

# モデルベース開発におけるテスト自動化フレームワーク

黒岩 正司  
富士設備工業株式会社

## 概要

要求仕様から実ターゲットまで開発プロセス全体に渡るテストの自動化フレームワークについて紹介する。T-VEC ツールは、要求仕様を表形式でモデル化したもの (T-VEC Tabular Modeler : TTM) や、Simulink に対しフォーマルメソッドを用いて解析しモデル上の欠陥を検出する。この解析は、モデル上のパスごとに取り得る入力範囲の境界値を攻めるテストベクタ(入力と期待値)を生成することで行われる。このテストベクタは、フロート、ダブルなど各種データ型、リニア/ノンリニア式にも対応し、実ターゲット環境での実行を通してモデルとコードの一致性の検証が行える。実行結果のカバレッジは、LDRA ツールを用いてあらゆる組込みターゲットで動的に解析される。これらテスト結果、カバレッジはテストベクタを介して、要求仕様、Simulink とのトレーサビリティが取られる。要求からテストのトレーサビリティによって得られる効果は、実装テストに於ける解析が迅速になること、要求が十分にテストされたことの証明、プロジェクト進捗管理の為の計測など。

## 1. はじめに

組込みソフトウェアは、私たちの身の回りにある様々なシステムの中核を担い、多様な機能やサービスを提供している。そういった中で、システムの要求からソフトウェア設計、コード生成までの工程で、モデルを用いるモデルベース開発手法が、ソフトウェア開発プロセスに導入されてきている。一方で、要求に対するモデルの検証、モデルとコードの一致性検証など大規模なソフトウェア開発を考えた場合、時間とコストが課題になってくる。昨今の組込みシステムは、製品競争の激化により、高機能、高品質、低コスト、短

納期への要求が強くなり、組込みソフトウェア開発の現場もこれらの課題解決が急務となっている。このような背景から、モデルベース開発においてフォーマルメソッドを利用し、システム要求のモデル検証からソフトウェア検証までを統合的に扱う枠組みが求められている。本稿では、要求モデル・デザインモデルに対してフォーマルメソッドを用いて解析しテストベクタを生成する T-VEC ツールと、静/カバレッジ解析、テストドライバ生成機能を有する LDRA ツールを紹介し、それらを統合したテスト自動化フレームワークについて述べる。

## 2. T-VEC / LDRA

### 2.1. T-VEC

T-VEC ツールは、フォーマルメソッドを利用したモデルベース検証ツールである。高信頼性ソフトウェアの Verification & Validation をサポートする目的で開発された。航空機開発におけるガイドライン FAA/DO-178B,C に関わり、モデルベース開発とその検証についてのガイドライン制定に貢献するなど実践的に産業界で使用されるツールである。例えば、次世代スペースシャトル(Project Orion)の主契約企業であるロッキードマーチン社は、このプロジェクトに T-VEC の TTM 要求仕様モデルと Simulink のモデル検証機能をサブコントラクタも含めて採用することを表明している。

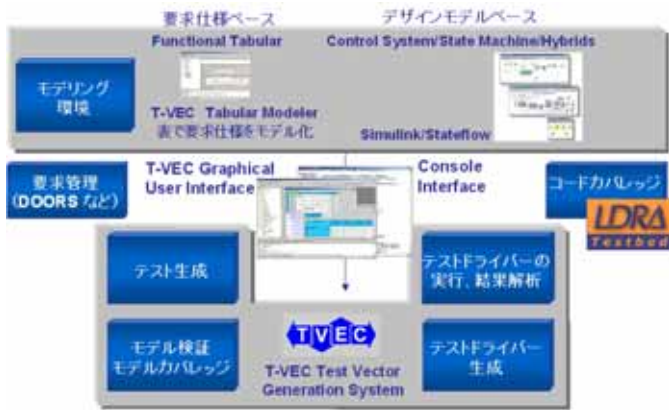


図 1. T-VEC モデル解析，検証基盤

T-VEC は検証に先駆けて要求モデルやデザインモデル (Simulink/Stateflow) を独自の階層的 Disjunctive Normal Form(DNF)に変換する。この形式に於ける各機能要求 (入出力間の振舞い) の入力域に対するコンストレインツは Domain Conversion Path(DCP)として抽出される。このパスに対し、テストベクタ生成システムは、上流の制約を受けて伝播されてくる入力範囲で期待出力との組合せを生成する。ここで、テストベクタを生成できない DCP は、モデル上の矛盾として検出される (モデル検査)。また、このテストベクタ生成機能は、フロート、ダブルなど各種データ型、リニア/ノンリニア式に対応しているため、矛盾無く生成されたテストベクタは、実システムに対する入力と期待値としてテストに用いられ、モデルに対するコードの一致性の検証が行える。[1]

このテストのカバレッジ解析 (2.2. LDRA) を行うことで、要求仕様，デザイン仕様，ソースコードの一貫性も検証できる。

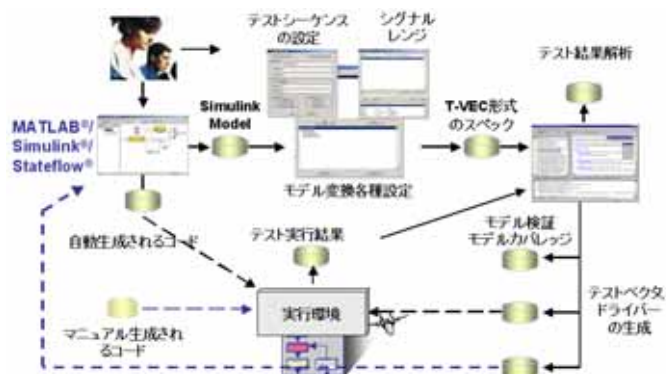


図 2. モデルベース検証の自動化

T-VEC は、以下の機能で構成される。

- T-VEC Tabular Modeler (TTM)
- Simulink Tester (SL2TVEC)
- T-VEC Vector Generation System (VGS)

TTM は、要求管理を提供し、要求とモデリングをサポートする。欠陥解析や自動テストケース生成のために、要求仕様を表形式の GUI を用いてモデル化する (型, I/O などのインタフェースや状態, イベント, 遷移テーブルなど振る舞いをモデル化する)。VGS でモデル検証し、要求仕様に基づいたテストベクタを生成する。DOORs 要求管理ツールなどと統合され、要求仕様とテストのトレーサビリティをとる。テストで問題が発生すれば、そのテストベクタ, モデル, 要求仕様と戻って解析できる。TTM に直接要求仕様を含めることも可能で、同様に管理される。

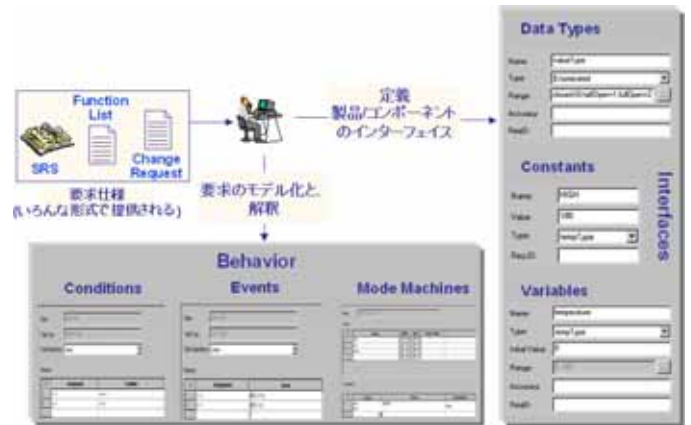


図 3. TTM による要求仕様のモデル化

SL2TVEC は、デザインモデルである Simulink や Stateflow を変換し解析する。モデルを VGS で検証し、テストベクタ生成, 実行, 結果判定を行う。モデルから自動生成されるソースコードに対するテストドライバも自動生成される。VGS と統合することでテストプロセスの多くを自動化する。モデル上の欠陥 (ゼロ割やレンジオーバーフローなど) を検出することや、アサーションによるモデルのセーフティプロパティの検証なども行える。

VGS は、モデル解析, テストベクタ生成, テストドライバ生成, テスト結果解析, 結果レポート生成などを行う。

## 2.2. LDRA ツール

LDRA ツールには、静的解析 / カバレッジ解析を行う Testbed と、単体テスト、リグレッションテストを自動化する TBrun がある。LDRA ツールは、高信頼性マーケットで 30 年に渡って使用されている。

LDRA Testbed は、ソースコードの静的解析とカバレッジ解析を提供するツールである。静的解析では、独自の構文解析エンジンを持ち、プログラミングスタンダードチェック、複雑度解析、データフロー解析、コードのビジュアル化(コールグラフ、フローグラフ)や詳細レポートを生成する。LDRA ツールは、プログラミングスタンダードルールである MISRA (Motor Industry Software Reliability) に対応し、DO-178B の認証に初めて利用されたツールでもある。カバレッジ解析では予めタグをソースコードに埋め込み、テストデータを実行することでテストされていない領域(未実行パスやデッドコード)を特定できる。ステートメント、ブランチ、MC/DC カバレッジなどを測定できる。

LDRA TBrun は、テストハーネス / ドライバを自動生成し単体テストを自動化する。追加のコーディングやスクリプトは不要で、ユニット単位のテスト実行を容易にかつ効果的に行える。テストケース(入出力値と結果)は保存され、リグレッションテストに対する解析の自動化に利用される。ソースコードの変更箇所を自動検出し(関数の引数が増えた、型が変更されたなど)、保存されているテストに対して変更が必要な部分はレポートされるため、テストデータのメンテナンスが効率よく行える。

TBrun は、Testbed による静解析から必要な情報(関数の引数、グローバル変数(入出力)、戻り値、変数の型、関数コールなど)を取得している。単体テストでは、ある単体から別の単体へのコールがあり、このようなコールで実体が記述されていない単体もある。この場合、TBrun はテスト内のコールに対するスタブを自動生成する。生成されるスタブは、グローバル変数やポインタ等の解決、関数戻り値の設定など柔軟に使用される。TBrun は、ホスト間、ホスト-ターゲット間など、あらゆる環境でのテスト実行とカ

バレッジ解析をサポートする。

## 3. 手法

上述で紹介したツールを使ったテスト自動化フレームワークについて、下記要求仕様を基に述べる。実行環境は、GreenHills MULTI 統合環境が持つ、ルネサス M32R CPU シミュレータ(MULTI M32R シミュレータ)を使用する。

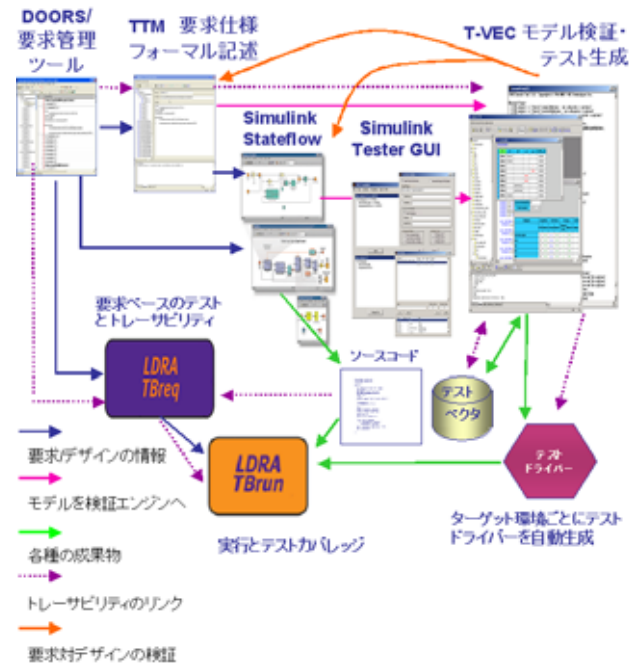


図 4. テスト自動化フレームワーク

### 3.1. 要求仕様

温度センサーにより温度を監視し、温度に合わせてバルブの開閉を調整するフローレギュレータ(flowRegulator)装置を考える。

- ・ 入力：温度(範囲：0~300)
- ・ 定数：Low : 120, High : 180
- ・ 温度が Low より小さい時、バルブは Closed
- ・ 温度が High より大きい時、バルブは Open し、温度が Low を下回るまで、バルブは開いている

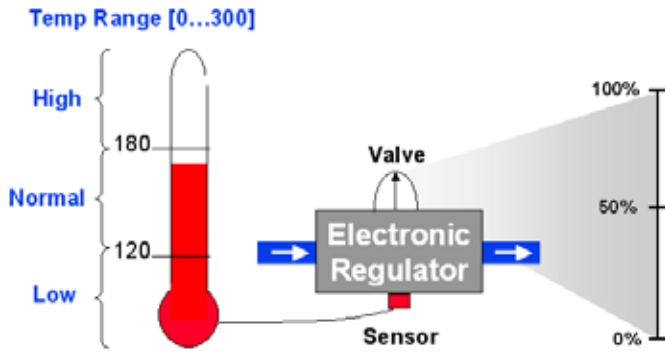


図 5. フローレギュレータ

flowRegulator モデルは 2 つのサブシステムを持つ .

- temperatureSensor
- flowControlLogic

temperatureSensor は , 入力温度の値に前回値を使った処理( $Y[n]=0.7U[n]+0.3Y[n-1]$ ) を行い , その結果を -100 ~ 300 の範囲でフィルタリングし出力する .

flowControlLogic は , temperatureSensor の値をチェックする . それにより , flowRegulator バルブの開閉が調整される . バルブが Closed の時は出力値 0% , バルブが Open の時 , 出力値は以下の方程式で表す .

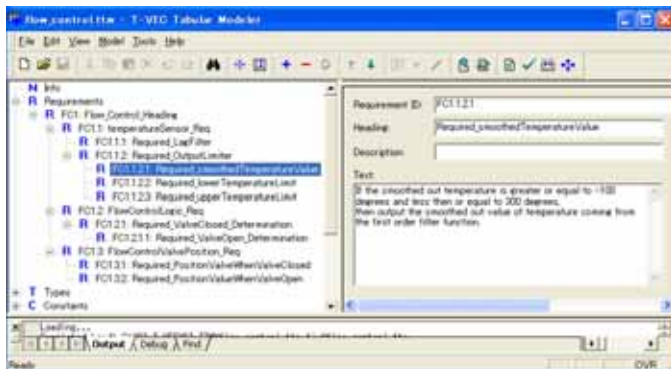
$$\text{Out} = ((\text{sensorTemperature} - 120) / (300 - 120)) * 100$$

### 3.2. 要求モデリング - TTM

上述の要求仕様を TTM でモデル化する . 要求仕様 , 要件管理番号 (Requirement-ID : 以下 Req-ID) , データ型 , 入力 , 定数 , 関数 , テーブル , 出力などを定義する .

#### [要求仕様定義]

要求内容と Req-ID を設定する .



#### [Input 定義]

- temperature ( 型 : temperatureType ) : 温度

Name:	temperature
Type:	temperatureType
Initial Value:	-
Range:	-150.0..400.0

Name:	temperatureType
Type:	Double
Range:	-150.0..400.0

- local\_SVs ( 型 : local\_SVs\_type ) : 前回値

Name:	local_SVs
Type:	local_SVs_type
Initial Value:	-
Range:	

Name:	local_SVs_type
Type:	Structure

#	Name	Type	Range	Accu
1	filteredTemperature	temperatureType	-150.0..400.0	

#### [定数定義]

- HIGH : 温度

Name:	HIGH
Type:	Double
Value:	180.0

- LOW : 温度

Name:	LOW
Type:	Double
Value:	120.0

- HIGH\_TEMP\_LIMIT : フィルタ値

Name:	HIGH_TEMP_LIMIT
Type:	Double
Value:	300.0

- LOW\_TEMP\_LIMIT : フィルタ値

Name:	LOW_TEMP_LIMIT
Type:	Double
Value:	-100.0

#### [関数定義]

Functions テーブルでは , 与えられたパラメータで , 構成された処理に応じた値が設定される .

firstOrderFilter :  $Y[n]=0.7U[n]+0.3Y[n-1]$  を関数のように使用する .

Name: firstOrderFilter  
 Type: Double  
 Range: -1.0e04, 1.0e04  
 Accuracy:

Main Parameters Requirements Comment

Behavior: firstOrderFilter

#	Assignment	Condition	Requirement ID
1	(input * input_gain) + (prevFilter_output + feedback_gain)	TRUE	FCI.1: temperatureSensor_Req

[一時保持変数定義]

Terms 変数は、システムの入出力の中間値に相当する。  
 temperatureSensor : firstOrderFilter をコールし、  
 temperatureSensor 出力と前回値を取得する。

Name: temperatureSensor  
 Type: temperatureSensor Type  
 Initial Value:  
 Range:  
 Accuracy:

Main Mode Dependency Requirements Inlined Variables Comment

Behavior: temperatureSensor

#	Assignment	Condition	Requirement ID
1	filteredTemperature = INTERM_temp_output; local_S_Vs_filteredTemperature = INTERM_temp_output;	INTERM_temp_output = firstOrderFilter(temperature, rTemperature, local_S_Vs_filteredTemperature, 0.7, 0.3) AND INTERM_temp_output >= LOW_TEMP_LIMIT AND INTERM_temp_output < 0	FCI.1.2.1: Required smoothed Temperature Value
2	filteredTemperature = INTERM_temp_output; local_S_Vs_filteredTemperature = INTERM_temp_output;	INTERM_temp_output = firstOrderFilter(temperature, rTemperature, local_S_Vs_filteredTemperature, 0.7, 0.3) AND INTERM_temp_output >= 0 AND INTERM_temp_output <= HIGH_TEMP_LIMIT	FCI.1.2.1: Required smoothed Temperature Value
3	filteredTemperature = LOW_TEMP_LIMIT; local_S_Vs_filteredTemperature = INTERM_temp_output;	INTERM_temp_output = firstOrderFilter(temperature, rTemperature, local_S_Vs_filteredTemperature, 0.7, 0.3) AND (INTERM_temp_output < LOW_TEMP_LIMIT)	FCI.1.2.2: Required lower Temperature Limit
4	filteredTemperature = HIGH_TEMP_LIMIT; local_S_Vs_filteredTemperature = INTERM_temp_output;	INTERM_temp_output = firstOrderFilter(temperature, rTemperature, local_S_Vs_filteredTemperature, 0.7, 0.3) AND (INTERM_temp_output > HIGH_TEMP_LIMIT)	FCI.1.2.3: Required upper Temperature Limit

[状態遷移定義]

Mode Machine テーブルは、状態遷移を定義する。  
 flowControlLogic : バルブが Closed の時に、温度が High を超えれば Open に遷移し、Open の時に、Low を下回れば Closed に遷移する。

#	Name	Initial	Final	Enum Value
1	CLOSED	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
2	OPEN	<input type="checkbox"/>	<input type="checkbox"/>	

Main Modes Requirements Inlined Variables Comment

Transitions: flowControlLogic

#	Source	Guard	Destination	Requirement ID
1	CLOSED	@!(temperatureSensor.filteredTemperature > HIGH)	OPEN	FCI.2.1.1: Required_ValveOpen_Determination
2	OPEN	@!(temperatureSensor.filteredTemperature < LOW)	CLOSED	FCI.2.1: Required_ValveClosed_Determination

[出力定義]

Outputs テーブルは、モデル全体の出力を定義する。  
 ValvePosition : 上述の変数やテーブルを受け、バルブ

の位置(0 ~ 100%) , 前回値を出力する。

Name: ValvePosition  
 Type: valveOutputType  
 Initial Value:  
 Range:  
 Accuracy:

Main Mode Dependency Requirements Inlined Variables Comment

Behavior: ValvePosition

#	Assignment	Condition	Mode	Requirement ID
1	valvePosition = (temperatureSensor.filteredTemperature - 120) / (300 - 120) * 100; local_S_Vs_filteredTemperature = temperatureSensor.filteredTemperature; valvePosition = 0;	TRUE	OPEN	FCI.2.2: Required_Position Value When Valve Open
2	valvePosition = 0; local_S_Vs_filteredTemperature = 0;	TRUE	CLOSED	FCI.2.1: Required_Position Value When Valve Closed

3.3.デザインモデリング - Simulink

上述の要求仕様を Simulink でモデル化する。

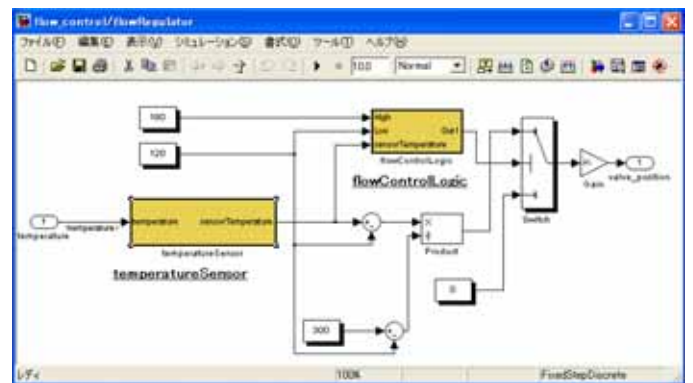


図 6. flowRegulator モデル

各ブロックのプロパティに、TTM で記述した Req-ID を設定する。

temperatureSensor : 図 7 参照.

[Saturation ブロック]

- $-100 \leq Y[n] \leq 300$  の時、 $Y[n]$  を出力
- $Y[n] < -100$  の時、-100 を出力
- $Y[n] > 300$  の時、300 を出力

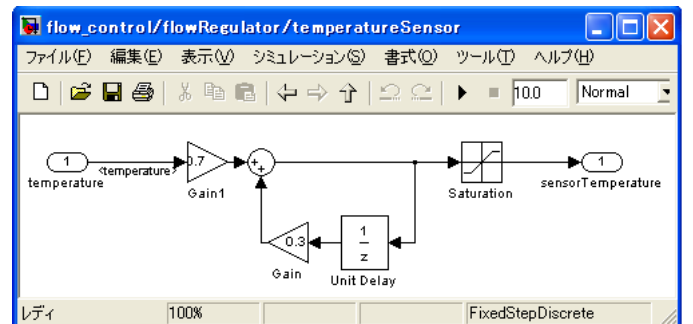


図 7. temperatureSensor

flowControlLogic と flowRegulator バルブの開閉 : 図 8 参照 . temperatureSensor の値を入力とし、バルブ

の開閉を調整する。

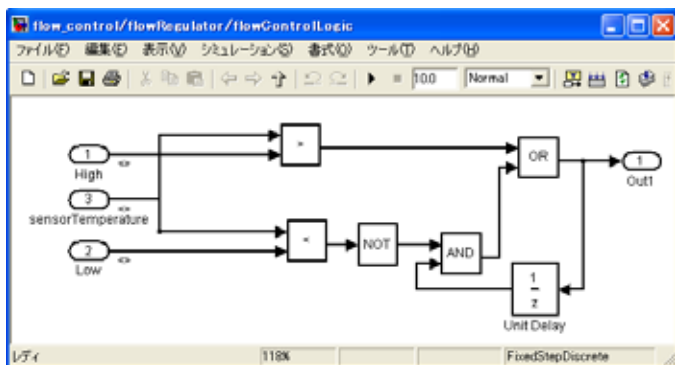


図 8. flowControlLogic

### 3.4. モデル検証

要求モデル, デザインモデルの検証結果から, DCP 毎で取り得た入力と期待出力の組合せ (テストベクタ), モデル対ソースコードのマップ情報, ソースコードへのパス情報などを LDRA ツールが読み込める形式 (.tcf) にし, インポートさせる. LDRA/Testbed でカバレッジ測定の為のタグ付けをソースコードに行い, LDRA/TBrun でテストを実行するためのドライバーを自動生成させる. これらは, ソースコードと共にコンパイル・リンクされ, ターゲットにダウンロードされ, 実行される. 結果は, ターゲット上のメモリに格納され, I/O あるいはデバッグなどを介してホスト上に吸い上げられ, テストの Pass/Fail 判定とカバレッジの解析が行われる. 以上のせいかぶつ, ソースコードに対する一連のテストベクタ, ドライバー, テスト結果は TBrun により管理され, リグレッションテスト時の自動化が支援されるようになる.

#### 3.4.1. 要求モデル - TTM

TTM モデルを T-VEC フォーマル言語に自動変換する.

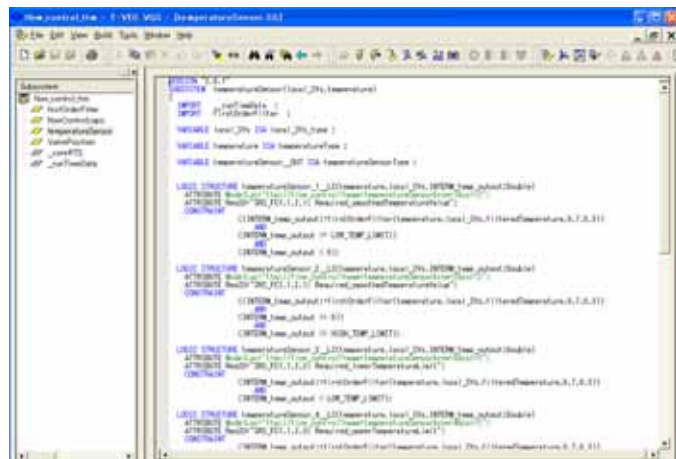


図 9. フォーマル言語

VGS でフォーマル言語をビルドし, テストベクタを生成する. temperatureSensor サブシステムでは, 6 件のテストベクタが生成される.

ReqID	Value	Type	Followed Stimuli
ReqID(1) : SRS_FC1.1.1.1 Required smoothed Temperature Value (Unity ReqID) : SRS_FC1.1.1.1 temperatureSensor_Rng			
temperatureSensor_Curr_000ns and Temperature	-3.0e+002	FLOAT64	-1.0e+002 3.0e+002
temperatureSensor_Curr_001ns and Temperature	-3.0e+002	FLOAT64	-3.0e+003 3.0e+003
ReqID(2) : SRS_FC1.1.1.1 Required Temperature (ReqID) : SRS_FC1.1.1.1 temperatureSensor_Rng			
temperatureSensor_Curr_000ns and Temperature	3.0e+002	FLOAT64	-1.0e+002 3.0e+002
temperatureSensor_Curr_001ns and Temperature	3.0e+002	FLOAT64	-3.0e+003 3.0e+003
ReqID(3) : SRS_FC1.1.2.1 Required Input Temperature at Input (Unity ReqID) : SRS_FC1.1.2.1 temperatureSensor_Rng			
temperatureSensor_Curr_000ns and Temperature	-3.0e+002	FLOAT64	-1.0e+002 3.0e+002
temperatureSensor_Curr_001ns and Temperature	-3.0e+002	FLOAT64	-3.0e+003 3.0e+003
ReqID(4) : SRS_FC1.1.2.2 Required Input Temperature at Input (Unity ReqID) : SRS_FC1.1.2.2 temperatureSensor_Rng			
temperatureSensor_Curr_000ns and Temperature	3.0e+002	FLOAT64	-1.0e+002 3.0e+002
temperatureSensor_Curr_001ns and Temperature	3.0e+002	FLOAT64	-3.0e+003 3.0e+003
ReqID(5) : SRS_FC1.1.2.3 Required Input Temperature at Input (Unity ReqID) : SRS_FC1.1.2.3 temperatureSensor_Rng			
temperatureSensor_Curr_000ns and Temperature	-3.0e+002	FLOAT64	-1.0e+002 3.0e+002
temperatureSensor_Curr_001ns and Temperature	-3.0e+002	FLOAT64	-3.0e+003 3.0e+003
ReqID(6) : SRS_FC1.1.2.3 Required Input Temperature at Input (Unity ReqID) : SRS_FC1.1.2.3 temperatureSensor_Rng			
temperatureSensor_Curr_000ns and Temperature	3.0e+002	FLOAT64	-1.0e+002 3.0e+002
temperatureSensor_Curr_001ns and Temperature	3.0e+002	FLOAT64	-3.0e+003 3.0e+003

図 10. temperatureSensor テストベクタ

Req-ID により, 要求からテストベクタへのトレーサビリティが取れる.

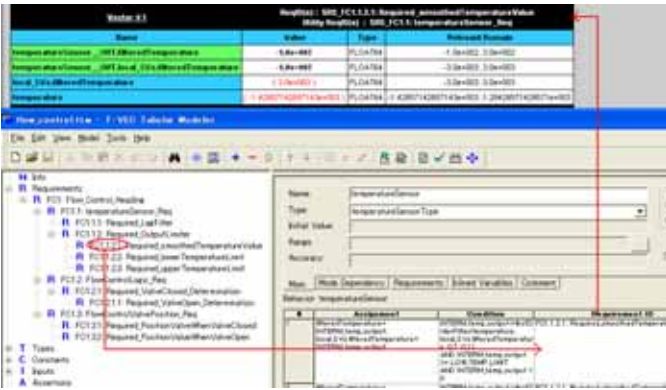


図 11. TTM とテストベクタの要求リンク

上記で生成されたテストベクタを VGS の機能を用いてテストドライバー化する。これは、TBrum にインポートされ、テスト対象プログラムと共にコンパイル及びリンクされ、MULTI M32R シミュレータにダウンロードされる。その後実行され、テスト、カバレッジの結果が解析される。

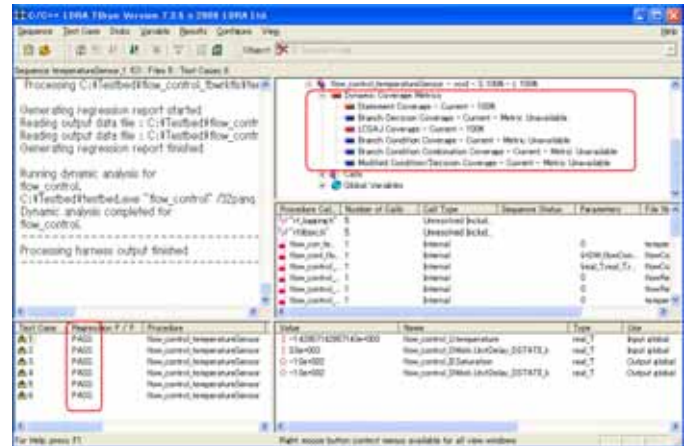


図 13. カバレッジ結果

Test	Object	Expected	Actual	Result
1	temperatureSensor_OUT.filteredTemperature	-1.0e+002	-1.0e+002	OK
1	temperatureSensor_OUT.local_SVs.filteredTemperature	-1.0e+002	-1.0e+002	OK
2	temperatureSensor_OUT.filteredTemperature	3.0e+002	3.0e+002	OK
2	temperatureSensor_OUT.local_SVs.filteredTemperature	3.0e+002	3.0e+002	OK
3	temperatureSensor_OUT.filteredTemperature	-1.0e+002	-1.0e+002	OK
3	temperatureSensor_OUT.local_SVs.filteredTemperature	-3.0e+003	-3.0e+003	OK
4	temperatureSensor_OUT.filteredTemperature	-1.0e+002	-1.0e+002	OK
4	temperatureSensor_OUT.local_SVs.filteredTemperature	-1.000000000012e+002	-1.000000000012e+002	OK
5	temperatureSensor_OUT.filteredTemperature	3.0e+002	3.0e+002	OK
5	temperatureSensor_OUT.local_SVs.filteredTemperature	3.000000000012e+002	3.000000000012e+002	OK
6	temperatureSensor_OUT.filteredTemperature	3.0e+002	3.0e+002	OK
6	temperatureSensor_OUT.local_SVs.filteredTemperature	3.0e+003	3.0e+003	OK

図 14. temperatureSensor テスト結果

### 3.4.2. デザインモデル - Simulink

SL2TVEC でモデルの Export Translate を実行し、T-VEC フォーマル言語に変換する。

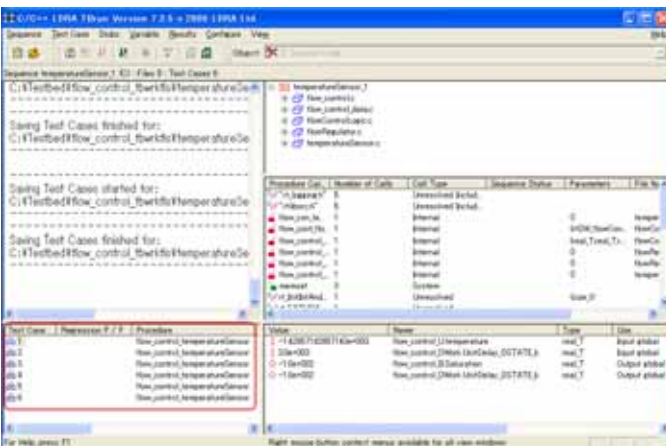


図 12. テストドライバーファイルのインポート

テストのカバレッジは 100%となった。また、期待値と実行値の比較結果は全て OK となり、要求モデルとソースコードの一致性が検証できた。

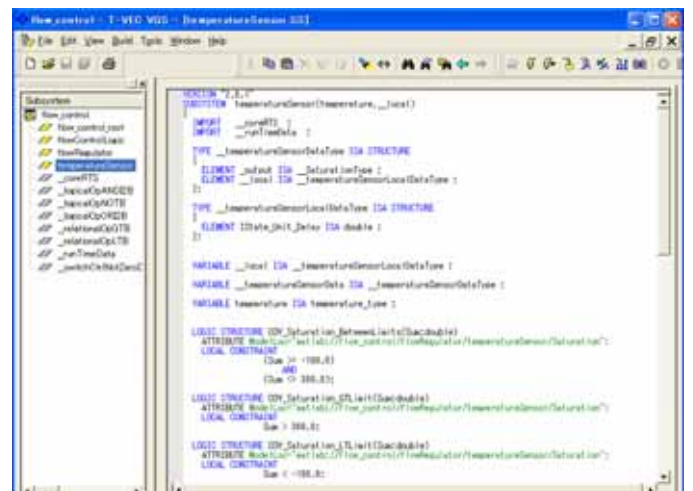


図 15. フォーマル言語

VGS でフォーマル言語をビルドし、テストベクタを生成する。temperatureSensor サブシステムでは、8件のテストベクタが生成される。

Test #1	ReqID(s)	FCI.1.1Required_Log/Wtr or FCI.1.2Required_Output/Limiter	
Name	Value	Type	Relevant Buttons
__temperatureSensorData__output	-6.9999998079071e+001	FL0A764	-1.0e+002, 3.0e+002
__temperatureSensorData__localIState_Unit_Delay	4.9999998079071e+001	FL0A764	-1.0e+012, 1.0e+012
temperature	-1.0e+002	FL0A764	-1.0e+002, 3.0e+002
__localIState_Unit_Delay	0.0e+000	FL0A764	0.0e+000, 0.0e+000
Test #2	ReqID(s)	FCI.1.1Required_Log/Wtr or FCI.1.2Required_Output/Limiter	
Name	Value	Type	Relevant Buttons
__temperatureSensorData__output	2.09999996423721e+002	FL0A764	-1.0e+002, 3.0e+002
__temperatureSensorData__localIState_Unit_Delay	2.09999996423721e+002	FL0A764	-1.0e+012, 1.0e+012
temperature	3.0e+002	FL0A764	-1.0e+002, 3.0e+002
__localIState_Unit_Delay	0.0e+000	FL0A764	0.0e+000, 0.0e+000
Test #3	ReqID(s)	FCI.1.1Required_Log/Wtr or FCI.1.2Required_Output/Limiter	
Name	Value	Type	Relevant Buttons
__temperatureSensorData__output	-6.0e+002	FL0A764	-1.0e+002, 3.0e+002
__temperatureSensorData__localIState_Unit_Delay	-1.0000000000000000e+010	FL0A764	-1.0e+012, 1.0e+012
temperature	-1.0e+002	FL0A764	-1.0e+002, 3.0e+002
__localIState_Unit_Delay	-3.33333333333333e+010	FL0A764	-3.33333333333333e+010, 3.33333333333333e+010
Test #4	ReqID(s)	FCI.1.1Required_Log/Wtr or FCI.1.2Required_Output/Limiter	
Name	Value	Type	Relevant Buttons
__temperatureSensorData__output	3.0e+002	FL0A764	-1.0e+002, 3.0e+002
__temperatureSensorData__localIState_Unit_Delay	1.0000000000000000e+010	FL0A764	-1.0e+012, 1.0e+012
temperature	3.0e+002	FL0A764	-1.0e+002, 3.0e+002
__localIState_Unit_Delay	3.33333333333333e+010	FL0A764	-3.33333333333333e+010, 3.33333333333333e+010
Test #5	ReqID(s)	FCI.1.1Required_Log/Wtr or FCI.1.2Required_Output/Limiter	
Name	Value	Type	Relevant Buttons
__temperatureSensorData__output	3.0e+002	FL0A764	-1.0e+002, 3.0e+002
__temperatureSensorData__localIState_Unit_Delay	1.0000000000000000e+010	FL0A764	-1.0e+012, 1.0e+012
temperature	-1.0e+002	FL0A764	-1.0e+002, 3.0e+002
__localIState_Unit_Delay	1.33333333333333e+003	FL0A764	1.29333333333333e+003, 3.33333333333333e+010
Test #6	ReqID(s)	FCI.1.1Required_Log/Wtr or FCI.1.2Required_Output/Limiter	
Name	Value	Type	Relevant Buttons
__temperatureSensorData__output	-6.0e+002	FL0A764	-1.0e+002, 3.0e+002
__temperatureSensorData__localIState_Unit_Delay	-1.0000000000000000e+010	FL0A764	-1.0e+012, 1.0e+012
temperature	-1.0e+002	FL0A764	-1.0e+002, 3.0e+002
__localIState_Unit_Delay	-1.000000021e+000	FL0A764	-3.33333333333333e+010, -1.000000021e+000
Test #7	ReqID(s)	FCI.1.1Required_Log/Wtr or FCI.1.2Required_Output/Limiter	
Name	Value	Type	Relevant Buttons
__temperatureSensorData__output	-6.0e+002	FL0A764	-1.0e+002, 3.0e+002
__temperatureSensorData__localIState_Unit_Delay	-6.0e+002	FL0A764	-1.0e+012, 1.0e+012
temperature	-1.0e+002	FL0A764	-1.0e+002, 3.0e+002
__localIState_Unit_Delay	-1.0000000979043e+002	FL0A764	-1.0000000979043e+002, 1.29333333333333e+003
Test #8	ReqID(s)	FCI.1.1Required_Log/Wtr or FCI.1.2Required_Output/Limiter	
Name	Value	Type	Relevant Buttons
__temperatureSensorData__output	3.0e+002	FL0A764	-1.0e+002, 3.0e+002
__temperatureSensorData__localIState_Unit_Delay	3.0000007102030e+002	FL0A764	-1.0e+012, 1.0e+012
temperature	3.0e+002	FL0A764	-1.0e+002, 3.0e+002
__localIState_Unit_Delay	-3.33333324124e+003	FL0A764	-3.33333324124e+003, 3.0000007102030e+002

図 16. temperatureSensor テストベクタ

Req-ID により、要求から、デザインモデル、テストベクタへとトレーサビリティが取れる。

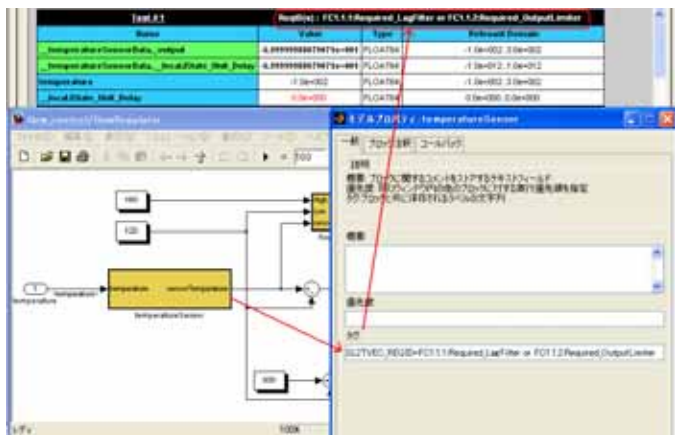


図 17. Simulink とテストベクタの要求リンク

上記で生成されたテストベクタを VGS の機能を用いてテストドライバー化する。これは、TBrun にインポートされ、テスト対象プログラムと共にコンパイル及びリンクされ、MULTI M32R シミュレータにダウ

ンロードされる。その後実行され、テスト、カバレッジの結果が解析される。

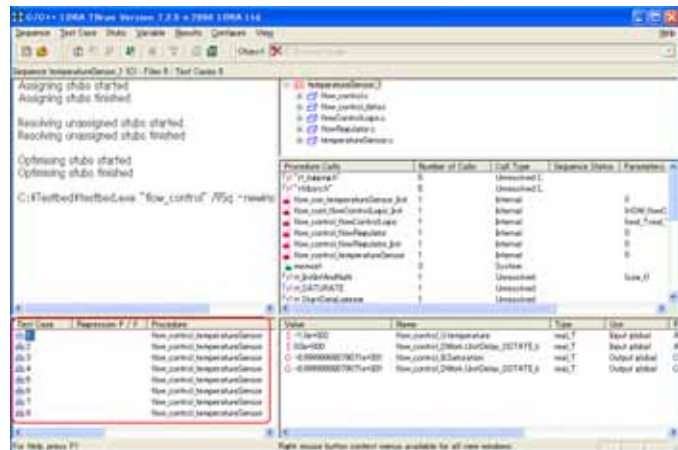


図 18. テストドライバーファイルのインポート

テストのカバレッジは 100%となった。また、期待値と実行値の比較結果は全て OK となり、要求モデルとソースコードの一致性が検証できた。

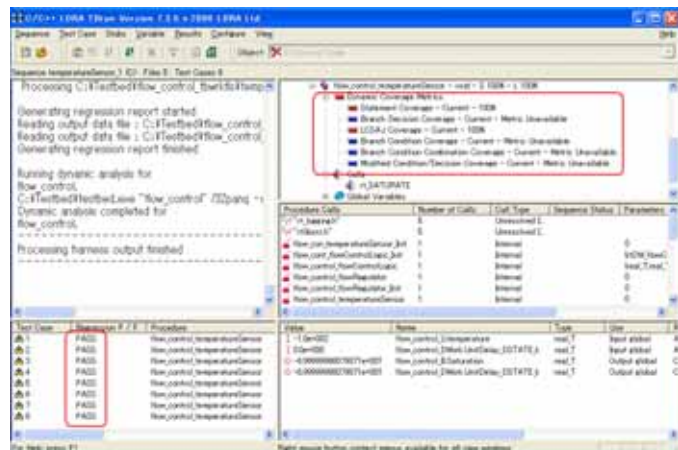


図 19. カバレッジ結果

Test	Object	Expected	Actual	Result
1	__temperatureSensorData__output	-6.9999998079071e+001	-6.9999998079071e+001	OK
1	temperatureSensorData__localIState_Unit_Delay	4.9999998079071e+001	4.9999998079071e+001	OK
2	temperatureSensorData__output	2.09999996423721e+002	2.09999996423721e+002	OK
2	temperatureSensorData__localIState_Unit_Delay	2.09999996423721e+002	2.09999996423721e+002	OK
3	temperatureSensorData__output	-1.0e+002	-1.0000000000000e+002	OK
3	temperatureSensorData__localIState_Unit_Delay	-1.000000007e+010	-1.0000000070000e+010	OK
4	temperatureSensorData__output	3.0e+002	3.0000000000000e+002	OK
4	temperatureSensorData__localIState_Unit_Delay	1.0000000021e+010	1.0000000021000e+010	OK
5	temperatureSensorData__output	3.0e+002	3.0000000000000e+002	OK
5	temperatureSensorData__localIState_Unit_Delay	3.0000000000000e+002	3.0000000000000e+002	OK
6	temperatureSensorData__output	-1.0e+002	-1.0000000000000e+002	OK
6	temperatureSensorData__localIState_Unit_Delay	-1.000000000e+002	-1.0000000000000e+002	OK
7	temperatureSensorData__output	-1.0e+002	-1.0000000000000e+002	OK
7	temperatureSensorData__localIState_Unit_Delay	-1.0e+002	-1.0000000000000e+002	OK
8	temperatureSensorData__output	3.0e+002	3.0000000000000e+002	OK
8	temperatureSensorData__localIState_Unit_Delay	3.0e+002	3.0000000000000e+002	OK

図 20. temperatureSensor テスト結果

#### 4. 考察

要求モデル，デザインモデルを解析してテストベクタを生成し，これらテストベクタをソースコードに対して実行することで得られるテスト結果，カバレッジ結果から，モデルとソースコードの一致性の検証が行えた．また，Req-ID によって，要求モデル，デザインモデル，テストベクタとトレーサビリティが取れ，要求が十分にテストされたことが証明された．

また今回のレポートには，MULTI M32R シミュレータを使用したけど，実ターゲット環境（ターゲット：ARM、ICE：ローターバツハ社）でも同様に実施したのものもある（下図 21）．これについては別途資料で説明する．



図 21. ターゲット環境

#### 5. おわりに

本稿で紹介したこのような開発プロセス全体に渡るテストの自動化フレームワークを実現するモデルベーステストでは，要求仕様の改善，より良いテストの生成，テスト自動化による効率など多くの成果が見込まれるが，大規模なシステム，要求にも適用可能であるかの証明がないと産業界では用いられることは無い．ライフサイクルを通して適応可能であること，容易で柔軟に拡張が出来るモデリング環境と言語，あらゆる環境におけるテスト実行までサポートできること，更には投資対効果が早期に明らかになること．これらの要件に加えて，要求管理ツールがモデルベーステスト環境に統合されることが実製品開発では求められる．

T-VEC ツールは，DOORS 要求管理ツールを TTM 要求モデリングツール，更には Simulink デザインツールを介して統合し，モデルからのテスト生成，テストドライバ生成を一貫してサポートしている．これにより要求からテストに至るトレーサビリティが提供され，産業界の実製品開発で実践的に用いられるようになっている．

更にこのプロセスを通じて得られる以下の効果は，採用を動機付けるものとなっている．

##### 1. リンク付けを意識して要求をモデル化する作業．

モデルにリンクされていない要求は見出され，まだモデル化されていないだけなのか，あるいはモデル化できない・テストできないなど改善が必要かを考察できる．

##### 2. 漠然とした要求は，モデル化段階で明らかにされる．

例えばある要件をモデルの 1 テーブル全体や，複数のテーブルに渡ってリンクしないといけない場合は複雑になるので，テーブルの一要素ごとへリンクできるように要件を効率的に分解・整理する作業を通じて．

##### 3. 要求へのリンクが存在しないモデルが明らかになる．

デザインサイクル以降で追加された，要求仕様書へ含められるべき要件．

##### 4. 要件がテストケースとリンクする．

実装のテスト結果が要求仕様にリンクされるので，テストで明らかになったエラーの解析にかかる時間とコストを飛躍的に削減できる．要求モデル，デザインモデルにハイパーリンクされる機能は，更なる効率に貢献する．

##### 5. プロジェクト，プロセスを計測し管理する．

モデルベース開発，テストで得られる成果物をメトリクスとし，それらの要求へのトレーサビリティは進捗管理の重要な情報となる．[2]

この資料で紹介したモデルベーステストによるトレーサビリティは、多くのユーザ事例によって支えられている。

TAF のモデルベーステストによるテスト自動化によるコスト削減効果について[3, 4]、要求使用の早期欠陥抽出による改訂作業削減効果[5]、高信頼性システムの欠陥を体系的に特定する方法論[6]、更に拡張したデザインモデル検証法 [7]、セキュリティ分野への適応事例 [8, 9]。

## リファレンス

[1] Blackburn, M.R., R.D. Busser, Automatic Generation of Test Vectors for SCR-Style Specifications. Proceeding of the 12th Annual Conference on Computer Assurance, June, 1997.

[http://www.t-vec.com/download/papers/tvec\\_scr2tvec.pdf](http://www.t-vec.com/download/papers/tvec_scr2tvec.pdf)

[2] Blackburn, M.R., Objective Measures from Model-Based Testing, STAREAST, Orlando, May 2004.

[3] Statezni, David, Industrial Application of Model-Based Testing, 16th International Conference and Exposition on Testing Computer Software, June 1999.

[4] Statezni, David. Test Automation Framework, State-based and Signal Flow Examples, Twelfth Annual Software Technology Conference, May 2000.

[5] Safford, Ed, L. Test Automation Framework, State-based and Signal Flow Examples, Twelfth Annual Software Technology Conference, May 2000.

[6] Blackburn, M. R., R.D. Busser, R. Knickerbocker, R. Kasuda, Mars Polar Lander Fault Identification Using Modelbased Testing, NASA Software Engineering Workshop, November 2001. See

[http://www.software.org/pub/taf/downloads/mars\\_polar\\_lander\\_2001.pdf](http://www.software.org/pub/taf/downloads/mars_polar_lander_2001.pdf).

[7] Busser, R.D., L. Boden, Adding Natural Relationships to Simulink Models to Improve Automated Model-based Testing, Digital Avionics Systems Conference, October 2004. See

<http://www.software.org/pub/externalpapers/papers/busser-2004-2.doc>.

[8] Chandramouli, R., M. R. Blackburn, Model-based Automated Security Functional Testing , 7th Annual Workshop on Distributed Objects and Components Security (DOCSEC), Baltimore, MD, April 2003.

[9] Blackburn, M. R., R. Chandramouli, Using Model-Based Testing to Assess Smart Card Interoperability Conformance, International Conference on Computing, Communications and Control Technologies, Austin, August 2004.