente sind jeweils selber als Simulink Blöcke, Modelica Komponenten, bzw. C oder Python Funktionen 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 reportern 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) illustriert. Abb. 3 zeigt ein Fahrzeugmodell mit Automatikgetriebe, das für TestWeaver instrumentiert wurde.

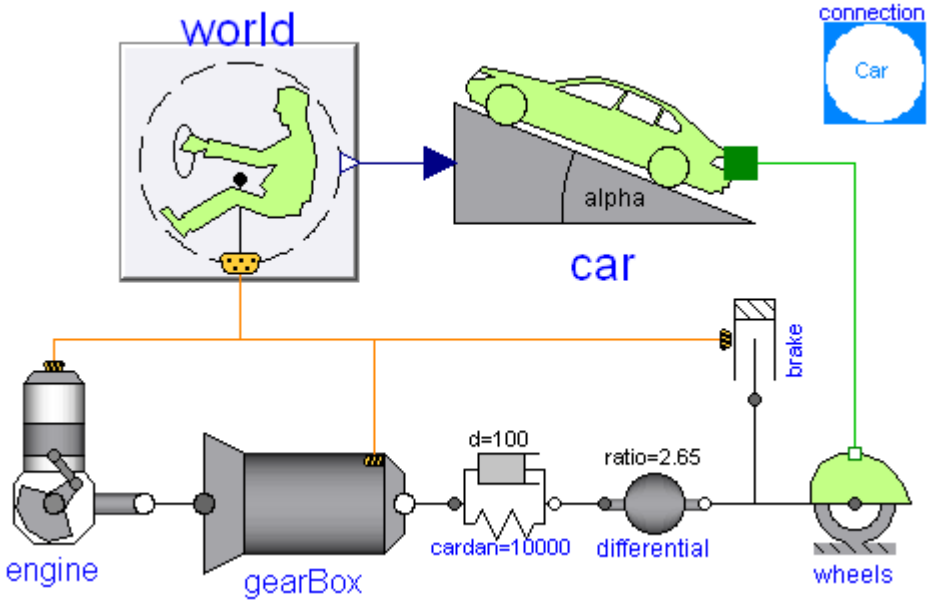


Abbildung 3: Ein instrumentiertes Fahrzeugmodell (Modelica)

Abb. 4 zeigt einen Reporter, der im *gearBox* Modell des Automatikgetriebes angebracht ist, eine Temperaturvariable frictionHeat überwacht und periodisch alle 0.2 Sekunden an TestWeaver berichtet.

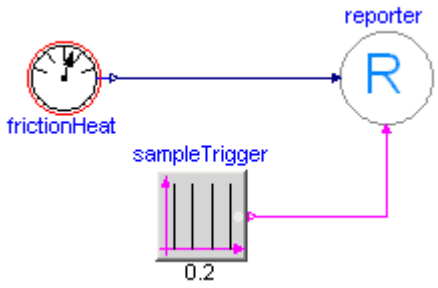


Abbildung 4: Reporter zum Berichten einer Temperatur

Entsprechend zeigt Abb. 5 zwei Chooser, die im Umwelt- und Fahrermodell *world* aus Abb. 3 platziert sind und periodisch einmal je Sekunde neue Werte für Brems- und Gaspedalstellung von TestWeaver erfragen.

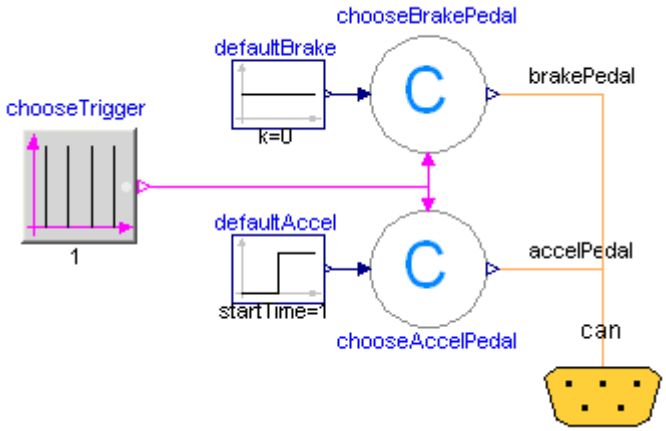


Abbildung 5: Zwei Chooser zum Steuern von Gas- und Bremspedal

4.2 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.

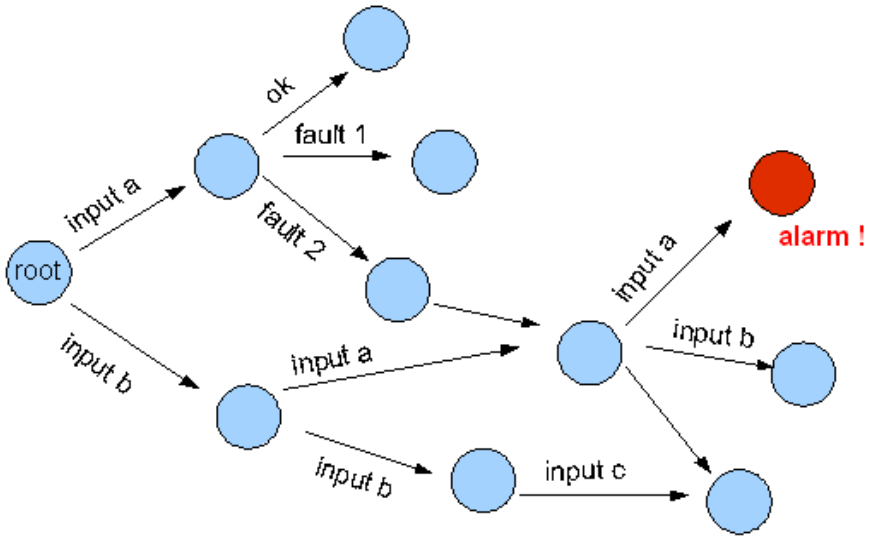


Abbildung 6: Szenarios, die während eines Experiments erzeugt wurden.

Zur Durchführung eines Experiments 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 Abb. 6 dargestellt. Jeder Pfad in diesem Graph ist ein Szenario.

Der Testingenieur kann die erreichten Systemzustände untersuchen. Dazu steht in TestWeaver eine Abfragekommando ähnlich dem SQL (Structured Query Language) Select Statement zur Verfügung. Experimentergebnisse werden als tabellarische Reports dargestellt. Struktur und Layout solcher Reports werden vom Testingenieur durch Reportvorlagen (Templates) mit eingebetteten Abfragekommandos festgelegt. Abb. 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.

4.3 Experiment Fokus

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 Fokus definieren kann. Der Fokus 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 Fokus beschrieben ist. Der Experiment Fokus 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. TestWeaver versucht, alle Zustände, die in einem Coveragereport berichtet werden, bevorzugt zu erreichen, d. h. möglichst viele Einträge in Coveragereports zu provozieren. 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 Foki.

4.4 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 Eingabesequenz.

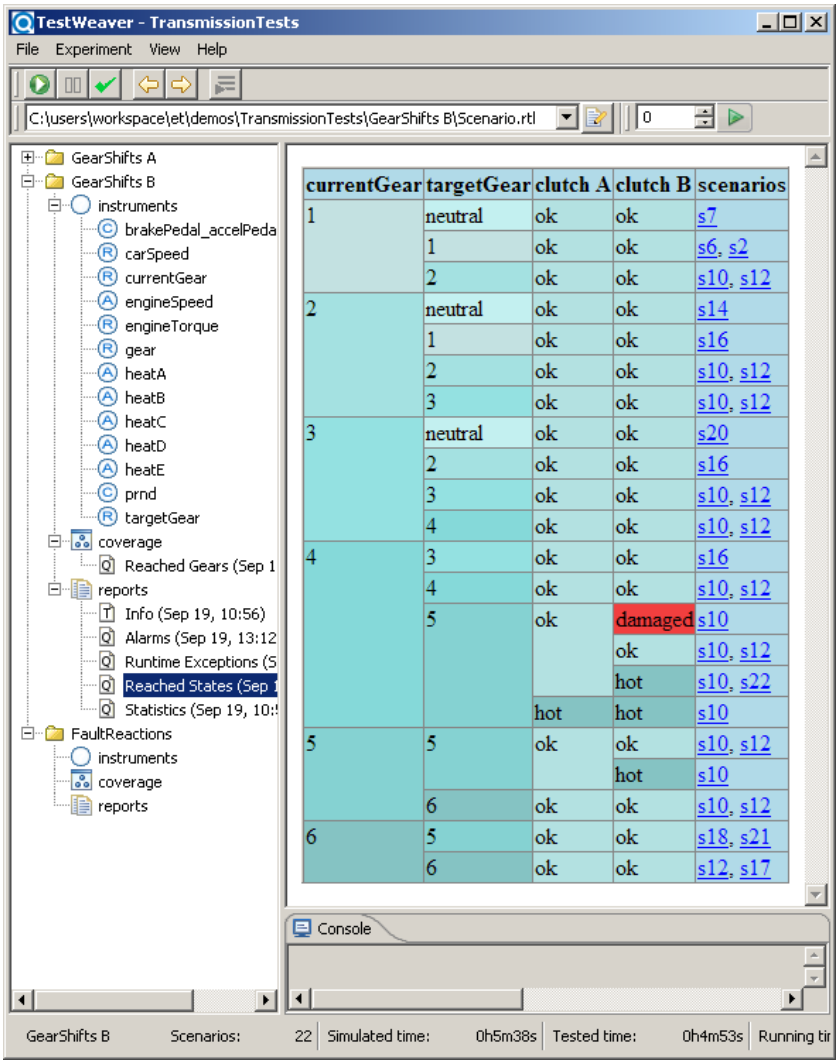
Je nach Laufzeitumgebung des SUTs erlaubt dies das detaillierte Debuggen mit Sourcecode Stepern, Setzen von Breakpoints und Plotten von Signalverläufen.

Abb. 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 Rückschaltung 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.

Der Überblicksreport aus Abb. 7 ist z.B. durch folgendes Kommando spezifiziert

```
select currentGear, targetGear, clutchA, clutchB, set(2, scenarios)
from States
group by currentGear, targetGear, clutchA, clutchB;
```

Die Farben der Tabellenzellen werden von TestWeaver automatsch vergeben, wobei rot Alarmzustände markiert und die Farbskala blau-grün nominale Zustände. Ein Report kann editierbare Kommentarspalten enthalten, in denen der Testingenieur seine Einschätzung der Relevanz von Alarmen und ggf. Folgemaßnahmen vermerken kann. Dadurch werden aufgetretene Probleme datenbankgestützt dokumentiert und bis zu ihrer Behebung verfolgbar.



currentGear	targetGear	clutch A	clutch B	scenarios
1	neutral	ok	ok	s7
	1	ok	ok	s6, s2
	2	ok	ok	s10, s12
2	neutral	ok	ok	s14
	1	ok	ok	s16
	2	ok	ok	s10, s12
	3	ok	ok	s10, s12
3	neutral	ok	ok	s20
	2	ok	ok	s16
	3	ok	ok	s10, s12
	4	ok	ok	s10, s12
4	3	ok	ok	s16
	4	ok	ok	s10, s12
	5	ok	damaged	s10
			ok	s10, s12
			hot	s10, s22
		hot	hot	s10
5	5	ok	ok	s10, s12
			hot	s10
	6	ok	ok	s10, s12
6	5	ok	ok	s18, s21
	6	ok	ok	s12, s17

Abbildung 7: Ein Übersichtsreport in TestWeaver

5 Realisierung des Simulationsmodells

Für den Test mit TestWeaver muss ein Simulationsmodell des zu testenden Systems (SUT) vorliegen. Solche Modelle werden im Rahmen des inzwischen weitgehend etablierten modellbasierten Entwicklungsprozesses sowieso entwickelt, sodass in diesem Fall keine zusätzliche Kosten für die Modellerstellung anfallen. TestWeaver kommuniziert mit dem SUT ausschließlich über die in 4.2 beschriebenen Instrumente. Dies vereinfacht die Anbindung von TestWeaver an beliebige Simulationsumgebungen erheblich. Konkret wurde TestWeaver so bisher an folgende Simulations- und Programierumgebungen angschlossen: Visual C/C++, (Microsoft), Python 2.5, Matlab/Simulink mit RealTime Workshop R2006 (MathWorks) oder TargetLink (dSpace), Modelica/Dymola 6.x (Dynasim), Modelica/SimulationX 3.1 (ITI). Eine Anbindung an Simpack (Intec) ist in Vorbereitung.

Das SUT kann auch als Co-Simulation mehrerer, mit unterschiedlichen Werkzeugen erstellten Teilmodellen realisiert werden. Zum Beispiel kann das Streckenmodell mit Modelica erstellt werden, die Funktionssoftware modellbasiert mit Matlab/Simulink und die Instrumentierung für TestWeaver mit einem Python Script. Als Ausführungsumgebung für ein solches verteilt entwickeltes Systemmodell kann Silver eingesetzt werden, siehe [Silver]. Dazu werden alle Teilmodelle aus ihren jeweiligen Entwicklungsumgebungen exportiert und liegen als kompilierte, selbstintegrierende Module (DLLs) vor. Diese Module werden dann von Silver per Co-Simulation zyklisch, z. B. im 10 ms Takt ausgeführt, wobei die Module zur Laufzeit Signale miteinander austauschen. Auf diese Weise kann das Zusammenspiel aller beteiligten Teilsysteme 'virtuell' geprüft werden.

Die Vorteile der Realisierung des Simulationsmodells mit Silver sind neben den in Abschnitt 3 genannten:

- Austausch von Teilmodellen ist ohne Weitergabe der Quellen möglich. Dies erleichtert die Zusammenarbeit zwischen OEMs und Zulieferern (Schutz von IP).
- Kompilierte Module laufen in der Regel sehr schnell ab, viel schneller als interpretierte Teilmodelle, z. B. in Matlab/Simulink. Wichtig, weil TestWeaver während eines Experiments sehr viele Simulationen durchführt.
- Module können mit verschiedenen Werkzeugen entwickelt werden. So können die für ein Teilmodell jeweils optimalen oder aus anderen Gründen bevorzugten Werkzeuge eingesetzt werden.
- Integration der Fahrzeug Applikationsparameter durch Anbindung der Datenbankformate ASAP2/A2L (ASAM) und DCM (ETAS), des Messdatenformats MDF und Unterstützung der Applikationsprotokolle CCP/XCP (ASAM). So können auch die Applikationsdaten mit TestWeaver getestet werden und an Silver dieselben Werkzeuge (z. B. CANape) angeschlossen werden, die auch im Fahrzeug zum Messen und Applizieren eingesetzt werden.
- Mächtige Test und Debuggingmöglichkeiten: Anbindung von Werkzeuge zur Testautomatisierung, Anbindung von Debuggern, z. B. JIT (just in time) debugging von Microsoft Visual C++, um Exceptions in der Funktionssoftware zu fangen und per Stepper im entsprechenden C Code zu analysieren.
- eingebaute TestWeaver Instrumente: Silver verfügt über fest eingebaute Instrumente zum Fangen und Berichten von Modellproblemen (Exceptions). Dabei wird z.B. die fehlerhafte Zeile im Programmcode automatisch ermittelt und zusätzliche Information zur Exception protokolliert.

Abb. 8 zeigt eine Fahrzeugsimulation in Silver. Links sind die Module und ihrer Variablen zu sehen, rechts die konfigurierbare Bedienschnittstelle, über die man das virtuelle Fahrzeug starten und fahren kann.

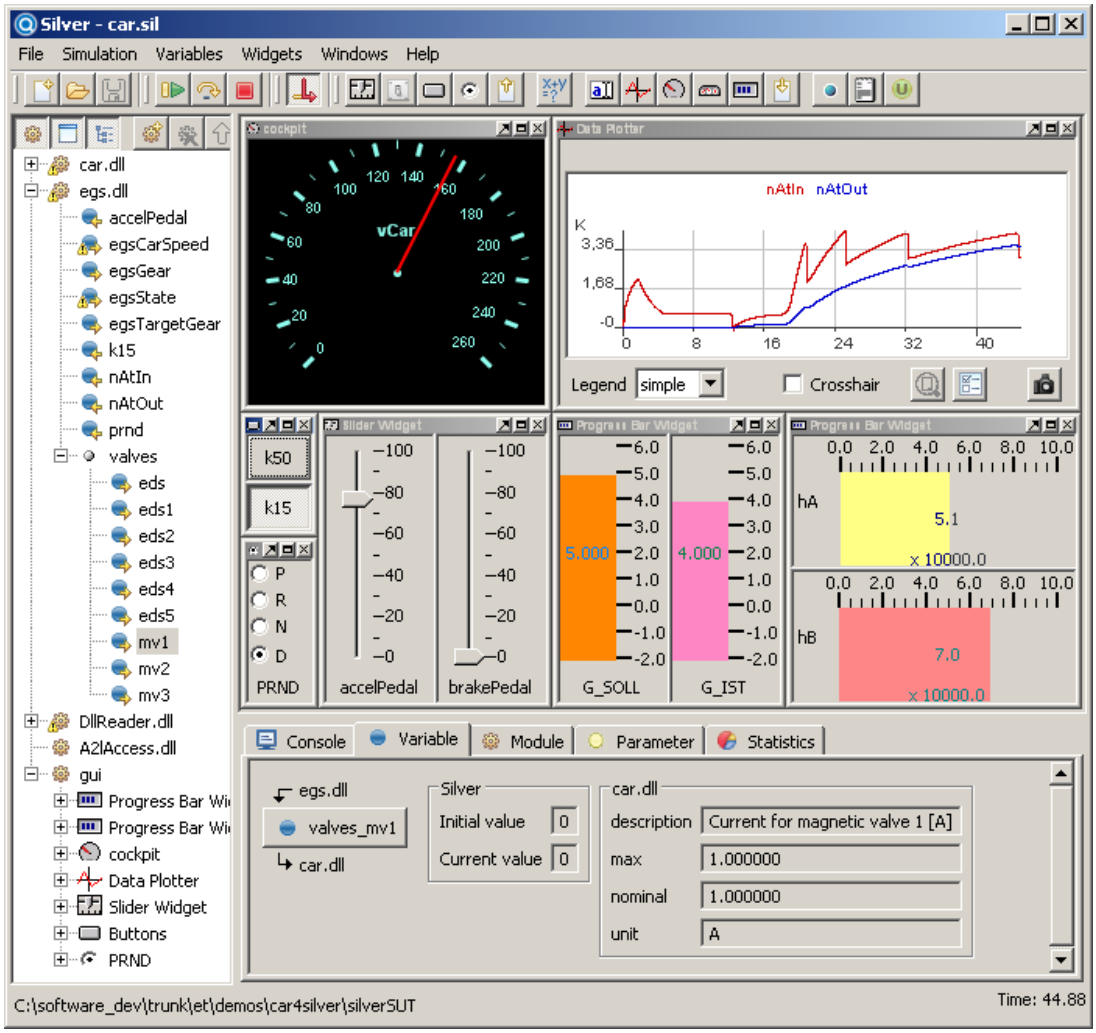


Abbildung 8: Co-Simulation mit Silver

Eine solche Simulation kann direkt mit TestWeaver getestet werden. Während eines solchen Experiments ist Silver's grafische Benutzerschnittstelle (Plotter, Anzeigen) deaktiviert, um Rechenzeit zu sparen.

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. 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 Steuergerätesoftware, Fahrer, Fahrdynamik und anderen Assistenzsystemen. Hier ergibt sich eine Unzahl relevanter Fahrszenarien, die entwicklungsbegleitend untersucht werden sollten. [FAS08] berichtet über den Einsatz von TestWeaver bei der Entwicklung eines Bremsassistenten.
- *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 herkömmlichen 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.

7 Literaturverzeichnis

[MC01] Berard et. al.: Systems and Software Verification: Model-Checking Techniques and Tools, Springer Verlag, 2001.

[EM03] S. Thomke: Experimentation Matters: Unlocking the Potential of New Technologies, Harvard Business School Press, 2003.

[SiL07] S. Rebeschieß., Th. Liebezeit, U. Bazarsuren, C. Gühmann: Automatisierter Closed-Loop-Testprozess für Steuergerätefunktionen - In: ATZ elektronik, 1/2007.

[Silver] Silver 1.0 - Software in the Loop für effiziente Funktionsentwicklung. http://www.qtronic.de/doc/Silver.pdf

[TW08] 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.

[FAS08] M. Gäfvert, J. Hultén, J. Andreasson, A. Junghanns, J. Mauss, M. Tatar: Simulation-Based Automated Verification of Safety-Critical Chassis-Control Systems. 9th International Symposium on Advanced Vehicle Control (AVEC2008), Kobe, Japan, 6. - 9.10.2008.