社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－SE－163 (38)
2009／3／19

シンプルかつ現実的なモデルベース・テストツールの提案

居駒幹夫† 大場みち子† 酒井三四郎††

†(株)日立製作所 ソフトウェア事業部 ††静岡大学 情報学部

モデルベーステストとはソフトウェアテスト関連のタスクをテスト対象のモデルを基本に使うアプローチである．本論文では，既存のソフトウェアテストの資産や，確立されているソフトウェアテスト向けの既存技術を再利用できるモデルベースのソフトウェアテストツールを紹介する．このソフトウェアテストツール，RandLは，熟練したテスト技術者による，ソフトウェアの新規開発向けだけでなく，普通のテスト技術者による再利用ベースのソフトウェア開発にも使用できる．

A Simple and Practical Model-Based Software Testing Tool

Mikio Ikoma† Michiko Oba† Sanshiro Sakai††

†Software Division, Hitachi, Ltd. ††Dept. of Computer Science, Shizuoka University

Model-based testing is an approach that bases common testing tasks on a model of the implementation under test. This paper proposes a model-based software-testing tool that enables to reuse past test resources and the established techniques for software testing. The software testing tool, called RandL, can be used not only for developing new software products by skilled testing engineers, but also for the reuse-based software development projects by average testing engineers.

1. はじめに

1970 年代以降，ソフトウェアの形式手法をソフトウェアテストにも活用する手法が多く提案されている．1970 年代から 1980 年代前半には形式的なソフトウェア開発が主目的で，ソフトウェア開発の形式化によりテストは不要になると期待されていた．1990 年代に入ると，実際のソフトウェア開発方法に即したモデルベースのテスト手法が多く提案された．しかし，21 世紀に入り，すでに最初の 10 年が経過しようとする現在，モデルベーステストが実際のソフトウェア開発の手法として根付いているとは言えない．Hartman[1] は，モデルベーステストの適用率を高々5%．1%に近いレベルと推定している．

一方，社会の基幹システムが情報化するに従い，基幹システムで使われるソフトウェアの大規模化，複雑化が進んでいる．また，そのようなソフトウェアの信頼性が損なわれることにより，社会的な影響を与えるような事故も多数発生している．これらのソフトウェアの品質を評価，保証するための手段としてのソフトウェアテストもその重要性がより認識されるようになってきた．

しかし，現状のソフトウェアテストのほとんどは手作業によるテストである．自動化テストというのは，形式的な仕様から自動的にソフトウェアテストを行うことではなく，画面系のソフトウェアテストをテスタが対話的に入力しなくてもできるソフトウェアテスト手法を指しているに過ぎない．同値分割，直交法によるテスト項目選択といった，ソフトウェアテスト固有の技法も形式的な手法というよりも，テスタがテスト項目を考える際の基礎知識といった側面が強い．この結果，経済産業省の調査[2]によると，ソフトウェアテストの工数は増加し，組込みシステムの場合，テストに関わるコストは全開発コストの50%以上を占めている．

本論文は，以上のようなソフトウェア開発の現状に基づき，ソフトウェア開発におけるテスト作業に容易に活用可能なモデルベース・テストツールの要件を提案する．また，

この要件に従って開発したソフトウェアテストツールを紹介し，その有効性を評価する．

2. モデルベーステストとは

モデルベーステストとは，テストケース作成やテスト結果の評価といったテスト関連のタスクを，テスト対象のモデルを基本に使うアプローチ[3]である．同様な用語としてモデル駆動テスト(model driven testing)がある．Hartman[1]によると，モデル駆動テストはモデルベーステストの特殊なケースであり，テスト対象を表す何らかの抽象モデルから自動的にテストを導出する手法である．

本論文では，狭い意味でのモデル駆動テストではなく，テスト対象のモデルをソフトウェアテストの実行に利用するという広義のモデルベーステストを対象とする．なお，西ら[4]は，詳細設計レベルのモデルや，テスト手法自体のモデルをベースにしたテスト手法も，モデルベーステストの一種としている．しかし，本論文ではテスト対象の形式的な機能モデルを使用したテスト手法を対象とし，それ以外のモデルを使用したテスト手法はモデルベーステストとしては扱わない．

3. モデルベーステストの分類と課題

Hartman[1]は，既存のモデルベーステストを5種類に分類して説明している．

(1) 事前条件・事後条件モデル
- 多くの形式的記述言語．Spec#, JML, B, UML with OCL, Z など

(2) 状態遷移モデル（FSM）
- ステートチャートや状態遷移図，状態遷移マトリクスなど
- GUI 系のテストやリアクティブシステムのテストなどに用いられる

(3) トレースベースドモデル
- シーケンス図やコラボレーション図
- 通信系のテストなどに用いられる

(4) 代数的モデル
- プロセス代数，ペトリネットなど

(5) 統計モデル
- ユーザプロファイルモデルなど

このうち，状態遷移モデルやトレースベースのモデルは，組み込みソフトウェアを中心に設計用のモデルとしてソフトウェア開発の現場で活用されている[2]．しかし，これらのモデルもほとんどはソフトウェアの設計が目的で，このモデルをベースにしたテストの普及は進んでいない．Hartman[1]は，ソフトウェア開発でモデルベーステストを使わない理由として，以下の4点を挙げている．

(a) 方法論の課題
方法論が複雑すぎ，平均的なテスタのスキルでは困難．

(b) テスト生成の課題
テスト生成の制限．厳密なモデルではテスト不能なほど状態の爆発する．

(c) ツールの課題
既存ツールのユーザビリティが悪い．

(d) 開発組織の課題
テスタが上流工程から関わるような体制になっていない．

これらの理由は，モデルベーステスト，特に狭義のモデル駆動テストが使われない理由としては正しいが，モデルベーステストという名前で，90 年代後半以降に登場しているツール（例えば，IBM 社の GOTCHA[5], Microsoft 社の AsmL[6]）が普及しない理由としては不十分だと考える．

ソフトウェア開発の実態を考えたとき，モデルベーステストを普及させるためには，Hartman の挙げた理由よりも，以下の課題を解決すべきであると考える．

(e) 新規開発以外での適用の課題
ほとんどのソフトウェア開発作業は，開発済みのソフトウェアの機能追加である．開発済みのソフトウェアをテスト用にモデル化する意義が薄い．また，現状の多くのモデルや方法論では，蓄積されたテスト資産の再利用を考慮されていない．

(f) テスト技術との連携の課題
ソフトウェアテストを効率的，効果的に実行するには，同値分割，境界値分析，カバレージ分析等の確立されたソフトウェアテスト固有の技術が必要であるが，現状の多くのモデルは連携困難である．

(g) テスタのモチベーションの課題
昨今の大規模ソフトウェア開発では，ソフトウェアテスタが職種化されてきている．モデルベーステストが普及するためには，テスト実行者であるテスタが何らかのメリットをモデルベーステストに見出す必要がある．現状の多くのモデルベーステスト手法は，形式化設計の研究の副次的な効果としてテストもできるというアプローチのものが多い．専門化されたソフトウェアテスタが誇りを持って使えるような方法論にはなっていない．

4. 現実的なモデルベーステストの要件と解決策

4.1. ソフトウェアテスタ向けのモデルベーステストの要件

3章で述べたモデルベーステストの課題を解消し，これらの手法，ツールをソフトウェアテスタが使うようになるためには，以下の要件を満足する必要があると考える．

要件１：テストという観点でのモデル化

内部構造等，現実的なテストで必要の無いような部分の形式的なモデル化を強要しないこと．さらに，既存のテスト技術で実行可能なモデルが提供できること．

要件２：既存のテスト方法からの連続性

テストプロセスの連続性．テスト資産（テストスィート，テスト基準，テストデータ，テスト結果）等が継続使用できること．

要件３：コスト対効果

モデルベーステストを採用することにより，テストの効率性，テストの十分度，テストにかかるコストという観点で効果があること

要件４：一般的なテスタで実行可能

一般的なテスタのスキルでモデル化及び，テストが実行できること

4.2. 現実的なモデルベース・テストツールの開発方針

(1)テスト観点でのモデル化

ソフトウェアテストのモデル化を考える場合，対象とするソフトウェアの内部の構成や動作よりも，外部からみた動作をモデル化する．既存のモデルのうち，UML のクラス図やステートチャートなどの，ソフトウェアの内部的なモデルよりも，LOTOS[7]，TTCN[8]といった外部からの動作を形式的に記述できるタイプのモデルを採用する．ただし，テスト対象の実装全体の厳密な動作モデルを形式化した場合，実際にテスト不能なテストケースを大量に生成する危険性がある．従って，LOTOS や TTCN における一部のポートの振る舞いを部分的に形式化できるようにする．

(2)既存テスト資産，テスト技術との連携

従来から持っている，テスト用のプログラム，ライブラリを最大限に再利用できることを目標とする．さらに，従来，テストケース作成のための技術として主に活用されている，同値分割，直交表等のソフトウェアテスト固有の確立されている技術をツールとして実装できるようにする．

(3)テスタのレベルに合わせた活用

テスト技術に対するスキルレベルや，形式化に対するスキルレベルなどに合わせ，初心者でも活用が可能で，さらに熟練者に対しては，厳密な形式化も可能なツールとする．

5. モデルベーステスタ RandL の紹介

5.1. 概要

RandL は，(株)日立製作所ソフトウェア事業部で開発中のソフトウェアテストツールである．本ツールは，当事業部の製品検査部署のツールとして，すでに活用実績のある検査ツールをベースにして，現在，多言語対応，テスト技術連携等の汎用化を進めている．現在開発中の機能については，本章の説明で《開発中》と注記する．

RandL は，既存のテストプログラムをグルー言語的に呼び出すことが可能なほか，既存のテストプログラムから，ライブラリとして呼び出されることも可能なモデルベース・テストツールである．RandL による，ソフトウェアテストの実行イメージを図１に示す．RandL 言語で記述されたモデル及び，既存のテストプログラムを入力に，RandL 言語プロセサは，一部形式化されたテストプログラムソースを生成し，これのコンパイル結果と，RandL が提供するランタイムにより，モデルベーステストが実行できる．

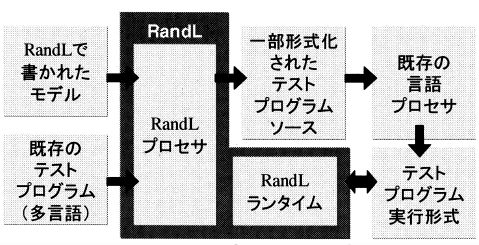

図1 RandL を使ったソフトウェアテスト自動実行のイメージ

さらに，多言語化対応，既存テスト技術との連携し，手動テスト用に最適化されたテスト項目を出力する用途(図 2)にも使用できる

機能を開発中である．《開発中》

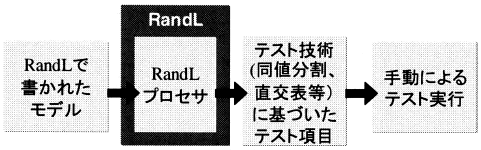

図2 RandL を使ったソフトウェアテスト項目出力のイメージ

5.2. RandL のプログラム構成

RandL は，図1の通り，言語プロセサ部分と，ランタイム部分から構成される．

(1)言語プロセサ

既存のテストプログラムと，IUT をテストするためのモデルを記述したソースを入力し，既存のテストプログラムを一部形式化したソースを出力する．出力されたソースは，テストプログラムが書かれた言語のコンパイラで実行形式とする．言語プロセサは，既存テストプログラムのプログラム言語に非依存であることを目標に設計したが，現状は，一部 C/C++言語に展開されることを前提にした処理がある（《開発中》）．

(2)ランタイムライブラリ

RandL 言語プロセサが出力した，プログラムを実行する際に必要となるランタイムライブラリ．この部分は，言語に依存している．現状は，C 言語と C++言語に対応したライブラリ関数を提供している．

5.3. RandL 言語の概要

RandL 言語プロセサの入力となる RandL 言語の全体の構成と，モデリングの主要部分である，テストシーケンスと，データの部分の特徴的な機能について紹介する．

(1)全体構成

RandL 言語の全体構成を図3に示す．

```
@option:
  // オプション(言語，実行環境等)．省略可．
@sequence:
  // 形式化されたテストシーケンス記述部．
@data:
  // 形式化されたデータ定義記述部．省略可．
@preamble:
  // 最初に一回だけ起動される．省略可．
@postamble:
  // 最後に一回だけ起動される．省略可．
@subroutine:
  // 既存のテストプログラム等．省略可．
```

図3 RandL 言語の全体構成

@option 部では，RandL で書かれたテストプログラムが，主プログラムとして動作するかサブルーチンで動作するかといった，テスト環境を記述する．@preamble 部，@postamble 部，@subroutine 部には，テストプログラム（現状 C/C++）のソースが記述，またはインポートできる．そのテスト資産をどのように実行するかという指定を，テストシーケンス部(@sequence 部)，データ定義記述部（@data 部）に記述できる．本論文では，モデルベースの核となるテストシーケンス部，データ定義記述部の特徴的な機能と，主なランタイム関数を紹介する．

(2)テストシーケンス部の文法

テストシーケンス部の基本的な文法は，OSI 製品の適合性試験用に開発された言語 TTCN-1[7]の動的ふるまい記述(dynamic behavior description)を参考にした．すなわち，各行の先頭カラムの位置が左側にあるほど，早く実行され，同じカラム位置の行は選択的に実行されるツリー構造を記述できる．ただし，RandL 言語では，カラム位置と先頭記号部分は規定するが，それ以外の部分は，独自の言語ではなく，既存のテストプログラムの記述言語（現状は，C/C++のみ）をそのまま記述できるようにする．

最も簡単な例として，テストを順番に実行する場合の RandL のテストシーケンス例を図4に示す．各文の先頭文字！は，テスト実行を表す．先頭文字の後は，この例の場合，C ライブラリの関数をそのまま記述している．ここに，@subroutine 部で記述した既存テストライブラリの関数を書いても良い．

関数の実行を順番に実行したい場合は，行ごとに開始カラム数を増やしながら指定する．

```
【記述方法】
@sequence:
 !printf("a");
   !printf("b");
     !printf("c");
【結果】
 出力は abc
```

図4 順序実行の例

複数の関数のどれかを実行させたい（選択実行の）場合，それらの関数の開始カラムを同じにして隣り合わせの行に指定する(図 5)．

```
【記述方法】
@sequence:
 !printf("a");
   !printf("b");
   !printf("c");
【結果】
出力は 50%の確率で ab 又は ac
```

図5 選択実行の例

複数の関数の発行順序をランダムにしたい（ランダム順序実行の）場合，先頭記号を＃にして，それらの関数の開始カラムを同じにして隣り合わせの行に指定する（図６）．

```
【記述方法】
@sequence:
 !printf("a");
   #printf("b");
   #printf("c");
【結果】
出力は 50%の確率で abc 又は acb
```

図6 ランダム順序実行の例

２つ以上の関数を指定した確率で実行させたい（パーセント実行の）場合は選択記号？のあとに文字％と実行させたいパーセント数を記入する(図７)．

```
【記述方法】
@sequence:
  ?%50
     !printf("b");
  ?%30
     !printf("c");
【結果】
 50%の確率で b，30%の確率で c，残りの
 20%の確率で何も出力しない．
```

図7 パーセント実行の例

関数の戻り値を参照して制御を振り分ける場合も，選択記号？を使用する．(図８)

```
【記述方法】
@sequence:
 !malloc(1000);
   ?NULL
     !printf("malloc NG¥n");
   ?OTHER
     !printf("malloc OK¥n");
【結果】
 malloc 関数の結果が NULL のとき(失敗時)
 "malloc NG"を出力する．それ以外の時は
 "malloc OK"を出力する．
```

図8 結果判定の例

(3)データ定義部の文法

データ定義部では，テストプログラムで使用するデータの定義域(入力ドメイン)，即ち，データの仕様を定義できる．整数，文字列といった，基本型のデータだけでなく，複雑な構造をもったデータ（バイナリを含む構造体，ASN1 の BER 等）も記述できる．固定の値を記述するのではなく，データの定義域を指定することにより，そのデータを呼び出されるごとに，そのデータの定義域に従った可変値を返すようにできる．この機能を使って，RandL を単に既存プログラムのテストデータ生成ツールとして使用することもできるし，シーケンス部の振る舞いと連携して，形式的なテスト実行を記述することもできる．

整数型のデータの記述例を図９に示す．

```
【記述方法】
@data: //データ定義
 i0_99  num1 //０以上９９以下の整数
 i1,2,4,8        num2 //1,2,4 か 8 の整数
 i1_16,32_128    num3 //1～16 か 32～128
@sequence:
  printf("%d,%d,%d", _num1, _num2, _num3);
【結果】
 例えば，0,4,99 など
```

図9 整数型のデータ定義

定義されたデータの呼び出し方により，同値分割や境界値解析と言ったソフトウェアテスト技法に従ったテストが可能となる《開発中》．図１０の例の場合，整数の変数 num4 の定義域指定から同値分割，境界値分析手法を使い，num4 をテストするために重要な値からテストできるように値を生成する．

```
【記述方法】
@option:
 loop 100;
@data: //データ定義
 i1_16,32_128    num4 //1～16 か 32～128
@sequence:
  printf("%d,", _e_num4);
【結果】
 例えば，1,16,32,128,2,4,8,64,...
```

図10 テスト技術との連携例《開発中》

構造型のデータを記述する場合，テストシーケンス部の記述と同様に，記述するカラム位置に依存した，内部構造の確率的な選択が

可能である(図１１).

```
【記述方法】
@data:
+ table1 // 構造型のデータ定義
   "magic    "     // 定数
   ?%50
     i(4)0-100    int1 //4バイトの整数
   ?%50
     d(4)         dec1 //4バイトの整数文字
【結果】
  50%の確率で整数型，50%の確率で整数文字
  型の値が選ばれる.
```

図11　構造型データでの確率的な選択例

その他，文字列，16 進バッファ，固定長の構造型データ，可変長データ，ASN.1 の基本エンコーディング(BER)等の指定が可能である.

(3)他の主な機能

どのように RandL が実行したかを追跡できるトレース機能．障害発生時に，その原因を追究するために使用できるリラン機能，格納されたデータのコレクションからランダムな順番で取り出すことができるランダムバッファなどの機能をランタイム関数として提供している(表 1).

表1　主なランタイム関数

ランタイム関数	概要
rnd_quit()	実行プロセス/スレッドの中止，または全体の停止
rnd_abort()	ダンプの採取および全体の停止
rnd_restart()	プロセスの停止および再実行.
rnd_que()	データのランダムバッファへの書き込み
rnd_deque()	データのランダムバッファからのランダムなデキュー
rnd_seeque()	データのランダムバッファからのランダムな読みだし
rnd_purque()	ランダムバッファのクリア
rnd_trace_on()	トレースの開始
rnd_trace_off()	トレースの中止
rnd_pid()	実行中のプロセスのＩＤ
rnd_tid()	実行中のスレッドのＩＤ
rnd_loop()	現在のループ数
rnd_max_loop()	最大のループ数
rnd_etime()	実行時間

テストシーケンス記述部や，@preamble 部，@postamble 部，@subroutine 部から表１のランタイム関数を発行できる.

5.4.　RandL 言語の評価

RandL のベースとなっているソフトウェアテストツールは，日立製作所社内で開発された大規模ソフトウェアのテストで適用実績があり，従来の，手操作のテストでは摘出困難なソフトウェアフォールトを摘出している.

(1)RandL 言語の評価

4.2 節で述べた，モデルベース・テストツールを普及させるための要件に対する RandL の評価をまとめる.

要件1:テスト観点でのモデル化

外部からの動作モデルをトリー状に記述できる文法を採用したことにより，テストの実施者から見て違和感が無く，かつテストの自動化に必要な部分的な形式化に特化したモデルが記述できるようになった.

要件2:既存のテスト方法からの連続性

既存のテストプログラムのグルー的なスクリプトとしても利用でき，また，既存のテストプログラムからサブルーチン的に呼び出すこともできる機能を提供した．これにより，既存のテスト資産を捨てることなく，テスト作業の連続性を確保できた.

要件3:コスト対効果

モデルを記述することによるコストは，形式化する部分を必要最小限にしたことにより，従来のモデルベーステストと比較して小額で実施可能となった．一方，ソフトウェアテストの十分度は手作業のテストに比べ格段に向上する．定量的な評価は未実施であるが従来のモデルベーステストに比べて格段にコスト対効果は向上していると考える.

要件4:一般的なテスタで実行可能

一部の機能や，一部の入力ドメインのみの形式化を可能にしたため，形式化のスキルがないテスト技術者でも容易にモデルベーステストが実施できるようにした.

(2)RandL 言語の現状の課題

現状の RandL の実装は，以下の課題がある.

- 対応言語が C 言語，C++言語のみ
- テストスクリプト等が完備している場合に有効だが，手作業ベースのテスト実行環境では使用困難

これらの課題に対応するため，RandL 言語

プロセサの特定言語依存処理を局所化し，どのような言語のテストでも使用できるように改造中である．また，本質的に手作業が必要なソフトウェアテスト対応に，既存テスト技術と連携した，テスト項目の自動作成機能（図2参照）を追加していく予定である．

6. 終わりに

本論文では，現状のモデルベーステスト技術の課題を明確化し，この課題を解決するための，現実的なテストツールの要件を明確にし，この要件に従って開発中のテストツールRandLの主な機能を紹介した．

今回紹介したツールは現状(株)日立製作所社内でのみ使用されている．今後は本ツールを社外に公開し，実用的なモデルテストツールとして活用を推進する．さらに，本ツールの機能の拡充，従来テスト技術の連携強化といった面でも，オープンな技術として公開していく予定である．

参考文献

1) Alan Hartman：“Model Based Testing: What? Why? How? and Who cares?,”, http://www.haifa.il.ibm.com/dept/services/papers/ISSTAKeynoteModelBasedTesting.Pdf, 2006.
2) 経済産業省商務情報政策局情報政策ユニット情報処理振興課: 2008年版組込みソフトウェア産業実態調査報告書: http://www.meti.go.jp/policy/mono_info_service/joho/downloadfiles/2008software_resarch/project_houkokusho.pdf
3) Ibrahim K. El-Far and James A. Whittaker: Model-based Software Testing: http://www.geocities.com/model_based_testing/ModelBasedSoftwareTesting.pdf
4) 西康晴，JaSST’07東京実行委員会: モデルベースドテスト入門: ソフトウェアテストシンポジウム2007東京: 2007:http://www.jasst.jp/archives/jasst07e/pdf/D4-1.pdf
5) G. Friedman, A. Hartman, K. Nagin, T. Shiran, Projected State Machine Coverage for Software Testing, to appear in Proceedings of ISSTA 2002 International Symposium on Software Testing and Analysis (July 2002).
6) Microsoft 社研究所の Web ページ: AsmL: Abstract State Machine Language: http://research.microsoft.com/en-us/projects/asml/
7) ISO 9646-3 Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation (TTCN)