情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-SLDM-144 No.12
Vol.2010-EMB-16 No.12
Vol.2010-MBL-53 No.12
Vol.2010-UBI-25 No.12
2010/3/26

ドメイン特化型開発における自動化テストプロセスの提案

森　奈実子[†1]　久住憲嗣[†2]
中西恒夫[†3]　福田晃[†3]

ドメイン特化型開発では，特定の市場領域に属するソフトウェアを効率的に開発できる DSL（Domain-Specific Language）を定義し，開発者はその言語を用いてソフトウェアを開発する．しかし，ドメイン特化型開発では特有のテストプロセスは定義されていない．そこで，本稿では DSL の分類に応じたテストケース自動生成手法を援用したテストプロセスを提案する．ケーススタディとして小規模な DSL に提案手法を適用し，その結果を基に従来手法との比較を行った．

A Proposal of Automatically Testing Process for Domain-Specific Modeling Language

Namiko Mori,[†1] Kenji Hisazumi,[†2]
Tsuneo Nakanishi[†3] and Akira Fukuda[†3]

In Domain-Specific Development, developers define DSL(Domain-Specific Language) which enables them to easily develop software that belong to a specific market domain. However, the peculiar testing process has not been defined in Domain-Specific Development. This paper proposes a testing process with test case generation for Domain-Specific Development. We applied this approach to small-scale DSL as a case study, and compared the result with that of the existing approach.

[†1]九州大学大学院 システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
[†2]九州大学システム LSI 研究センター
System LSI Research Center, Kyushu University
[†3]九州大学大学院 システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu University

1. はじめに

組み込みソフトウェア開発の現場では，ユーザの多様な要求に合わせて，サービスや機能が類似した複数の製品を同時並行的に開発する必要に迫られている．そこで，特定ドメインに属する製品群を効率的に開発するドメイン特化型開発[1)] に注目が集まっている．

ドメイン特化型開発では，特定の市場領域に属するソフトウェアを効率的に開発できる DSL（Domain-Specific Language）を定義し，開発者はその言語を用いてソフトウェアを開発する．ドメイン特化型開発は，ドメインに属するソフトウェアの実装の大部分を自動化する．これによって，全くの手動でソースコードを記述する場合と比べて，開発工数を大幅に削減し，より品質の高いソフトウェアの開発が可能となる．ドメイン特化型開発は，おおまかにドメイン定義・要件整理，DSL の定義，コード生成器の開発の 3 工程から構成される．しかしながら，ドメイン特化型開発では特有のテストプロセスは定義されていない．そのため，多くのドメイン特化型開発では，開発手法の特性にあったテストプロセスを使わず，従来のソフトウェアテスト技法のみを使ったテストプロセスを利用している．また，ドメイン特化型開発ではコード生成器が最終的なソフトウェアの品質を決定するため，品質を保証するにはコード生成器自体の品質が問題となる．ドメイン特化型開発は入力する DSL コードに応じて少しずつ異なるソフトウェアを生成する．その性質上，生成できるソフトウェアは多種多様であり，コード生成器の品質確保は難しいといえる．高品質が要求される組み込みシステム製品開発現場へのドメイン特化型開発の導入には，その特性に合ったテストプロセスの実現が課題となる．

一方，ドメイン特化型開発の基礎となるソフトウェアプロダクトライン開発方法論（Software Product Line Engineering）[2)3)]（以降，プロダクトライン開発方法論）がある．プロダクトライン開発方法論は，開発資産の計画的な再利用により，共通の特徴を有する製品群を効率的に開発することを目的としたソフトウェア開発方法論である．再利用される資産には，ソースコードだけでなく，テストケースなども含まれる．ドメイン特化型開発においても，テストケースなどの共有のテスト資産を積み重ねていき，アプリケーション開発時に再利用することで，テスト効率向上が期待できる．しかしながら，ドメイン特化型開発で生成されるソフトウェアの中には，DSL コードに応じて状態数が増えるなど，劇的にテストケースが変わる場合もあり，単純な再利用では対応できない．

そこで本研究では，グラフィカル形式の DSL である DSML（Domain-Specific Modeling Language）に焦点を当てて，ドメイン特化型開発のテストプロセスを提案する．その際

Vol.2010-SLDM-144 No.12
Vol.2010-EMB-16 No.12
Vol.2010-MBL-53 No.12
Vol.2010-UBI-25 No.12
2010/3/26

にシステムの性質を記述したモデルからテストケースを自動生成する MBT（Model-based Testing）手法[4] を援用することでテスト効率の向上を目指す．また，ドメインごとに DSML の性質は異なるため，DSML モデルごとの特性を考慮したテストケース生成手法を提案する．ケーススタディとして代表的な状態遷移モデルの DSML に提案手法を適用し，従来手法と比べた評価と考察を行う．

2. プロダクトライン開発方法論とドメイン特化型開発

この章では，プロダクトライン開発方法論，ならびに今回研究対象とする，プロダクトライン開発方法論を基盤としたドメイン特化型開発について概説する．

2.1 プロダクトライン開発方法論

プロダクトライン開発方法論とは，ソフトウェア開発資産の再利用による金銭的・時間的開発コストの削減を目標とするソフトウェア開発方法論である．プロダクトライン開発方法論では，特定の市場領域のニーズを満たす，特徴（フィーチャ）の似通った製品群を一括してひとつの開発対象とする．この開発対象としてまとめられた製品群を特にプロダクトラインという．このプロダクトライン内でプロダクト間の共通部と可変部を抽出し，それらの開発資産を戦略的に再利用し，組み合わせて製品を開発することで開発コストを低減させる．

そのためプロダクトライン開発方法論では，再利用可能資産を構築するフェーズと資産を再利用して個別の製品を開発するフェーズの二つに開発プロセスが分離されている．これら二つの開発プロセスは，それぞれドメインエンジニアリングとアプリケーションエンジニアリングと呼ばれる．さらに，これらを管理，支援するマネジメント活動が重視されており，これら 3 つがプロダクトライン開発方法論における基本活動とされる．

2.1.1 プロダクトライン開発におけるテスト

ドメインエンジニアリングでは主にプロダクトラインの共通部を，アプリケーションエンジニアリングでは主に可変部を含むテストを行うことになる．プロダクトライン開発におけるテスト戦略は，テストの設計，実行をドメインエンジニアリングとアプリケーションエンジニアリングのどちらでそれぞれを重点的に行うかによって大まかに 3 パターンに分類できる[2)5)]．

- アプリケーションエンジニアリングでのテスト設計・実行
- ドメインエンジニアリングでの可変性を考慮したテスト設計
- ドメインエンジニアリングでのテスト設計・実行

アプリケーションエンジニアリング毎にテストを実行することは従来の手法を用いる利点があるものの，再利用できるテストケースを何度も開発することで工数に無駄が生まれる．開発する製品が十分に多い場合，ドメインエンジニアリングでテスト設計を行うことが早期の品質保証と工数の観点から推奨される．

2.2 ドメイン特化型開発

ドメイン特化型開発は，特定目的向けの言語を設計することによってソフトウェア開発の生産性や品質を向上させるソフトウェア開発方法論である．ドメイン特化型開発では，アプリケーション開発者は C 言語や Java といった汎用プログラミング言語（General Purpose Languate；GPL）による開発をするのではなく，ドメイン特化言語（Domain-Specific Language；DSL）と呼ばれる特定の市場領域に属するソフトウェアを効率的に開発するための言語を開発し，その DSL を用いてソフトウェアを開発する．DSL を利用することで，少ない記述量で GPL で書かれたソースコードを自動生成できるようになるため，繰り返し同じドメインに属するソフトウェアを開発する場合，大幅に開発効率を向上させることができる．また，DSL にはテキスト形式のものとグラフィカルなモデル形式のものがあり，後者を特にドメイン特化モデリング言語（Domain-Specific Modeling Language；DSML）と呼ぶ．DSML はソリューションを直感的で分かりやすい図として視覚化できるので，テキスト形式の DSL よりも開発経験のないユーザに親しみやすい．

2.2.1 DSML を用いる開発の流れ

DSML を利用する開発の流れを図 1 に示す．また，DSML 環境を DSML モデルを入力してテンプレートからソフトウェアを生成する環境と定義したとき，DSML 環境は以下に示す 4 要素で構成される．

- DSML モデル：DSML を用いた製品開発において要件を表現するために DSML ユーザが DSML の文法に則って具体的に記述したモデルである．
- ドメインモデル：DSML モデルに記述する際の文法や制約を定義するモデルである．
- コード生成器：DSML ユーザが記述した DSML モデルを基にテンプレートから，開発対象の製品に必要なソフトウェアのソースコードや設定ファイルを生成するものである．
- テンプレート：DSML モデルの記述に従って，ソフトウェアを生成する生成規則が記述された，再利用される開発資産である．

DSML を用いてアプリケーションを開発する場合，まず DSML ユーザは，その要件を DSML モデルで記述する．この時，DSML モデルにアプリケーションのデータ構造あるいは振る舞い，もしくはその両方を記述するかは，DSML 文法を定義するドメインモデルに依存する．次にその DSML モデルをコード生成器に入力することで，コード生成器はテン

Vol.2010-SLDM-144 No.12
Vol.2010-EMB-16 No.12
Vol.2010-MBL-53 No.12
Vol.2010-UBI-25 No.12
2010/3/26

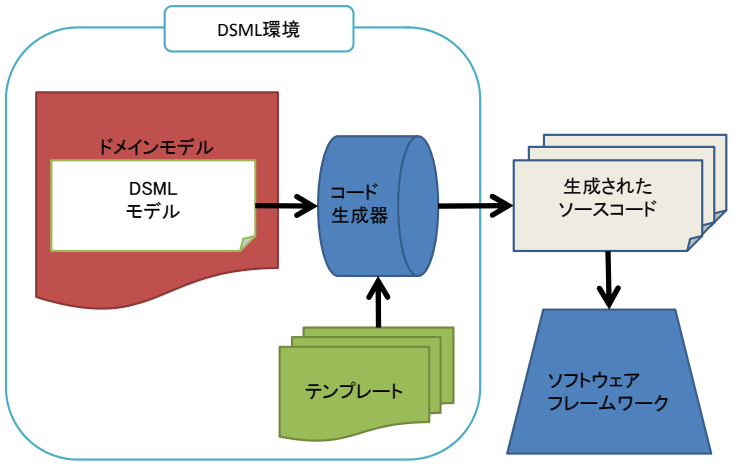

図 1 DSML を用いた開発の流れ

プレートに基づきソースコードを生成する．

2.2.2 DSML の開発工程

DSML の開発工程は以下の 3 工程から成る[1)6)7)8)]．

(1) **ドメイン定義・要件整理**：DSML 開発者はドメイン専門家の意見を受けて，DSML で開発できるアプリケーションの範囲を定義する．次に対象ドメインに属するアプリケーションが持つべき要件であるフィーチャを洗い出し整理することで，DSML で表現すべき機能を決定する．

(2) **DSML の定義**：まず作成したフィーチャモデルに従って，DSML を設計・開発する．その際に，以下に示す，DSML を構成する 2 項目を定義する．

- **ドメインモデルの定義**
- **表記法の定義**

また，テキスト形式の DSL の場合は，汎用的なテキストエディタで DSL が記述可能となるためツールを開発する必要はないが，DSML の場合は DSML モデルを記述するためのツールを開発する必要がある．

(3) **コード生成器の開発**：先に定義した DSML モデルによる生成物とそれを生成するためのテンプレートを定義する．

2.2.3 ドメイン特化型開発におけるテストの問題点

第 2.1.1 節では，プロダクトライン開発の推奨するテストアプローチとして，ドメインエンジニアリングでのテスト設計を挙げている．ドメインエンジニアリングでテスト設計を行うことで，テストケースの再利用ができ，工数の削減に繋がる．フィーチャと関連付けられたコア資産のテストケースのみを抽出することで，このようなテストケースの再利用が可能になる．しかし，これはテスト対象が含む可変性によってテストケースが劇的に変化する場合には有効ではない．DSML では，その表記法となるモデルの特性によってテストケースの変化が激しい．例えば，状態遷移を表すような DSML モデルの場合，構成要素が増えるたびに状態や遷移が増えるため，状態遷移テストのテストケースが大きく変わる．

また，前節で述べた通り，DSML では特定のソフトウェアを想定した DSML モデルは DSML 全体のテストケースとして見られる．それは，可変点の増加に伴い，DSML のテストにかかるコストも膨大になることを意味する．DSML はドメインの可変点で選択肢を自由に取ることができるため，k 個の可変点のそれぞれの選択肢を $\boldsymbol{N} = \{n_1, \cdots, n_k\}$ とすると，$\Pi_{i=1}^{k} n_i$ 個のソフトウェアを生成可能となる．それら全てを網羅してテストを行うことは非常に困難であり，またそれぞれのテストケースも膨大になる．

この問題に対して，最も単純な解決方法としてテストケースの自動生成手法が挙げられる．その解決方法のひとつとして，ドメイン特化型開発では，コード生成器によってテストケース生成を行う．通常コードを生成するために使われる DSML モデルはテストを生成するには貧弱であるが，DSML モデルの情報を基に開発する製品に適切な既存のテスト手法を選択できる．テスト手法や製品概念を利用して，テストケースとテストされるソースコードを生成できる[6)]．しかしながら，上記の方法ではテストケース自動生成器をすべて手作業で開発することになり，開発コストが高くなる．そこで，本研究ではテストケース自動生成手法の Model-based Testing（MBT）手法を援用する．具体的には DSML モデルおよび仕様書から MBT ツールへの入力となるモデルを生成し，MBT ツールによってテストケースを生成する．ただし，ドメイン特化型開発では開発する度に性質のことなる DSML が定義されるため，そのモデルの特性に応じたテストケース生成が必要であると考える．DSML としては，状態遷移モデル，データフローモデル，データ構造，その他の多数の特性を持つ言語が考えられる．また，MBT 手法では，事前・事後条件モデルなどの形式モデル，UML ステートチャート図やラベル付け遷移システムなどの状態遷移モデル，その他多数のモデルを利用して，テストケースを生成する．そこで本研究では，DSML の性質を整理・分類し，その性質に応じた MBT 手法の援用によるテストプロセスを提案する．

Vol.2010-SLDM-144 No.12
Vol.2010-EMB-16 No.12
Vol.2010-MBL-53 No.12
Vol.2010-UBI-25 No.12
2010/3/26

3. 要素技術

この章では，提案手法で用いる要素技術である，Model-based Testing 手法と，プロダクトライン開発方法論のひとつである PLUS（Product Line UML-Based Software Engineering）のパラメータ化有限状態機械について概説する．

3.1 Model-based Testing

MBT（Model-based Testing）手法はシステムが意図した振る舞い，あるいはその環境の振る舞いを明白に記述したモデルを使ったソフトウェアテスト手法である．モデルの入出力は実装したコードの入出力と解釈できるため，そのモデルからテストケースを自動的に生成してブラックボックステストを行う．

MBT 手法のモデルは，その理論的枠組みから分類ができる．MBT 手法はブラックボックス・テストの一種であるため，そこで用いられるモデルは基本的にはソフトウェアの入出力関係や振る舞いなどに着目したものが中心である．それらは組み合わせモデル，形式モデル，トレース・ベースドモデル，代数的モデル，状態遷移モデル，統計モデル，フローモデルの 7 種に大別できる[4]．

このように MBT 手法はモデルを入力としてテストケースを自動生成できるので，機械可読なモデルを定義して開発を行うドメイン特化型開発と親和性が高いと考えられる．したがって，DSML モデルから MBT 手法で扱えるモデルを生成することでテスト効率の向上が期待できる．

3.2 Product Line UML-Based Software Engineering

PLUS（Product Line UML-Based Software Engineering）は米 George Mason 大学の H.Gomma 教授によって開発されたオブジェクト指向プロダクトライン開発方法論である[9]．PLUS の特徴は UML 図を用いてコア資産を構築するという点である．また，開発プロセスの各手順が手続的に明快に示されているという特徴も有する．PLUS では，開発の各プロセスにおいてほとんど全ての設計資産が UML で表され，これらの資産はプロダクトラインで共有される．それゆえ，UML 表現において共通部と可変部が明確になるような工夫がされている．

PLUS におけるドメインエンジニアリングの工程のひとつに，状態遷移モデリングがある．状態遷移モデリングで作成されるオブジェクト間の状態遷移図は，可変部も記述される．その状態遷移図はパラメータ化有限状態機械と呼ばれる．選択的に備えられるフィーチャと択一的に備えられるフィーチャが状態，遷移，処理に影響を与えるとき，パラメータ化有限状態機械は，そのフィーチャを Boolean 型のガード条件として記述する．

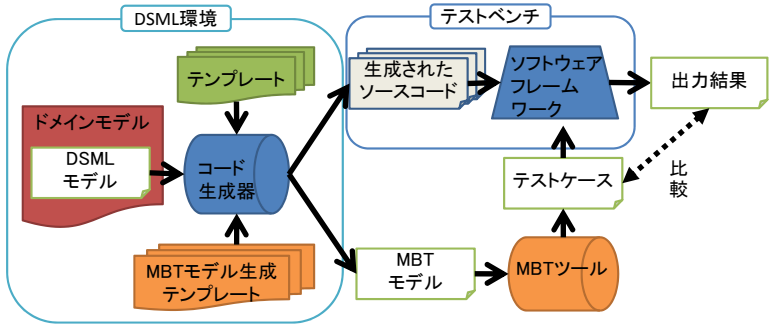

図 2 MBT モデル生成テンプレートを用いたテストの一連の流れ

4. テストケース自動生成手法を用いた DSML のためのテストプロセス

第 2.2.3 節で提起した問題を踏まえ，本研究では DSML の特性を考えて，テストケース自動生成手法を用いたテストプロセスを提案する．テストケース生成の具体的な手段として，機械可読な言語を使用することで親和性の高い MBT 手法を利用する．

本章では，まず提案手法で用いられる用語の定義をした後，提案する手法の概要やテストプロセスについて解説する．次に事前に述べたプロセスでも，特に説明が必要なものに焦点を当てて詳細に述べる．

4.1 概　　要

本研究では，テストケース自動生成手法を援用した DSML 環境のためのテストプロセスを定義して，それを適用することでドメイン特化型開発におけるテスト効率の向上を目的とする．具体的には，DSML モデルの特性を加味して，DSML モデルから MBT ツールで扱えるモデルを自動生成し，入力として MBT ツールに与えることでテストケースの自動生成を可能にする．図 2 に MBT ツールを援用したテストの一連の流れを示す．

提案テストプロセスは，テスト設計，テスト実行の 2 段階に分けられる．テスト設計では，まず MBT によるテストケース生成のために用いる MBT モデル生成テンプレートをコア資産として開発する．次に，MBT モデル生成テンプレートを再利用してテスト対象の DSML モデルから MBT モデルを生成して，そのモデルを入力としたテストケースを生成する．テスト実行では，生成したテストケースを基にテストを実行する．

Vol.2010-SLDM-144 No.12
Vol.2010-EMB-16 No.12
Vol.2010-MBL-53 No.12
Vol.2010-UBI-25 No.12
2010/3/26

提案手法適用のタイミングは，ドメイン特化型開発におけるテストの設計，実行のタイミングに依存する．後述する提案手法の各工程において，工程 1〜5 はテスト設計で行い，工程 6 はテスト実行で行う．前章で述べたプロダクトライン開発方法論でのテスト戦略に従うと，提案手法適用のタイミングは以下の 3 パターンが挙げられる．

- ドメインエンジニアリングでテスト設計，アプリケーションエンジニアリングでテスト実行
- ドメインエンジニアリングでテスト設計，テスト実行
- 上記の二つのアプローチの組み合わせ

提案手法を適用する前に，3 パターンのうちどのテスト戦略を行うかはその利点・欠点を加味してテスト計画で選択する必要がある．

4.2 テストプロセス

提案手法では，以下のような六つの工程を定義する．

工程 1　DSML の種類判別：テスト対象の DSML モデルの特性・文法に応じて，モデルの種類を判別する．判別する指標として，モデルの構成要素とその各要素間の関係の特徴を参考にする．

工程 2　MBT 手法の決定：形式モデル，状態遷移モデルを入力とする MBT 手法から，テストに用いるものを決定する．入力として用いるモデルの特性で分類された MBT 手法と各 DSML モデルとのテスト容易性の関係を指標を参考にする．また，利用する MBT 手法に対応した MBT ツールも決定する．

工程 3　MBT モデル生成テンプレート開発：DSML モデルから，決定した MBT ツールの入力モデルの生成を行うテンプレートを開発する．設計書などに基づいて各 DSML モデルから，MBT ツールへの入力となる MBT モデルや設定ファイルを生成する際の指標を参考にする．必要な設計書が存在しない場合は，DSML 開発者の意見を基に作成する．また，1 種類以上の有効な DSML モデルをテストケースとして，MBT モデル生成テンプレートのテストを行う．

工程 4　MBT モデル生成：テスト計画で決定した，テスト対象のソフトウェアの DSML モデルをコード生成器に入力し，MBT ツールの入力として必要な MBT モデルや設定ファイルを生成する．また，同時にテスト対象のソフトウェアのソースコードも生成する．

工程 5　テストケース生成：MBT モデルや設定ファイルを MBT ツールに入力し，テストケースを生成する．生成されたテストケースがテスト実行には抽象的すぎる場合は，テストケースの具体化を行う．

工程 6　テスト実行：テストケースをもとにテストを実行し，コード生成器およびソフトウェアフレームワークの不具合がないかを確認する．発見した不具合を修正し，テスト計画で定めたカバレッジに達するまで繰り返しテストを実行する．テスト結果をステークホルダに報告するためのサマリレポートにまとめる．

次節以降では，MBT モデル生成テンプレート開発までの 3 工程それぞれについて詳細に説明する．

表 1　DSML モデルの構成

モデルの種類	モデル	構成要素	要素間の関係
静的モデル	データ関連構造モデル	クラス	関連，汎化，集約，依存
動的モデル	状態遷移モデル	状態	遷移
	データフローモデル	プロセス	データの流れ
	制御フローモデル	処理または判断	制御の流れ

4.3 DSML の種類判別

この工程では，テスト対象の DSML モデルの特性に応じて，モデルの種類を判別する．判別する指標として，モデルの構成要素とその各要素間の関係の特徴を参考にする．

DSML モデルは構造を表す静的なモデルと振舞いを表す動的なモデル静的なモデルに分類できる．そこで DSML モデル図式を判別する際に指標となる構成要素と要素間の関係の組み合わせを表 1 に示す．

DSML モデルの種類の判別には，そのモデルが意図する意味を理解することが非常に重要である．静的モデルと動的モデルを混同することは少ないが，動的モデル同士の混同は多いので判別には注意が必要である．動的モデルはニュアンス次第で，状態遷移とも制御フローとも取れることがあり，ステークホルダーである DSML 開発者とテスト設計者の相互理解が欠かせないだろう．

4.4 MBT 手法の決定

この工程では，形式モデル，状態遷移モデルを入力とする MBT 手法から，テストに用いるものを決定する．その際に，入力として用いるモデルの特性で分類された MBT 手法と各 DSML モデルとのテスト容易性の関係を指標を参考にする．

第 3.1 節にて示した MBT モデルのうち，形式モデルおよび状態遷移モデルが，提案手法において適用する MBT 手法の入力モデルとして妥当と判断する．ここで，形式モデルおよび状態遷移モデルを利用する MBT 手法と，モデルの特性で分類した DSML モデルとのテ

Vol.2010-SLDM-144 No.12
Vol.2010-EMB-16 No.12
Vol.2010-MBL-53 No.12
Vol.2010-UBI-25 No.12
2010/3/26

表 2 MBT モデルと DSML モデルとのテスト容易性の関係

		DSML モデル			
		データ関連構造モデル	状態遷移モデル	データフローモデル	制御フローモデル
MBT モデル	形式モデル	＋	0	＋	0
	状態遷移モデル	−	＋	−	0

スト容易性の関係を表 2 で示す.「＋」は，DSML モデルの情報を最大限に利用でき，MBT モデル生成テンプレートを開発しやすいためテスト容易性が高いことを示し，逆に「−」は，DSML モデルで利用できる情報が少なくテンプレートを開発することが難しいためテスト容易性が低いことを示す．また,「0」は DSML モデルと DSML モデル以外の成果物の情報の利用バランスがほぼ同等であり，ややコストはかかるもののテンプレートを開発できることを示す．テスト容易性については，次章で述べる MBT モデル生成テンプレート開発が容易かどうかで判断している．

4.5 MBT モデル生成テンプレート開発

この工程では，DSML で扱われるモデルから MBT ツールで扱えるモデルを生成するためにテンプレートを開発する．この節では，各 DSML モデルを対象に MBT モデルの生成手法を概説する．生成する MBT モデルとしては，第 4.4 節で DSML テストへの援用の有効性を述べた形式モデルと状態遷移モデルを対象とする．また，状態遷移 MBT モデル生成テンプレート開発の際に利用する可変点付き状態遷移図について解説する．

4.5.1 各 DSML モデルからの MBT モデル生成

- **状態遷移 MBT モデルの生成**：状態遷移 MBT モデルで必要な情報として，状態，遷移，事象などの振る舞いに関する情報が挙げられる．静的モデルであるデータ関連構造 DSML モデルは，クラス同士の関連などが記述されており，その振る舞いについては言及されていない．したがって，データ関連構造 DSML モデルから状態遷移 MBT モデルの生成は，DSML モデルだけではほぼ不可能である．動的モデルのデータフロー DSML モデルも状態，事象という概念が欠如しているため，同様である．また，制御フロー DSML モデルには事象がプロセスの一部として記述される場合があるので，DSML モデル内に状態遷移 MBT モデルを生成する情報を持つと考えられる．しかし，状態遷移 DSML モデルのようにほぼ直接的に変換して生成することはできない．また，その状態遷移 DSML モデルでも生成されるソフトウェアの全ての状態遷移を網羅していない場合がある．このような場合は，ソフトウェア全体の状態遷移を網羅するために，状態遷移図などの設計書を利用することで MBT モデルの生成を補完することができる．この利用できる設計書のひとつとして可変点付き状態遷移図を作成することで，より MBT モデル生成テンプレートの開発が容易になる．
- **形式 MBT モデルの生成**：形式 MBT モデルで必要な情報として，変数の値域やデータ型などが挙げられる．データ関連構造 DSML モデルやデータフロー DSML モデルでは，クラスの属性の名前・データ型，データ変数などの情報が得られるので，それらを基に設計書から変数の値域などの情報を抽出することができると考える．しかし，制御フロー DSML モデルや状態遷移 DSML モデルは基本的にはデータについての情報は持たない．ただし DSML モデルの文法で構成要素が持つ属性を定義することが多く，その名前・データ型などは利用できる．それらを利用して，テンプレートおよび設計書から変数のデータ型や変数の値域の情報を抽出する必要がある．

4.5.2 可変点付き状態遷移図

本節では，状態遷移 MBT モデルの生成で作成されるべき可変点付き状態遷移図について詳しく解説する．

先に述べたが，各 DSML モデルから状態遷移 MBT モデルの生成には状態遷移図の作成が必要になる．これはほぼ直接変換ができる状態遷移 DSML モデルにおいても，DSML モデルに生成されるソフトウェアの状態遷移が網羅されていない場合は適用される．それらは以下の二つの場合である．

- ドメイン固有の暗黙知が存在する．
- 要件レベルの抽象度では見えない状態遷移が存在する．

ここで作成する可変点付き状態遷移図は，第 3.2 節で解説した PLUS で作成されるパラメータ化有限状態機械を基にしている．この可変点付き状態遷移図では，DSML モデルで選択する可変点によって変化する状態，遷移，処理をガード条件を用いて表現する．ガード条件は，各可変点で選択されるフィーチャの名前と論理演算子を用いて条件を定義している．モデルの構成要素が保持するフィーチャが可変のフィーチャとなる場合は,「Element.x」のように構成要素の名前（Element）とフィーチャの名前（x）を関連付けて表記する．

このような状態遷移図を作成することで，各可変点の選択によって変更される箇所が明らかになる．この状態遷移図を参考に，可変のフィーチャを条件式とした条件文を記述したテンプレートを開発することで，容易に DSML モデルから，状態遷移 MBT モデルに変換できる．

Vol.2010-SLDM-144 No.12
Vol.2010-EMB-16 No.12
Vol.2010-MBL-53 No.12
Vol.2010-UBI-25 No.12
2010/3/26

5. ケーススタディと検討

本章では，ケーススタディとして提案したテストプロセスを小規模な DSML のテストに適用し，さらに従来手法と比較することで提案手法の評価を行う．

5.1 ケーススタディ対象の DSML

ケーススタディ対象は，著者らが開発した，交通信号機システムを対象ドメインとした DSML である．

第 4.2 節で述べたとおり，提案手法を利用するタイミングはドメイン特化型開発におけるテストの設計，実行に依存する．本ケーススタディでは，ドメインエンジニアリングでテストの設計・実行を行い，アプリケーションエンジニアリングでもテストの実行を行うことを想定する．ドメインエンジニアリングでのテストでは，代表的な信号機システムとして以下をテスト対象とする．

テスト対象 1　十字路に配置される，ニ対の定周期式車両用信号機を有する信号機システム

テスト対象 2　車両用信号機と歩行者用信号機を有する押ボタン式信号機システム

アプリケーションエンジニアリングでのテストでは，ドメインエンジニアリングではテストしなかった機能を有する信号機システムとして以下をテスト対象とする．

テスト対象 3　六ツ角交差点などに配置される，4 種の感応式歩行者用信号機付き車両用信号機を有する信号機システム

5.2 提案手法の適用

信号機システム DSML に対して，提案手法を適用する．

工程 1　DSML モデルの種類判別：信号機システム DSML モデルの文法について書かれた仕様書やドメインモデルから，DSML モデルの構成要素と要素間の関係の意味を読み取る．本ケーススタディでは，構成要素はその信号機が青，黄，赤に変化し，他の信号機は赤の状態と捉え，要素間の関係は「設定した時間が経過した」あるいは「押ボタンが押された」という事象をラベル付けした遷移と捉える．したがって，信号機システム DSML モデルは状態遷移モデルであると判別できる．

工程 2　MBT 手法の決定：前工程にて，テスト対象の DSML モデルは状態遷移モデルと判別した．表 2 の DSML モデルと MBT モデルとのテスト容易性の関係から，本ケーススタディでは状態遷移 DSML モデルに対して，最もテスト容易性が高い状態遷移モデルを入力とする MBT 手法を用いる．状態遷移モデルを入力とする MBT ツールとして Conformiq 社の Conformiq Qtronic[10] を利用する．

工程 3　MBT モデル生成テンプレート開発：第 4.5.2 節で述べたように，状態遷移 DSML モデルから状態遷移 MBT モデルを生成するときには，場合によっては可変点付き状態遷移図を作成する必要がある．本ケーススタディの対象である信号機システム DSML は，ドメイン固有の暗黙知を持つため，可変点付き状態遷移図が必要になる．可変点付き状態遷移図を作成した後は，それを参考に MBT モデル生成テンプレートを作成する．本ケーススタディでは，Conformiq Qtronic への入力として UML のステートマシン図と，Java によるステートマシン図上のデータを定義した設定ファイルを生成するテンプレートを開発した．

工程 4　MBT モデル生成：本ケーススタディでは，先に述べたテスト対象の DSML モデルを作成し，前工程で開発したテンプレートに基づき，Conformiq Qtronic への入力となるモデルと設定ファイルを生成した．また，先に述べたテスト対象のソフトウェアのソースコードも生成した．

工程 5　テストケース生成：Conformiq Qtronic に前工程で生成した入力を与えて，テストケースを生成した．各テスト対象に対するテストケースの数などは，第 5.4 節にて後述する．

工程 6　テスト実行：本ケーススタディでは，Conformiq Qtronic によって生成されたテストケースを，テスト対象のソフトウェアをソフトウェアフレームワーク上で実行した．

5.3 従来手法の適用

提案手法と比較するために，従来手法として状態遷移テストを信号機システム DSML に適用した．状態遷移テストとは，有効と無効の状態遷移を実行するテストケースを設計するブラックボックステストの設計技法である[11]．状態遷移テストの手順は以下に従う．

手順 1　テスト対象の信号機システムの状態遷移図を作成する．

手順 2　状態遷移図を基に状態遷移表を作成する．

手順 3　状態遷移表から有効と無効の状態遷移を実行するテストケースを作成する．

5.4 提案手法と従来手法との比較

表 3 に，コア資産開発時の実装工数とコード行数を示す．また，表 4 に，各テスト対象に対する，従来手法でのテストケース生成工数とテストケース数，提案手法でのテストケース生成工数とテストケース数を示す．ただし，従来手法はテストケース生成の際に，順次作成した成果物の中で再利用できるものは再利用しているため，テスト対象 1 より後のテストケース生成工数がやや削減されていることを明記しておく．また，テスト対象 3 において，従来手法と提案手法とのテストケース数に差があるが，カバレッジに差はない．提案手法で

Vol.2010-SLDM-144 No.12
Vol.2010-EMB-16 No.12
Vol.2010-MBL-53 No.12
Vol.2010-UBI-25 No.12
2010/3/26

表 3 提案手法に関するデータ

	工数 [人・分]	コード行数 [行]
MBT モデル生成テンプレート開発	425	680

表 4 従来手法と提案手法との比較

		従来手法	提案手法
テスト対象 1	テストケース生成工数 [人・分]	53	4
	テストケース数 [個]	1	1
テスト対象 2	テストケース生成工数 [人・分]	64	4
	テストケース数 [個]	7	7
テスト対象 3	テストケース生成工数 [人・分]	188	8
	テストケース数 [個]	85	84

は，有効な遷移と無効な遷移を同時にテストするテストケースを生成しているため，その数に違いがある．

表 4 から，テストケース生成工数は従来手法よりも提案手法の方が大幅に小さいことが分かる．しかし，MBT モデル生成テンプレート開発に 425 人・分かかっているため，総合的にみると提案手法の方がコストがかかっている．しかし，テストする DSML モデルの種類が増えるほど，従来手法のコストは劇的に増え，提案手法のコストはほとんど増えないことが分かる．したがって，従来手法と提案手法との比較によって，ドメイン特化型開発によって開発する製品が多いほど，テストケース生成にかかるコストを削減できる．

6. おわりに

本研究では，ドメイン特化型開発におけるテスト効率の向上のために，テストケース自動生成手法を援用したテストプロセスを提案した．テストケース自動生成手法として，評価対象のシステムのモデルを入力に用いる MBT 手法を利用する．提案手法では，DSML モデルおよび仕様書から MBT ツールへの入力となるモデルを生成し，MBT ツールによってテストケースを生成する．その際に，ドメインごとに異なる DSML モデルの特性と MBT 手法を利用したときのテスト容易性を整理した．また，各種 DSML モデルから状態遷移 MBT モデルを生成するテンプレートの開発のために，PLUS が提唱するパラメータ化有限状態遷移機械を基にした可変点付き状態遷移図を用いることを提案した．

ケーススタディでは，小規模な DSML である信号機システム DSML を対象として，提案手法を適用した．また従来手法である状態遷移テストを適用してテストケースを作成し，提案手法との比較を行った．その結果，1 製品分のテストケース作成にかかる工数は従来手法に比べて提案手法が 39%上回った．しかし，提案手法によるテストケース生成工数は従来手法の 4%〜8%の工数で済むため，ドメイン特化型開発で開発する製品が多いほど，テストにかかる工数は削減できることを示した．

今後の課題として，形式 MBT モデルの生成に対する更なる詳細化やより大規模な DSML や複数のモデルの特性を持つ DSML に対して提案手法の適用を検討し，更なる改善に取り組む．

謝辞 本研究は科学研究費補助金特定領域研究情報爆発 IT 基盤 (21013038) による助成を受けている．

参 考 文 献

1) Cook, S., Jones, G., Kent, S. and Wills, A.C.: *Domain-Specific Development with Visual Studio DSL Tools*, Addison-Wesley(2007).
2) Pohl, K., Böckle, G. and Linden, F.v.d.: *Software Product Line Engineering;Foundations, Principles, and Techniques*, Springer(2005).
3) Clements, P. and Northrop, L.: *Software Product Lines; Practices and Patterns*, Addison-Wesley(2001).
4) Utting, M., Pretschner, A. and Legeard, B.: A TAXONOMY OF MODEL-BASED TESTING, *Working paper series, ISSN 1170-487X, Department of Computer Science, University of Waikato*, No.04/2006(2006).
5) Tevanlinna, A. and Taina, J.: Product Family Testing: a Survey, *ACM SIGSOFT Software Engineering Notes*, Vol.29, No.2, pp.12–18(2004).
6) Kelly, S. and Tolvanen, J.-P.: *Domain-Specific Modeling; Enabling Full Code Generation*, WILEY-INTERSCIENCE(2008).
7) Mernik, M., Heering, J. and Sloane, A.M.: When and How to Develop Domain-Specific Languages, *ACM Computing Surveys*, Vol.37, No.4, pp.316–344(2005).
8) 一野 浩太朗, 久住 憲嗣, 井上 創造, 中西 恒夫, 福田 晃: センサネットワーク向けドメイン特化型言語の提案, 情報処理学会 *DICOMO シンポジウム 2009* 論文集, pp.1578–1586(2009).
9) Gomaa, H.: *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*, Addison-Wesley(2004).
10) Conformiq: Conformiq Qtronic, *http://www.conformiq.com/qtronic.php*.
11) Graham, D., Veenendaal, E.V., Evans, I. and Black, R.: *Foundations of Software Testing: ISTQB Certification*, Thomson Learning(2007).