要件定義プロセスと保守プロセスにおけるモデル検査技術の開発現場への適用





松浦　佐江子[†1]　小形　真平[†2]　青木　善貴[†3]　谷沢　智史[†4]　西村　一彦[†4]

システム構築の上流工程における仕様の実現可能性や，保守工程における仕様及び仕様の誤解に起因する不具合現象に対し，形式検証技術であるモデル検査技術を一般的な開発者がそれぞれの工程において有効利用可能なシナリオを想定し，その支援方法を提案する．

Effective Model Checking Techniques Usage for Requirements Analysis and Maintenance Process

Saeko Matsuura[†1], Shinpei Ogata[†2], Yoshitaka Aoki[†3], Satoshi Yazawa[†4], Kazuhiko Nishimura[†4]

Abstract:　To verify the feasibility of the specification in the requirements analysis process of a system development and detect the defects caused by misunderstanding of the specification in the maintenance process, we propose an effective usage of model checking technique for non-specialized developers based on several reasonable development scenarios.

1.　はじめに

近年，システムの利用シナリオの多様性と広がりに加えて，ハードウェアやアーキテクチャの多様性が増加しており，手戻りを防ぎ，高品質なソフトウェアを開発するために，開発工程における検証プロセスの重要性が増している．モデル検査技術は，システムの振舞いを状態遷移システムとみなし，システムの満たすべき性質を，状態空間の探索により検証する技術である．テストでは実現できない網羅的検査に特徴があり，システム構築の上流工程において，その仕様の妥当性を検証するための形式検証技術として注目を集めている．しかし，実際に開発現場で用いるためには，開発現場での適用シナリオを想定して，検査対象システムのモデルとその検証したい性質を現場の開発者が容易かつ適切に定義できるようにすることが必要である．本研究では，要件定義プロセス，運用時の保守プロセスといった開発現場でのモデル検査技術利用のシナリオを想定し，それぞれの場面で，現場の技術者が利用可能な検証方法とその支援ツールを研究開発した．

2.　モデル検査技術適用の課題

モデル検査技術をシステムの振舞いの検証に適用するためには，一般に，振舞いを正確かつ少ない状態数と遷

【脚注】
† 1　芝浦工業大学　Shibaura Institute of Technology
† 2　信州大学　Shinsyu University
† 3　日本ユニシス株式会社　Nihon Unisys, Ltd.
† 4　（株）ボイスリサーチ　Voice Research Inc.

移数でモデル化し，そのモデルに対して検査したい性質を検査式として定義しなければならない．

VDM [VDM] や B-method [B-Method] のような形式仕様化技術は設計工程において，ドキュメントを形式化する有望な技術ではあるが，一般には，あいまいな要求から，要求抽出，要求定義を行う混沌とした要求分析工程において，はじめから厳密な形式手法を用いることは困難である．一方，UML（Unified Modeling Language）[UML] は要求仕様を記述するための自然言語を含む柔軟な記述を許す道具として広く多くの開発者に使用されている．クラス定義のように識別子を構造的に定義することは可能であるが，例えばアクティビティ図においても，アクションの記述や分岐の条件を表すガードの記述の自由度は高く，そのままでは形式仕様のように機械的な検証はできないという問題がある．

一方，ソースコードに対する不具合や仕様を満たしているかを検証する際には，大規模で複雑なソースコードをモデル化しても，状態爆発により検査ができないことがある．更に，検査したい性質の定義は一般にはソースコードの識別子と対応付けなければならず，ソースコードを仕様と対応付けて読み解くことは困難な作業であるという問題がある．

このように，開発工程に検証プロセスを導入するためには，ソフトウェア開発の上流工程では，初めから厳密な振舞いモデルを定義することが，下流工程では，ソースコードとして定義された振舞いと，その満たすべき性質である仕様を結びつけて，検証したい性質を定義することが難しい．

本研究では，我々がこれまで研究開発してきたUML を用いたモデル駆動要求分析手法 [Ogata2010a, Ogata2010b] 並びに，モデル検査ツールの 1 つである UPPAAL[UPPAAL] を用いたソースコードの欠陥抽出手法 [Aoki2011a,Aoki2011b] に基づき，ソフトウェア開発の上流と下流の両面から，開発者が想定した振舞い定義モデル（要求仕様書やソースコード）を特定の性質を満たす抽象的なシステムの振舞いモデルとして捉え，その特定の性質が取り得る状態の遷移モデルと結び付けることで，検査したい性質を表す論理式を自動生成する方法を研究する．

3. 開発現場での検査へのアプローチ

3.1. 業務セオリー

ソフトウェア開発工程には要求定義・設計・実装・テスト・運用の工程がある．これらの工程において，作りたいシステムを利用する際の作業手順，システムを利用してできることの条件，期待される状態，あってはならない状態，期待される効果，といった様々な形で，システムに対する要求が存在する．要件定義段階では，これらの全てを考慮するわけではないが，これらのレベルの異なる性質は最終的にはすべてのソースコードが満たさなければならない性質であり，違反することがあってはならない．そこで，本研究では，ソフトウェア開発工程で考慮される，ソフトウェアの満たすべき性質を「業務セオリー」と呼び，様々なシステムの性質の検証を開発現場で行えるように，整理することを目指す．例えば，個々のアプリケーション依存のセオリーだけでなく，ドメイン依存のセオリー，セキュリティ要件のセオリー，ソフトウェア構造依存のセオリー，ドメイン依存のフレームワークによるセオリー，ハードウェアアーキテクチャの性能制約によるセオリー，無限ループやデッドロック等プログラムで生じてはならない一般的なセオリー等の観点から検査したい性質の容易な定義方法を研究する．

3.2. 利用シナリオ

開発現場で検証技術を利用するためには，どのような場面であるならば，開発者が検査の実効性を享受できるかを考える必要がある．検証技術の利用が有効な場面を「シナリオ」と呼び．シナリオを実現する検査方法とその支援ツールを研究する．

要求をシステムの要件として定義する段階では，試行錯誤的に要求を確認しながらモデリングしなければならない．この段階で要求の妥当性を確認するには，我々のモデル駆動要求分析手法で行うように，要件定義から生成される最終形態を模したプロトタイプにより，利用者の観点からシステムの振舞いを確認することが有効である．しかし，要件定義は 1 つ 1 つのユースケースという機能的要件を部分的な観点から定義したものであり，入力から期待される出力のデータは本当に生成できるのか，ユースケースをつなげたトータルなサービスは，必要なデータを供給しながら，うまく実現できるのかといった要件の実現可能性や，非機能要件に関する妥当性や整合性を検査することは，人手では確認に手間がかかり，確認漏れや誤解が生じやすい．そこで，モデル検査を利用する第一の利用シナリオは「要件定義プロセスにおける要件定義の不整合の早期発見」とする．

テスト開発者は業務の知識を用いてテストを行なうことができるが，すべてのケースを網羅的にテストすることはできないため，実装者がその業務について十分理解

していない場合に，仕様の誤解によりテストでは発見できなかった不具合が運用時に生じることがある．このような場合，不具合現象は分かるが，ユーザからは不具合原因を特定する情報をあまり得ることができずに，検査者は経験に頼って，不具合の発見と修正を行わなければならないことが多い．こうした不具合は，特定するために，しらみつぶし的なテストを繰り返さなければならず，再現を保証することも経験の少ない検査者には困難である．こうした「運用時の想定外の使用による不具合の特定」を第二の利用シナリオとする．

3.3. 検査方式

業務セオリーとモデル検査による基本的な検査手順は図 1 に示す通りである．検査対象は各フェーズにおけるシステムの振舞いを定義したドキュメントである．上流工程のドキュメントに対し，開発工程が進むにつれ，システムの満たすべき性質である業務セオリーは開発者により具体化され，段階的にシステムの振舞いモデルに導入されていると考えられる．そこで，第一のシナリオでは，この導入作業において，検査対象の振舞いモデルと検査したい性質である業務セオリーの接点に着目する．一方，第二のシナリオでは，導入されているはずの業務セオリーを検査者が接点を確認しながら検査したい性質を定義する．この作業により，モデル検査の入力となるシステムモデル（有限オートマトン）と検査式を生成し，振舞いモデルが，検査したい性質である業務セオリーに違反することがないことをモデル検査ツールを用いて網羅的に検査する．振舞いモデルをシステムモデルに自動変換することで，反例の得られた個所を振舞いモデル上で特定できることから，これを修正する．

モデル検査を開発現場で用いるためには，モデル検査ツールを直接操作しないことが一つの解決策となる．「検査したい性質」を「検査対象の振舞いモデル」と対応付けて，開発者が理解しやすい形式で定義できるかということを解決しなければならない．また，モデル検査における状態爆発を回避できるように振舞いモデルの自動変換にも工夫が必要である．

第一の利用シナリオでは，モデル駆動要求分析手法による成果物である UML モデルを検査対象とする．2 節で述べた通り，UML は VDM 等の形式仕様記述言語と比べて，あいまい性はあるが，一般の開発者にも受け入れやすいという利点があり，開発現場への適用には向いていると考えられる．

本稿では，つぎの業務セオリーに着目し，データの基本的な性質に違反しないことを検査する．

- 要件定義の実現可能性の観点から，入力から出力を保証するエンティティ・データは，すべてのユースケースを継続して，複数のユーザに同等のサービスを提供できるように，その基本的な性質であるデータの CRUD（生成・参照・更新・削除）に関する振舞いに矛盾しない．例えば，まだどこでも生成されていないデータは参照・更新することはできない．このような性質は入力から出力を得る処理フローを分析している段階では，厳密に定義することが可能になっていると考えられる．

第二の利用シナリオでは Java のソースコードを検査対象とする．ソースコードはすべての要件が定義されているが，その表現が多様であり，アプリケーションとして満たすべき性質，使用しているハードウェアの制約，使用しているソフトウェアアーキテクチャの性質，言語特有の性質，プログラム一般の性質等，満たすべき性質，起きてはいけない現象は分かっていても，それをソースコードの識別子と対応付けることは困難である．本稿ではプログラム一般の性質として「停止しない」という不具合現象を業務セオリーとして，モデル化する．「停止しない」という現象は，個々のアプリケーションに依存しない振舞いであり，一般的にモデル化が可能である．

本稿では，それぞれの工程でのデータの振舞いの制約と現象の振舞いをモデル化し，二つのモデルの接点を段階的に定義・設定することで，接合したシステム（有限オートマトン）モデルを生成し，モデル検査によって，起こってはならない状態に到達しないことを検査するプロセスを開発現場でも利用できるように支援する．このために，モデル検査ツール UPPAAL を用いて，要件定義段階の支援ツール UML2UPPAAL と保守段階の支援ツール Source2UPPAAL を開発した．

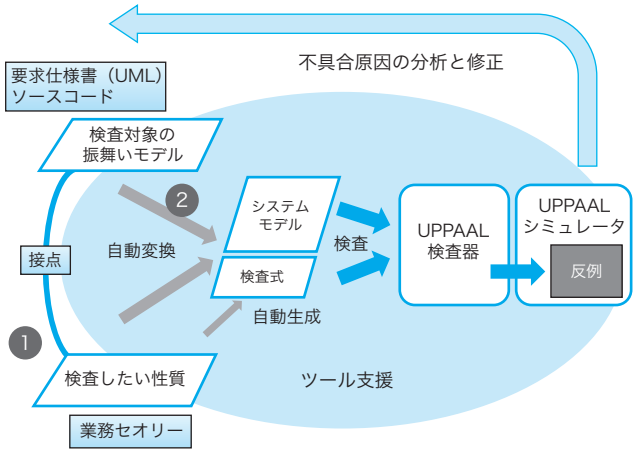

図 1　業務セオリーと検査方式

4. 要件定義プロセスにおける検査

4.1. UML2UPPAAL による検査プロセス

図 2 は，UML2UPPAAL を用いた要件定義プロセスにおける検査のプロセスを示している [Ogata2012,Aoki2012a,Aoki2012b]．要求分析モデルは，ユースケース分析に基づき，システムの提供するユースケースの振舞いを UML のアクティビティ図で，システムの利用するデータを UML のクラス図で定義したものであり，ユースケースの呼び出し関係もアクティビティ図で定義し，システム全体の振舞いを規定している．これを Navigation モデルと呼ぶ．アクティビティ図には振舞いの対象となるデータがオブジェクトノードとして定義されており，業務セオリーは，このデータの満たすべき性質をクラス毎に UML のステートマシン図で定義したものである．要求分析モデルは，各ノードと遷移に対応した UPPAAL モデル（有限オートマトン）に変換される．ステートマシン図は，すべてのアクティビティ図内で同一視されたオブジェクトノード毎に，状態と遷移に対応した UPPAAL モデルに変換される．このとき，アクティビティ図のアクションとステートマシン図のイベント，及びアクティビティ図のガードとステートマシン図のステートがこれらのモデル間の接点となり，UPPAAL のチャネルに変換され，モデル間の同期が定義される．検査式はステートマシン図において到達し得ない状態を解釈することにより，自動的に生成される．これらのモデルを UPPAAL モデル検査ツール上で実行した結果から検査式を満たしていないフローを UML 要求分析モデル上にフィードバックし，このフローと検査結果から要求分析モデルの問題点を発見して，モデルの修正を行う．

UML2UPPAAL は UML モデルの UPPAAL モデルへの変換，検査式の生成，検査の実行，反例のモデルへの反映の機能を持ち，UML モデリングツール astah*[astah] のプラグインとして実装されている．

4.2. CRUD に関する業務セオリー

要求分析モデルでは，登場するオブジェクトノードで表されるデータに対して，様々な処理を行って，期待される出力への変換手続きが定義される．ここで，処理が適用可能な最低条件は，そのデータが存在すること，すなわち，オブジェクトが「値あり」の状態になっていることである．存在しないデータに対しては何ら処理を行うことはできないため，対処方法を定めておかなければならない．そして，一般に，オブジェクトが「値なし」の状態から「値あり」の状態になるのは，そのオブジェクトの生成や取得の振舞いによってである．また，「値あり」のオブジェクトに対しては参照や更新，削除を行うことが可能である．このようにオブジェクトが適切に処理されるためには，その基本的なライフサイクルの性質である CRUD（Create・Read・Update・Delete）に関する普遍的な性質を満たすことが必要である．そこで，要求分析モデルのアクティビティ図が表す「入力データから出力データを生成する振る舞い系列」において，各アクションを CRUD 機能とその対象データによって整理することにより，開発者が定義した「実現するに足るシステム内部データ」が，要求分析モデルのすべての経路において，そのデータライフサイクルの性質を満たすことを検査することができる．

まず，アクションに記述する動詞を CRUD 機能と対応付けて定義する．これらの動詞は，アクティビティ図のシステムのパーティションにおいて，開発者が慣習的に使用する動詞を参考に決定した．例えばアクションの動詞が「取得する」場合，その行為は Read であると認識する．アクションノードに記述される「動詞」とその振

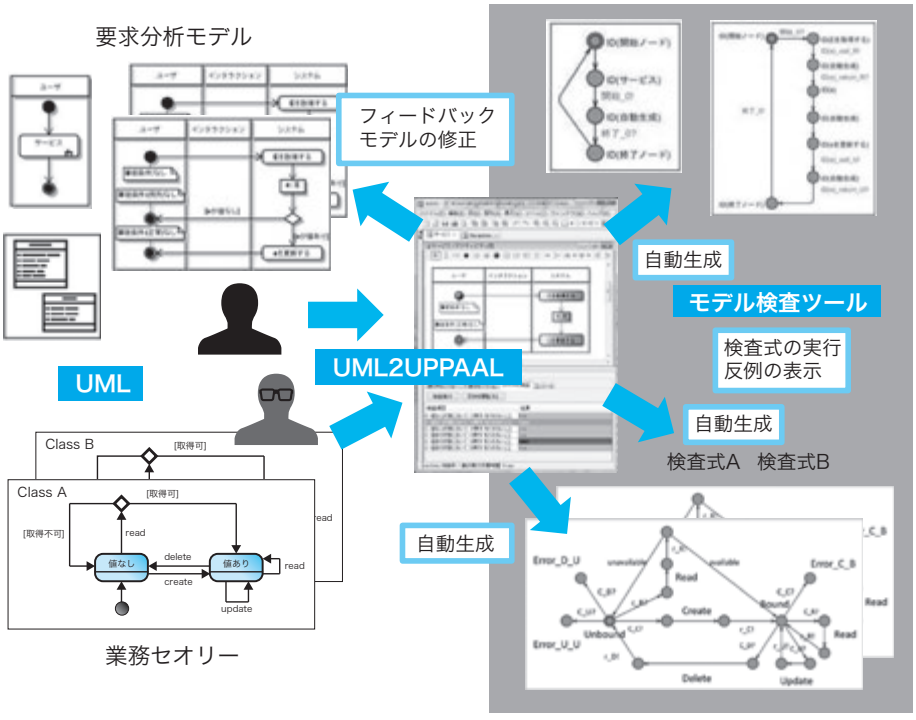

図 2　要件定義プロセスにおける検査プロセス

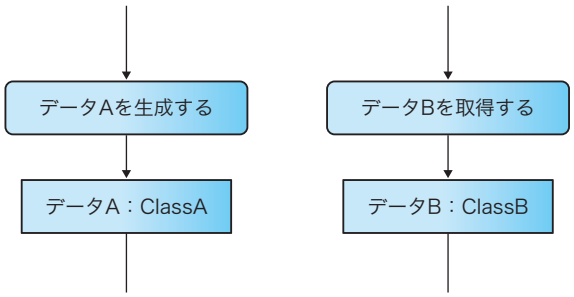

図３　Create・Read アクションの動詞とオブジェクトノードの関係

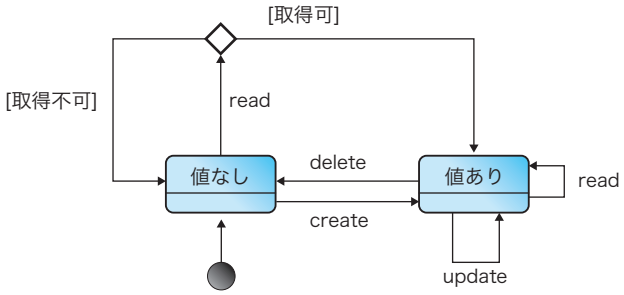

図４　クラスの CRUD に関する業務セオリー

る舞い対象である「目的語」と，それに対応する「オブジェクトノード」の位置から読み取れる意図を解釈する事で，振舞いにおける CRUD 機能の呼び出しが，図３のようにアクティビティ図に定義される．

これらの動詞により，アクティビティ図上のオブジェクトノードが「値なし」状態と，「値あり」の状態において，上記の CRUD の振舞いによって遷移可能なモデルを図４のようにステートマシン図を用いて定義する．これが CRUD に関する業務セオリーである．図４は一般的なライフサイクルであり，更新や削除を許可しないデータの場合には，これらの遷移を削除する．

ここで，重要なことは要求分析モデル内の定義要素と満たすべき性質を表すモデルの定義要素が，この段階において厳密に定義できる事柄に基づいて，対応づいたことである．これにより，要求分析モデルにおける性質を検査することが可能になる．

ステートマシン図は CRUD アクションに関するクラスのデータライフサイクルを定義しており，そのクラスに属するすべてのインスタンスの振舞いを限定するものである．言いかえると，この定義は「想定できない悪いことは決して起こってはならない」というモデル検査で検査できる性質の１つである「安全性」を示している．図４のステートマシン図は「インスタンスが値なしの状態であれば，そのインスタンスに対する Update 及び Delete の操作は決して起こってはならない」ことを定義しているわけである．そこで，この決して起こってはならない状態を UPPAAL モデル内に明示的にロケーションとして定義し，このロケーションを用いて「安全性」に関する検査式を自動的に生成する．ここで，ロケーションとは有限状態モデルである UPPAAL モデルの状態を表す要素の名称である．

4.3. 事例による検証

本手法の有効性を確認するために，適用実験を行った．本学で運用されているシステムや演習で作成されたシステム等４つのシステムの UML 要求分析モデルを５人の大学院生が，CRUD アクションの記述ルールを用いて整理した．次に，被験者はステートマシン図を用いて，要求分析モデルに登場するエンティティクラスのデータライフサイクルを各クラスの業務セオリーとして定義した．ここでは，図４に示す基本的な CRUD モデルを参考に，データ毎にカスタマイズしてモデルを作成した．被験者は UPPAAL の知識をほとんど持っていないが，UML2UPPAAL を用いて定義ミスの発見も含めて，83 個の問題点を発見することができた．適用実験で発見された主な問題点は次の通りである．

- 非決定的な Read アクションに対して，アクティビティ図内に正しくガードの条件を設定できていないケースが 10 個発見できた．これは，そのデータの想定される性質が不適切であるか，アクティビティ図のフローにおいて，データが取得できないケースを見逃していたことに起因する．
- Update と Delete 操作が「値なし」のオブジェクトに適用されていることを２個所で発見した．これは，振舞いモデルにおける「値なし」のケースの処理についての検討不足であった．
- Create 操作を定義する場所を間違えていたために．このサービスの間にはあるオブジェクトが全く生成できないという問題が１つ発見できた．このような問題が生じた１つの要因は，ユースケースの記述が複雑になり，目視のレビューでは発見が困難であったことであると考えられる．

事例による実験の結果，要求分析モデルを UPPAAL モデルに変換した結果，状態数が増加し，モデル検査が終了しないケースがあった．変換した UPPAAL モデルの状態数を抑えるため，モデル定義に工夫が必要であることが分かった．要求分析モデルでは，複数のユースケースを Navigation モデルにより統合する．この統合において，繰り返し処理を行うと，呼び出される実行モデル

が大きい場合，ロケーションへの到達可能性の検査において，out of memory になる可能性がある．そこで，Navigation モデルでは繰り返しが起こらないようにモデルを定義することで状態数の削減を行うことができた．CRUD の振舞いの検査を行うには，各ユースケースのつながりが分かれば良いので，検査上は問題ないが，要求仕様として考えた際には，自然な記述ではなくなる可能性があり，要求分析モデルの定義と検査の役割について検討が必要である．

5. 保守プロセスにおける検査

5.1. Source2UPPAAL による検査プロセス

図 5 は，Source2UPPAAL を用いた保守プロセスにおける検査のプロセスを示している．[Yazawa2012, Aoki2012c]

ソースコードは Source2UPPAAL によりソースコードモデルへ変換される．また仕様及び不具合の現象から検査者は独自のモデルを定義する．ソースコードモデルと独自モデルは Source2UPPAAL 上で，検査者の指示により結合されてシステムモデルとなり，生成された検査式により，モデル検査が実行される．開発者は UPPAAL の反例からソースコードの不具合原因を分析する．

ソースコードは AST パーサーにより抽象構文木（AST）に変換される．Source2UPPAAL が抽象構文木に基づきモデル化候補となるメソッドを表示するので，変換対象のメソッドをドラッグ＆ドロップ操作で選択することで，モデルの状態数を制限して検査する．選択されたメソッド定義において，ステートメントは UPPAAL モデルのロケーションに，整数と真偽値の変数は UPPAAL の変数に，変数への代入は UPPAAL の更新式に変換される．メソッドや API の呼び出しに対しては，図 6 に示すように，検査者が必要に応じて非決定性を与える値を提示されるパターンの候補から割り当てる．また，検査者が UPPAAL モデルとして定義した独自モデルをツールに登録することでメソッドや式にその振舞いを割り当てることができる．Source2UPPAAL はこのようなソースコードのモデル化個所の指定や，非決定性や独自モデルの割り当てのためのドラッグ＆ドロップ方式のユーザインターフェースを提供する．そして，検査者の指定が終了すると，UPPAAL モデルと検査式を自動生成し，検査を実行する．

5.2. 事例による検証

下記の事例を用いて，Source2UPPAAL により，モデル検査で検査できる性質の 1 つである「到達可能性」と「停止しない」という，起こってはならない不具合現象の検査を行い，その原因を発見することができた．

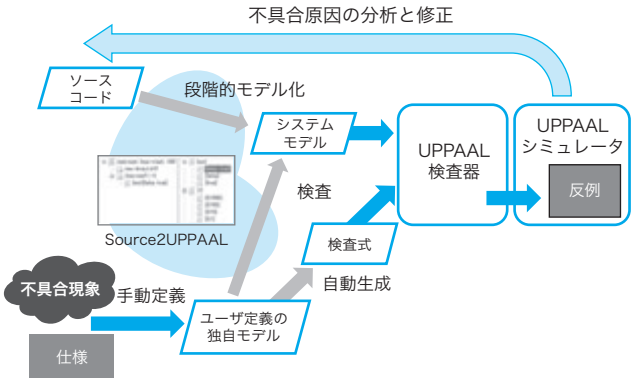

図 5　保守プロセスにおける検査プロセス

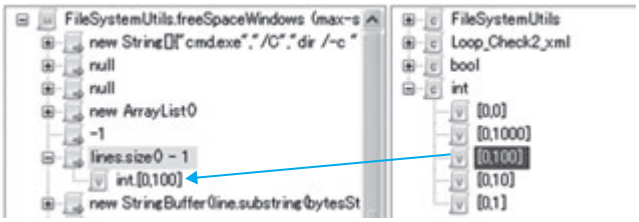

図 6　非決定性の値の割り当て

1）規定の走行体を用いて，難所を含むコースのライントレース自律走行を競う ET ロボコンにて使用したロボットの走行制御プログラムにおける制御フローの正当性の確認

2）オープンソース Apache Commons の既知のバグ（無限ループ）

3）会計伝票発行における停止しない現象（無限ループ）

1）の事例では，制御フローの正当性を確認するために，すべての制御フローがすべて通りえる状態であるかを到達可能性により検査する．検査式は，生成されたすべてのロケーションへの到達の検査であり，「ロケーション ** にいつかは到達する」という意味の検査式を自動的に生成する．検査結果が「満たされない」の場合は，そのロケーションへの到達はできないことになり，その反例がトレースファイルとして記録される．これを分析した結果，到達できないソースコードへの分岐条件式に問題があることがわかった．その条件式に登場する変数の変化をトレースファイルから確認し，その値を計算している部分を特定して，修正を行い，再度検査をして，すべての検査式が満たされることを確認した．

2）及び 3）の事例では，「停止しない」という観測できる状態に対して，無限ループのモデルを独自モデルとして定義し，制御の条件式に対して設定することで，無

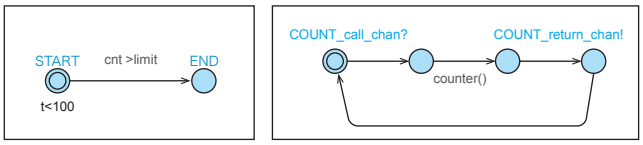

図 7　無限ループ判定用モデル

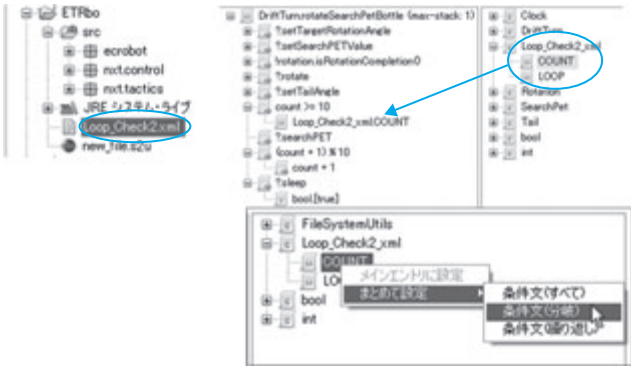

図 8　独自モデルの割り当て

限ループの発生に起因するこの現象を検査する．検査式は無限ループ状態のロケーションへの到達の検査であり，これも自動的に生成することができる．

この検査は，制御構造の未到達を検査するのではなく，既知の不具合の現象をモデル化して，それにより不具合の原因の特定を行うものである．ここでの「不具合の現象」は無限ループであるため，無限ループ判定用のモデルを定義する．これはすべてのプログラムが犯してはならない 1 つの業務セオリーである． 無限ループの判定モデルは図 7（左）のように定義し，閾値（limit）を決めておき，それを超えた場合に無限ループと判定する．無限ループは条件分岐で抜けられない可能性が高いため，条件分岐している部分にループをカウントするための関数 counter() を配置する．counter() を呼び出すモデルは図 7（右）のように定義する．ここでは，このモデルは UPPAAL を直接使用して作成している．

定義した無限ループ判定用のモデルをつぎのようにツールに追加する．まず，ツール上で検査対象と同じプロジェクト内に追加モデルのファイルを配置すると，図 8 のように表示される．ここでの追加モデルのファイルは図 8 の四角形で囲われた Loop_Check.xml である．ここでは，無限ループの発生を検査するため，条件文の式に対して，まとめて独自モデルを割り当てている．検査式は「図 7（左）のモデルのロケーション END に到達することは決してない」という安全性の検査式が自動生成される．検査式が満たされなかった場合，反例が示され，遷移の過程がトレースファイルに記録される．このトレースファイルを用いて無限ループの発生状態の分析を行う．全ロケーションの出現頻度をグラフで表すことで，その頻度が高い個所から，ループしているロケーションが特定でき，これにより，対応するループしているソースコードが特定できる．その部分を含むループ構造の条件式または break によるループの脱出に至る条件式のどちらかに問題があるので，これらに登場する変数の変化をトレースファイルから確認し，修正を行い，再度検査をして，すべての検査式が満たされることを確認した．

現実のソースコードは複雑であり，これをすべて UPPAAL モデルに変換しても，現実的な検査は行えない．本研究でのアプローチの特徴は，アプリケーションコードに依存しない，不具合現象をモデル化し，開発者が，関連するソースコードのメソッドを段階的に，非決定性を持つ値を想定しながら，検査できる機能を持つ支援ツールを提供していることである．到達可能性と無限ループ以外にも，ライントレースロボットのコース逸脱現象やデータベースロックの不具合といった不具合現象を検出できることが分かっているが，例えば前者ではロボットの走行環境であるコースのモデル化が，後者では，システムが利用している特定言語のモジュール拡張機能の仕組みを独自にモデル化する必要がある．

6.　関連研究

モデル検査ツールの効果的な利用方法に関する研究がある [Tracka2009, Bose1999, Jing2009]．Trcka[Tracka 2009] はペトリネットを利用して，read, write, delete といった振舞いを特定する予め用意した 9 つの検査式を用いて検査を行う方法を提案している．この研究も我々のアプローチと類似しているが，我々の方法ではステートマシン図を用いて，データの性質の着目点を限定して容易に定義することのみによって検査式を自動生成できるため，固定した CRUD だけのルールではなく，他の性質への拡張が可能である．[Bose1999, Jing2009] の研究では，UML モデルを PROMELA へ変換し，SPIN[SPIN] を利用して，モデル検査を行う方法を提案している．しかし，検査を行うためには，一般の開発者が直接モデル検査ツールを操作する必要があり，開発者は UML と SPIN の両方の知識を要求される．UML2UPPAAL は UML の知識だけで検査を実行することができる．

ソースコードの不具合検出にモデル検査を使用する研究は多い．Java Pathfinder[Pathfinder] は実行可能な Java バイトコードよりプログラムを検証してデッドロックとアサーションエラーを検出する．Pathfinder は複雑

で大規模なソースコードに対しては「状態爆発」の注意が特に必要である．

Bandera[Corbett2000] は．抽象化とスライシングにより，Java プログラムのソースコードから，コンパクトな有限状態モデルの自動抽出が可能である．出力モデルは，SPIN や SMV[McMillan1993] 等の既存のモデル検査ツールで検証することができる．このアプローチは，「状態爆発」の抑制には有効であるが，検査者にはモデル検査の深い知識が必要となる．

7. まとめと今後の課題

モデル検査を実施する上で，第一に克服しなければならない課題は，検査対象の定義と検査したい性質の定義をその構成要素間で容易に対応付けることができることである．本研究では，要求仕様とソースコードに着目し，段階的に検査対象と検査したい性質としての業務セオリーを結び付けることにより，要求定義段階では，モデル検査技術の知識がなくても検査が可能な UML2UPPAAL を実現した．保守段階でも，少ないモデル検査技術の知識で，検査を支援する Source2UPPAAL を開発し，開発現場での検証技術の利用が有効な場面であるシナリオを実現する検査方法と支援ツールを提案した．

要件定義プロセスにおける要件定義の不整合の早期発見に関しては，UML2UPPAAL として，開発者がモデル検査技術の知識を持たなくても，網羅的な検査を実施できることがわかった．しかし，コレクションとその要素に関する振舞い，オブジェクトとその属性であるオブジェクトの連動等の定義方法は検討中である．要求分析モデルを段階的に形式化することの見通しはあるが，定義方式が複雑にならないよう，検討する必要がある．

今後は，本手法を情報システムのセキュリティ保証要件を定めた国際標準（ISO/IEC 15408）である Common Criteria [Common Criteria] に基づき，要求仕様がセキュリティ機能要件を満たすことを検査すること [Noro2013] に応用する．

運用時の想定外の使用による不具合の特定に関しては，無限ループ以外の不具合現象例のモデル化を要件定義プロセスと同様に仕様をステートマシン図で定義し，検査式を自動生成することを検討する．また，検査によって出力される反例の解析により，修正を支援する必要もある．成功事例と失敗事例の対比により，不具合を特定する方法を検討する．

謝辞

本研究は，独立行政法人情報処理推進機構技術本部ソフトウェア高信頼化センター（SEC: Software Reliability Enhancement Center）が実施した「2012 年度ソフトウェア工学分野の先導的研究支援事業」の支援を受けたものである．

【参考文献】

[VDM] VDMTools, http://www.vdmtools.jp/

[B-Method] K. Lano, H. Haughton: Specification in B: An Introduction Using the B Toolkit, Imperial College Press, 1996.

[UML] OMG Unified Modeling Language, OMG, http://www.uml.org/

[Ogata2010a] Shinpei Ogata, Saeko Matsuura, Evaluation of a Use-Case-Driven Requirements Analysis Tool Employing Web UI Prototype Generation, WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS, Issue 2, Volume 7, pp.273-282, February 2010.

[Ogata2010b] 小形，松浦：UML 要求分析モデルからの段階的な Web UI プロトタイプ自動生成手法，日本ソフトウェア科学会，コンピュータソフトウェア，Vol.27，No.2，pp.14-32，2010.

[UPPAAL] UPPAAL, http://www.uppaal.com/

[Aoki2011a] Y. Aoki and S. Matsuura, Verification of Embedded System by a Method for Detecting Defects in Source Codes Using Model Checking , IEEE Symposium on Computers & Informatics, pp. 530-535, 2011.

[Aoki2011b] Y. Aoki and S. Matsuura, A Method for Detecting Defects in Source Codes Using Model Checking Techniques, Proc of the 34th Annual IEEE International Computer Software and Applications Conference, pp.543-544, 2010.

[astah] astah*, http://www.change-vision.com/

[Tracka2009] N. Trcka, et al., “Data-Flow Anti-Patterns: Discovering Dataflow Errors in Workflows,” Proc. of the CAiSE 2009, pp.425-439 Amsterdam, The Netherlands, 2009.

[Bose1999] P. Bose, “Automated translation of UML models of architectures for verification and simulation using SPIN,” Proc. of the ASE, pp.102-109, Fairfax, VA, 1999.

[Jing2009] L. Jing, et al. “Model Checking UML Activity Diagrams with SPIN,” Proc. of the CiSE 2009, pp.1-4, Qingdao, China, 2009.

[Pathfinder] Pathfinder, http://javapathfinder.sourceforge.net/, 2013

[Corbett2000] Corbett, J., Dwyer, M., Hatcliff, J., Laubach, S., Pasareanu, C., Robby, and Zheng, H., "Bandera: extracting Finite-state models from Java source code," Proc. the 22nd Int'l Conf. on Softw. Eng. (ICSE 2000), pp. 439–448, 2000

[SPIN] SPIN, http://spinroot.com/spin/whatispin.html, 2013

[Ogata2012] S. Ogata, Y. Aoki, H. Okuda and S. Matsuura, An Automation of Check Focusing on CRUD for Requirements Analysis Model in UML, Proc of ICSCE 2012, pp.1095-1103, 2012.

[Aoki2012a] Y. Aoki, S. Ogata, H. Okuda and S. Matsuura, Quality Improvement of Requirements Specification Using Model Checking Technique, Proc of ICEIS 2012, Vol.2,pp401-406, 2012.

[Aoki2012b] 青木，小形，奥田，松浦，要求分析における CRUD 観点のモデル検査技術の適用，ソフトウェア工学の基礎 XIX，日本ソフトウェア科学会 FOSE 2012，pp. 75-80.

[Yazawa2012] 谷沢，西村，青木，小形，松浦，Source2UPPAAL: ソースコードの効率的な検証へ向けた開発者支援ツールの検討，ソフトウェア工学の基礎 XIX，日本ソフトウェア科学会 FOSE 2012，pp. 241-242，2012.

[Aoki2012c] 青木，松浦，開発現場を想定したモデル検査に基づくプログラムの不具合検証，電子情報通信学会 , 信学技報 , vol.112,no.496, KBSE2012-69, pp. 1-6, 2013.

[McMillan1993]K. L. McMillan: Symbolic Model Checking. Kluwer Academic Publishers, 1993

[Common Criteria] Common Criteria, CC/CEM v3.1Release4, http://www.commoncriteriaportal.org/cc/

[Noro2013] A. Noro and S. Matsuura, UML based Security Function Policy Verification Method for Requirements Specification, Proc of 2013 IEEE 37th International Conference on Computer Software and Applications, pp.832-833,2013.