Vol.2011-SLDM-149 No.40
Vol.2011-EMB-20 No.40
2011/3/19

ドメイン特化型開発における網羅性を考慮したテストケース削減手法の提案

岡田敬弘[†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，最終的なソースコードは自動生成されるため，ソフトウェアの品質は DSL 環境の構成要素であるコード生成器が決める．しかしながら，可変点の増加に伴うテストケースの爆発などの問題もあり，網羅的にテストを行うことは困難である．そこで，本研究では新たな網羅性基準とこの基準を満たすような最小のテストケースを機械的に作成する手法を提案する．ケーススタディを用いて手法の評価を行い，提案した網羅性基準に対するカバレッジが 100%となった．また，対象とした DSL の考えられる全テストケースからの削減率が約 3.6%という結果を得た．

A Coverage-Based Test Suite Reduction Method for Domain-Specific Modeling Languages

Takahiro Okada,[†1] Kenji Hisazumi,[†2]
Tsuneo Nakanishi[†3] and Akira Fukuda[†3]

In domain specific development, quality of software depends on the code generator which generates source codes from DSLs(domain-specific languages). However, testing all possible test cases is very difficult due to the test case explosion caused by existing many variable points in the DSL. This paper proposes a novel coverage measure which takes into account code generators for DSLs. The paper also proposes a methodology to reduce test cases automatically fulfilling the proposed coverage measure. We applied this methodology to a small-scale DSL for its evaluation. The evaluation indicates the proposed methodology can cover all elements of the proposed coverage measure and can reduce test cases by 3.6% of all possible test cases of the DSL.

1. はじめに

組込みソフトウェアの現場では，ユーザからの多様な要求により，機能やサービスが類似した製品を同時に開発するという必要性が生じている．そこで，特定のドメインに属する製品を効率的に開発するドメイン特化型開発に注目が集まっている．

ドメイン特化型開発では，あるドメインに適した開発言語であるドメイン特化言語 (DSL:Domain Specific Language) を定義し，開発者はこの DSL を用いてソフトウェアを開発する．DSL を用いた開発では，DSL 環境の構成要素であるコード生成器がソフトウェアのソースコードを自動生成する．ここで，ソフトウェアのソースコードが自動で生成されるという DSL の特性上，DSL によって開発されたソフトウェアの品質は，コード生成器によって決定される．そのため，コード生成器に対して十分なテストを行う必要があるが，テストケースの爆発などの問題により，適したテスト手法は確立されていない．

そこで，本研究ではグラフィカルなモデルからソースコード生成を行う DSML(Domain Specific Modeling Language) に着目し，莫大なテストケースから網羅性を保持したままテストケース数を削減する手法を提案する．また，この時新たに DSML に対する網羅性基準を定義する．この網羅性基準を満たすような最小のテストケースを機械的に作成する手法について述べる．また，本論文ではケーススタディとして小規模な DSL にこの手法を用いてテストケースを作成し，提案手法の評価と考察を行う．

本論文の構成は以下のとおりである．第 2 章では，ドメイン特化型開発とそのテストの問題点について述べる．第 3 章では，ドメイン特化型開発のテストに対する関連研究を紹介する．第 4 章では，DSML に対する新たな網羅性の基準の提案と，この網羅性基準を満たすようなテストケース削減手法について提案する．第 5 章では，実際に提案手法に基づいてケーススタディを行い，その結果から提案手法の評価と考察を行う．最後に第 6 章で，本研究のまとめについて述べる．

†1 九州大学大学院システム情報科学府情報知能工学専攻
Graduate School and Faculty 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

Vol.2011-SLDM-149 No.40
Vol.2011-EMB-20 No.40
2011/3/19

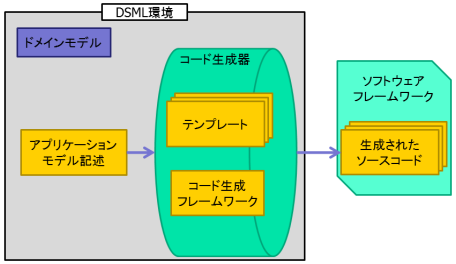

図 1 DSML を用いた開発の流れ

2. ドメイン特化型開発

ドメイン特化型開発[1] とは，プロダクトライン開発方法論を実現する開発手法の 1 つである．ドメイン特化型開発では，ソフトウェアの開発時に汎用プログラミング言語 (GPL : General Purpose Language) を用いるのではなく，あるドメインに適した開発言語であるドメイン特化言語 (DSL:Domain Specific Language) を定義し，開発者はこの DSL を用いてソフトウェアの開発を行う．DSL を用いた開発では，DSL の構成要素であるコード生成器がソフトウェアのソースコードを自動生成する．

ドメイン特化モデリング言語 (DSML : Domain Specific Modeling Language) は，問題空間における記述を直観的でわかりやすい図として表現することが可能なモデリング言語である．図 1 に DSML を用いた開発の概要を示す． 続いて，DSML の構成要素の説明を以下に示す．

- ドメインモデル:DSML を記述する際の文法や制約を定義したモデル
- アプリケーションモデル記述:ドメインモデルを基に開発者が具体的なアプリケーションを記述したモデル
- コード生成器:アプリケーションモデル記述を基に，テンプレートから必要なソースコードや設定ファイルを生成
- テンプレート:アプリケーションモデル記述に従い，ソフトウェアを生成する規則が記述されたソースコードや設定ファイル
- コード生成フレームワーク：アプリケーションモデル記述に従い，ソフトウェアを生成する規則を記述するためのフレームワーク

DSML を用いた開発では，開発者は開発したいアプリケーションの要件を DSML を用いて記述する．その後，記述したアプリケーションモデルをコード生成器に入力する．コード生成器は入力されたモデルに応じてテンプレートから必要なソースコードなどを選択し，自動的にアプリケーションモデルに適したソースコードを出力する．

2.1 DSML 開発の流れ

以下に DSML を開発する手順を示す[2)3)]．

(1) ドメイン定義・要件整理：DSML が対象とするドメイン領域を定義し，開発可能なアプリケーション群を定義する．また，要件整理においては，アプリケーション群が備えておくべき要件であるフィーチャを洗い出すことで，DSML の要件を定義する．

(2) DSML の定義：DSML の文法であるドメインモデルの定義と表記方法を決定する．また，DSML モデルを記述する際のエディタも開発する．

(3) コード生成器の開発：手順 1 で定義したアプリケーション群を生成するためのテンプレートを開発する．

2.2 ドメイン特化型開発におけるテスト

ドメイン特化開発における欠陥の要因としては以下が挙げられる．

- ドメインモデルの不具合：想定するソフトウェアを生成するモデルを記述できない不具合．無効な DSML モデルを作成できる不具合．
- アプリケーションモデルの不具合：開発者が，アプリケーションモデルを記述する際に要件を理解できずに，誤ったモデルを作成する不具合．
- テンプレート・コード生成器の不具合：有効なアプリケーションモデルを入力しても，モデルに沿った振る舞いのソースコードが得られない不具合．ソフトウェアフレームワークに沿ったソースコードが得られない不具合．
- ソフトウェアフレームワークの不具合：仕様書に対応した振る舞いをしない不具合．

ここで，ドメインモデル・アプリケーションモデルの不具合については，DSML の設計時に，想定するアプリケーション群が持つ要件を記述できるように設計を行うため，その評価については形式的検証技術などを利用するほかなく，本研究の対象外とする．

本研究が対象とするのは，テンプレート・コード生成器の不具合，ソフトウェアフレームワークの不具合とする．テンプレート・コード生成器のテストでは，あらゆる種類のアプリケーションモデルを入力とし，それに対応したソースコードが生成されているかを検証する．しかしながら，記述可能なすべてのモデルを網羅的にテストすることは現実的ではなく，どのようなモデルをテストケースとするかという戦略が必要となる．また，ソフトウェアフレームワークの不具合については，少なくとも 1 度全てのコードを実行しそれらが正

Vol.2011-SLDM-149 No.40
Vol.2011-EMB-20 No.40
2011/3/19

しく実行されることを確認する必要がある．

2.3 ドメイン特化型開発におけるテストの問題点

プロダクトライン開発におけるテストでは，共通性テストの再利用を行っている．これは，過去に行ったテストを再利用することで，工数の削減を図っている．しかしながら，この方法はテスト対象が持つ可変性によって，テストケースが大幅に変わる場合においては有効ではない．例えば，状態遷移の特性を持つ DSML においては，要素が増えるたびに状態・遷移が増えるため，テストケースが大幅に変更されてしまい，前述の戦略は利用できない．

また，前述したように，DSML で記述可能なあらゆるアプリケーションモデルも，DSML のテストケースとなり得る．つまり，可変点が増加すればするほど，テストに要する工数が爆発的に増加する．例えば，3 系列の値を持つ可変点が 10 種類ある場合の全組み合わせを考えると，59049 通りものテストケースが存在することになる．このように莫大なテストケースをすべて実行することは現実的ではない．

そこで，本研究ではテストの網羅性を確保しつつ，機械的にテストケース数を削減するための手法について提案する．この手法の提案と同時に，新たな DSML の網羅性基準として，コード生成フレームワーク記述に対するカバレッジ，アプリケーションのソースコードに対するカバレッジを同時に考慮できるカバレッジを提案する．

3. 関 連 研 究

テスト対象のソースコードに対してコード解析を行い，必要最低限のテストケースを発見する手法を紹介する．Reisner らは Symbolic Evaluation を用いたテストケース削減手法を提案している[4]．

Symbolic Evaluation は，各コンフィグ項目によって，各分岐命令でどのように分岐するかを解析し，全ての通りうるパスを列挙している．Reisner らは，この結果から全てのパスを通過するコンフィグの設定を解析し，それらの組み合わせをテストケースとして限定している．この論文中では，vsftpd/ngIRCd/grep の 3 つのソフトウェアに対して，この解析を行っている．

この解析の結果から，実行の手間としては 0.0001%〜1%程度に削減されている．また，コードに対するカバレッジは 60%程度とあったが，これはマルチプロセスでしか実行できなかったり，到達不可能なパスが含まれているためである．これらを勘案することで，実質のカバレッジは 80%程度となる．

しかしながら，この手法は DSML に対するパスの解析について，DSML においてはパスが変化する要因が分岐命令に限らないため，この手法をそのまま利用することはできない．

4. 網羅性を考慮したテストケース削減手法

本章では，網羅性を考慮しつつテストケース数を削減する提案手法について述べる．まず提案する網羅性の基準について述べ，その後この網羅性を満たすためのテストケース削減手法について述べる．

4.1 コードテンプレートを考慮した網羅性基準

DSML におけるテスト網羅性の対象として，生成されるソフトウェアに対する網羅性と，コード生成フレームワークの記述に対する網羅性が存在する．そこで，両者を考慮したカバレッジを定義する必要がある．

生成されるソフトウェアに対する網羅性基準には C0 カバレッジを用いる．ここで，C0 カバレッジを用いる理由としては，各ステートメントを必ず一回実行するため，デッドコードがないかや，ステートメントの正当性等を確認できること，C1・C2 カバレッジと比較すると，実行する組み合わせが減るため信頼性が下がるが，全数テストを行いやすいことが挙げられる．

続いて，コード生成フレームワーク記述について述べる．コード生成フレームワークの記述が行う処理は，アプリケーションモデルから値を取得してソースコードに埋め込む処理，取得した値を分岐処理用に加工する処理，生成するソースコードブロックを分岐によって変更する処理に大別できる．ここで，注意することとして，分岐処理は生成されるソフトウェアのカバレッジに関係しているということである．もし，あるソースコードブロックが 1 度も生成されなければ，生成されるソフトウェアのカバレッジを向上させることは不可能である．そのため，各分岐について全てのパターンでソフトウェアを生成する必要がある．

ここで，値埋め込み処理，分岐処理に使用するデータを処理する部分，生成するコードブロックを変更する分岐処理の 3 種類をステートメントと捉える．このステートメントは，本項で述べた全てのカバレッジ測定要素を過不足なく含む．以上のことから，このカバレッジ基準は，コード生成フレームワーク記述に対する C0 カバレッジと捉えることができる．

以上のことから，新たなカバレッジの定義を以下で与える．

$$Coverage(TAS) = \frac{Exe\_Statement(code) + Exe\_Statement(gen)}{Statement(code) + Statement(gen)} \quad (1)$$

ここで，分母は測定対象となる各ステートメント，分子は実際に実行したステートメント数である．

Vol.2011-SLDM-149 No.40
Vol.2011-EMB-20 No.40
2011/3/19

また，このカバレッジを TAS(Template-All Statement) カバレッジと呼ぶこととする．このカバレッジが 100%に達していないということは，テスト対象にデッドコードが存在する・テストケースが不十分等の理由により，1 度も実行していないテスト対象が残っているということである．これは，テンプレートの品質確保の面からも望ましくないことであり，DSML の開発においては可能な限り TAS カバレッジ 100%を満たすことが望ましいと言える．本論文中では，DSML のテストにおける網羅性の定義をこの TAS カバレッジで与える．

TAS カバレッジにおいて利用するカバレッジを変更した際の考察

TAS カバレッジで採用している C0 カバレッジでは，C1・C2 カバレッジと比較して全数テストを行いやすいという特徴がある．しかし，処理の組み合わせによる不具合の考慮ができない．最もソースコードに対する網羅性が高いものは C2 カバレッジであるが，DSML におけるテストの問題点であるテストケースの爆発が発生してしまうため，全数テストは現実的ではない．この問題は C1 カバレッジでも同様に発生する．しかしながら，C0 カバレッジのみでテスト可能な範囲は限定的であり，品質の確保という観点から考えると C1 カバレッジや C2 カバレッジの観点が必要となる．

そこで，文献 3) の手法を併用し，具体的なアプリケーションモデル記述からテストケースを作成しテストを実施することで，限定的な分岐網羅・条件網羅を達成することが考えられる．また，DSML が生成するアプリケーションの網羅的なテストが可能という点で優れていると言える．

また，テストしたい可変点の組み合わせなどが具体的に決定できる場合は直交表[5] を用いることも有効である．直交表を用いることで，特定の 2 可変点間の組み合わせ網羅率を 100%にすることができる．また，直交表への割り付け方を工夫することで，3 可変点間の組み合わせ網羅率を向上させることが可能である．直交表は，C1・C2 カバレッジのようにすべての組み合わせをテストするわけではないので，これらを 100%にすることは不可能である．しかしながら，特定の可変点間の組み合わせ網羅率を向上させることができるため，テスト対象の組み合わせが限定できる場合においては有効である．また，組込みシステムの不具合は 3 機能間の組み合わせで約 98%を網羅できるという調査結果も発表されている[6]．この調査結果を DSML にそのまま適用できるわけではないが，直交表のテストケース作成能力の高さを保証する 1 つの結果であり，この面からも直交表を用いることは有効であると言える．

4.2 網羅性を考慮したテストケース削減手法

本項では，先に提案した TAS カバレッジを最大にしつつ，テストケースを削減する手法について述べる．基本的な考え方として，文献 4) を参考とする．この手法により，生成されるソースコードについてはカバレッジを向上させることができる．しかし，この手法は DSML におけるコード生成フレームワークの記述に対するカバレッジを考慮することができない．そこで，この手法をコード生成フレームワーク記述に対しての考慮を行えるように拡張する．

コード生成フレームワーク記述が行う，モデルから値を取得してソースコードに埋め込むという処理については，従来の手法の中における 1 つのステートメントとして考えることで考慮することができる．また，モデルから値を取得してその値に応じて最終的なソースコードに組み込むコードを変更する処理については，分岐によって行う処理が変わるという，通常のソースコードと同様に見做すことができるため，Symbolic Evaluation における分岐の要因としてとらえることができる．

また，従来手法に対する DSML 特有の問題点として，別のパラメータを設定してソースコードを生成した場合でも，部分的に同じソースコードを含むソフトウェアを生成する可能性がある．従来手法は，解析にツリー構造を用いているため，このような構造が存在する場合には同じコードに対して何度もテストを行うという冗長なテストを行う可能性がある．そこで，このような共通な処理はある 1 つのパスで 1 度テストを行えば，その部分のカバレッジは満たしていると判断し，他のパスではテスト対象としないこととする．

以上より，提案手法の流れを以下に示す．

(1) コード生成フレームワーク記述における分岐点を洗い出す．
(2) 洗い出した分岐点毎に分岐条件を解析する．
(3) 各分岐において生成するソースコードのブロックを関連付ける．この際，このブロックにどのような値読み込み処理が含まれているかも合わせて解析しておく．
(4) 生成するソースコード内部における各分岐を洗い出す．
(5) 洗い出した分岐点毎に分岐条件を解析する．この時，ループ処理などによって，一連の処理の流れの中で結局すべてのパターンが満たされる分岐については考慮の対象外とする．
(6) 各分岐において実行するソースコードのブロックを関連付ける．
(7) 以上の手順で洗い出したすべてのパスを通るような，最小の可変点の組を探し，それらをテストケースとする．
(8) 生成するコードの分岐において，DSML の可変点で設定できない値を参照する際には仕様書の情報を参照し，テストケースに組み込む．

Vol.2011-SLDM-149 No.40
Vol.2011-EMB-20 No.40
2011/3/19

この手法の中で，特に手順 5 については注意が必要である．例として，無限ループの中である値を変化させていき，それが一定以上になった場合にループを抜けるという処理があった場合を考える．この時，ループ内で分岐処理が発生するが，最終的にどちらも通ることになるためここで考慮する必要がない．

また，各可変点に対して同値分割を行い，実行するべき可変点の値を限定する．本手法は TAS カバレッジを最大化することを目標としているため，主に代表値分析を行う．また，この考えはモデル要素についても同様のことが言える．

モデルの同値分割

モデル要素の同値分割とは，同じソースコードブロックを生成し，対象とするカバレッジ内容が同一である場合，それらは同値として判断する．これにより，テストケースにおけるモデル要素の簡易化を実現でき，より簡素なテストケース作成が可能になる．また，DSML で開発可能なアプリケーションは，可変点の組み合わせだけではなくモデル要素の組み合わせという意味でも多岐にわたり，テストケース爆発の一因となる可能性がある．しかしながら，ある網羅性基準に従いモデルの同値分割を行うことで，テストケースを有限に保つことができる．以上のことから，網羅性基準を満たすようなモデル要素の組み合わせを発見することは，テストケースの簡素化のみならず，テストケースを有限にするという視点からも有効である．

5. 評　　価

本章では，提案手法の評価を行う．手法の評価のために，ケーススタディとして信号機 DSML[3] を利用する．信号機 DSML は 1 つの交差点における信号機制御システムを開発可能である．信号機 DSML の構成要素として，車両用信号 (NormalSignal) と歩行者用信号 (PedestrianSignal) がある．車両用信号では青を点灯する時間 (Duration)，歩行者用信号の有無，感応式かどうかを設定できる．歩行者用信号では，青を点灯する時間を設定できる．

5.1 テスト空間

テスト空間とは考えられるすべてのテストケースの集合である．DSML のテスト空間は，設定可能な可変点の全組み合わせと開発可能なアプリケーションモデルの総数によって求められる．設定可能な可変点の組み合わせは，代表値分析などを行った結果 32 種類となった．また，開発可能なアプリケーションモデルについて考察する．今回対象とする DSML は，状態遷移的な特性を持ち，テストケースが無限になることが考えられる．しかしながら，DSML が対象としているドメインが信号機システムであるため，現実的には 6 つ角の信号機システム程度が最も複雑なシステムと捉える．これにより列挙されたアプリケーションモデルのパターンは，20 パターンである．以上より，信号機 DSML のテスト空間は 56 個である．

5.2 提案手法によるテストケース作成

信号機 DSML に対して提案手法を用いたテストケース作成を行う．まず，信号機 DSML のコード生成器のコード生成分岐についての解析を行う．ここではアプリケーションモデル中に，NormalSignal が存在しているか，NormalSignal における設定項目の歩行者用信号の有無・感応式かどうか，PedestrianSignal が存在しているかという分岐が存在していることが明らかになった．このうち，NormalSignal において歩行者用信号があると設定されている場合，PedestrianSignal が存在することは明らかなので，最後の分岐を内包する．

続いて，生成するソフトウェアについての分岐の解析を行う．ここでは，NormalSignal が存在しているか，PedestrianSignal が存在しているか，次に遷移する信号が感応式かどうかという分岐が明らかになった．信号機 DSML には，これら以外の分岐も多数存在するが，それらは信号機ソフトウェアを動作させた際に結局は分岐中の処理をすべて実行するような分岐であるため，ここでは考慮しない．また，信号機 DSML には DSML で記述できないパラメータによって分岐が決定される個所はないため，テストケースに新たなパラメータ項目を追加する必要はない．さらに，内部構造上，NormalSignal と PedestrianSignal の Duration については，どのような値を入力しても分岐が発生しないため，最低でも 1 種類テストすれば TAS カバレッジが満たされることも明らかになった．

さらに，モデルの同値分割について考察する．先に述べたモデルの同値分割の定義に則って信号機 DSML に対して適用した．この時，NormalSignal は歩行者用信号の有無，感応式かどうかの 2 つの可変点が同じであれば同じコードブロックを，PedestrianSignal は Duration に関わらず同じコードブロックを生成することが判明した．これにより，同じコードブロックを生成するモデル要素を複数持つようなアプリケーションモデルは，作成するテストケースに含まないこととする．

以上の分析の結果，作成したテストケースは 2 つとなった．作成したテストケースを表 1 に示す．

5.3 作成したテストケースに対する評価

まず，作成したテストケースと TAS カバレッジの関係について考察する．TAS カバレッジの各要素について，どの程度実行できているかを確認する．まず，各要素規模を計測する．今回適用した信号機 DSML は小規模であるため，ステートメント数と値埋め込み処理，

Vol.2011-SLDM-149 No.40
Vol.2011-EMB-20 No.40
2011/3/19

表 1 提案手法を用いて作成した信号機 DSML に対するテストケース

テストケース	入力クラス	入力クラスの値
テストケース 1	NormalSignal	Duration=20 歩行者用信号なし 感応式ではない
テストケース 2	NormalSignal1	Duration=30 歩行者用信号なし 感応式ではない
	NormalSignal2	Duration=25 歩行者用信号あり 感応式である
	PedestrianSignal	Duration=15

表 2 信号機 DSML における TAS カバレッジの各要素数と実行数

	計測した要素数	実行した要素数
生成するコードのステートメント数	219	219
コード生成フレームワーク記述の値埋め込み処理数	38	38
コード生成フレームワーク記述の分岐前処理数	15	15
コード生成フレームワーク記述の分岐処理数	3	3
合計	275	275

分岐前処理については手作業で計測した．また，コード生成フレームワーク記述の分岐処理については，前節で計測済みである．さらに，前節で作成したテストケースを実行した際に実行された要素数を計測した．これらの結果を表 2 に示す． この結果から TAS カバレッジは 100%となり，提案手法は TAS カバレッジを満たす手法として有用であるといえる．

続いて，どの程度テストケースを削減できているかについての評価を行う．先述した信号機 DSML のテスト空間と比較すると，テストケース数は約 3.6%と大幅に削減されている．大幅削減の要因として，モデルの同値分割によるアプリケーションモデルの削減が挙げられる．また，並列的に考慮することが可能な可変点については，1 つのテストケース中で同時に考慮することにより，より少ないテストケースでテストすることが可能であったことも大きな要因である．

6. おわりに

本論文では，ドメイン特化型開発におけるテスト効率を向上させるために，2 つの事項について提案した．1 つ目は，DSML 向けの新たな網羅性基準として TAS カバレッジを提案した．このカバレッジでは，生成するソースコードとコード生成フレームワークの記述を統合して考慮することを可能とした．2 つ目は，TAS カバレッジを満たす機械的なテストケース作成手法を提案した．

ケーススタディでは，小規模な DSML である信号機用 DSML に提案手法を適用した．この DSML のコード生成器に対して解析を行い，提案したテストケース作成手法を用いてテストケースを作成し，TAS カバレッジが 100%になることを確認した．また，信号機 DSML が持つテスト空間と作成したテストケース数を比較すると，約 3.6%と大幅にテストケースを削減できた．

また，今後の課題として，他の DSML で提案手法を適用することで一般性を向上させること，より大規模な DSML に適用し削減効果の違いを考察すること．提案手法を適用してテストした DSML を運用し，不具合の検出漏れなどについての考察を行うことが挙げられる．

今後の展望としては，本手法を自動化し，ツール化することがある．これにより，DSML の解析が容易になり，デッドコードの検出やテンプレートの構造について考慮しやすくなると考えられる．今後，先に挙げた課題について検討しつつ，自動化に向けてさらなる手法の改善を進める．

参 考 文 献

1) Cook, S., Jones, G., Kent, S. and Cameron, A.: *Domain-Specific Development with Visual Studio DSL Tools*, Addison-Wesley Professional (2007).
2) 一野浩太郎，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).
3) 森奈美子，久住憲嗣，中西恒夫， 福田晃：ドメイン特化型開発におけるテストケース自動生成手法を用いたテストプロセスの提案，情報処理学会ソフトウェアエンジニアリングシンポジウム 2010，pp.113–118 (2010).
4) Reisner, E., Song, C., Ma, K.-K., Foster, J.S. and Porter, A.: Using Symbolic Evaluation to Understand Behavior in Configurable Software Systems, *Proc. of the 32nd ACM/IEEE International Conference on Software Engineering*, Vol.1, pp.445–454 (2010).
5) 吉澤正孝，秋山浩一，仙石太郎：ソフトウェアテスト HAYST 法入門，日科技連 (2007).
6) Kuhn, D.R., Wallace, D.R. and Jr., A. M.G.: Software fault interactions and implications for software testing, *IEEE Transactions on Software Engineering*, Vol.30, pp.418–421 (2004).