独立行政法人情報処理推進機構 委託

2012 年度ソフトウェア工学分野の先導的研究支援事業
「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に関する研究」
成果報告書

平成 25 年 1 月

学校法人芝浦工業大学

本報告書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・エンジニアリング・センターが実施した「2012 年度ソフトウェア工学分野の先導的研究支援事業」の公募による採択を受け芝浦工業大学デザイン工学部（研究責任者　松浦佐江子）が実施した研究の成果をとりまとめたものである.

目次

研究成果概要

１．背景

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，手戻りを防ぎ，高品質なソフトウェアを開発することがより強く求められており，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．モデル検査技術はテストでは実現できない網羅的検査に特徴があるが，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の検査式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大きな課題である．

２．研究目標

本研究では，要件定義プロセス，運用時およびマイグレーションによる再構築を含む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する．本研究の研究課題としては，芝浦工業大学松浦研究室でこれまで研究開発してきたUML(Unified Modeling Language)を用いたモデル駆動要求分析手法ならびに，モデル検査ツールの１つであるUPPAALを用いたソースコードの欠陥抽出手法に基づき,下記の研究課題に取り組む.

1) 新規開発システムにおける要件定義で得られた仕様を客観的に評価する業務セオリーとその評価手法
2) 想定外の使い方等による不具合を既存のソースコードから発見する業務セオリーとその評価方法
3) 仕様書を喪失した稼働中のシステムの業務セオリーを抽出し，評価する手法

モデル検査手法をシステムの振舞いの検証に適用するためには，一般に振舞いをどのように正確かつ少ない状態数と遷移数でモデル化するかが難しい課題である．本研究の着眼点は，開発者が想定した振舞い定義モデルを特定の性質を満たす抽象的なモデルとして捉え，その特定の性質が取り得る状態とその遷移モデルと結び付けることで，検査項目となる論理式を自動生成することである．

３．開発現場での利用シナリオと業務セオリー

ソフトウェア開発工程は大きく分けると要求定義・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，つく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．要求定義段階では，これらの全てを考慮するわけではないが，最終的なソースコードは本来全ての要求を満たさなければならない，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

トウェア開発の全工程で考慮される，ソフトウェアの満たすべき性質を「業務セオリー」と呼び，これを整理することを目標とする．整理の目的は利用あるいは再利用しやすいようにある程度汎用的，あるいは定型的に定義できることを目指すという意味である．すなわち，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構築を目指す.

検査対象はそれぞれのフェーズのドキュメントであり，本研究では，それぞれのフェーズにおけるシステムの振舞いがそこに定義されているものとする．検査対象に対し，上述のような満たすべき性質を業務セオリーとして，汎用的な雛型から具体的アプリケーションに対して具体化し，検査対象と性質を規定する振舞いと状態の組とに対応付ける．これにより，検査対象の定義要素で検査したい性質が定義できることになり，モデル検査技術の難しさを解消する１つの方法となる．

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，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支援ツールを研究する．

要求をシステムの要件として捉え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，われわれ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プロトタイプにより，最終形態に近い形で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部分的な観点から定義したものであり，入力から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うまく実現できるのかといった要件の実現可能性の観点からの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．そこで第一のシナリオは「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，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，不具合現象はわかるが，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，技術者は経験に頼って，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技術者には困難である．こうした「運用時の想定外の使用による不具合の特定」を第二のシナリオとする．

既存のシステムを別の環境へマイグレーションする場合，通常のテストでは言語が違うため変換コードの正しさは，テストを実施した範囲でしか確認できない．旧システムの仕様を業務セオリーとして定義できれば，新旧システムを同じモデル検査用のモデルに変換することで，同じ仕様を満たすことを確認でき，効率的な検査ができると考えられる．このような「マイグレーションによるシステムの再構築時のシステムの仕様の保証」を第三のシナリオとする．

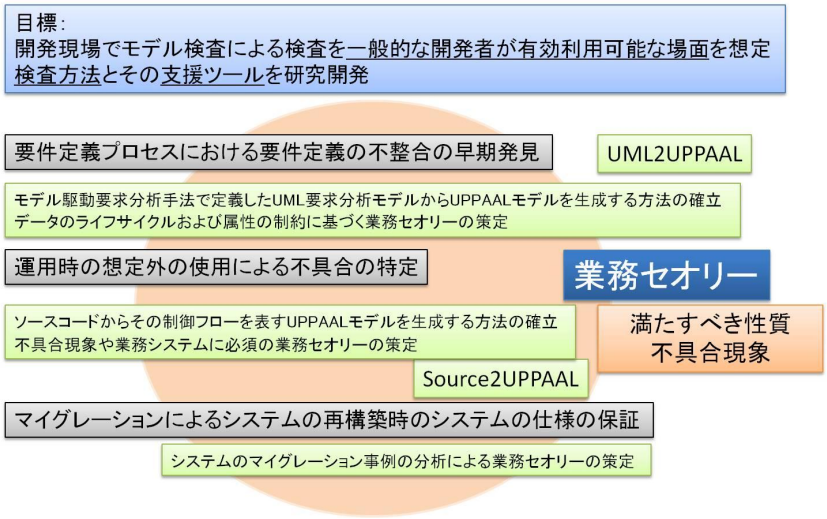


図１　目標の設定

３つの開発現場での利用シナリオに対し，図１のように目標を具現化する５つの研究目標を設定した．それぞれの要求定義段階の支援ツールを UML2UPPAL，保守段階の支援ツールを Source2UPPAAL と呼ぶ．

４．検査方式

モデル検査手法をシステムの振舞いの検証に適用するためには，一般に振舞いをどのように正確かつ少ない状態数と遷移数でモデル化するかが難しい課題であるといわれている．要求定義の段階では，一般にはドキュメントは自然言語で定義される．自然言語記述では，柔軟な記述が可能であるが，曖昧性があり，そのまま検証することはできない．われわれの研究しているモデル駆動要求分析手法の成果物は，UML のモデルを用いて，形式化している．UML は VDM 等の形式仕様記述言語と比べて，曖昧性はあるが，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．

しかし，要求定義の段階でも満たすべき性質に従って，要求を整理しなければ，アプリケーション自体の仕様における定義要素によって検査式を定義することは困難である．本研究では，つぎの２つに着目し，基本的な性質を満たすことや，要求を複雑化する非機能要求を満たしているかを判断できるように，要求を整理することを検討する．

- 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・データが，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がないことを保証する．
- 要求分析の複雑化の一因である非機能要件の内，セキュリティ要件を満たすことを保証する．セキュリティ要件は，データに対するセキュリティ属性とそれに対する振舞いにより定義が可能である．

一方，ソースコードは全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，プログラムの一般の性質等，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．

そこで，本研究ではシステムに対する満たすべき性質，起きてはいけない不具合現象を上流と下流から検討し，成果物である要求仕様（モデル駆動要求分析手法による UML モデル）と Java のソースコードを「検査対象」として，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としてモデル化し，２つのモデルの接点を段階的に定義することで，接合したモデル検査におけるモデルである有限オートマトン上で，矛盾や到達できない状態を検査することにする．

５．UML2UPPAAL

要件定義プロセスにおける検査のプロセスは図 2 に示す通りである．要求分析モデルと業務セオリーは，各ノードと遷移とアクティビティ図内のオブジェクトノードを基に，それぞれ UPPAAL モデルに変換される．この時，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，およ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 の同期関数で定義される．検査式もステートマシン図の解釈により，自動で生成できる．これらのモデルを UPPAAL 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表示し，このフローと検査結果から問題点を発見する．

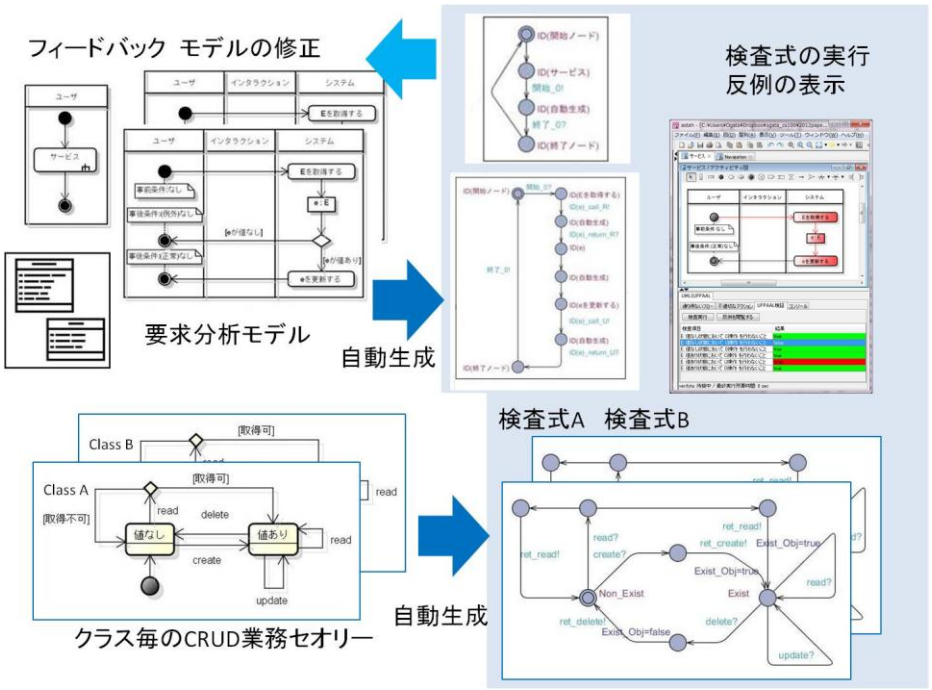


図 2　モデル検査手順

６．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，期待される出力へと変換する．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である．すなわち，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 に関する普遍的な性質を満たすことが必要である．サービスを全てのユーザに誤りなく提供するために，システム内部の永続化対象データはその基本機能である CRUD に関する普遍的な性質を満たすことを検査する．そこで，「入力データから出力データを生成する振る舞い系列」で定義する振舞いを CRUD 機能とその対象データを整理することにより，開発者が各アクティビティ図で定義した「実現するに足るシステム内部データの抽出と定義」のデータ・ライフサイクルが要求分析モデルのすべての経路において満たされることをモデル検査により検証する．

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，システム・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 機能の呼出しが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 3 のようにステートマシン図を用いて定義する．

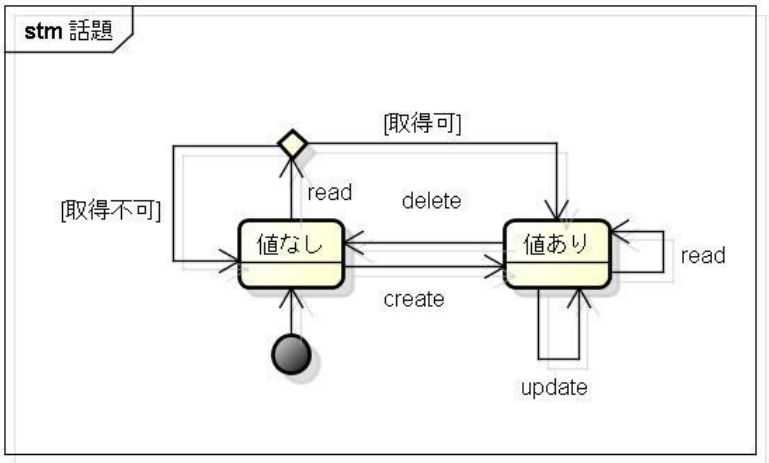


図 3　オブジェクトの CRUD に関する業務セオリー

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対応づくことである．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できる．

下記の 5 つの事例を用いて，モデル検査実験を実施し，その結果，表 1 に示す問題点に加え，計 83 個の問題点が発見できた．

① 大学院授業課題：大学生協教科書販売システム

② 授業で用いるグループワーク支援システム GWSS
③ 大学で運用している LMS LUMINOUS の拡張機能(BBS)
④ 研究室内図書管理システム（2009 年度版）
⑤ 研究室内図書管理システム（2011 年度版）

表 1　発見された問題

名称	説明	発見数
分岐判定記述漏れ	業務セオリー上でReadの値あり／なしを非決定に決定する分岐が存在する場合に、それに対応したReadの値あり／なし分岐を記述していない問題	10
分岐判定記述漏れによって発見されたモデルの問題	値あり／なしによる分岐判定を考慮しない事によって生じた、値なし状態でのUpdate／Deleteに関する問題	2
値ありフローの中でのオブジェクトのCreate問題	Createは値なし状態において作成させるべきで有り、値あり状態でのCreateでは、サービス期間中にオブジェクトを作成されない問題	1

7. Source2UPPAAL

保守プロセスにおける検査のプロセスは図 4 に示す通りである.

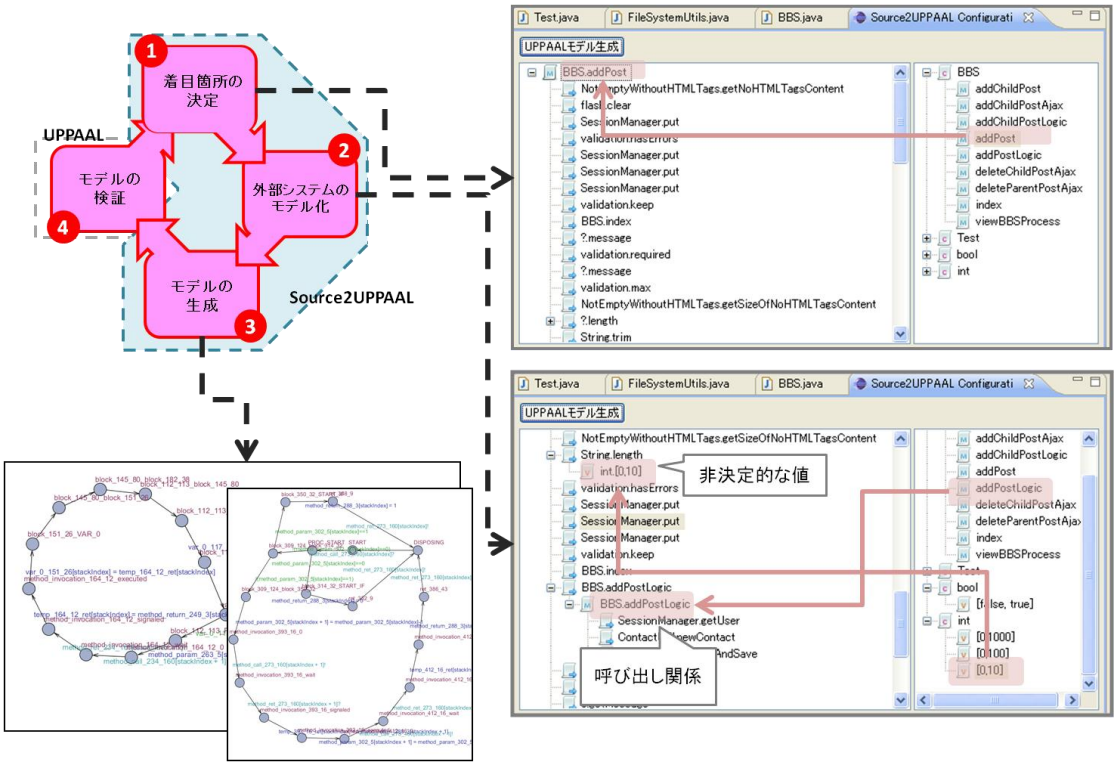


図 4　Source2UPPAAL の実行結果

利用者からはプログラム言語（Source）とモデル検査式を部品化した検査補助検査式が与えられる．ツールには Source を解析した結果得られたプログラムの抽象構文木が表示されるので該当する構成要素に対して変換定義を指定する．この指定作業が完了すると

Source2UPPAAL は UPPAAL モデルと検査式（全ノードに到達できるかを検査する式）を生成する．利用者は，ツールを通じて，非決定的な値を設定し，着目している領域を徐々に広げながら検査を進める．

下記の事例を用いて，Souce2UPPAL により，到達可能性と「停止しない」という不具合現象の検査を行い，原因を発見することができた．

- ET ロボコンプログラムへの適用
- Apache Commons のバグへの適用
- 会計伝票発行における無限ループ

到達可能性については，制御により期待される結果が得られない場合に，すべての制御フローが全て通りえる状態であるかを検査する．検査式は，生成されたロケーションへの到達の検査であり，自動的に生成できる．また，「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制御の条件式に対して設定することで，無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，自動で生成できる．

後者の検査は，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用意する．このモデルは UPPAAL を直接使い作成したものである．検査支援ツール Source2UPPAAL はソースコードから検査モデルを自動生成するとともに独自モデルを付加することもできる．検査するメソッドをツールに設定し，モデル化対象項目の想定される値を boolean 型もしくは int 型の非決定リストから，図 5 のように割り当てる．

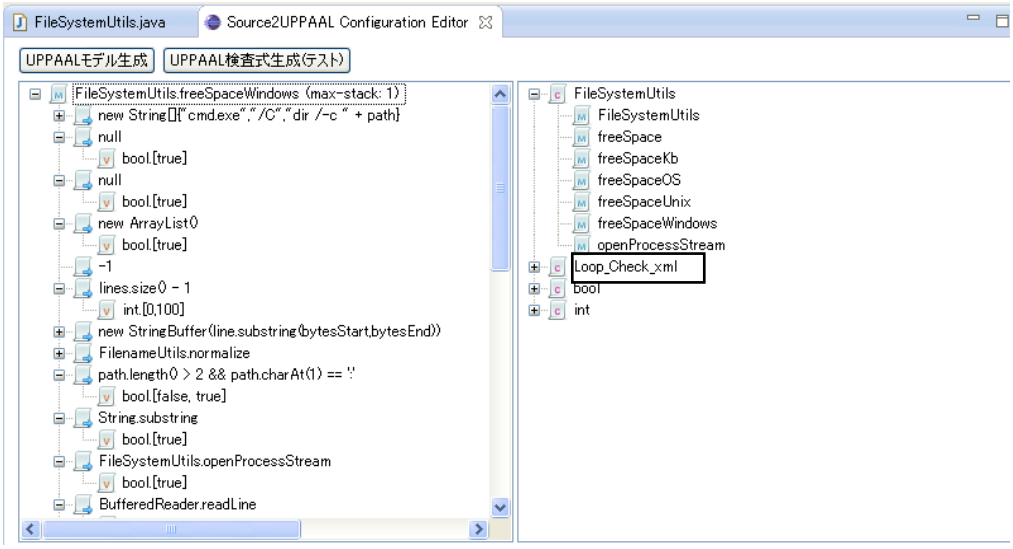


図 5　Source2UPPAAL 設定画面

さらに無限ループ判定用のモデルを追加する．追加は支援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れば自動的に表示される．今回の追加モデルのファイルは図中の四角形で囲われた Loop_Check.xml である．無限ループの判定モデル

は下図のように定義し，敷居値(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()を呼び出すモデルも図 6 のように定義する．

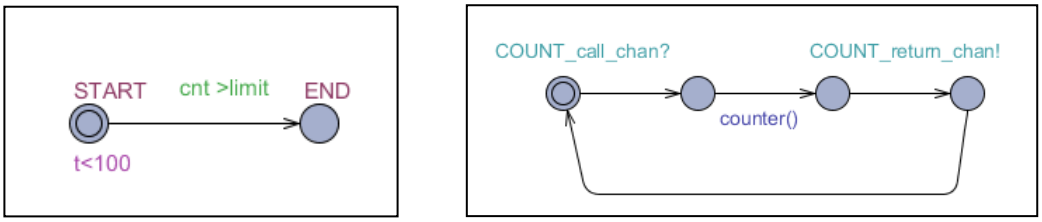


図 6　無限ループ判定用モデル

8．考察

本研究では，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，運用時の想定外の使用による不具合の特定，マイグレーションによるシステムの再構築時のシステムの仕様の保証といった開発現場でのモデル検査利用のシナリオを想定して課題に取り組んできた．ここでは，それぞれのシナリオで明らかになった事柄と今後の課題について述べる．モデル検査を実施する上での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ことの定義をその構成要素間で対応付けることである．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．

8.1 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ついての考察

要求分析モデルに対して，業務セオリーを検査したい対象データのステートマシン図で定義することで，UPPAAL モデルとその検査式を自動生成することができ，つぎの性質を検査することができた．

1) システムの永続化データに対して定義した基本的な性質である CRUD の振舞いとシステムの全サービスを定義した振舞いモデル間に矛盾が発生しない．
2) セキュリティ属性が定義されたデータのセキュリティ属性に関する基本的な性質に対して，システムのあるサービスがその属性に関する適切な振舞いを満たしている．ここで，「適切な」の意味は，セキュリティ評価の標準規格であるコモンクライテリアによって定めている．

これらの成果に関する発表論文[7]に対するコメントでは「CRUD の観点からシステムの振舞いを抽象化してモデル検査を適用するというアプローチは，現実的な形式検証の実現を目指す上で独創的かつ有望と考えます．」「分析者が検査式（CTL 式）をそのつど新規に書きくだす必要がなく，実務者向きの技術です．」といった，当初の狙い通りの評価を得ている．さらに，求められているのは，「CRUD 以外の観点」であり，本研究では，非機能要

件の１つであるセキュリティ要件をそのセキュリティ属性に基づき検査する方法を提案した.

属性に関するステートマシン図を定義し，その遷移を引き起こす振舞いとアクティビティ図の属性に対する振舞いの対応付けにより，CRUD の振舞いだけでなく，オブジェクトの属性に関連付けられた振舞いを扱うことができるようになった.

事例による実験の結果，モデル検査が終了しないケースへの対処がモデル定義に必要であることが判った.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定義する.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判ればよいので，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.

8.2 運用時の想定外の使用による不具合の特定についての考察

ソースコードを制御構造に基づき，UPPAAL モデルに変換することで，ソースコードの構造的欠陥を特定することを支援するツール Source2UPPAAL を開発した．ソースコードの不具合現象は，アプリケーション独自の性質に依存する場合，その依存する要素を特定しなければならない．これは一般には困難であり，本研究では，つぎのようにソースコードのアプリケーション独自の性質ではなく，不具合現象として生じるプログラムの一般的性質をモデル化することで検査を行った.

- 制御により期待される結果が得られない場合に，すべての制御フローが全て通りえる状態であるかを検査する.検査式は,生成されたロケーションへの到達の検査であり，自動的に生成できる.
- 「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制御の条件式に対して設定することで，無限ループ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，自動で生成できる.

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，非決定性をもつ値を想定しながら，検査できる機能をも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判っているが，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モデル化する必要がある.

8.3 マイグレーションによるシステムの再構築時のシステムの仕様の保証についての考察

上記２つのシナリオにおける業務セオリーの考え方をヒントに，マイグレーションによるシステム再構築時への適用を検討した．現段階では，仕様を理解している技術者が，あ

るデータの識別したい状態とそれらの間の遷移によって，そのデータの満たすべき性質を定義することができ，その振舞いが，ソースコードのどの要素に対応づくかを特定できた場合に，検査を行うことが可能であることがわかった．部分的な仕様をソースコードの要素と対応付けるためには，現状では開発者の理解に依存しているという問題がある．

９．今後の課題

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．

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の方針と合わせて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．この場合には，プログラム一般レベルではない不具合現象をモデル化する能力が要求されることと，ソースコードの要素との対応付けを行うために，ソースコード理解の支援が必要であると考える．

マイグレーションによるシステムの再構築時のシステムの仕様の保証に関しては，満たすべき性質を着目するデータに関するステートマシン図によって定義することを検討する．この場合，属性だけでなく，要件定義プロセスでの定義対象の拡張と同じ問題を解決する必要がある．また，マイグレーションを対象とするためには，複数の言語に対応した検査ツールを実現する必要がある．

１０．発表論文

[1] 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. （査読あり　委託研究に関連する研究）

[2] S. Ogata and S. Matsuura, A Review Method of Requirements Analysis Model in UML with Prototyping, Knowledge-Based Software Engineering, Proc of the 10th Joint conference on Knowledge-Based Software Engineering, IOS Press, pp. 181-190, 2012 （査読あり　委託研究に関連する研究）

[3] H. Okuda, S. Ogata and S. Matsuura, Mapping Rule Between Requirements Analysis Model and Web Framework Specific Design Model, Knowledge-Based Software Engineering, Proc of the 10th Joint conference on Knowledge-Based Software Engineering, IOS Press, pp. 207-216, 2012 （査読あり　委託研究に関連する研究）

[4] 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.（査読あり　委託研究に関連する研究）

[5] 奥田，松井，式見，野呂，岡田，小形，松浦，ユースケース記述の意図の明確化を目的とした初学者特有の問題点の分析,電子情報通信学会，信学技報，vol. 112，no. 314，KBSE2012-45，pp. 43-48，2012.（査読なし　委託研究に関連する研究）
[6] 小形，青木，奥田，松浦，データライフサイクルの妥当性に着目したモデル検査ツールの自動利用法,電子情報通信学会，信学技報，vol.112，no. 314，KBSE2012-56，pp.109-114，2012.（査読なし　委託研究の成果）
[7] 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 XVIX，日本ソフトウェア科学会 FOSE 2012，pp. 75-80.（査読あり　委託研究の成果）
[8] 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 XVIII，日本ソフトウェア科学会 FOSE 2012，pp. 241-242，2012.（査読なし　委託研究の成果）
[9] 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 KBSE 研究会　2013 年 3 月発表予定.（査読なし　委託研究の成果）

1．研究の背景および目的

1.1 背景

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，手戻りを防ぎ，高品質なソフトウェアを開発することがより強く求められており，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．モデル検査技術はテストでは実現できない網羅的検査に特徴があるが，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の検査式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大きな課題である．

本研究では，要件定義プロセス，保守プロセス，システム再構築におけるマイグレーションといった開発現場でのモデル検査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する．

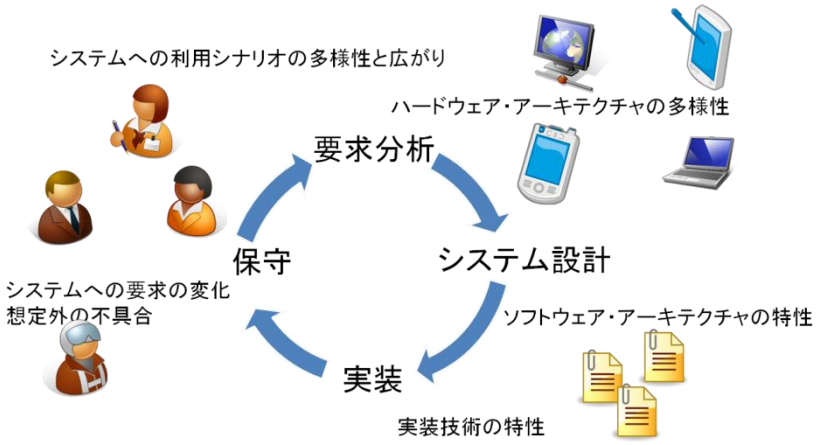


図 1-1　ソフトウェア開発の背景

1.2 研究課題

日本における企業はすでに何らかの業務システムを持っており，今後はそうしたレガシーシステムの様々な形態の再構築ビジネスが見込まれる．しかし，構築してから年月がたっているシステムでは，本来の仕様がわからなくなり，システムを再構築する規範となる仕様確定に苦労することが多い．また，顧客との間で合意した仕様に基づき作成された大規模システムが業務ルールを完全に網羅しているかを確認することも困難であり，想定外の使い方による不具合に対処することに多大な労力を要している．仕様書を見ても業務ルールが遵守されているかを確認するすべは無い．こうした現状の業務システムの満たすべき業務セオリーを第三者からも客観的に同じ評価ができるプロダクトを開発することが必要である．一方で，これらの問題の根本的要因でもある不十分な要件定義に対処するために，上流工程においても，客観的指標による早期の検証を行うことが必要である．

モデル検査技術は，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おり，上記の客観的指標となり得る技術である．し

かし，上流工程で検査可能なモデルを一般的な開発現場の技術者が定義することは容易ではない．モデル検査技術はテストでは実現できない網羅的検査に特徴があるが，開発現場で用いるためには，検査対象のシステムのモデルを検査可能な粒度で定義する方法と，その検証したい性質を検査式として定義する方法を，実際の開発現場での適用シナリオに対して適切に定義することが大きな課題である．

本研究では，要件定義プロセス，運用時およびマイグレーションによる再構築を含む保守プロセスといった開発現場でのモデル検査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する．本研究の研究課題としては，松浦研究室でこれまで研究開発してきたUML(Unified Modeling Language[1])を用いたモデル駆動要求分析手法ならびに，モデル検査ツールの１つであるUPPAAL[1][2]を用いたソースコードの欠陥抽出手法に基づき，下記の研究課題に取り組む．

1) 新規開発システムにおける要件定義で得られた仕様を客観的に評価する業務セオリーとその評価手法
2) 想定外の使い方等による不具合を既存のソースコードから発見する業務セオリーとその評価方法
3) 仕様書を喪失した稼働中のシステムの業務セオリーを抽出し，評価する手法

形式手法は検証における有効性が高いと考えられるが，一般的にその学習に時間がかかるだけでなく，要件定義のように，要素間の関係における不確定要素が多い段階で矛盾なく定義することが困難である．UMLを用いたモデル駆動要求分析手法では，開発者がユーザとシステムのインタラクションの観点から定義した要求分析モデルから顧客が理解しやすいプロトタイプを自動生成するモデル駆動開発手法である．UMLのような曖昧性を許容するが，一般的に理解しやすい形式の定義から，業務セオリーで検査することを通じて段階的に厳密な定義を導いていく開発手順が，多くの開発者が低い学習コストで高品質なプロダクトを開発するために有効であると考えている．

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，運用時の想定外の使用による不具合の特定，マイグレーションによるシステムの再構築時のシステムの仕様の保証といった開発現場でのモデル検査利用のシナリオを想定し，つぎのように課題に取り組む．

1) モデル駆動要求分析手法で定義したUML要求分析モデルからUPPAALモデルを生成する方法を確立し，データのライフサイクルおよび属性の制約に基づく業務セオリーの検証方

[1] UPPAALの説明
一般にモデル検査技術とは，検証したいシステムの構造に対し，検証したい性質を表す様相論理式’を定義し，構造がこの論理式を満たすかを調べることと，その検査ソフトウェアを指す．UPPAALはスウェーデンのUPPSALA大学とデンマークAALBORG大学によって開発されたモデリング，シミュレーション，検証のための統合ツールである．UPPAALでは複数プロセスから構成されるシステムを時間の概念を含んだ有限オートマトンの集合として，その構造を記述する．

法を確立する.

2) ソースコードからその制御フローを表すUPPAALモデルを生成し，想定外の使い方等による不具合現象や業務システムに必須の業務セオリーの検証方法を確立する.

3) 一般的なソースコードからの業務セオリーの抽出は困難であることから，システムのマイグレーション時にUPPAALモデルに基づき，元のシステムの仕様の保証を業務セオリーとして定義することを検討する.

具体的には，要求分析モデルやソースコードからの UPPAAL モデルへの変換ツール，検証式の定義支援ツールを研究開発する.

1.3 研究の意義

自然言語で記述された仕様が要求を満たしているかを客観的に判断することは難しい. 開発現場においては，仕様の正しさを判断する基準が，レビューの実施回数であったり，不具合発見率であったりする. 仕様の内容そのものを判断する試みもされているがまだ実施者のスキルへの依存が大きい. 開発現場で多発している不十分な要件定義や想定外の使い方による不具合を，形式検証技術であるモデル検査技術を適用して，現状あるプロダクト(UML モデル，ソースコード)を基に仕様を確認することができれば，プロダクトの品質向上や作業効率の観点からその成果は大きい. 経験重視の傾向が強い開発現場においてソフトウェア工学を導入することにより経験的なアプローチだけではなく，工学的なアプローチが行われる土壌が養われる.

高品質なソフトウェアを効率よく開発するためには，ソフトウェアに対する顧客の要求を定義する要求分析の工程が重要であり，その成果物である要求仕様は，後工程への情報を正確かつ十分に伝えなければならない. 要求仕様が達成すべき目的は，顧客がその妥当性を確認できること（顧客サイドの妥当性確認)，開発者が要求仕様を検査し，後工程への必要な情報を正確かつ十分に定義できること（開発者サイドの実現可能性の確認）が必要である. IEEE830[3]には要求仕様で定義すべき項目が述べられているが，項目の相互作用を十分理解しながら，要求仕様の目的を満たすように定義することは難しい. 本研究における手法はこうした要求仕様の品質を保証する 1 つの方法である.

具体的なソフトウェア開発の中で，本研究の提案方法を利用する場面としては，要求定義段階における仕様の不具合の早期発見，および，運用時の不具合現象に基づく，不具合の原因の発見を想定している.

さらに，開発現場でモデル検査による検査を一般的な開発者が有効利用可能な場面を想定した方法とその支援ツールを研究開発することで，つぎの効果が期待される，

- 大規模システムにおける仕様誤解や仕様の実現可能性の不具合の上流工程における早期発見
- 明確な仕様がわからないシステムのマイグレーションの精度向上
- これらによるプロダクトの品質向上，納期短縮，コストオーバの防止
- 経験的アプローチにのみに頼らない工学的アプローチへの意識改革と推進

２．実施内容

2.1 研究アプローチ

本節では，本研究のアプローチについて，本研究の基となる 2 つの関連研究「「UML を用いたモデル駆動要求分析手法」および「モデル検査ツールの１つである UPPAAL を用いたソースコードの欠陥抽出手法」に基づき説明する．

2.1.1 研究の全体像

松浦研究室ではこれまで UML(Unified Modeling Language)を用いたモデル駆動要求分析手法[7, 8, 9]ならびに，モデル検査ツールの１つである UPPAAL を用いたソースコードの欠陥抽出手法を研究開発してきた[4, 5, 6, 10]．

これらを基に，つぎのような複数の方向から研究課題に取り組む．

- 仕様漏れの原因の調査とシステム不具合の種類を文献等から調査し，自然言語記述の仕様から漏れる業務ルールとの相関関係を抽出することによって，検査により発見したい対象を確定し，業務セオリーとして定義する．
- 要求分析モデルでは，複数の機能を UML のアクティビティ図を用いて定義する．アクティビティ図では，その機能の振舞いのフローをアクションにより，データのフローをオブジェクトノードによって定義する．図 2-1 に示すように，このモデルから UI(ユーザインタフェース)プロトタイプを自動生成する．またソフトウェア開発の V モデルにならい，テスト設計者の観点から要求分析モデルにオブジェクト図を用いてテストケースを定義し，シナリオベース・プロトタイプを自動生成することもできる．これらのプロトタイプは顧客視点の妥当性確認と，テスト設計者視点からの要件定義を支援するが，これらの要件定義の実現可能性をシステムサイドから確認する必要がある．われわれは，機能の要件となる入出力データとその機能を実現するためのシステム内のエンティティ･データを分析し，要求分析モデルから UPPAAL モデルを自動生成して，すべての機能間でエンティティ･データのライフサイクルを CRUD の観点から検証する方法をモデル検査技術を用いて試みている．これは１つの業務セオリーであり，この方式をさらに検討して，データの属性に関する業務セオリーを策定する．要件を多角的に確認・検証することで，定義漏れや誤りを減少させる．

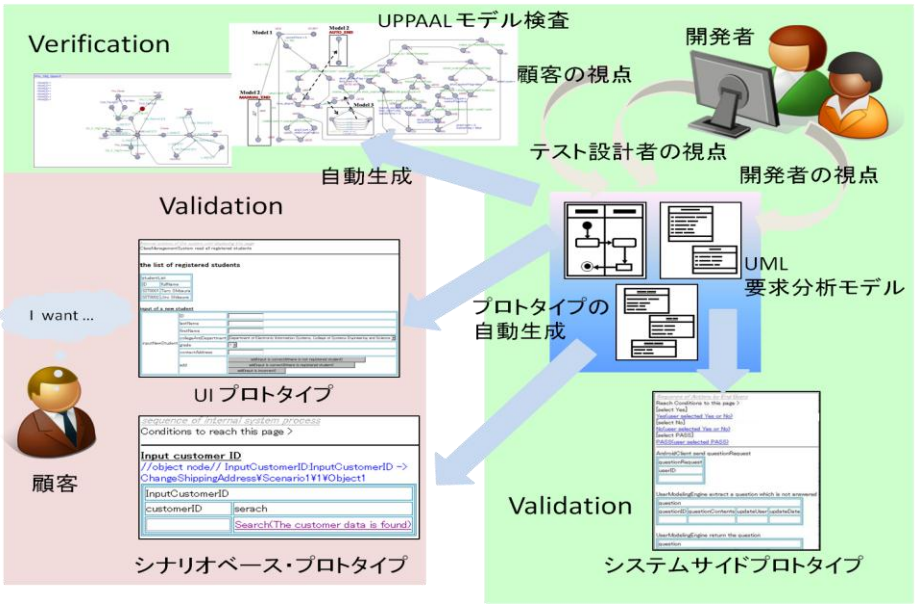


図 2-1　要求分析における要求仕様の定義と検証

- 業務における条件制御は複雑であり，制御項目の多くはデータベース上の条件マスタに格納され，これらの複数のマスタとプログラムのロジックを組み合わせて業務を実現している．そのため不具合原因が実データに依存している場合，テストデータでデバックしても再現できず，不具合の原因追求が難しい．このような場合，条件マスタのセオリーをモデル化し，遵守しなければならない仕様を検査式にすることでソースコードから生成した UPPAAL モデルに対し，想定外の動作を発見することができる．業務セオリーは仕様書がある場合には，そこから策定し，仕様書が無い場合には運用担当者の持つ知識もしくは運用マニュアルから策定する．
- 大規模なシステムの分析と検証には，規模とその処理時間の問題を解決しなければならない．UML 図と検査モデルの間のスケーラビリティを重視した，UML 図を検査モデルに変換する文法を考案する．
- 本学で運用している学習支援システム LUMINOUS の拡張機能の開発ではモデル駆動要求分析手法を用いて要求仕様を定義する．これらの例題に，検証支援ツールを適用し，問題点を分析し，手法を改善する．
- 業務セオリーに基づき，対象とする課題を記述可能な UML 図の記述手法を再検討する．

2.1.2 関連するこれまでの研究について

本節では，本研究の基となる 2 つの研究について説明する．本研究で検証対象とする要求分析モデルはこの「UML を用いたモデル駆動要求分析手法」により定義されたものである.また,「モデル検査ツールの 1 つである UPPAAL を用いたソースコードの欠陥抽出手法」に基づき，ソースコードならびに，UML 要求分析モデルを UPPAAL モデルに変換する方法を定義する．

(1) UML(Unified Modeling Language)を用いたモデル駆動要求分析手法

① 概要

要求分析段階は，顧客要求を分析し，要求仕様としてまとめる段階である．当該研究では，基本的にはユースケース分析と同様に，要求の最も核となる機能要求を中心に分析を行う．特に，以下の観点から，ユーザが直接操作するシステムの UI(User Interface)に機能要求が顕在する部分を分析対象とする．これを，ユーザとシステムの「インタラクション」と呼ぶ．具体的には， 業務プロセスをシステムのサービスとして定義する場合，ユーザが操作上で理解すべき点であるつぎの 4 つの観点から分析を行う．

1) 業務規則に基づくサービス成立時の入力データとその条件
2) サービス不成立時の条件
3) 各条件下の振る舞い
4) 振る舞いの結果としての出力

要求分析手法では，開発者が，上記 1) から 4) の観点から，業務遂行に必須な業務フローと業務データを UML モデルであるアクティビティ図とクラス図により定義する．アクティビティ図では，フローを構成するアクションと，オブジェクトノードの分類子となるクラスに定義されるデータとの関連を定義する．特に上記の観点における，ユーザの入力，サービスの不成立の条件,出力に関わるフローとデータが UI に顕在する要求に当たることから，これらを識別できるように，アクティビティ図を「ユーザ」「インタラクション」「システム」のパーティションに分けて定義する．さらに，業務データの具体例をオブジェクト図を用いて定義し，これらのモデルから HTML (Hyper Text Markup Language) の Web ページで構成される UI プロトタイプを自動生成する．このプロトタイプは，最終プロダクトの機能におけるシステムの内部処理および UI における外観を除いたシステムの模型であり，顧客は，この模型を通して顧客が業務遂行に必須なフローとデータを確認する．UI プロトタイプは，顧客による要求の妥当性確認のみならず，開発者のモデル理解にも有効である．それは，複数のモデルの整合性を 1 つの模型を通して確認することができるからである．開発者が各モデルとプロトタイプの関係を十分に理解できるように，要求分析手法では，図 2-2 に示すような要求分析モデルの各種定義段階に応じて段階的にリッチ化する UI プロトタイプ自動生成を実現している．なお，UML モデリングツールは，ChangeVision 社の astah*[11]を利用する．

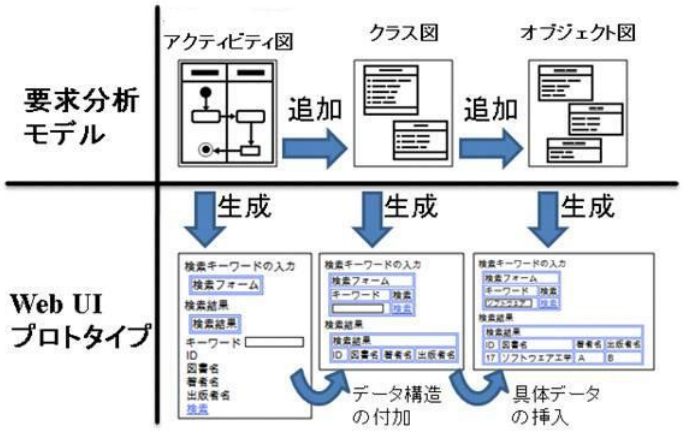


図 2-2　段階的な Web UI プロトタイプ自動生成

つぎに，プロトタイプ生成を行いながら，上記の観点から業務フローと業務データをある程度定義できた段階で，各フローに登場するデータがそのフローに合致するように具体データを定義したシナリオを定義する．この作業は，定義されたすべてのパスに対し，そのフローが成立する具体データを交えて，各フローが適切な処理を行っていることをプロトタイプにより確認するものである．最終的に，この確認の結果として得られた，妥当性確認済みの要求分析モデルから統合テスト仕様を自動生成することができる．生成される統合テスト仕様は，Web ブラウザを利用したテスト自動実行ツールである Selenium IDE 上で実行可能な形式でも生成される．

② モデル駆動要求分析の事例

本学で運用している学習支援システム LUMINOUS の拡張機能の要求分析を例に本手法を説明する．

学習支援システムは，講義や演習といったコース毎に，教材の配布，レポートの回収，アンケートの実施，履修者管理を行うことができる一般的な LMS ： Learning Management System の機能に加え，利用者である教員や学生の役割に応じた柔軟な権限管理が可能であることを特徴としている．本事例では，教員と学生間で，質問と回答のやり取りを支援する BBS 機能を開発した．

図 2-3 は BBS のユースケースである．アクターは LUMINOUS のユーザである学生と教員である．この BBS は単に学生と教員が意見交換を行うのではなく，学生の質問に対して，教員が回答し，教員がその質疑応答（FAQ）が質問を行った学生だけでなくすべての履修者にとって有益なやり取りであると判断した場合に，公開することができる仕組みをもつ．公開する場合には，学生の氏名を匿名にする．また，教員には主担当・副担当の役割があり，TA（Teaching Assistant）も BBS の利用に際しては，副担当の教員と同じ役割を果たす．

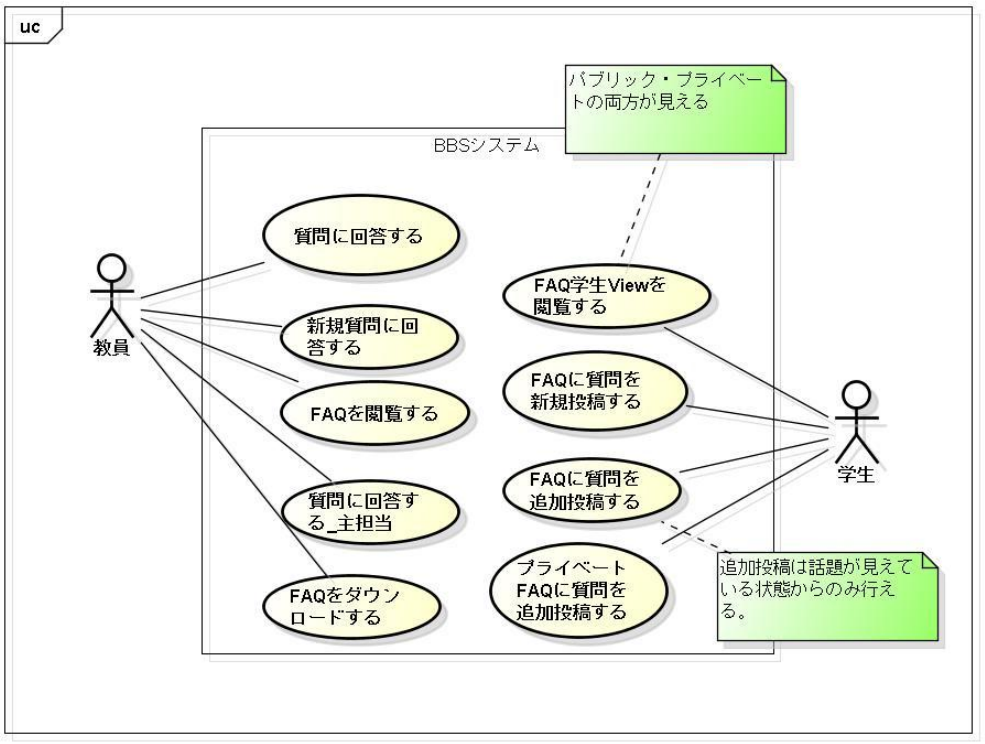


図 2-3　BBS のユースケース図

ユースケース分析は，ユーザとシステムのインタラクションを通して，システムの機能要件を明らかにすることを目的とする．ユースケースの詳細であるユースケース記述は，一般には自然言語記述のテンプレートを用いて定義するが，本手法では，アクティビティ図とクラス図を用いて，図 2-4 のように定義する．アクティビティ図では，開始ノードから始まり，ユースケースを実現するアクションの系列を逐次・分岐・反復・並列構造により，終了ノードまで定義する．これらの系列により，ユースケースの基本フロー・代替フロー・例外フローを定義する．系列はアクションノードをデシジョン・マージノードまたはフォーク・ジョインノードを用いて，フローにより連結することで定義することができる．アクションノードには自由な自然言語による，そのアクションノードが存在するパーティションを主体とする振舞いを定義する．

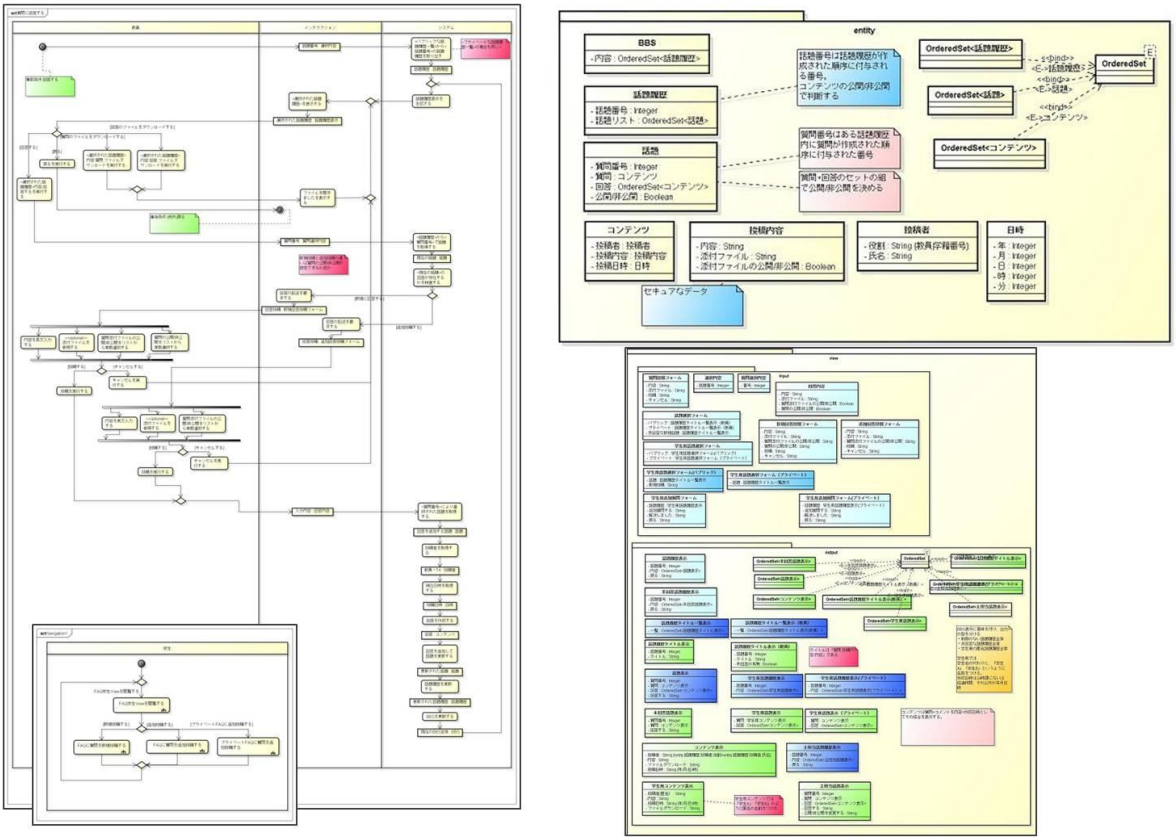


図 2-4　BBS のユースケース記述とクラス図

図 2-5 は，ユースケースを表すアクティビティ図の一部を拡大したものである．赤線の枠で囲んだ部分がアクティビティ図のパーティションの名前であり，アクションの主体を表す．ここでは，アクターである「教員」と「システム」の他に，これらの相互作用を明確にするための役割をもち，システムが直接ユーザとやり取りする振舞いを定義する「インタラクション」をパーティションとして明記している．各アクションはパーティションに所属することで，その主体が明確になり，ユースケースがユーザとシステムのやり取りとして定義される．

アクティビティ図には，振舞いを表すアクションノードだけではなく，データを表すオブジェクトノードを定義する事ができ，これにより，ユースケースにおいて必要なデータの定義を行う．このオブジェクトノードは，ユースケースの文脈で登場するインスタンスを表現するので，その構造を定義するクラスにより，その型を規定することができる．例えば，図 2-5 のシステム・パーティションに登場する「話題履歴」は，図 2-4 右側のクラス図内のクラスとして，その構造が定義されている．さらに，このオブジェクトノードの名前はアクションノード内の単語と対応付けることができる．これにより，ユースケースにおいては，どのような構造のデータをどのアクションで，どのように扱うかを定義することができる．ただし，一定の形式があるわけではないため，書き手により曖昧かつ不正確になることがある．

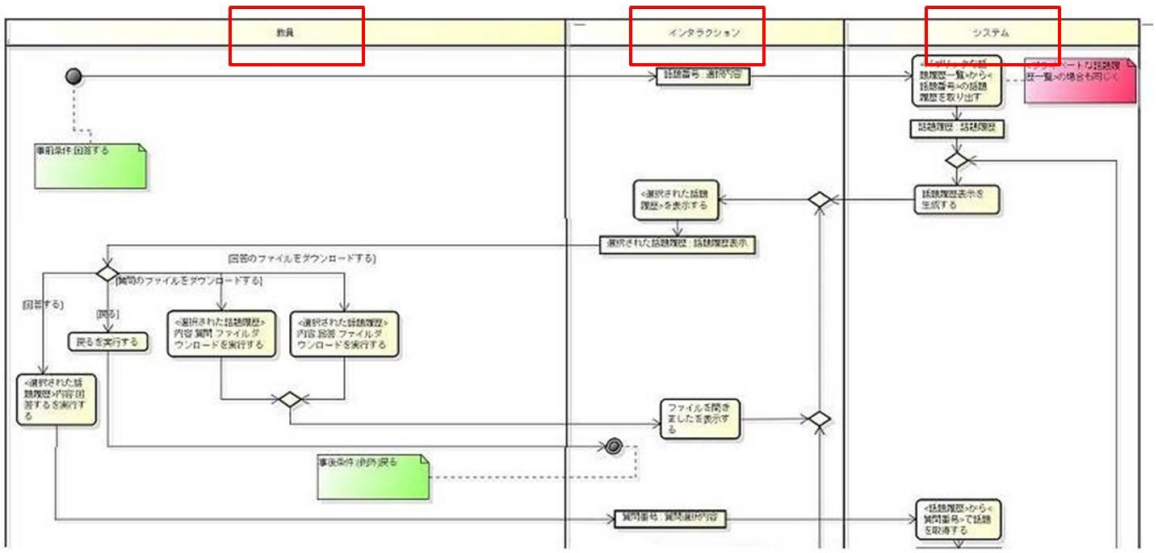


図 2-5　パーティションの役割

ユースケースは「システムを利用してユーザができること」をそれぞれ表しているが，システム全体として，ユースケースをどのような順序で利用するかのシナリオは定義されていない．すなわち，各ユースケースにはその事前条件，事後条件が存在し，この条件に従って，システム全体のサービスが決定される．図 2-6 は各ユースケースの関係を定義したナビゲーション図である．各サブアクティビティ・ノードがユースケース記述のアクティビティ図の呼出しになっており，この接合により，ユースケースを統合する．

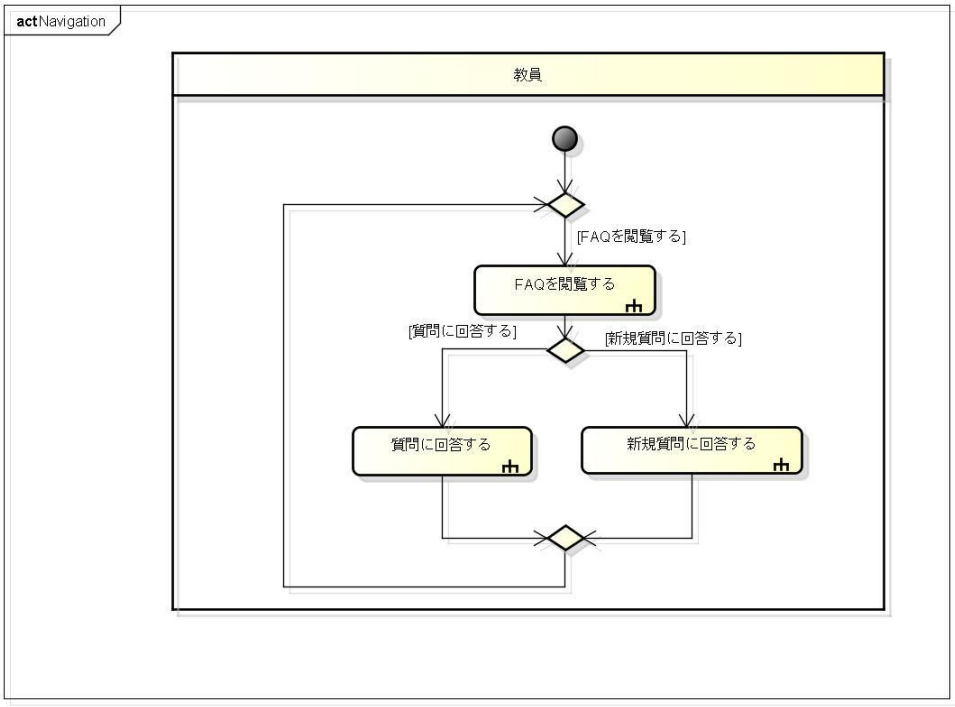


図 2-6　ユースケースの統合

モデル駆動要求分析手法では，ユーザとインタラクションパーティションのアクションノードの動詞を規定することで，ユーザインタフェースプロトタイプをモデルから自動生成することができる．図 2-7 と図 2-8 は図 2-6 で示される教員の利用画面である．図 2-7 の「タイトル」をクリックすると図 2-8 の画面が表示される．ここから，必要なファイルをダウンロードしたり，質問に回答することができるといった一連のサービスを操作しながら確認することができる．

図 2-7　プロトタイプ画面（その 1）

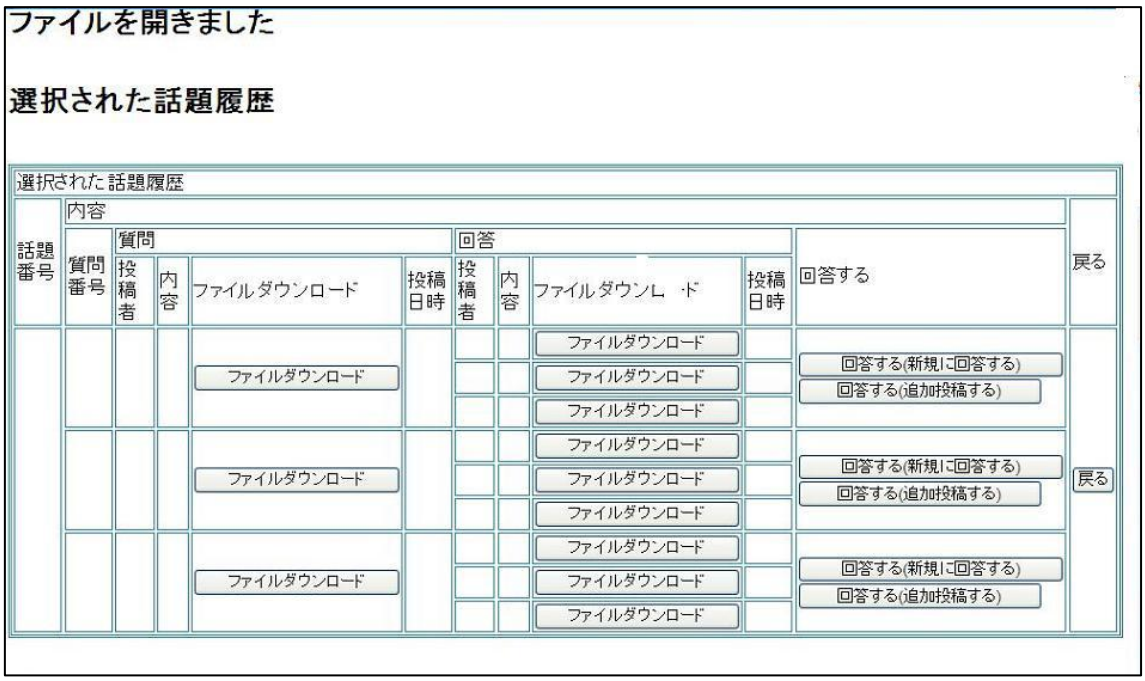


図 2-8　プロトタイプ画面（その 2）

(2) モデル検査ツールの１つである UPPAAL を用いたソースコードの欠陥抽出手法

① はじめに

一般に企業における業務システムは，紙ベースの業務をシステム化したものが多い．システム化に際しては処理の速さ，正確さ，確実さが求められるが，書類作業をシステム処理に置き換えているため，処理自体は基本的に人手でもできる．つまり個々の処理は簡単明瞭なロジックで構成されているといえる．システム化されても，業務継続の兼ね合いから業務フロー自体はそれほど変化しない．そのため業務システムの機能は，その種類，実行順序，実行条件の組合せが多岐わたる．

こうした処理の組合せが，業務ルールを構成しているため，プログラム開発者の業務知識が十分でない場合，システム設計者の意図を十分に理解せずに，設計者の意図しない組合せをソースコードに定義してしまうことがある．このような「意図しない組合せ」による不具合は，設計者が元々意図していないことから，レビューやテストにおいても発見が難しい再現性の低い潜在的欠陥である．

検査者が設計者と同等の知識を持っていたとしても，発見にはしらみつぶし的な調査・テストが必要になるため，不具合の発見には多大な労力を要する．また，不具合解消の確認は，見つけた不具合に対してのみであり，他に不具合がない保障にはならない．

近年，システム開発の上流工程において，システムが取りうる様々な状態を満たすかを検査するモデル検査が注目されている．モデル検査は，状態遷移系として定義されたシステムに対し，要求する性質を論理式で記述し，状態遷移系がこの論理式を満たすことを検査する手法である．

当該研究では，このモデル検査の技術を用いて，上述の「意図しない組合せ」が存在するソースコードから再現性の低い潜在的欠陥の発見及び修正後の不具合解消確認を行う手法を提案する．モデル検査技術を用いるにあたり既存のモデル検査ツールを利用する．通常モデル検査ツールはシステム開発の上流工程で利用する開発ツールであるが，本研究ではソースコードの欠陥抽出に利用する．また検査者の負担軽減のため検査モデルの生成を支援する支援ツールを作成した． 当該研究では Java で記述されたプログラムを検査対象とする．

② モデル検査ツールの選定

一般に開発現場にいるプログラム開発者の多くは，モデル検査に関する知識はほとんど持っていない.そのため開発現場での利用を想定すると,直感的に操作できる必要がある.ツールのグラフィカルな表示は直観的な理解の助けになる．モデル検査ツールはモデル検査のアルゴリズムを用いて自動的に検査を行う.今回,代表的なツールである UPPAAL[1], SPIN[2], PathFinder[3]を比較し，UPPAAL を使用することとした.

グラフィカルな検査結果表示は各ツールとも対応するが，モデル作成に関しては UPPAAL のみであった.また UPPAAL は時間制約を扱うことができ，リアルタイムシステムを検査する場合に有利である．UPPAAL は図 2-9 のようにロケーションとそれらをつなぐ遷移により構成される.ロケーションはシステムの状態を表し,遷移はシステムの振る舞いを表す.また遷移可能条件(Guard) 設定，変数値の更新(Update)，他プロセスとの同期通信ができ

る．与えられた論理式が満たされないと，その満たされない状態系列を反例として出力する．

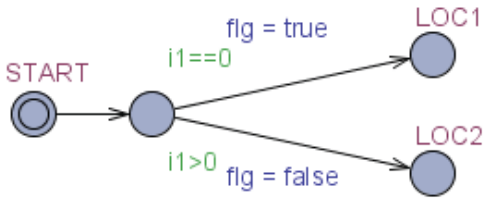


図 2-9　UPPAAL の基本モデル

③　ソースコードのモデル化

1)　モデル化の問題

実行順序・実行条件の組合せによる不具合を発見するためには，欠陥を内在しているソースコードを忠実にモデルに変換する必要がある．しかし，ソースコード内の全てのステートメントを検査モデルに変換すると検査する状態数が膨大になり，モデル検査ツールが状態爆発を起こす可能性が非常に高い．

2)　モデル化の方針

状態爆発をさけるために，ソースコードは抽象化してモデルに変換する必要がある．「意図しない組合せ」による不具合は，例えば出力結果の誤り，無限ループ，オーバフロー，タイムアウトの様に現象自体は明確に認識できるが，原因箇所の特定が難しい．モデル検査を行うためには，まず，ソースコード上の捉えたい現象を捉えることが可能な範囲で抽象化したモデルを状態爆発が生じない粒度で作成する必要がある．本提案手法はプログラムの実行順序や条件分岐の間違いにより発生する不具合を対象と考えている．そのため対象プログラムを for，if 文に着目し，それらの条件式に現れない変数を除外するように抽象化する．また，第 1 段階ではメソッド呼び出しも抽象化して検査し，必要に応じて抽象化した部分を再度モデル化する．

if 文については図 2-10 のようにモデル化する．条件式は true 状態[a] と false 状態[b]を表す二つの遷移に置き換える．if 文のボディは if 文の終点につながる遷移[c] に置き換える．遷移[d] は else 文のボディに置き換える．for 文についても同様に定義する．

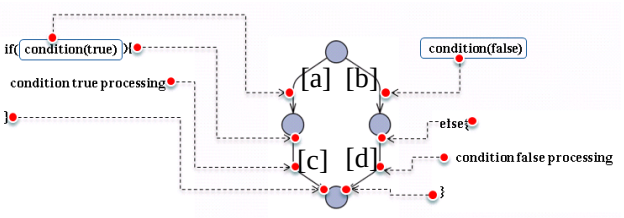


図 2-10　if 文のモデル化

3)　モデル検査の方針

欠陥の発見は安全性の検査により行う．安全性とは「システムがある条件下である正当でない状況に陥ることが決してない」ということである．

モデル検査でプログラムの安全性の検査を行う．安全性が確認できない場合，つまり不具合状態に陥るならば，モデル検査は不具合状態に至る反例を提示する．提示された反例を分析すれば正常状態から分岐して不具合状態へ到る状態変化の経路が見つかる．その経路に該当するソースコードの部分が欠陥である．

但し抽象化の粒度が粗いと，検査モデルでは安全性が確認できない(不具合が発生する)が実プログラムでは安全性が確認できる(不具合が発生しない)場合がある．そのため反例が実反例(実プログラムで存在する反例)であるかを UPPAAL 上のシミュレータで状態変化を追って確認する．もし偽反例(実プログラムで存在しない反例)ならばその判定根拠となる状態の発生を防ぐ制限をモデルへ追加して再度検査を行う．

④　欠陥抽出のプロセス

検査プロセスは支援ツールによる解析プロセスと UPPAAL による検査プロセスにより構成される(図 2-11)．支援ツールはプログラムを一旦 JavaML に変換し，構成要素(for 文・if 文）を抽出し，UPPAAL のモデルを自動生成する．検査者は生成モデルを UPPAAL で検査する．

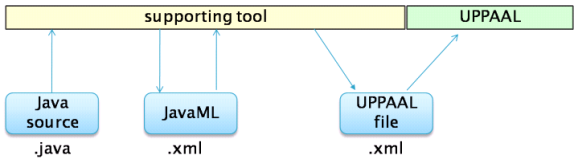


図 2-11　欠陥抽出プロセス

⑤　適用事例

1)　適用事例概要

適用対象は指定された会計期間のデータを検索して請求書を発行する会計システムのプログラムである．検索対象になる会計期間(From-To）を指定し，その範囲内のデータの検索･出力処理を行う.会計期間の値は 1～12(4 月～3 月に対応）で,”From”の値は常に”To”の値以下になる．検索条件は”From ”と”To ”で 1 セットとなるが，複数セット指定して一度に検索することもできる．但し”From”または”To”に”1”が指定された場合は，特殊処理として”From” を前年度の特別会計期間 13 に置き換えた検索期間を 1 セット追加する．これは前年度未処理データも含めた表示をするためである．このプログラムは特定の条件で無限ループを発生させる．元のプログラムは ERP(Enterprise Resource Planning）である SAP R/3 のアドオン開発言語 ABAP(Advanced Business Application Programming)により作成されたものである．サンプルプログラムは，帳票出力処理，画面制御等を除いたデータ抽出部の基本部分を Java で作成したものである．

2)　モデルの作成

無限ループは反復処理の制御フローに原因があると想定されるので，ソースコード上の該当部を支援ツールでモデル化する．これを Model1 とする．今回 Model1 は 5 つ生成された．

3)　モデル検査の実施

無限ループを表すモデルを Model2 として用意し，“無限ループが決して発生しない”という性質が満たされるかを Model1 と Model2 の結合モデルで検査する．性質が”満たされる”場合，そのモデルは検査対象から除外する．今回 5 つのうち 4 つを除外した．”満たされない”となった 1 つは反例が提示されるので次へ進む．

4)　実反例の判定

得られた反例が実反例か偽反例かを判定する．実反例ならば検証は終了である．今回入力データが実プログラムの想定値の範囲外であったため偽反例と判定した．

5)　モデルの再作成

入力値を想定値の範囲に制約するモデルを追加する．今回は実プログラムの入力画面と同じ入力値の制約を持たせたモデルを Model3 として追加した．また入力値に応じた状態変化を確認するために，Model1 のステートメントにおけるメソッド呼び出しのモデル化も行う．Model1～3 を結合したモデルを作成する(図 2-12)．

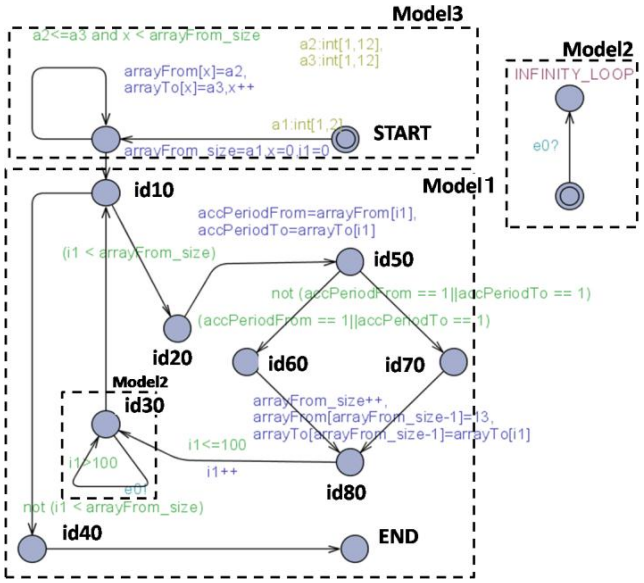


図 2-12　検査モデル

6)　モデル検査再実施

このモデルを検査すると反例が提示された．シミュレータ画面で反例を確認すると”From”=13，”To”=1 のレコードが大量に作成されることが判明した．入力値が仕様に合致しているので実反例と判定する．この反例で不具合に至る状態変化を確認する．”From”=13，”To”=1 の場合，id60 から id80 へのパスが疑わしいので，無限ループの原因は前遷移の Guard にあると考えられる．id50 から id60 のパスの条件式に問題が

ある．条件式(accPiriodFrom == 1 ||accPiriodTo == 1)における accPiriodTo は不要であり，条件式を accPiriodFrom == 1 に変更する．再度検査すると実行結果は”満たされる”となり，このモデルにおいて無限ループが起きないことがわかる．サンプルプログラムにも修正を行い無限ループが起きないことを確認した．

⑥　評価と考察

組込型システムであるライントレースロボットにも本手法を適用した．リアルタイムシステムへの有効性を確認するためである．組込システムはセンサー等のハードウェアを経由した入力や出力がその都度若干変わるため，不具合の原因の特定が難しい．我々はロボットおよび走行コース等の外部環境も含めたモデル化により不具合の原因となりうる要素を発見した．これらの適用事例より本手法に適している不具合は，明確に現象を認識できるが発生条件の特定が難しい不具合で，なおかつその状態変化をモデルに定義可能なものといえる．

モデル検査ツールはシステムの動作を有限状態モデルで表現し，網羅的に探索する．そのため本手法には状態を定義できることと，状態間の遷移条件を定義できることは必須である．ゆえに不具合の原因が検査者の認識できない内部ロジック(Java のライブラリ等)にあると状態変化をモデル化できず本手法での不具合発見は難しい．また，人の操作等の外部環境が不具合の主な原因ならば，その外部環境も含めたモデル化が必要である．

2.1.3 研究目標

(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，手戻りを防ぎ，高品質なソフトウェアを開発することがより強く求められている．モデル検査技術は，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．モデル検査手法をシステムの振舞いの検証に適用するためには，一般に振舞いをどのように正確かつ少ない状態数と遷移数でモデル化するかが難しい課題である．本研究の着眼点は，開発者が想定した振舞い定義モデルを特定の性質を満たす抽象的なモデルとして捉え，その特定の性質が取り得る状態とその遷移モデルと結び付けることで，検査項目となる論理式を自動生成することである．

(2) 業務セオリーの考え方と到達目標

①　業務セオリー

ソフトウェア開発工程は大きく分けると要求定義・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，つく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．要求定義段階では，これらの全

てを考慮するわけではないが，最終的なソースコードは本来全ての要求を満たさなければならない，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，ソフトウェア開発の全工程で考慮される，ソフトウェアの満たすべき性質を「業務セオリー」と呼び，これを整理することを目標とする．整理の目的は利用あるいは再利用しやすいようにある程度汎用的，あるいは定型的に定義できることを目指すという意味である．すなわち，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構築を目指す．

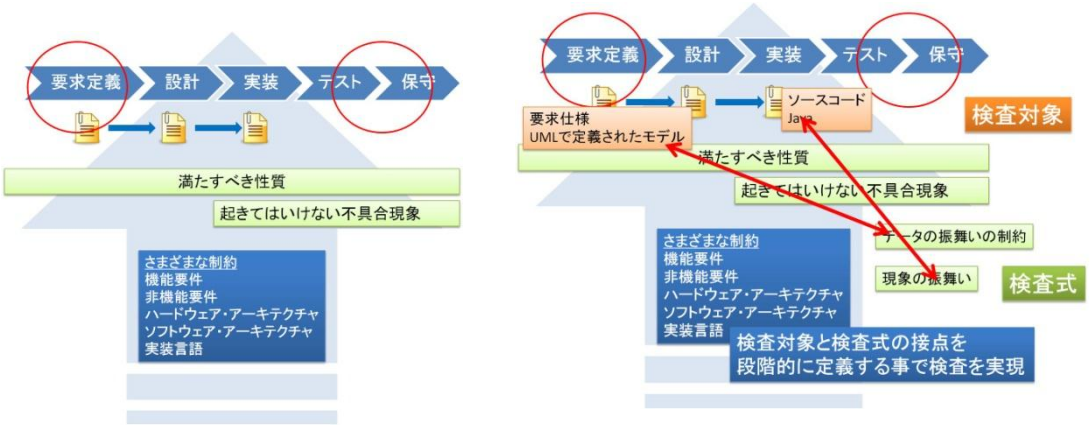


図 2-13　業務セオリーと検査の考え方

検査対象はそれぞれのフェーズのドキュメントであり，本研究では，それぞれのフェーズにおけるシステムの振舞いがそこに定義されているものとする．検査対象に対し，上述のような満たすべき性質を業務セオリーとして，汎用的な雛型から具体的アプリケーションに対して具体化し，検査対象と性質を規定する振舞いと状態の組とに対応付ける(図 2-13)．これにより，検査対象の定義要素で検査したい性質が定義できることになり，モデル検査技術の難しさを解消する１つの方法となる．

② 開発現場での利用シナリオ

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，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支援ツールを研究する．

要求をシステムの要件として捉え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，われわれ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プロトタイプにより，最終形態に近い形で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部分的な観点から定義したものであり，入力から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うまく実現で

きるのかといった要件の実現可能性の観点からの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．そこで第一のシナリオは「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，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，不具合現象はわかるが，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，技術者は経験に頼って，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技術者には困難である．こうした「運用時の想定外の使用による不具合の特定」を第二のシナリオとする．

既存のシステムを別の環境へマイグレーションする場合，通常のテストでは言語が違うため変換コードの正しさは，テストを実施した範囲でしか確認できない．旧システムの仕様を業務セオリーとして定義できれば，新旧システムを同じモデル検査用のモデルに変換することで，同じ仕様を満たすことを確認でき，効率的な検査ができると考えられる．このような「マイグレーションによるシステムの再構築時のシステムの仕様の保証」を第三のシナリオとする．

以上により，本研究では，モデル検査を開発現場で具体的に利用できそうな３つのシナリオを想定して，研究を進める．

- 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
- 運用時の想定外の使用による不具合の特定
- マイグレーションによるシステムの再構築時のシステムの仕様の保証

③ 検査対象と検査式

モデル検査手法をシステムの振舞いの検証に適用するためには，一般に振舞いをどのように正確かつ少ない状態数と遷移数でモデル化するかが難しい課題であるといわれている．要求定義の段階では,一般にはドキュメントは自然言語で定義される.自然言語記述では，柔軟な記述が可能であるが，曖昧性があり，そのまま検証することはできない．われわれの研究しているモデル駆動要求分析手法の成果物は，UML のモデルを用いて，形式化している．UML は VDM 等の形式仕様記述言語と比べて，曖昧性はあるが，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．

モデル駆動要求分析では，まず，ユーザとシステムのインタラクションを通して，システムの機能要件を明らかにすることを目的としたユースケースを抽出する．ユースケースの詳細であるユースケース記述は，アクティビティ図とクラス図を用いて定義する．アクティビティ図においては，下記の点を留意して，機能要件を明らかにする．

- 各ユースケース名に対応するアクティビティ図を定義する．
- 振舞いの主体（アクター，システム）をパーティションで表す．
- アクターとシステムのインタラクションを明確にするために，システムが直接，アクターに働きかける振舞いを「インタラクション」というパーティションに明記する．
- 主体のパーティションに書かれたアクションで，主体の振舞いを表す．

- 開始から終了までのアクションのフローで，振舞いの順序を表す.
- アクションの記述文で，振舞いの対象と動作を表す.
- デシジョン・マージノードにより，振舞いの代替および例外の分岐と合流を表す.
- フロー上のガードにより，代替および例外時の条件を表す.
- 事前条件・事後条件はそれぞれ開始・終了ノードにノートとして記述する.
- アクションの対象となるデータをオブジェクトノードで適切な位置に適切なインスタンス名と分類子となるクラスで定義する.

一方クラス図では,アクティビティ図に登場するデータをクラスとその属性により構造化する.

上記の観点からユースケース記述を行っても，UML のアクティビティ図においては，アクション記述，オブジェクトの識別子，ガード条件，事前条件，事後条件には自然言語記述による柔軟ではあるが，検証が困難な記述が多数ある．例えば，集合を規定する制約，すなわち，ある条件を満たす集合を表現する場合，仕様においてはそのデータのオブジェクト識別子を「パブリックな話題」のように名前の付け方で区別することがあり，簡潔に意味を表現する上で適している．この「パブリックな話題」の意味は,「話題履歴の話題リストのすべての要素である話題の属性「公開/非公開」は公開である.」というように，その型であるクラス定義により，厳密に定義することも可能である．しかし，要求定義の段階では，試行錯誤を伴うこともあり，はじめから後者のように厳密に定義を行うことは困難であり，段階的に厳密な定義を導入することが必要である．このため，モデル検査手法のみを単独で要求分析段階に用いることは難しい．しかし，要求定義の段階でも満たすべき性質に従って，要求を整理しなければ，アプリケーション自体の仕様における定義要素によって検査式を定義することは困難である．本研究では，つぎの２つに着目し，基本的な性質を満たすことや，要求を複雑化する非機能要求を満たしているかを判断できるように，要求を整理することを検討する.

- 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･データが，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がないことを保証する.
- 要求分析の複雑化の一因である非機能要件の内，セキュリティ要件を満たすことを保証する．セキュリティ要件は，データに対するセキュリティ属性とそれに対する振舞いにより定義が可能である.

一方，ソースコードは全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，プログラムの一般の性質等，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.

そこで，本研究ではシステムに対する満たすべき性質，起きてはいけない不具合現象を上流と下流から検討し，成果物である要求仕様（モデル駆動要求分析手法による UML モデル）と Java のソースコードを「検査対象」として，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としてモデル化し，２つのモデルの接点を段階的に定義することで，接合したモデル検査におけるモデルである有限状態機械上で，矛盾や到達できない状態を検査することにする．

(3) 研究目標の設定

本研究では，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，運用時の想定外の使用による不具合の特定，マイグレーションによるシステムの再構築時のシステムの仕様の保証といった開発現場でのモデル検査利用のシナリオを想定し，つぎのように課題に取り組むことを目標とする．

1) モデル駆動要求分析手法で定義したUML要求分析モデルからUPPAALモデルを生成する方法を確立し，データのライフサイクルおよび属性の制約に基づく業務セオリーの検証方法を確立する．
2) ソースコードからその制御フローを表すUPPAALモデルを生成し，想定外の使い方等による不具合現象や業務システムに必須の業務セオリーの検証方法を確立する．
3) 一般的なソースコードからの業務セオリーの抽出は困難であることから，システムのマイグレーション時にUPPAALモデルに基づき，元のシステムの仕様の保証を業務セオリーとして定義することを検討する．

到達目標に対し，つぎの研究目標を設定し，研究開発を進める．具体的には 2.1.2 節で述べたこれまでの研究成果を基に，小規模な事例開発に適用し，改善を進める．図 2-14 に示すように，上記の開発現場でのモデル検査利用のシナリオ 1)に対しては①と②の研究目標を，2)に対しては③④の研究目標を 3)に対しては⑤の研究目標を設定した．ただし，業務セオリーの考え方については個別ではなく，3 つの利用シナリオで統一感のある考え方を検討したいと考える．

① モデル駆動要求分析手法で定義したUML要求分析モデルからUPPAALモデルを生成する方法の確立：アウトプットは文法の定義と変換ツールの実装である．
② データのライフサイクルおよび属性の制約に基づく業務セオリーの策定：アウトプットは業務セオリーの一覧
③ ソースコードからその制御フローを表すUPPAALモデルを生成する方法の確立:アウトプットは文法の定義と変換ツールの実装である．
④ 不具合現象や業務システムに必須の業務セオリーの策定：アウトプットは業務セオリーの一覧
⑤ システムのマイグレーション事例の分析による業務セオリーの策定：アウトプットは業務セオリーの一覧

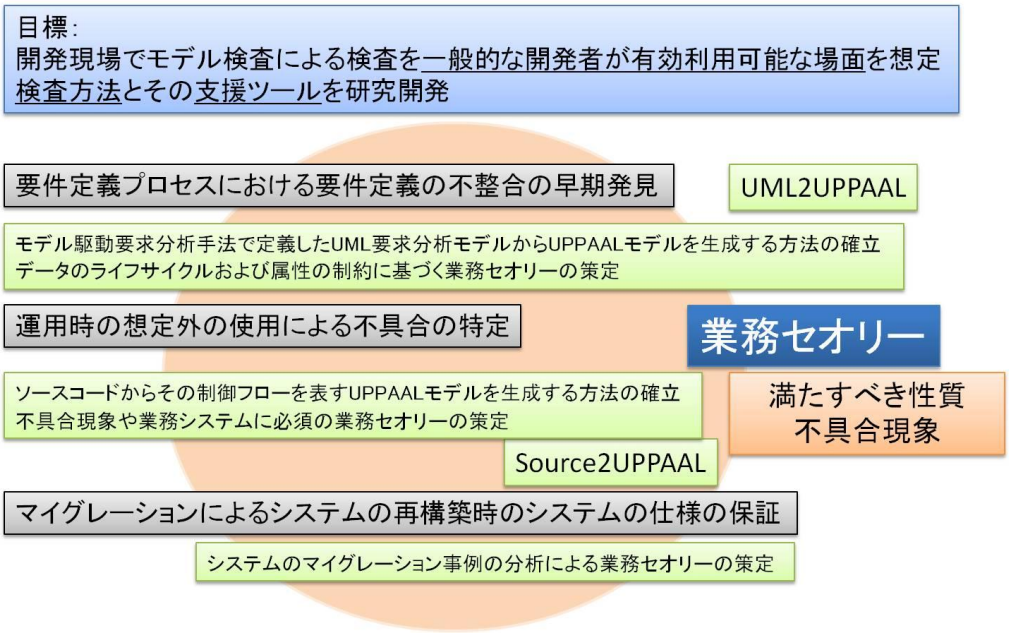

目標:
開発現場でモデル検査による検査を一般的な開発者が有効利用可能な場面を想定
検査方法とその支援ツールを研究開発
要件定義プロセスにおける要件定義の不整合の早期発見
UML2UPPAAL
モデル駆動要求分析手法で定義したUML要求分析モデルからUPPAALモデルを生成する方法の確立
データのライフサイクルおよび属性の制約に基づく業務セオリーの策定
運用時の想定外の使用による不具合の特定
業務セオリー
ソースコードからその制御フローを表すUPPAALモデルを生成する方法の確立
不具合現象や業務システムに必須の業務セオリーの策定
満たすべき性質
不具合現象
Source2UPPAAL
マイグレーションによるシステムの再構築時のシステムの仕様の保証
システムのマイグレーション事例の分析による業務セオリーの策定

図 2-14　目標の設定

2.2 研究の活動実績・経緯

本節では，委託研究をどのような手順で進めて行ったかを，関連する研究の学会発表内容を踏まえて，説明する．

2.2.1 研究の進め方

- 2.1.2 節で述べた研究について，全体ミーティングを行い，これまでの研究に関する情報の共有を図った．
- 研究目標とスケジュールに沿って，定期的（ほぼ毎週）にミーティングを行いながら，つぎの方針で研究を進めた．
  - 問題を解決するための方法を議論を通じて検討する
  - 方法を確認するためのツールを試作する
  - 方法を確認するための記述実験やツールの評価実験を行う
  - 実験結果を分析し，方法の問題を議論を通じて明らかにする
- ミーティングの資料はドロップボックスを用いて共有した．各回の議事の内容は議事録を参照のこと．ミーティングには，松浦研究室の大学院博士課程 1 年青木善貴，修士課程 2 年奥田博隆，式見遼，修士課程 1 年岡田康治，松井駿介，野呂惇の 6 名が参加した．
- 12 月にモデル駆動要求分析手法を用いて，以下の課題を用いた UML2UPPAAL の評価実験を集中的に行った．実験を行うまでに，各モデルを精査し，必要な CRUD アクションで整理した．
  - 大学院授業課題（生協図書販売システム・図書管理システム）
  - 研究室内図書管理システム 2009 年度版，2011 年度版
  - 大学で運用している LMS LUMINOUS の拡張機能(BBS)
  - 授業で用いるグループワーク支援システム GWSS
- 2.1.2 節で述べた研究に関連した成果や，本事業での成果について，随時，学会で論文発表を行った．学会では本委託研究を円滑に進めるため，情報収集や意見交換を行った．

2.2.2 発表論文

2012 年 7 月から 12 月に，下記の通り，2.1.2 節で述べた研究に関連した成果および本事業での成果について学会で発表を行った．

[1] 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. （査読あり　委託研究に関連する研究）

Abstract:

A key to success of high quality software development is to define valid and feasible requirements specification. We have proposed a method of model-driven requirements analysis using Unified Modelling Language (UML). The main feature of our method is to automatically generate a Web user interface prototype from UML requirements analysis model so that we can confirm validity of input/output

data for each page and page transition on the system by directly operating the prototype. This paper proposes a data life cycle verification method using a model checking technique UPPAAL. Exhaustive checking improves the quality of requirements analysis model which are validated by the customers through automatically generated prototype,

[2] S. Ogata and S. Matsuura, A Review Method of Requirements Analysis Model in UML with Prototyping, Knowledge-Based Software Engineering, Proc of the 10th Joint conference on Knowledge-Based Software Engineering, IOS Press, pp. 181-190, 2012 (査読あり 委託研究に関連する研究)

Abstract:

User interface prototyping is an effective method for users to validate the requirements defined by analysts at an early stage of a software development. However, a user interface prototype system offers weak support for the analysts to verify the consistency of the specifications about internal aspects of a system such as business logic. As the result, the inconsistency causes a lot of rework costs because the inconsistency often makes the developers impossible to actualize the system based on the specifications. For verifying such consistency, functional prototyping is an effective method for the analysts, but it needs a lot of costs and more detailed specifications. In this paper, we propose a review method so that analysts can verify the consistency among several different kinds of diagrams in UML efficiently by employing system-side prototyping without the detailed model. The systemside prototype system does not have any functions to achieve business logic, but visualizes the results of the integration among the diagrams in UML as Web pages. The usefulness of our proposal was evaluated by applying our proposal into a development of Library Management System (LMS) for a laboratory. This development was conducted by a group. As the result, our proposal was useful for discovering the serious inconsistency caused by the misunderstanding among the members of the group.

[3] H. Okuda, S. Ogata and S. Matsuura, Mapping Rule Between Requirements Analysis Model and Web Framework Specific Design Model, Knowledge-Based Software Engineering, Proc of the 10th Joint conference on Knowledge-Based Software Engineering, IOS Press, pp. 207-216, 2012 (査読あり 委託研究に関連する研究)

Abstract:

Model Driven Development is a promising approach to develop high quality software systems. We have proposed a method of model-driven requirements analysis using Unified Modeling Language (UML). The main feature of our method is to automatically generate a Web user interface prototype from UML requirements

analysis model so that we can confirm validity of in-put/output data for each page and page transition on the system by directly operating the prototype. We have shown that the requirements analysis model has traceability to the final product by implementation experiment from the requirements analysis model. This paper proposes a mapping rule in which design information independent of each web application framework implementation is defined based on the requirements analysis model.

[4] 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. (査読あり　委託研究に関連する研究)

Abstract:

A key to success of high quality software development is to define valid and feasible requirements specification. We have proposed a method of model-driven requirements analysis using Unified Modeling Language (UML). The main feature of our method is to automatically generate a Web user interface mock-up from UML requirements analysis model so that we can confirm validity of input/output data for each page and page transition on the system by directly operating the mock-up. This paper proposes a support method to check the validity of a data life cycle by using a model checking tool “UPPAAL” focusing on CRUD (Create, Read, Update and Delete). Exhaustive checking improves the quality of requirements analysis model which are validated by the customers through automatically generated mock-up. The effectiveness of our method is discussed by a case study of requirements modeling of two small projects which are a library management system and a supportive sales system for text books in a university.

[5] 奥田, 松井, 式見, 野呂, 岡田, 小形, 松浦, ユースケース記述の意図の明確化を目的とした初学者特有の問題点の分析,電子情報通信学会, 信学技報, vol. 112, no. 314, KBSE2012-45, pp. 43-48, 2012. (査読なし　委託研究に関連する研究)

抄録：

ユースケース分析はシステムに対する機能要求をユーザとシステムのやり取りとして明確に定義する有効な手段である. ユースケース記述は一般にテンプレートとして記述項目は定まっているが, 記述は非形式的であり, 記述の良し悪しを判断しにくい. 本学ではこれらの項目をUML(Unified Modeling Language)のフローやデータの可視化の有用性に注目し, 形式化の足がかりとして, アクティビティ図で記述することで, より明確な機能要件を定義することを目的とした学習を行っている.

われわれは, ユースケースの記述観点を理解していない為に, 実現したい意図が不明確となる初学者特有の問題点を分類し,修正を支援する研究を行っている.本稿では, 複数年度を対象に, 学習時間の異なる初学者のモデルとシステム構築を経験した大学院生のモデルに対して比較実験を行い, この分類の有効性を議論する.

[6] 小形，青木，奥田，松浦，データライフサイクルの妥当性に着目したモデル検査ツールの自動利用法，電子情報通信学会， 信学技報， vol.112， no.314， KBSE2012-56， pp.109-114，2012.（査読なし　委託研究の成果）
抄録：
モデル検査技術は，仕様を効率的かつ網羅的に検査する有望な技術である．しかし，開発者が適切にモデルと仕様を書くノウハウや，モデル検査の基礎概念等を習得するコストが高いことが問題である．そこで，これらの習得コストを一切不要とし，かつモデル検査技術の知識を持たない開発者がモデル検査ツールを活用できるように，オブジェクト指向設計支援としてモデル検査ツールの自動利用法を提案する．本研究では，基本設計段階において定義量の少ないUML要求分析モデルを段階的に洗練することを目的に，データの有無（値あり，値なしの2状態）に着目したCRUDによるデータの状態変化（データライフサイクル）の妥当性を効率的に検査することにモデル検査ツールを利用する．

[7] 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 XVIX，日本ソフトウェア科学会 FOSE 2012，pp. 75-80.（査読あり　委託研究の成果）
抄録： 高品質ソフトウェア開発の成功の鍵は妥当かつ実現可能な要求仕様を定義することである．本稿では，ユーザとシステムのインタラクションを表す要求仕様に対して，データの基本機能であるCRUD(Create， Read， Update， Delete)に着目し，モデル検査技術を利用して，システムが扱う永続化候補のデータが普遍的に満たすべき性質を表すデータ・ライフサイクルの観点から要求仕様の実現可能性を検証する手法を提案する．

[8] 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XVIII，日本ソフトウェア科学会FOSE 2012，pp. 241-242， 2012.（査読なし　委託研究の成果）
抄録：我々は形式的手法の開発工程への適用のしにくさを解決するため，利用者のモデルへの理解を促しつつ，段階的にモデルを抽出させるツールを検討している．本論文ではそのプロトタイプについて報告する．

今後の本事業の成果に関連する発表の予定は次のとおりである．
[9] 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KBSE研究会　2013年3月発表予定(査読なし　委託研究の成果)

2.2.3 発表論文，発表に対する意見等

[7]および[8]の論文は湯布院で開催された日本ソフトウェア科学会ソフトウェア工学の基礎ワークショップで発表した論文である．本ワークショップは3日間の合宿形式のワークショップで，ソフトウェア工学に関する査読を通過したプルペーパー10件とショートペーパー19件の発表と，27件のライブ論文・ポスター発表が行われた．[7]「要求分析にお

ける CRUD 観点のモデル検査技術の適用」は，松浦研究室の D1 生である青木善貴さんが，ショートペーパーとして研究発表を行った．[8]「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」は研究員の谷沢智史さんがライブ論発表とポスター発表として研究発表を行った．谷沢さんの発表は本ワークショップにおける参加者の投票により，優秀なポスター発表であるとして，図 2-15 の通りライブ論文賞を受賞した．[7]については下記のコメントがあった．

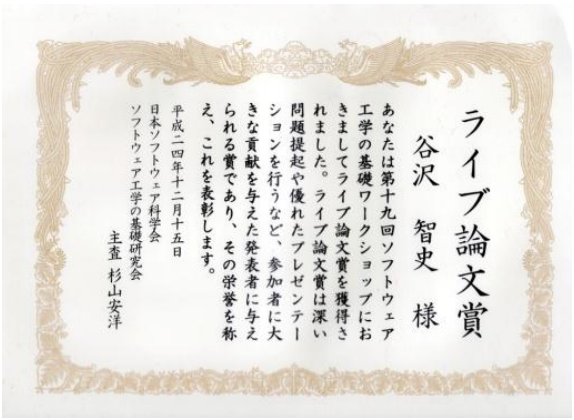
コメント内容

アクティビティ図のデータ処理部分に焦点を絞り，CRUD の観点から不具合がないかどうかをモデルチェッカーで検査する手法で，分析者が検査式（CTL 式）をそのつど新規に書きくだす必要がなく，実務者向きの技術です．

CRUD の観点からシステムの振舞いを抽象化してモデル検査を適用するというアプローチは，現実的な形式検証の実現を目指す上で独創的かつ有望と考えます．実験も現実に近い例題を用いて定量的に評価しています．皆様には，手法の理論的な妥当性，および現実のシステム検証に対してどの程度有用かを御議論頂きたく存じます．

CRUD は情報システムの基本という割り切りをもとに，形式手法導入の突破口としようという姿勢が読み取れ，興味深いです．

また，CRUD 観点で何ができるかは記述されていますが，逆に CRUD では把握できないシステム要素は何なのか，言い方を変えると「CRUD 以外」を特定の（複数の）用語でどう表現できるのかについても，御議論いただきたく存じます．


ライブ論文賞

谷沢　智史　様

あなたは第十九回ソフトウェア工学の基礎ワークショップにおきましてライブ論文賞を獲得されました。ライブ論文賞は深い問題提起や優れたプレゼンテーションを行うなど、参加者に大きな貢献を与えた発表者に与えられる賞であり、その栄誉を称え、これを表彰します。

平成二四年十二月十五日

日本ソフトウェア科学会

ソフトウェア工学の基礎研究会

主査　杉山安洋

図 2-15　受賞

2.2.4 進捗状況実績

進捗状況表（実績）は表 2-1 のとおりである.

表 2-1　進捗状況表（実績）

研究テーマ名	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に関する研究

作業項目		6月	7月	8月	9月	10月	11月	12月	1月	進捗状況
研究準備(仕様漏れの原因の調査)	実	→								完了
研究準備(システム不具合の種類の調査)	実	→								完了
研究準備(関連研究の調査)	実	→								完了
中間目標1(モデル駆動開発要求分析手法で定義したUML要求分析モデルからUPPAALモデルを生成する方法の確立)	実	→	→	→	→	→	→			完了
中間目標2(データのライフサイクルおよび属性の制約に基づく業務セオリーの策定)	実		→	→	→	→	→			完了
中間目標3(ソースコードからその制御フローを表すUPPAALモデルを生成する方法の確立)	実	→	→	→	→	→	→			完了
中間目標4(不具合現象や業務システムに必須の業務セオリーの策定)	実		→	→	→	→	→	→		完了
中間目標5(システムのマイグレーション事例の分析による業務セオリーの策定)	実					→	→	→		完了
中間報告の準備	実		→	→	→	→				完了
最終成果のとりまとめ	実							→	→	完了

2.2.5 学会参加

本委託研究を円滑に進めるため，ソフトウェア工学に関する下記の国際会議，国内ワークショップ，研究会に参加し，情報収集および意見交換を行った.

- 国際会議 COMPSAC2012（7 月）
- 国際会議 JCKBSE2012（8 月）
- 知能ソフトウェア工学研究会（7 月，11 月）
- ソフトウェア工学の基礎ワークショップ（12 月）

COMPSAC2012(the 36th Annual IEEE International Computer Software and Applications Conference)はIEEEが開催する今年で36回目の開催というソフトウェアに関する伝統のある国際会議である．今年のキーワードは Trust ， Security，Cloud であり，大規模なシステムやサービスの信頼性や安全性の保証に関する研究発表やパネル討論が行われた．大規模なシステムやサービスの信頼性や安全性の保証に関する多くの知見を得ることができ，本委託研究の方向性を確認することができ，有意義であった．また，本委託研究のテーマでもある要件定義プロセスと保守プロセスにおける信頼性や安全性を検討する上での参考になった.

JCKBSE2012 (the 10th Joint Conference on Knowledge-Based Software Engineering) は電子情報通信学会知能ソフトウェア工学研究会が企画し，ギリシャのピレウス大学が今回の開催を行った国際会議である．松浦が本会議の Program co-chair を務めた．本会議はソフトウェア工学，知識工学，人工知能分野にまたがる知的ソフトウェア開発環境に関する様々な研究分野の論文が発表された．それぞれ，活発な議論が行われ，本委託研究のテーマでもある要件定義プロセスと保守プロセスにおける知的支援を検討する上での参考になった．

電子情報通信学会知能ソフトウェア工学研究会がはこだて未来大学で，電子情報通信学会ソフトウェアサイエンス研究会との合同研究会として開催された．本委託研究のテーマでもある要件定義プロセスにおける業務セオリーの 1 つとして考えているセキュリティ要件に関して,「UML 要求分析モデルとコモンクライテリアに基づくセキュリティ要求分析の統合手法」のタイトルで発表を行い，10 分の質疑応答時間にさまざまな議論が行われ，本研究の方向性が確認でき，大変有意義であった．また，ディペンダビリティ等ソフトウェア工学の動向を知る上で役に立った．

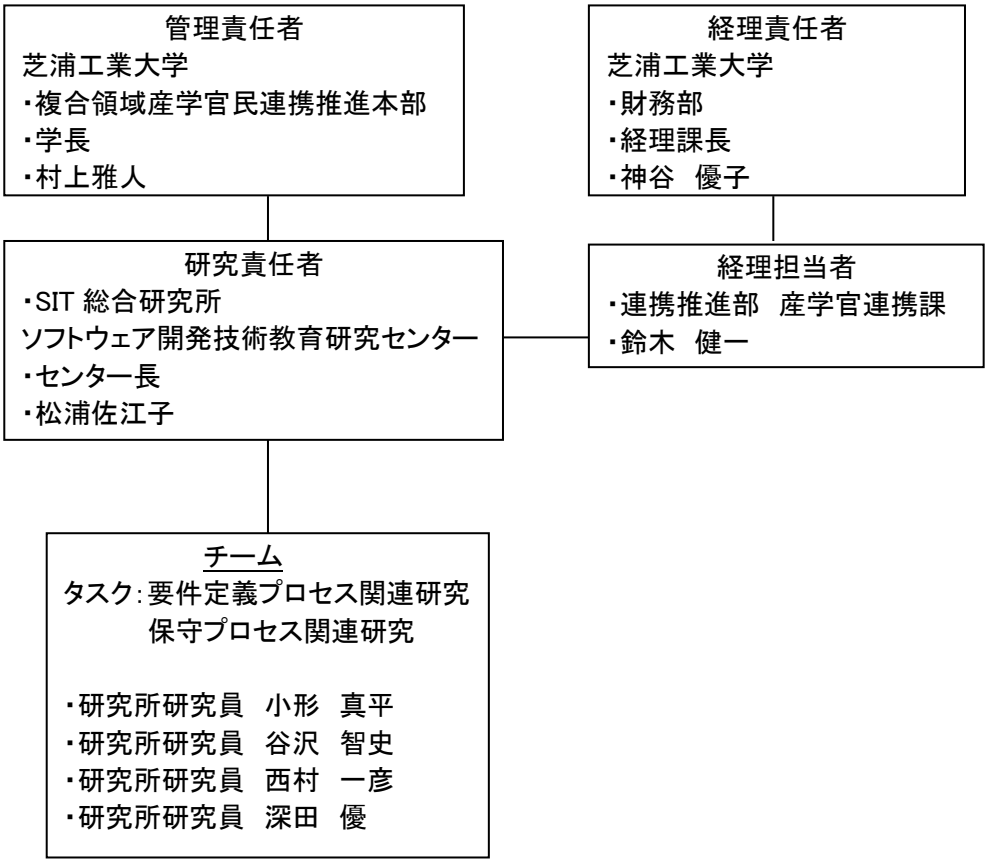
電子情報通信学会知能ソフトウェア工学研究会が金沢大学で開催された．小形氏が IPA プロジェクトの成果を「データライフサイクルの妥当性に着目したモデル検査ツールの自動利用法」と題して研究発表を行った．本委託研究のテーマであるモデル検査ツールの利用方法についての発表を行い，ステートマシン図の意義をもう少し分かりやすく説明する必要があることがわかった．他にもモデル検査をシステム運用作業時の運用手順書の不具合を発見するために利用する研究があり，モデル検査の利用方法として参考になった．

日本ソフトウェア科学会ソフトウェア工学の基礎ワークショップが湯布院で開催された．本ワークショップは 3 日間の合宿形式のワークショップで，ソフトウェア工学に関する査読を通過したプルペーパー10 件とショートペーパー19 件の発表と，27 件のライブ論文・ポスター発表が行われた．今回は松浦研究室のＤ1 生の青木氏が本委託研究の成果である「要求分析における CRUD 観点のモデル検査技術の適用」についてショートペーパーとして研究発表を行った．また谷沢氏が本委託研究の成果を「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」としてライブ論文発表とポスター発表を行った．ポスター発表では谷沢・松浦・西村・小形がそれぞれ参加者に説明し，意見交換を行った．本委託研究のテーマであるモデル検査ツールの利用方法について上流および下流からのアプローチについて発表し，実用性を考慮した研究であるとのコメントをいただいた．谷沢氏の発表は本ワークショップにおける参加者の投票により，優秀なポスター発表であるとして，ライブ論文賞を受賞した．モデル検査を専門にする研究者との意見交換を行い,他のモデル検査技術を利用することの可能性に関する意見をいただくことができた．

2.3 研究実施体制

本節では，本事業の研究実施体制および研究者のプロフィール，研究チームの役割を説明する．

2.3.1 研究実施体制



- 学生アルバイト

  芝浦工業大学　松浦研究室

  大学院博士課程 1 年　青木善貴
  大学院修士課程 2 年　奥田博隆
  大学院修士課程 2 年　式見遼
  大学院修士課程 1 年　岡田康治
  大学院修士課程 1 年　松井駿介
  大学院修士課程 1 年　野呂惇

2.3.2 研究責任者およびメンバーのプロフィール

（ふりがな） 氏　名	まつうら　さえこ 松浦　佐江子
所属機関	芝浦工業大学
所属 （部署名）	デザイン工学部デザイン工学科 SIT 総合研究所ソフトウェア開発技術教育研究センター
役　職	教授 センター長
住　所	〒337-8570 埼玉県さいたま市見沼区深作 307
TEL	048-687-5094
E-mail	matsuura@se.shibaura-it.ac.jp

【学歴（大学卒業以降)】
津田塾大学学芸学部数学科卒業
津田塾大学理学研究科数学専攻修士課程修了
津田塾大学理学研究科数学専攻博士課程単位取得退学
博士（情報科学）早稲田大学.

【職歴】
(株)管理工学研究所・研究員
情報処理振興事業協会新ソフトウェア構造化モデル研究本部・研究員
津田塾大学学芸学部情報数理科学科・非常勤講師
芝浦工業大学システム工学部電子情報システム学科・助教授
芝浦工業大学システム工学部電子情報システム学科・教授
芝浦工業大学デザイン工学部デザイン工学科・教授

【研究実績】
- 文部科学省大学教育・学生推進事業【テーマ A】大学教育推進プログラム（平成 21 年度～23 年度）工学系技術者のソフトウェア開発技能育成
- 科学研究費基盤研究（C）(平成 22 年度～24 年度）ソフトウェア開発技術者育成 PBL のためのモデル駆動型要求分析支援ツールの研究
- 科学研究費基盤研究（C）(平成 18 年度～21 年度）e-Learning を活用したソフトウェア工学教育の研究
- 科学研究費基盤研究（C）(平成 15 年度～17 年度）ユーザの意図に反する「悪意のある」振る舞いパターン検出のフレームワークに関する研究

- 経済産業省，平成１７年度産学協同実践的ＩＴ教育促進事業「産学協同実践的ＩＴ教育基盤強化事業」，平成 17 年度，組込みソフトウェア教育プログラム開発・実証
- 芝浦工業大学特別経費予算・プロジェクト研究助成，平成 18 年度－平成 20 年度，オブジェクト指向開発における妥当性検査
- 芝浦工業大学特別経費予算・プロジェクト研究助成，平成 21 年度，UML に基づく非機能要求モデリング手法

【主な論文・著書】

[1] 青木，松浦，ソースコード解析を利用したモデル検査に基づく欠陥抽出手法，ソフトウェア工学の基礎 XVII，日本ソフトウェア科学会 FOSE 2010，pp. 95-100， 2010.
[2] Shinpei Ogata, Saeko Matsuura, A Method of Automatic Integration Test Case Generation from UML-based Scenario, WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS, Issue 4, Volume 7, pp. 598-607, April 2010.
[3] 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.
[4] 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.
[5] 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，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論文集，pp. 79-86，2008.
[6] 松浦：実践的ソフトウェア開発実習によるソフトウェア工学教育，情報処理学会論文誌，Vol. 48，No. 8，pp. 2578-2595， 2007.
[7] Matsuura S., Kuruma H. and Honiden S., EVA ： A Flexible Programming Method for Evolving Systems, IEEE Transactions on Software Engineering, Special Issue on Formal Methods in Software Practice, Vol. 23. No. 5, 1997, pp. 296-313.

小形真平
2012 年　芝浦工業大学大学院工学研究科博士課程終了，博士(工学)取得．現在，信州大学工学部情報工学科助教．オブジェクト指向開発技術および要求工学手法に関する研究に従事．IEEE，ACM，情報処理学会，電子情報通信学会，日本ソフトウェア科学会，各会員．

谷沢智史
2008 年 電気通信大学大学院情報システム学研究科博士後期課程単位取得満期退学．在学中に業務システム，オンラインゲームなどの開発を経て，ベンチャー企業の立ち上げに参加し，DVD プレイヤーなど映像応用ソフトウェア，センサーネットワークに関する組込みシステム開発などを行ってきた．現在はソフトウェア設計技術，可視化技術などソフトウェア工学関連の研究開発に従事．

西村一彦
1988 年 東京理科大学理工学研究科修士課程修了．同年(株)東芝入社．人工知能・ソフトウェア工学に関する研究開発に従事. 2003 年(株)ボイスリサーチ設立. 現在, 同社取締役. 2011 年 電気通信大学大学院情報システム学研究科博士後期課程修了．博士（工学）．2009 年合同エージェントワークショップ&シンポジウム 2009 にて優秀論文賞受賞．青山学院大学大学院国際マネジメント研究科非常勤講師(2007 年～現在)．情報処理学会，人工知能学会各会員.

深田優
1986 年 上智大学理工学部機械工学科卒. 同年(株)東芝入社. コンピュータ商品技術開発, 特にオンライン高速処理技術開発を担当．米国駐在時代は北米地域のベンチャ企業 200 社を訪問し，DWH 製品を日本市場にて展開．その後，IT 関連調査会社にて IT アナリストとして活動し，(株)ボイスリサーチを設立．2005 年，神奈川サイエンスパーク主催ベンチャービジネススクールにおいてビジネスモデル論文作成最優秀賞を受賞.

2.3.3 研究チームの役割

本研究では大きく分けると要件定義プロセス関連研究　保守プロセス関連研究の２つの方向からの研究テーマがある．しかし，2.1.3 節で述べたとおり，業務セオリーという考え方をソフトウェア開発の上流から下流まで共通の考え方を検討するために，２つの方向性について全員で検討しながら研究を進めた. 2.2 節（１）研究の進め方で述べたとおり，つぎの方針で研究を進めた.

- 問題を解決するための方法を議論を通じて検討する
- 方法を確認するためのツールを試作する
- 方法を確認するための記述実験やツールの評価実験を行う
- 実験結果を分析し，方法の問題を議論を通じて明らかにする

この中で，研究責任者，研究メンバー，アルバイト学生の役割は次のとおりである.

1) 研究責任者の役割
   - 事業全体を統括し，研究の方向性を指導する.
2) 研究員の役割
   - 仕様漏れの原因・不具合の種類等の調査を行う
   - 問題事例を作成する（要求分析モデル・ソースコード）
   - 業務セオリーを検討する
   - ツールを開発する
   - ツールをテストする
3) アルバイトの役割

   アルバイトの大学院生 6 名はすべて 2.1.2 節で述べた研究および，それに関連する研究を行っている．彼らの研究の中から得られた知見をもとに，下記の作業を行い，定例ミーティングにおいて議論に参加し，研究支援を行った.
   - 例題モデルの作成ならびに変換ツールのテストを実施する

- 開発現場シナリオに基づくモデル検査適用およびツールの設計・業務セオリーの策定について他の研究員の作業を補助する.

３．研究成果

3.1 研究目標１「モデル駆動要求分析手法で定義したUML要求分析モデルからUPPAALモデルを生成する方法の確立」

3.1.1 当初の想定

(1) 想定する仮説等

UML要求分析モデルのアクティビティ図では，ユーザの要求の流れがノードとエッジで定義されている．アクティビティ図の全ノードに対し，UPPAALのロケーションを対応させ，変換を行う．この際，ノードのもつ特性が検査のキーとなる．本研究目標では文法の定義と変換ツールの実装を行う．

モデル検査は基本的に「しらみつぶし」に状態を検査することで，性質を満たさない状態を発見するものである．このため，検証したいシステムのモデルが無限の状態を取るような変換には適用できない．また，有限の状態でも複雑なシステムでは状態爆発が生じるという問題がある．状態爆発の原因は検査モデル上で検査項目に対して識別したい状態数が多すぎることであるので，UML要求分析モデルからUPPAALモデルを生成する場合には，つぎのようにモデルを縮退させる方法を検討して解決する．

UML要求分析モデルのアクティビティ図では，パーティションを用いて，ユーザの振舞い，システムの振舞い，その間のインタラクションを区別して定義することにより，ユーザの要求を整理する．例えば，システム内部のエンティティ・データのデータライフサイクルに着目する場合には，システム内部以外のUIに関わるパーティション(ユーザ，インタラクション)のモデルの変換を除外することにより，UPPAALモデルを縮退させることができる．

(2) 当初の到達目標

UML要求分析モデルと業務セオリーのモデルからUPPAALモデルと検査式を自動生成する．本研究目標ではこの文法の定義と変換ツールの実装を行う．

(3) 当初の期待される効果

この変換ツールにより，モデル検査技術の知識のない技術者にも，モデル検査手法を利用した検査を実施することができることが期待される．

3.1.2 研究プロセスと成果

(1) 研究プロセス

以下のプロセスに従い研究を行う．

① UML要求分析モデルからUPPAALモデルへの変換の文法を定義する．
② 変換を自動化するツールを実装する．
③ 事例によるテストを行う．

なお，モデル検査における状態爆発の対処方法として，パーティションによる状態の削除を想定したが，事例によるテストの結果，モデル検査において状態爆発に影響するモデルの形式は，ユースケースを接続する際の繰り返しの構造が大きく影響することが判ったため，アクティビティ図によるモデルは忠実に変換する方式をとった.

(2) 具体的な研究成果の内容

本節では，UML 要求分析モデルから，検証のための UPPAAL モデルへの変換方式と，業務セオリーからの検査式の生成方法を説明し，要求分析モデルと業務セオリーの定義から UPPAAL の知識がなくても，業務セオリーに基づく検査が実施できることを示す.

本ツール（UML2UPPAAL）の構成を図 3-1-1 に示す. UML2UPPAAL は UML 要求分析モデルを作成する astah* プラグインとして構築されている. 開発者の指示によって要求分析モデルから UPPAAL モデルを自動的に変換し，その変換結果（UPPAAL モデル）を UPPAAL に引き渡す.UPPAAL は渡された変換結果を検査し,その検査結果を UML2UPPAAL に戻す.UML2UPPAAL は検査結果を astah*上に表示する. 構成上は UML2UPPAAL と UPPAAL 検証システムは分離しているが，利用者から見た場合には UPPAAL 検証システムは隠蔽されており，利用者が直接操作する必要はない.

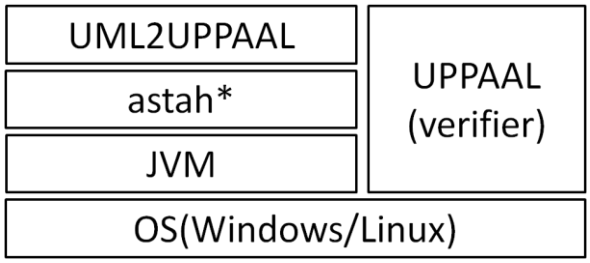


図 3-1-1　UML2UPPAAL の構成

UML2UPPAAL は次の 3 つの機能を提供する.

① CRUD 検査機能

アクティビティ図に記述されたエンティティ・データに対する操作に関して，エンティティ・データごとに定められたステートマシン図が許容しないような操作をおこなっていないことを検査する機能である.

② 通り得ないフローの検査（到達可能性）

アクティビティ図に関して，すべてのアクションへと到達する可能性があるかどうかを検査する機能である.複数のアクティビティ図間の包含関係や事前･事後条件を考慮する.

③ 不適切なアクション

モデルにおけるアクションの自然言語記述，アクションとオブジェクトノードの構造などの整合性をチェックし，問題箇所を指摘する機能である.

次に，要求分析モデルから UPPAAL モデルへの変換処理の概要を説明する. 変換は大きく

2 つのステップから構成される.

① 中間 XML 変換

要求分析モデル（astah* 形式）のうちクラス一覧とアクティビティ図に基づき，中間 XML 形式に変換する．この中間 XML 形式には，アクション記述から得られた CRUD の識別，ガードの識別，事前・事後条件の変数の識別，オブジェクトの識別が含まれる.

② UPPAAL モデル変換

要求分析モデル（astah* 形式）のうちステートマシン図に関する定義に基づき，①で識別したオブジェクトとアクションをステートマシン図と対応付け，UPPALL が解釈できるデータ形式（UPPAAL モデル）に変換する.

ツール内部で行われている変換プロセスの詳細を説明する．図 3-1-2 は変換処理の概要を表している．UML2UPPAAL では，UML モデルのうち Navigation，アクティビティ図，ステートマシン図の各モデルを入力として，これらのモデルを関連付けることによって UPPAAL モデルと検査式を自動生成する.

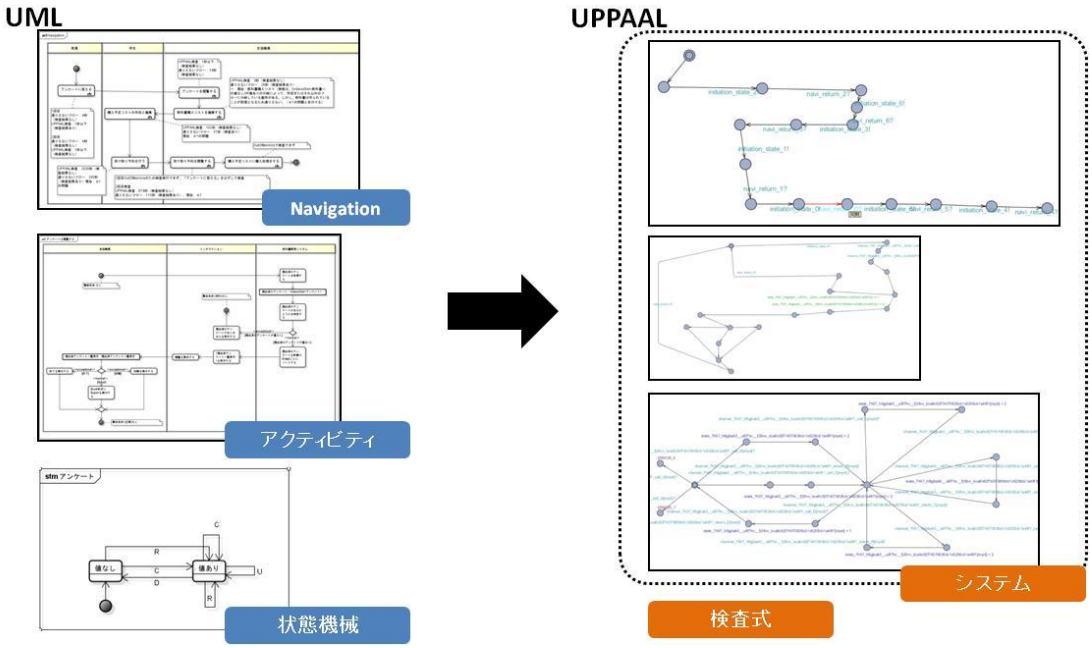


図 3-1-2　変換処理の概要

Navigation モデルと各ユースケースのアクティビティ図の同期関係の変換の一例を図 3-1-3 に示す．ここでは，UML で定義されたアクションとフローの関係を UPPAAL モデルにおける Location と Transition の組みとして定義される.

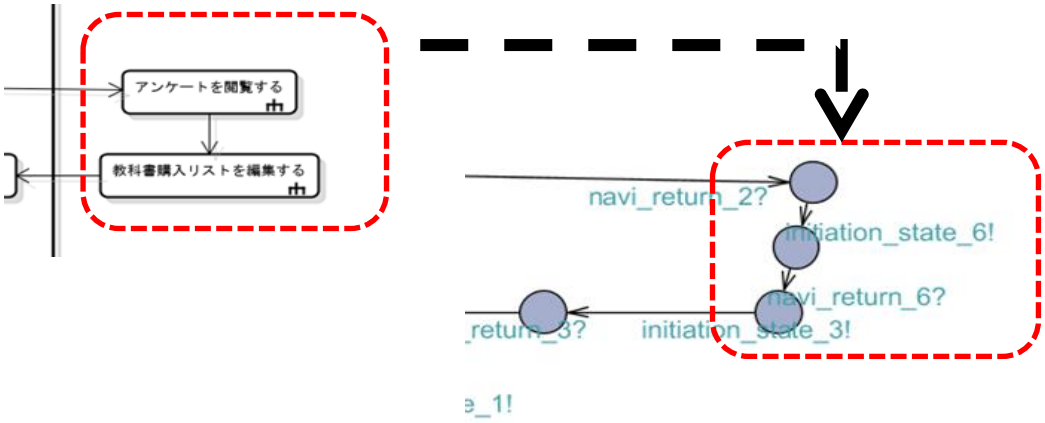


図 3-1-3　Navigation 関連の変換処理の例

次の例 (図 3-1-4) は, エンティティに対するアクションの変換処理 (「エンティティ名」を「動詞」するという形式で表現されるアクション) の例である. 左上が UML モデルのアクティビティを定義したものであり, 右下が変換された UPPAAL モデルである (channel で始まる名称によって同期チャンネルの名前を識別する). これは, アクティビティ図とステートマシン図のアクションとイベントの関係による同期関係を定義するモデルである.

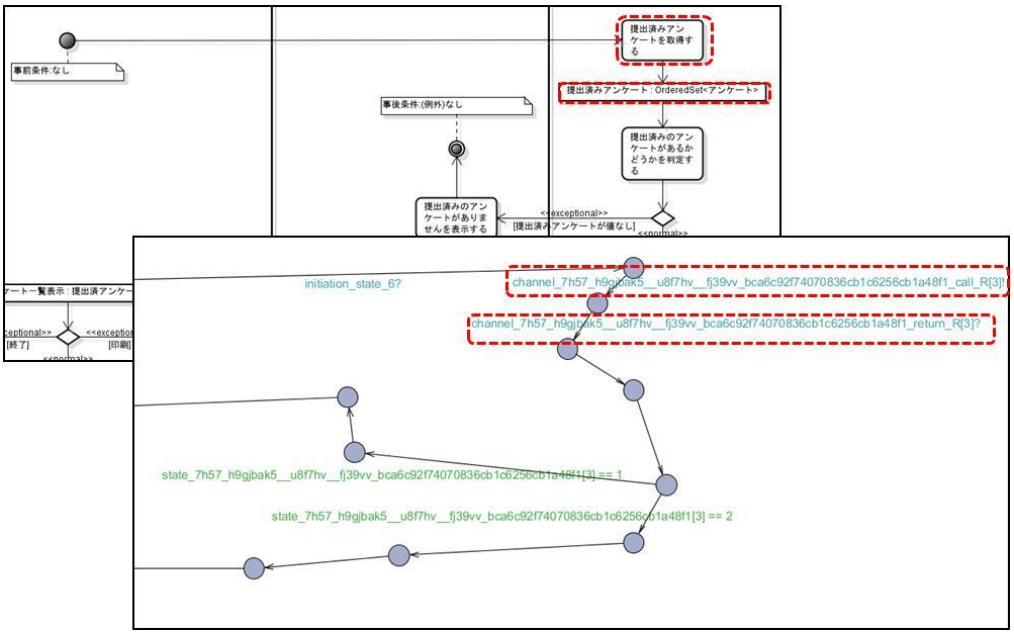


図 3-1-4　アクティビティの変換処理（アクション）の例

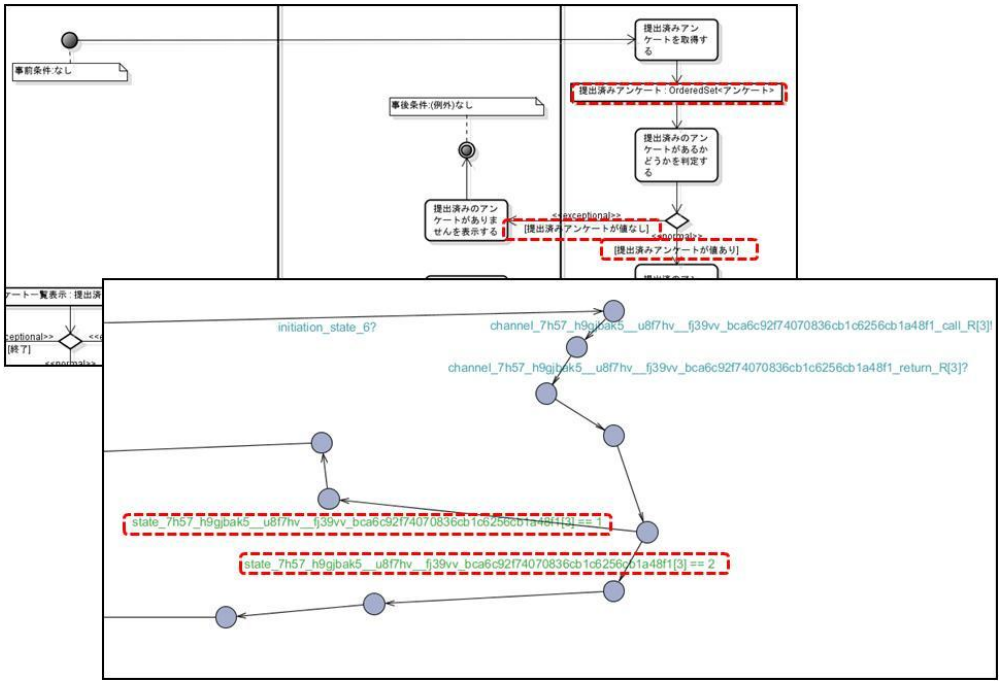


図 3-1-5　アクティビティの変換処理（ガード）の例

図 3-1-5 はガード（エンティティのインスタンス)が(状態名)という形式で表現される）の変換について説明したものである.

ステートマシン図に関連する変換の一例を図 3-1-6 に示す．ここでは同期チャンネル名はアクティビティ図での変換ルールに従っている．赤線で囲んだ部分が変換前と変換後に該当する.

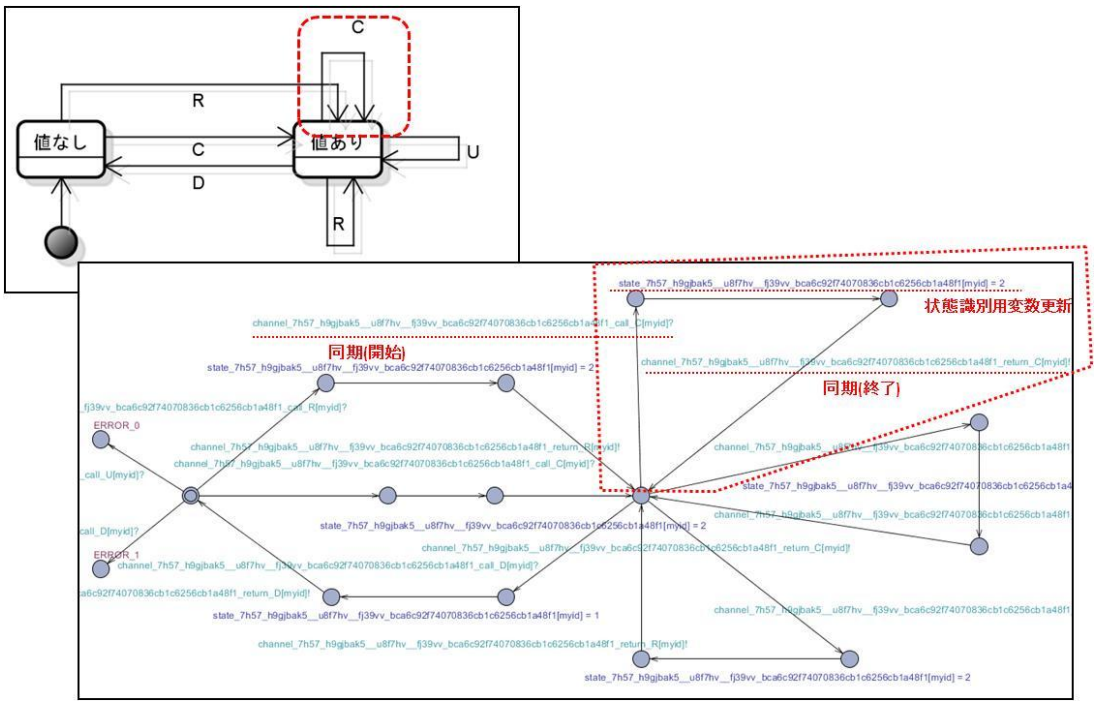


図 3-1-6　ステートマシンの変換処理の例

最後に自動生成される検査式について説明する．検査式は次の 2 つの種類の検査に関するものを自動的に作成する．

- ステートマシン図が受け付けないイベントを発火しているかどうかを検査（例：A[]！STATE_7h57_h9gjbak5__u8f7hv__fj39vv_bca6c92f74070836cb1c6256cb1a48f1(3).ERROR_0）
- アクティビティ図の中で書かれるアサーションに関するもので，あるフローを通るときに必ず到達しなければならない状態であることを検査（例：A[] error_assert_1scf_h54yjog4__7wfsa9__fj39vv_8e994a8890fc89103532d0c8c50b3abd_253t_h54yjog4__7wfsa9__fj39vv_8e994a8890fc89103532d0c8c50b3abd == false）

具体的な例を図 3-1-7 に示す．ここでは，「提出済みアンケート:アンケート[instance=3]: 値なし状態において U 操作 を行わないこと」という検査項目に対する UPPAAL モデルの例である（検査式は上記のステートマシン図が受け付けないイベントを発火しているかどうかの検査式例と同じである）．

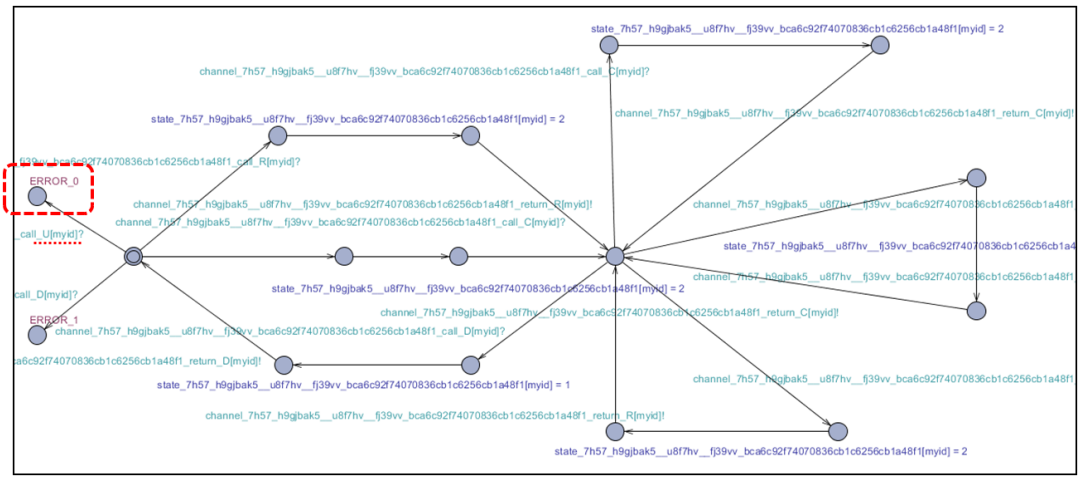


図 3-1-7　検査式

次にツールの利用例を説明する．UML2UPPAAL は図 3-1-8 の赤線で囲まれた部分であり，4 つのタブから構成される（通り得ないフロー，不適切なアクション，UPPAAL 検証，コンソール）．図 3-1-8 は，①の CRUD 検査機能を実施した時の画面表示例である．変換結果の一部を示す．右上の UML モデルに対して，検査を実行した結果が右下に表示されている．図中，緑色のハイライトが検査に成功したものを表し，赤色のハイライトが検査に失敗したことを表している．

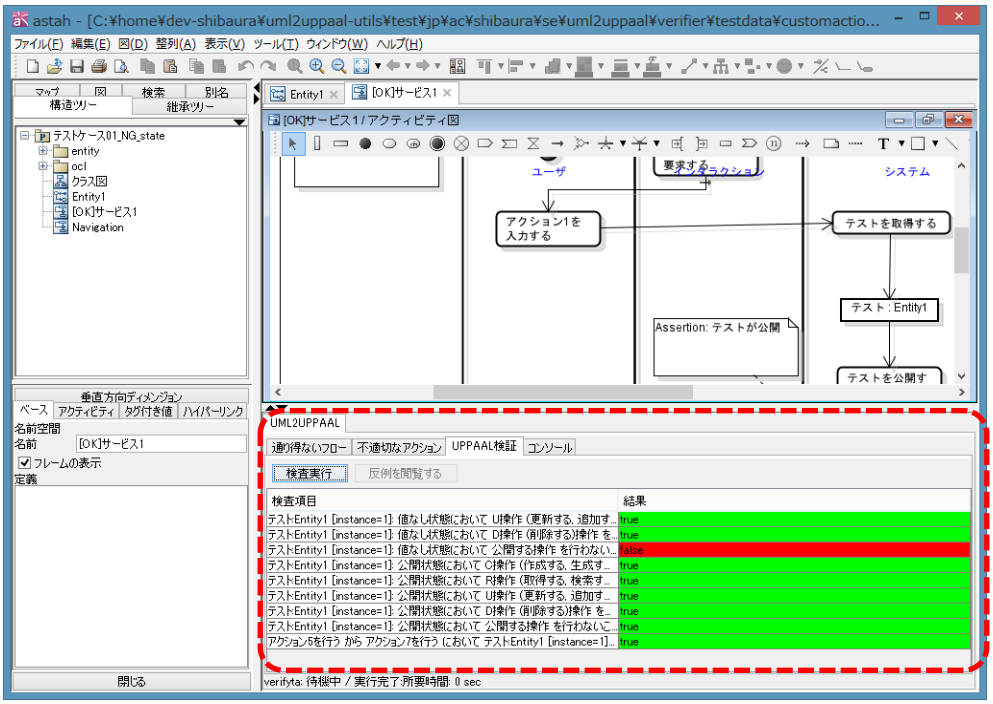


図 3-1-8　UML2UPPAAL の実行結果（その 1）

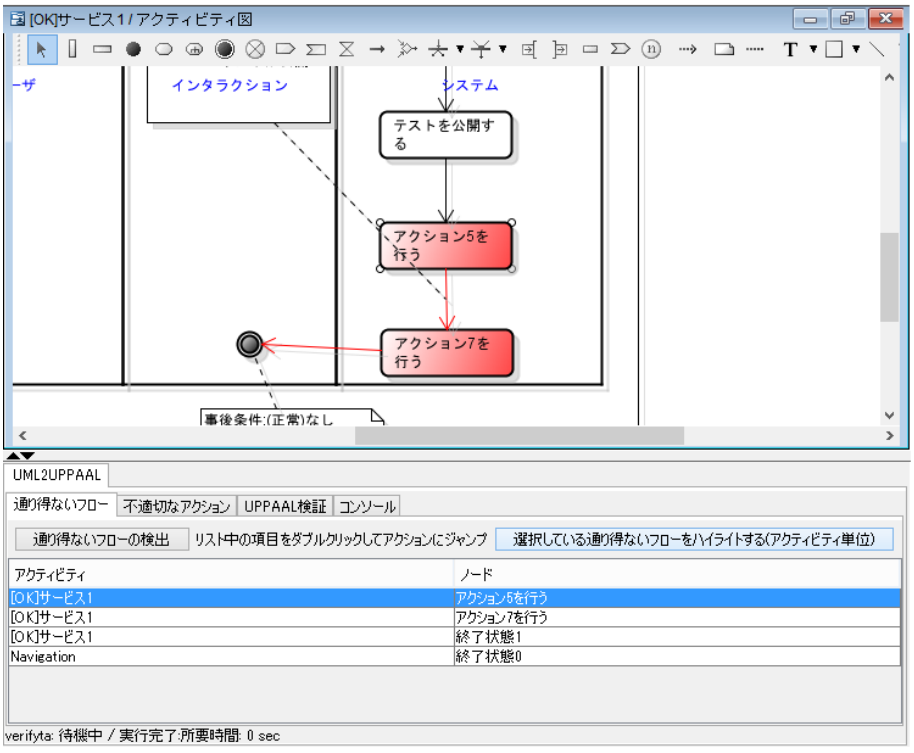


図 3-1-9　UML2UPPAAL の実行結果（その 2）

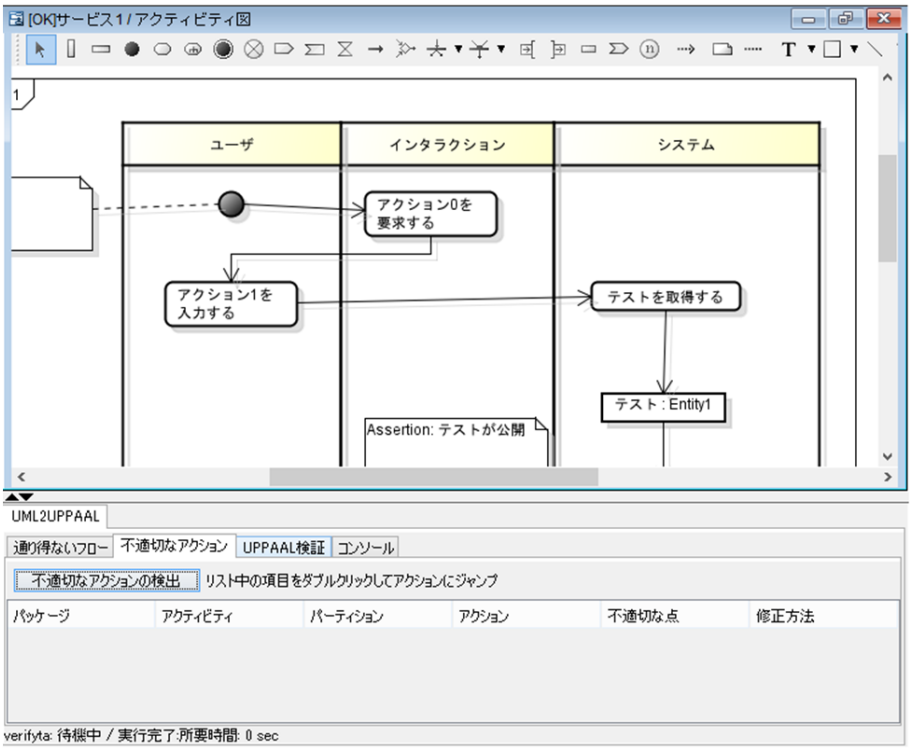


図 3-1-10　UML2UPPAAL の実行結果（その 3）

図 3-1-9 は，②通り得ないフローの検査を行った時の画面表示例である．UML モデル中のどの部分が該当するかを色で識別することができる．図 3-1-10 は，③不適切なアクションの検査を行った時の画面表示例である．

検査結果の標準出力や標準エラー出力を見る場合には，図 3-1-11 に示すように「コンソール」タブを選択する．

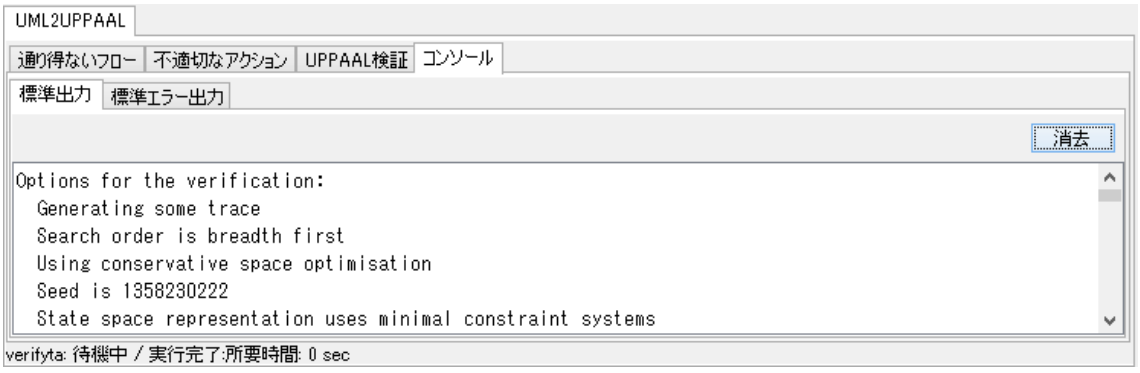


図 3-1-11　UML2UPPAAL の実行結果（その 4）

3.1.3 実用化へ向けた課題と問題点

(1) 課題と問題点

今後の課題として次の 2 点が挙げられる.

① 自然言語の曖昧性への対応

UML モデルから中間 XML 形式に変換する際に，アクション記述等の自然言語処理を行なっているが，日本語が持つ曖昧性を完全に排除できているわけではない．現時点ではモデルを定義する際に利用可能な言葉を制限しているので大きな問題にはならないが，今後この制限をはずして自由度をあげた場合に問題になると思われる.

② UPPAAL モデルの最適化

UPPAAL 検査の結果を UML モデルに対して反映する必要上,UML モデルから UPPAAL モデルに対する変換は単純な置き換えによって実装している．そのため，変換元のモデルの規模や複雑さによっては検査にかかる時間が非常に長くなってしまう場合がある．そのため，元のモデルとの対応関係を維持しつつ,モデルを最適化する方法を考えていく必要がある.

(2) 将来の応用方法

現在は，エンティティ・データの汎用的な性質のステートマシン図から，そのステートマシン図からは起こり得ないロケーションへの到達可能性により，ユースケースの特定ユーザによる特定のシナリオにおいて矛盾が生じる振舞いを検査している．Web アプリケーションの設計においては，複数ユーザが同時に同じサービスと利用するので，こうした状況における満たすべき性質への応用を検討する.

3.2 研究目標 2「データのライフサイクルおよび属性の制約に基づく業務セオリーの策定」

3.2.1 当初の想定

(1) 想定する仮説等

UML 要求分析モデルはユースケース毎にアクティビティ図を定義する．システム内で使用されるエンティティ・データへの操作を CRUD の観点から整理し，すべてのユースケース間でエンティティ･データのライフサイクルを CRUD の観点から検証する.この方法を基に，データの属性に着目して，業務セオリーの策定を行う.

(2) 当初の到達目標

UML 要求分析モデルに対し，UPPAAL モデルと検査式を自動生成できるように，業務セオリーモデルの定義方法を検討する．本研究目標ではデータの属性に着目して，業務セオリーの策定を行う.

(3) 当初の期待される効果

この定義方法により，モデル検査技術の知識のない技術者にも，モデル検査手法を利用した検査を実施することができるようになることが期待される．

3.2.2 研究プロセスと成果

(1) 研究プロセス

以下のプロセスに従い研究を行う．

① 仕様漏れの原因を文献により調査し，業務セオリーを検討する．
② 一般的な CRUD に関する業務セオリーを定義する．
③ クラス間の関連および属性情報から業務セオリーを定義する．
④ 変換ツールにより，事例を用いて UPPAAL での検証を行い，業務セオリーの妥当性をテストする．

(2) 具体的な研究成果の内容

① 仕様漏れの原因の調査

表 3-2-1 の文献により，仕様漏れの原因に関する調査を行った．結果を表 3-2-2 にまとめる．

表 3-2-1　調査対象文献の一覧

No	資料名	入手元	著者	所属
01	成果物からみた要求定義	CINI	鎌田真由美	日本IBM
02	サービス・プロジェクトの成功条件を探る	CINI	板倉真由美	日本IBM
03	要求仕様書品質とプロジェクト成否の関連	CINI	鎌田真由美	日本IBM
04	要求定義フェーズの終了判定予測	CINI	渡辺千恵子	日本IBM
05	Customer requirements and user requirements: why the discrepancies?	IEEE		
06	Incremental Maintenance of Software Artifacts	IEEE		
07	Hipikat: a project memory for software development	IEEE		
08	Toward understanding the rhetoric of small source code changes	IEEE		
09	Visualizing Co-Change Information with the Evolution Radar	IEEE		
10	大規模並行開発での運用を考慮した成果物間整合保証方式の提案(開発プロセス)	CINI	野尻 周平	日立製作所システム開発研究所

表 3-2-2　仕様漏れの原因

No	仕様漏れの原因	備考
01	情報量が多すぎてまとめきれず抜け落ちる情報がある	No.01より
02	システム構築の目的が曖昧なため情報の取捨選択を誤る	No.02より
03	仕様記述の標準化が徹底されていないために発生する仕様のレベル不整合	No.03より
04	要件定義を終了する基準があいまいであるため、重要な情報を取得できないまま感覚的な判断で終了	No.04より
05	要求の言い忘れ、伝達漏れ、暗黙的な要求(業界の常識)	No.04より
06	要求分析者が顧客要求をそのままドキュメント化してしまう(問題点が明確にならない)	No.04より
07	顧客とユーザでの要求が違う場合(真の要求がわからない)	No.05より
08	ドキュメント(コーディング規約等)は更新されるがソースコードは更新されないまたはその逆の場合、(結果的に仕様漏れになる)	No.06より
09	成果物が体系化されていないと、経験が少ない要員は成果物の中から適切な情報を取り出せない	No.07より 仕様書にはあるがソースへの反映で漏れるような場合

われわれの提案するモデル駆動要求分析手法では，要求分析モデルからのプロトタイプ生成により，顧客とプロトタイプを通じて主に機能要求に関する要求の妥当性を確認し，05 ，07 の問題に対処する．また，要求分析モデルから，既存のフレームワークを用いた Web アプリケーション開発における設計モデルの定義方法[20],により，08 および 09 に対して，要求分析モデルから実装までのトレーサビリティを研究している．要求仕様の実現可能性の観点からは，W モデル開発[21]を実現するシミュレーションによるテスト設計手法[22]により，要求仕様の実現可能性の観点からの精査を行っている．これにより 01 の問題や 04 の問題への１つの解決策となると考える.テスト設計は詳細な具体データを用いて検査することで，具体的な検査が可能であり，要求仕様の理解にも貢献するが，全てのステップに対して，逐一テストケースを想定しなければならない．要求分析モデルは，個々のユースケースの観点から，要求を定義し，それを統合したサービスが，顧客の要求するものと合致するように定義する．この際，開発者はサービスの手順を定義しているが，サービスに必要なデータに不足がないか，それらの扱いに不整合がないかを確認することには労力を要する．さらにセキュリティ要求のような非機能要求を機能要求と同時に検討することは，仕様の複雑さを招き，01 や 03 のような仕様漏れの大きな要因でもある．

② CRUD に関する業務セオリー

業務システムのサービスの妥当性が確認された入力データから出力データ生成の実現可能性を保証するためには，次の 2 つの定義を行う必要がある．

1)　実現するに足るシステム内部データの抽出と定義

2)　入力データから出力データを生成する振る舞い系列の定義

入力データから適切なフローにより出力データを得る過程はプロトタイプからは見えないため，開発者がその生成フローを確認しながら定義する必要がある．この時，妥当性の確認された入出力データが 1）および 2）の定義の１つの指標となる．しかし，抽出したシステム内部データがシステム全体のライフサイクルにおいて妥当であるかを保証することは開発者の負荷が大きい．モデル検査技術はシステムの振舞いを状態遷移としてモデル化し，その満たすべき性質を論理式により定義することで，網羅的な検証が可能であるが，

満たすべき性質の検査式を定義するには，アクティビティ図のアクション記述形式が定まっていないため，そのままでは検査式を定義できない．

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，期待される出力へと変換する．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である．すなわち，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 に関する普遍的な性質を満たすことが必要である．サービスを全てのユーザに誤りなく提供するために，システム内部の永続化対象データはその基本機能である CRUD に関する普遍的な性質を満たすことを検査する．そこで，2）で定義する振舞いを CRUD 機能とその対象データを整理することにより，開発者が各アクティビティ図で定義した 1） のデータ・ライフサイクルが要求分析モデルのすべての経路において満たされることをモデル検査により検証する．

アクションに記述する動詞を CRUD 機能と対応付けて表 3-2-3 のように定義する．これらの動詞は，システム・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図 3-2-1 のように解釈する事で，振舞いにおける CRUD 機能の呼出しがアクティビティ図に定義される．

表 3-2-3　CRUD に対応する振舞いを表す動詞

	Create (作成)	Read (検索)	Update (更新)	Delete (削除)
単数	を作成する を生成する	を検索する を取得する	を更新する を変更する	を削除する
複数(※)		を検索する を取得する	を追加する を登録する	を削除する

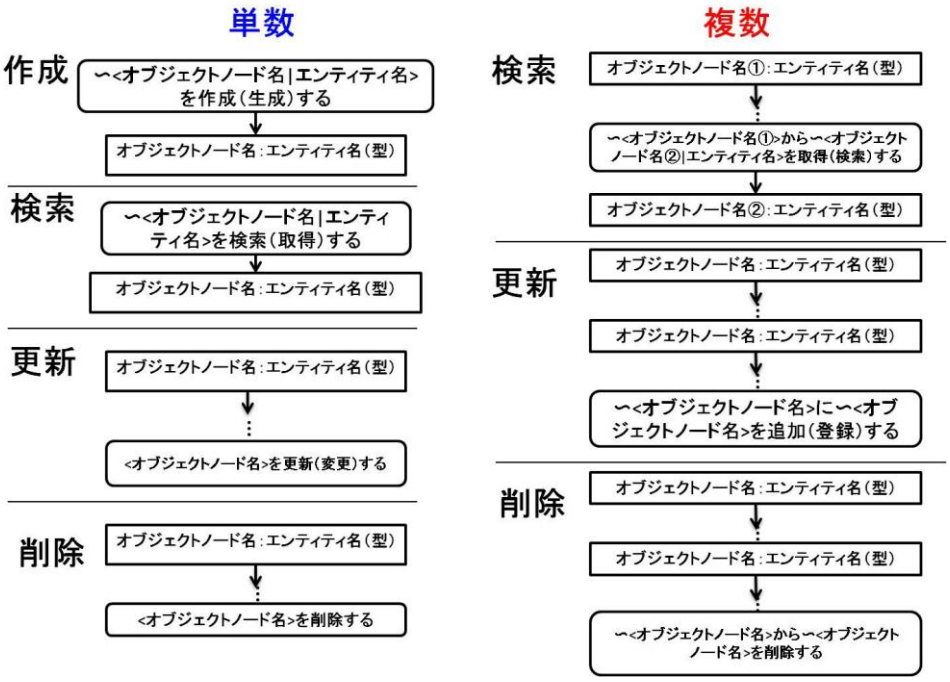


図 3-2-1　振舞いの目的語とオブジェクトノードの関係

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 3-2-2 のようにステートマシン図を用いて定義する．

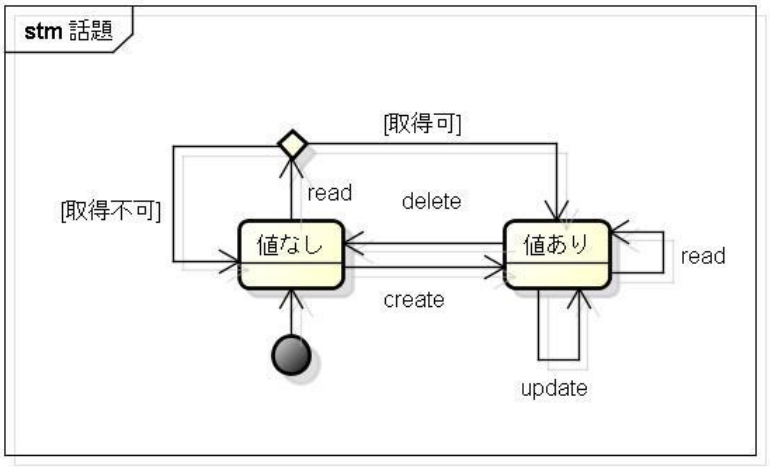


図 3-2-2　オブジェクトの CRUD に関する業務セオリー

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対応づくことである．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できる．表 3-2-4 と表 3-2-5 はこれらの関係を表している．

表 3-2-4　アクションとイベントの関係

アクティビティ図のアクション記述	**オブジェクト**のステートマシン図イベント
オブジェクトを生成する	Create
オブジェクトを取得する	Read
オブジェクトを更新する	Update
オブジェクトを削除する	Delete

表 3-2-5　ガードとステートの関係

アクティビティ図のガード	**オブジェクト**のステートマシン図のステート
オブジェクトが値なし	値なし
オブジェクトが値あり	値あり

この関係式は一般化でき，一般的には表 3-2-6 の通り(n≧2)である．

表 3-2-6　一般化したガードとステートの関係

アクティビティ図のガード	**オブジェクト**のステートマシン図のステート
オブジェクトがステート n	ステート n

表 3-2-4 のアクションとイベントの関係も対象となる動詞に関する規則があればよいが，目的語も含めて検討する必要がある．

図 3-2-2 から読み取れる CRUD の基本的な性質は次の通りである．

1) Create もしくは Read によりオブジェクトが値を持っている状態「値あり」に変化する．
2) Read により値が取得できる場合とできない場合がある．
3) Delete によりオブジェクトが値を持っていない状態「値なし」に変化する
4) オブジェクトが Update または Delete されるならば，Create もしくは Read によりオブジェクトが予め値を持っている，すなわち「値あり」の状態でのみ適用できる．

Read の場合，単数のオブジェクト自身を取得したい場合と，複数のオブジェクトの中から特定したオブジェクトを得たい場合がある．どちらの場合も，オブジェクトが取得できないケースがあり得る．これは一般に業務ルールとしても，例外を明示していないという，見落としがちなケースに相当する．

しかし，すべてのクラスが対象システムにおいて，図 3-2-2 の状態遷移を行うわけではない．対象システムでは，そのクラスのデータは常に存在することが仮定できる場合がある．このような場合には，「値なし」の状態はなく，Create しなくても値は存在する．こ

の時は図 3-2-2 のステートマシン図から状態および遷移を削除して，当該クラスの業務セオリーを定義する．

CRUD に関する業務セオリーは，対象システムに登場するすべてのエンティティ・クラスに対して，図 3-2-2 の基本形を基に定義する必要がある．アクティビティ図には，このクラスにより分類されるオブジェクトノードが複数登場する．しかし，これらはこのアクティビティ図上では名前で識別される個々のインスタンスであり，そのライフサイクルは別々のものであるべきである．一方，2.1.3 節でも述べたとおり，オブジェクトノードの識別子は，仕様として簡潔に読みやすく表現することが必要である．そこで，名前が同じであることに加えて，名前の異なる同じクラスのオブジェクトノードが同等であるかを定義し，そのライフサイクルを共有できるようにする必要がある．本研究では astah*のハイパーリンク機能を利用して，これを実現する．

③ セキュリティ要件に関する業務セオリー

セキュリティ要件は，対象アプリケーションに付随して定義される．例えば，2.1.2 節で述べた LMS の拡張機能である BBS の開発の例を考える．このシステムでは，教員と学生の一連の質疑応答を表す「話題」というオブジェクトが登場する．質疑応答は，未設定の状態から，初期値が非公開状態として，生成され，教員の判断で公開され，全学生が閲覧できるようになる．BBS のクラス「話題」は，これを扱うユースケースに対して，属性「公開／非公開」の値が適切に変更されなければならない．こうしたセキュリティ要件における対象クラスのもつセキュリティ属性の取り得る状態とその遷移イベントを業務セオリーとして定義する．図 3-2-3 は「話題」クラスのセキュリティ属性「公開／非公開」の取りえる値「未設定」「公開」「非公開」をステートとし，これらのステート間の遷移イベントを定義している．

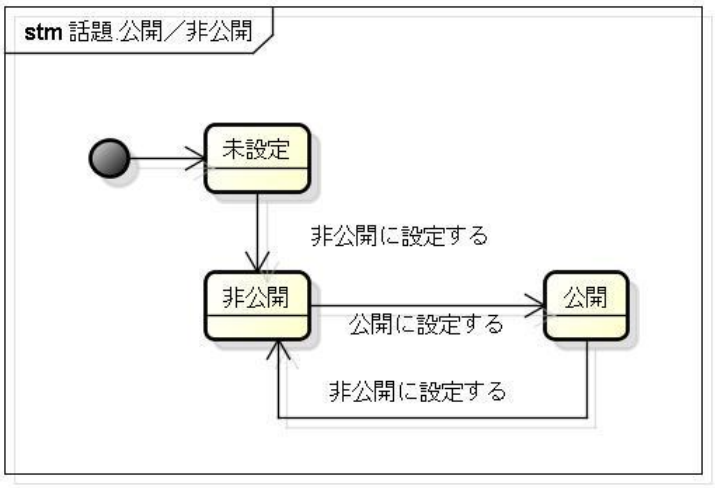


図 3-2-3　オブジェクトのセキュリティ属性に関する業務セオリー

この場合のアクティビティ図上のガードとステートの関係は表 3-2-7 のようになる．また，アクションとステートマシン図のイベントの関係は表 3-2-8 のように一般化される．

ステートマシン図の名前をクラスの属性を指定することで，アクティビティ図に登場するオブジェクトノードに対する属性のライフサイクルが定義される．

表 3-2-7　ガードとステートの関係

アクティビティ図のガード	**話題.公開／非公開**のステートマシン図のステート
話題.公開／非公開が公開	公開
話題.公開／非公開が非公開	非公開
話題.公開／非公開が未設定	未設定

表 3-2-8　一般的なアクションとイベントの関係

アクティビティ図アクション	**オブジェクト.属性**のステートマシン図イベント
オブジェクト.属性をステートに変更する	**オブジェクト.属性をステートに変更する**

図 3-2-4 および図 3-2-5 を用いて，セキュリティ要件に関する検査方法を説明する．

表 3-2-9　アクセス制御方針

サブジェクト		オブジェクト		操作		ルール		
名前	セキュリティ属性	名前	セキュリティ属性	アクティビティ図	アクション	FDP_ACF.1 セキュリティ属性によるアクセス制御	FMT_MSA.3 静的属性初期化	FMT_MSA.1 セキュリティ属性の管理
学生	役割(学籍番号)	BBS		話題を選択する(学生)	現在のBBSを取得する			
				質問を投稿する	BBSに新規話題履歴を追加する			
		コンテンツ		質問を投稿する	質問のコンテンツを生成する			
		話題履歴		話題を選択する(学生)	〈現在のBBS〉から話題を取り出す			
				話題を閲覧する(学生)	〈話題履歴一覧〉から〈話題番号〉の話題履歴を取り出す			
				質問を投稿する	話題履歴を生成する			
		話題	公開/非公開	質問を投稿する	話題を生成する		公開/非公開=非公開で生成される	
		日時		質問を投稿する	現在日時を取得する			
		投稿者	役割	話題を選択する(学生)	現在の利用者を取得する			
				質問を投稿する	投稿者を取得する			
		投稿内容		話題を閲覧する(学生)	投稿内容を取得する			
		添付ファイル	公開/非公開	話題を閲覧する(学生)	添付ファイルをダウンロードする	公開非公開=公開 ‖ 投稿者.役割=サブジェクト.役割である場合実行できる		
				質問を投稿する	添付ファイルを登録する		公開/非公開=非公開で生成される	
教員	役割(教員)	BBS		話題を選択する(教員)	現在のBBSを取得する			
				質問に回答する	BBSを更新する			
		コンテンツ		質問に回答する	回答を作成する			
		話題履歴		話題を選択する(教員)	〈現在のBBS〉から話題を取り出す			
				話題を閲覧する(教員)	〈話題履歴一覧〉から〈話題番号〉の話題履歴を取り出す			
				質問に回答する	話題履歴を更新する			
				質問に回答する	〈話題履歴〉から〈質問番号〉で話題を取得する			
		話題	公開/非公開	質問に回答する	〈質問番号〉により選択された話題を取得する			
				質問に回答する	回答を追加して話題を更新する			
					公開/非公開を公開に変更する			公開/非公開=公開に変更する
					公開/非公開を非公開に変更する			公開非公開=非公開に変更する
		日時		質問に回答する	現在日時を取得する			
		投稿者	役割	話題を選択する(教員)	現在の利用者を取得する			
				質問に回答する	投稿者を取得する			
		投稿内容		話題を閲覧する(教員)	投稿内容を取得する			
		添付ファイル	公開/非公開	話題を閲覧する(教員)	添付ファイルをダウンロードする	公開/非公開の値に関わらず実行できる		
				質問に回答する	添付ファイルを登録する		公開/非公開=非公開 ‖ 公開/非公開=公開で生成される	
					公開/非公開を公開に変更する			公開/非公開=公開に変更する
					公開/非公開を非公開に変更する			公開非公開=非公開に変更する

図 3-2-4 では「質問に回答する」というアクティビティ図のフローである．ここでは「話題」に対して，公開／非公開を設定するべきであることが，表 3-2-9 に示すコモンクライテリアのアクセス制御方針より，わかっている．しかし，このフローには，公開／非公開属性の変更と同期するアクションが存在しないため，「更新された話題」が登場した後の遷

移に記述された Assertion「更新された話題の公開／非公開が非公開」は満たされないことが検査される.

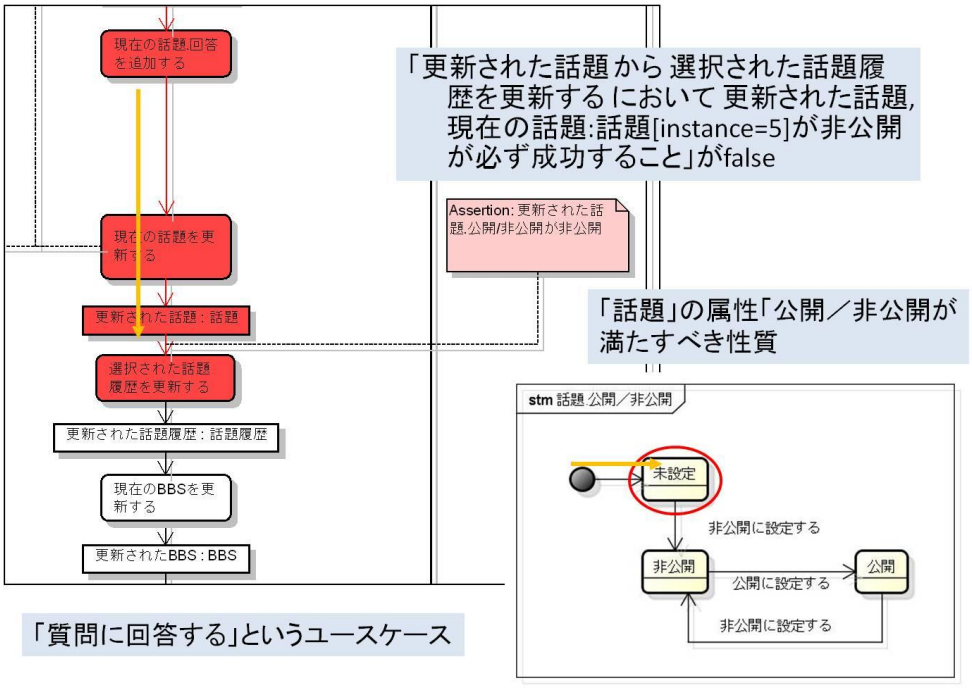


図 3-2-4　セキュリティ属性に関する検査（その 1）

一方，図 3-2-5 ではこのフローに赤丸で示された公開／非公開属性の変更と同期するアクションが存在するため，「話題.公開／非公開」の状態が「非公開」に遷移し，「更新された話題」が登場した後の遷移に記述された Assertion「更新された話題の公開／非公開が非公開」は満たされることが検査される.

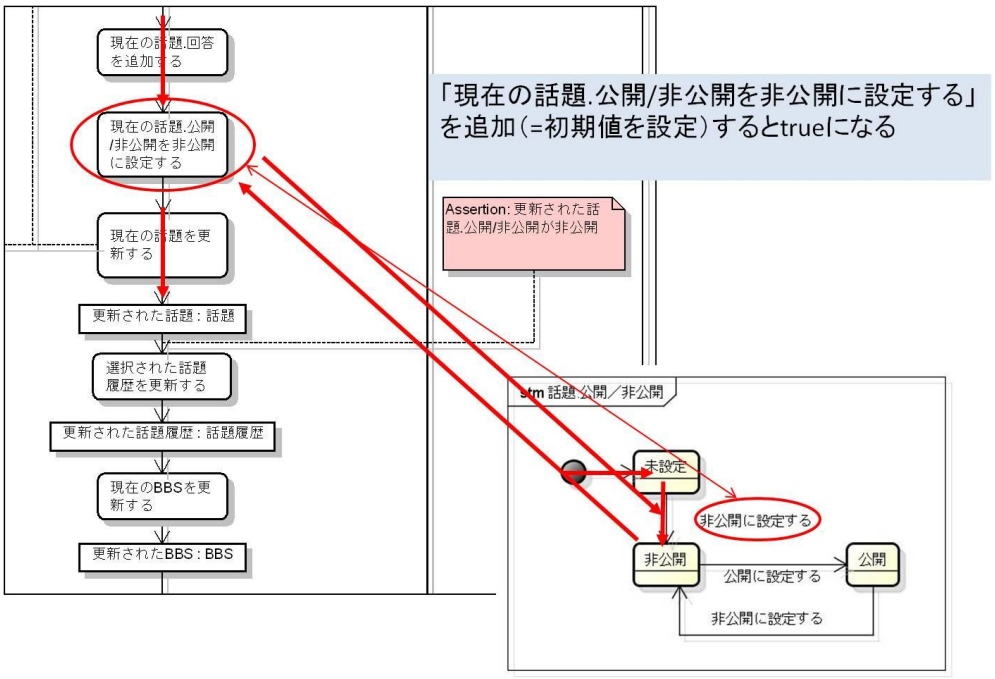


図 3-2-5　セキュリティ属性に関する検査（その 2）

④ 検証実験概要

検査のプロセスは図 3-2-6 に示す通りである．要求分析モデルと業務セオリーは 3.1 節で述べた方法で，UPPAAL モデルに変換される．この時，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，およ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 の同期関数で定義される．検査式もステートマシン図または上記の Assertion により，自動で生成できる．これらのモデルを UPPAAL 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表示し，このフローと検査結果から問題点を発見する．

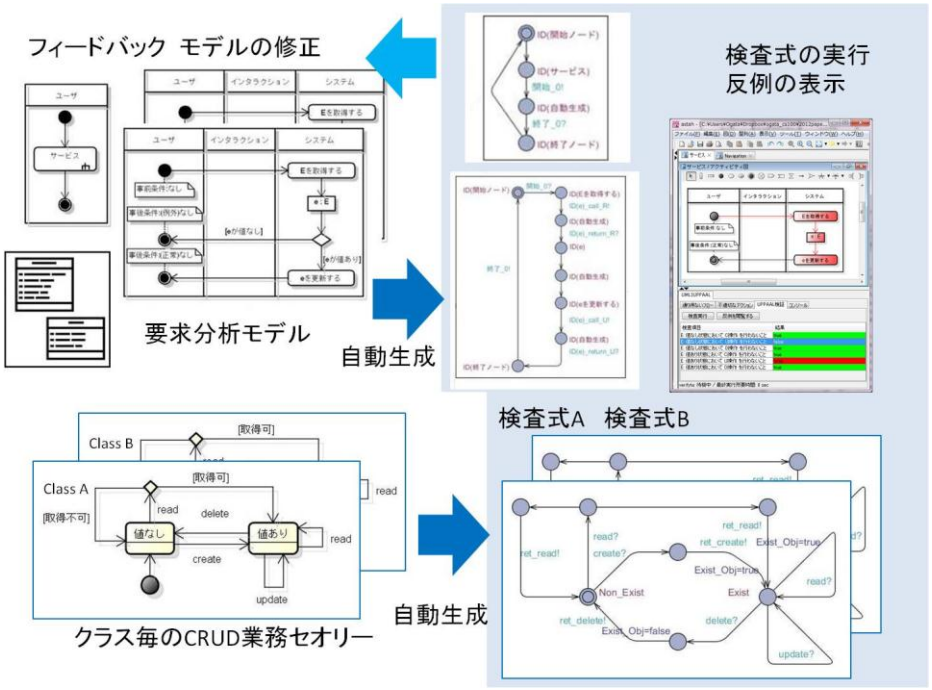


図 3-2-6　モデル検査手順

下記の 5 つの事例を用いて，モデル検査実験を実施した．

① 大学院授業課題：大学生協教科書販売システム
② 授業で用いるグループワーク支援システム GWSS
③ 大学で運用している LMS　LUMINOUS の拡張機能(BBS)
④ 研究室内図書管理システム（2009 年度版）
⑤ 研究室内図書管理システム（2011 年度版）

実験の手順は次のとおりである(図 3-2-7 参照)．

(1) アクションの振る舞いとオブジェクトノードを対応付ける為の CRUD 記述を要求分析モデルに導入する．
(2) CRUD 記述の導入によって CRUD テーブルがアクティビティ図より自動生成できる．その CRUD テーブルを用いて要求分析モデル上の CRUD 記述が開発者の意図及び顧客の意図が適切に反映しているかを確認し，それらの意図に合致するまで要求分析モデルの修正を行う．
(3) CRUD テーブルのエンティティの振る舞いに基づいて，エンティティ毎の業務セオリーをステートマシン図により定義する．
(4) シナリオの観点からサービスフローを Navigation モデルとして定義する．このサービスフローはプロトタイプ生成時に用いられるユーザ遷移可能なサービスをユーザの意図としてつなぎ合わせたものを表す．
(5) UML2UPPAAL を用いて業務セオリーの検査をデータの妥当性違反が無くなるまで行う．

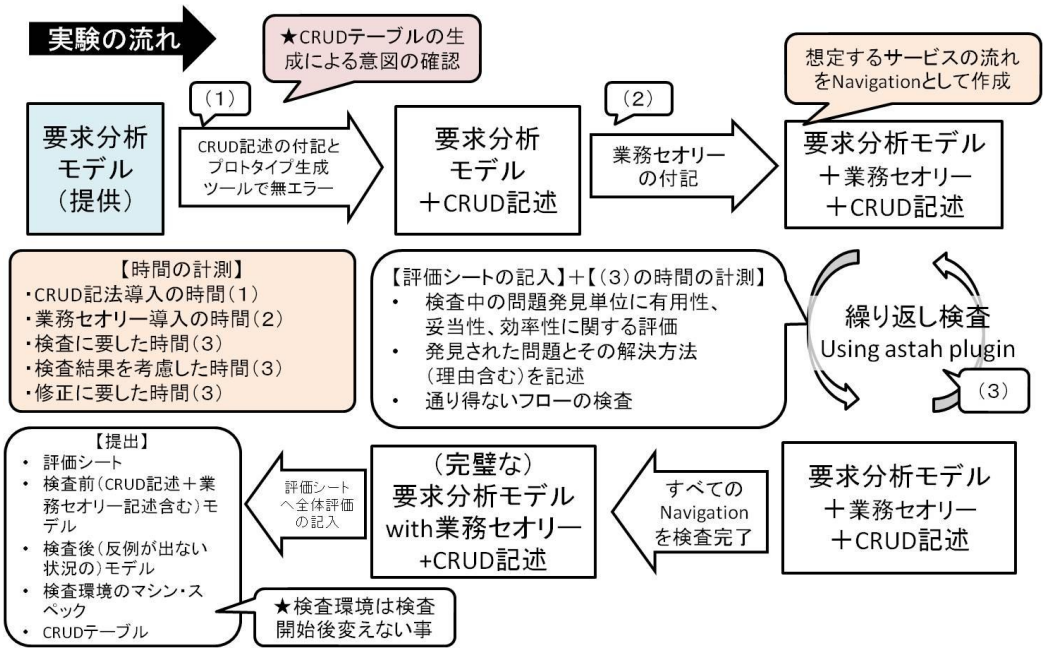


図 3-2-7　実験のプロセス

各事例の概要はつぎのとおりである．

① 大学生協教科書販売システム：大学生協の混雑緩和と購入手続きの迅速化を目指した教科書購入の支援システムである．システムのシナリオは，教員が購入させたい教科書をアンケートとして登録し，それに基づき生協職員が業者への発注し，学生は購入したい教科書を登録し，生協職員は生協窓口での購入処理を行うというものである．

② 授業で用いるグループワーク支援システム GWSS：学部 3 年生対象のソフトウェア開発演を行うグループワーク支援のシステムである．主に学生間，教員−学生間のコミュニケーションの為の掲示板，学生の作業報告や議事録が記述できるシステムである．

③ 大学で運用している LMS　LUMINOUS の拡張機能(BBS)：授業支援システムの拡張機能で，教員と学生間の記事を学習目的に合わせて非公開・公開を選択できる事など学習支援を特徴とする掲示板システムである．

④ 研究室内図書管理システム（2009 年度版，1 名）

⑤ 研究室内図書管理システム（2011 年度版，2 名）

④及び⑤は，研究室内の図書を効率的に検索・整理する事を目的とした図書管理システムである．

ここで，②および③は，実装を行い，実稼働しているモデルである．

また，それぞれのモデル規模を，クラス数，属性数，アクティビティ図（ユースケース）の数，アクションノードの数，アクティビティ図のサイクロマチック数の平均，アクティビティ図のフローとアクション数の平均，アクティビティ図の要素数の平均を用いて，表 3-2-10 に示す．

表 3-2-10　事例の規模

モデル名	①	②	③	④	⑤
クラス数	110	162	58	33	45
属性数	387	157	112	91	125
ユースケース数	11	13	9	6	7
アクション数	390	315	183	119	138
サイクロマチック数の平均	22.9	28.2	14.9	15	12.3
フローとアクション数の平均	106.5	85.7	56.1	64.3	58
要素数の平均	65.5	60.5	39.5	43.5	39.57

⑥　実験結果の分析

実験の結果，表 3-2-11 および表 3-2-12 に示す問題点が計 83 個発見できた．

表 3-2-11　発見された問題（その 1）

名称	説明	発見数
分岐判定記述漏れ	業務セオリー上でReadの値あり／なしを非決定に決定する分岐が存在する場合に、それに対応したReadの値あり／なし分岐を記述していない問題	10
分岐判定記述漏れによって発見されたモデルの問題	値あり／なしによる分岐判定を考慮しない事によって生じた、値なし状態でのUpdate／Deleteに関する問題	2
値ありフローの中でのオブジェクトのCreate問題	Createは値なし状態において作成させるべきで有り、値あり状態でのCreateでは、サービス期間中にオブジェクトを作成されない問題	1

表 3-2-12　発見された問題（その 2）

名称	説明	発見数
クラス間の関連・属性が解釈されない問題	クラス間の関連と属性を対象とする解釈機構が存在しない為、その意図を正しくモデルに反映できない問題	4
値なしが取り得ないRead	Readの結果が必ず値ありになる場合に対する問題。業務セオリーの定義はエンティティ単位にしか行えないが、オブジェクト単位に定義したい場合がある。	7

以下に業務セオリーを用いた検査によって発見できた問題について説明する．

まず，教科書販売システムでは，「値ありフローの中でのオブジェクトの Create 問題」に関する問題が発見された．教科書販売システムでは，学生が生協の販売する教科書を必要な物を選んで購入する為に購入予定リストを作成するというユースケースがある．その中で,既に学生があるか無いかによって図 3-2-8 のようにフローを制御している．しかし，図 3-2-9 のように，学生の値ありのフローの中で学生の作成を行っているため，このサービスでは永久に学生が作成できないという状態に陥っている．ここで，「XX さん」という学生を表すオブジェクトノードと「入力内容に一致する学生」は，このユースケースでは購入リストを作成する学生を表すことから，同一視している．これにより，提案手法によってこの問題を発見する事ができた．

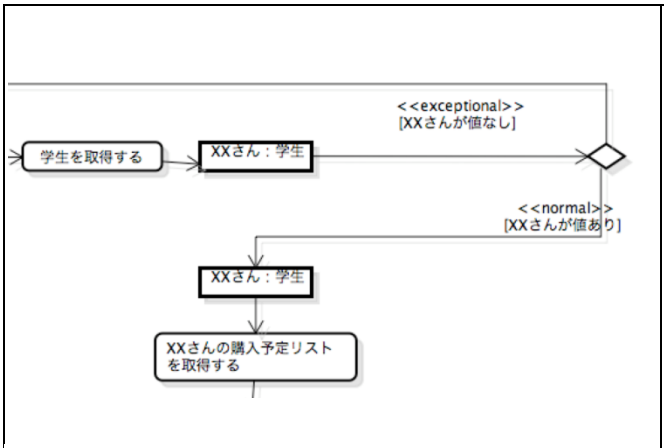


図 3-2-8　学生の値あり・なし分岐

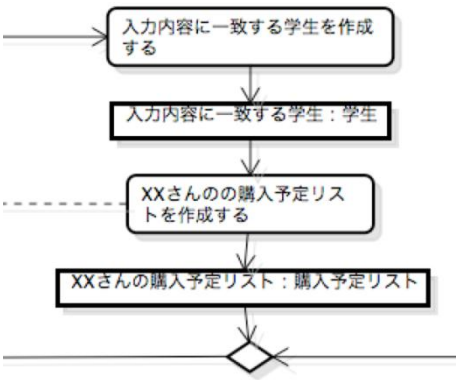


図 3-2-9　学生オブジェクトの作成フロー

「分岐判定記述漏れ」に関する問題は，全てのモデルにおいて合計 10 件が発見された．Read の結果によっては，その値の有無により，正常ケース及び例外ケースを考慮する必要がある箇所を発見する事ができた．例えば，教科書販売システムでは，教科書の受け取り業務を高速化する為，「受け取り予約」サービスを任意で利用する事ができる．その為，実際に受け渡し処理を行う時に，「受け取り予約」が無い人とある人の両方が存在する．しかし，図 3-2-10 に示すように，受け取り予約を持ってない学生に対する代替フローが記述されていない問題を発見する事ができた．

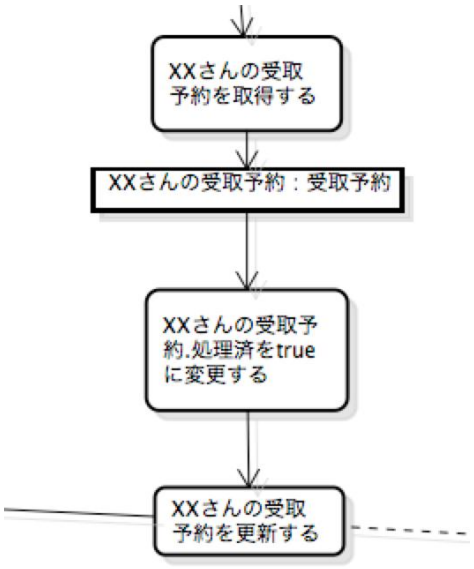


図 3-2-10　受け取り予約の取得フロー

更に，この「分岐判定記述漏れ」によって例外フローが考慮されない事で，LMS　LUMINOUS の拡張機能(BBS)では，値なし状態における更新が存在するフローを発見（「分岐判定記述漏れによって発見されたモデルの問題」）する事ができた．

LMS 拡張機能の BBS では，システムのユーザである学生は質問を新規投稿するか，投稿されている質問に追加投稿するかを選択できる．そのため，追加投稿できる質問がない場合は新規投稿のみ可能であるべきである．しかし，質問が既にあるかどうかを判定するフローが無いため質問が存在しなくても，図 3-2-11 のように，追加投稿を選択することができるので，図 3-2-12 のように，学生の追加質問のフローにおいて値なし状態における更新が存在していることを発見することができた．

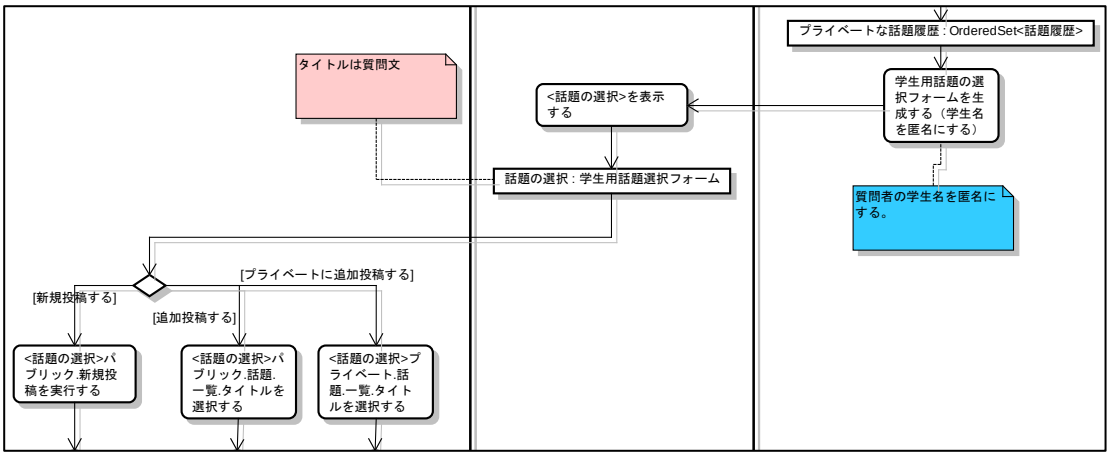


図 3-2-11　新規投稿・追加投稿を選択するフロー

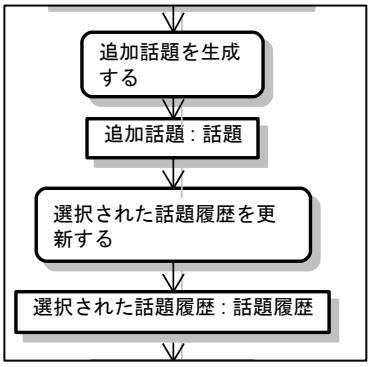


図 3-2-12　追加投稿を更新するフロー

3.2.3 実用化へ向けた課題と問題点

(1) 課題と問題点

要求分析モデルは，モデル駆動開発手法において，プロトタイプを生成することで，顧客との要求の妥当性確認を行うことが可能である．要求をまず，システムの外側から確認する上で有効な手段であり，実際の開発における要求仕様としての役割を果たしている．しかし，最終形態を想定して，ユースケースをつなげるため，モデルが複雑になる傾向がある．記述の単位を検討する必要がある．

現在の CRUD に関する業務セオリーは，基本的な雛型はあるが，対象アプリケーションのエンティティ・クラスに対して全て定義しなければならない．ステートマシン図の作成支援は可能であることから，整理と管理方法を検討する．
セキュリティ要件については，コモンクライテリアから導かれるセキュリティ機能方針を表により作成し，これによって，検査式を整理する方法を検討する．

(2) 将来の応用方法

われわれは要求分析モデルから実装までのトレーサビリティを研究しているが，システムに対する要求はソフトウェアアーキテクチャに関する要件もある．要求分析モデルと，既存のフレームワークとの適合性の検査への拡張を検討する．

3.3 研究目標 3「ソースコードからその制御フローを表す UPPAAL モデルを生成する方法の確立」

3.3.1 当初の想定

(1) 想定する仮説等

モデル検査は基本的に「しらみつぶし」に状態を検査することで，性質を満たさない状態を発見するものである．このため，検証したいシステムのモデルが無限の状態を取るような変換には適用できない．また，有限の状態でも複雑なシステムでは状態爆発が生じるという問題がある．状態爆発の原因は検査モデル上で検査項目に対して識別したい状態数が多すぎることであるので，ソースコードから制御フローを表す UPPAAL モデルを生成する場合に，つぎのようにモデルを縮退させる方法を検討して解決する．複雑度を表すソフトウェアメトリクス(サイクロマチック)を利用して状態爆発のしやすさを測る状態爆発指数を考案し，この状態爆発指数に基づきソースコードを分割する．さらに，業務セオリーに関連する記述を基にソースコードのスライシングを行い，モデル抽象化し，UPPAAL モデルを縮退させる．

(2) 当初の到達目標

2.2.2(2)で述べた「モデル検査ツールの１つである UPPAAL を用いたソースコードの欠陥抽出手法」におけるソースコードからの制御フローを表す UPPAAL モデルの生成手法を精査し，Java，C，C++，C#，COBOL，VB を対象としたソースコードからの UPPAAL モデルへの変換を定義する．本研究目標では，段階的な変換を自動化するツールと UPPAAL モデルに検査情報を設定することが容易に可能な支援ツールを開発する．

(3) 当初の期待される効果

このツールにより，対象システムに対する知識は十分に持っているが，モデル検査技術に関する知識の少ない技術者にも，モデル検査手法を利用した検査を実施できるようになることが期待される．

3.3.2 研究プロセスと成果

(1) 研究プロセス

多言語対応を行うことは，実用的な観点から重要であるが，ソースコードを忠実にモデル検査用モデルに変換する観点からは，汎用的な中間形式を定義しなければならず，作業に時間を要する．そこで，まず，１つの言語に対して，ソースコードを忠実にモデル検査用モデルに変換し，不具合現象を検証する方法を優先的に検討することを優先することとした．

そこで，本研究目標では，段階的な変換を自動化するツールと UPPAAL モデルに検査情報を設定することが容易に可能な支援ツールを開発することとし，以下のプロセスに従い研究を行う．

① Java のソースコードから UPPAAL モデルへの変換の文法を定義する．
② 変換を支援するツールを実装する．
③ 事例によるテストを行う．

(2) 具体的な研究成果の内容

本節では，Java のソースコードから，検証のための UPPAAL モデルへの変換方式と，ツールによる変換の支援方法を説明する．

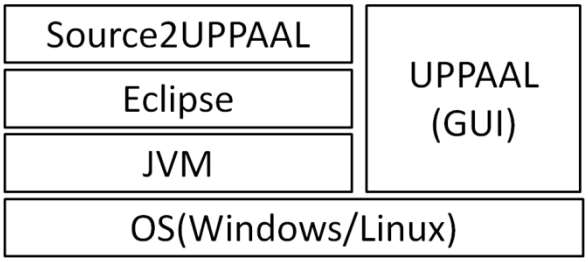


図 3-3-1　Source2UPPAAL の構成

図 3-3-1 は Source2UPPAAL のシステム構成を表している．Source2UPPAAL は統合開発環境である Eclipse のプラグインとして構築されている． Source2UPPAAL は次の 2 つの機能を提供する．

- ソースコードから UPPAAL モデルを半自動で生成する
- 検査式を生成する．ここでは各ステートメントへの到達可能性の検査式と検査補助モデル（検査補助システムと検査補助検査式）として与えられる検査式からなる．

図 3-3-2 に利用者から見た Source2UPPAAL の動作を示す．利用者からはプログラム言語（Source）とモデル検査式を部品化した「検査補助システムおよび検査補助検査式」が与えられる．Eclipse には Source を解析した結果得られたプログラムの抽象構文木が表示されるので検査補助システムとの対応関係（変換定義）を指定する．この指定作業が完了

すると Source2UPPAAL は UPPAAL モデルと検査式（全ノードに到達できるかを検査する式）を生成する．

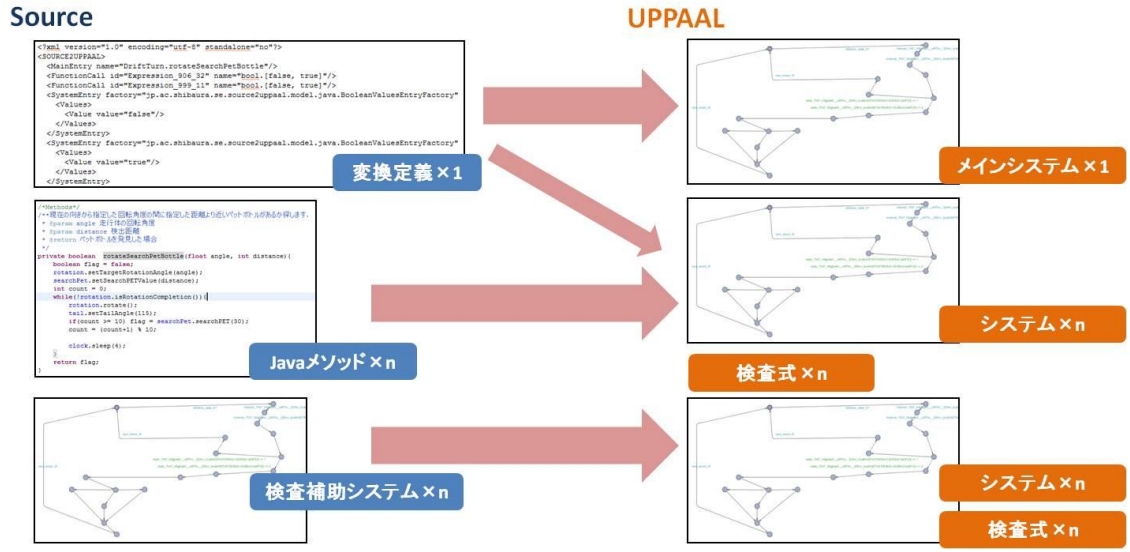


図 3-3-2　Source2UPPAAL の動作概要

図 3-3-3 は，Source2UPPAAL の内部で行われる変換処理の過程である．SOURCE2UPPAAL が生成した UPPAAL モデルの検証は利用者が手動で行う．

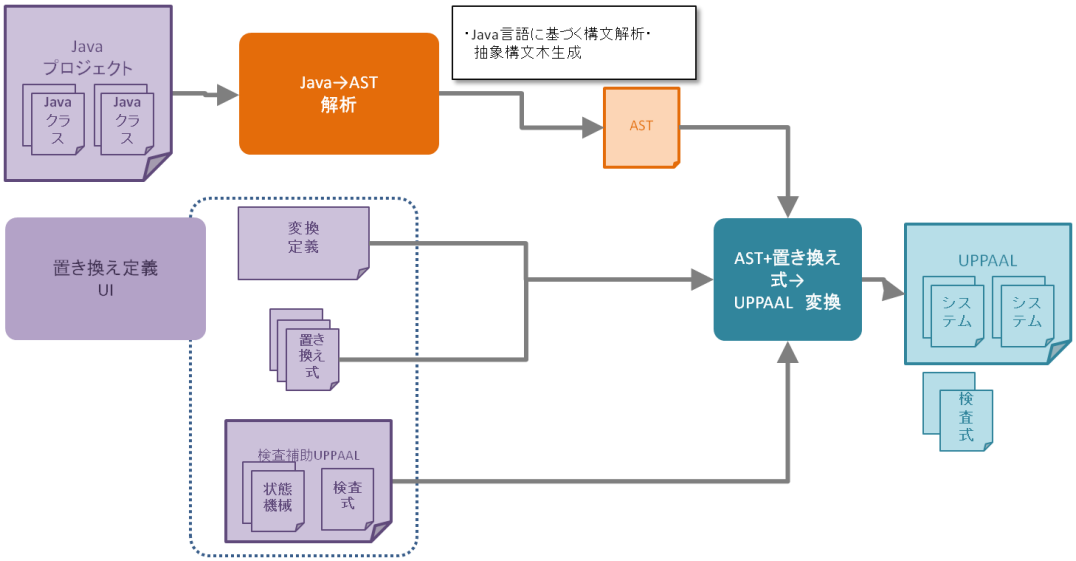


図 3-3-3　変換処理の構成

Source2UPPAAL の特徴は以下の通りである．

- 着目するソースコードの制御構造，変数に従って忠実にモデル化することができる
- 付随するシステムは非決定的な値を出力するシステムとみなす．なお，値の集合は利用者が与えるものとする
- 着目している領域を徐々に広げることができる

Source2UPPAAL を使った検査プロセスは図 3-3-4 に示すように基本的に 4 つのステップから構成される.

① 着目箇所の決定：
② 外部システムのモデル化
③ モデルの生成
④ モデルの検証

利用者は検査対象となるソースコードの完全なモデルを構築するのではなく，部分的なモデルを作りながら，段階的に検証を行うことができる．なぜなら，一般にデバッグ作業では問題の原因を特定する際には，問題の原因を徐々に絞込んでいくからである．なお，外部システムから与えられる値は範囲を持った非決定的な値として与えることもできる．以上のように，ソースコードを段階的に検証することで，ソースコードを完全に理解せずとも検証を行うことが可能となる.

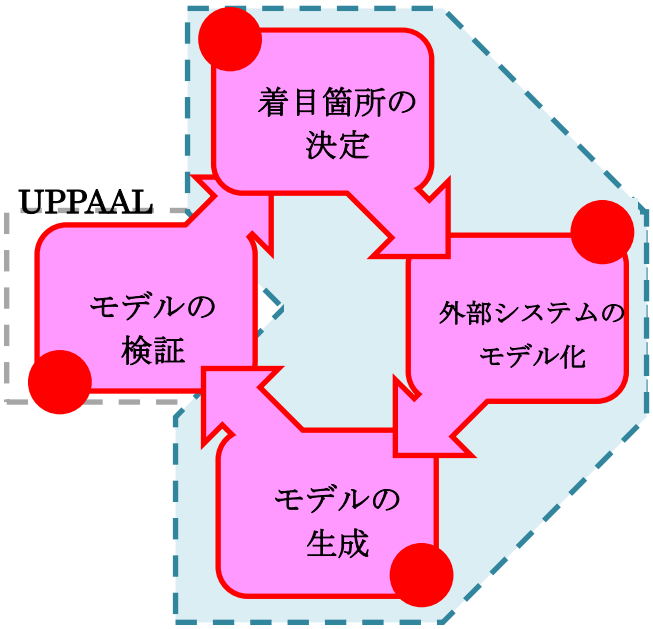


図 3-3-4　Source2UPPAAL を使った検証プロセスの概要

上記のステップの中で，Source2UPPAAL が直接的に支援するステップは①から③である.

次に，図 3-3-5 は，Source2UPPAAL の利用ステップ（①から③）と各ステップでの画面出力例を表している.

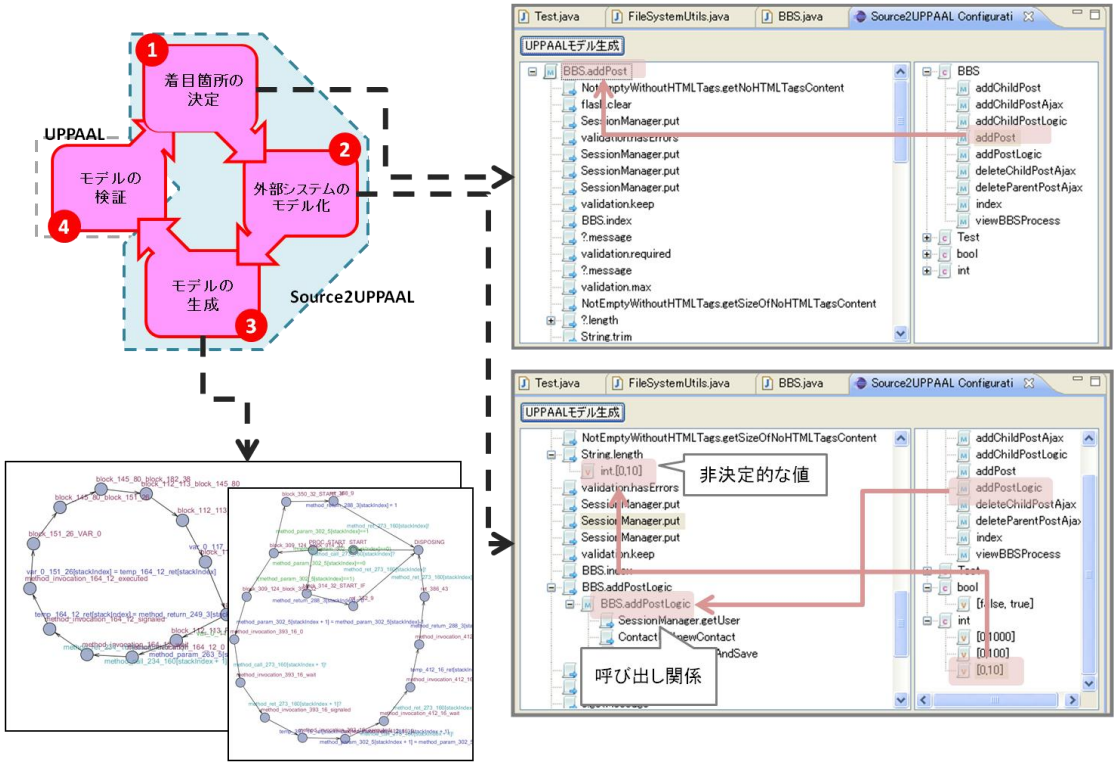


図 3-3-5　Source2UPPAAL の実行結果

変換定義を行うエディタ（図 3-3-6）ではドラッグ＆ドロップベースで次の 4 種類の変換を定義することができる．

① Java の static メソッドの UPPAAL モデル化
② Java メソッド呼び出しと非決定な値の関連付け
③ Java メソッド呼び出しと別の Java メソッドの関連付け
④ Java メソッド呼び出しと UPPAAL モデルの関連付け

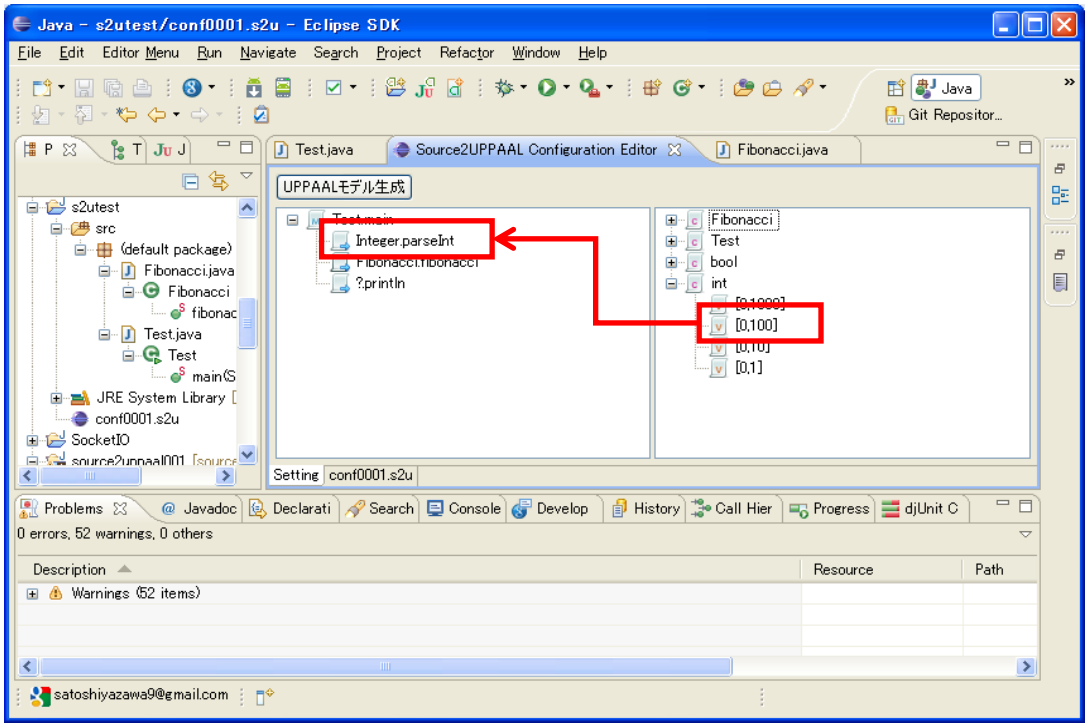


図 3-3-6　変換エディタ（非決定な値の関連付け）の画面例

関連する情報（ソースコード，定義ファイル，UPPAALL システムなど）は，Eclipse のプロジェクトを単位として管理される．プロジェクトの作成を行うウィザードを提供しており，利用者は指示される情報に従うだけでよい．

図 3-3-7 から図 3-3-11 は，非決定的な値を関連付ける一連の作業に対する画面例を表している．

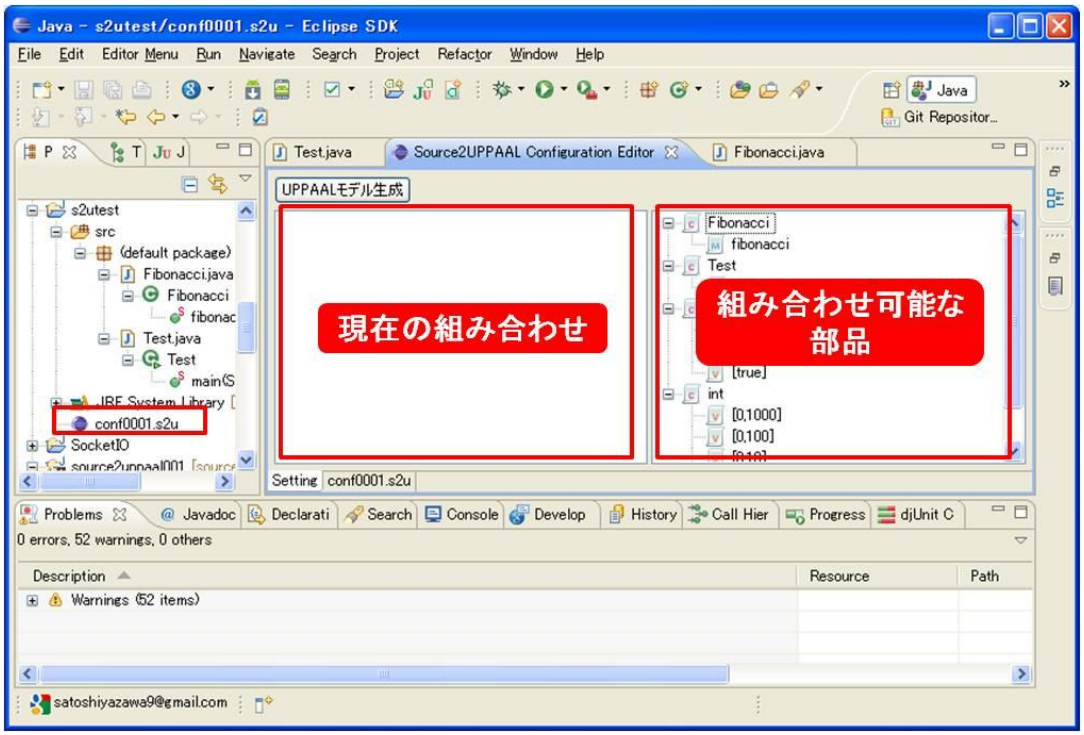


図 3-3-7　定義ファイルの作成（起動時）

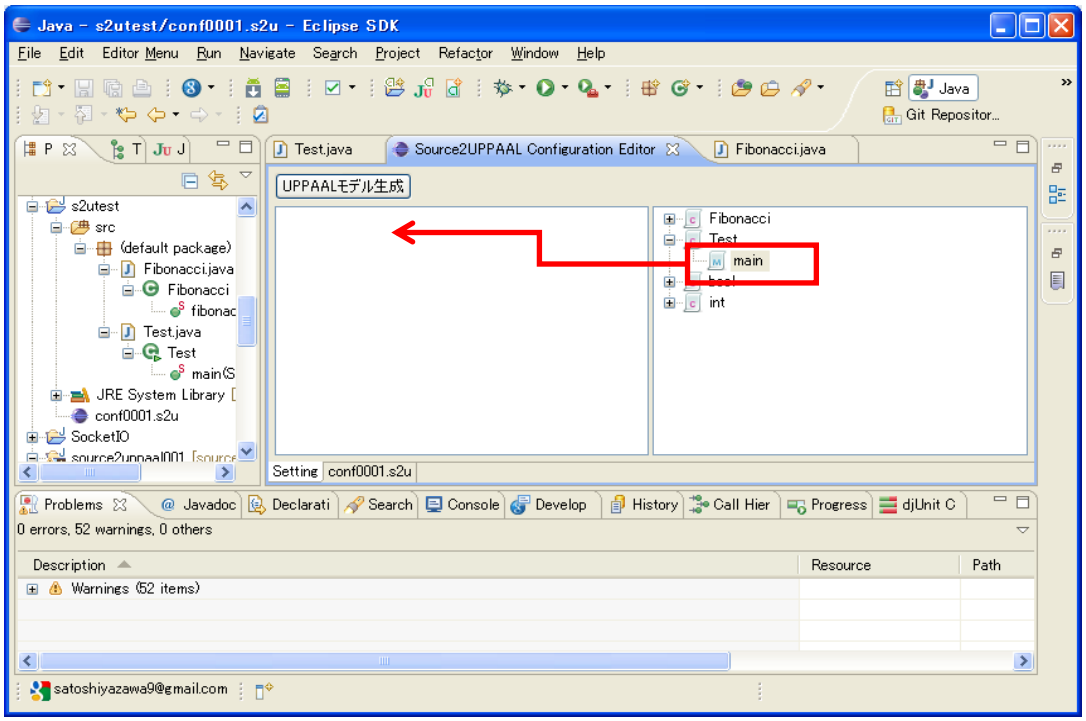


図 3-3-8　メインエントリの作成（1）

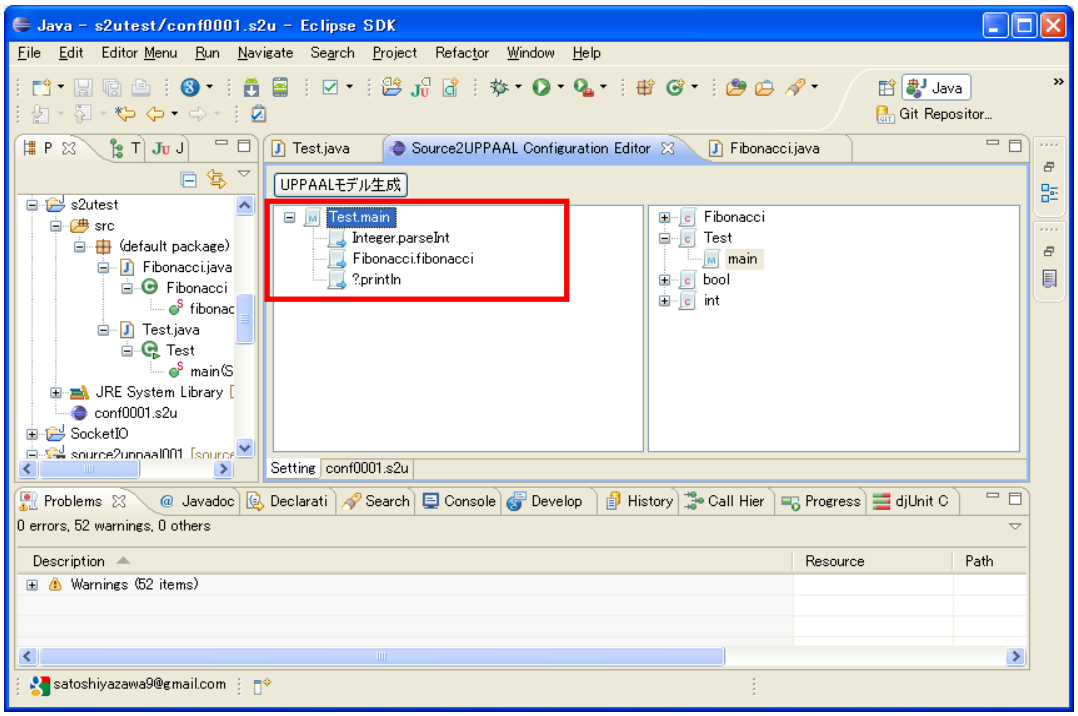


図 3-3-9　メインエントリの作成（2）

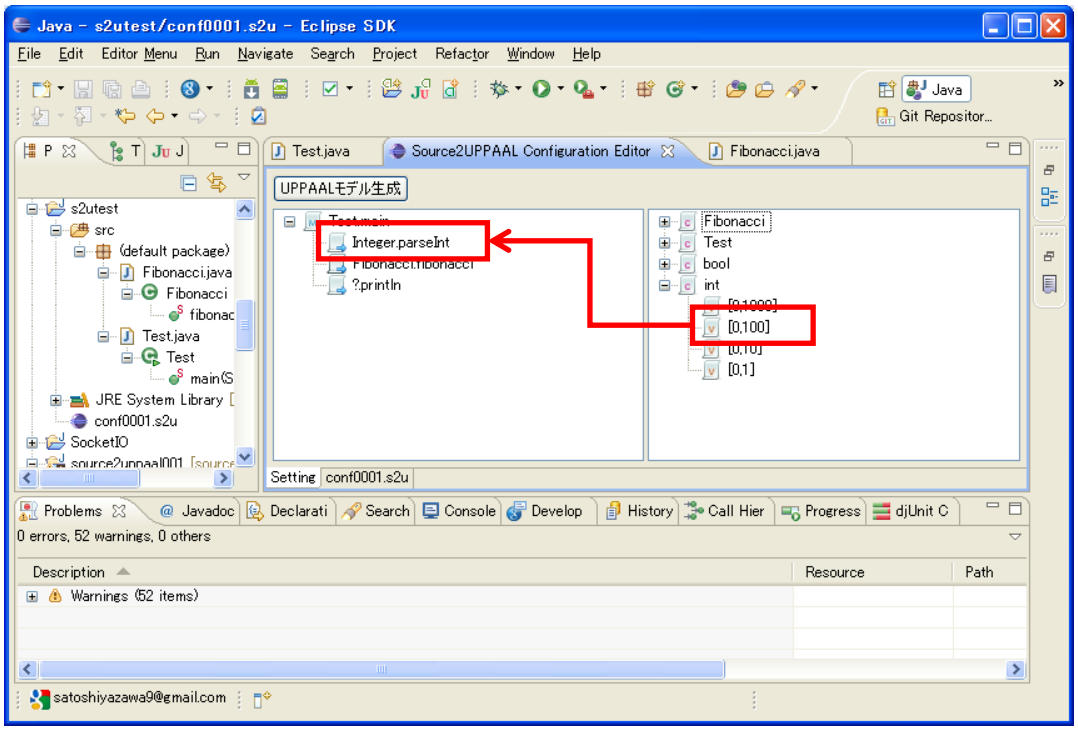


図 3-3-10　Drag&Drop で関連付け

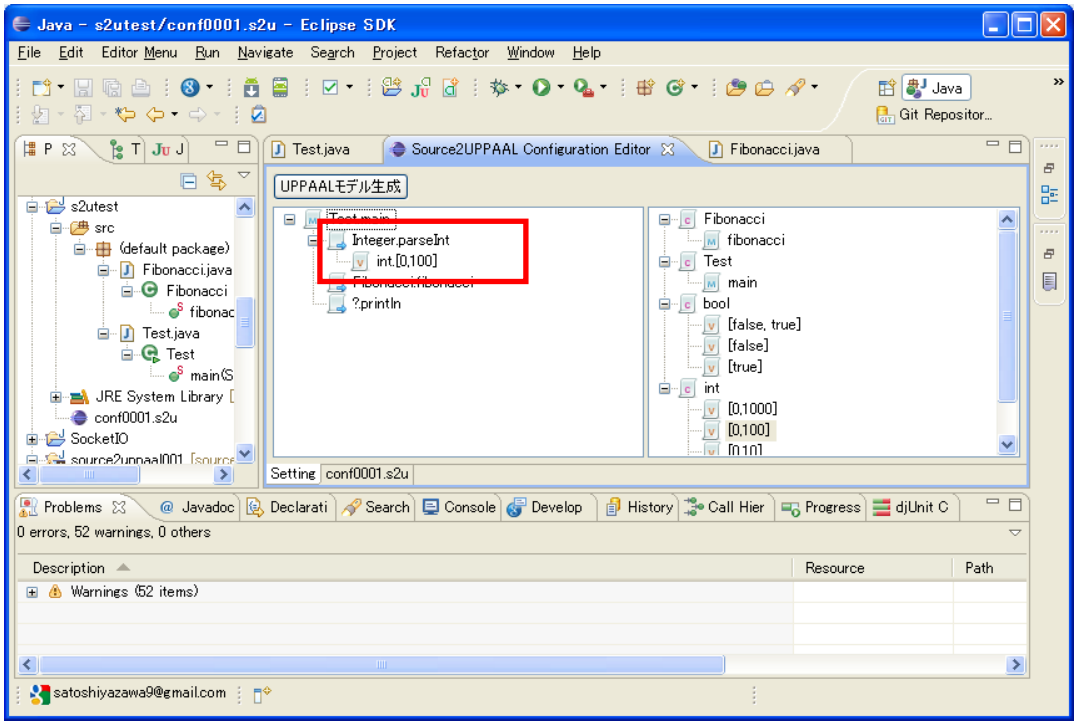


図 3-3-11　呼び出しと実体の関連付け

実際に Source2UPPAAL を使った変換の例を説明する.

まず，図 3-3-12 は非決定的な値を対応づける例である．図の左上が検査対象の Java ソースコードであり，左下がこのソースコードに対応する変換定義である．これらの情報から右側のような UPPAAL モデルを生成する.

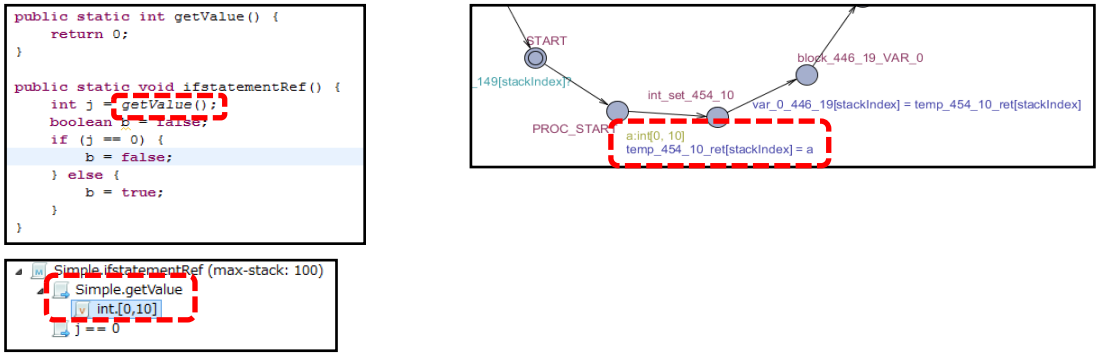


図 3-3-12　非決定的な値の定義と変換後

図 3-3-13 は他システムを呼び出す場合の例である. 左上の Java ソースコードに対して，変換定義において他システムの呼び出しを定義した場合, 図下に示す UPPAAL モデルを生成する.

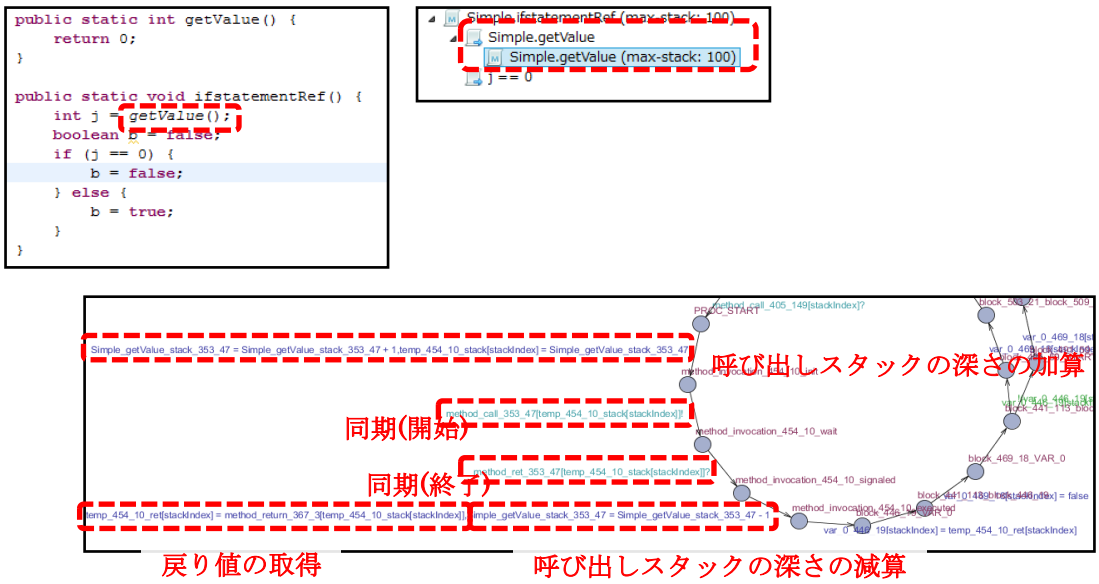


図 3-3-13　他システムの呼び出しの定義と変換後

関連付けを終えた後，実際にモデルを生成するには，「UPPAAL モデル生成」というボタンをクリックする（図 3-3-14）．もし，参照漏れなどのエラーがあれば，*. java エディタにエラーとしてマークされる．

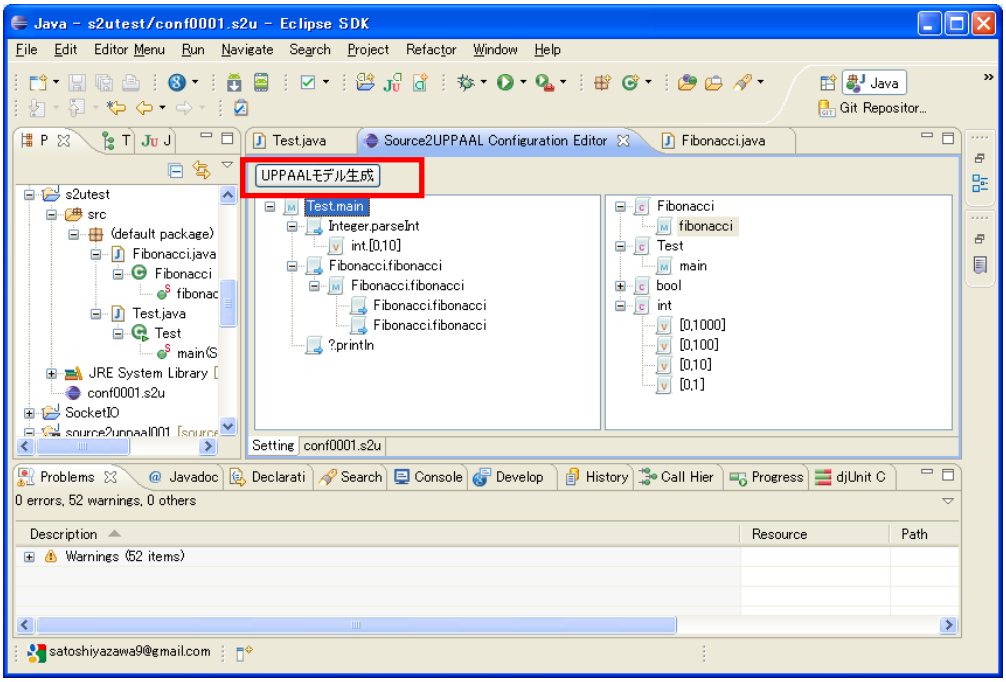


図 3-3-14　モデル生成

Source2UPPAAL は，Java 言語を対象にプロトタイプを構築した．UPPAAL モデルに関する知識が十分でない利用者でも，着目点を設けて小さなモデルを生成，動作を実際に確認した上で，徐々に目的に応じた詳細度のモデルを構築できるようになる．

3.3.3 実用化へ向けた課題と問題点

(1) 課題と問題点

今後の課題として次の 2 点が挙げられる．

① ベースとする言語

今回，Java 言語を対象に一連の作業を支援するツールを試作した．極力 Java 固有の処理方式に依存しないように考慮してきているが,今後は Java 固有の処理と他の言語の相違点などを整理した上でこの試作を通じて得た知見を活用し，今後は他の言語についても検討を進めたい．

② UPPAAL モデルの最適化

ソースコードとの対応関係を理解できるようにするため,UPPAAL モデルへの変換はできるだけ単純な置き換えになるように実装している．そのため，UML2UPPAAL 同様，検査にかかる時間が非常に長くなってしまう場合がある．Source2UPPAAL についてもソースコードとの対応関係などの理解のしやすさを維持しつつ UPPAAL モデルに変換することを考えていく必要がある．

(2) 将来の応用方法

多言語対応を行うことは，実用的な観点から重要であるが，ソースコードを忠実にモデル検査用モデルに変換する観点からは，汎用的な中間形式を定義しなければならず，作業に時間を要することから，今回は，Java 言語に絞って，ソースコードを忠実にモデル検査用モデルに変換し，不具合現象を検証する方法を優先的に検討することを優先することとした．（1）でも述べたとおり，他言語への適用を検討する．これにより研究目標 5 のシステムのマイグレーション事例への適用が可能になる．

3.4 研究目標 4「不具合現象や業務システムに必須の業務セオリーの策定」

3.4.1 当初の想定

(1) 想定する仮説等

無限ループ，オーバーフロー，タイムアウトの様な現象は，その現象自体は明確に認識できるが，原因箇所を特定することが難しい．こうした不具合の要因は発生した現象を分析するだけで特定できるわけではない．そのアプリケーションが実現しようとしている企業が業務遂行上，遵守すべき手順である業務セオリーを考慮する必要がある．

個々の企業においては業務を確実に遂行するために業務セオリーを策定する．業務セオリーは企業の基幹業務システムにおいて多様な条件マスタにより表現されることが多い．

例えば，購買業務における発注処理の場合，相手先・仕入方法・品物・日付・数量等の組合せ(業務セオリー)を条件マスタで表し，これを利用して通貨・換算レート・単価・値引き・運賃・金額端数処理等の決定を行う．この場合，条件マスタをモデル化することにより，正しい業務セオリー（組合せ）で処理が行われているかを検査することができると考える．

(2) 当初の到達目標

無限ループ，オーバーフロー，タイムアウト等の原因箇所の特定が困難な不具合を解決するために，業務遂行上遵守すべき手順である業務セオリーを定義し利用する．そのために，本研究目標では業務セオリーを検査可能なモデルにする手法の策定を行う．

(3) 当初の期待される効果

この手法により,経験者以外には特定が困難な不具合原因を見つけることが期待される．

3.4.2 研究プロセスと成果

(1) 研究プロセス

以下のプロセスに従い研究を行う．

① 一般的なプログラムの不具合現象に関する業務セオリーを定義する．
② システム不具合の調査を文献により行い，業務セオリーを検討する．
③ 企業の基幹業務システムの条件マスタを基に業務セオリーを定義する．
④ 変換ツールにより，事例を用いて UPPAAL での検証を行い，業務セオリーの妥当性をテストする．

(2) 具体的な研究成果の内容

① プログラムの不具合現象に関する調査

文献[12,13,14,15,16,17,18]により，プログラムの不具合原因に関する調査を行った．結果を表 3-4-1 にまとめる．

表 3-4-1　プログラムの不具合原因

No	不具合原因
01	mem: Memory clobbered or used
	メモリの上書きや不足。 予約してあるメモリーパーティションへの書き込みすぎによるシステムのクラッシュ 配列に対する範囲外の添字指定
02	vendor: Vendor's problem (hardware or software).
	ベンダーの問題。(ハードウェア・ソフトウェア) バグがあるコンパイラーや欠陥のあるロジックボード(論理回路基板) 代替手段を開発する、もしくはベンダーからの改善措置待ちになるもの。
03	des.logic: Unanticipated case (faulty design logic).
	ロジックの設計の失敗 想定外のケース。(設計に欠陥がある)
04	init: Wrong initialization
	初期処理の失敗。
	間違ったタイプの宣言、定義の重複。データ構造の間違いなど
05	var: Wrong variable or operator.
	誤った変数や演算子
06	lex: Lexical problem, bad parse, or ambiguous syntax
	語彙の問題、悪い文法解析、不明瞭な文法
07	unsolved: unknown and still unsolved to this day
	まだ知られていないもしくはまだ解決されていない問題
08	lang: language semantics ambiguous or misunderstood .....
	言語のあいまいな意味、誤解
09	behav: End-user's (or programmer's) subtle behaviour.
	エンドユーザ、プログラマの潜在的な振る舞い あいまいでいい加減なキー操作、冗談で入力したてきとうなコード

② 事例を用いた UPPAAL での検証

1) ET ロボコンプログラムへの適用

検証の対象は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[19]にて使用したロボットの走行制御プログラムである．ドリフトターンと呼ばれるレース開始後に決定されるペットボトルの位置によって，ターンエリアの旋回ルートを変更するという難所の走行プログラムにおける不具合の現象は走行アクションの基点となるペットボトルの検出ができないというものである．

この事例は直接的に業務セオリーを扱うものではなく，プログラムの制御フローの正当性を確認するものであるが，業務セオリーはプログラムの制御フローと密接に関係するため，制御フローの正当性を確認することが可能か検証するために最初の検査対象とした．

検査には検査支援ツール Source2UPPAAL を使用する．対象となるペットボトルの認識処理を行っているソースコード(DriftTurn. java)はわかっているためこれを検査する．検査はメソッド単位で行うため，メソッド rotateSearchPetBottle(図 3-4-1)を Source2 UPPAAL で指定する．

```
1         /*Methods*/
2         /**現在の向きから指定した回転角度の間に指定した距離より近いペットボトルがあるか探します.
3          * @param angle 走行体の回転角度
4          * @param distance 検出距離
5          * @return ペットボトルを発見した場合
6          */
7         private boolean  rotateSearchPetBottle(float angle, int distance){
8                 boolean flag = false;
9                 rotation.setTargetRotationAngle(angle);
10                searchPet.setSearchPETValue(distance);
11                int count = 0;
12                while(!rotation.isRotationCompletion()){
13                        rotation.rotate();
14                        tail.setTailAngle(115);
15                        if(count >= 10) flag = searchPet.searchPET(30);
16                        count = (count+1) % 10;
17                        clock.sleep(4);
18                }
19                return flag;
20        }
```

図 3-4-1　ソースコード

Source2 UPPAAL により，その構文木を解析し，変数に対してモデル化対象項目に用意してある boolean 型もしくは int 型の非決定リストを割り当てる(図 3-4-2)．ただし計算可能な計算式はそのまま利用する．

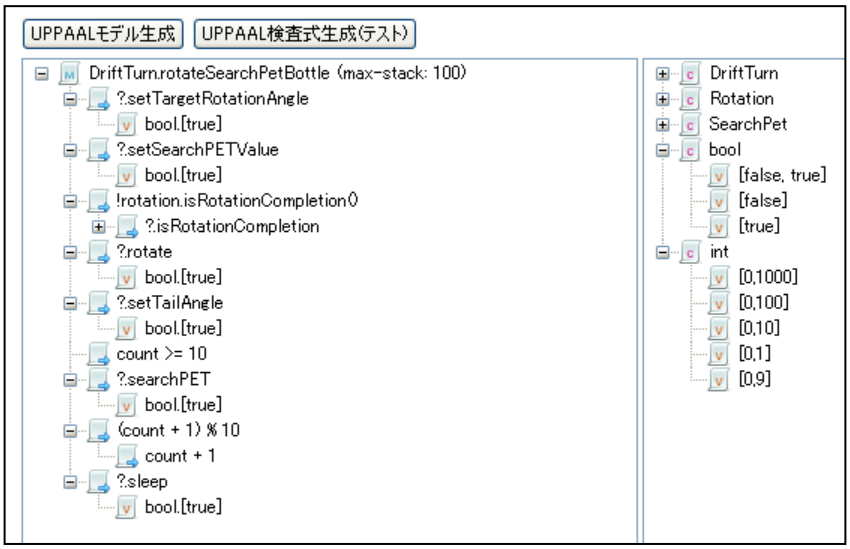


図 3-4-2　Source2UPPAAL 設定画面

この状態で UPPAAL の検査モデルを自動生成する．生成されたモデルは図 3-4-3 のようになる．

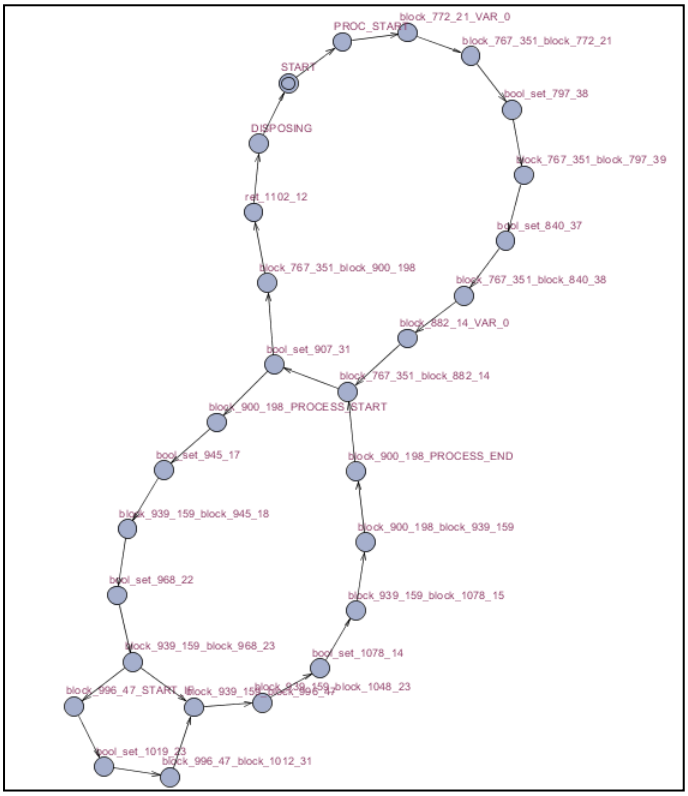


図 3-4-3　生成モデル

このモデルに対して検査を行う.ここでは,制御結果が期待する結果を出力しないため,最初にこのソースコードの制御フローが全て通りえる状態になっているかの検査を行う.検査モデルは制御フローと同様の特性をモデル検査の網羅性と非決定性により持っているため，検査モデルの全ロケーションのへの到達可能性の検査を行えば制御フローを全て通りうるかを確認できる.

全ロケーションへの到達可能性検査の検査式は自動生成することができる．例として以下の検査式をあげる．ここで START が到達可能性を検査するロケーションである.

A[] not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).START

検査式の意味は「ロケーション START に到達することは決してない」である．従ってこの検査式を実行した検査結果が Property is NOT satisfied(満たされない)となった場合はそのロケーションへの到達が可能である．Property is satisfied(満たされる)となった場合はそのロケーションへの到達はできないことになる.

この検査式を実行すると結果は表 3-4-2 のようになる.No22～No24 の検査式で到達できないロケーションがあることが判明する.

表 3-4-2　検査式及び検査結果一覧

No	検査式	検査結果	到達可否
01	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).START	Property is NOT satisfied.	到達できる
02	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).PROC_START	Property is NOT satisfied.	到達できる
03	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).DISPOSING	Property is NOT satisfied.	到達できる
04	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_767_351_block_772_21	Property is NOT satisfied.	到達できる
05	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_772_21_VAR_0	Property is NOT satisfied.	到達できる
06	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_767_351_block_797_39	Property is NOT satisfied.	到達できる
07	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).bool_set_797_38	Property is NOT satisfied.	到達できる
08	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_767_351_block_840_38	Property is NOT satisfied.	到達できる
09	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).bool_set_840_37	Property is NOT satisfied.	到達できる
10	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_767_351_block_882_14	Property is NOT satisfied.	到達できる
11	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_882_14_VAR_0	Property is NOT satisfied.	到達できる
12	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_767_351_block_900_198	Property is NOT satisfied.	到達できる
13	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).bool_set_907_31	Property is NOT satisfied.	到達できる
14	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_900_198_PROCESS_STAR	Property is NOT satisfied.	到達できる
15	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_900_198_PROCESS_END	Property is NOT satisfied.	到達できる
16	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_900_198_block_939_159	Property is NOT satisfied.	到達できる
17	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_939_159_block_945_18	Property is NOT satisfied.	到達できる
18	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).bool_set_945_17	Property is NOT satisfied.	到達できる
19	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_939_159_block_968_23	Property is NOT satisfied.	到達できる
20	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).bool_set_968_22	Property is NOT satisfied.	到達できる
21	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_939_159_block_996_47	Property is NOT satisfied.	到達できる
22	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_996_47_START_IF	Property is satisfied.	到達できない
23	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_996_47_block_1012_31	Property is satisfied.	到達できない
24	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).bool_set_1019_23	Property is satisfied.	到達できない
25	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_939_159_block_1048_23	Property is NOT satisfied.	到達できる
26	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).block_939_159_block_1078_15	Property is NOT satisfied.	到達できる
27	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).bool_set_1078_14	Property is NOT satisfied.	到達できる
28	A□ not SYSTEM_DriftTurn_rotateSearchPetBottle(0,0).ret_1102_12	Property is NOT satisfied.	到達できる

到達できないロケーションは block_996_47_START_IF，block_996_47_block_1012_31，block_996_47_START_IF の 3 つである．検査モデル上は図 3-4-4 の 3 つのロケーションである．これにより，ロケーション block_939_159_block_968_23 の分岐処理に問題があることが想定できる．

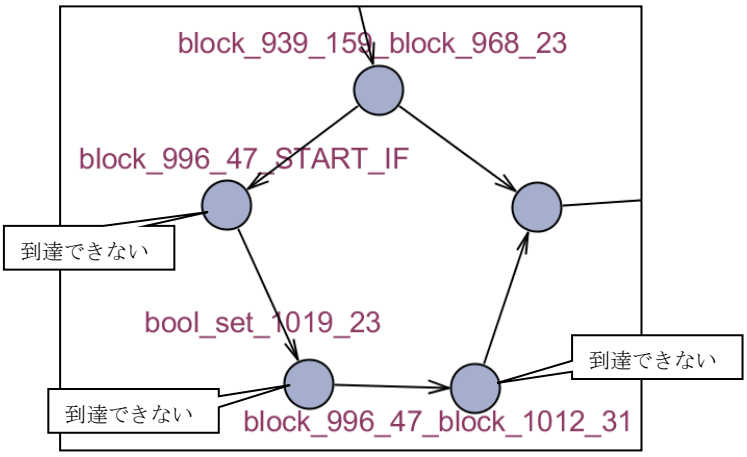


図 3-4-4　到達できないロケーション

検査モデルのこの部分はソースコードの図 3-4-5 の点線の部分にあたる．

```
if(count >= 10)
        flag = searchPet.searchPET(30);
```

図 3-4-5　対応するソースコード

変数 count が 10 に達しないことが予想される.モデル検査ツールは検査式が満たされなかった場合，その満たされない状態になる過程を反例として提示する．反例はトレースファイルとして記録されるため検査結果の分析に利用することができる．変数 count の値をトレースファイルから取り出しグラフ化した(図 3-4-6)．縦軸が count の値で，横軸はトレースのレコード数である.グラフを見ると変数 count は 9 のあと 10 にならずに 0 に戻されているのがわかる.

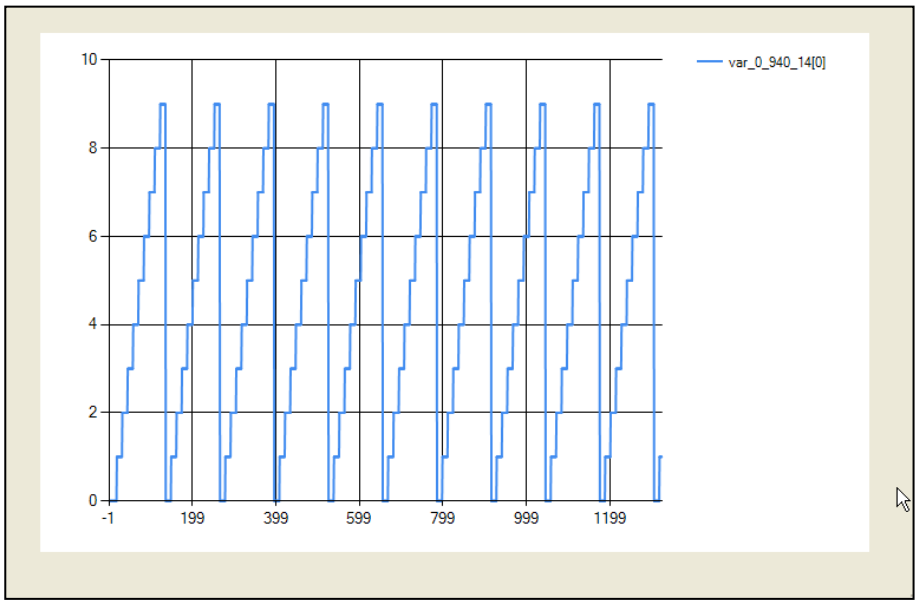


図 3-4-6　トレースファイルのグラフ化

Count が 9 から 0 にリセットされている部分をトレースファイルから作成したロケーション位置と count 値の対比表で特定する(表 3-4-2).No.24 のロケーションに到達した時にリセットされていることがわかる．このロケーションに該当するソースコードは 16 行目の”count = (count+1) % 10;”である.この式は 10 で割った余りを求める計算なので count は絶対に 10 になることがなく，ここが不具合の原因であることが特定できる．本来 10 回のループ中 1 回実行したかったのを作成者が誤ってコーディングしたことが判明した.

表 3-4-2　ロケーション位置と count 値の対比

Location No	count
16	9
19	9
18	9
20	9
24	0

2)　Apache Commons のバグへの適用

適用事例の不具合として無限ループを選んだ．これは無限ループが制御フローに関わる不具合で，誰でも経験したことがあり，理解しやすい現象だからである．オープンソースの既知のバグを開発作業時に発生した不具合に見立てて適用を行う．バグは Apache Commons IO 1.2 で発見された，IO-90 “Infinite loop in FileSystemUtils.freeSpaceWindows if share directory empty”である．既知のバグに対して本手法を適用することにより，同じ欠陥を発見できることと，欠陥が修正されたバージョンが既に存在するためそのバージョンでは不具合が発生しないことを確認する．検査するに当たって参考とするのは The Apache Software Foundation (https://issues.apache.org/jira/browse/IO-90)にあるバグの記述のみである．このメソッドの機能は dir コマンドを実行してそこからファイルの空きスペースを取得するものである．

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用意した．このモデルは UPPAAL を直接使い作成したものである．検査支援ツール Source2UPPAAL はソースコードから検査モデルを自動生成するとともに独自モデルを付加することもできる．1)と同様に検査するメソッドをツールに設定し，モデル化対象項目の想定される値を boolean 型もしくは int 型の非決定リストから割り当てる(図 3-4-7)．

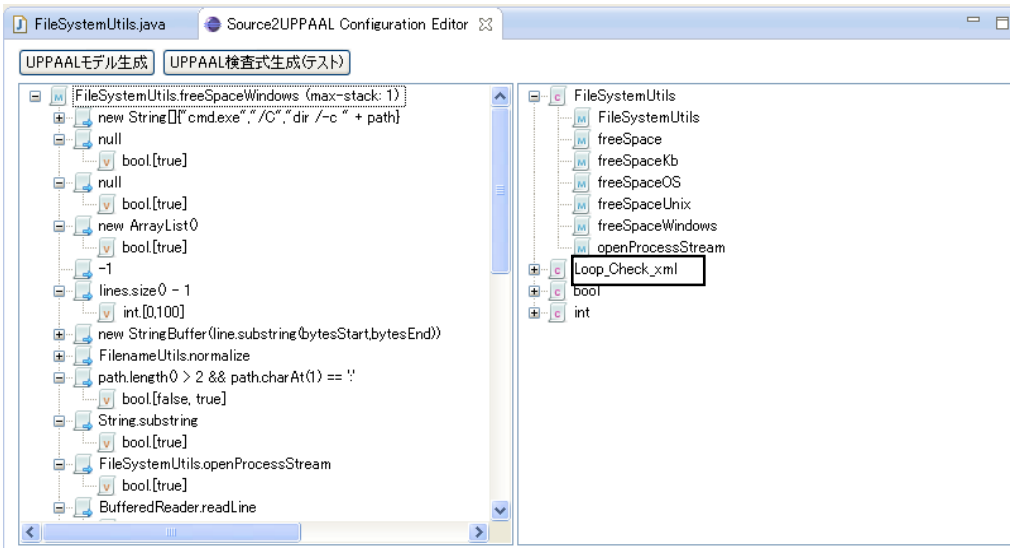


図 3-4-7　Source2UPPAAL 設定画面

さらに無限ループ判定用のモデルを追加する．追加は eclipse 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れば自動的に表示される．今回の追加モデルのファイルは Loop_Check.xml である(図 3-4-7 の四角の中)．

無限ループの判定モデルは図 3-4-8 である．敷居値(limit)を決めておきそれを超えた場合無限ループと判定される．今回閾値は 999 である．

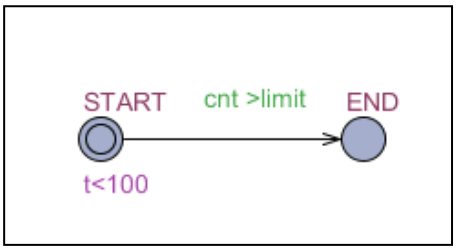


図 3-4-8　無限ループ判定モデル

またこの追加モデルにはカウントするための関数 counter()も定義されている．この関数は検査モデル上で利用できる．無限ループは条件分岐で抜けられない可能性が高いので条件分岐している部分に counter()を配置する．counter()を呼び出すモデルも用意する（図 3-4-9 ）．

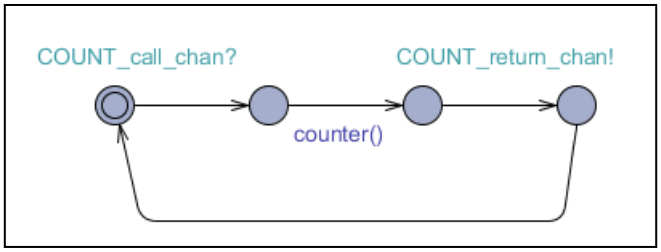


図 3-4-9　カウンター関数実行モデル

モデルの作成が完了したので検査を実施する．1)の例と同様に，まず最初に全ロケーションへの到達可能性の検査を行ったが，今回は到達できないロケーションは存在しなかった．次に無限ループの状態になるかを検査する．追加した無限ループモデルでロケーションの移動があるかを検査すれば確認できるため，検査式は以下のようになる．

A[] not LOOP.END

検査式の意味は「LOOP モデルのロケーション END に到達することは決してない」である．検査結果は Property is NOT satisfied(満たされない)となるため無限ループになることがわかる．無限ループ状態になることがわかったためさらに詳細な検査を行う．UPPAAL では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おこなう．全ロケーションの出現頻度をグラフで表す(図 3-4-10)．No.40～60 の間でループしているのが分かる．その範囲に絞り，ロケーションがトレースファイル上に現れる回数をカウントした（表 3-4-3)．これによりモデル上でループしている部分が特定できた（図 3-4-11)．

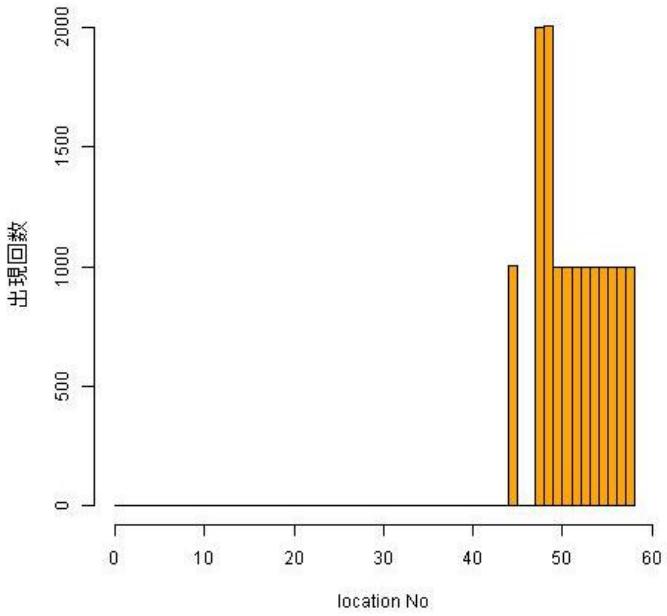


図 3-4-10　ロケーション出現頻度グラフ

表 3-4-3　無限ループ時おけるロケーション登場回数

No	ロケーション名	登場回数
45	block_7692_3370_block_9182_17	1001
48	method_invocation_9227_9_0	2001
49	method_invocation_9227_9_wait	2003
50	method_invocation_9227_9_signaled	1000
51	method_invocation_9227_9_executed	1000
52	block_9220_1442_PROCESS_START	1000
53	block_9220_1442_PROCESS_END	1000
54	block_9220_1442_block_9238_1424	1000
55	block_9238_1424_block_9253_29	1000
56	bool_set_9260_21	1000
57	block_9238_1424_block_9296_1355	1000
58	int_set_9300_13	1000

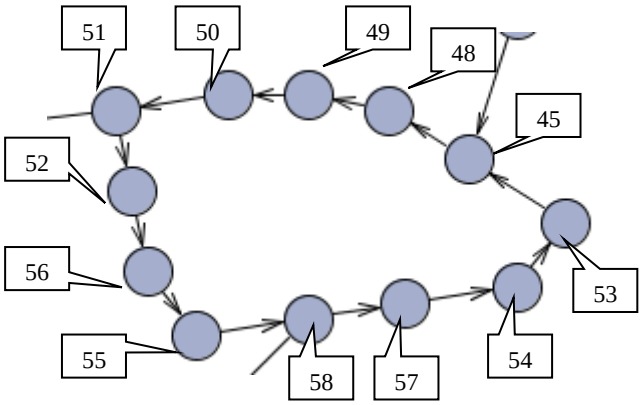


図 3-4-11　UPPAAL モデル上での登場頻度が高いロケーション

頻繁に登場するロケーションでループしていることが想定される．ロケーションの位置より対応するソースコードの位置も判明する．図 3-4-12 の点線内が該当する．

```
230         int bytesStart = 0;
231         int bytesEnd = 0;
232         outerLoop: while (i <= 9990) {
233            line = (String) lines.get(i);
234            if (line.length() > 0) {
```

図 3-4-12　ソースコード上のループしている部分

このソースコードの部分を調査する．233-234 行目が繰り返されそれ以降に進めないということは 232 行目の while を抜ける状態になれないことがあることを示している．ループを抜けるためには 234 行目の if 文の内部を実行しなければならない．234 行目の if 文の条件が満たされることが無ければ無限ループになることが予想され，トレースファイルを調査すると line.length()の値が常に 0 になっていることで確認できる．取得した文字列 line の長さが 0 の場合，無限ループになることが判明した．

3)　会計伝票発行における無限ループ

適用対象は指定された会計期間のデータを検索して請求書を発行する会計システムのプログラムである．検索対象になる会計期間(From-To) を指定し，その範囲内のデータの検索･出力処理を行う.会計期間の値は 1～12(4 月～3 月に対応) で,”From”の値は常に”To”の値以下になる．検索条件は”From”と”To”で 1 セットとなるが，複数セット指定して一度に検索することもできる．但し”From”または”To”に”1”が指定された場合は，特殊処理として”From”を前年度の特別会計期間 13 に置き換えた検索期間を 1 セット追加する．これは前年度未処理データも含めた表示をするためである．このプログラムは特定の条件で無限ループを発生させる．

元のプログラムは ERP(Enterprise Resource Planning) である SAP R/3 のアドオン開発言語 ABAP(Advanced Business Application Programming)により作成されたものである．サンプルプログラムは，帳票出力処理，画面制御等を除いたデータ抽出部の基本部分を Java で作成したものである(図 3-4-13)．

```
package InfinityLoop_test;
import java.util.ArrayList;

public class TableAdd {

    public void  table_add(ArrayList<Integer> arrayFrom,ArrayList<Integer> arrayTo ){
        int accPeriodFrom;
        int accPeriodTo;

        for(int i1 = 0 ;i1 < arrayFrom.size();i1++){
            accPeriodFrom =  arrayFrom.get(i1);
            accPeriodTo =  arrayTo.get(i1);
            if(accPeriodFrom ==1 ||accPeriodTo==1 ){
                arrayFrom.add(13);
                arrayTo.add(accPeriodTo);
            }
        }
 }
}
```

図 3-4-13　ソースコード

このソースコード Source2UPPAAL に設定する(図 3-4-14)．モデル化対象項目の想定される値を boolean 型，int 型の非決定リストから割り当てる．ループカウント用のモデル”COUNT”も設定する．

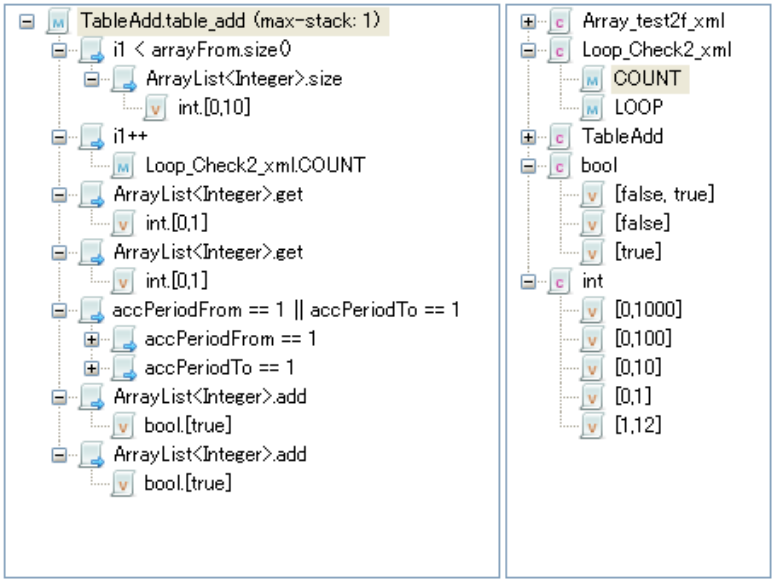


図 3-4-14　Source2UPPAAL 設定画面

上記設定から UPPAAL の検査モデルを作成する(図 3-4-15)．

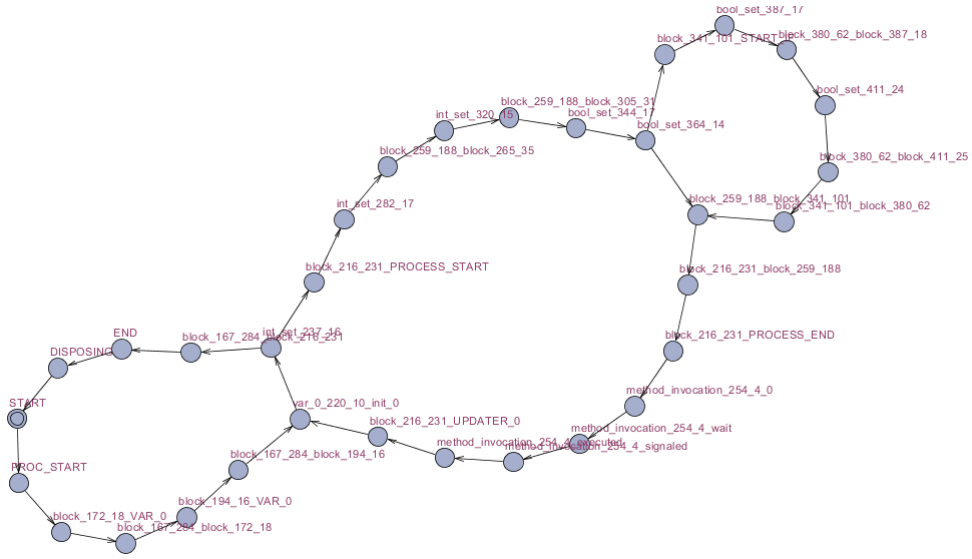


図 3-4-15　Source2UPPAAL 設定画面

このモデルに対して無限ループが発生するか確認するための以下の検査式を実行する.

A[] not LOOP.END

検査式の意味は「LOOP モデルのロケーション END に到達することは決してない」である. 検査結果は Property is NOT satisfied(満たされない)となるため無限ループになることがわかる. 無限ループ状態になることがわかったためさらに詳細な検査を行う.

3.4.3 実用化へ向けた課題と問題点

(1) 課題と問題点

今回不具合原因の文献の調査を行いその結果として得られた内容は, 企業において実務作業を行う中で経験したことがあるが, 単なるバグとして扱われ現場では明文化されていなかったものである. ここで得られた不具合原因のどれがモデル検査に適しているか検討を行う必要がある.

(2) 将来の応用方法

不具合の原因をモデル検査で特定することはもちろん行うが, そのプログラムが満たすべき仕様を実現しているかの検査もできるようにすることにより, 仕様漏れの確認も可能にする.

3.5 研究目標5「システムのマイグレーション事例の分析による業務セオリーの策定」

3.5.1 当初の想定

(1) 想定する仮説等

システムのマイグレーション時には，新システムが旧システムの仕様を満たしていることを保証する必要がある．しかし旧システムの仕様書がない場合，完全に保証することは難しい．仕様書があったとしても新システムが旧システムと同様に仕様を網羅していることを完全に確認することは困難である.新旧プログラムをそれぞれUPPAALモデルに変換して，その振舞いを比較することでマイグレーションの妥当性を検証することは有効であると考える．マイグレーションの場合，通常のテストでは言語が違うため変換コードの正しさは，テストを実施した範囲でしか確認できないが，新旧プログラムをUPPAALモデル化して比較検査することにより，設定可能な入力値の範囲内で新旧プログラムが異なる結果を出すことがあるかを全網羅的に検査できると想定される．

(2) 当初の到達目標

テストでは発見できないマイグレーション時の新旧システムの仕様齟齬を発見する手法を確立する．

(3) 当初の期待される効果

マイグレーションを行った新システムにおいて仕様書がない旧システムと開発言語が異なっていても仕様を満たしていることを保証することができる．

3.5.2 研究プロセスと成果

(1)研究プロセス

以下のプロセスに従い研究を行う．

① 仕様書を要求分析モデルの枠組みで精査し，これまでに得られた業務セオリーを基に業務セオリーを検討する．仕様書が無い場合は運用担当者の持つ知識もしくは運用マニュアルを精査して行う．

② 変換ツールにより,新旧システムを用いてUPPAALでの検証を行いその振舞いを検査する．

③ これまでに得られた業務セオリーからマイグレーションに適合できる業務セオリーを想定し，検査結果から，マイグレーション事例に適合しているかを検討する．

(2) 具体的な研究成果の内容

① 事例を用いたUPPAALでの検証

1) 会計伝票発行における無限ループ

3.4.2-(2)-②-3)と同じ事例である.適用対象は指定された会計期間のデータを検索して

請求書を発行する会計システムのプログラムである．検索対象になる会計期間(From-To)を指定し，その範囲内のデータの検索・出力処理を行う．会計期間の値は 1～12(4 月～3月に対応）で，”From”の値は常に”To”の値以下になる．検索条件は”From”と”To”で１セットとなるが，複数セット指定して一度に検索することもできる．但し”From”または”To”に”1”が指定された場合は,特殊処理として”From”を前年度の特別会計期間13 に置き換えた検索期間を１セット追加する.これは前年度未処理データも含めた表示をするためである．このプログラムは特定の条件で無限ループを発生させる．

この適用事例はマイグレーション時に仕様書があった場合を想定し，そこから作成される業務セオリーによりモデル検査が可能であることを確認するものである．

上記の仕様より業務セオリーをディシジョンテーブルとして表す．作成手順は次のとおりである．

業務セオリーの項目を抽出する

業務セオリー項目の種別を決定(条件/アクション)する

業務セオリー項目の仕様内容を記述する(複数可)

作成したディシジョンテーブルは以下のようになる(表 3-5-1)．

表 3-5-1　会計伝票発行　業務セオリー仕様ディシジョンテーブル

No	業務セオリー項目	種別	仕様1	仕様2	仕様3	仕様4	仕様5	仕様6	仕様7	仕様8	仕様9	仕様10
01	会計期間 From	条件	1～12		1～12 以外		会計期間 To以下		会計期間To より大きい		1	
02	会計期間 To	条件		1～12		1～12 以外		会計期間 From以上		会計期間From より小さい		1
03	入力エラーチェック	アクション	OK	OK	NG	NG	OK	OK	NG	NG	*Nothing	*Nothing
04	会計期間置き換え	アクション	*Nothing	*Nothing	*Nothing	*Nothing	*Nothing	*Nothing	*Nothing	*Nothing	前年度会計期間13を追加	前年度会計期間13を追加

同様に不具合についてもディシジョンテーブルを作成する(表 3-5-2)．

表 3-5-2　会計伝票発行　業務セオリー不具合ディシジョンテーブル

No	業務セオリー項目	種別	仕様1
01	プログラム実行回数	条件	999回以上
02	無限ループ	アクション	発生

これを基にステートマシン図の業務セオリーモデルを作成する．業務セオリー項目“会計期間 From”のモデルは以下のようになる(図 3-5-1)．

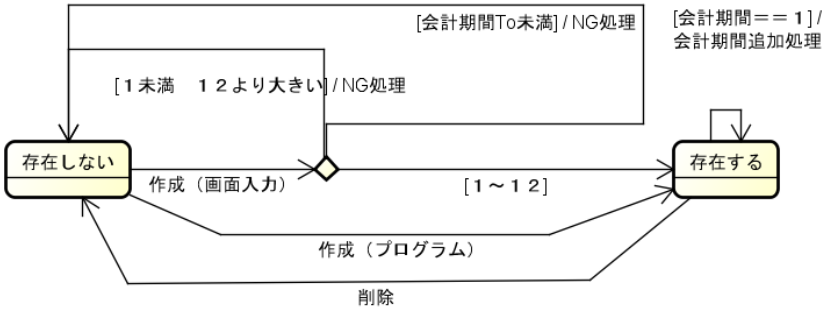


図 3-5-1　会計期間 From　業務セオリーモデル

作成した業務セオリーのモデルを UPPAAL のモデルに変換する(図 3-5-2).

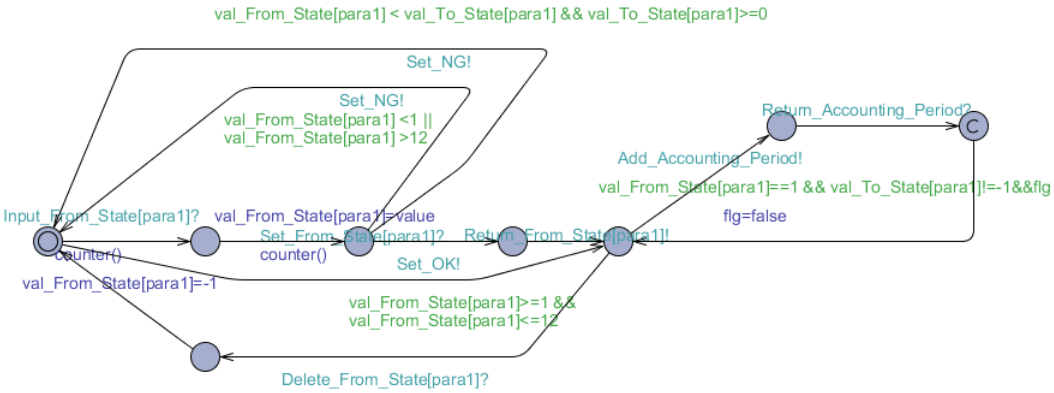


図 3-5-2　会計期間 From　検査モデル

無限ループの状態に到達するかの検査を行う．検査式は以下のようになる．

A[] not LOOP.END

検査式の意味は「LOOP モデルのロケーション END に到達することは決してない」である．検査結果は Property is NOT satisfied(満たされない)となるため無限ループになることがわかる．

以上のことより，ソースコードをモデル化して検査した結果と仕様をディシジョンテーブルからモデル化して検査した場合でも同じ検査結果が得られることが分かる．

3.5.3 実用化へ向けた課題と問題点

(1) 課題と問題点

現状では仕様をディシジョンテーブルに変換する作業は人手により行うしか方法が無く，属人性の高いものになっており，検査結果の精度を高めるためには業務セオリーの明確な抽出手順が必要である．

(2) 将来の応用方法

仕様とソースコードをモデル化して比較することにより，仕様が漏れなくソースコードに反映されているかの確認を行えるようになる．マイグレーションされたソースコード同士も同様にモデル化して比較することにより新旧で仕様に齟齬が無いことが確認できるようになりプログラムの品質の向上が望める．

４．考察

本研究では，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，運用時の想定外の使用による不具合の特定，マイグレーションによるシステムの再構築時のシステムの仕様の保証といった開発現場でのモデル検査利用のシナリオを想定して課題に取り組んできた．本章では，それぞれのシナリオで明らかになった事柄と今後の課題について述べる．モデル検査を実施する上での第一に克服しなければならない課題は，図 4-1 に示すように検査対象の定義と検査したいことの定義をその構成要素間で対応付けることである．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.

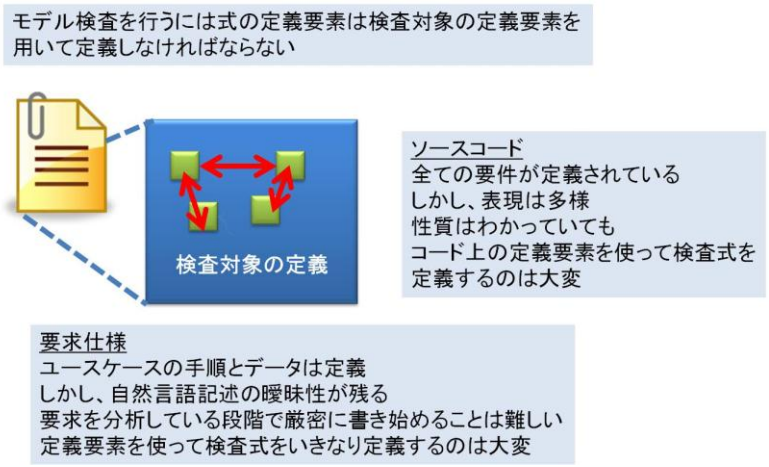


図 4-1　モデル検査を実施する上での問題点

4.1 研究により判明した効果や問題点等

4.1.1 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

要求分析モデルに対して，業務セオリーを検査したい対象データのステートマシン図で定義することで，UPPAAL モデルとその検査式を自動生成することができ，つぎの性質を検査することができた.

1) システムの永続化データに対して定義した基本的な性質である CRUD の振舞いとシステムの全サービスを定義した振舞いモデル間に矛盾が発生しない.
2) セキュリティ属性が定義されたデータのセキュリティ属性に関する基本的な性質に対して，システムのあるサービスがその属性に関する適切な振舞いを満たしている．ここで,「適切な」の意味は,セキュリティ評価の標準規格であるコモンクライテリア[24]によって定めている.

発表論文に対するコメントにも記載したが「CRUD の観点からシステムの振舞いを抽象化してモデル検査を適用するというアプローチは，現実的な形式検証の実現を目指す上で独創的かつ有望と考えます.」「分析者が検査式（CTL 式）をそのつど新規に書きくだす必要がなく，実務者向きの技術です.」といった，当初の狙い通りの評価を得ている．さらに，求められているのは，「CRUD 以外の観点」であり，本研究では，非機能要件の 1 つであるセキュリティ要件をそのセキュリティ属性に基づき検査する方法を提案した.

属性に関するステートマシン図を定義し，その遷移を引き起こす振舞いとアクティビティ図の属性に対する振舞いの対応付けにより，CRUD の振舞いだけでなく，オブジェクトの属性に関連付けられた振舞いを扱うことができるようになった.

今回の実験で，モデル検査が終了しないケースへの対処がモデル定義に必要であることが判った.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定義する．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判ればよいので，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.

CRUD テーブル[24]はデータ操作分析に用いられ，振舞いとデータの関係を俯瞰し，その妥当性を確認することに役立つ．しかし，例えば，ユーザの利用シナリオに基づき，特定のフローのパスに沿って振舞いとデータの関係を確認できない．本研究では，取り得る全てのパスに沿って，CRUD に関する振舞いとデータの関係の妥当性を網羅的に確認できる方法を提案しており，CRUD テーブルのみでは確認の困難な点を支援している.

検査式の捉え方について，崔ら[26]は，画面遷移仕様（要求仕様）に対する画面遷移時の処理を記述したフローチャート（設計仕様）の整合性を検証する方法を提案している．また，Sciascio ら[28]は作成済みのモデル上で考えられる組み合わせを検査式として網羅的に作成する方法を提案している．Li ら[25]は特定のドメインに特化した仕様に対して検査式を作成する方法を提案している．このように上位や作成済みの仕様を検査式として捉える提案はあるが，開発の初期段階においては，それらの仕様が存在しないことがあり，適用が困難な側面がある．本稿では，CRUD に関する普遍的な性質やセキュリティ要件を検査式として表しており，開発の初期段階においても分析者は負担なく検査式を得られる.

形式仕様の導入の難しさについて，矢竹ら[29]は，複数のオブジェクトの協調動作の中で，オブジェクトの状態が不変条件を満たすかどうかを検証する方法を提案している．しかし，曖昧かつ不完全な要求から不変条件に係わる手続きの順序や条件および振舞いの事前・事後条件を厳密かつ妥当に定義することは，具体的なガイドラインなしには困難である.本研究では,モデルの段階的な形式化による形式仕様の導入の容易化を目指しており，本研究において提案した簡潔な記法や自動化された検査法は，より詳細な形式仕様を記述するための１つのガイドラインとなる.

4.1.2 運用時の想定外の使用による不具合の特定

ソースコードを制御構造に基づき，UPPAAL モデルに変換することで，ソースコードの構造的欠陥を特定することを支援するツール Source2UPPAAL を開発した．ソースコードの不具合現象は，アプリケーション独自の性質に依存する場合，その依存する要素を特定しなければならない．これは一般には困難であり，本研究では，つぎのようにソースコードのアプリケーション独自の性質ではなく，不具合現象として生じるプログラムの一般的性質をモデル化することで検査を行った．

- 制御により期待される結果が得られない場合に，すべての制御フローが全て通りえる状態であるかを検査する.検査式は,生成されたロケーションへの到達の検査であり，自動的に生成できる．
- 「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制御の条件式に対して設定することで，無限ループ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，自動で生成できる．

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，非決定性をもつ値を想定しながら，検査できる機能をも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判っているが，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モデル化する必要がある．

Achenbach[30]は様々なモデルチェックツールの抽象化テクニックを比較し,それぞれのツールを実際の問題に適用している．例題としてファイル IO のエラー処理をオープン，クローズ，エラーの 3 状態のモデルを使って抽象化しており，ソースコードの抽象化の考え方として参考になる．このアプローチは我々のものに近いが，これは各種モデル検査ツールの抽象化機能の比較が主な目的で，ソースコードより不具合を発見するという観点はなかった.Bao[31]も Achenbach と同様に確認したい状態のみ見ることができるようにプログラムを抽象化する手順の例として参考になった．これはソフトウェアコンポーネントリリース時のテストを念頭に置いたものであり，実際のプログラム開発における現場サイドでの不具合発見の運用を強く意識したものではなかった．Thomas[32]はリアルタイムの組込みシステムのために Java ソースコードから忠実にモデル化された UPPAAL モデルを定義している.適用対象は実時間処理の組み込みシステムで色付きレンガのソートを行うが，外部環境の影響が小さいため外部環境のモデル化は行われていない．Jianguo [33]と Bradbury [33]は，モデル検査とテストを併用することによって検証プロセスの精度を改良することを目指している． しかしながら，彼らはソースコードから忠実なモデルを定義しているわけではない．

4.1.3 マイグレーションによるシステムの再構築時のシステムの仕様の保証

上記 2 つのシナリオにおける業務セオリーの考え方をヒントに，マイグレーションによるシステム再構築時への適用を検討した．現段階では，仕様を理解している技術者が，あるデータの識別したい状態とそれらの間の遷移によって，そのデータの満たすべき性質を定義することができ，その振舞いが，ソースコードのどの要素に対応づくかを特定できた場合に，検査を行うことが可能であることがわかった．

部分的な仕様をソースコードの要素と対応付けるためには，現状では開発者の理解に依存しているという問題がある．

4.2 今後の課題

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．

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の方針と合わせて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，事例で述べたように反例の解析により，修正を支援する必要もある．この場合には，プログラム一般レベルではない不具合現象をモデル化する能力が要求されることと，ソースコードの要素との対応付けを行うために，ソースコード理解の支援が必要であると考える．

マイグレーションによるシステムの再構築時のシステムの仕様の保証に関しては，満たすべき性質を着目するデータに関するステートマシン図によって定義することを検討する．この場合，属性だけでなく，要件定義プロセスでの定義対象の拡張と同じ問題を解決する必要がある．また，マイグレーションを対象とするためには，複数の言語に対応した検査ツールを実現する必要がある．

参考文献

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

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

[3] IEEE Computer Society, IEEE Recommended Practice for Software Requirements Specifications, IEEE Std 830-1998 (1998).

[4] 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.

[5] Y. Aoki and S. Matsuura, A Method for Detecting Unusual Defects in Enterprise System , Proc of SOFTWARE ENGINEERING, PARALLEL and DISTRIBUTED SYSTEMS (SEPADS '11), pp.165-171, 2011.

[6] 青木, 松浦, ソースコード解析を利用したモデル検査に基づく欠陥抽出手法, ソフトウェア工学の基礎 XVII, 日本ソフトウェア科学会 FOSE 2010, pp. 95-100, 2010.

[7] Shinpei Ogata, Saeko Matsuura, A Method of Automatic Integration Test Case Generation from UML-based Scenario, WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS, Issue 4, Volume 7, pp.598-607, April 2010.

[8] 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.

[9] 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.

[10]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.

[11]Astah*, ChangeVision, http://astah.change-vision.com/ja/

[12]J.D. Gould, Some psychological evidence on how people debug computer programs, International Journal of Man-Machine Studies 7 (2) (1975) 151-182.″

[13]M. Eisenberg, H.A. Peelle, APL learning bugs, APL Conference, Washington, DC, 1983, pp. 11-16.

[14]W.L. Johnson, E. Soloway, B. Cutler, S. Draper, Bug catalogue: I, Yale University, Boston, MA,Technical Report 286, 1983.″

[15]D.N. Perkins, F. Martin, Fragile knowledge and neglected strategies in novice programmers,empirical studies of programmers, 1st Workshop, Washington, DC, 1986, pp. 213-229.

[16]D. Knuth, The errors of TeX, Software: Practice and Experience 19 (7) (1989) 607-685.

[17]M. Eisenstadt, Tales of debugging from the front lines, Empirical Studies of Programmers, 5th Workshop, Palo Alto, CA, 1993, pp. 86-112.″

[18]R. Panko, What we know about spreadsheet errors, Journal of End User Computing 2 (1998) 15-21.
[19]ET ロボコン 2012 公式サイト, http://www.etrobo.jp/2012/, (2013/01/07)
[20]H. Okuda, S. Ogata and S. Matsuura, Mapping Rule Between Requirements Analysis Model and Web Framework Specific Design Model, Knowledge-Based Software Engineering, Proc of the 10th Joint conference on Knowledge-Based Software Engineering, IOS Press, pp. 207-216, 2012
[21] A. Spillner. The W-MODEL. Strengthening the Bond Between Development and Test. In Int. Conf. on Software Testing, Analysis and Review, 2002.
[22]R. Shikimi , S. Ogata and S. Matsuura, Test Case Generation by Simulating Requirements Analysis Model, Proc of 2012 IEEE 36th International Conference on Computer Software and Applications, pp 356-357, 2012
[23]野呂, 小形, 松浦, UML 要求分析モデルとコモンクライテリアに基づくセキュリティ要求分析の統合手法情報処理学会 FIT2012 講演論文集 RO-008, pp. 77-80(第 4 分冊), 2012.
[24]独立行政法人 情報処理推進機構, “CC/CEM バージョン 3.1 リリース 3”, http://www.ipa.go.jp/security/jisec/cc/index.html
[25]van den Brink, H.; van der Leek, R.; Visser, J., Quality Assessment for Embedded SQL, Proc. of Seventh IEEE International Working Conference on Source Code Analysis and Manipulation, 2007 (SCAM 2007), pp. 163-170, 2007.
[26]崔銀惠, 河本貴則, 渡邊宏, 画面遷移仕様のモデル検査, コンピュータソフトウェア, Vol. 22, No. 3, pp. 146-153, 2005.
[27]Sciascio, E. D., Donini, F. M., Mongiello, M., and Piscitelli, G.: Web Applications Design and Maintenance Using Symbolic Model Checking, Proc. of the 7th European Conference on Software Maintenance and Reengineering (CSMR 2003), 2003, pp. 63-72.
[28]Li, H., Krishnamurthi, S. and Fisler, K.: Verifying cross-cutting features as open systems, in international conference on Foundation of Software Engineering ,2002
[29]矢竹健朗, 青木利晃, 片山卓也, コラボレーションに基づくオブジェクト指向モデルの検証, コンピュータソフトウェア, Vol. 22, No. 1, pp. 58-76, 2005.
[30]Achenbach, M., Ostermann, K. 2009. Engineering Abstractions in Model Checking and Testing. 9th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2009 , art. no. 5279920, pp. 137-146
[31]Bao, T., Jones, M.D. 2009. Test Case Generation Using Model Checking for Software Components Deployed into New Environments. Proc. of ICSTW ’09, pp. 57-66, 2009.
[32]Thomas Bogholm, Henrik Kragh-Hansen. 2008. Model-Based Schedulability Analysis of Safety Critical Hard Real-Time Java Programs, Proc. of JTRES 2008, pp. 106-114.
[33]Jianguo, Chen., Hangxia, Z., Bruda, S.D. 2008. Combining Model Checking and Testing for Software Analysis, Proceedings - International Conference on Computer Science and Software Engineering, CSSE 2008 2, art. no. 4722035, pp. 206-209

[34]Bradbury, J.S., Cordy, J.R., Dingel, J. 2007 Comparative Assessment of Testing and Model Checking Using Program Mutation. Testing: Proc. of TAICPART-MUTATION 2007., pp. 210-222,2007