コンピュータチェスの考えを応用したテスト自動化

Dr. Andreas Junghanns, Dr. Jakob Mauss, Dr. Mugur Tatar
QTronic GmbH, Alt-Moabit 91d, D-10559 Berlin
{andreas.junghanns, jakob.mauss, mugur.tatar}@qtronic.de

（本資料は QTronic 社の許可を得て、株式会社エーアイコーポレーションが翻訳しました）

摘要

自動車のトランスミッションシステムは、メカニカル、油圧系統、電気回路、トランスミッション等の制御ソフトウェアの複雑な相互作用があるため、テストや検証は困難を極める。増加し続けているソフトウェア機能とハードウェアのサブシステムの相互関係により、新たな複雑さが発生し、メカトロニクスの設計の管理が難しくなってきている。システムテストは膨大なテストケースを行なう必要がある。時間とコスト等が限られたリソースで多くの検証しなければならないのは開発チームにとってチャレンジングな状況である。我々は複雑なトランスミッションシステムのテストと検証を行っている開発者を支援する、インスツルメントを使った新たな手法を紹介する。TestWeaver はテストエンジニアが指定するテストケースに基づき、テストにかかる工数をおさえつつテスト網羅性を最大にする新たなアプローチである。この手法はシミュレーション MiL（Model in the Loop） や SiL（Software in the Loop）に統合され、テストケースを自動的に、生成、検証することで、既に自動車産業のアプリケーションで成功している。本事例では 6 速のオートマチックトランスミッションを用いる。

1　序論

ハイブリッドドライブトレーンや自動車用オートマチックトランスミッションなどの複雑なメカトロシステムを開発するためには、異なった分野の設計や開発の協力が必要であり、また組織にまたがって連携する必要があるため、複雑な設計プロセスとなる。結果として、開発中に、設計ミスやコーディングエラーが起こることは避けられないものとなっている。OEM（自動車メーカー）にとってこれら全てのバグや欠陥を時間内、すなわち生産工程や、顧客に納品する前に見つけて、取り除くことはとても重要である。そこを失敗すると、高価なリコールや補償のコスト、顧客の不満につながる。OEM は 40%以上の開発予算をテストやそれに関連した活動に費やしている。ソフトウェアは新たな機能の追加に非常に柔軟だが、見つけにくい不具合を含んでしまう可能性もある。しかもソフトウェアと物理システムの相互作用による複雑な振る舞いを、完全に解析し検証することはできない。多くは、制限された、いくつかのポイントを実車、もしくは仮想的に評価するのみである。開発チームはシステムテストで多くの関連した膨大なテストケースの空間（スペース）を網羅する必要がある一方、（時間やコストなどの）リソースは限られたものになっているというジレンマに直面している。

本論文はテストエンジニアの負担を増やすことなく、劇的にカバレッジを増やす可能性のある新たなテスト手法を提案している。我々はテスト結果の自動チェックを含む何千ものテストを自動で生成することを達成した。テスト生成は特定のステート空間、制約の使用、カバレッジ目標にフォーカスすることが可能である。

本論文は、以下の構成となっている。2 章でメカトロニクスシステムのテストの概要を述べ、3 章で自動車産業での今日の主なテスト手法を調査した結果を述べ、4 章でテスト手法を提唱する。5 章で乗用車の 6 速オートマチックトランスミッションの事例を示す。6 章で要求されたシミュレーションモデルでの実装オプションを述べ、結論で我々のテスト手法の利点、その他のエンジニアリングドメインへの適用について述べる。

2　テストの難しさ

メカトロニックシステムをテストするとき、一般的には実験室内の条件でいくつかの理想的なユースケースをテストするだけでは不十分である。潜んでいる全てのバグ、設計ミスを発見するチャンスを増やすために、システムは異なったなるべく多くの、関連した条件をテストされるべきである。一例として乗用車のオートマチックトランスミッション制御の例を考察する。

この場合、以下のような観点でテストを行なう必要がある。

- 天候：例えば、気温が－40 度から 40 度まで変化すると、油圧システムの油温に大きな影響を及ぼす。
- 道路：路面の性質、上り、下り、カーブ、異なった摩擦係数
- ドライバー：車の運転方法、振る舞い、癖のある運転方法
- 突然の部品の故障：運転中に部品はいつでも壊れる可能性がある。運転者にとって安全で、危険を避けるために、制御ソフトウェアはそれらを検出し、適切に反応する必要がある。
- 生産時の誤差：生産プロセスによってメカ、エレクトロニクス等の関連するコンポーネントの物理的な特性は異なる。
- 劣化：部品の経年変化によって特性は変わってくる。
- 部品間での相互作用：トランスミッションは分散した機能を他のパーツ（エンジン、ブレーキシステム）とネットワークを介して通信することにより実現する。例えば、ギアシフトの間、トランスミッションはエンジンにスイッチングする部品を守るためにトルクを減らすよう指示する。

このように一つの部品でも可能性のある操作の条件は莫大なスペースに広がる。複数の部品だと各々の次元の可能性が積算されるのでさらに膨大になる。それらのスペースの各々の全てを適切にシステム上で検証することがテストの究極の目的である。数学的にシステムのプロパティを検証（例えば適切でない振る舞いが無いこと）できれば素晴らしい。それがあればテストエンジニアは無限の多くのケースをワンステップで実現できる。しかしそういった証明のテクニック（たとえば、モデルチェッキング）はここで考察しているシステムレベルのテストの複雑さに関して非常に限られた範囲のことしかできない。現実的には全ての状態スペースを網羅するという目的は、有限のスペースのテストケースに近似することにより実現されている。

3　既存のテスト手法の限界

（コンポーネント、モジュール、システム、実車）などの異なった機能結合レベルのテストや MiL、SiL、HiL（Hardware in the Loop）、実車プロトタイプ等の異なったテストは現在重要で不可欠な開発プロセスである。早期に問題が発見され修正される。しかしながら、

- 関連したテストのみシステムレベルで繰り返し実行されるのみである。例えば、部品の故障が起こった場合のシステムの反応など。
- システムレベルのテストは HiL でのみ、もしくはプロトタイプ時のみ行われる

後者になりがちな理由はいくつかある。ひとつの重要な例はメカトロニック開発プロセスの複雑さからである。いくつかの部門、チーム、ツール、サプライヤーが共同作業をする上で、標準が欠如していたり、実行可能な機能モデルの結合と交換ができなかったりするためである。

簡潔に HiL と実車プロトタイプベースのテストについての幾つかの限界を以下に述べる。

- 時間、コスト、安全：実車のプロトタイプや HiL ではセットアップにコストがかかり、多くのリソースを必要とする。テストは開発サイクルの後期のフェーズであり、数多くのテストを実行することはできない。特定のコンポーネントの故障を起こすような異常系のテストは安全面から、実車プロトタイプではテストできない。
- アジャイルさの欠如：一般的にソフトウェアの機能の変更を行ってからテストで確認するまでに時間がかかる。
- 正確さの制限：HiL では物理システムモジュールのリアルタイム性が要求されるので、設定はかなり単純化され、正確さが損なわれる。正確な設定無しでは、隠れたシステムの特性のデバッグや検査は難しい。

上記の制限は MiL、SiL のセットアップには表れないことに注意。HiL と実車プロトタイプのテストは重要で過小評価するべきではないが、我々の主張は、システムレベルのテストは MiL や SiL により重きを置くことにより、上記の制限を補完できるということである。[2], [3].参照

使われるテスト手法に関係なく、システムレベルのテストにおいて共通の限界は労力に対して、得られる網羅度に限界があることである。たとえば、HiL、HiL でのテスト自動化は、基本的にハンドコーディングによるテストスクリプトであり、テストは部品に対してのシーケンス的な入力である。それらは反応を測定して検証するコードも含む。それらのテストスクリプトのコーディングとデバッグは相応の時間を要する。時間とマンパワーの制約があるので、スクリプトベースのアプローチでは膨大な可能性のあるユースケースのスペースのうちのわずかな（言ってみれば数ダースほど）のケースしか網羅できない。車を道路で運転することによるテストにおいてこの数はさらに悪くなる。例えば、一つのコンポーネントの欠陥が起こった場合や複数の部品が故障を起こした場合の応答のテストなどをシステマチックに行うことは不可能に近い。例えばシステムの特定のまれにしか起こらない特性のテスト（例えば、クラッチはギアシフト中に温度が高くなりすぎてはいけない等）の有無をスクリプトベースで、設計されたシナリオだけで、数多くのテストを行わない場合、発見するのは困難である。実際には、多くのシナリオはシステムテストでは決して探索できず、わずかな関連したシステムの特性しかテストされない。結果としてバグや、設計の欠陥は全てのテストを行っても多くは生き残る。これらのリスクはここで提唱する手法によって減少し、さらにデザインプロセスに堅牢性を加える。

4　TestWeaver によるシステムの振舞いの調査

TestWeaver は自動探索の手法を用いた、複雑なシステムをシステマチックにテストを行うことを支援するツールである。この手法は HiL でのテストにも方法を適用できるが、主に MiL と SiL をサポートする。 システムテストに TestWeaver を使用する主な利点は以下の通りである

- システムの振る舞いに関するテスト網羅率を劇的に向上させる
- テストエンジニアの負担が非常に低い

これを実現するために TestWeaver はテストケースを自動生成する。手書きのテストスクリプトを使うこともできるが、全体的なテストプロセスは、手書きのテストスクリプトに依存することはなく、テストカバレッジに対する問題を克服することが可能である。

4.1　チェスの原理

TestWeaver のに背後にある重要な考えは、テスト対象システム（system under test、SUT）のテストを行うということは、テスト対象とチェスを行うようなもので、その仕様に違反する状況に導こうとするものである。もしテスターがテスト対象に潜む、好ましくないステートに行くような動きのシーケンスを見つければ、ゲームに勝つということであり、そのシーケンスの動きがテストを意味している。更なる類似点は、次の最適な一手を打つ前に、チェスのコンピューターは多くの正当な可能性のある現在のステートに対しての全ての動きを単純に繰り返しながら探索し、それらが目的のステートになるかをテストする。この検索プロセスは莫大な選択肢（枝）のツリーを生成する（図 3）。TestWeaver ではテスト対象が、実行可能なシミュレーション（MiL）もしくはいくつかの通信しあうコシミュレーションとしての幾つかのモジュール（SiL）が利用可能である必要がある。よくやられるように、テスト対象はテストドライバーと通信するいくつかのコンポーネントを使用する。これらの通信コミュニケーションはインスツルメントと呼ばれ、暗示的なゲームのルールを持ち、TestWeaver はテスト対象のインスツルメントとゲームをプレイする。すなわち、それらはテスト対象の以下の情報を持つ。特定のシチュエーションでの正当な制御アクション、テスト対象によって到達した、面白そうなステート、結果として得られた、システム要求からの違反。各々のインスツルメントは関連したテスト対象のステート空間に関連した次元を決定する。各々の次元に含まれるドメインの値は、いくつかの有限なパーティションの組に分けられる。テスト対象、もしくはテスト対象モジュールには、パラメータ等を設定されたインスツルメントが入る。ゲームはパーティションに分けられた複数の次元で構成されたシステムスペースでプレイされる。

インスツルメントされたテスト対象に対して、TestWeaver はシステマチックに何千もの異なったシミュレーションシナリオを生成する。TestWeaver は（a）インテリジェントに仕様の違反を探す（b）テスト網羅度を最大化する、目的から過去のシミュレーションの結果を解析する。テスト網羅は以下のように定義され

る。上記で述べられたように、TestWeaver によりコントロールされた、もしくはモニターされたインスツルメントは各々の変数のドメインはインスツルメントの小さな間隔の組に分割される。これにより、モデルのインスツルメントは n 次元のディスクリート（すなわち有限な）ステートの空間として定義される。テスト網羅の目的は少なくとも一回はスペース内の各々の到達可能なディスクリートステートに到達することである。

図 1：チェスの原理

4.2 インスツルメント

インスツルメントとは、基本的に小さなコードで、インスツルメントされないバージョンのテスト対象に加えられ、Modelica、Matlab/Simulink、Python もしくは C のように実行可能な言語が使用される。インスツルメントはモデル（例えば Simulink、もしくは Modelica）の内部に入れられたり、もしくはコシミュレーションの場合は異なったモジュールに実装されたりする（例：Python）。インスツルメントは TestWeaver とテスト実行中に通信を行い、TestWeaver のテスト実行を可能にし、到達したステートを保持し、テスト実行中に好ましくないステート（失敗）かどうかを決定する。TestWeaver は基本的に 2 種類のインスツルメントをサポートする。それは action chooser と state reporter であり、以下の特徴を持つ。

1. state reporter: このインスツルメントはテスト対象のディスクリートもしくは連続の値（例：double）をモニターし、この値をいくつかの組のパーティションもしくはディスクリート値に（例：ロー、ミディアム、ハイ）マッピングする。テスト中に、このインスツルメントは TestWeaver にモニターした変数のディスクリート値を報告する。これは TestWeaver に到達したステートを追跡し、分割されたパーティション、もしくはディスクリートステートスペースの網羅を最大化するために使用される。

2.alarm reporter：これは実際には state reporter である。さらに重要度のレベル、例えば、通知、警告、アラーム、エラーなどに分割され関連付けられている。“悪い”ステートへの到達率は現在実行されているテストの失敗に対応している。注意：これらの失敗条件は TestWeaver により実行される全てのテストで検証される。

3. action chooser：このインスツルメントはテスト対象の入力変数と関連付けられる。自動車のアプリケーションでは入力変数は車のアクセルペダルやブレーキペダル等となる。インスツルメンテーションの設定によりこのインスツルメントは周期的もしくは、トリガー条件が真になった時に、パーティションに分けられたドメインの入力変数のディスクリート値をどう設定するか、TestWeaver に尋ねる。

4. fault chooser: これは action chooser の特殊なケースである。ドメインの値は名称と、欠陥のパーティションに分けられ、テスト対象のコンポーネントの故障モードを表現するために使わる。例えば、シフトバルブモデルは ok、閉固着、開固着等のモードを持つかもしれない。このようなインスツルメントは TestWeaver にテスト実行中に自然に故障コンポーネントを注入（アクティベート）するために使われる。

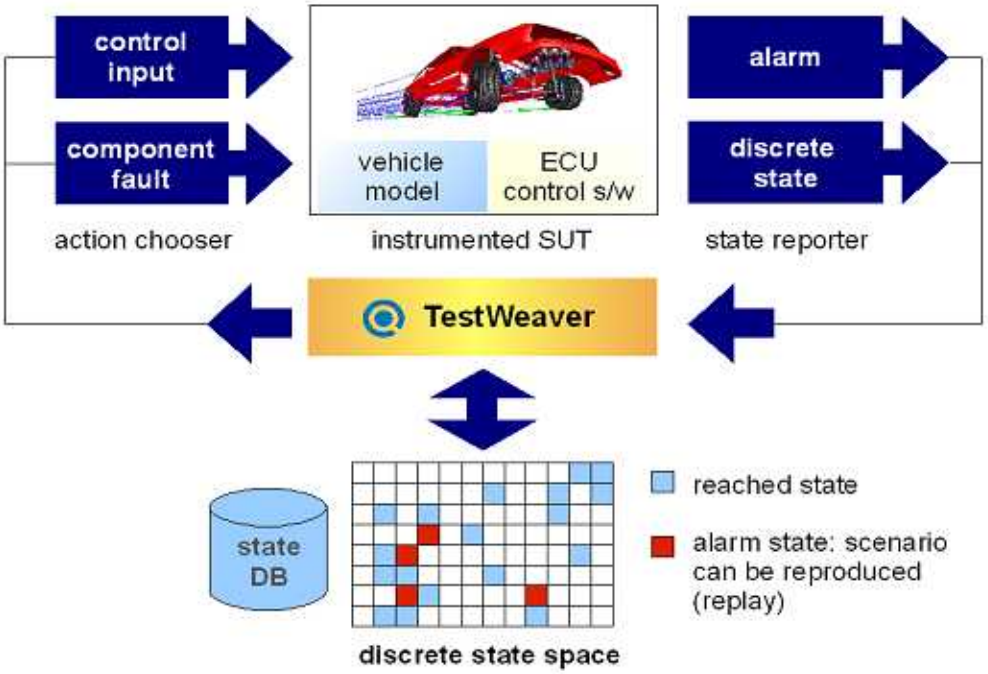

図 2：テスト対象と TestWeaver を接続するインスツルメント

またエンジニアは自分の好みのモデリング環境を使うのを好む。従って、上記のインスツルメントは多くのモデル環境とプログラミング言語をサポートしている。例：Matlab/Simulink, Modelica, Python, Silver, C/C++。これにより、彼らの使っている好みの、テスト対象で実装されている言語もしくは、テスト対象モジュールが使えるようになる。明示的なインスツルメントに加えて、TestWeaver はテスト対象を実行するプロセスをモニターし、ゼロ割やメモリアクセス違反、通信のタイムアウトなどの問題を記録する。

4.3 Experiment、シナリオ、およびレポート

TestWeaver で言う、experiment とは特定の期間、テスト対象で到達したステートの探索とドキュメント作成のプロセスであり、追加の探索の制約と網羅度の達成が考慮される。Experiment は長時間、一般的には何時間も完全に自動的に実行され、ユーザーの関与は必要ない。Experiment が実行されているとき、TestWeaver は action chooser 対して異なったシーケンスを選ぶことにより、多くの異なったシナリオを生成する。シナリオはステート空間に分割された、テスト対象を実行するシミュレーションのトレース、もしくはプロトコルである。TestWeaver はシステムの各ステートに到達する網羅度を最大化し、かつ欠陥を見つける可能性を増加させために、幾つかの戦略を組み合わせる。結果は experiment のデータベースシナリオ、すなわち図 3 のようなシナリオ（実際にはダイレクトグラフ）ツリーとして保存される。

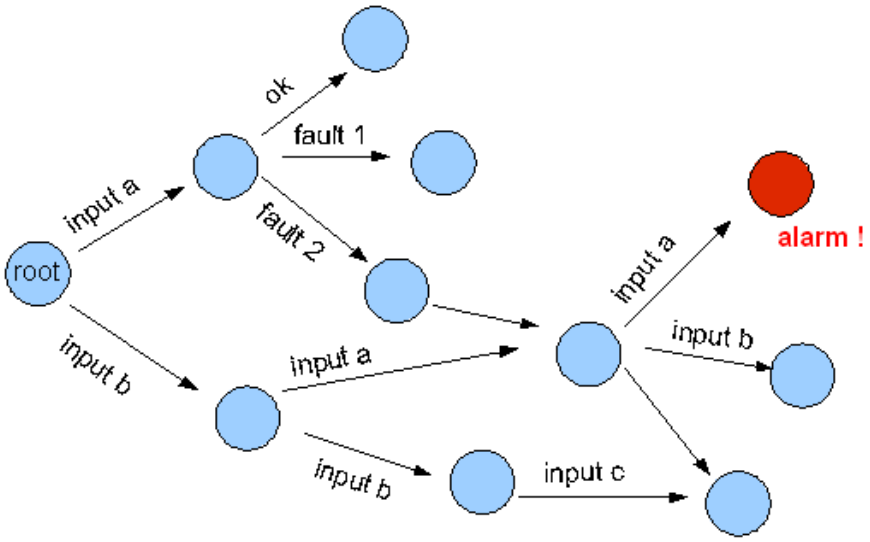

図 3：Experiment により生成されたシナリオ

ユーザーは SQL と類似の高級クエリ言語を使うことにより experiment 内で到達したステートを調査することができる。結果はレポートとして表示される。レポートは基本的に表であり、シナリオデータベースに保存されたシナリオのプロパティを選択して表示する。ユーザーはテンプレートにより表の構造とレイアウトを設定し、表の内容はシナリオデータベースの内容に依存する。これらには 2 つの種類のレポートがある。ステートの到達度を報告するオーバービューレポートと、個々のシナリオの詳細を報告するシナリオレポートである。

例えば、図 8 のオーバービューレポートを表示するには以下の選択文を定義する。

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

表のセルの色は TestWeaver により自動的に生成され、赤はアラームで、グリーンとブルーは通常のステートである。レポートはまたテストエンジニア使用する編集可能なコラムを含み、アラームに対する評価や、修正方法などを記入できる。これにより全ての発見された問題のトレーサビリティをサポートする。

ユーザーは experiment の開始、終了を設定でき、またたとえ experiment の実行中でも experiment のデータベースをリセットし、experiment により生成されたレポートを調査することが可能である。各々のシナリオはリプレイすることができる。すなわち、テスト対象は再実行され、記録された入力と同じシーケンスを再現できる。それにより、例えばシグナルをプロットし、問題の詳細なデバッグを行うといったことができる。

4.4 Experiment のフォーカス

ステート空間のディメンションとパーティションはテスト対象システムのインスツルメントにより定義される。それらとは別に、インスツルメントされたテスト対象システムもしくは TestWeaver の experiment にフォーカスされた仕様を明示的に定義し、探索を制約することができる。Experiment のフォーカスは experiment 実行中にどの空間ステートが調査されるかを定義する。TestWeaver は Experiment 実行中に、experiment がフォーカスしたそれらのステートに行くようにテスト対象を制御するように試みる。Experiment のフォーカスは現状では以下の 2 つの種類がある。

- 制約：制約は experiment の空間ステートのサイズを制約する。これらは例えば、シナリオの持続時間、もしくは入力とステートの組み合わせの数を制限する。制約するための高級言語がこの目的で供給される。例えば自動車のアプリケーションでは、ブレーキペダルとアクセルペダルが同時に使われる全てのシナリオを除外することができる。故障を解析する場合、検査の特定の故障モードを

除外することにこの制約が使用されたり、0,1,2 等の典型的な値が、シナリオに挿入され、故障の数を制限する。例えば複雑な故障検出や再設定メカニズムのシステムのようなフォールトトレラントシステムを調査する場合、妥当な数字を入力できる。

- 網羅性：ユーザーは TestWeaver に experiment のカバレッジの目標を定義するという意味で、experiment の幾つかのレポートを使うことができる。この意味でつかわれるレポートはカバレッジレポートと呼ばれる。

異なったテスト対象システムのバージョンの Experiments は異なったフォーカスの定義を生成し、実行し、比較することができる。

4.5 問題を分析して、デバッグ

テスト対象システムのアラームとエラーステートはオーバービューレポートにレポートされる。一つもしくは複数のステートに到達したシナリオで発生した問題は、シナリオデータベースから再現できる。シナリオは再生され、追加の調査が可能である。テスト対象システムのシミュレーション環境によって、例えば、追加のシグナルをプロットしたり、アニメーションなどを追加して視覚化をしたり、ブレークポイントの設定や、C での開発モジュールのように、ソースコードデバッガでのステップ検証などができる。

5 例: オートマチックトランスミッション

TestWeaver を使用したアプリケーションの例として、オートマチックトランスミッションの制御ソフトウェアの開発を考察する。トランスミッションを含んでいる、インスツルメントされたの車の Modelica モデルを図 4 に示す。TestWeaver とモデルとの接続は TCP/IP で確立され、TestWeaver はモデルをリモートにコントロールし、振る舞いを監視する。

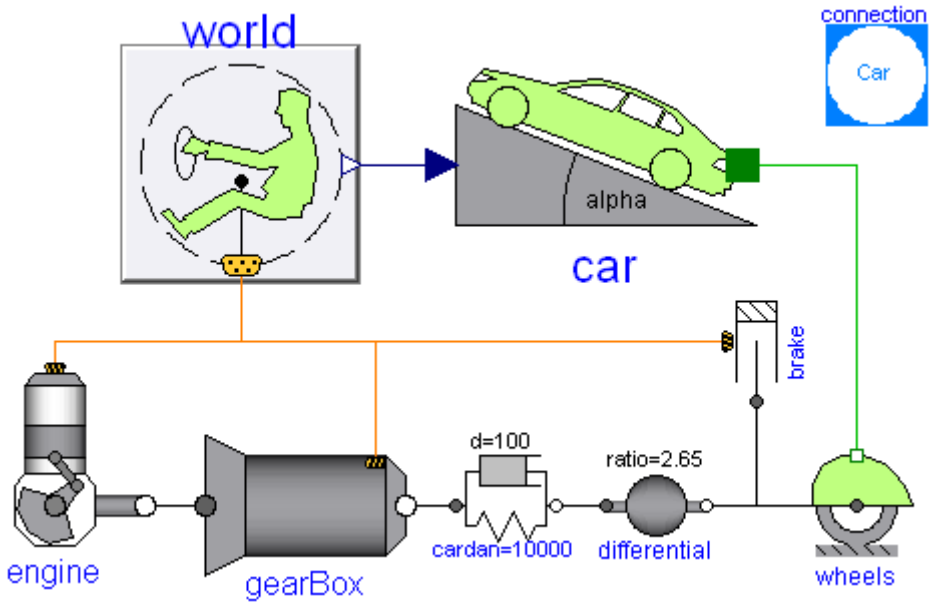

図 4:インスツルメントされた車両モデル

この例では、制御ソフトウェアは Simulink を使って開発されている。実行可能なテスト対象システムは 2 つのモジュールの Silver コシミュレーションとして生成されていて、コントロールモデルの Simulink と図 4 の Modelica モデルがコンパイルされている。Modelica モデルがインスツルメントされているので、テスト対象システムはまた TestWeaver と通信する機能を含んでいる。TestWeaver がシステムテストをテスト対象システムで開始したときに、全ての登録されたインスツルメントは静的なプロパティ（インターバル、ラベル、深刻度）として宣言される。TestWeaver はこれらのインスツルメントのリストを表示する。図 8 参照。インスツルメントのツリーを選択すると全てのプロパティが表示される。図 8 は図 5 の温度レポーターの TestWeaver 上での表示が示されている。

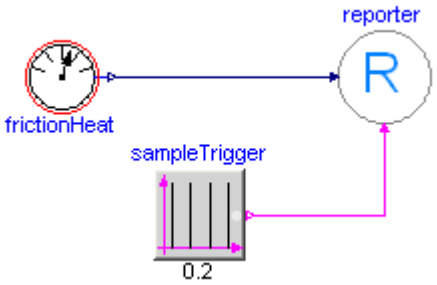

図 5：温度のレポート

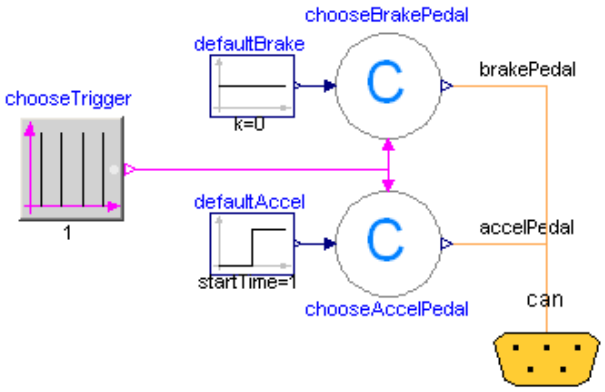

図 6：車の 2 つのペダルのコントロール

Instrument heatA

Instrument role: ALARM
Instrument type: REAL
Instrument unit: J
Declared by: Car

partitions	occurence	severity	color	values
damaged	10	10	#F03939	80000.0 : 1.0E10
ok	10	0	#AFDFDF	-0.1 : 40000.0
hot	10	0	#7FBFBF	40000.0 : 80000.0

図 7：TestWeaver によって表示された図 5 のレポーター

また、図 8 に示されたツリーは experiment の各レポートのための項目を含んでいる。そのレポートを選択すると図 8 に示されたような表を表示する。図 8 はどのギアシフトが experiment 実行中に既に到達しクラッチ A と B がクリティカルな温度に到達したかどうかを示している。各々のステートは、一番右側のコラムに、最大 2 つまでのシナリオとして参照される。それをクリックすると、ディスクリートステートのシーケンスとしてシナリオを表示する。テストエンジニアがその全ての詳細にアクセスすることができるように、個別のシミュレーションで実行されたシナリオの全ては再度生成することが可能である。例えば、ゼロ割などの制御ソフトウェアの実行時エラーはこの方法により再度発生させることができ、デバッグツールを使って調査できる。

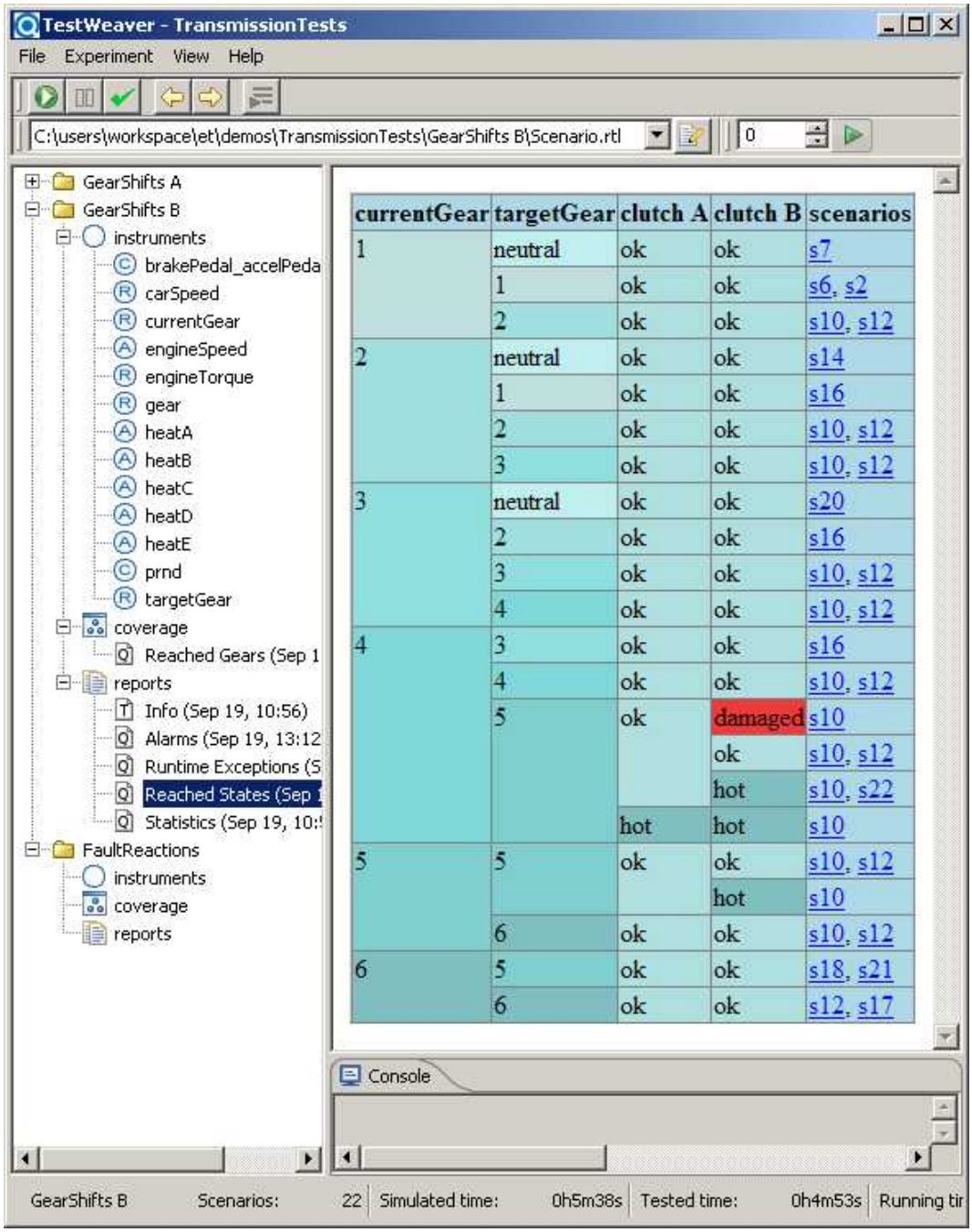

図 8：TestWeaver によるオーバービューレポート

6　シミュレーションモデルの実装

TestWeaver はテスト対象のシステムのシミュレーションモデルを必要とする。それらのモデルは一般的に入手できるが、モデルベースの開発プロセスの発展により、自動車のソフトウェアの開発では今日標準的なアプローチとなってきている。TestWeaver は 4 章で記述されているように、専用のインスツルメントを通じてテスト対象システムと通信する。これにより TestWeaver とシミュレーションツール、および環境との接続を単純化する。TestWeaver は以下のツールやプログラミング環境の通信をサポートしている。Visual C/C++ (Microsoft), Python 2.5, Matlab/Simulink RealTime Workshop R2006 (MathWorks) TargetLink (dSpace), Modelica/Dymola 6.x (Dynasim), Modelica/SimulationX 3.1 (ITI). Simpack (Intec) は現在開発中である。

テスト対象システムはまた、他のツールで開発された幾つかのモジュールのコシミュレーションとして実装されていてもよい。例えば車のモデルは Modelica を使用して実装され、制御ソフトウェアは Simulink で実装され、また TestWeaver は Python スクリプトを使用してインスツルメントされているような場合である。複合モデルを実行する Silver などのコシミュレーションツールも使われる。Silver を使用すると全てのモジュールはネイティブの開発環境にコンパイルされ、エクスポートされ、セルフ統合モジュール（DLL、ダイナミックリンクライブラリ）となる。エクスポートされたモジュールは固定されたマクロステップ幅を使用した一つのプロセス内で Silver によりサイクリックに実行される。モジュールは各々のマクロステップで信号がやりとりされる。マクロステップ内では、モジュールは実数の結合のような、より細かなスケールの可変のステップサイズを実行することができる。マクロステップ幅は、各々の ECU のサンプリングレートと一致する、例えば通常の TCU では 10ms となる。この方法を取れば Silver はバーチャルなシステム結合とシステムの振る舞いの結果のアセスメントが可能になる。

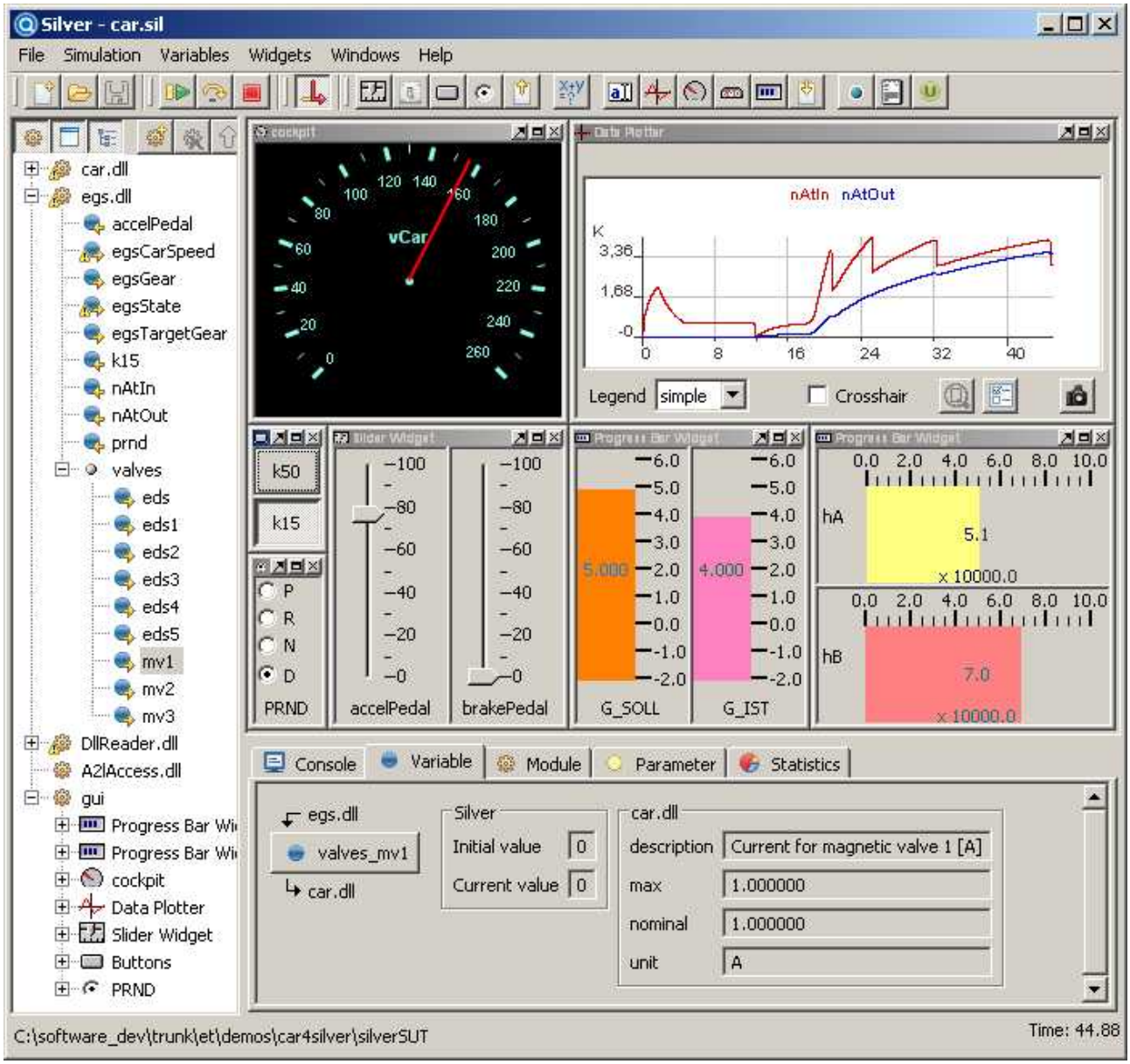

図 9：Silver によるコシミュレーション

3 章で既に述べられたこと以外の Silver の SiL/MiL に使用する利点は以下となる。

- コンパイルされたモジュールはモジュールのソースを公開しない。これにより OEM とサプライヤーのコラボレーションを簡潔にする。

- コンパイルされたモジュールはインタープリター、例えば Simulink 等に比べて非常に高速である。従って、TestWeaver は experiment 実行中に多くのシミュレーションを実行できる。
- モジュールは異なったツールで開発できる。従って、各々のモデリング開発に最適なツールを使用できる。
- MDF を介して ASAP2/A2L (ASAM), DCM (ETAS) 等の自動車向けキャリブレーションデータを結合できる。XCP (ASAM)を介して測定とキャリブレーションができる。これにより TestWeaver はキャリブレーションデータに基づいてシステムをテストできる。さらに Silver は同じキャリブレーションツール例えば実車で使われる CANape に繋げることができる。
- 強力なテストとデバッグオプション：TestWeaver などのテスト自動化ツール、Microsoft Visual C/C++ のような JIT（Just in time）デバッギングツールにより ECU ソフトウェアとプログラムコードの例外をキャッチすることができる。
- ビルトイン TestWeaver インスツルメント：Silver は TestWeaver にレポートするプログラム例外をキャッチするビルトインインスツルメントを含む。これは例外の種類やプログラムソースの行番号などの追加の情報を含む。

図 9 は Silver の車両シミュレーションを表す。モジュールとそれらの変更可能な入力、出力変数はユーザーインターフェースの左側に表示され、シミュレーションされた車のスタートと運転に使われる。

7　概要と結論

ますます複雑化する製品開発、開発期間短縮、コスト削減のプレッシャーから、新たな戦略が要求される。今日我々は HiL やテストリグのような早期のモジュールテストと後期のシステムレベルのテストが最先端の技術と考える。モジュールの間の連携の複雑化が増加し、システムレベルのバグが起こりやすくなっているが、後期の発見と修正では大きなコストがかかるので、早期のシステムレベルのテストの重要さが増している。実車でのプロトタイプの前のコントローラーとハードウェアのテストは早期のシステムレベルのテストに必要なテスト手法である。

単純な入力を使い、簡単にシステムの振る舞いを表現することが可能であれば、スクリプトベースのテストの戦略は簡単に実現可能であるが、システムの複雑さが増すにつれ、現実的なコストで、必要なテスト網羅を実現することは難しくなる。しかしながら、我々のテスト手法は以下を可能にする。

- 大きなステート空間を低い定義コストでシステマチックに調査できる。ゲームのルールのみ設定する必要があり、各々のシナリオ作成は必要ない。
- 従来のテスト手法による事前に定義されたテストシナリオを使うのみでは発見できない新たな問題を発見する。TestWeaver は新たな、品質の高い、さまざまなテスト、何千ものシナリオを自動的に生成することができる。
- 隠れた設計ミスの存在を無くし、信頼性の向上の向上が可能

5 章でオートマチックトランスミッションを TestWeaver と SiL ベースのシステムテストで行うアプリケーション例を述べた。我々はこの種のアプリケーションテストに長年の経験がある。しかしながら、TestWeaver は他のドメインでも、特にソフトウェアと物理世界で複雑な相互作用があるシステムに対して応用できると考える。

例えば、

- ドライバーアシスタンスシステム：ABS や ESP(Electronic Stability Program) 等の車のシステム。我々はこれらの制御ソフトウェアと車両のダイナミクスとドライバーの相互関係の複雑さに対面している。これは無数の関連したシナリオが設計時に調査されるべきである。
- プラントコントロールシステム：化学プロセスのプラント、発電プラントなど。我々はプラント物理とオペレーションの制御ソフトウェアの相互関係に直面している。同様に、これらの複雑さは設計時にシステマチックに調査されるべきである。

TestWeaver は Windows プラットフォーム上で実行される。これは強力で、使いやすいツールである。ユーザーはネイティブな仕様もしくはモデリング環境で使用でき、別のテスト仕様言語の使い方を覚える必要はない。

References

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

[2] Rebeschieß, S., Liebezeit, Th., Bazarsuren, U., Gühmann, C.: Automatisierter Closed-Loop-Testprozess für Steuergerätefunktionen. ATZ elektronik, 1/2007 (in German).

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

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

[5] Junghanns, A., Mauss, J., Tatar, M.: TestWeaver - Simulationbased Test of Mechatronic Designs - In: Proceedings International Modelica Conference, Bielefeld, 2008.

[6] Junghanns, A., Mauss, J., Tatar, M.: TestWeaver - Funktionstest nach dem Schachspieler Prinzip. In:2nd Conference on Testing of Hardware and Software in Automotive Design (AutoTest 2008), Stuttgart, 2008 (in German).

[7] M. Gäfvert, J. Hultén, J. Andreasson, A. Junghanns, Simulation-Based Automated Verification of Safety-Control Systems. 9th International Symposium on Control (AVEC2008), Kobe, Japan, 2008.