独立行政法人情報処理推進機構 委託

2014 年度ソフトウェア工学分野の先導的研究支援事業
「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 27 年 2 月

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「2014 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えたものである.

目次

研究成果概要

１．背景

ソフトウェア開発には要求定義・設計・実装・テスト・運用の工程がある．これらの工程において，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．要件定義段階で，これらの全てを考慮するわけではないが，レベルの異なるこれらの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．開発者はこれらの性質を各工程で作り込まなければならないが，適切に定義できているかを検証することは困難である．それは，開発工程において，仕様のドキュメント化が十分でない，ドキュメント化されていても自然言語による非形式的な記述であることに起因する．このため，形式的記述により検証が可能な形式手法の導入が注目されている．形式手法の１つであるモデル検査とは「有限の状態空間を網羅的に調べて，与えられた性質が成り立つか否かを調べる」技術であり，検査を行うモデル検査器と状態空間を探索するシミュレータから構成され，性質が成り立たない場合には反例を提示するモデル検査ツールが提供されている．ここで，利用者は検査したい対象のシステムモデルと検査式の両方を定義することで，モデル検査器を用いて，その性質が成り立つか否かを検査することができ，反例により，問題点を発見する．モデル検査は，このように，レビューやテストとは異なり，システムの振る舞いがある性質を満たすか否かを網羅的な探索により自動的に検証する方法であり，検証の手段として期待できる．我々は，モデル検査に着目して，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」において，上述のソフトウェア開発工程で考慮されるソフトウェアの満たすべき性質を「業務セオリー」と呼び，要件定義プロセスと保守プロセスの観点から，現場の開発者がモデル検査ツールを用いて様々なシステムの性質の検証を行なう方法を研究した．そして，モデル検査ツールの入力となるシステムモデルと検査式を現場の開発者が要求分析モデルやソースコードから生成する方法の見通しを得た．

本研究では，現場の開発者がモデル検査ツールを直接操作せずに，モデル検査の恩恵を受けられる方法を開発するために，下記の研究課題を設定した．図１は研究のアプローチと解決しなければならない課題を示している．

①検査モデルと検査式の定義方法：

開発者が理解しやすい形式で「検査したい性質（機能要求・非機能要求)」を「検査対象の振舞いモデル（ソースコード)」と関連付けて定義することで，検査式を自動生成し，到達可能性や安全性の検査を行えるようにする．関連付けにはテストによく用いられるデシジョンテーブルで定義された仕様や非機能要求の１つであるセキュリティ要件定義を用いて検討する．

②システムモデルへの変換方法：

「検査対象の振舞いモデル」を状態爆発が起こらないようにシステムモデルへ変換する．検査したい性質に基づく抽象化方法を定義し，作業容易化のための自動化の方法を検討する．

③反例解析支援方法：

検査から得られた反例から問題点を発見する過程を支援する．

①から③の作業を現場の開発者が適切かつ容易に実施できるように支援する方法を検討するとともに，現場の開発者が容易に操作・理解できる変換支援ならびに反例解析支援ツールを開発することが本研究の課題である．

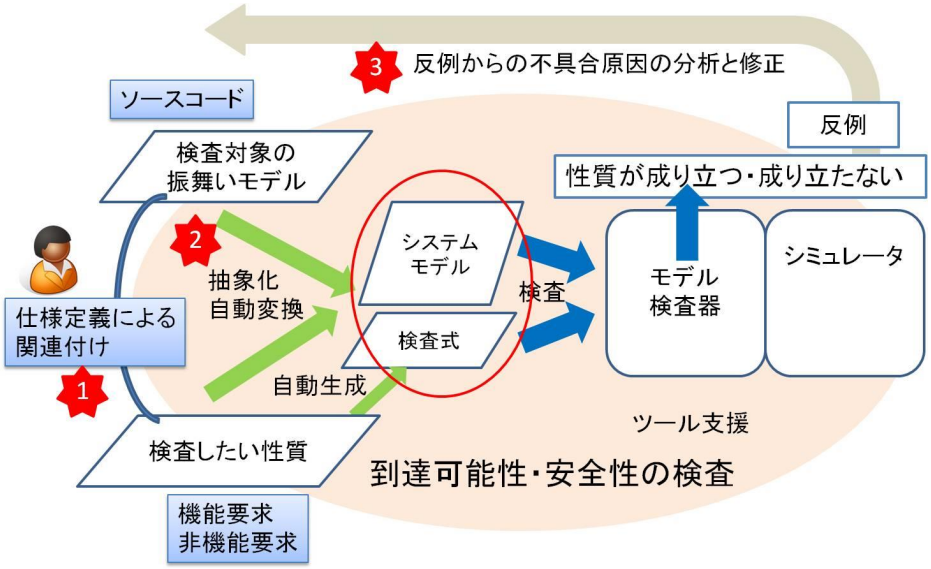


図１　研究のアプローチと解決すべき課題

２．研究目標

1 で述べた各課題に対する到達目標はつぎのとおりであり，図 2 がソースコードを対象とした本研究における研究アプローチを示している．

1)課題①「検査モデルと検査式の定義方法」については、検査したい性質（機能要求・非機能要求）を「システムが、ある条件のもとで、ある正当でない状況に陥ることが決して起こらない 」という安全性の観点から検査できるように、デシジョンテーブルおよびセキュリティ機能方針表の形式で整理する方法を定義する。

2)課題②「システムモデルへの変換方法」については、1）の方法に基づき、ソースコードをモデル検査用のシステムモデルに段階的に変換する方法を定義し、支援ツールを開発する。モデル検査ツールは UPPAAL を用いる。

3)課題③「反例解析支援方法」についてはモデル検査ツールから得られる反例を解析して、仕様を満たしていない原因を特定する方法を検討する。ここでは状態を特定する変数を導入して検査式を自動生成し、原因の絞り込み方法を検討する。

本研究では,本学で実際に稼働している学習支援システム LUMINOUS をセキュリティ要件の検証対象とする．本システムは，学生が開発したシステムを基に，再開発し，機能追加や権限の追加により拡張するという開発形態を取ってきた．仕様書としては，利用シナリオや，一部には UML モデルが存在する．開発現場での利用を図るため，ある程度の規模（ロジック 36,352 ステップ，画面 25,973 ステップ）の実システムを対象として，研究目標を

以下のように設定した．具体的には，これまで研究してきた UML 要求分析手法や CC を用いたセキュリティ要求分析手法および 2012 年度の研究成果を基に研究を進める．

(1) 仕様定義に基づく検査モデル・検査式の生成方法の定義
(2) 変換支援ツールの設計と開発
(3) LUMINOUS のセキュリティ要件の定義
(4) LUMINOUS のセキュリティ要件の検証
(5) 反例解析方法の検討

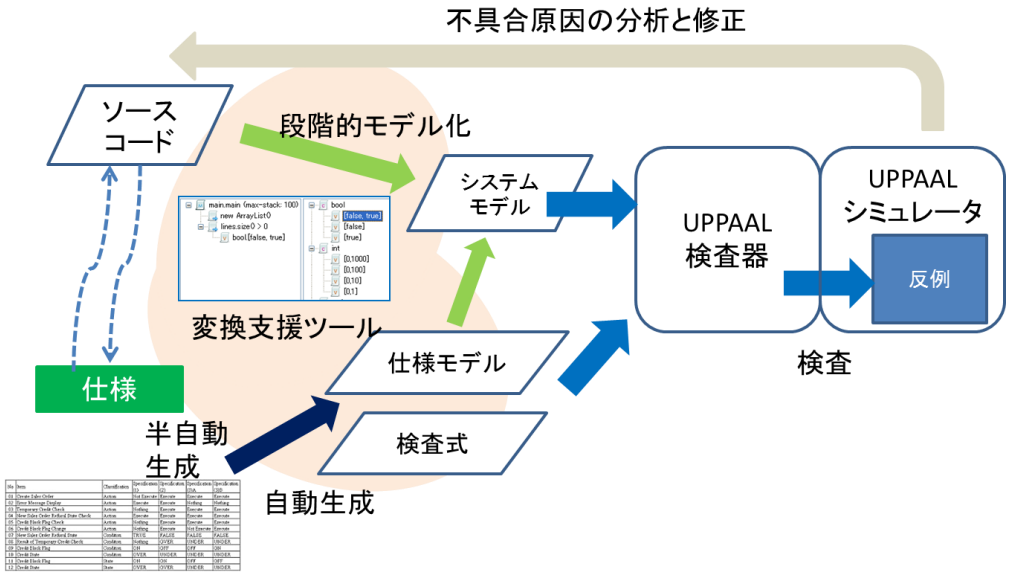


図 2　ソースコードを対象とした研究のアプローチ

３．仕様定義に基づく検査モデル・検査式の生成方法

以下の手順で，システムに対する仕様ならびにセキュリティ要件をソースコードと結びつけながら定義し，システムモデル．仕様モデル．検査式を生成して，検査を実施する．

1) LUMINOUS の要求分析モデル（ユースケース図，各ユースケースのアクティビティ図，クラス図）を要求分析手法に従い，既存システムを操作しながら，つぎのように定義する（図 3 参照）．まず，基本フローを事前条件・事後条件とともに定義する．基本フローに対し，例外フローは処理フローの行先ごとにまとめてガードに記述する．オブジェクトノードのクラス名はソースコードから抽出したエンティティモデルのクラス名を参照し，名前はアクションの目的語と一致させる．事前条件・事後条件・ガードに登場する場合はそれとも一致させる．さらに，画面構成に相当するクラスを定義する．ここで，クラス名は各ユースケースの作業状態を表すものとする．項目名は画面上のラベルと一致させる．また，ユースケースの使用するエンティティデータとそのセキュリティ属性をクラス図から特定する．そのセキュリティ属性の満たすべき性質はステートマシン図で定義する．

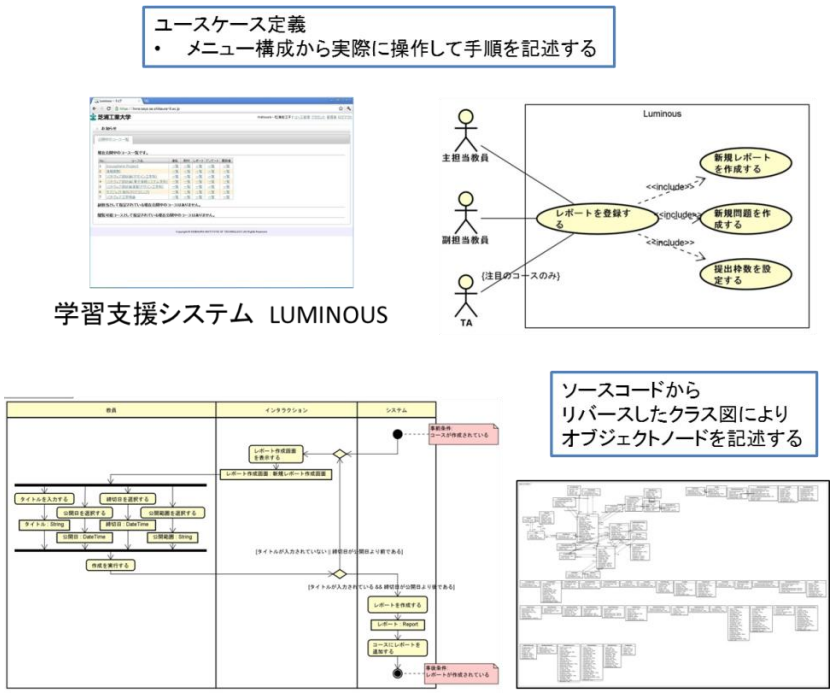


図 3 仕様定義の手順

2) ソースコードの静的解析により UI 要素とそのソースコード表現（UI コード）の対応表を抽出する．ソースコードの静的解析により抽出する情報は，entity クラス，entity クラスの属性，UI クラス，UI クラスのメソッド，ボタンやチェックボックスなどの UI 要素とそのソースコード表現の 5 種類である．

3) ユースケースの事前条件に従い組み合わされた検査シナリオとそれに対応するセキュリティ機能方針表のルールをつぎのように定義する．情報セキュリティの国際評価基準（ISO/IEC15408）である Common Criteria（CC）に定義されたセキュリティ機能コンポーネントモデルと要求分析モデルとの対応により，対象システムのサブジェクト（アクター）とオブジェクトに対してセキュリティ属性を定義する．要求分析モデルのアクティビティ図より，すべてのオブジェクトを抜き出し，それぞれを対象とするアクションを抽出する．ここで，各ユースケースにおいて対象とするオブジェクトの資産としての利用環境における価値を考慮し，資産とそれを何から守るべきかを検討し，オブジェクトにセキュリティ属性を付加して，守るべきルールを表 1 のようなセキュリティ機能方針表として定義する．

表 1 セキュリティ機能方針表

サブジェクト		オブジェクト		操作		ルール		
アクタ	セキュリティ属性	クラス	セキュリティ属性	ユースケース	アクション	FDP_ACF.1	FMT_MSA.3	FMT_MSA.1
学生	役割(学籍番号)	話題	公開／非公開	質問を投稿する	話題を生成する		ルールB1	
		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	ルールA		
				質問を投稿する	添付ファイルを生成する		ルールB2	
教員	役割(教員)	話題	公開／非公開	質問に回答する	回答を追加して話題を更新する			
					話題の公開／非公開を公開に変更する			ルールC1
					話題の公開／非公開を非公開に変更する			ルールD1
		添付ファイル	公開／非公開	質問に回答する	添付ファイルを生成する		ルールB3	
					添付ファイルの公開／非公開を公開に変更する			ルールC2
					添付ファイルの公開／非公開を非公開に変更する			ルールD2

図 4 は検査シナリオに基づく検査モデルの全体像を表している．検査シナリオは複数のユースケースに対応するメソッドのモデルを呼び出す．これはソースコードのメソッドに対応する UPPAAL モデルである．各メソッドが実行される間に，セキュリティ属性の更新メソッドが呼び出され，それに同期して，属性の仕様である状態遷移モデルが変化する．これにより，ユースケースが終了した時点で，セキュリティ属性の状態が想定の値であるかという上述の検査式を検査することで，ソースコードがルールを満たしているかを検査できる．

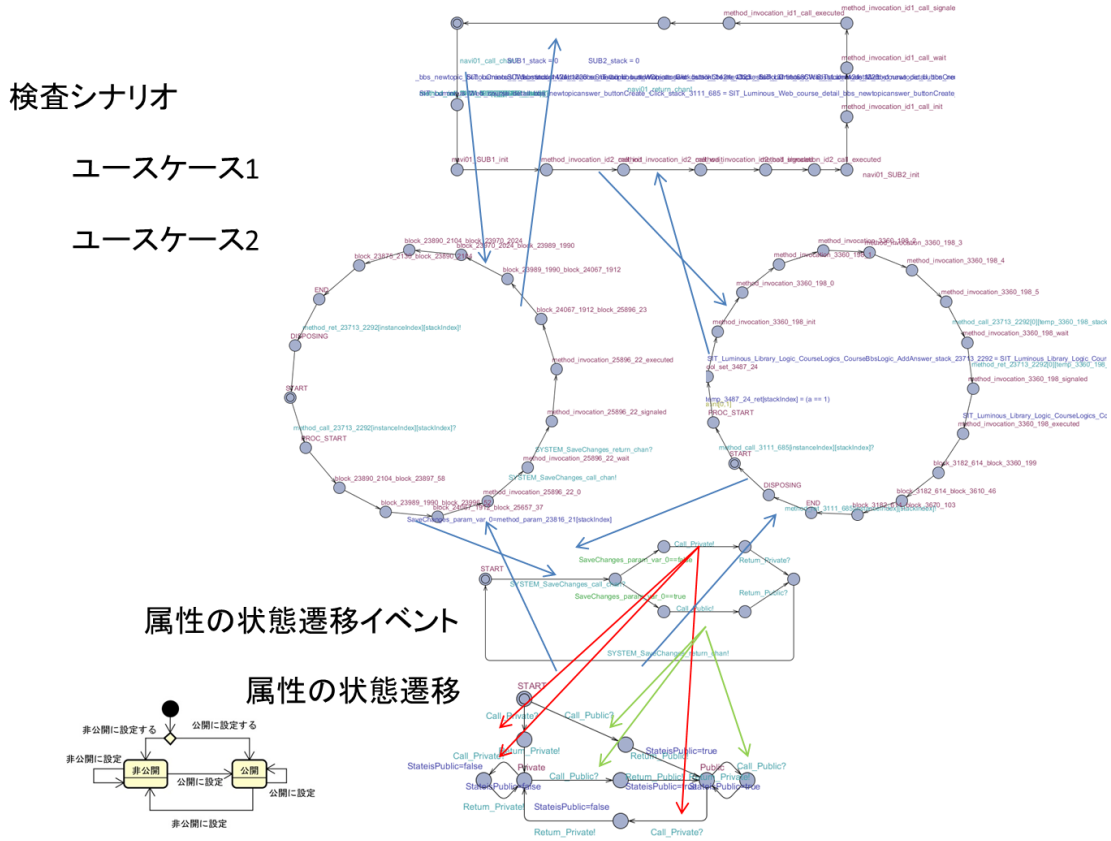


図 4 検査モデル

4．変換支援ツール

本研究では,つぎのユースケースをもつ変換支援ツールを 2012 年度と同様にソフトウェアの統合開発環境の 1 つである eclipse のプラグインとして開発する．旧バージョンの Source2UPPAAL ではベースモデルの可読性が低いという問題があり，ベースモデル定義における作業効率を改善するための機能も開発する．図 5 は変換支援ツールのユースケース図である．ここで,「仕様モデル」は，UML 記述によるユースケースに基づいて定義するセキュリティ要件定義テーブルおよびセキュリティ属性のモデルから生成される UPPAAL モデルである．

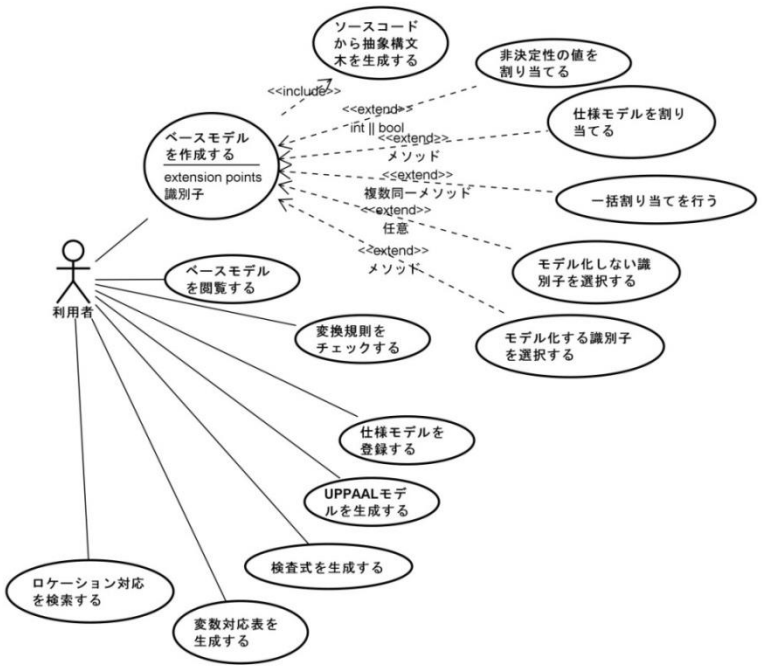


図 5 変換支援ツールのユースケース

５．LUMINOUS のセキュリティ要件

CC においては「資産の多くは情報の形をとり，情報の所有者が規定した要件を満たす IT 製品によって保存されたり，処理されたり，伝送されたりする．情報の所有者は，このような情報の可用性，まき散らし，および改変を厳しく管理し，対抗策によって資産を脅威から保護することが必要になろう.」と書かれている．LUMINOUS は資産としての表 2 の情報をもつ.

表 2 LUMINOUS の資産

資産		内容
コース		授業や演習の単位であり，以下の情報を扱う.
	連絡	コースに関する連絡事項である.
	教材	コースの教員が提供する教材ファイルまたはアーカイブである．コースの履修者またはそのグループに対して提供する．許可された学生がダウンロードすることができる.
	レポート	コースの課題に対して学生が提出するファイルまたはアーカイブである.
	評価	レポートに対する教員の評価レポートまたは評価の集計データである．評価レポートは指定された学生個人のみがダウンロードできる.
	アンケート	コースで行われたアンケートの質問内容や集計データである．教員の指定により，選択された質問と回答が公開できる.
	履修者	コースの履修者データである.
	BBS	コースにおける学生の質問に対して，教員が回答した話題である．質問や回答にはファイルを添付することができる.
	TA	コースに登録された学生のデータである.
	ストレージ	コースの教員およびTA間でのみ共有するデータであり，履修者には非公開である.

ISO/IEC27001 では情報セキュリティにおける機密性（Confidentiality）とは，アクセスを認可された者だけが情報にアクセスできることを確実にすることである．本研究では機密性の観点から，LUMINOUS の資産に対するセキュリティを検討する.

LUMINOUS における利用者の役割には，学生・主担当教員・副担当教員・ティーチングアシスタント（TA）の 4 つがあり，各役割に対する権限の設定により，まず，第一段階として，上記の資産へアクセスする機能の認可を定める．これはユースケースレベルの機能である．つぎに，機能を認可した後に，特定の情報へのアクセスをその情報の属性によって制御する．これがアクセス制御である．情報フロー制御はその情報の Read 機能の結果としてアクセスできる情報をその属性によって制御するものであると考えられる．

「コース」はその担当教員 ID と履修者学生の ID により，それ以外の利用者から「コース」内の資産を保護する．例えば「BBS」は学生が質問し，教員が回答することで生成される「話題」という情報をもつ．質問や回答にはファイルを添付することができ，当該学生や教員が許可した履修者は，それをダウンロードすることができる．教員の回答や添付ファイルが不必要に質問者以外の履修者に公開されてはいけない．さらに，学生の質問が公開されても個人が特定できてはならない．このために，教員は回答や添付ファイルに対して，その公開／非公開を設定することができる．この要件を表 1 におけるルールとして定義することで，モデル検査における検査式を導く．

ユースケースは，これらの資産を扱う機能である．システムの操作を通じて，LUMINOUS の要求分析モデル（ユースケース図，各ユースケースのアクティビティ図，クラス図）を定義する．この際，各ユースケースの使用するエンティティデータが上述のどの資産であるかを特定し，表 1 のもととなるデータを抽出する．サブジェクトとオブジェクトに対して，それぞれ必要なセキュリティ属性を定義する．さらに，セキュリティ属性の満たすべき性質をステートマシン図で定義する．CC のセキュリティコンポーネントを参考にして，設定したセキュリティ属性を用いてルールを定義する．この段階では，ソースコードから抽出したエンティティクラスのモデル図により，エンティティクラスとそのセキュリティ属性は仕様とソースコードが結びついた状態となる．そして，ユースケース上の UI 要素と UI コードを対応付けることにより，対象ソースコードのセキュリティ要件を検証することができるようになる．

６． LUMINOUS のセキュリティ要件の検証

変換支援ツール Source2UPPAAL を用いて，以下のように検査モデルを作成し，検査を行う．

1) 検査シナリオを「UI 要素と UI コードの対応表（以下，対応表と呼ぶ.)」を用いて，ソースコードからモデル化すべきメソッドおよびその定義場所の情報を抽出する．この情報から，ユースケースに対応する内部メソッドの特定を行い，図 6 のように「UI コード特定表」を作成する．

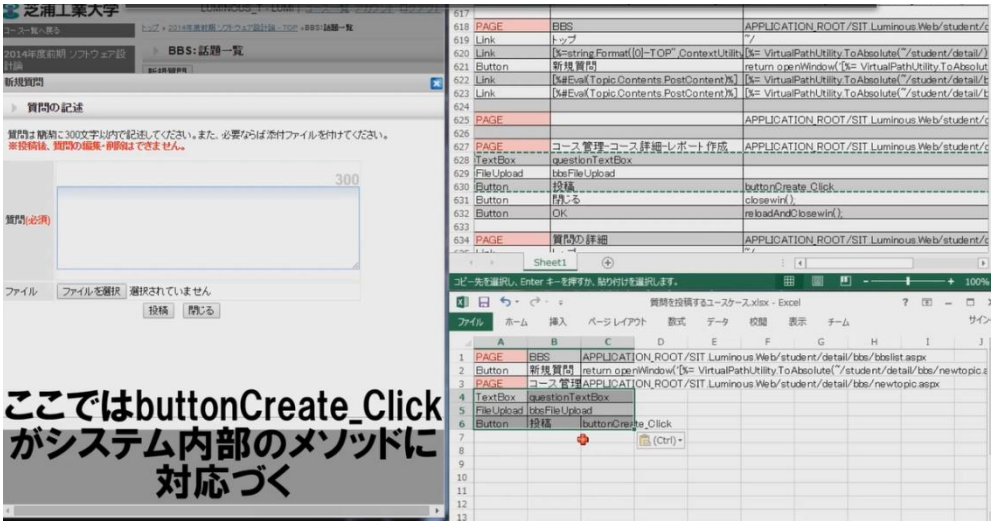


図 6 ユースケースに対応する内部メソッドの特定

2)　「UI コード特定表」をもとに，図 7 の変換支援ツール Source2UPPAAL を用いて，UPPAAL への変換対象となる検査シナリオを作成する．

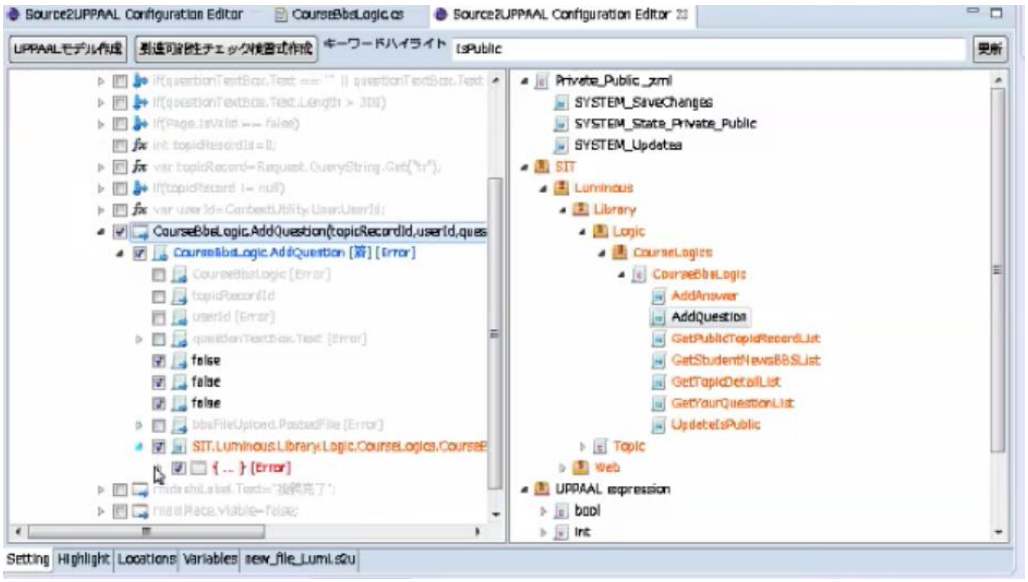


図 7　Source2UPPAAL

7.　反例解析

UPPAAL の検査で検査式に対する反例が得られた場合には，「想定される状態」になった場合と「想定されない状態」になった場合の両者の反例をグラフ化し，それらを比較して差異ができる部分を表示することにより反例解析を支援する．

検査において「想定されない状態」になるかの検査で反例がでた場合，それと対になる「想定される状態」になるかの検査を行う．単純に「想定される状態」の検査を行ってもまったく関係ない状態遷移になる可能性が高い．そのため「想定されない状態」の反例からシステムの仕様で特定できる識別子に着目して，その識別子に関係する UPPAAL の変数を

検査式に取り込む．システムの振舞いを特定できる状態をそろえ，類似した状態遷移になるため比較が可能になり，図８左側に示すように「想定される状態」「想定されない状態」の分岐ポイントを見つけることができる．

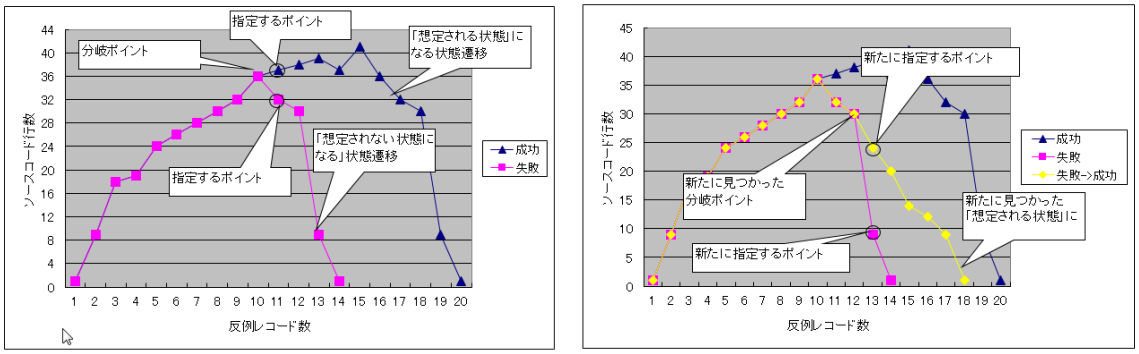


図 8 反例解析

しかし，その分岐後で最終的なシステムの振る舞いの状態が特定できるとは限らない．分岐ポイント以降の状態遷移を規定するため，分岐ポイントより先の通過指定ポイントを指定した式を追加した検査式を検査する．これを繰り返すことにより検査結果の精度が高められる．図８右図のように新たな分岐ポイントが見つかるかもしれず，最終的なシステムの振る舞いの状態が変化しない分岐ポイントが確定するまで検査を行う．

８．考察

本研究では，開発現場におけるシステムのリリースの判断やマイグレーション時に，システムが「仕様」を満たしているかの確認作業を高効率・高品質で実施するために，モデル検査技術を用いたソースコード検証を，形式手法のスペシャリストでなくても、その恩恵が受けられるような検査方法を構築することを課題として取り組んできた．図９に１で述べた目標に対する本研究での課題解決方法を示す．

①の課題「検査モデルと検査式の定義方法」については，ユースケースの UI 要素とセキュリティ要件の対象となるエンティティクラスとその属性をソースコードと仕様の関連付けに用いることで，ソースコードの意味解析を行わずに，システムの開発者以外でも容易に対応付けを定義することができた．

②の「システムモデルへの変換方法」においても，2012 年度の成果である Source2UPPAAL の問題点を改善し，上述の対応表に基づき，容易に検査モデルを設定できた．さらに，設定した検査モデルから変換した UPPAAL モデルはエラーなく検査を行えた．LUMINOUS のソースコードはロジック 36,352 ステップであるが，検査シナリオに基づいて，不要なステートメントをモデル化しないことにより，問題なく検査を実施することができた．

③の「反例解析支援方法」については，反例のグラフ化により，段階的に問題個所を絞り込む方法を提案できた．絞り込みの操作が手動のため，作業負担軽減のため，自動化を行う必要がある．

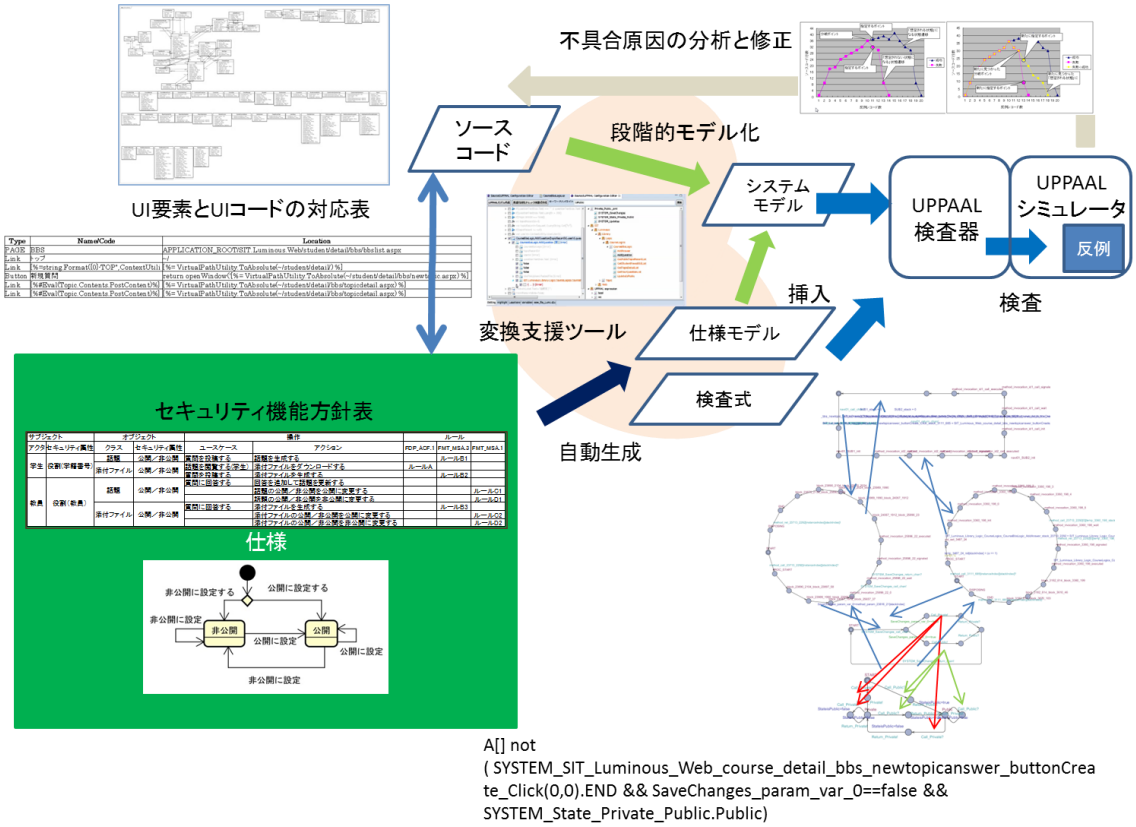


図 9 本研究での課題解決方法

ある程度の規模の仕様定義がないシステムから，システムの開発者ではない技術者が，システムの機能要件に対して，機能要件の振舞いと深くかかわるセキュリティ要件を検査することができた．

9．今後の課題と産業界への要望

形式手法のスペシャリストではない現場の開発者が，モデル検査の恩恵を受けられる方法を研究し，ユースケースにより定義した機能要件に対して，セキュリティ要件という非機能要件の 1 つをセキュリティ機能方針表により定義し，検査を実施するという方法を提案することができた．しかし，システムにはセキュリティ要件だけでなく，ハードウェアや通信の条件，使用性，信頼性，効率性，保守性等の品質に関わる要求等，様々な要求がある，これらの要件は，どのように整理すれば，本提案のようにモデル検査を使って，その要件を検証できるのか，また，機能要件においてもテストでは発見しにくい問題に対して，どのように仕様を整理すれば，モデル検査を活用できるのかといった問題を整理する必要がある．必ずしもモデル検査を利用する必要がなく，実環境でのテストが必要な場合もあると考える．技術が低コストで適用可能な基準を明確にすることが，産業界へ普及を図る上で重要である．また，普及を図るためには，こうした技術を試行する題材の提供と技術者による環境づくりを共同して行うことを産業界へ要望する．提案技術が開発現場で活用できるためには，高品質なツールの開発が必須であると考える．

１０．発表論文

[1] 松浦，小形，青木，谷沢，西村，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,SEC journal No.37，第 10 巻,第 2 号，2014，pp.8-15.（査読有　2012 年度委託研究に関連する研究）

[2] Saeko Matsuura, Yoshitaka Aoki and Shinpei Ogata, Practical Behavioral Inconsistency Detection between Source Code and Specification using Model Checking, ISSRE2014, pp.124-125, 2014.（査読有　委託研究に関連する研究）

[3] Yoshitaka Aoki and Saeko Matsuura, Verifying Security Requirements using Model Checking Technique for UML-Based Requirements Specification, Proc. of 1st International Workshop on Requirements Engineering and Testing, pp.18-25, 2014.（査読有　委託研究に関連する研究）

[4] 小形，青木，谷沢，松浦，ユースケースモデルに基づくソースコード検証のためのリバースエンジニアリング手法の検討－ASP.NET アプリケーションを事例として－，信学技報，vol. 114，no. 420，KBSE2014-42，pp. 19-24，2015.（査読なし　委託研究に関連する研究）

[5] 青木,松浦,反例からの検査式自動生成による不具合原因特定支援,信学技報，vol. 114，no. 128，KBSE2014-19，pp. 87-92，2014.（査読なし　委託研究に関連する研究）

[6] 青木，松浦，モデル検査における反例解析容易化支援，信学技報，vol. 113，no. 475，KBSE2013-79，pp. 1-6,2014（査読なし　委託研究に関連する研究）

[7] SIT　産官学連携研究交流会　ポスター展示　2015 年 3 月発表予定　（査読なし　委託研究に関連する研究）

[8] 青木，小形，谷沢，松浦，ユースケースモデルに基づくソースコード検証　-ASP.NET アプリケーションを事例として-，電子情報通信学会 KBSE 研究会　2015 年 3 月発表予定.（査読なし　委託研究に関連する研究）

1 研究の背景および目的

1.1 背景

ソフトウェア開発には要求定義・設計・実装・テスト・運用の工程がある．これらの工程において，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．要件定義段階で，これらの全てを考慮するわけではないが，レベルの異なるこれらの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．開発者はこれらの性質を各工程で作り込まなければならないが，適切に定義できているかを検証することは困難である．それは，開発工程において，仕様のドキュメント化が十分でない，ドキュメント化されていても自然言語による非形式的な記述であることに起因する．このため，形式的記述により検証が可能な形式手法の導入が注目されている[1]．形式手法の 1 つであるモデル検査とは「有限の状態空間を網羅的に調べて，与えられた性質が成り立つか否かを調べる」技術であり，図 1-1 に示すように，検査を行うモデル検査器と状態空間を探索するシミュレータから構成され，性質が成り立たない場合には反例を提示するモデル検査ツールが提供されている．ここで，利用者は検査したい対象のシステムモデルと検査式の両方を定義することで，モデル検査器を用いて，その性質が成り立つか否かを検査することができ，反例により，問題点を発見する．モデル検査は，このように，レビューやテストとは異なり，システムの振る舞いがある性質を満たすか否かを網羅的な探索により自動的に検証する方法であり，検証の手段として期待できる．我々は，モデル検査に着目して，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」において，上述のソフトウェア開発工程で考慮されるソフトウェアの満たすべき性質を「業務セオリー」と呼び，要件定義プロセスと保守プロセスの観点から，現場の開発者がモデル検査ツールを用いて様々なシステムの性質の検証を行なう方法を研究した．そして，モデル検査ツールの入力となるシステムモデルと検査式を現場の開発者が要求分析モデルやソースコードから生成する方法の見通しを得た．

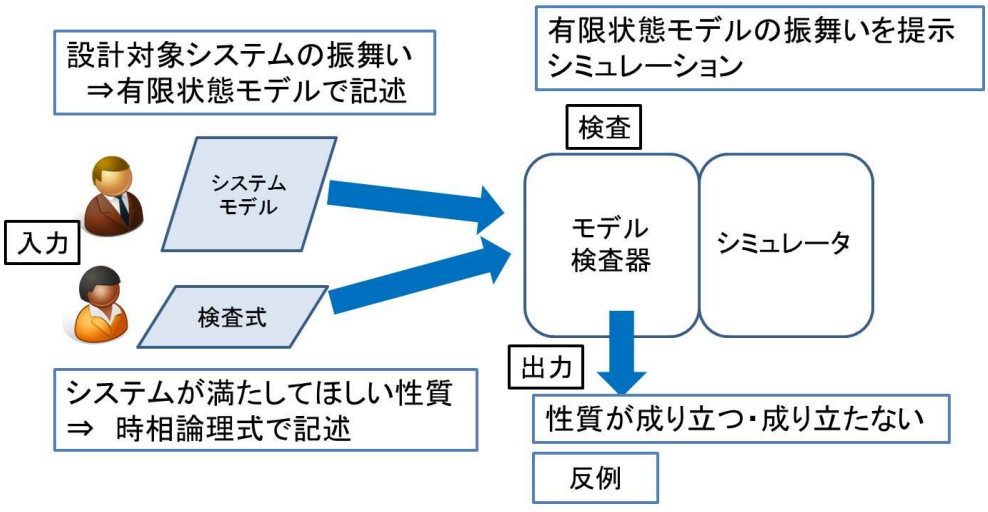


図 1-1　モデル検査

1.2 研究課題

開発現場においては，システムが仕様を満たしているかを判断する基準が，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から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．開発されたソースコードが研究の背景で述べた様々な性質，すなわち仕様を満たしているかをモデル検査により検証することは，システムのリリース判定やマイグレーションによるシステムの再構築時に，プロダクトの品質向上や作業効率の観点から重要であり，現場の開発者が適切かつ容易に検査を実施できることが求められている．しかし，モデル検査を用いるには，その入力となるシステムモデルと検査式を，特定の言語を用いて定義しなければならない．現場の開発者にとっては学習コストが大きいため，モデル検査ツールを直接操作しないことが一つの解決策となる．そのためには「検査したい性質」を「検査対象の振舞いモデル（ここではソースコード)」と対応付けて，開発者が理解しやすい形式で定義できるようにしなければならない．また，ソースコードは大規模であることから，モデル検査における状態爆発を回避して，検査したい性質を検査できるように振舞いモデルを適切に変換する方法を工夫することが求められる．

本研究では，現場の開発者がモデル検査ツールを直接操作せずに，モデル検査の恩恵を受けられる方法を開発するために，下記の研究課題を設定した．図 1-2 は研究のアプローチと解決しなければならない課題を示している．

①検査モデルと検査式の定義方法：

開発者が理解しやすい形式で「検査したい性質（機能要求・非機能要求)」を「検査対象の振舞いモデル（ソースコード)」と関連付けて定義することで，検査式を自動生成し，到達可能性や安全性の検査を行えるようにする．関連付けにはテストによく用いられるデシジョンテーブルで定義された仕様や非機能要求の 1 つであるセキュリティ要件定義を用いて検討する．

②システムモデルへの変換方法：

「検査対象の振舞いモデル」を状態爆発が起こらないようにシステムモデルへ変換する．検査したい性質に基づく抽象化方法を定義し，作業容易化のための自動化の方法を検討する．

③反例解析支援方法：

検査から得られた反例から問題点を発見する過程を支援する．

①から③の作業を現場の開発者が適切かつ容易に実施できるように支援する方法を検討するとともに，現場の開発者が容易に操作・理解できる変換支援ならびに反例解析支援ツールを開発することが本研究の課題である．

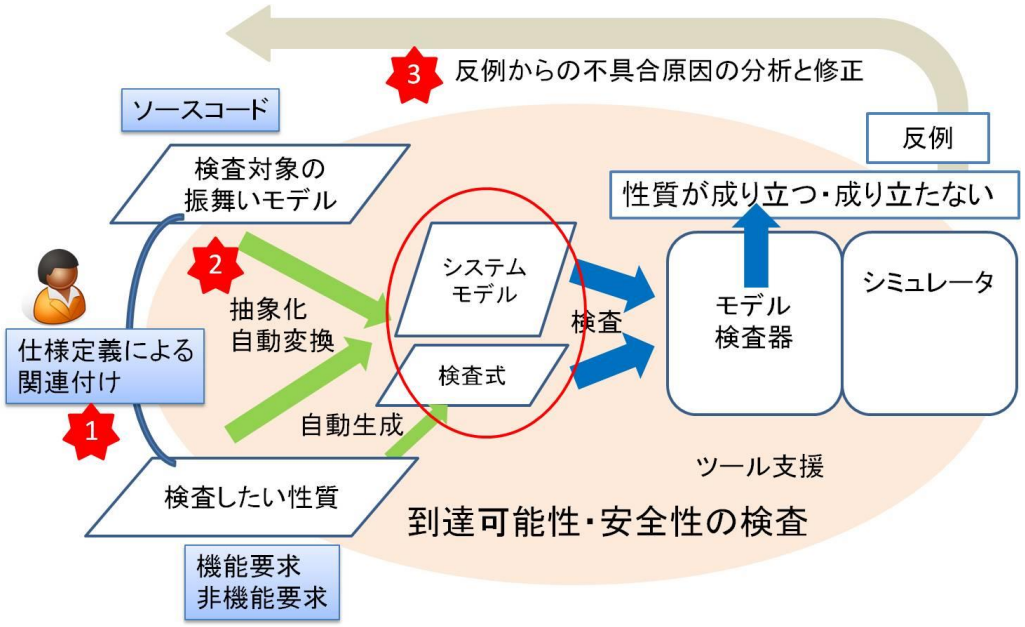


図 1-2　研究のアプローチと解決すべき課題

1.3 研究の意義

ソースコードからのモデル検査では，検査したい性質に依存したモデルの抽象化が大きな課題である．既存の研究は，API の使用方法が正しいかといった特定の検査や，ソースコードに注釈として検査したい性質を書き込むといった検査方法であり，現場の開発者が容易に仕様の検証に利用することは困難である．現場の開発者が検査したい性質を容易に定義できるように，自然言語記述の仕様書からデシジョンテーブルに整理したり，脅威分析に基づきセキュリティ要件を定義することによって，既存のシステムに対する仕様に基づく検査方式が策定できると考える．これらの仕様定義方法は，開発時に利用できるだけでなく，開発ドキュメントが不十分である場合の保守プロセスにおける検査に用いることができる．これにより，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ことが期待される．さらに，1.2 節の課題①～③を解決し、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、

- 非機能要件の１つであるセキュリティ要件を満たしているかを既存システムに対して容易に確認できる。
- 明確な仕様がわからないシステムのマイグレーション時に、既存システムの仕様を整理し、検査することで、マイグレーション時に新規のシステムの要件の精度を向上する。
- これらの検証によりプロダクトの品質向上、納期短縮、コストオーバの防止。

また、産業界の開発においては、仕様書とソースコードのトレーサビリティが不完全な場合が多く見受けられる。こうしたシステムを改善する際には、本来満たすべき仕様との齟齬がないかを確認する必要があり、本手法を適用できると考える。マイグレーション時

に、旧システムに対して確認した仕様を移行後のシステムに対して確認する際や、システムのリリース判定時に、セキュリティ要件等の確認を行うことに使用することを想定している。

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 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」の成果ならびに，2 つの関連研究「UML 要求分析手法」と「CC を用いたセキュリティ要求分析手法」に基づき説明する．また，検証事例の対象とする学習支援システムについても説明する．

2.1.1 研究の全体像

ソースコードを対象とするモデル検査では，ソースコードモデルからシステムモデルを作成する「抽象化」の技術が必要であり，データマッピングや述語抽象が知られている．システムモデルが「仕様」すなわち検査したい性質に関わる振舞いをモデル化していなければ検査は意味のないものになる．仕様に基づいた適切な，抽象化の方法を決定しなければならない．しかし，「仕様」は検査可能な形式で記述されることはほとんどない．また，現場の開発者が容易に定義方法を学習できることが重要である．本研究では，1.2 で述べた①～③の研究課題に対して，つぎのアプローチで，既存システムの仕様および検査したい性質とソースコードの抽象化手順を研究し，目標の達成を目指す．

- ①の研究課題に対して，論文[2]で提案した方法をブラッシュアップする．その方法は，自然言語で記述された機能仕様に基づき，仕様を構成するアクションと条件をデシジョンテーブルとして整理し，その機能の実行時にシステムの取るべき状態を明らかにし，定義されたデシジョンテーブルから，自動的に検査用モデルと検査式を生成し，ソースコードから得られた振舞いモデルをアクションによって同期させることで検査を実行するものである．
- ①の研究課題に対して，セキュリティ要件の定義は，2.1.2に示す「CCを用いたセキュリティ要求分析手法」を用いて，以下のように行う．既存システムのユースケース（システムの境界に見えているアクション）を2.1.2に示す「UML[3]要求分析手法」を用いて定義する．一方，データベースのデータとオブジェクトのマッピング定義から，ソースコード内でのデータのオブジェクモデルを特定する．UML要求分析モデルから定義されるセキュリティ機能方針表や，ユースケースの権限表に基づき，セキュリティ属性とその更新メソッドをデータオブジェクトに対して特定する．これによりソースコードの振舞いモデルの抽象化方法と検査モデルを決定し，セキュリティ機能方針表より検査式を自動生成して，検査を実行する．この際に，セキュリティ機能方針表に現れるアクションに対応するソースコード上のメソッドを特定するために，ソースコードの静的解析により情報を取得する．
- Source2UPPAAL[4]は，図2-1に示すようにJavaのソースコードから，値の割り当て，メソッドの割り当て，ユーザ定義モデルに基づく抽象化により，UPPAAL[5]のシステムモデルを段階的に生成する機能をもつツールである．実装の構成は図2-2のとおりである．本研究では，②の研究課題に対して，上記の仕様定義方法に従い，仕様に基づく抽象化方法の自動化およびその支援方法を検討する．また，事例として本学で稼働中の学

習支援システムLUMINOUS※に適用するため，C#からのUPPAALモデル生成を実現する．

※学習支援環境 LUMINOUS は芝浦工業大学で開発した e-Learning のプラットフォームとなる授業支援システム（Learning Management System）であり，教材の配付，レポートの受理，アンケートの実施，学生への連絡等に加え，学生参加型の学習環境として，日常の授業運営上の問題を改良したシステムであり，これまでに本学 3 学科において 120 件程度の講義・演習等に活用している．

- モデル検査ツールから提示される反例は状態遷移の経路である．検査モデルの各状態はソースコードのステートメントと対応はしているが，そのまま見ても問題点を読み解くことは難しい．③の課題に対して，反例，すなわち失敗している経路と成功している経路の比較により，失敗している場合の通過点を限定しながら，再検査を行い，失敗経路の通過点を順次限定することで，失敗の原因となるソースコード上のステートメントを特定することを検討する．
- 事例として，本学で稼働中の学習支援システム LUMINOUS が，そのセキュリティ要件を満たしているかを，Common Criteria [6]に基づき検査することを想定して，仕様とソースコード要素とを対応づける機能，検査したい性質に基づき，ソースコードから UPPAAL のシステムモデルへの変換方式を指定する方法，検査結果から問題箇所を詳細に特定することの支援機能等を研究開発する．
- 研究に必要な情報を得るために，国際会議等に参加し，最新情報の入手や参加者とディスカッションを行う．また，LIMINOUS の専門家を招致することにより，必要な情報を得る．

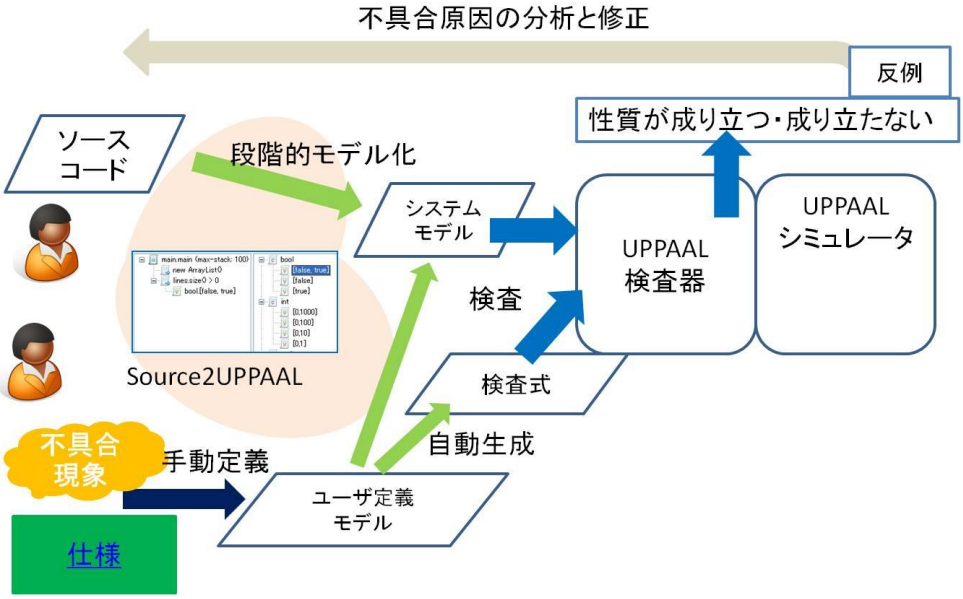


図2-1　Source2UPPAALにおける検査方法

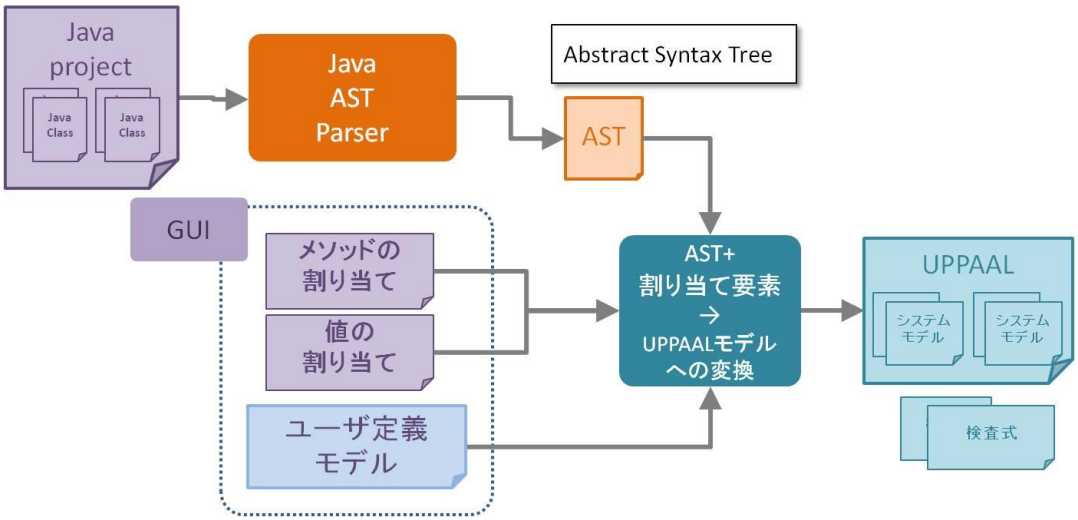


図2-2　Source2UPPAALの構成

2.1.2 関連するこれまでの研究について

(1) UML要求分析手法

論文[7,8,9]で提案した本手法では，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．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2-3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*[10]を利用する．

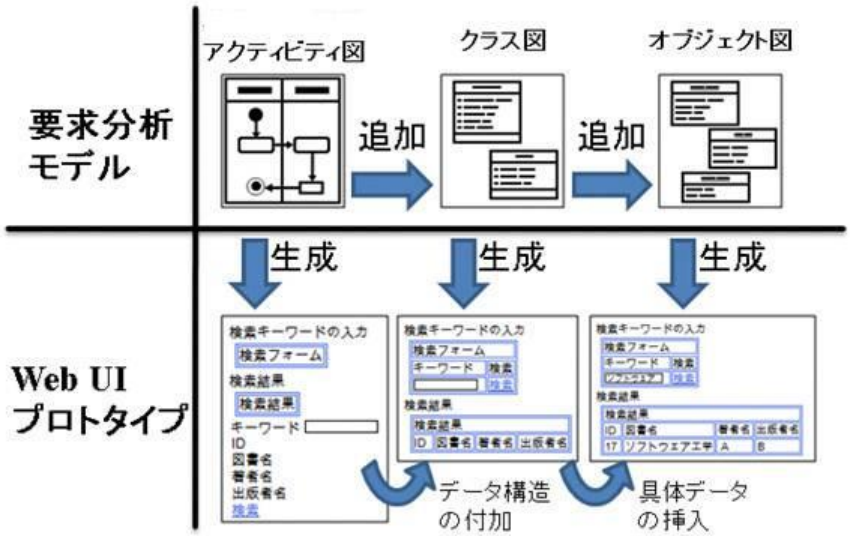


図 2-3　段階的な Web UI プロトタイプ自動生成

(2) CCを用いたセキュリティ要求分析手法

論文[11,12,13]で提案したCommon Criteria(CC)を用いたセキュリティ要求分析手法は，セキュリティの知識をCCを用いて補いながら，UML要求分析手法によって定義された要求分析モデルに対するセキュリティ要件を定義する方法である．セキュリティの専門家ではない，一般の開発者がセキュリティ要件を漏れなく定義することは困難であることからCCに着目する．CCは，情報セキュリティの国際評価基準（ISO/IEC15408）であり，公開されているバージョンには、” CC/CEM v3.1 Release4 ”がある．CCは，Part 1：概要，Part2:セキュリティ機能コンポーネント，Part3：セキュリティ保証要件から構成されており，セキュリティ機能コンポーネントには利用者データ保護，通信，暗号サポート，認証と識別，セキュリティ管理等の11のクラスが定義されている．「概要」に定義された用語から，「セキュリティ機能コンポーネント」を定義する「セキュリティ機能方針(SFP)」をモデル化し，セキュリティ機能コンポーネントを解釈して，対象システムのモデル要素との対応を図2-4のように定義する．図2-4は「SFPは複数の規則から構成されており，規則はあるサブジェクトが実行主体となる操作に対して適用される．操作はオブジェクトをその対象とし，規則はサブジェクトとオブジェクトの特性として定義されるセキュリティ属性によって制御される」ことを表している．一方，対象システムはUML要求分析手法を用いて分析する．UML要求分析モデルでは，１つのユースケースを表すアクティビティ図において，SFPで制御対象となり得る操作にはアクションが，操作の実行主体であるサブジェクトにはアクターが対応する．さらに，操作の対象であるオブジェクトはオブジェクトノードとして定義され

ている．そこで，対象システムのSFPをサブジェクト・操作・オブジェクト・サブジェクトのセキュリティ属性・オブジェクトのセキュリティ属性からなる規則の集合として定義する．

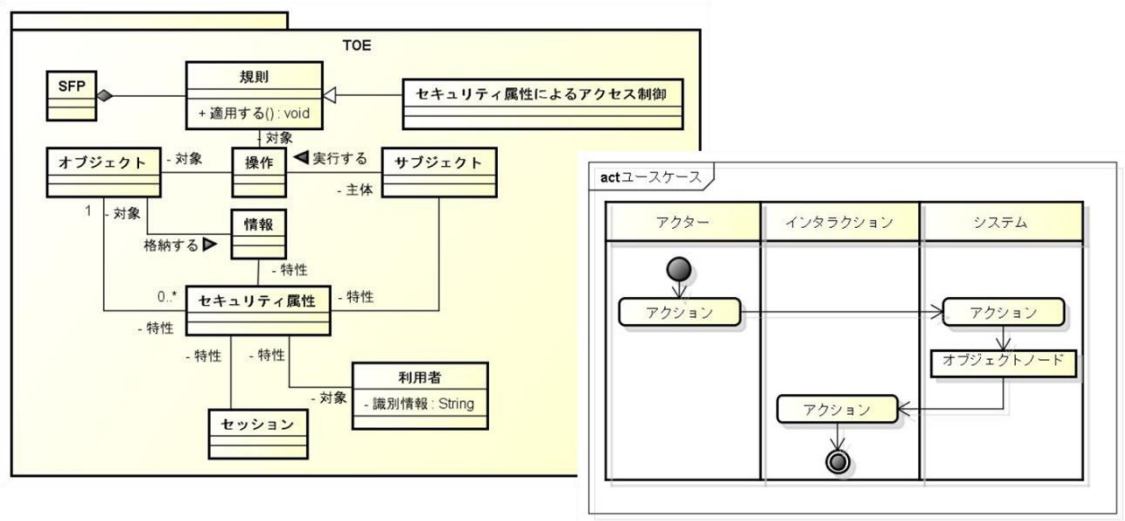


図 2-4　セキュリティ機能方針と対象システムのモデル

CC のセキュリティ機能コンポーネントはクラス，ファミリ，コンポーネントの階層構造を持ち，コンポーネントは複数のエレメントと他のコンポーネントとの依存関係を持つ．図 2-5 は利用者データ保護クラス FDP のファミリであるアクセス制御機能 FDP_ACC のコンポーネントの依存関係である．ここで FDP は利用者データ保護（user Data Protection）のクラスを表す記号であり，このクラスに属するファミリの名前は FDP_*と表される．

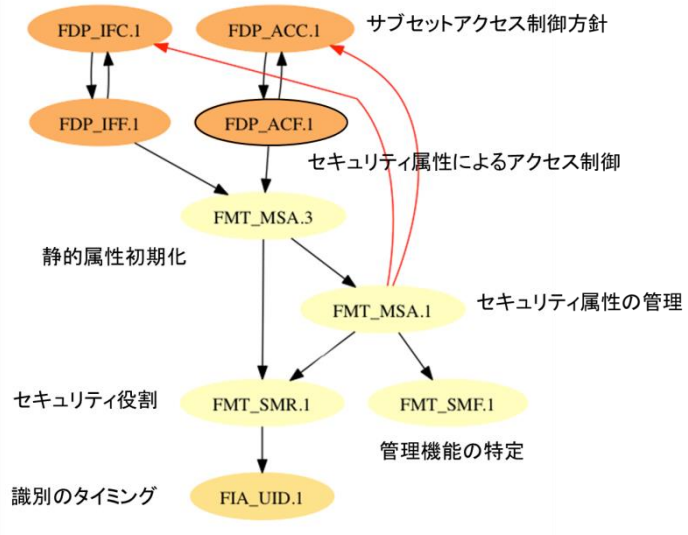


図 2−5　コンポーネントの依存関係

このことから，例えば，アクセス制御のセキュリティ要求をアクセス制御機能のファミリ（FDP_ACF）に基づき定義する場合には，依存関係を持つ全てのコンポーネントに対して，

規則を整理することで，セキュリティの深い知識を持たない開発者でも，SFPを漏れなく定義することができる．なお，CCの構成や，脅威の分析の詳細は上記の論文に記載されている．表2-1は，本学で稼働中のLMS(Learning Management System)であるLUMINOUSのBBS機能のモデルから定義されたSFPである．2つのアクターである学生と教員に対して，ユースケースは「学生が質問を投稿する」「教員が質問に回答する」「学生・教員が話題（質問・回答のセット）を閲覧する」である．本機能のセキュリティ要求は，教員の回答や添付ファイルが不必要に質問者以外の学生に公開されないことと，学生の質問が公開されても個人が特定できないことである．教員は回答や添付ファイルに対して，その公開／非公開を設定できる．

表 2-1　セキュリティ機能方針

サブジェクト		オブジェクト		操作		ルール		
アクタ	セキュリティ属性	クラス	セキュリティ属性	ユースケース	アクション	FDP_ACF.1	FMT_MSA.3	FMT_MSA.1
学生	役割(学籍番号)	話題	公開／非公開	質問を投稿する	話題を生成する		ルールB1	
		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	ルールA		
				質問を投稿する	添付ファイルを生成する		ルールB2	
教員	役割(教員)	話題	公開／非公開	質問に回答する	回答を追加して話題を更新する			
					話題の公開／非公開を公開に変更する			ルールC1
					話題の公開／非公開を非公開に変更する			ルールD1
		添付ファイル	公開／非公開	質問に回答する	添付ファイルを生成する		ルールB3	
					添付ファイルの公開／非公開を公開に変更する			ルールC2
					添付ファイルの公開／非公開を非公開に変更する			ルールD2

(3) 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」の位置づけ

2012 年度の研究では「処理手順の定義に対し，開発段階で考慮する制約を処理対象データの取り得る状態とその変化として捉え，その変化の操作と条件を処理手順の定義と段階的に対応付けることにより，段階的に検査を行う」という業務セオリーと検査方法の考え方に基づき，その支援ツール UML2UPPAAL と Source2UPPAAL の開発を行った．本研究では，「検査したい性質」である業務セオリーを，テストでよく用いられるデシジョンテーブルや情報セキュリティの国際評価基準（ISO/IEC15408）である Common Criteria に基づくセキュリティ機能方針表といった形式の仕様定義とする．ここでは，UML2UPPAAL で用いた要件定義方法に基づき仕様定義を検討する．そして，ソースコードがその仕様を満たしているかを検査する検査方法と支援ツールを Source2UPPAAL のソースコード変換方式と支援方法の考え方を基に研究開発する．

2.1.3 研究目標

(1) モデル検査によるソースコード検査の問題点

ソースコードの不具合検出にモデル検査を使用する研究は多い．Java Pathfinder[14]は実行可能な Java バイトコードよりプログラムを検証してデッドロックとアサーションエラーを検出する.Pathfinder は複雑で大規模なソースコードに対しては″状態爆発″の注意が特に必要である．Markosian [15]は Java Pathfinder により NASA のフライトシミュレータ開発時に検査を行っている．リファクタリングにより状態数を減らすことにより効率的にモデル検査を行う手法を説明した．この手法は有用な 技術であるが，ユーザが定義する仕様に基づくモデルで検証することは難しい．

Bandera[16]は抽象化とスライシングにより，Java プログラムのソースコードから，コンパクトな有限状態モデルの自動抽出が可能である.出力モデルは,SPIN[17]や NuSMV [18]等の既存のモデル検査ツールで検証することができる．このアプローチは、″状態爆発″には有効であるが，検査者にはモデル検査の深い知識が要求される.

モデル検査の非専門家が利用するという観点での研究も行われている[19]．この研究では，システムの振舞いの重要な側面を定義したパターンと適用すべきスコープを予め用意し，それを会話形式による入力やドラッグ＆ドロップによる選択を行うことにより検査式を作成する試みをしている．検査式の記述を助けることには確かに有効であるが，それを利用するためにはパターンの意味やスコープの意味を理解する必要があり，これはモデル検査の非専門家にとっては困難である.

ソースコードのモデル検査では，ソースコードの制御フローに基づいてモデルを自動生成することができる．しかし，検査を行うには次の 2 つの問題点がある．第一に，ソースコードは規模が大きく，その制御構造は複雑であるため，モデルの状態爆発が発生し，検査ができないことである．第 2 には，デッドロックや到達可能性の検査は，前者は何も指示する必要はなく，後者も到達すべき状態，すなわち，ソースコードの位置を指定すれば検査が可能である．しかし，アプリケーションの性質に依存した検査を行う場合には，ソースコードに記述されている識別子を用いて，検査式を定義しなければならず，ソースコードを読み解く必要がある.

第一の問題に対しては，モデルの抽象化により状態数の削減を行うが，抽象化によっては，検査したい性質が隠れてしまうことがある．第 2 の問題に対しては上述のとおり，ソースコードを読み解いて，検査式を書く方法やアプリケーション固有ではない性質の検査が行われている．これらの作業は，開発現場の技術者が容易に行えるものではない.

(2) 研究目標の設定

1.2 で述べた各課題に対する到達目標はつぎのとおりであり，図 2-6 がソースコードを対象とした本研究における研究アプローチを示している.

1)課題①「検査モデルと検査式の定義方法」については、検査したい性質（機能要求・非機能要求）を「システムが、ある条件のもとで、ある正当でない状況に陥ることが決して起こらない 」という安全性の観点から検査できるように、デシジョンテーブルおよびセキュリティ機能方針表の形式で整理する方法を定義する。

2)課題②「システムモデルへの変換方法」については、1）の方法に基づき、ソースコードをモデル検査用のシステムモデルに段階的に変換する方法を定義し、支援ツールを開発する。モデル検査ツールは UPPAAL を用いる。

3)課題③「反例解析支援方法」についてはモデル検査ツールから得られる反例を解析して、仕様を満たしていない原因を特定する方法を検討する。ここでは状態を特定する変数を導入して検査式を自動生成し、原因の絞り込み方法を検討する。

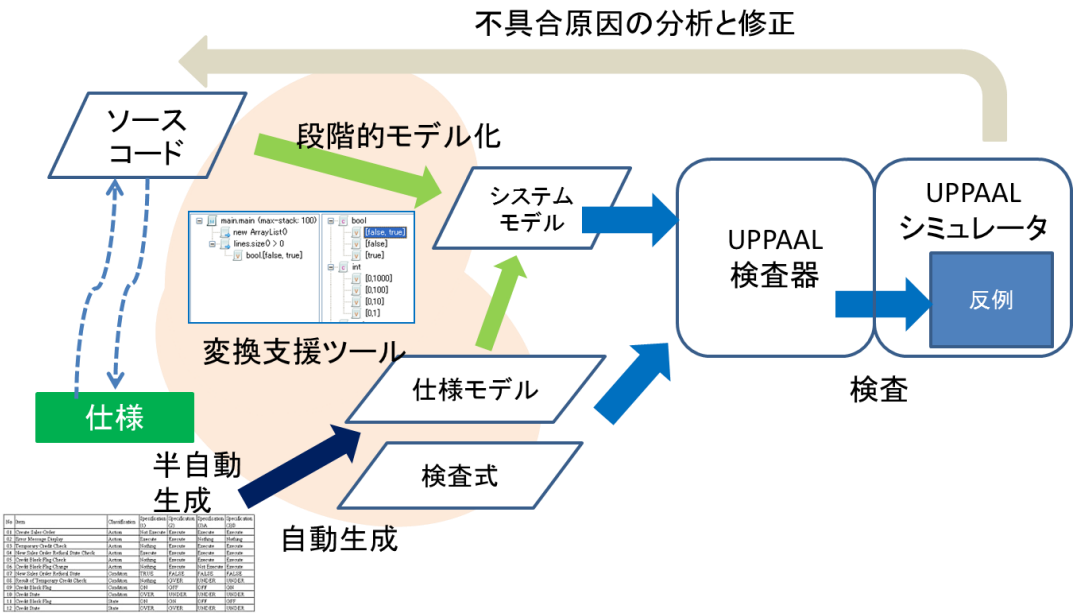


図 2−6　ソースコードを対象とした研究のアプローチ

本研究では,本学で実際に稼働している学習支援システム LUMINOUS をセキュリティ要件の検証対象とする．本システムは，学生が開発したシステムを基に，再開発し，機能追加や権限の追加により拡張するという開発形態を取ってきた．仕様書としては，利用シナリオや，一部には UML モデルが存在する．規模は以下のとおりである．

- ユースケース数　65
- アクターとしては，学生・主担当教員・副担当教員・TA の 4 つの役割がある．
- 対象とするコースの種類は 2 種類
- ソースコードの規模　：　ロジック 36,352 ステップ，画面 25,973 ステップ
- 現在のユーザ数は 11,183
- 扱っているデータはレポートファイル数 69,313，教材ファイル数 1,375

これらのすべての機能に対して，セキュリティ要件としては，学生・主担当教員・副担当教員・TA に加えて，学生および教員に設定できるグループの役割とコースの種類の組み合わせに対して，アクセス制御，情報フロー制御，認証，プライバシ等の Common Criteria の 11 のセキュリティ機能コンポーネントの内の 4 つのクラスに関係する要件を扱っている．これらの要件は，個々の機能と密接な関係にあり，システム内部の実装の複雑化の要因となっている．セキュリティ要件はシステムの持つ資産に対する脅威への対策として定義されると考えるが，LUMINOUS では学生のレポート・評価等の個人情報を扱うため，特に上記のセキュリティが要求されている．こうした個人資産を保護する必要のあるシステムは多く，セキュリティ要件としては一般的に存在するものであり，他システムにおけるセキュリティ要件の検証にも適用できるものであることから，本研究における検証対象としては適切であると考える．

以上のことを踏まえ，到達目標に対し，研究目標を以下のように設定した．具体的には，

2.1.2 で述べた UML 要求分析手法や CC を用いたセキュリティ要求分析手法および 2012 年度の研究成果を基に研究を進める.

(1) 仕様定義に基づく検査モデル・検査式の生成方法の定義
(2) 変換支援ツールの設計と開発
(3) LUMINOUS のセキュリティ要件の定義
(4) LUMINOUS のセキュリティ要件の検証
(5) 反例解析方法の検討

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

- 2.1.3 節で述べた研究目標について，これまでの研究内容を踏まえ，研究メンバーで打ち合わせを行い，情報の共有を図った.
- 研究目標とスケジュールに沿って，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記述実験に基づき，モデル検査ツールを用いて検証実験を行う.
  - 実験結果を分析し，方法の問題点を議論を通じて明らかにする.
- 予備実験として本学が教育用に開発した長方形エディタの検証を例に検査を実施しながら検査方式の見直しを行った。
- LUMINOUS の情報収集のミーティングを LUMINOUS 開発者と研究メンバーで実施した。
- LUMINOUS からの仕様定義を行うため，実験用環境を構築し，モデル定義を実施した. 本作業は主に，本研究室学部生 5 名（川合怜，池田滝飛，小林雅弥，杉田宗一郎，藤田朋邦）および修士課程大学院生 2 名（加藤真，松井浩二）が行った.
- LUMINOUS のソースコード解析については，主に小形研究員が担当した.
- 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，論文発表を行った.

2.2.2 発表論文

2014 年 7 月から 12 月に，下記のとおり，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] 松浦，小形，青木，谷沢，西村，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,SEC journal No.37，第 10 巻,第 2 号，2014，pp.8-15.（査読有）

抄録：

近年，システムの利用シナリオの多様性と広がりに加えて，ハードウェアやアーキテクチャの多様性が増加しており，手戻りを防ぎ，高品質なソフトウェアを開発するた

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

[2] Saeko Matsuura, Yoshitaka Aoki and Shinpei Ogata, Practical Behavioral Inconsistency Detection between Source Code and Specification using Model Checking, ISSRE2014, pp.124-125, 2014. （査読有）

Abstract:

To achieve practical use of model checking, we propose a method to find the discrepancy between the behavior of the source code and the specifications written in UML by using a decision table.

Model checking is an attractive and effective approach to verify the behavior of the system different from usual testing. However, there are some problems for general developers to use it practically. We propose a method to find the discrepancy between the behavior of the source code and the specifications written in UML by using a decision table and model checking technology. In this paper, we discuss this method by using an example verification process of a source code which is assigned in a class of our university with the UML-based specification.

[3] Yoshitaka Aoki and Saeko Matsuura, Verifying Security Requirements using Model Checking Technique for UML-Based Requirements Specification, Proc. of 1st International Workshop on Requirements Engineering and Testing, pp.18-25, 2014. （査読有）

Abstract:

Use case analysis is known to be an effective method to clarify functional requirements. Security requirements such as access or information control tend to increase the complexity of functional requirements, and therefore, need to be correctly implemented to minimize risks. However, general developers find it difficult to correctly specify adequate security requirements during the initial phases of the software development process.

We propose a method to verify security requirements whose specifications are based on Unified Modeling Language (UML) using the model checking technique and Common Criteria security knowledge. Common Criteria assists in defining adequate security requirements in the form of a table. This helps developers verify whether

UML-based requirements analysis models meet those requirements in the early stages of software development. The UML model and the table are transformed into a finite automaton in the UPPAAL model checking tool.

[4] 小形, 青木, 谷沢, 松浦, ユースケースモデルに基づくソースコード検証のためのリバースエンジニアリング手法の検討－ASP.NET アプリケーションを事例として－, 信学技報, vol. 114, no. 420, KBSE2014-42, pp. 19-24, 2015.
抄録：
ソフトウェア開発において, 要求仕様書とソースコードの対応関係を保持することは容易ではなく, 最終的にソースコードが要求仕様を満たしていることの検証は困難である. 本研究では, UML によるユースケースモデルに沿って, モデル検査技術によるソースコード検証を行いやすいように, ソースコードから検証に必要な情報を抽出する手法の実現を目指す. 本論文では, 業務系ソフトウェアを対象に Common Criteria によるセキュリティ要求の検証支援を目標として, 大学で実稼動する授業支援 ASP.NET アプリケーションを事例に, 静的解析によるソースコードのリバースエンジニアリング手法を検討し, その実現の可能性と課題について考察する.

[5] 青木,松浦,反例からの検査式自動生成による不具合原因特定支援,信学技報, vol. 114, no. 128, KBSE2014-19, pp. 87-92, 2014.
抄録：
モデル検査はシステムの振る舞いを検証するためには有効な技術である. 我々はモデル検査技術を利用して仕様書とソースコードの振る舞いの不一致を発見する手法を提案してきた. この手法により不一致の有無は明示されるが, その不一致が起こる原因を特定するためには反例を解析する必要があった. 出力される反例は状態遷移のトレースであり, その意味を理解するには複雑な状態遷移を追跡して意味を読み取らなければならないため、一般的な開発者にとっては解析が難しい. 本稿では, 出力された反例にあるシステムの振舞いの状態を抽出して, 検査式に付加することにより状態遷移の特定を試みる. 反例が出たら, 再度システムの振舞いの状態を抽出して検査式に付加して検査するサイクルを繰り返すことにより、問題となる状態遷移の特定を容易にする手法を提案する.

[6] 青木, 松浦, モデル検査における反例解析容易化支援, 信学技報, vol. 113, no. 475, KBSE2013-79, pp. 1-6,2014
抄録：
モデル検査はシステムの振る舞いを検証するためには有効な技術である. 我々はモデル検査技術を利用して仕様書とソースコードの振る舞いの齟齬を発見する手法を提案してきた. この手法で導かれた検査式により齟齬の有無は明示されるが, 齟齬があった場合, その原因を特定するには反例を解析する必要がある. しかし, 出力される反例は状態遷移のトレースであり, その意味を理解するには複雑な状態遷移を追跡して意味を読み取らなければならないため, 一般的な開発者にとっては解析が難しい. 本稿では, 出力された反例を基に状態遷移を制限する式を生成し, 検査式に付加して再検査するというサイクルを繰り返すことにより, 問題となる状態遷移の特定を容易にする手法を提案する.

[7] SIT　産官学連携研究交流会　ポスター展示　2015 年 3 月発表予定

[8] 青木，小形，谷沢，松浦，ユースケースモデルに基づくソースコード検証 -ASP.NET アプリケーションを事例として-，電子情報通信学会 KBSE 研究会 2015 年 3 月発表予定.

2.2.3 進捗状況実績

進捗状況表（実績）は表 2-2 のとおりである.

表 2-2 進捗状況表（実績）

研究機関名	芝浦工業大学						研究責任者名			松浦佐江子	
研究テーマ名	2014年度ソフトウェア工学分野の先導的研究支援事業 研究テーマ名 保守プロセスにおけるモデル検査技術の開発現場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	2月	進捗状況	備考
研究目標(1)仕様定義に基づく検査モデル・検査方式の生成方法の定義	実										完了	
研究目標(2)変換支援ツールの設計と開発	実										完了	
研究目標(3)LUMINOUSのセキュリティ要件の定義	実										完了	
研究目標(4)LUMINOUSのセキュリティ要件の検証	実										完了	
研究目標(5)反例解析支援ツールの設計・開発	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	

2.2.4 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 COMPSAC2014（7 月）
- 国際会議 RE2014・RET2014 （8 月）
- 国際会議 ICSEA2014（10 月）
- 国際会議 ISSRE2014（11 月）
- 知能ソフトウェア工学研究会（7 月，1 月）
- ソフトウェア工学の基礎ワークショップ（12 月）

国際会議 COMPSAC2014

COMPSAC2014(the 38th Annual IEEE International Computer Software and Applications Conference)はIEEEが開催する今年で38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．今年のテーマは The Integration of Heterogeneous and Mobile Services in Smart Environments. であり，これからのデジタル社会における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，基調講演，パネルおよび Validation & Verification,Spec & Requirements などのセッションを聴講した.特に David Harel が Programming Liberated: Playing and Talking Behavioral Scenarios というタイトルで行った基調講演では，ソフトウェアで実現したい要求はソフトウェアを利用する人間から発生するものである一方，そのソフトウェアを実行するのはコンピュータであこ

とから生じるソフトウェア開発の難しさに対する，State Chart から Behavioral Programming によるアプローチならびに最近の研究を紹介していたが，本プロジェクトのテーマも含めたわれわれのモデル駆動ソフトウェア開発へのアプローチの適切さを確認することができ，大変有意義であった．また，聴講した講演は，本プロジェクトのテーマでもある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．本研究室 M2 生の加藤真さんが Improve User’s Security Literacy by Experiencing Behavior of Pseudo Android Malware というタイトルで Android のマルウェア対策に関する論文発表を行った．

国際会議 RE2014・RET2014

国際会議 RET2014 および RE2014 において要求工学に関する最新動向の情報を入手し，要求の検証について議論するため，会議に発表・出席したものである．RET2014 は，要求工学分野とテスト分野をどう連携すべきかを研究発表・質疑応答およびグループ討論によって議論するワークショップである．学術界および産業界の参加者の混成３グループにて行った討論では，学術界の参加者が，要求の適切な抽象化やモデル化が重要であるとの主張が多い一方で，産業界の参加者はテストの選択・実施を如何に適切・効率的に行えるかが興味の中心であった．双方が異論なく合意したトピックは，Validation&Verification の自動化であり，我々の研究にまさに合致することが分かった．研究発表では，我々の研究の関連研究となるべき要求モデルからテストケースを生成する手法がいくつか紹介されていた．RE2014 では，ソフトウェア開発における要求工学全般をトピックとしたトップカンファレンス（採録率 27%）であり，200 人規模の参加があった．Industrial セッションにて産業界の現状を調査する発表は無論，Horkoff 氏の Supporting Early Decision-Making in the Presence of Uncertainty の発表が興味深かった．これはゴール指向モデルの形式化であり，不確かなゴールを含むモデルからゴールを絞り込む意思決定の支援法を提案している．我々が研究対象とする保守プロセスでは，保守の要因がそもそも顧客のゴールが変わることに起因することが少なからずある．例えば，不確かなゴールの下，システムのライフサイクルの中でゴールがどう変わったかまでトレースできれば，我々の研究において行うべきモデル検査の内容もそれに応じて柔軟に変更できる可能性があるであろう．

国際会議 ICSEA2014

ICSEA2014(The Ninth International Conference on Software Engineering Advances)がフランスのニースで開催された．Advances in fundamentals for software development, Agile software techniques, Improving productivity in research on software engineering, Advanced design tools for developing software 等のテーマに関する研究発表やパネル討論が行われ，基調講演の他，これらの論文セッションを聴講した．Prof. Dr. Alain Abram 氏 の基調講演 Software Estimation: Practical Insights & Orphean Research Issues では，プロダクトの見積もりデータが取得可能なプロダクトの開発が有益な見積もりを行う上で重要であるとの話があった．Philipp Hell 氏の基調講演 Research Challenges and Opportunities for the Development of Complex Systems: An Industrial Perspective では，エアバスのような企業の大規模開発における形式手法を取り入れた開発手法について

の講演があり，形式手法を取り入れることで品質向上が望めるが，現状のツールは支援する箇所ごとのツールであり，統合した利用が困難であるという話があった．これらの講演において，形式化による定義と検証の統合が重要であることや，方法論に対して，ツールの使いにくさが開発者の障害となっており，既存のワークフローとツールの環境の統合が重要であるといったわれわれの目指している方向性が望まれていることが確認でき，大変有意義であった．また，本研究室 M1 生の松井浩司さんが MDD for Smartphone Application with Smartphone Feature Specific Model and GUI Builder というタイトルでスマートフォンをターゲットとしたモデル駆動開発手法に関する論文発表を行った．

国際会議 ISSRE2014

信頼性工学の国際会議である ISSRE2014 にて，機能および信頼性に関してソフトウェアを仕様に基づき検証する手法の発表と関連する研究の聴講を行った．本発表における提案手法は,ソフトウェアと仕様が存在する前提において,それらの間のベリフィケーションを,モデル検査技術を用いて如何に効率的かつ現実的に検証するかの課題に取り組んだものであるため，保守プロセスおよびモデル検査技術に強く関連したものである．提案手法の特徴は，形式的に仕様を扱う難しさと，そこからモデル検査するためのクエリを導く難しさを解決するために，UML ベースの要求仕様書から，モデル検査上のクエリに変換できるデシジョンテーブルを系統的に導出する点である．また，本研究に関連する研究として，Model-Based Testing 等のセッションがあった．

知能ソフトウェア工学研究会（7 月　富良野）

電子情報通信学会知能ソフトウェア工学研究会がソフトウェアサイエンス研究会および情報処理学会ソフトウェア工学研究会との合同研究会として富良野文化会館で行われた．松浦研究室の博士課程の学生 1 名が，本プロジェクトのテーマであるモデル検査ツールの利用方法について「反例からの検査式自動生成による不具合原因特定支援」というタイトルで研究発表を行った．さらに，われわれの研究の基礎となるセキュリティ対策およびモデル駆動開発に関して，修士課程の学生 2 名が「擬似マルウェア体験による Android パーミッションと脅威の関係の学習」「GUI ビルダによるスマートフォンアプリケーション向けモデル駆動開発手法」というタイトルで研究発表を行った．10 分間の質疑応答を行い，多くの有意義な意見をもらうことができた．形式手法やレビュー・分析等のソフトウェア工学における最新の研究動向を知る上で大変有意義であった．

知能ソフトウェア工学研究会（1 月　東京）

電子情報通信学会知能ソフトウェア工学研究会が機械振興会館で行われた.小形研究員が,本プロジェクトの成果の発表として「ユースケースモデルに基づくソースコード検証のためのリバースエンジニアリング手法の検討 ～ ASP.NET アプリケーションを事例として～」の研究発表を行った．30 分の発表，15 分の質疑応答を行い，意見交換を行った．アシュアランスケースの最新動向も知ることができ，大変有意義であった．

2014 ソフトウェア工学の基礎ワークショップ

日本ソフトウェア科学会が主催するソフトウェア工学基礎ワークショップ FOSE2014 が霧島国際ホテルで行われた．SRA の岸田孝一氏が「ソフトウェア工学のパラダイムシフト - Immaterial labor の視点から」というタイトルで基調講演を行った．20 年以上前から言われているようにソフトウェアは形式化された数学的な対象であると同時に，人間活動を支援する道具あるいは作業環境であり，このようなソフトウェア開発の相補的な視点が我々の提案する定義と検証のプロセスにつながるという再認識を持てたことは有意義であった．その他に，実践ソフト開発，OSS，プログラム解析，形式手法，開発支援ツール，企画・設計，モデル検査，プロジェクト管理，要件定義，リファクタリング，テスト・教育のセッションが 2 つの会場で行われ，分担して聴講した．ライブ論文 15 件，ポスター25 件の発表では，分担して，議論を行った．本研究に関連するモデル検査を含め形式手法等のソフトウェア工学における最新の研究動向を知る上で大変有意義であった．

2.2.5 外注

(1) 目的および概要

2.1.3 節で述べた研究目標「仕様定義に基づく検査モデル・検査方式の生成方法の定義」の検討に基づき，ソースコードからモデル検査用のシステムモデルへの変換を支援するツールを設計し，設計に基づきツールを開発することを外注の目的とする．本外注では，C# で定義されたソースコードをモデル検査ツールの１つである UPPAAL のモデルに変換するツールを開発する．この変換においては，検査者が段階的に検査用のモデル（これをベースモデルと呼ぶ）を作成することを支援する機能，ソースコードから UPPAAL モデルへの変換機能，変換後の UPPAAL モデルとソースコードの対応関係を検索する機能が中核となる．本外注では，2012 年度に開発した Source2UPPAAL の基本機能を踏襲し，機能拡張を行う．

(2) 実績

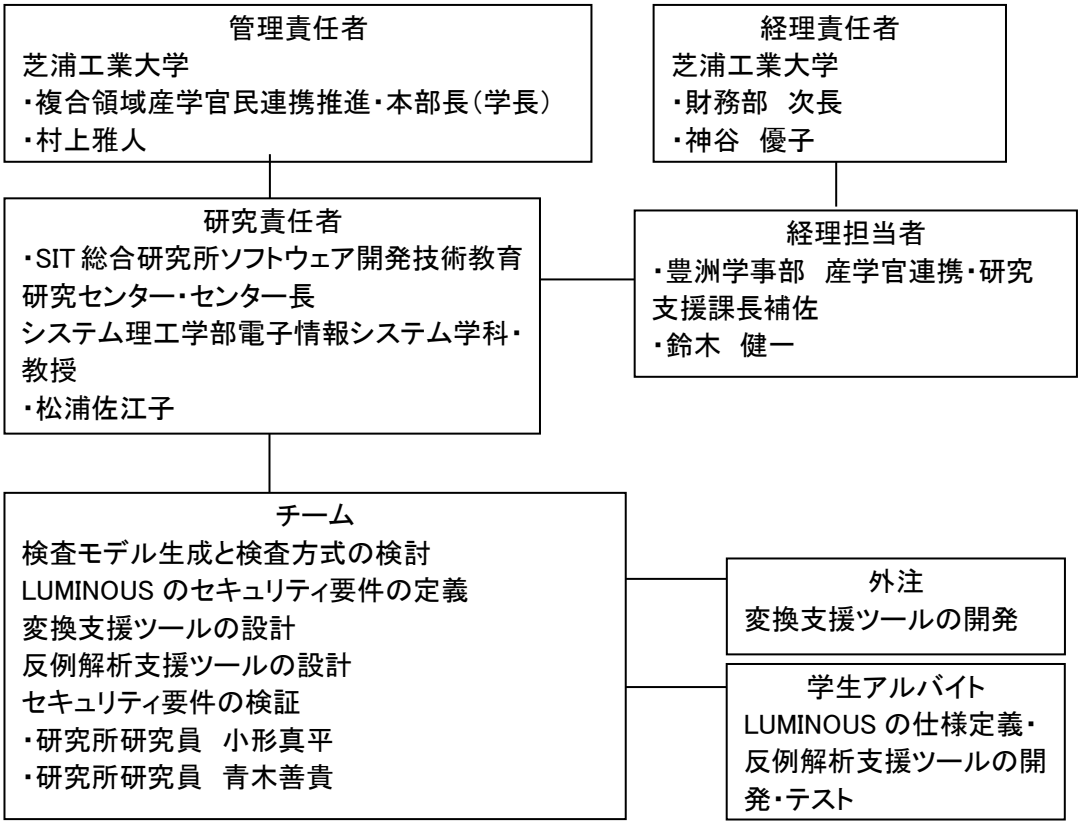
- 本学へ指名業者選定審議申請を行い，学内の委員会の決定を受け，9 月 8 日に「「保守プロセスにおけるモデル検査技術の開発現場への適用に関する研究」変換支援ツール仕様書」に基づき，変換支援ツールを（株）ボイスリサーチに発注した．
- 10 月下旬より，週 1 回の定期的なミーティングを研究メンバーと外注業者で行い，実装内容の確認を行いながら開発を進めた．
- 1 月 22 日に下記物件が納品され，松浦が検収を行った．
  1) ソースコード一式
  2) 設計書
  3) テスト結果報告書
  4) API ドキュメント

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 実施体制

芝浦工業大学
提案責任者　村上雅人（複合領域産学官民連携推進・本部長（学長））
研究責任者　松浦佐江子（SIT 総合研究所ソフトウェア開発技術教育研究センター・センター長
　　システム理工学部電子情報システム学科・教授）
事務責任者　豊洲学事部長　山下修



- 外注
  （株）ボイスリサーチ
- 学生アルバイト
  芝浦工業大学　松浦研究室
  大学院修士課程 2 年　加藤真
  大学院修士課程 1 年　松井浩二
  学部 4 年生　池田滝飛
  学部 4 年生　川合怜
  学部 4 年生　小林雅弥
  学部 4 年生　杉田宗一郎
  学部 4 年生　藤田朋邦

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)	システム理工学部電子情報システム学科 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（2011 年 4 月から 2013 年 3 月）

【研究実績】
- 2012 年度 IPA ソフトウェア工学分野の先導的研究支援事業
- 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」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] 式見，松浦，要求定義の実現可能性保証のためのシミュレーションによるテスト設計手法，電子情報通信学会和文論文誌 D，Vol.J97-D,No.3，pp. 427-436，2014.

[2] 松井，奥田，野呂，岡田，加藤，渡辺，松浦，モデリング能力育成を目的としたユースケース記述の自動評価手法，電子情報通信学会和文論文誌 D ，Vol.J97-D,No.3，pp. 461-472，2014.

[3] 式見，若林，松浦，WebStudy:ソフトウェア開発技能育成を目的としたプログラムの実行・評価機構を持つ Web 教科書の開発，電子情報通信学会和文論文誌 D，Vol. J96-D, No.10，pp.1-12，2013.

[4] 松浦，プログラムの実行・評価機構を持つ Web 教科書によるソフトウェア開発技能育成，論文誌 ICT 情報教育方法研究，（社）私立大学情報教育協会，第 15 巻，第 1 号，pp13-18，2013.

[5] Hirotaka Okuda, Shinpei Ogata, Saeko Matsuura, Experimental development based on mapping rule between requirements analysis model and web framework specific design model, SpringerPlus 2013, 2:123 doi:10.1186/2193-1801-2-123

[6] 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.75-80,2012.

[7] 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.

[8] 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.

[9] 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.

[10]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，コンピュータソフトウェア，Vol.27，No.2，pp.14-32，2010.

[11]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.

[12]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.

[13]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.
[14]松浦, 本位田, 仕様変更プロセスの効果的な再利用 ― まね方をまねる ― , 情報処理学会論文誌, Vol.36, No.11, 1995, pp.2666-2680.

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.

青木善貴
2011 芝浦工業大学大学院修士課程卒. 2012 同大大学院博士課程在学中. 日本ユニシス株式会社にて IT 業務に従事.

2.3.3 研究チームの役割

本研究では LUMINOUS という既存システムのセキュリティ要件を検証するために, LUMINOUS の仕様定義とセキュリティ要件の定義, 検査モデル生成と検査方式の策定, 変換支援ツールを開発し, セキュリティ要件を検証する.

研究メンバーの 3 名が主導して, 学生アルバイトにより, 仕様定義等を行った. 変換支援ツールは外注により開発した.

3 研究成果

3.1 研究目標 1「仕様定義に基づく検査モデル・検査方式の生成方法の定義」

3.1.1 当初の想定

(1) 研究内容

①概要

これまでの Source2UPPAAL で実現している値の割り当て，メソッドの割り当て，ユーザ定義モデルに基づく抽象化について，デシジョンテーブルやセキュリティ機能方針表等の仕様に基づくソースコードの抽象化方法を検討し，再整理する．同時に，変換における自動化の検討を行い，デシジョンテーブル・セキュリティ機能方針表からの検査モデル・検査式の生成方式を決定する．さらに，仕様とソースコードの接点をマッピングする方式を決定する．

②想定される課題と解決策

これまで提案してきた要件定義プロセスにおけるセキュリティ要件の検証方法をもとに検討を進める．提案手法は，図 3.1-1 の赤枠で示すような，セキュリティ要求分析を行い，得られたセキュリティ機能方針の表から，UML 要求分析モデルに対する検査式を自動生成する方法である．

本研究では，この赤枠部分の要件定義を 2.1.3 節の図 2-6 の「仕様」として，ソースコードの検証に用いる．2.1.3 節の図 2-6 のシステムモデルと仕様の接点は，つぎの方針で検討する．

ユースケースレベルの振舞いに対するセキュリティ要件では，対象となるエンティティデータに対する属性を追加して，属性に対する制約により，振舞いを制御する．すなわち，ユースケースの振舞いを規定するアクションと，対象となるデータに付加した属性が，ソースコードと仕様の接点となる．十分なドキュメントのないシステムに対し，その画面に登場するメニュー構成と，その利用手順からユースケースを定義し，ソースコード内のデータモデルから，セキュリティ属性と，属性の更新メソッドを特定することにより検査を実施する．

また，要求分析モデルによりシステムの仕様を定義し，これに従い，2.1.2 節で述べた CC を用いたセキュリティ要求分析手法を用いて，セキュリティ機能方針表 (SFP) を作成するが，SFP に登場するユースケース名やアクションに対応するメソッドをソースコード上で特定する必要がある．

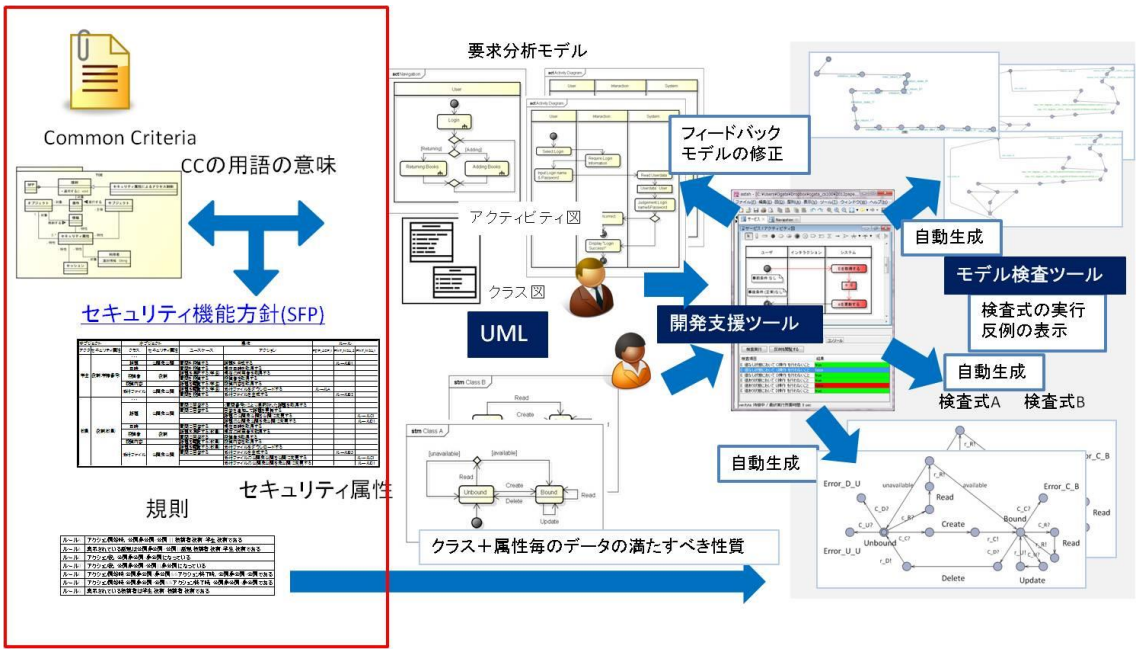


図 3.1-1 要件定義プロセスにおけるセキュリティ要件の検証

(2) 当初の到達目標と期待される結果

既存システムから仕様を UML およびセキュリティ機能方針表を用いて構築し，ソースコードの要素との対応付けを行って，「仕様定義に基づく検査モデル・検査方式の生成方法の定義」方法を確立する．

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 小事例（大学授業課題・与信管理業務事例）による検査モデル生成と検査方式を検討する。

② LUMINOUS の BBS に関するセキュリティ機能方針表による検査モデル生成と検査方式を検討する。

③ 事例検討に基づき、デシジョンテーブルならびにセキュリティ機能方針表の仕様定義に基づく検査モデル・検査式の生成方法を定義する。

④ 研究目標 2 および研究目標 3 の結果に基づき、上記の方法を改善する。

(2) 具体的な研究成果の内容

本節では（1）のプロセスに従って研究を行った成果を，ユースケースによる仕様定義，仕様定義に基づくセキュリティ要件の定義方法，これらによる検査モデルの生成方法と，仕様とソースコードの対応付け方式という順序で説明する．

① ユースケースによる仕様定義

2.1.2 で述べた UML 要求分析手法の考え方に基づき，ユースケース分析による仕様定義

を行う．ここでは，LUMINOUS の BBS の基本機能のモデルを例にユースケース分析を説明する．

UML 要求分析手法では，ユースケースをアクティビティ図とクラス図で定義する．図 3.1-2 のようにシステムは一般に複数のユースケースから構成される．ユースケースはその事前条件，事後条件を持つが，ユースケース図だけでは，システムの利用シナリオは読み取れない．そこで，各ユースケースをどのような順序で利用するかを図 3.1-3 のようなアクティビティ図で表す．ここでは，各ユースケースをサブアクティビティの呼び出しで表している．ガードは，パーティションに記されたアクターのシステム利用時の選択の意思を表す．

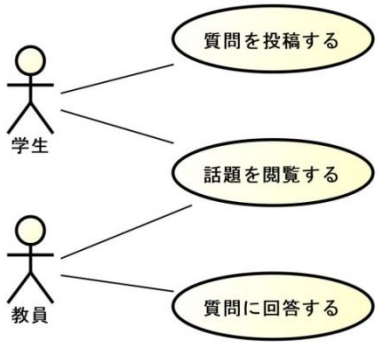


図 3.1-2 BBS のユースケース図

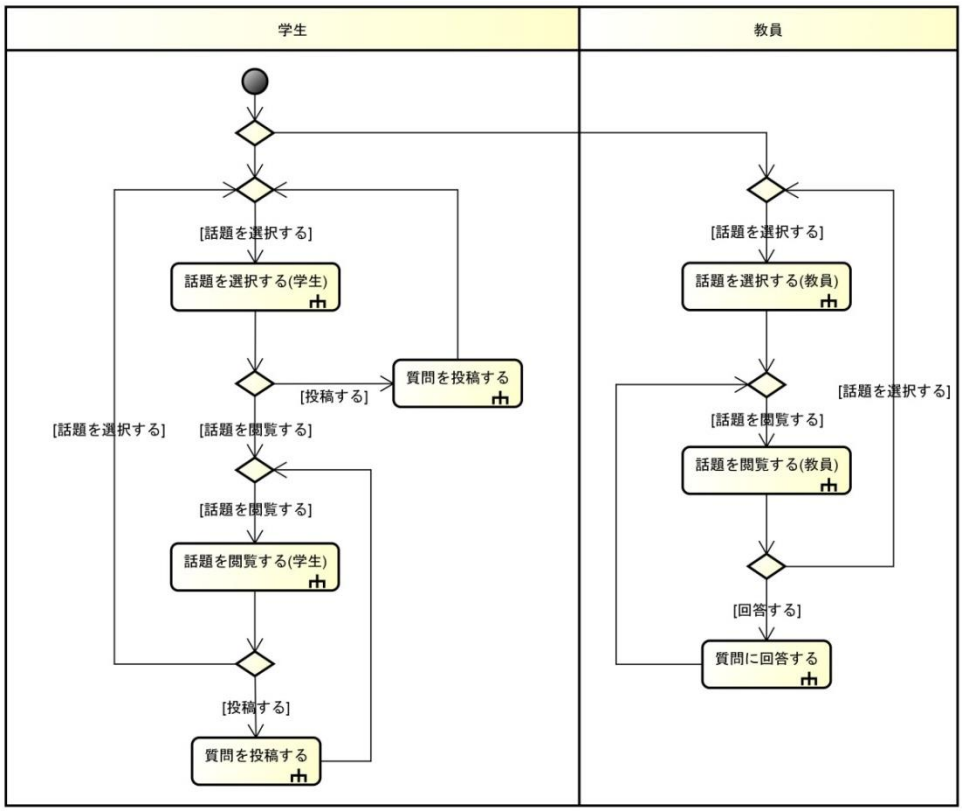


図 3.1-3 システムの利用シナリオ

図 3.1-4 から図 3.1-8 は各ユースケースの定義である．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．

アクティビティ図のパーティションの名前が，アクションの主体を表す．ここでは，アクターである「教員」や「学生」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．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．

ユーザであるアクターのパーティションにはユーザのシステムに対する入力アクションが定義され，インタラクションパーティションにはシステムからの出力アクションと出力データのオブジェクトノードが定義される．出力のデータはクラスとして定義することにより，入出力項目を明らかにするとともに，必要なエンティティデータとの関係を検討することで，ユースケースを実行する上で必要なデータの抽出を行う役割をもつ．

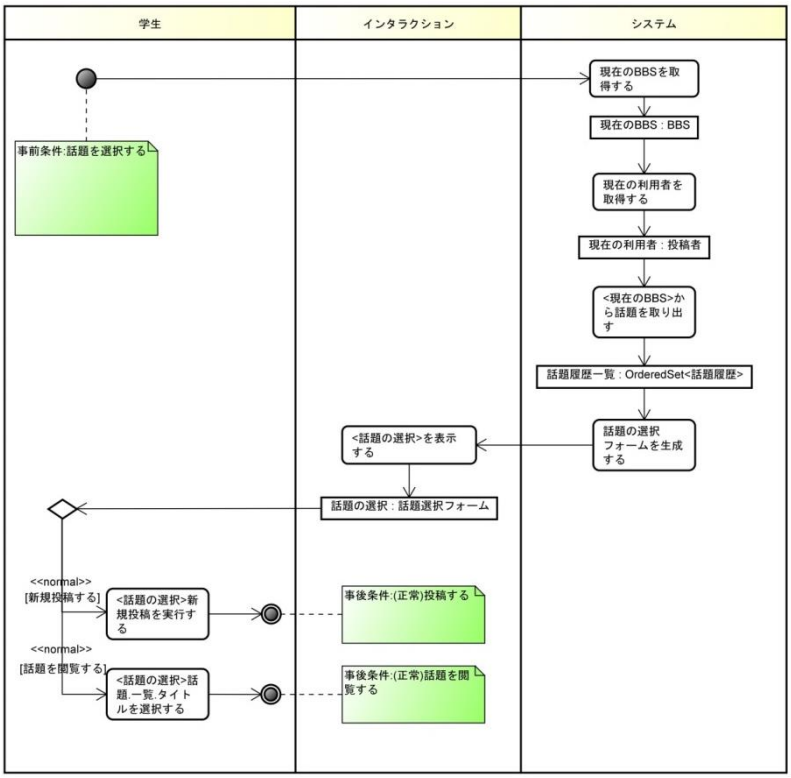


図 3.1-4 話題を選択する（学生）

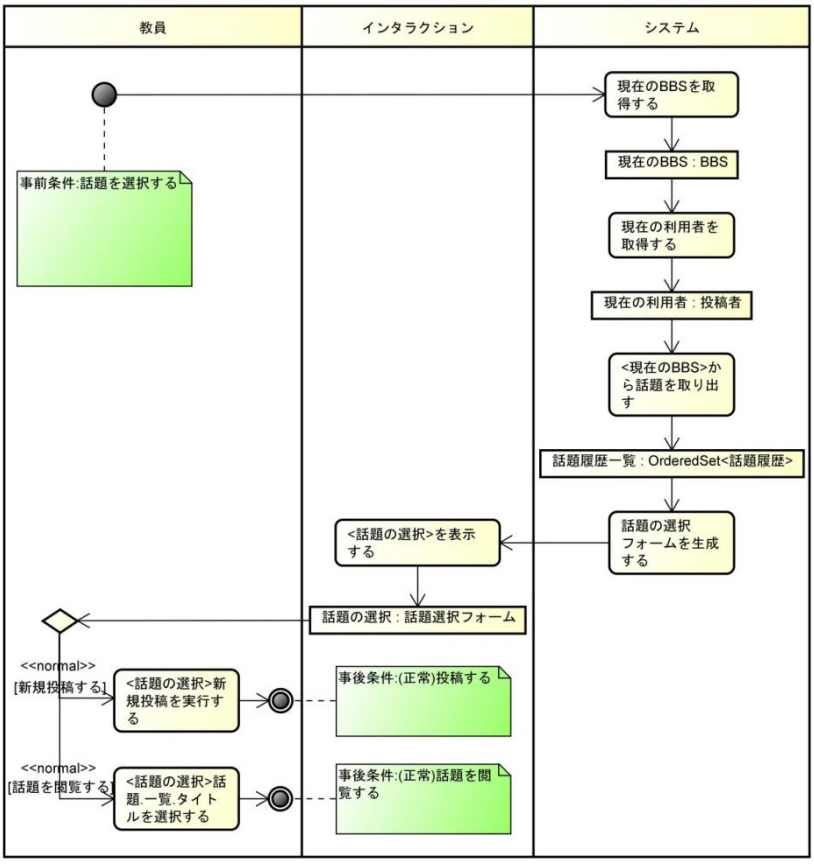


図 3.1-5 話題を選択する（教員）

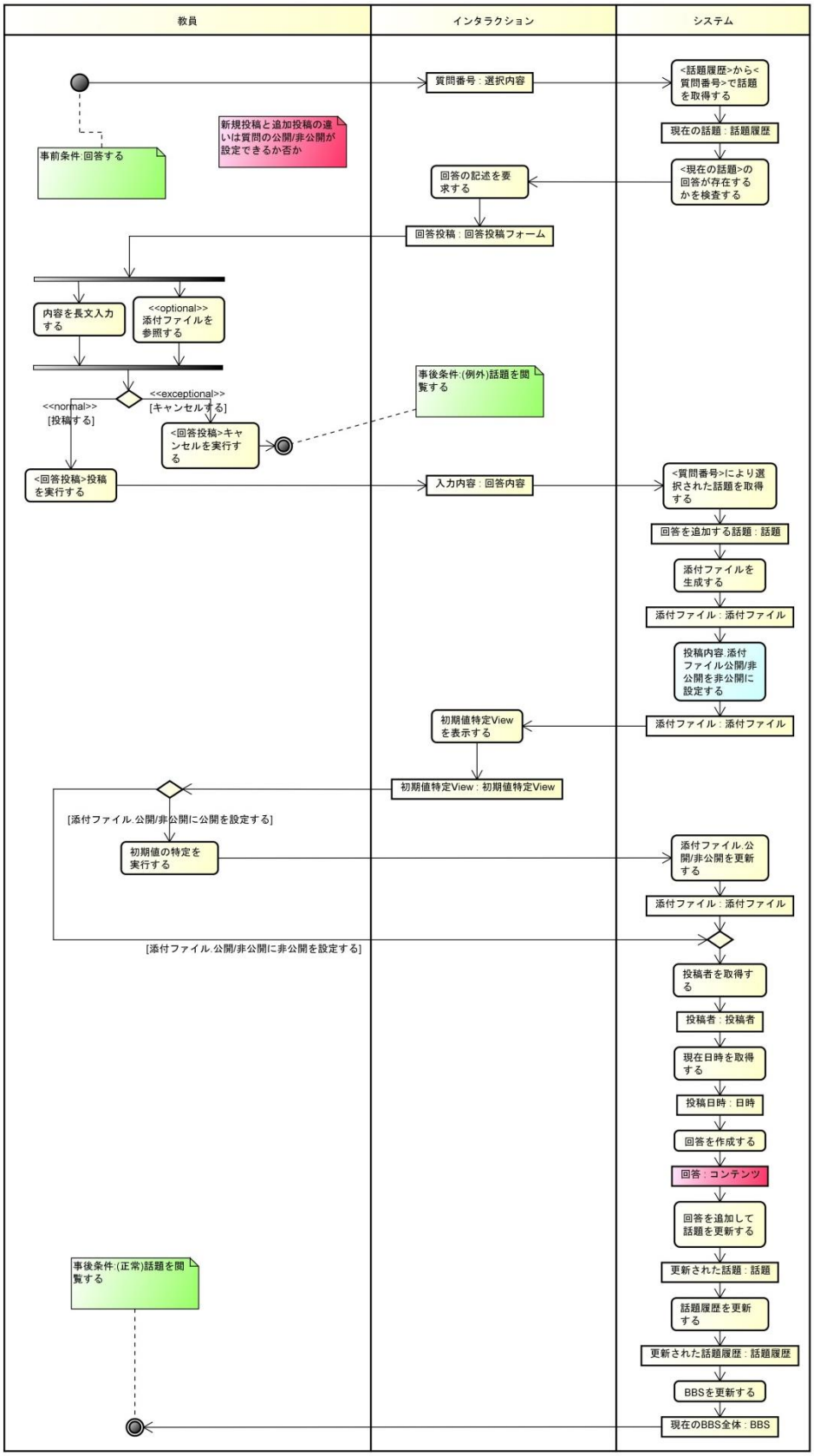


図 3.1-6 質問に回答する

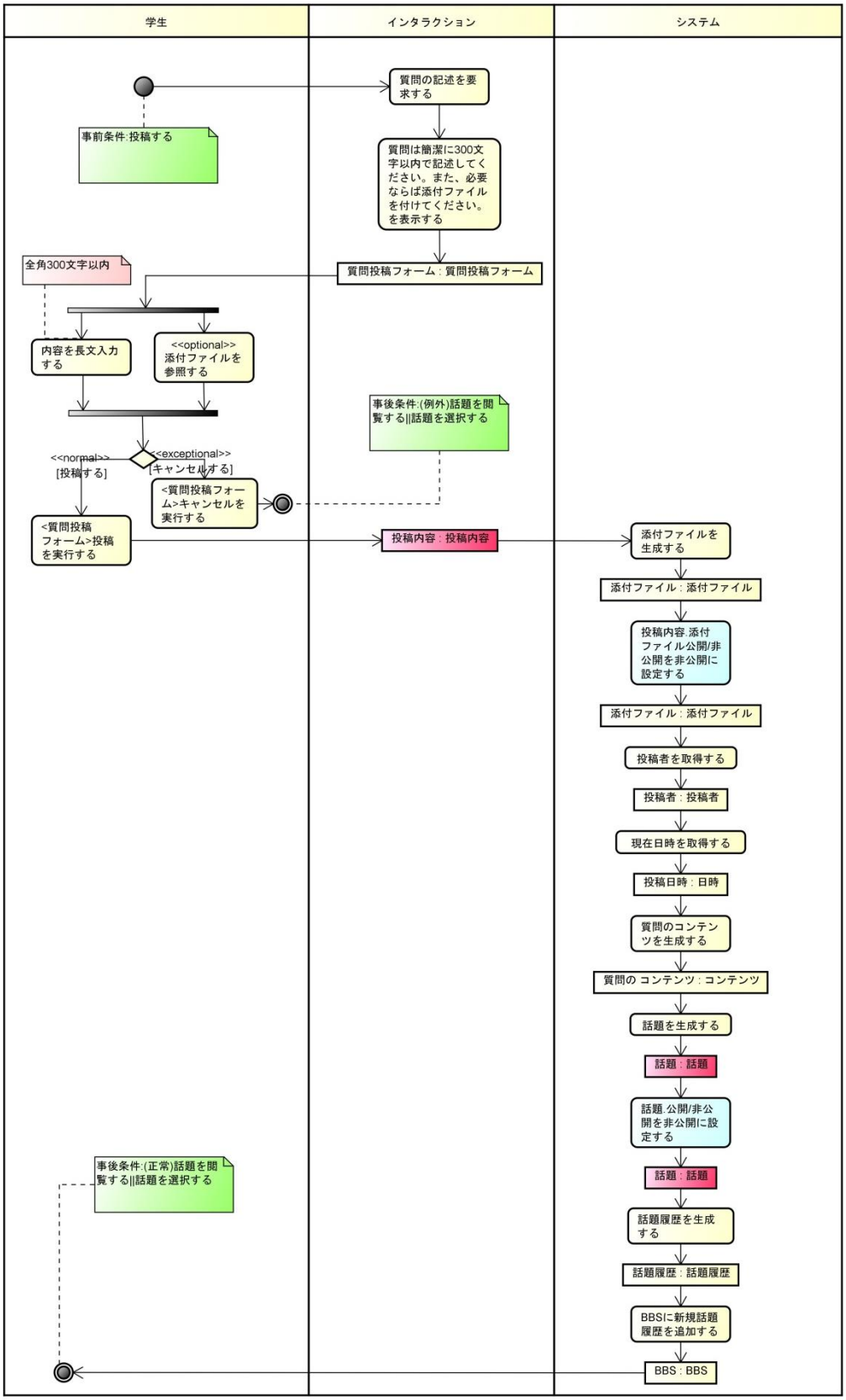


図 3.1-7 質問を投稿する

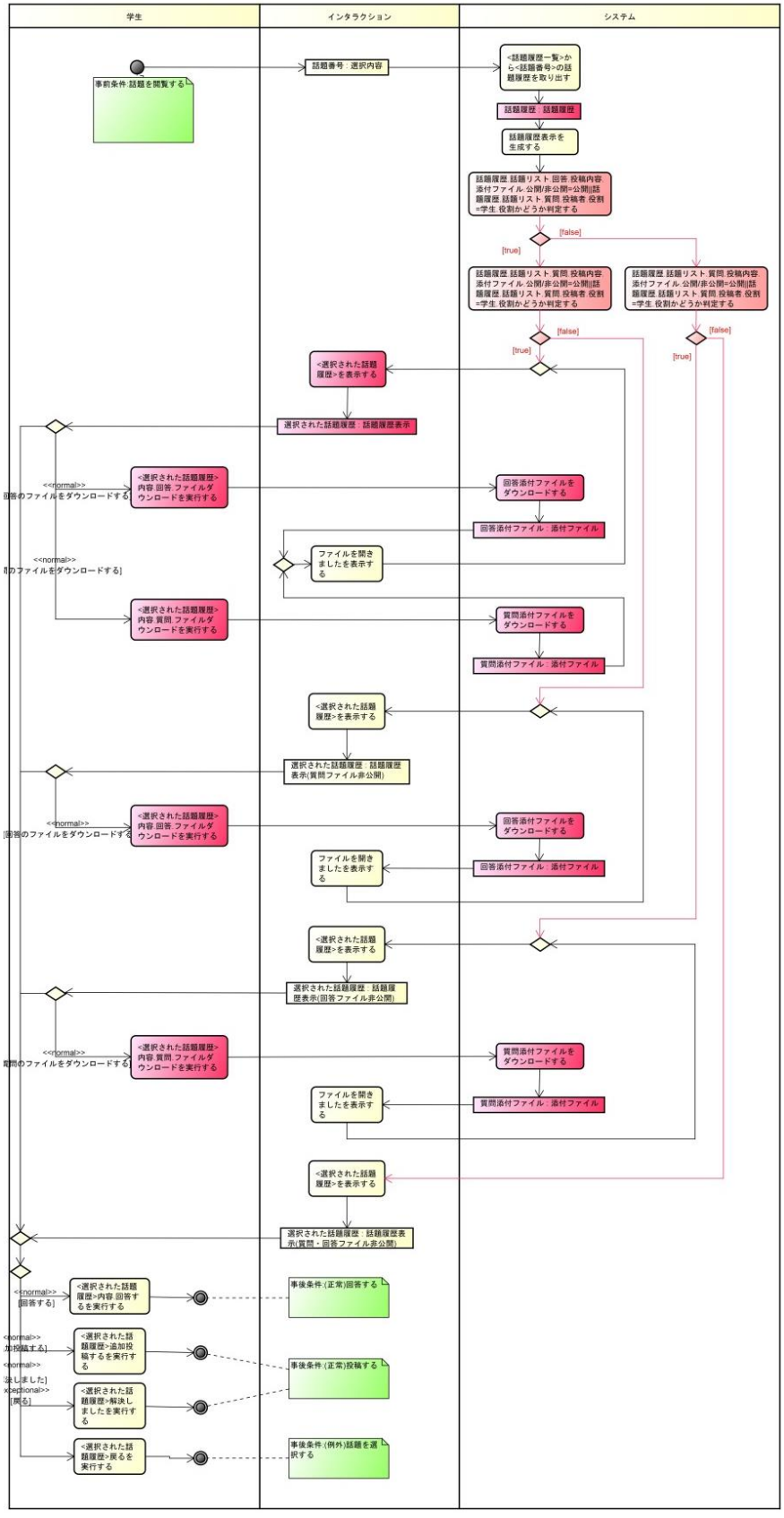


図 3.1-8 話題を閲覧する（学生）

上述の要求分析手法に従い，既存システムのユースケースを定義する．システムを操作し，システムのトップメニューから，メニュー項目を辿り，ユースケース単位を探す．そして，下記の要領で，ユースケースをアクティビティ図で定義する．

- 基本フローを定義する．
- 事前条件をノートに記述する．
- 事後条件をノートに記述する．
- 例外フローは処理フローの行先ごとにまとめてガードに記述する．
- オブジェクトノードのクラス名は図 3.1-9 のエンティティモデルのクラス名を参照する．
- オブジェクトノード名は，アクションの目的語ならびに事前条件・事後条件・ガードに登場する場合はそれと一致させる．
- 画面構成に相当するクラスを定義する．クラス名は各ユースケースの作業状態を表すものとする．項目名は画面上のラベルと一致させる．

ここで，エンティティモデルはシステムのソースコードから抽出されたエンティティデータのクラス図であり，クラス名および属性はソースコードの識別子である．このクラス図の作成方法は，下記の「④ソースコードと仕様の対応付け方式」において説明する．

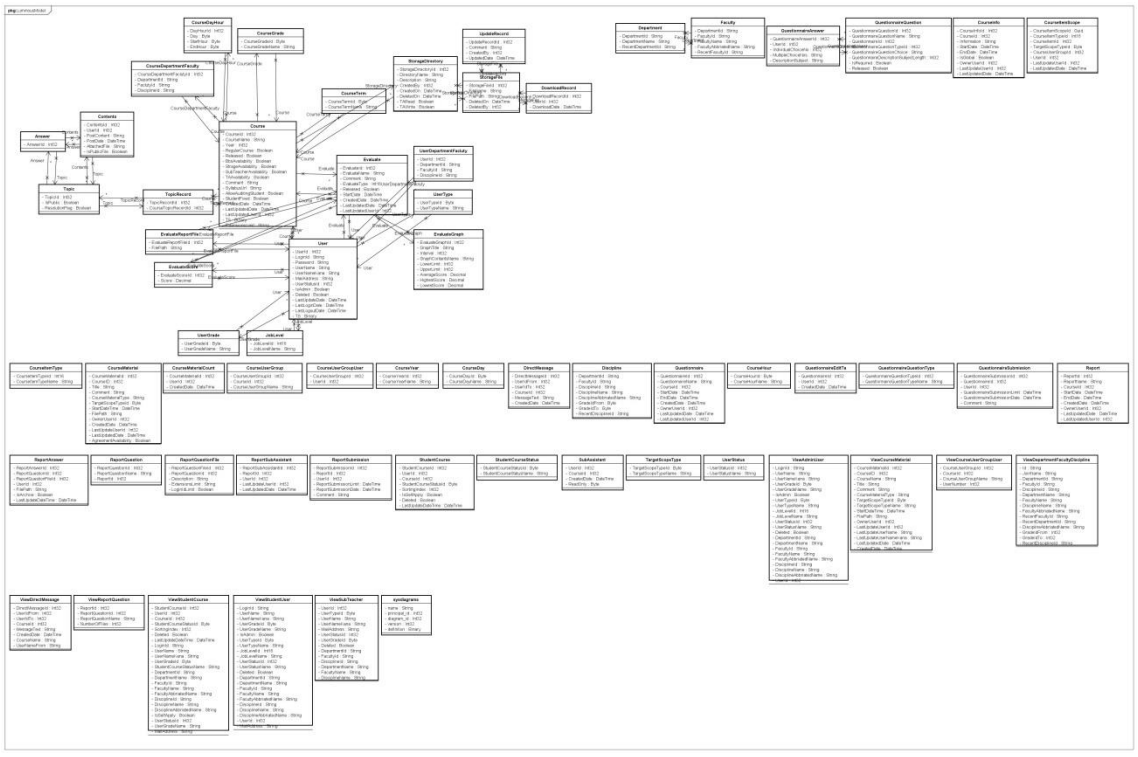


図 3.1-9 エンティティモデルのクラス図

図 3.1-10 および図 3.1-11 はシステムの画面イメージである．図 3.1-10 の左側のメニュー項目のリンクから各機能が呼び出される．両図のボタンがユースケースまたはユースケ

ースのアクションに対応する.

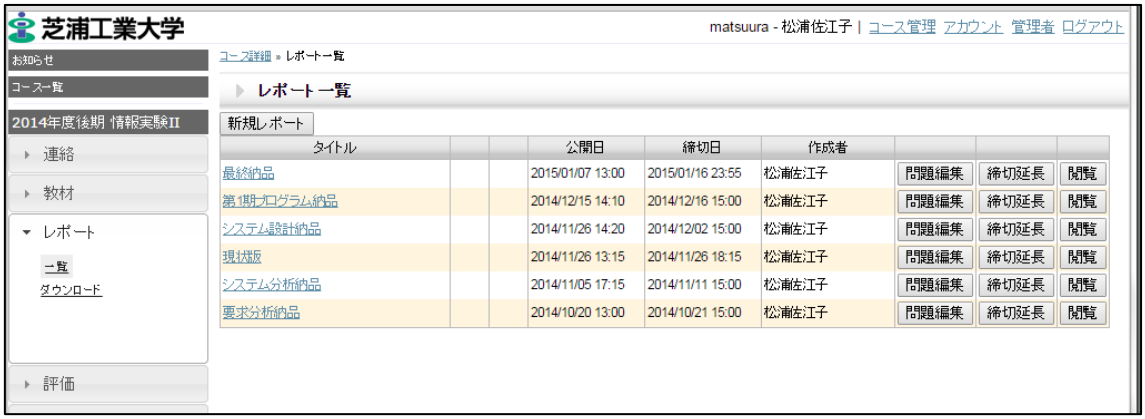


図 3.1-10 システムの画面－1

図 3.1-11 システムの画面－2

図 3.1-12 はトップ画面のメニューに対応する項目のユースケース図である.システムのアクターにより，同じ機能でも，異なる利用方法となる場合は，アクター毎のユースケースを定義する.

図 3.1-13 は各画面の項目を定義したクラス図である．各画面に名前を付けるとともに，その URL を記録する．画面に登場する項目名をそのまま属性とする．ただし，ボタンはその役割がわかるように，「**ボタン」という名称とする.

図 3.1-14 から図 3.1-16 は 3 つのユースケース定義である．このユースケース記述においては，ユーザおよびインタラクションパーティションのオブジェクトノードのクラスは図 3.1-13 に記載されているクラスであり，実際の画面上のラベルと対応付く．また，システムパーティションのオブジェクトノードのクラスは図 3.1-9 エンティティモデルのクラス図にあるクラスであり，これにより各ユースケースが扱うエンティティデータをソースコードと対応付けている.

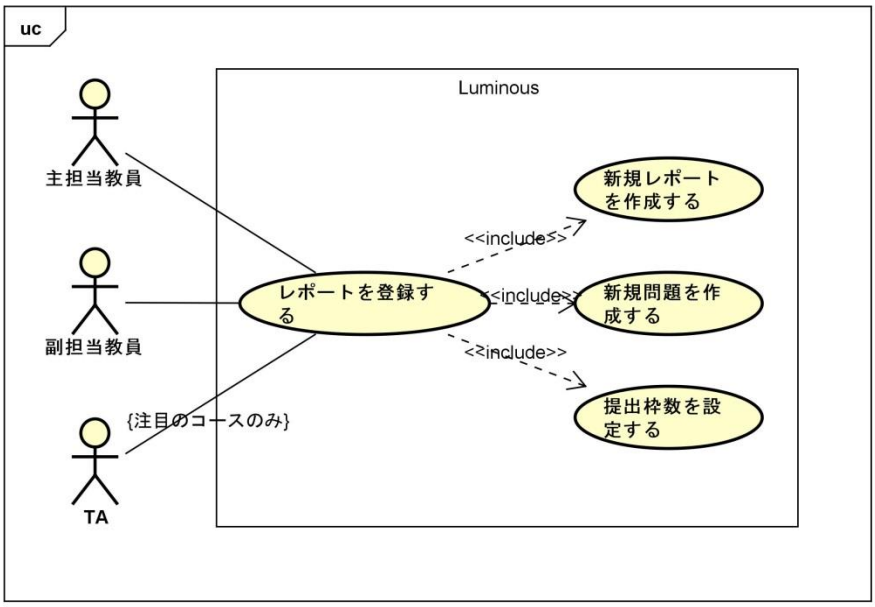


図 3.1-12 「レポートを登録する」のユースケース図

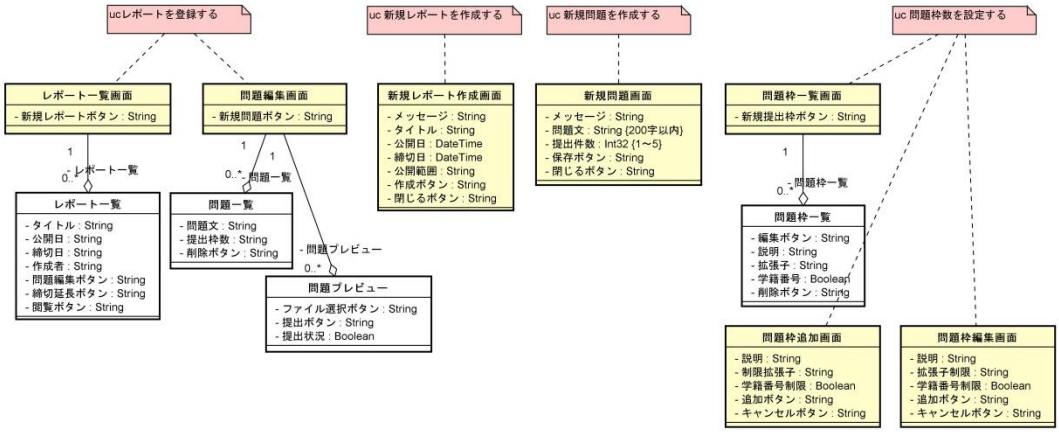


図 3.1-13 画面に対応するクラス図

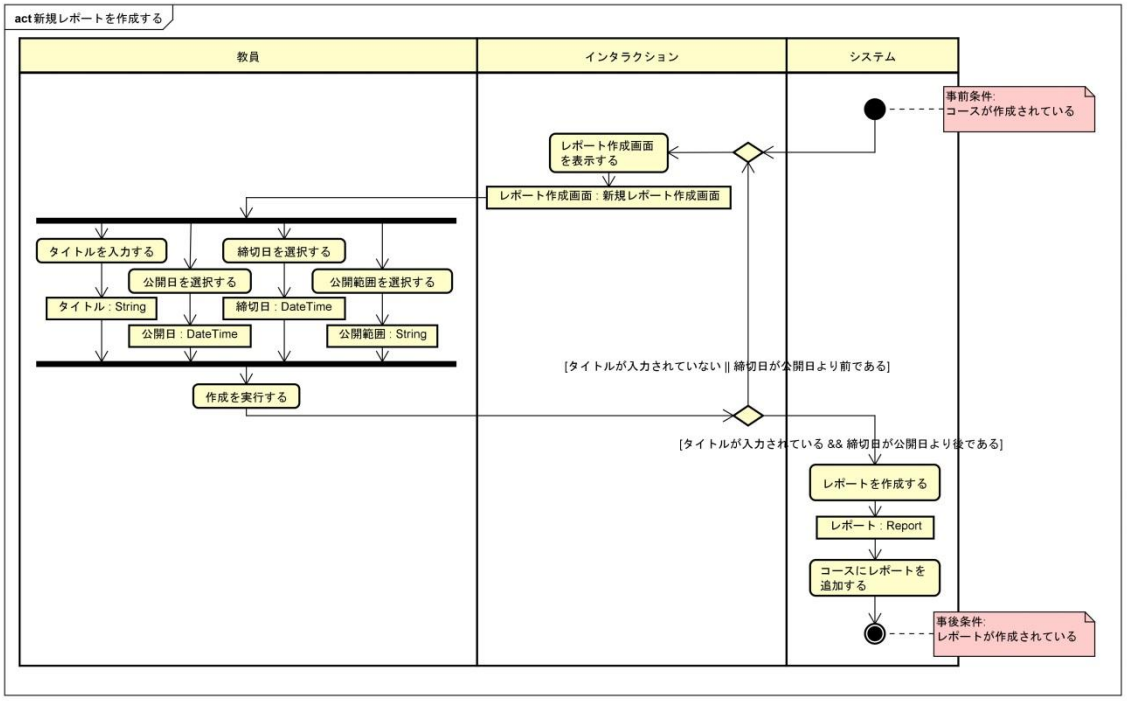


図 3.1-14 ユースケース「新規レポートを作成する」

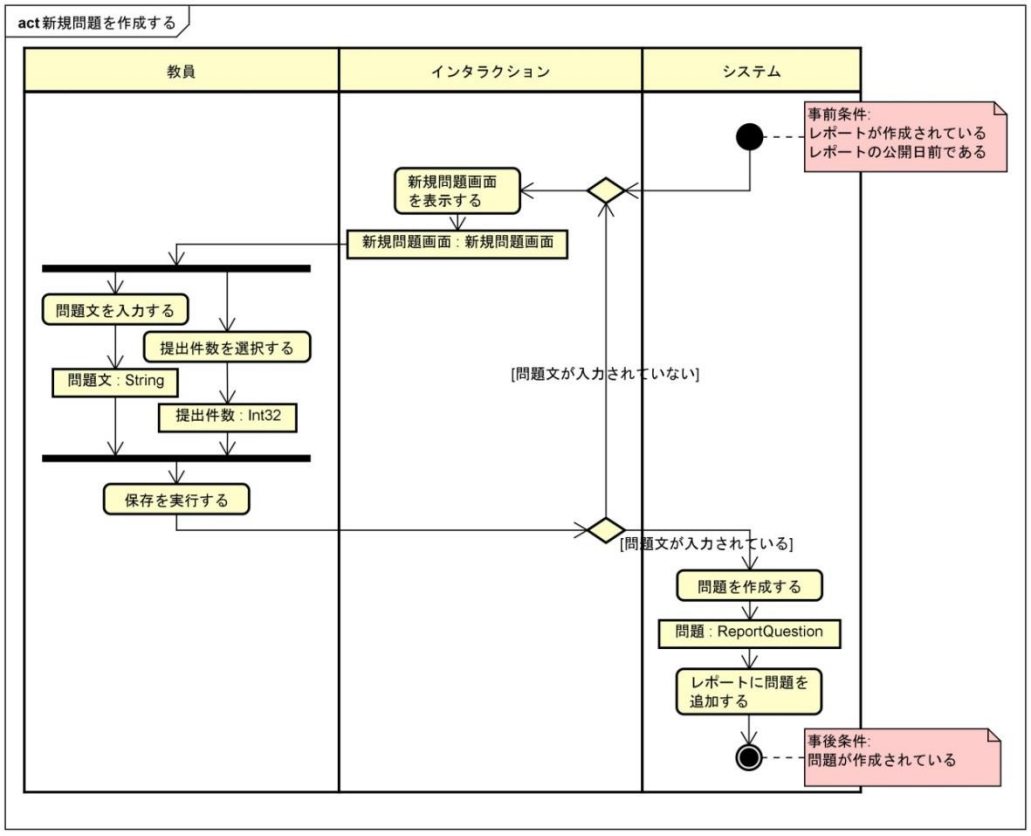


図 3.1-15 ユースケース「新規問題を作成する」

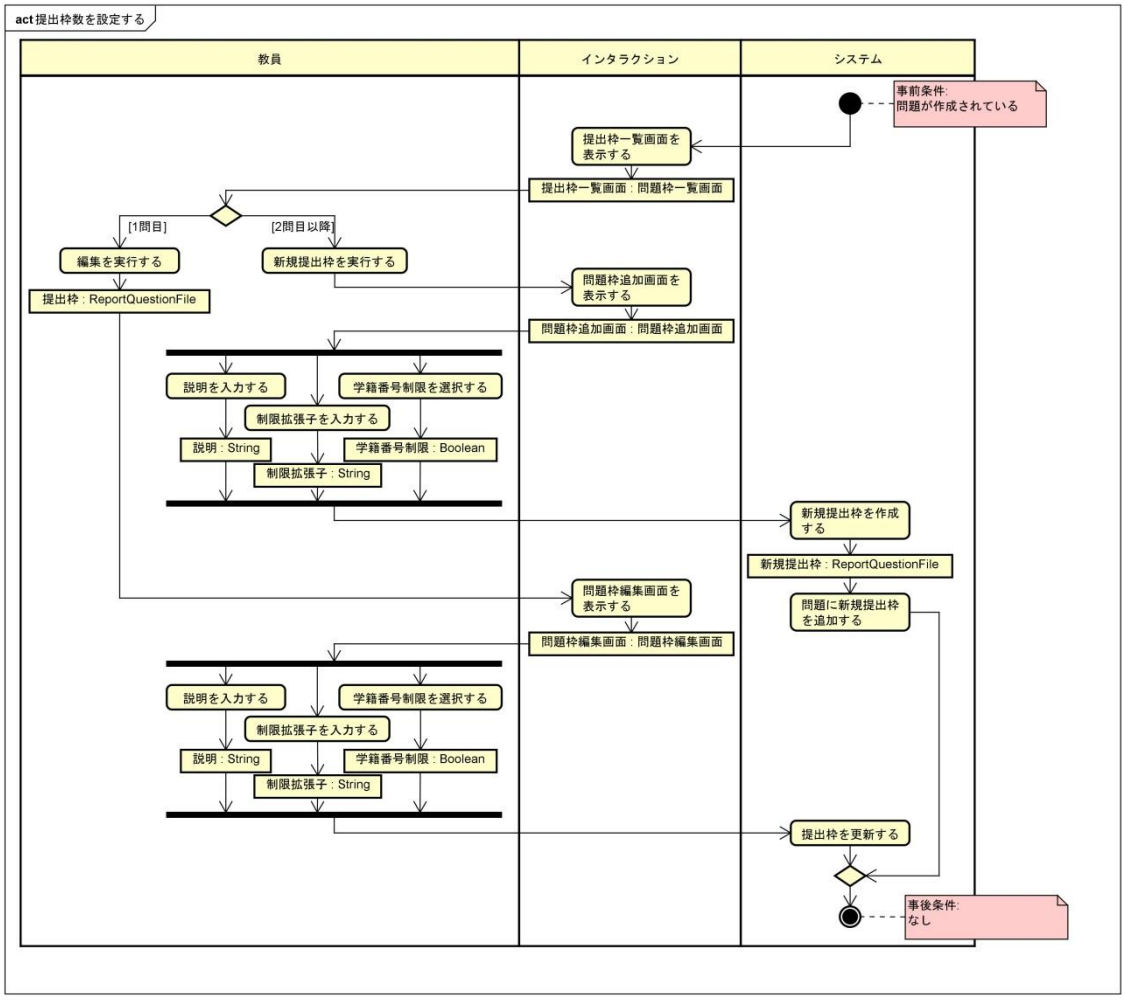


図 3.1-16 ユースケース「提出枠数を設定する」

② セキュリティ要件の定義

図 3.1-4 から図 3.1-8 のユースケース記述から定義された SFP が表 3.1-1 である．2 つのアクターである学生と教員に対して，ユースケースは「学生が質問を投稿する」「教員が質問に回答する」「学生・教員が話題（質問・回答のセット）を閲覧する」である．ここで，質問・回答にファイルを添付し，それをダウンロードすることができる．本機能のセキュリティ要求は，教員の回答や添付ファイルが不必要に質問者以外の学生に公開されないことと，学生の質問が公開されても個人が特定できないことである．教員は回答や添付ファイルに対して，その公開／非公開を設定することができる．

表 3.1-1 の定義手順は以下のとおりである．まず，2.1.2 節の図 2-4 で示したセキュリティ機能コンポーネントモデルとユースケースモデルとの対応により，対象システムのサブジェクト（アクター）とオブジェクトに対してセキュリティ属性を定義する．要求分析モデルのアクティビティ図より，すべてのオブジェクトを抜き出し，それぞれを対象とするアクションを表 3.1-2 のように抽出する．ここで，各ユースケースにおいて対象とする

オブジェクトの利用環境における資産としての価値を考える．各ユースケースの対象となる資産の価値についての詳細は，3.3 節で説明する．資産とそれを何から守るべきかにより，オブジェクトにセキュリティ属性を付加し，守るべきルールを定義する．

上記のセキュリティ要求を満たすためには，学生アクターが個人を識別する情報をもたなければならない．この場合は，対象システムにおける学生の属性である「学籍番号」がそのまま，個人を識別するセキュリティ属性とみなせる．話題や添付ファイルは教員の意思により，その公開／非公開を決定し，学生は公開されているものは閲覧でき，非公開のものは閲覧できないことになる．すなわち，これら 2 つのオブジェクトがこの制御を識別できるように「公開／非公開」というセキュリティ属性を持つ．表 3.1-2 の赤字が設定したセキュリティ属性である．

表 3.1-1 セキュリティ機能方針表

サブジェクト		オブジェクト		操作		ルール		
アクタ	セキュリティ属性	クラス	セキュリティ属性	ユースケース	アクション	FDP_ACF.1	FMT_MSA.3	FMT_MSA.1
学生	役割(学籍番号)	話題	公開／非公開	質問を投稿する	話題を生成する		ルールB1	
		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	ルールA		
				質問を投稿する	添付ファイルを生成する		ルールB2	
教員	役割(教員)	話題	公開／非公開	質問に回答する	回答を追加して話題を更新する			
					話題の公開／非公開を公開に変更する			ルールC1
					話題の公開／非公開を非公開に変更する			ルールD1
		添付ファイル	公開／非公開	質問に回答する	添付ファイルを生成する		ルールB3	
					添付ファイルの公開／非公開を公開に変更する			ルールC2
					添付ファイルの公開／非公開を非公開に変更する			ルールD2

表 3.1-2 セキュリティ属性の定義

サブジェクト		オブジェクト		操作	
名前	セキュリティ属性	名前	セキュリティ属性	ユースケース	アクション
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
		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
				質問を投稿する	添付ファイルを生成する
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
					話題の公開非公開を公開に変更する
					話題の公開非公開を非公開に変更する
		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
				質問に回答する	添付ファイルを生成する
					添付ファイルの公開非公開を公開に変更する
					添付ファイルの公開非公開を非公開に変更する

表 3.1-2 に対し，2.1.2 節の図 2-5 に示した CC のコンポーネントの依存関係に従い，必

要なルールを設定し，表 3.1-1 のようにセキュリティ機能方針（SFP）を整理する． この時，表 3.1-2 において，セキュリティ属性と関係しないアクションや Read のアクションは除く．

つぎにセキュリティ属性の持つべき状態を識別し，これらの間の遷移をステートマシン図により図 3.1-17 のように定義する．ここでの状態遷移のイベントが，「話題」というオブジェクトの属性「公開／非公開」の値（未設定・公開・非公開の 3 つ）を遷移させる属性の更新アクションに対応する．このイベント名が対象システムのアクティビティのアクション名と対応付けられる．ここで，CC のクラス FMT（セキュリティ管理）のコンポーネント FMT_MSA（セキュリティ属性の管理）により，セキュリティ属性を持つ 2 つのオブジェクト「話題」と「添付ファイル」に，新たに，セキュリティ属性の更新アクションが表に追加されると共に，そのルールが定義される．ここで，「話題」は図 3.1-18 におけるソースコードのエンティティモデルのクラス Topic に，「添付ファイル」は AttachedFile に対応する．

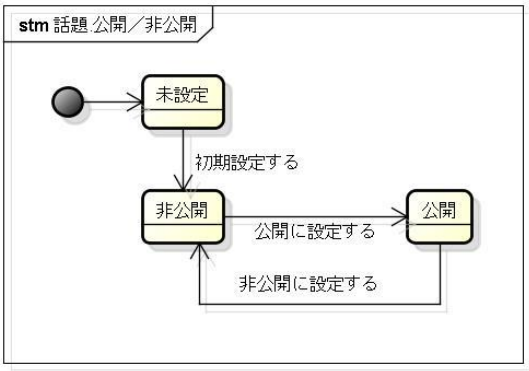


図 3.1-17　セキュリティ属性の定義

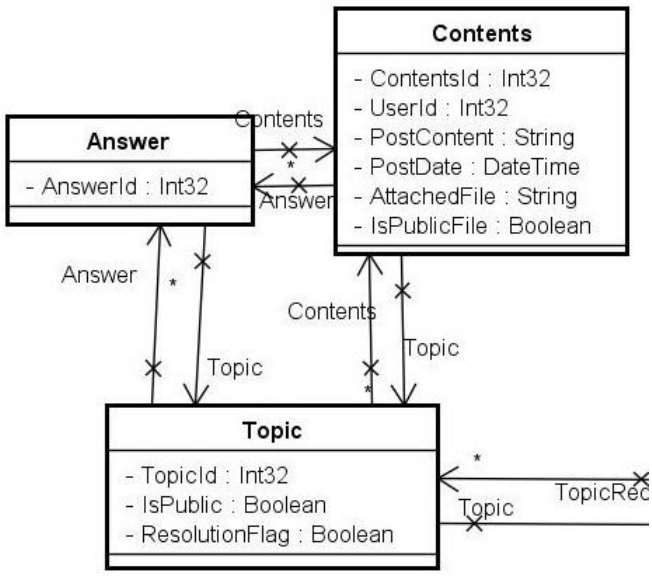


図 3.1-18 エンティティモデル（部分）

ルールは，表 3.1-1 の要素を用いて，CC の機能コンポーネントの構造ならびに依存関係から選定されたコンポーネント毎に，制御が必要なアクションに対して，表 3.1-3 のように定義する．例えば，ルール A はコンポーネント FDP_ACF.1（セキュリティ属性によるアクセス制御）に基づき，「学生が話題を閲覧する場合にはその添付ファイルが公開になっている，または投稿者自身ならばダウンロードできる」という規則をアクターと添付ファイルのセキュリティ属性を使って定義している．ルール B3 は，FMT_MSA.3（静的属性初期化）に基づき，「教員が質問に回答する際に添付ファイルを投稿した場合，話題が公開ならば公開または非公開，話題が非公開ならば非公開」という規則を話題と添付ファイルのセキュリティ属性を用いて定義している．ここで，「話題」オブジェクトは「添付ファイル」を含んでいることから，このような階層的な制御を行う．

表 3.1-3 ルールの定義

ルール A	アクション開始時，添付ファイル.公開／非公開==公開 || 投稿者.役割==学生.役割
ルール B1	アクション終了時，話題.公開／非公開==非公開
ルール B2	アクション終了時，添付ファイル.公開／非公開==非公開
ルール B3	アクション終了時，（話題.公開／非公開==公開ならば添付ファイル.公開／非公開==公開 || 添付ファイル.公開／非公開==非公開） && （話題.公開／非公開==非公開ならば添付ファイル.公開／非公開==非公開）
ルール C1	アクション開始時に話題.公開／非公開==非公開ならば，アクション終了時に話題.公開／非公開==公開
ルール C2	アクション開始時に添付ファイル.公開／非公開== 非公開ならば，アクション終了時に添付ファイル.公開／非公開== 公開
ルール D1	アクション開始時に話題.公開／非公開== 公開ならば，アクション終了時に話題.公開／非公開==非公開
ルール D2	アクション開始時に添付ファイル.公開／非公開== 公開ならば，アクション終了時に添付ファイル.公開／非公開==非公開

③　検査モデルの生成方式

ユースケース記述の要素から，セキュリティ機能方針表を導き，検査すべき規則をこれらの要素で定義することができた．例えば，表 3.1-3 のルール B1 は

アクション終了時，話題.公開／非公開==非公開

であるが，このアクションは表 3.1-1 より，ユースケース「質問を投稿する」のアクション「話題を生成する」である．アクションの終了時に成り立つならば，ユースケース終了時にも成り立つため，このルールは

ユースケース「質問を投稿する」終了時，話題.公開／非公開==非公開

となる．

すなわち，ユースケースを実行した後に，セキュリティ属性の値が指定したものになっていることが必要であり，図 3.1-17 のセキュリティ属性の取るべき状態遷移がこの間に生

じていることを確認できればよい.

そこで，検査式をソースコードの識別子と結びつけるためには，ユースケースの呼び出しメソッドとセキュリティ属性の更新メソッドの特定が必要になる．この特定方法については「④ソースコードと仕様の対応付け方式」において述べる.

図 3.1-19 は検査シナリオに基づく検査モデルの全体像を表している.検査シナリオは複数のユースケースに対応するメソッドのモデルを呼び出す．これはソースコードのメソッドに対応する UPPAAL モデルである．各メソッドが実行される間に，セキュリティ属性の更新メソッドが呼び出され,それに同期して,属性の仕様である状態遷移モデルが変化する．これにより，ユースケースが終了した時点で，セキュリティ属性の状態が想定の値であるかという上述の検査式を検査することで，ソースコードがルールを満たしているかを検査できる.

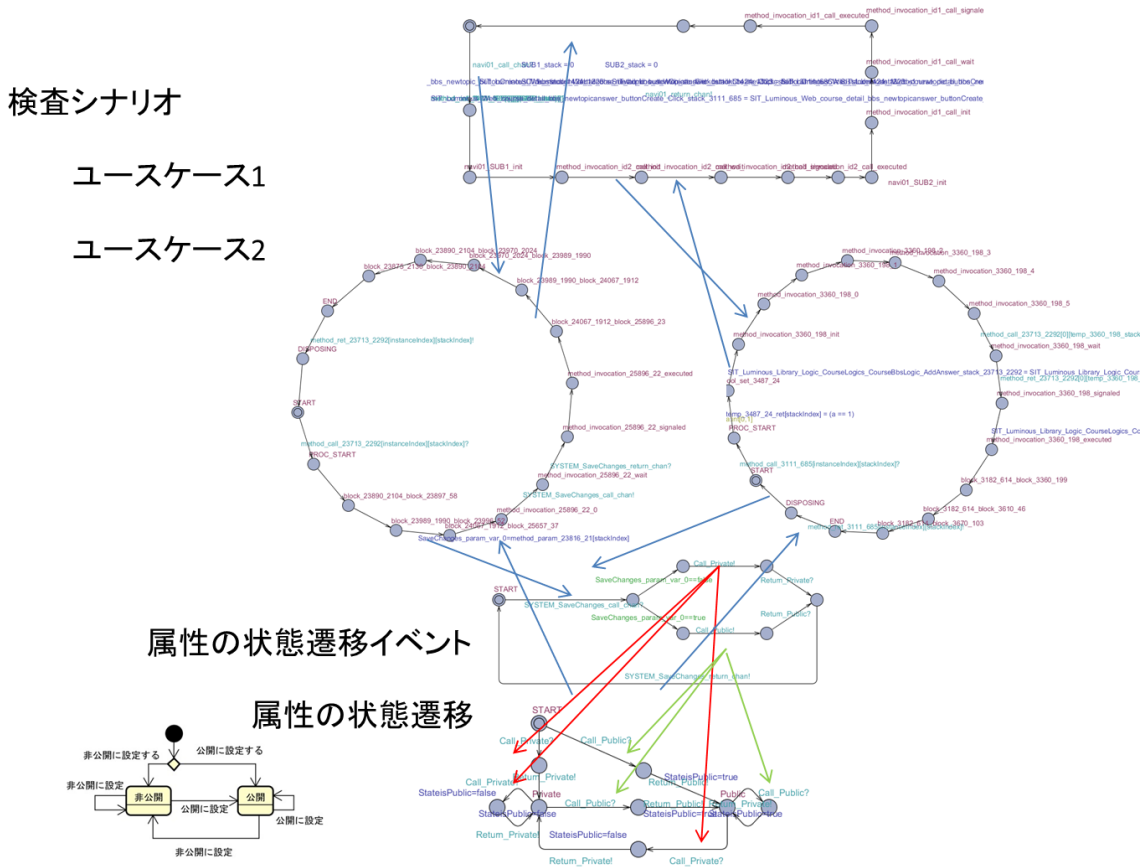


図 3.1-19 検査モデル

図 3.1-20 は，図 3.1-17 のセキュリティ属性に関する仕様モデルの変換とその属性の更新メソッドとの対応を示している．これらのモデルが，図 3.1-19 のようにソースコードを変換したモデルと連動して，上記の検査式を検査することができる.

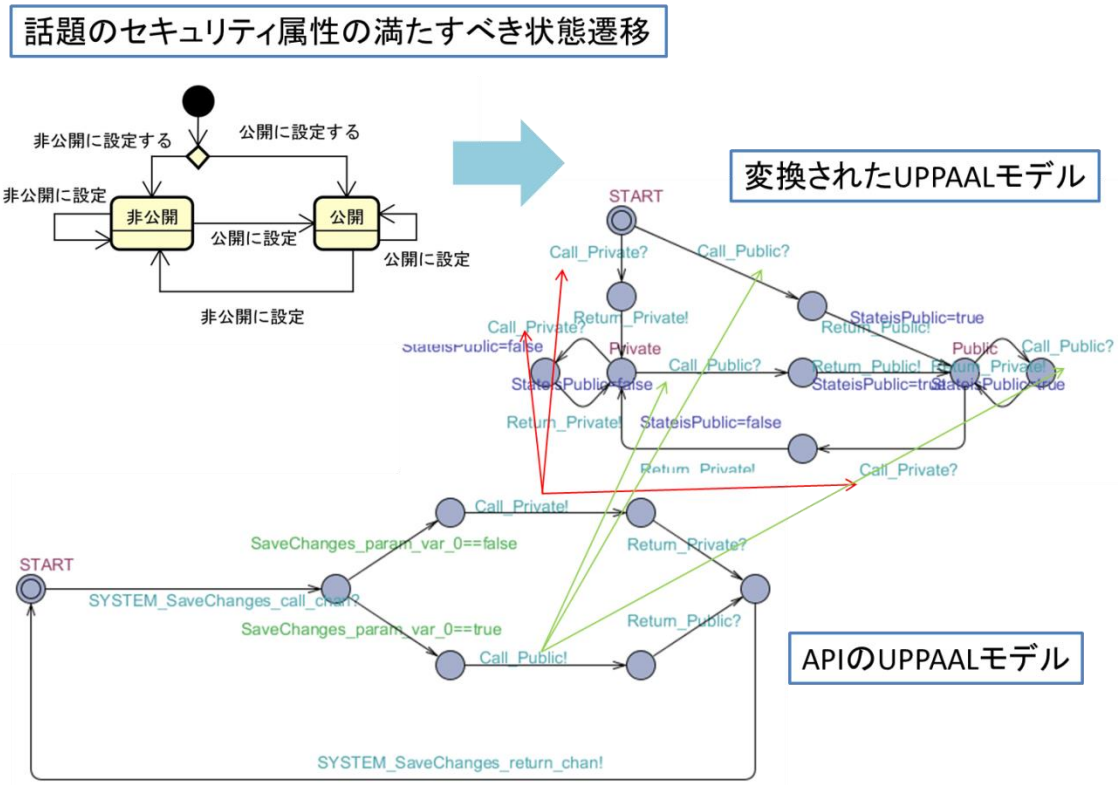


図 3.1-20 セキュリティ属性の更新の検査モデル

検査は，下記の 2 種類をこの順序で行う．

1) 各ユースケース単位のルールの検査
2) 検査シナリオによる検査

システムには複数の役割の異なるアクターがおり，ユースケースの事前条件による利用シナリオがある．そこで，それぞれのユースケースの実行による結果に対して，保証しなければならないことを検査シナリオとして定義し，検査する．

例えば，教員のユースケース「質問に回答する」は事前条件として，学生の質問があることが規定されている．そこで，表 3.1-4 のルールを組み合わせて，学生のユースケース「質問を投稿する」を実行後に下記の項目を検査するシナリオを設定する．検査式は図 3.1-21 のようになる．実際の検査式はこのように，UPPAAL のロケーション名や変数により定義されている．

表 3.1-4 検査シナリオのベースとなる規則

サブジェクト		オブジェクト		操作		ルール		
アクタ	セキュリティ属性	クラス	セキュリティ属性	ユースケース	アクション	FDP_ACF.1	FMT_MSA.3	FMT_MSA.1
学生	役割(学籍番号)	話題	公開／非公開	質問を投稿する	話題を生成する		ルールB1	
教員	役割	話題	公開／非公開	質問に回答する	話題の公開／非公開を非公開に変更する			ルールD1

02	教員が質問に回答する	回答処理終了時に非公開指定で公開になることはない	A[] not （SYSTEM_SIT_Luminous_Web_course_detail_bbs_newtopicanswer_buttonCreate_Click(0,0).END && SaveChanges_param_var_0==false && SYSTEM_State_Private_Public.Public)

図 3.1-21 検査シナリオに基づく検査式

変換支援ツールを用いた検査モデルの生成方式は 3.4 で説明する．

④　ソースコードと仕様の対応付け方式

ユースケース記述では，ユーザの操作やエンティティに対する CRUD 処理が，アクションやオブジェクトノードとして記述されている．これまでのユースケースによる仕様定義とソースコードの対応関係を表 3.1-5 に示す．

表 3.1-5　ユースケースモデルとソースコードの対応関係

ユースケース記述	ソースコード
ユーザの操作	UI 要素．例えば，ボタンやチェックボックスなど．Web アプリケーションであれば，input タグなどに相当する．また，画面を遷移させる submit 系のボタンに基づいて，送信先の処理メソッドを特定できる．
エンティティおよびセキュリティ属性	データベース内のデータ構造．例えば，アクセス制御においては，クラスもしくは属性を単位とするアクセス制御対象のデータ，およびその可視／不可視を決めるセキュリティ属性をデータベース定義から指定する．
CRUD 処理	データベースへの処理メソッド．外部ライブラリなどを用いて，データベースへの CRUD 処理を行うことを前提とし，その呼び出し API および実引数から CRUD 処理の内容と場所を特定できる．但し，ライブラリ個別に当該 API を特定する必要はある．

ユースケース記述の要素から，セキュリティ機能方針表を導き，検査すべきルールをこれらの要素に対応付けることにより，ソースコードから検査モデルを定義し，検査モデルに対応させて，ルールから検査式を定義することができる．

本研究ではソースコードの静的解析により，図 3.1-22 のように，以下の 5 種類の情報をソースコードから全て抽出する．

1)　entity クラス
2)　entity クラスの属性（entity 属性）
3)　UI クラス
4)　UI クラスのメソッド（UI メソッド）
5)　ボタンやチェックボックスなどの UI 要素とそのソースコード表現（UI コード）

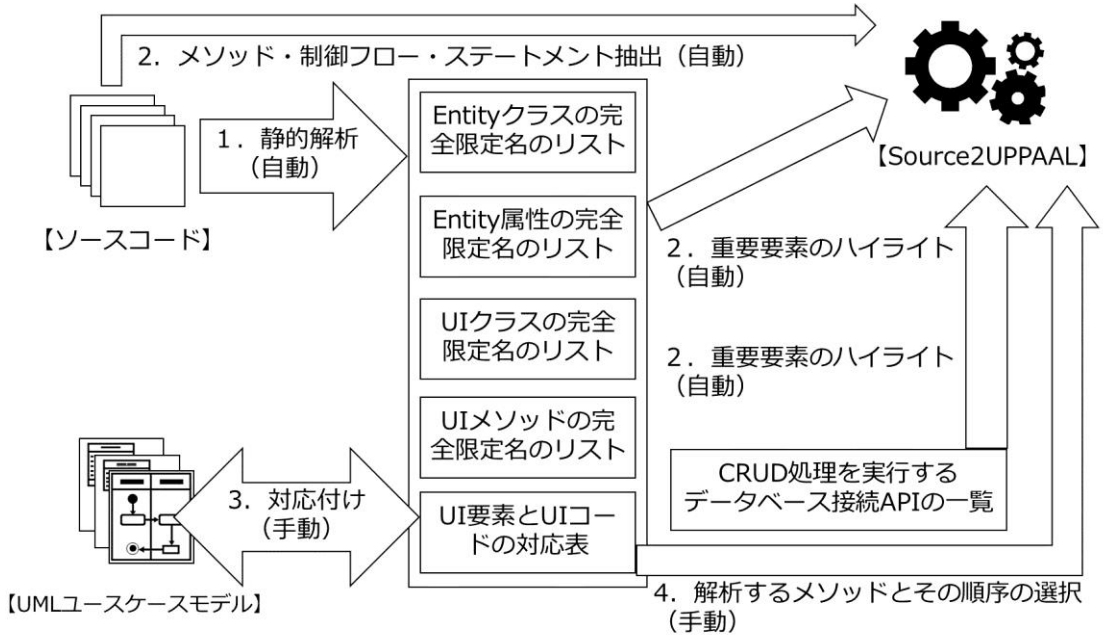


図 3.1-22 ソースコードからの情報の取得

本研究で対象とする LUMINOUS は ASP.NET[20]アプリケーションである．そこで，C#の Entity Framework 中の entity data model を表現する edmx ファイルが，entity クラスや entity 属性の抽出対象になる．また，Web ページを表現する aspx や ascx ファイルのコードビハインドとなる cs ファイルが，UI クラスおよび UI メソッドの抽出対象となる．

抽出された情報から，検査者は手動でこれらの 1)～4)から必要な情報（例えばアクセス制御に係る要素）のみを残すように選別する．

5)については，ユースケースモデルと対照しながら，ユースケース実行に必要な UI 要素を並べていくことで，そのときに実行される UI コードを把握する．例えば，LUMINOUS では，表 3.1-5 が 5)の内容に相当する．

表 3.1-5　UI 要素と UI コードの対応表

Type	Name/Code	Location
PAGE	BBS	APPLICATION_ROOT/SIT.Luminous.Web/student/detail/bbs/bbslist.aspx
Link	トップ	~/
Link	[%=string.Format({0}-TOP",ContextUtili	[%= VirtualPathUtility.ToAbsolute(~/student/detail/)%]
Button	新規質問	return openWindow('[%= VirtualPathUtility.ToAbsolute(~/student/detail/bbs/newtopic.aspx)%]
Link	[%#Eval(Topic.Contents.PostContent)%]	[%= VirtualPathUtility.ToAbsolute(~/student/detail/bbs/topicdetail.aspx)%]
Link	[%#Eval(Topic.Contents.PostContent)%]	[%= VirtualPathUtility.ToAbsolute(~/student/detail/bbs/topicdetail.aspx)%]

Type は，Web ページ上の UI 要素の種別を表す．例えば Page や Link，Button，CheckBox などがある．この種別は実装プラットフォームにより異なるため，一意に定めることは難しい．ここでは，解析対象となる aspx と ascx を静的解析して，使用されているタグのセットを取り，そこから種別全体を定めている．

Name/Code は,UI 要素の種別ごとにその要素のラベルを表す.例えば,Page の場合は title タグの内容を解析し，Link の場合は anchor タグに挟まれた文字列を取得する．現状では

UI 要素のタグに無関係な箇所にラベルが存在する場合や，そもそもラベルが欠落する場合は空欄となる．なお，このコーディング方法の不統一による問題はコーディング規約により解決を図れる．

また，静的解析であるために，テンプレート化されたコードがそのまま表示される箇所がある．これは実装に用いたフレームワーク等の併用よるものであり，例えば，HTML 形式のコード中にテンプレート化されたコードが登場し，そのテンプレートコードの中に JavaScript のスクリプトが登場することがある．この種の多技術併用のコードを正確に解析することは，大きな研究課題の１つになる．

Location は，Page の場所や Link の遷移先などの場所を示す．Button の場合は遷移先のヒントを得るために onClick 時に実行するコードを表す．本対応表の行をユーザがユースケース実行時に操作する順序に並び替えることにより，検査者がバックエンドで呼び出される．メソッドとその順序を理解できる．

そして，Source2UPPAAL において，1)～4)の要素を強調表示して，検査に関連するエンティティなどを特定した後に，5)の情報を基に実行するメソッドを順序も含めて選択することで，ユースケースモデルに沿ったソースコード検証を，ソースコード全体を読み解くことなく実施できるようになる．詳細は 3.4 で説明する．

3.1.3 実用化へ向けた課題と問題点

(1) 課題と問題点

ソースコードが仕様を満たしているかを検証するためには，仕様とソースコードの要素を対応付けることが必須である．本研究では，ソースコードの静的解析により，これらの情報を活用する．しかし，解析ツールの構築において，以下のように，対処困難な内容があった．

第一に，実装プラットフォームの多様性とそれによる解析の一般化の難しさが挙げられる．HTML，JavaScript，テンプレートコード，C#の多言語利用や，Script Manger や Entity Framework などの多フレームワーク利用，ラムダ式などの特定のプログラミング言語固有の複雑な構文規則が，これらの全てに完全に対応する静的解析を困難にしていた．

第二に，開発者によるコーディング規約の不統一がある．例えばデータベースからデータを取得する場合に，SQL 文を使用するケースや，ラムダ式による Where メソッドを使用するケースの双方が存在していた．また，新技術の登場により開発途中で利用技術を変更していた箇所も存在した．このような不統一により，簡易な解析では検出漏れが生じていた．とはいえ，完全な解析ツールを開発することは，前述の多技術併用の問題からバリエーションが多く，開発負担も大きいため，現実的ではない．簡易な解析でも，その有効性を高められるように，利用技術やその使い所を明確に定めるようにコーディング規約を決定する必要がある．

第三に，セキュリティ属性が entity の構造と分離されて定義されているケースがあったが，開発者以外では直感的に分かり難い問題があった．そのため，当該属性が，分割統治の観点から分離することが適切であっても，派生属性としてでも entity から分離せずに定義することが重要であると考えられる．

(2) 将来の応用方法

システムのマイグレーション時に，本研究で示した仕様定義方法により，仕様の十分でないシステムの仕様を整理することで，旧システムおよび新システムのソースコードを検証することが期待される．

3.2 研究目標 2「変換支援ツールの設計と開発」

3.2.1 当初の想定

(1) 研究内容

①概要

研究目標 1「仕様定義に基づく検査モデル・検査方式の生成方法の定義」の検討に基づき，ソースコードからモデル検査用のシステムモデルへの変換支援ツールの設計（C#からの UPPAAL モデル生成）を行う．仕様書に基づき，外注により変換支援ツールを開発する．

②想定される課題と解決策

Source2UPPAAL では Java のソースコードから段階的に，UPPAAL モデルを生成し，メソッドへのモデルおよび値の割り当てを支援している．これらの割り当ては，手動で開発者が行うが，自動化できる部分を検討する必要がある．Java で記述された小規模事例の検査を実施し，モデルの生成およびこれらの割り当て作業について検討する．LUMINOUS のロジック部は C#で記述されているため，LUMINOUS の構成を整理し，C#から UPPAAL モデルの生成方式を定義する．研究目標 1 の検討に基づき，割り当てモデルの生成規則を仕様に定義する．

(2) 当初の到達目標と期待される結果

ソースコードからモデル検査用のシステムモデルへの変換支援ツールの設計を行い，外注により変換支援ツールを開発する．開発した変換支援ツールを用いて，研究目標 4 の「LUMINOUS のセキュリティ要件の検証」を実施できるようにする．

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 小事例（大学授業課題・与信管理業務事例）を Source2UPPAAL で検査し、検査手順ならびに割り当て作業について確認する。
② 上記の事例開発を踏まえて、変換支援ツールの仕様書を作成する。
③ 仕様書に基づき、変換支援ツールの開発を発注する。

(2) 具体的な研究成果の内容

2012 年度の本事業において開発した Source2UPPAAL は統合開発環境である eclipse のプラグインであり，2.1.1 節の図 2-1 に示すように Java のソースコードから，値の割り当て，メソッドの割り当て，ユーザ定義モデルに基づく抽象化により，UPPAAL のシステムモデルを段階的に生成する機能をもつツールである．実装の構成は 2.1.1 節の図 2-2 のとおりである．

本研究では，つぎのユースケースをもつ変換支援ツールを 2012 年度と同様にソフトウェアの統合開発環境の１つである eclipse のプラグインとして開発する．旧バージョンの

Source2UPPAAL ではベースモデルの可読性が低いという問題があり，ベースモデル定義における作業効率を改善するための機能も開発する．図 3.2-1 は変換支援ツールのユースケース図である．ここで，「仕様モデル」は，UML 記述によるユースケースに基づいて定義するセキュリティ要件定義テーブルおよびセキュリティ属性のモデルから生成される UPPAAL モデルである．これは，2.1.1 節の図 2-1 の「ユーザ定義モデル」の 1 種である．

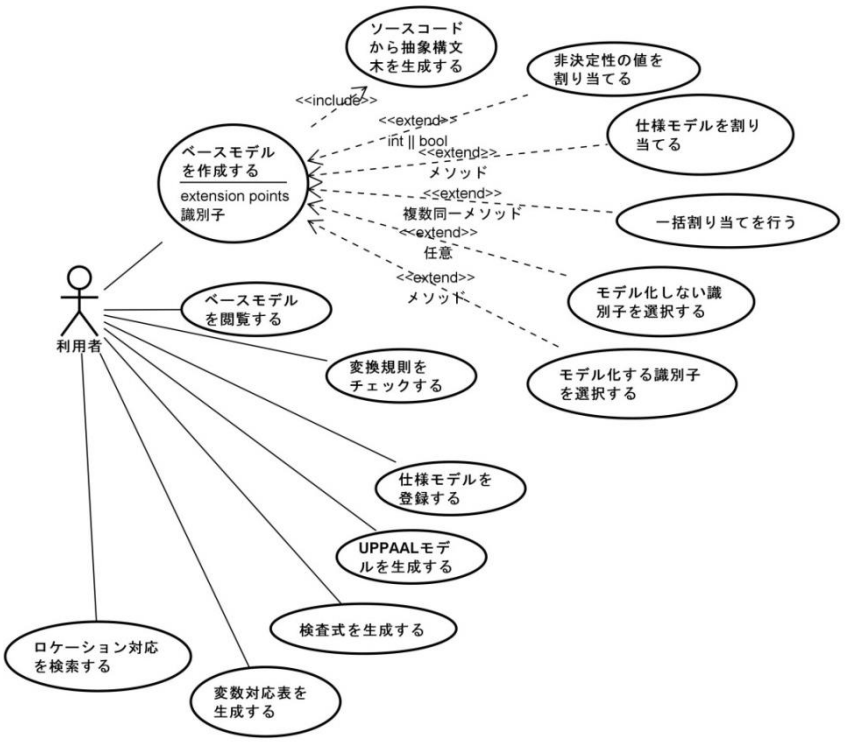


図 3.2-1 変換支援ツールのユースケース

各ユースケースについて説明する．

① ソースコードから抽象構文木を生成する

C#のソースコードから抽象構文木を生成する機能である．

② ベースモデルを作成する

ベースモデルに対し，識別子毎に割り当て等の操作を行い，ベースモデルの段階的な構築を支援する機能である．具体的には，非決定性の値の割り当て・仕様モデルの割り当て・メソッドの選択・変換しない識別子の選択・メソッドへの値および仕様モデルの一括割り当ての機能を持つ．

ベースモデルは検査者が段階的に作成する検査用のモデルで，これが UPPAAL モデルに変換される．変換支援ツールでは C#のソースコードから抽象構文木を生成し，表 3.2-1 のステートメントを選択して取得し，これらをツール上の識別子として取り扱う．

表 3.2-1　識別対象ステートメント

No	ステートメント	
01	MethodInvocation	メソッド
02	Assignment	代入式
03	IfStatement	if文
04	ForStatement	for文
05	WhileStatement	while文
06	VariableDeclarationStatement	変数宣言
07	Infix Exoression	中置計算式
08	Postfix Expression	後置計算式
09	Prefix Expression	前置計算式

各識別子の表示形式は下記のとおりである.

(1)　MethodInvocation

メソッド名称を表示する.

(2)　IfStatement ForStatement WhileStatement

条件式と各ステートメントの種別を表示する. 条件式は(5)に準じる

(3)　Assignment

扱う型, 代入する変数, 式を表示する. 式の表示は(5)に準じる.

(4)　VariableDeclarationStatement

変数定義宣言を表示する.

(5) Infix Exoression,Postfix Expression,Prefix Expression

中置計算式,前置計算式,後置計算式を表示する.

ツールは図 3.2-2 に示すように左ペインと右ペインにより構成する. 右ペインにはプロジェクト内にあるすべてのメソッドを表示し, 左ペインにはモデル化するメソッドを表示する.

初期状態では, 図 3.2-2 に示す通り, 右ペインにプロジェクト内のメソッド一覧と非決定値の一覧を表示する. この時, 左ペインは空白とする.

右ペインの識別子を左ペインへドラッグ＆ドロップすることにより, 図 3.2-3 に示すように, 左ペインにも識別子を表示する. 左ペインの識別子の内, メソッドに相当する識別子をクリックすることで, 図 3.2-4 に示すように, 内部にある識別子を展開することができる.

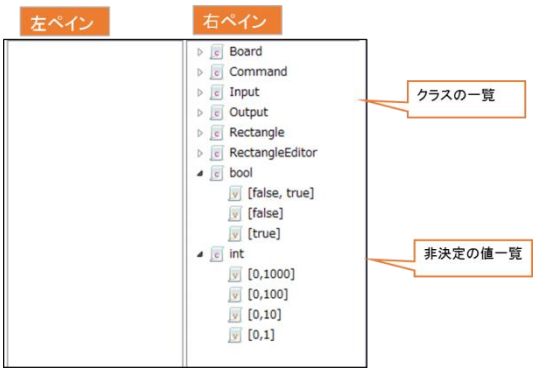


図 3.2-2 ツール初期画面

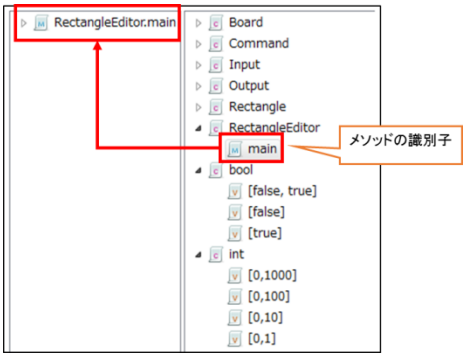


図 3.2-3　ドラッグ＆ドロップによる要素の選択

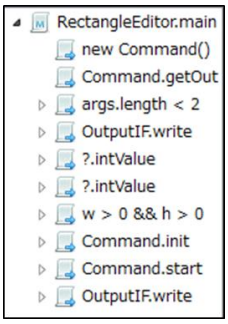


図 3.2-4　識別子展開

③　非決定性の値を割り当てる

式またはステートメントに非決定性の値を割り当てる．割り当て可能な int 型，bool 型の値のパターンは以下のとおりである．

int：0 もしくは 1，0～10（連続した数値の範囲を任意に設定）

bool：true のみ，false のみ，true もしくは false

④　仕様モデルを割り当てる

UPPAAL のモデルをメソッド呼び出しのステートメントに対して割り当てる．左ペイン側の識別子の種類が MethodInvocation の場合，既存の UPPAAL モデル(*.xml)を割り当てることができる．UPPAAL モデルは他の識別子と同様に左ペインに表示して，右ペイン識別子に割り当てる．割り当てられたモデルは元のモデルと UPPAAL の同期処理により連結する．呼び出し元は呼び出し先の処理が終了するまで待機し，呼び出し先の処理終了後に処理を再開するものとする．

⑤　モデル化する識別子を選択する

メソッドの定義を展開し，その振舞いを検査対象とする場合に当該メソッドの識別子を割り当てる．左ペイン側の識別子の種類が MethodInvocation の場合，右ペイン側の MethodInvocation の識別子を割り当てることができる．割り当てられてた MethodInvocation の識別子の階層構造を左ペイン側に展開する．

⑥　モデル化しない識別子を選択する

ステートメントが，プリント文や代入式・条件式でも検査者が不要と判断したものを，モデル化対象から除外することで，検査の効率を上げる．

選択した識別子をモデル化対象から除外し，除外した識別子には，「停止」のマークを図 3.2-5 のように表示する．識別子は階層構造を持つので，ツリー構造のおりたたみ，再展開で識別子を再表示する場合には，「停止」も再表示する．マークはこの限りではない．

指定された識別子は UPPAAL モデル変換時に変換対象にせず，UPPAAL の更新・条件部分に対する反映を行わない．

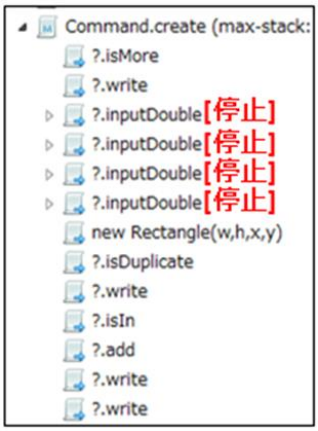


図 3.2-5「停止」表示イメージ

⑦　一括割り当てを行う

あるメソッドに仕様モデルを割り当てる場合，そのメソッドを呼び出しているすべての箇所に同じモデルを割り当てたい場合がある．このような場合に，一括割り当てを行う．左ペインにあるモデル化対象とした同一メソッドに対しては一括して割り当て処理を行う．ここで，同一のメソッドとみなす条件は，参照先＋メソッド名が同じであることである．

以下のような手順で割り当てを行う．

- 左ペインにあるメソッド識別子を一括割り当ての対象として指定する．
- 左ペイン内にある他の同一参照先＋同一名称のメソッドの識別子を一括割り当て対象にする．どこが対象になるかは識別できるようにする．
- 最初のメソッドに割り当て処理を行う．
- 一括割り当て対象になっている他のメソッドに対しても同一の割り当てを行う．

⑧　ベースモデルを閲覧する

ベースモデルの表示並びに，操作の状態の反映に関する機能である．表示に際しては，ベースモデルの構造的特徴を木構造として表現するが，構文要素の特徴は値割り付けや仕様モデルの挿入時に重要な情報である．そこで，適切な割り当てを行えるように，目的のステートメントが理解しやすい構文の特徴を画面に表示する．例えば if 文や for 文の条件式が識別できるようにする．以下に，ベースモデルを構築する画面構成ならびに，表示内

容について説明する.

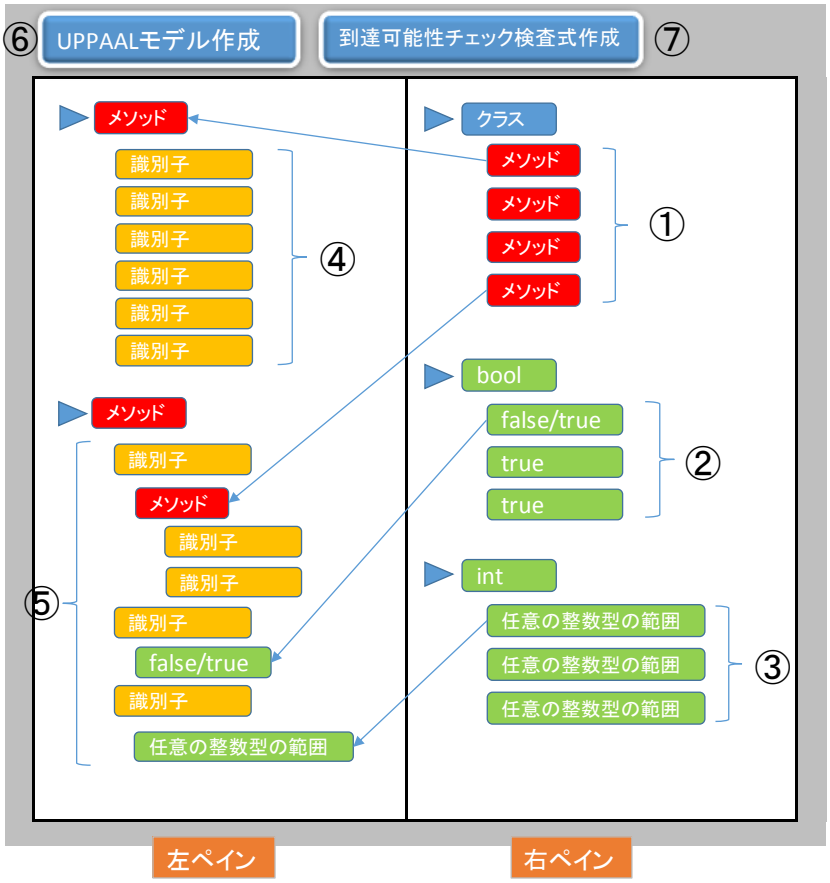


図 3.2-6 ベースモデル表示イメージ

ベースモデルの表示

図 3.2-5 が変換支援ツールの画イメージである．各要素は以下のとおりである．

(1)ツールは図 3.2-6 にあるように左ペインと右ペインにより構成する．

三角印が付記された識別子は展開・折りたたみができる．

(2)右ペインに表示される識別子

- クラスをトップとするメソッド識別子の一群（①）プロジェクト内のクラスすべてを対象とする．
- bool をトップとする bool 型の非決定性の値の識別子（②）
- int をトップとする int 型の非決定性の値の識別子（③）

(3)左ペインに表示される識別子

- 左ペインの初期状態は空白である．
- 右ペインのメソッドをドラッグ＆ドロップで左ペインに表示する．
- メソッド識別子をトップとするメソッドの内部構造の識別子の一群（④）
  - ④で表示される識別子は表 3.2-1 で提示したものすべてである．

(4)割り当て可能な識別子

①②③の識別子が④に対して割り当てが可能であり，割り当てた結果が⑤である．

①を割り当てた場合，そのメソッドを構成する識別子を展開する．

(5)モデル作成と検査式作成

⑥の”UPPAAL モデル作成”ボタン押下で検査モデルを作成する.

⑦の”到達可能性チェック検査式作成”ボタン押下で到達可能性の検査式を作成する.

割り当て済の識別子の明示

割り当ては多数行われるため,どの識別子の割り当てが終了しているかがわからないと,作業効率が低下する.そこで,割り当て処理がなされている識別子には「済」を表示する.

識別子に割り当て処理を行った場合には,識別子には「済」のマークを図 3.2-7 のように表示する.識別子の割り当て処理を削除した場合,「済」のマークを消す.識別子は階層構造をもつので,ツリー構造のおりたたみ,再展開で識別子を再表示する場合には,割り当て済みの識別子については「済」も再表示する.

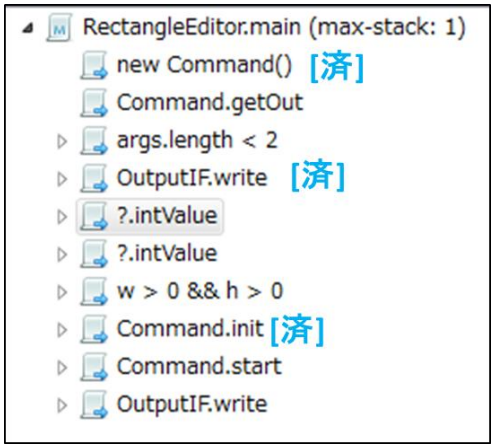


図 3.2-7 「済」表示イメージ

ステートメント種類の表示

ツール上に表示される識別子をステートメント種別ごとに表示を変えて,認識しやすいようにする.例として図 3.2-8 に種別ごとの表示形式を示すが,この限りではない.表 3.2-2 に示した各ステートメントが標記 1～4 を満たす形で識別子を表示する.

表 3.2-2 識別子別表記

ステートメント	表記1	表記2	表記3	表記4
MethodInvocation	メソッドである表記	メソッド名		
Assignment	代入式である表記	扱う型 int/bool/etc	代入する変数名	代入式
IfStatement	if文である表記	条件式		
ForStatement	for文である表記	条件式		
WhileStatement	while文である表記	条件式		
VariableDeclarationStatement	変数宣言である表記	扱う型 int/bool/etc	変数名	(代入式)

()はある場合

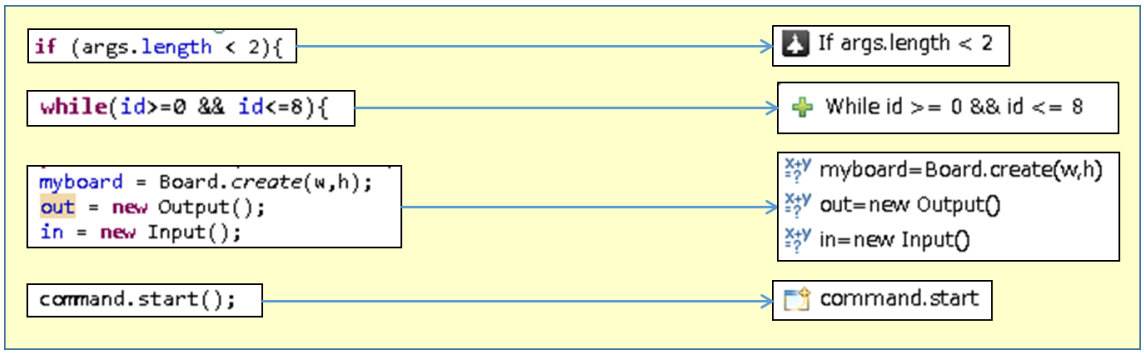


図 3.2-8　変換のイメージ

⑨　仕様モデルを登録する

ユーザ定義の UPPAAL モデルをツールに登録する機能である.

⑩　変換規則をチェックする

ソースコードの扱う型と，UPPAAL モデルで扱う型は異なる．このために，構文によっては,値割り当ておよび仕様モデルの割り当てが行われていないと UPPAAL モデルに変換できない場合がある．このような構文を変換前に通知し，対応が可能なようにするため，UPPAAL のモデルへの変換規則の適用可能性を調べる機能である.

ベースモデルを UPPAAL モデルに変換する際に，「未定義」の文法エラーが生じる識別子をツール上に表示し，変換前に認識できるようにする．下記の手順でチェックを行う.

1. 識別子が代入式もしくは条件式であるかを判定する．代入式でも条件式でもない場合には，処理を終了する.
2. 識別子が代入式か条件式の場合には，もとのステートメントに参照型のオブジェクトが含まれるか否かを判定する．参照型のオブジェクトが含まれない場合には処理を終了する.
3. 参照型のオブジェクトが含まれる場合には，そのオブジェクトに割り当て処理がなされているか否かを判定する.割り当てがされていなければ,該当する識別子に「Error」のマークを図 3.2-9 のように表示する.
4. 上記処理をモデル化対象となる識別子すべてに対して行う.

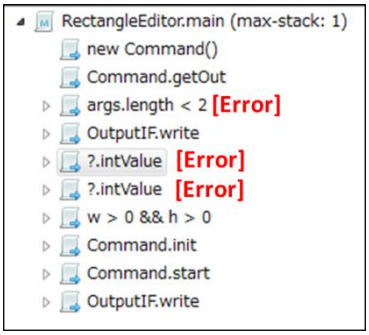


図 3.2-9 「Error」表示イメージ

⑪　UPPAAL モデルを生成する

ベースモデルから UPPAAL モデルを生成する機能である.

変数のモデル化規則

ソースコード上に定義されている変数を UPPAAL 上の変数に変換して定義する.規則は表 3.2-3 および表 3.2-4 のとおりである.

表 3.2-3　型変換規則

型	変換規則
基本型	boolean型⇒bool char型⇒int 数値型(byte、short、int、long、float、double) ⇒int
参照型	割り当てなし⇒　変換しない 割り当てあり⇒　非決定の値の設定(true/falseもしくはint型数値) 割当先のUPPAALモデルからの返値(boolもしくはint型)

※割り当ては 2) で説明した割り当てである.

表 3.2-4　変数種類変換規則

変数種類	変換規則
メンバ変数	基本型:プロジェクト内のものは全てUPPAL上に定義する 参照型:割り当てありの場合UPPAAL上に定義する
ローカル変数	基本型:左ペインへ移動したもののみ定義する 参照型:割り当てありで左ペインへ移動したもののみ定義する

変数名命名規則は以下のとおりである.

- メンバ変数：クラス名＋"_field_"+変数定義のソースコード上の開始ポイント+"_"+ステートメント文字数（例：Command_field_56_6）
- ローカル変数："var_"+インクリメント数値+"_"+変数定義のソースコード上の開始ポイント+"_"+ステートメント文字数（例： var_0_2290_39）
  インクリメント数値の初期値は 0，名称が重複した場合には＋１する.

ステートメントのモデル変換規則

表 3.2-5 の規則に従い，ソースコードのステートメントを UPPAAL モデルに変換する.

(1)ロケーション名命名規則

- 開始ロケーション：START
- 終了ロケーション：END
- その他："block_"+そのステートメントが含まれる上位のブロック開始ポイントと文字数+ステートメントの開始位置+文字数

(2)ステートメントの変換規則

表 3.2-5　ステートメント変換規則

ステートメント	割り当て有無	モデル化対象	ステートメントが扱う型	反映先	備考
MethodInvocation	なし	対象外	すべて	-	
メソッド	あり	対象	すべて	UPPAALのエッジの更新	
Assignment	なし	対象	すべて	UPPAALのエッジの更新	
代入式	あり	対象	すべて	UPPAALのエッジの更新	
VariableDeclarationStatement	なし	対象	int/bool	UPPAALのエッジの更新	
変数宣言	なし	対象外	int/bool以外	-	
	あり	対象	すべて	UPPAALのエッジの更新	
IfStatement+Infix Exoression	あり	対象	すべて	UPPAALのエッジの条件	Prefix,Postfixも同様
if文＋中値式	なし	対象	すべて	UPPAALのエッジの条件	
ForStatement+Infix Exoression	あり	対象	すべて	UPPAALのエッジの条件	Prefix,Postfixも同様
for文＋中値式	なし	対象	すべて	UPPAALのエッジの条件	
WhileStatement+Infix Exoression	あり	対象	すべて	UPPAALのエッジの条件	Prefix,Postfixも同様
While文＋中値式	なし	対象	すべて	UPPAALのエッジの条件	

※割り当ては 2) で説明した割り当てである.

UPPAAL モデルの作成

(1)作成する単位

メソッド単位でモデル（UPPAAL のテンプレート）を作成する.

(2)モデル名命名規則

モデルの名称はメソッドの名称を使用する.

- ソースコードから作成するもの
  "System_"+メソッド名（空白は"_"置き換え）
- あらかじめ用意しておく UPPAAL モデル
  "Act_"+任意の文言

(3)ステートメントのモデル化（図 3.2-9 参照）

- 開始
  "START"のロケーションから開始する.上位モデルの同期呼び出しを受けて遷移を開始する.
- 終了
  "END"のロケーションで終了する.上位モデルへの同期呼び出しを行い,遷移を戻す.
- ステートメントのつなげ方
  ステートメントはロケーションとエッジの組み合わせに変換する. これを接続していくことでモデルを構成する.

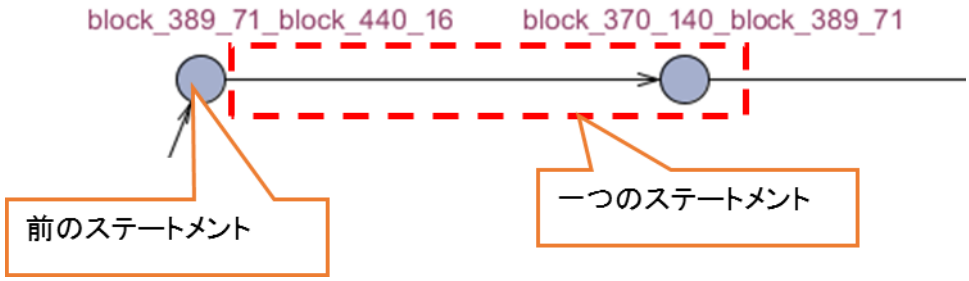


図 3.2-9　ステートメントモデル化例

(4)条件分岐のモデル化

if 文での条件分岐は，モデル上では状態遷移の分岐により表す．UPPAAL のエッジの条件部分に条件式を設定する．条件式は必ず全遷移に設定する．if 文の else の場合は not(条件式)の形式で条件設定を行う．（図 3.2-11 参照）

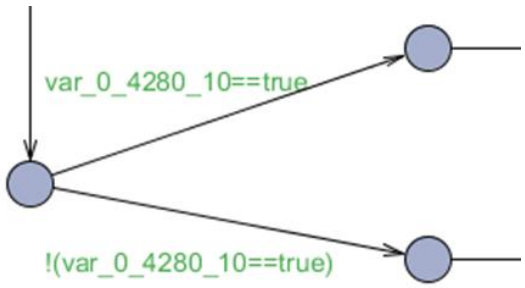


図 3.2-11　条件分岐モデル化例

(5)繰り返し処理のモデル化

for 文，while 文による繰り返し処理はモデル上では状態遷移のループにより表す．ループの基点となるロケーションからでるエッジに条件分岐を用意し，ループから抜ける条件を設定する．（図 3.2-12 参照）

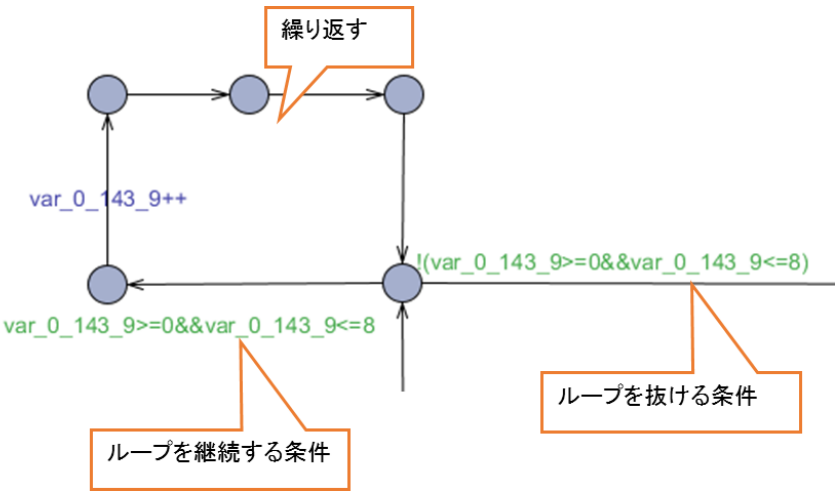


図 3.2-12　繰り返し処理モデル化例

(6)モデル間の連携

モデル間の連携は UPPAAL の同期処理により行う．ツールの左ペインで紐付けられたメソッド間の連携に用いる．（図 3.2-13 参照）

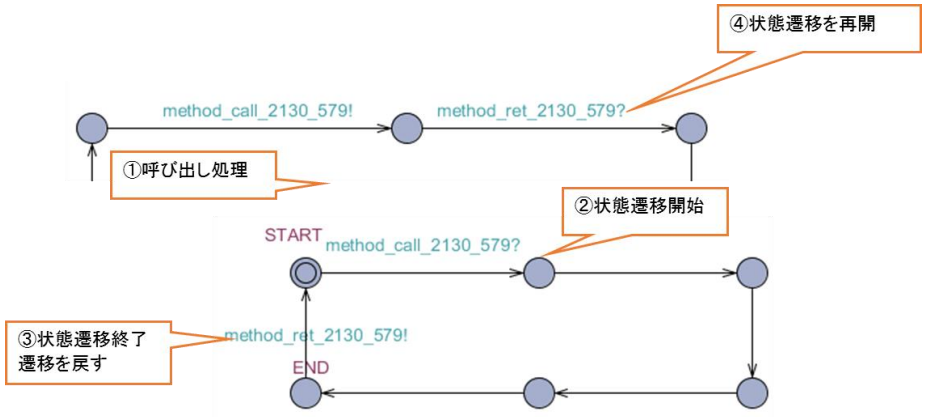
連携の動きは以下のとおりである．

- 呼び出し元は同期処理で，呼び出し先の状態遷移を開始させる．
- 呼び出し元は呼び出し先の状態遷移が終了するまで待機する．
- 呼び出し先は状態遷移が終了したら，呼び出し元の状態遷移を同期処理を用いて再開させる．

同期処理の命名規則は以下のとおりである.

- 呼び出し元⇒呼び出し先 method_call_+呼び出し先開始ポイント+"_"+文字数
- 呼び出し元⇒呼び出し先 method_ret_+呼び出し先開始ポイント+"_"+文字数

呼び出し元上位メソッド



呼び出し先下位メソッド

図 3.2-13 モデル間の連携例

(7)モデルファイルの作成

ソースコードから変換した UPPAAL のロケーション, エッジ, 変数の定義を xml ファイルへ出力して UPPAAL の検査モデルを図 3.2-14 に示す構造で作成する.

- ツール画面上の"UPPAAL モデル作成"ボタン押下でファイル名を入力するポップアップを表示する. ここで, 作成ディレクトリは指定可能, ファイル名は任意とし, 同一ファイル名がある場合はその旨を表示する.
- ファイル名入力後, UPPAAL モデルを作成する.

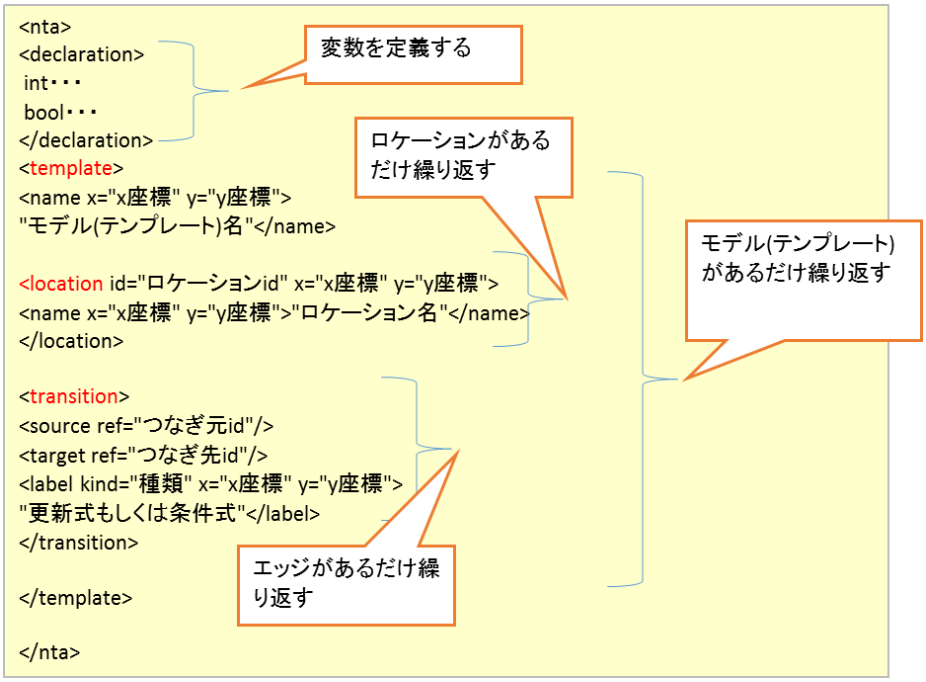


図 3.2-14　UPPAALxml構造

⑫　検査式を生成する

UPPAAL の到達可能性に関する検査式を生成する機能である．作成した UPPAAL の各ロケーションへの到達可能性検査の検査式を作成し，ファイルに出力する．
作成したロケーション名を元につぎの検査式を自動生成する．

E<>　モデル（テンプレート）名.ロケーション名

- ツール画面上の”到達可能性チェック検査式作成”ボタン押下でファイル名を入力するポップアップを表示する．作成ディレクトリは指定可能，ファイル名は任意とし，同一ファイル名がある場合はその旨を表示する．
- ファイル名入力後，検査式のファイルを作成する．

⑬　変数対応表を生成する

UPPAAL モデルの変数とソースコードの変数の対応表を生成する機能である．ソースコードの識別子が UPPAAL モデルのどの変数と対応しているかを理解できるように,変数の対応表を生成する．

ソースコードの識別子が UPPAAL モデルのどの変数と対応しているかを理解できるように，変数の対応表を生成する．表 3.2-6 は対応表の例である．対応表には以下の項目を表示する．

- ソースコードのファイル名
- クラス名
- 変数の修類（メンバ/ローカル）
- ローカル変数ならメソッド名

- ソースコードの変数名
- UPPAAL の変数名

表 3.2-6 変数対応表

ファイル	クラス	メソッド	種類	ソースコード変数	UPPAAL変数
Board.java	Board		メンバ	double width;	int Board_field_83_5
			メンバ	double height;	int Board_field_110_6
Command.java	Command		メンバ	int id	Command_field_56_6
			メンバ	int MAX	Command_field_174_8
		create	ローカル	double w	int var_0_2290_39

⑭ ロケーション対応を検索する

UPPAAL モデルのロケーションからソースコードの位置を検索する機能である．UPPAAL モデルのロケーションがソースコードのどの行と対応しているかを理解できるように，UPPAAL モデル上のロケーション位置から検索する．ロケーション名にはソースコードの位置情報が含まれるので，それを基にソースコード位置を特定して表 3.2-7 のような情報を表示する．

(1)モデル（テンプレート）名でメソッド名を判別する．これにより，ソースコードファイルも確定する．

(2)ロケーション名でソースコード上の位置を判別する．位置を行数に変換する．

(3)以下の内容を表示する．

- ソースコードファイル名
- クラス名
- メソッド名（ロケーションの位置がメソッド内ならば）
- 行数
- ステートメント

表 3.2-7 ロケーション位置表示イメージ

モデル名	ロケーション名	ソースコードファイル名	クラス	メソッド	行数	ステートメント
SYSTEM_Command_start	block_1083_73_block_1103_53	Command.java	command	create	100	double y = in.inputDouble("y = ?");

3.2.3 実用化へ向けた課題と問題点

(1) 課題と問題点

ソースコードが仕様を満たしているかを検証するには，仕様とソースコードの要素の対応付けが重要である．変換支援ツールでは，仕様における検査シナリオに基づき，こうした対応関係の情報を用いて，インタラクティブに変換を支援している．しかし，キーワードによる単純検索のため，部分的にはソースコードを確認する必要がある．必要な情報を確実に絞り込むために，段階的なキーワード検索を行うことや，ソースコードの意味的解析により，より効率的な変換支援を検討する必要がある．

(2) 将来の応用方法

現状では，ソースコードの言語毎に抽象構文木を作成する必要がある．検査モデル作成の過程は言語非依存に開発できるため，多言語対応が期待される．

3.3 研究目標 3「LUMINOUS のセキュリティ要件の定義」

3.3.1 当初の想定

(1) 研究内容

①概要

LUMINOUS の機能仕様を UML 要求分析モデルを用いて整理する．また，データベース定義とソースコード内のオブジェクトマッピングにより，エンティティデータを抽出する．これらのモデルおよび開発時の権限設定表から，LUMINOUS のセキュリティ要件を表形式で定義する．データ抽出や仕様定義の支援方法を検討する．

②想定される課題と解決策

脅威分析は LUMINOUS の資産である学生のレポート・配布される評価レポート・ストレージ内の採点表といった個人情報に対して行う．LUMINOUS では権限として，学生・主担当教員・副担当教員・TA に加えて，学生および教員に設定できるグループの役割とコースの種類がある．そこで，これらの組み合わせに対して，アクセス制御，情報フロー制御，認証，プライバシ等の Common Criteria の 11 のセキュリティ機能コンポーネントの内の 4 つのクラスに関係するセキュリティ要件を扱う．クラスごとに，その定義方法は異なると考えられるので，アクセス制御に関する方針をもとに，他のセキュリティ要件の定義方法を検討し，研究目標 1 の定義にフィードバックする．

(2) 当初の到達目標と期待される結果

LUMINOUS の仕様定義に対し，セキュリティ要件を定義する．一般的にドキュメントのない既存システムの機能要件およびセキュリティ要件を定義する方法も検討する．

3.3.2 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 LUMINOUS のエンティティモデルを VisualStudio の開発環境から得られる ER 図をもとに，UML モデリングツール astah のクラス図に変換する。

② LUMINOUS システムを実際に利用しながら，権限毎に LUMINOUS の機能仕様であるユースケースを UML 要求分析手法に基づき,アクティビティ図を用いて定義する。この際，上記のクラス定義に従い，オブジェクトノードを定義する。

③ UML 要求分析モデルからセキュリティ要件に関わるデータやアクションを抽出するプログラムを開発する。

④ アクセス制御については，上述のように定義されたアクティビティ図からセキュリティ機能方針表のスケルトンを生成し，セキュリティ属性を定義し，セキュリティ属性によりその規則を CC に基づいて定義する。

⑤ その他のセキュリティ要件についても研究目標 1「仕様定義に基づく検査モデル・検査方式の生成方法の定義」で検討する定義方法に従い定義する。

(2) 具体的な研究成果の内容

本節では，(1) のプロセスに従い研究した成果として，事例である LUMINOUS のセキュリティ要件について述べる．LUMINOUS では学生のレポート・評価等の個人情報を扱うため，特にアクセス制御，情報フロー制御，認証，プライバシ等のセキュリティが要求されている．セキュリティ要件はシステムの持つ資産に対する脅威への対策として定義される．脅威への対策として，CC を用いてセキュリティ要件を定義する．SFP の作成方法については 3.1.2 節で述べたので，ここでは LUMINOUS の機能と資産，資産に対する脅威について説明する．

① LUMINOUS の構成

LUMINOUS のユーザには以下の種類がある．

- 教員:本学の教員であり,本システムにより主担当教員としてコースを開設できる．
- 大学院生：本システムにより，講義を履修または聴講※するとともに TA としてコースに参加することができる．
- 学生：本システムにより，講義を履修または聴講することができる．
- 管理者：本システムの管理を行うためのユースケースと主担当教員の代行としてコースの作成のユースケースを利用できる．

ユーザには，システム登録時に，以下のように，管理者・主担当教員・履修生の権限が与えられる．権限をもつことにより，利用可能なユースケースが決定する．

教員：主担当教員と履修生の権限をもつ．さらに，コースにおける設定により，副担当教員の権限をもつことができ，履修生の権限を制限して聴講生の権限になることもある．

大学院生:履修生の権限をもつ.コースにおける設定により TA の権限をもつことができ，履修生の権限を制限して聴講生の権限になることもある．

学生：履修生の権限をもつ．コースの設定により履修生の権限を制限して聴講生の権限になることもある．

管理者：管理者の権限と，コース設定のサービスを提供する意味での主担当教員の権限をもつ．

※聴講とは正規の履修生ではなく，履修生として利用できる「コースを利用する」ユースケースが制限されている履修生を指す．

LUMINOUS のユーザである学生・教員・管理者は，それぞれ，以下の機能を持つ．

□ 学生側機能

- コース選択
- 連絡閲覧
- 教材（閲覧・ダウンロード）
- レポート提出・確認
- アンケート回答・閲覧
- 評価閲覧
- BBS 投稿・閲覧
- トップページへの情報提示

□　教員側機能

- コース作成・変更・削除
- 連絡作成・変更・削除
- 教材登録・変更・削除・閲覧・ダウンロード状況確認
- レポート作成・変更・削除・閲覧・ダウンロード・締め切り延長・提出状況ダウンロード
- 評価登録・削除
- 履修者確認・パスワード初期化・履修者名簿更新・グループ作成・更新・削除
- アンケート作成・変更・削除・閲覧・締め切り延長・集計・集計結果ダウンロード・提出状況ダウンロード・質問項目インポート・エクスポート
- TA の登録・削除・閲覧
- BBS 質問回答・閲覧
- ストレージ作成・登録・閲覧．ファイル登録・更新・削除

□　管理者側機能

- ユーザ登録・変更
- 疑似ログイン

上述のとおり，LUMINOUS のアクターには，学生・主担当教員・副担当教員・TA の 4 つの役割がある．さらに，学生および教員にはグループの役割を設定できる．コースは通常の授業・演習に相当するコースと，すべての学生が自由に参加できる注目のコースの 2 種類がある．仕様として，想定されたシステムの権限表は表 3.3-1 のとおりである．

表 3.3-1　権限表

権限一覧			正規コース		注目のコース	
カテゴリ	画面名	アクション	副担当	TA	副担当	TA
連絡	新規作成	作成ボタン	可	不可	可	可
	編集	編集確定ボタン	可	不可	可	可
	一覧	一覧表示	可	不可	可	可
	一覧	削除ボタン	自分のもののみ可	不可	自分のもののみ可	自分のもののみ可
教材	新規作成	作成ボタン	可	可	可	可
	編集	編集確定ボタン	可	可	可	可
	一覧	一覧表示	可	可	可	可
	一覧	削除ボタン	自分のもののみ可	自分のもののみ可	自分のもののみ可	自分のもののみ可
レポート	一覧	一覧表示	可	可	可	可
	一覧	新規レポートボタン	可	不可	可	可
	一覧	問題編集ボタン	可	不可	可	可
	一覧	締切時間延長ボタン	可	不可	可	可
	一覧	ダウンロードボタン	可	不可	可	可
	問題編集	新規問題ボタン	可	不可	可	可
	問題編集	削除ボタン	自分のもののみ可	不可	自分のもののみ可	自分のもののみ可
アンケート	新規作成	作成ボタン	可	不可	可	可
	アンケート編集	編集確定ボタン	可	不可	可	可
	アンケート編集	削除ボタン	自分のもののみ可	不可	自分のもののみ可	自分のもののみ可
	一覧	一覧表示	可	可	可	可
	一覧	質問編集ボタン	可	選択されたTAのみ	可	可
	一覧	締切時間延長ボタン	可	不可	可	可
	一覧	ダウンロードボタン	可	選択されたTAのみ	可	可
	一覧	集計ボタン	可	選択されたTAのみ	可	可
履修者	履修者一覧	一覧表示	可	可	可	可
	履修者一覧	パスワード初期化ボタン	可	可	可	可
	履修者グループ作成	作成ボタン	不可	不可	不可	不可
	履修者グループ一覧	一覧表示	可	可	可	可
副担当	選択者一覧	一覧表示	不可	不可	不可	不可
	副担当登録	選択チェックボックス	不可	不可	不可	不可
TA	選択者一覧	一覧表示	不可	不可	不可	不可
	TA登録	選択チェックボックス	不可	不可	不可	不可

図 3.3-1 は LUMINOUS の教員のコース管理のトップ画面である．教員は，ここからコースの作成等のコース管理を行うことができる．コース一覧からは，各コースの概要を見ることができ，コース名をクリックすることで，図 3.3-2 のようにコース内の編集が可能となる．

図 3.3-1　LUMINOUS トップ画面（教員）

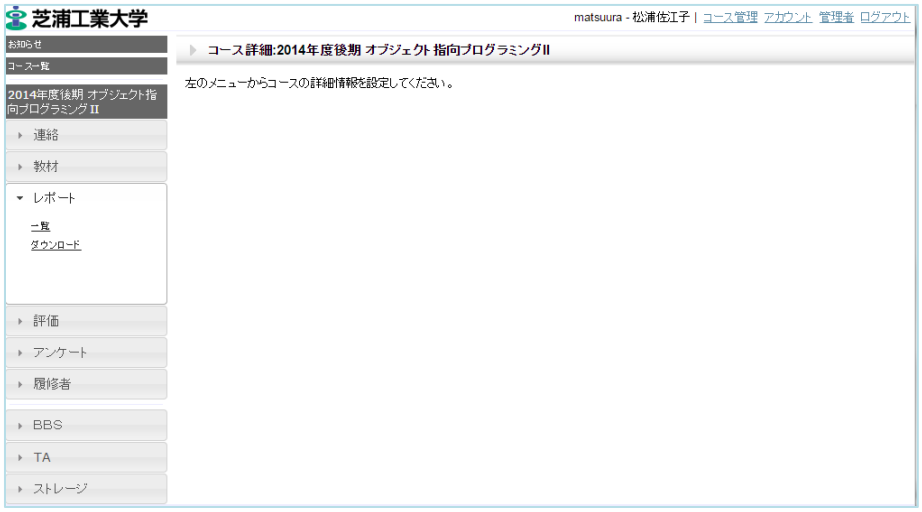


図 3.3-2　LUMINOUS コース・トップ画面（教員）

② LUMINOUS の資産と脅威

LUMINOUS では，上述の機能に対して，以下のようなデータをもつ．セキュリティ要件を考える上で，これらの資産としての意味を検討する．

CC においては「資産の多くは情報の形をとり，情報の所有者が規定した要件を満たす IT 製品によって保存されたり，処理されたり，伝送されたりする．情報の所有者は，このような情報の可用性，まき散らし，および改変を厳しく管理し，対抗策によって資産を脅威から保護することが必要になろう．」と書かれている．LUMINOUS は資産として表 3.3-2 の情報をもつ．

表 3.3-2　LUMINOUS の資産

資産		内容
コース		授業や演習の単位であり，以下の情報を扱う.
	連絡	コースに関する連絡事項である.
	教材	コースの教員が提供する教材ファイルまたはアーカイブである. コースの履修者またはそのグループに対して提供する. 許可された学生がダウンロードすることができる.
	レポート	コースの課題に対して学生が提出するファイルまたはアーカイブである.
	評価	レポートに対する教員の評価レポートまたは評価の集計データである. 評価レポートは指定された学生個人のみがダウンロードできる.
	アンケート	コースで行われたアンケートの質問内容や集計データである. 教員の指定により，選択された質問と回答が公開できる.
	履修者	コースの履修者データである.
	BBS	コースにおける学生の質問に対して，教員が回答した話題である. 質問や回答にはファイルを添付することができる.
	TA	コースに登録された学生のデータである.
	ストレージ	コースの教員およびTA間でのみ共有するデータであり，履修者には非公開である.

ISO/IEC27001 では情報セキュリティを以下のように定義している.
「情報の機密性、完全性、可用性を維持すること．さらに，真正性，責任追跡性，否認防止および，信頼性のような特性を維持することを含めてもよい.」ここで，機密性（Confidentiality）とは,アクセスを認可された者だけが情報にアクセス出来ることを確実にすることである．完全性（Integrity)とは，情報および処理方法が正確であること，および完全であることを保護することである．可用性（Availability）とは，認可された利用者が必要な時に情報および関連する資産にアクセスできることを確実にすることである．本研究では機密性の観点から，LUMINOUS の資産に対するセキュリティを検討する.
LUMINOUS における利用者の役割には，学生・主担当教員・副担当教員・ティーチングアシスタント（TA）4 つがあり，表 3.3-1 の権限表により，まず，第一段階として，上記の資産へアクセスする機能の認可を定める．これはユースケースレベルの機能である．つぎに，機能を認可した後に，特定の情報へのアクセスをその情報の属性によって制御する．これがアクセス制御である.情報フロー制御はその情報の Read 機能の結果としてアクセスできる情報をその属性によって制御するものであると考えられる.

「コース」はその担当教員 ID と履修者学生の ID により，それ以外のユーザから「コース」内の資産を保護する．以下，コース内の各資産に対するセキュリティを説明する.
「連絡」はコースの教員・学生・TA のすべてに対して制限されない情報である.
「教材」は教員が登録することで，学生が閲覧できる．学生に「グループ」の属性を付加することで，グループ外の学生は閲覧できないようにする.
「レポート」は個々の学生が投稿するもので，教員は投稿者全員のレポートを閲覧・ダウンロードできる．当該レポートを投稿した学生以外の学生はレポートのもつ投稿学生の ID 属性の識別により，これを閲覧することはできない.
「評価」には個々の学生に配布されるレポートがあり，教員が指定した ID 以外の学生からは閲覧できない.
「アンケート」は教員が作成し，学生が回答する．回答の集計結果を教員の指定した項目に従って，公開することで，履修者の学生はこれを閲覧できる．ただし，回答者の氏名は秘匿される.

「履修者」は教員が履修者名簿を設定することで，名簿に登録されていない学生からはコース自体が見えなくなる．
「BBS」は学生が質問し，教員が回答することで生成される「話題」という情報をもつ．質問や回答にはファイルを添付することができ，当該学生や教員が許可した履修者は，それをダウンロードすることができる．教員の回答や添付ファイルが不必要に質問者以外の履修者に公開されてはいけない．さらに，学生の質問が公開されても個人が特定できてはならない．このために，教員は回答や添付ファイルに対して，その公開／非公開を設定することができる．
「TA」はコースに対するアクセス権限を設定するものである．教員によってコースに登録された TA がコースの資産にアクセスが可能となり，それ以外の学生は TA としてはコースの内容を見ることはできない．
「ストレージ」はコースに関わる教員・TA 間での情報共有の仕組みである．これ以外のユーザがアクセスできてはならない．TA は教員の設定により，ストレージ内の情報へのアクセスの許可が設定されている場合のみ，これらの情報の登録やダウンロードができる．

これらの要件を 3.1 節で述べたように，LUMINOUS の要求分析モデルに対して，表 3.1-1 のようにセキュリティ要件を定義する．

ユースケースは，これらの資産を扱う機能である．システムの操作を通じて，LUMINOUS の要求分析モデル（ユースケース図，各ユースケースのアクティビティ図，クラス図）を定義する．この際，各ユースケースの使用するエンティティデータが上述のどの資産であるかを特定し，表 3.1-2 を抽出する．サブジェクトとオブジェクトに対して，それぞれ必要なセキュリティ属性を定義する．さらに，セキュリティ属性の満たすべき性質を図 3.1-17 のように，ステートマシン図で定義する．CC のセキュリティコンポーネントを参考にして，設定したセキュリティ属性を用いてルールを定義する．この段階では，ソースコードから抽出したエンティティクラスのモデル図により，エンティティクラスとそのセキュリティ属性については仕様とソースコードが結びついている．そして，3.4 節で述べるように，UI 要素と UI コードを対応付けることにより，対象ソースコードのセキュリティ要件を検証することができる．

3.3.3 実用化へ向けた課題と問題点

(1) 課題と問題点

ソースコードに対してセキュリティ要件を満たすか否かを検証する際に，表 3.1-1 のルールをソースコードと対応付けるためには，ソースコードのエンティティクラス図からサブジェクトやオブジェクトを特定し，その資産管理のためのセキュリティ属性を特定する必要がある．LUMINOUS の開発において，機能拡張をする際には，本研究と同じく要求分析モデルを定義し，各資産に対して，そのクラスにセキュリティ属性を追加したモデルを基に開発を行った．このため，エンティティモデルには，このクラス定義が，そのまま踏襲されていた．しかし，モデルが存在しなかった部分に関しては，資産に対するクラスにセキュリティ属性を定義していないケースがあった．このような場合には，ソースコードを追って，属性のありかと制御方法を特定する必要がある．

(2) 将来の応用方法

ユースケースに対するセキュリティ要件の整理方法とソースコードの対応付けの方法により，設計が十分ではない場合や，LUMINOUS の開発のように，複数の開発者による，コーディング方針の違いによる，対応付けが一意に決まらないケースが発見された．ソースコードの要件に基づくリファクタリングに応用することが考えられる．

3.4 研究目標 4「LUMINOUS のセキュリティ要件の検証」

3.4.1 当初の想定

(1) 研究内容

①概要

開発したツールを用いて，LUMINOUS のセキュリティ要件の検証を実験する．実験結果を踏まえて，ツールの改善を検討する．

②想定される課題と解決策

LUMINOUS の仕様はユースケース単位で整理する予定であるが，検査を実施する際にモデル化するソースコードの範囲を限定するために，ユースケースの事前条件の定義を検査モデルに組み入れる必要がある．これは検査シナリオとして定義する．また，想定した要件が検査できない場合に，ツールの問題，仕様定義の問題であるかを分析する必要がある．また，ソースコードがセキュリティ要件を完全に満たしている場合は，欠陥を発見できないこともある．この場合には，想定した検査ができているかを，ソースコードを修正して検査する必要がある．

(2) 当初の到達目標と期待される結果

ある程度の規模の実運用システムに対して，ソースコードがセキュリティ要件を満たすかを検証することで，マイグレーションを行う際に，実システムの仕様を再定義し，再構築したソースコードを検証する方法を検討する．

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 検査シナリオを作成し，セキュリティ要件の検証を行い、結果を確認する。

(2) 具体的な研究成果の内容

本節では，(1) のプロセスに従って研究を行った成果を変換支援ツールを用いた検査事例により説明する．

以下のモデルおよびセキュリティ要件の定義は 3.1 および 3.3 で説明した方法により，既知のものであるとする．

- LUMINOUS の要求分析モデル（ユースケース図，各ユースケースのアクティビティ図，クラス図）

- ユースケースの使用するエンティティデータとそのセキュリティ属性の特定
- セキュリティ属性の満たすべき性質を表したステートマシン図
- ソースコードの静的解析により抽出された UI 要素と UI コードの対応表
- 事前条件に従い組み合わされた検査シナリオとそれに対応する SFP のルール定義

3.2 で説明した変換支援ツール Source2UPPAAL を用いて，以下のように検査モデルを作成し，検査を行う．

1) 検査シナリオを「UI 要素と UI コードの対応表（以下，対応表と呼ぶ.)」を用いて，ソースコードからモデル化すべきメソッドおよびその定義場所の情報を抽出する．図 3.4-1 に示すように，該当するユースケースのブラウザ上のページの URL により，対応表を検索し，ページの情報を検索し，エクセルファイルに情報をコピーする．基本的には，このページにあるボタンに対応するメソッドを対応表から検索するが，図 3.4-2 のように，ポップアップ画面で，ブラウザからは URL が不明な画面もあるので，その場合には，対応表のアドレスから，該当するページ情報を検索する．これにより，ユースケースに対応する内部メソッドの特定が図 3.4-3 のように行われる．このユースケースに対応する UI コードを特定した表 3.4-1 を，「UI コード特定表」と呼ぶ．

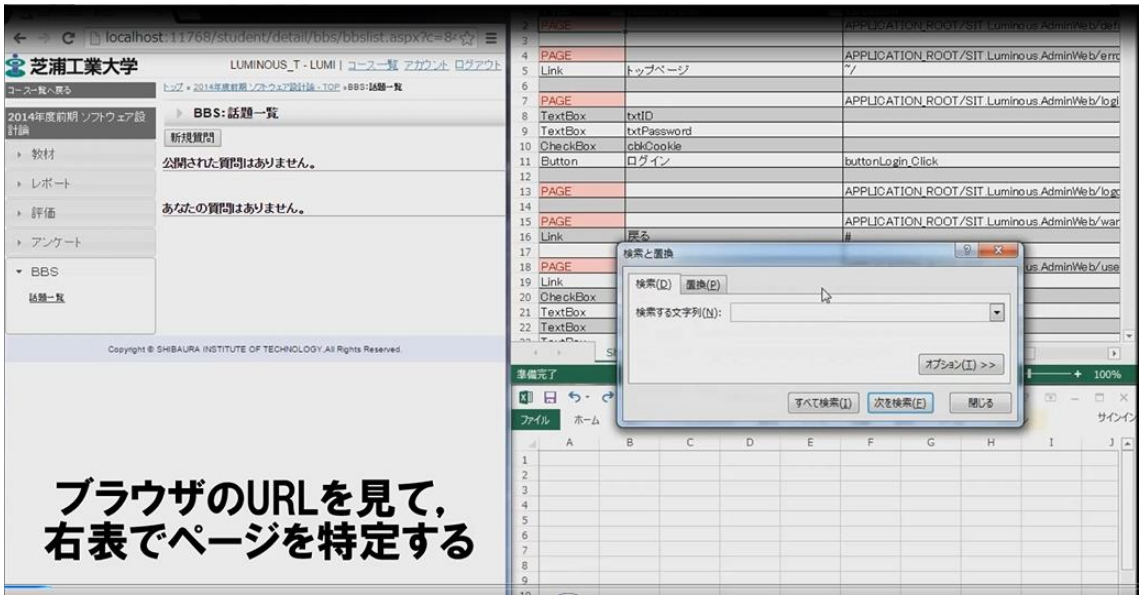


図 3.4-1 検査シナリオの作成

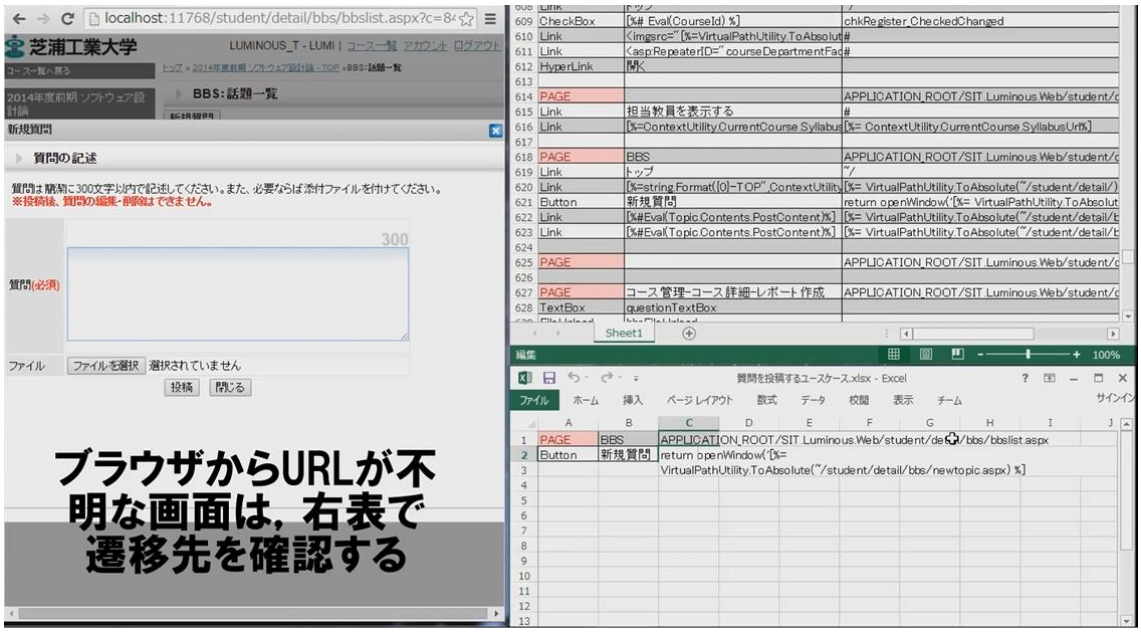


図 3.4-2 検査シナリオの作成（つづき）

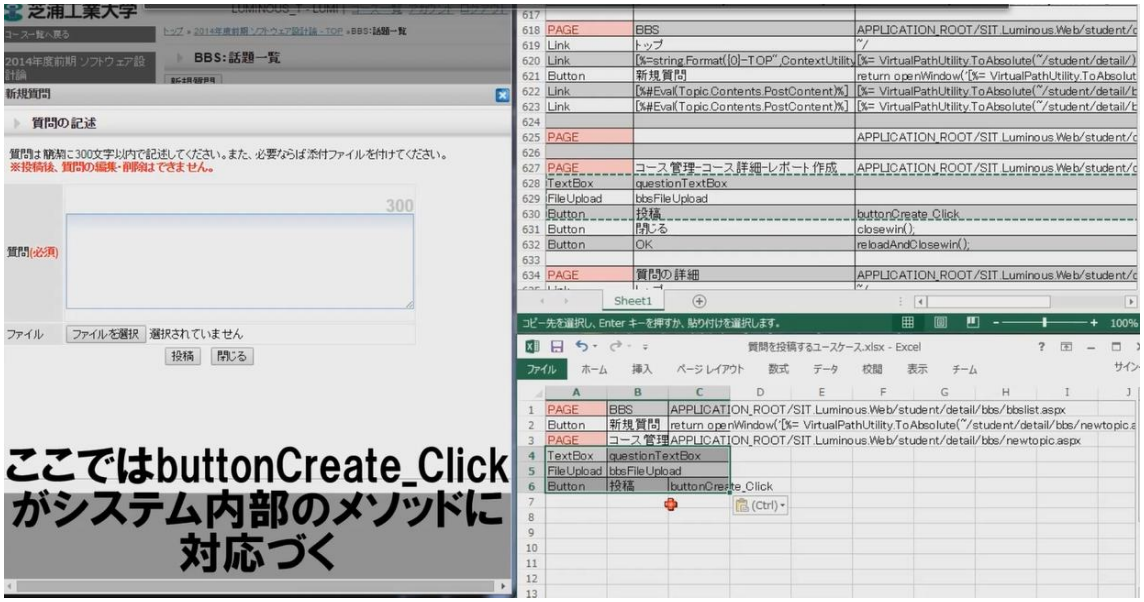


図 3.4-3 ユースケースに対応する内部メソッドの特定

表 3.4-1 UI コード特定表

PAGE	BBS	APPLICATION_ROOT/SIT.Luminous.Web/student/detail/bbs/bbslist.aspx
Button	新規質問	return openWindow('[%= VirtualPathUtility.ToAbsolute("~/student/detail/bbs/newtopic.aspx) %]
PAGE	コース管理-コース詳細-レポート作成	APPLICATION_ROOT/SIT.Luminous.Web/student/detail/bbs/newtopic.aspx
TextBox	questionTextBox	
FileUpload	bbsFileUpload	
Button	投稿	buttonCreate_Click

2) 「UI コード特定表」をもとに，変換支援ツール Source2UPPAAL を用いて，UPPAAL へ変換対象となる検査シナリオを作成する．図 3.4-4 は，2 つのユースケースの組み合わせによるモデルを作成されるための，テンプレートを表している．これを「ナビモデ

ル」と呼ぶ．画面の右側はソースコードの抽象構文木を表している．

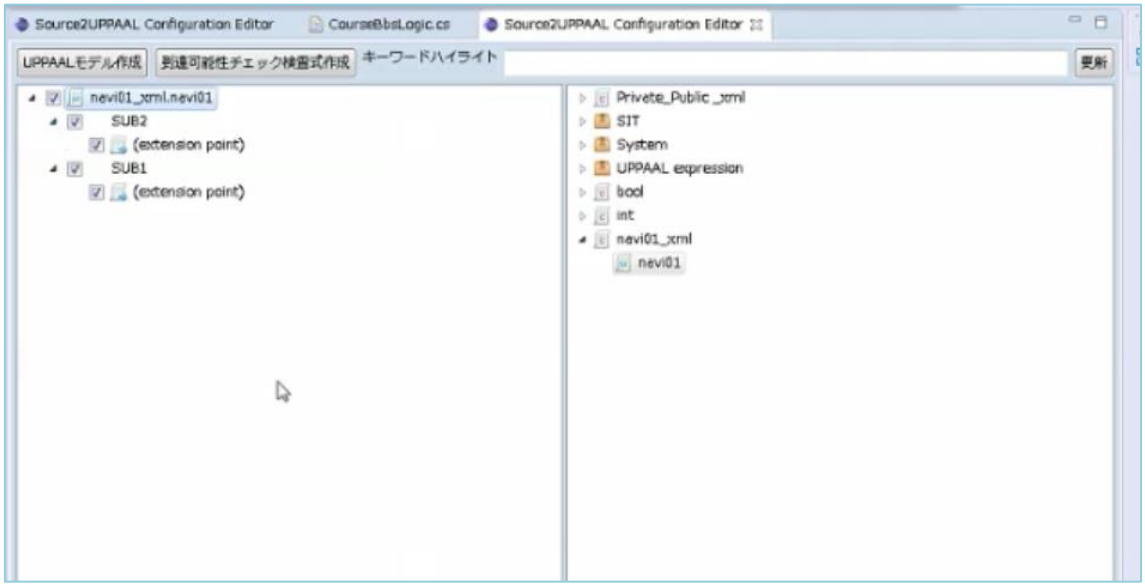


図 3.4-4 Source2UPPAAL における検査シナリオの作成

3) ナビモデルに「UI コード特定表」で見つけたユースケースに対応するメソッド名を右側のソースコードの抽象構文木の階層を辿って探す．この時 UI 対応表で得られた下記のパスを参照して階層を辿る．

SIT.Luminous.Web.student.detail.bbs::buttonCreate_Click

見つけたメソッド buttonCreate_Click を図 3.4-5 のように左ペインのステートメントに割り当てる．これにより，検査対象として，当該メソッドが展開できる．

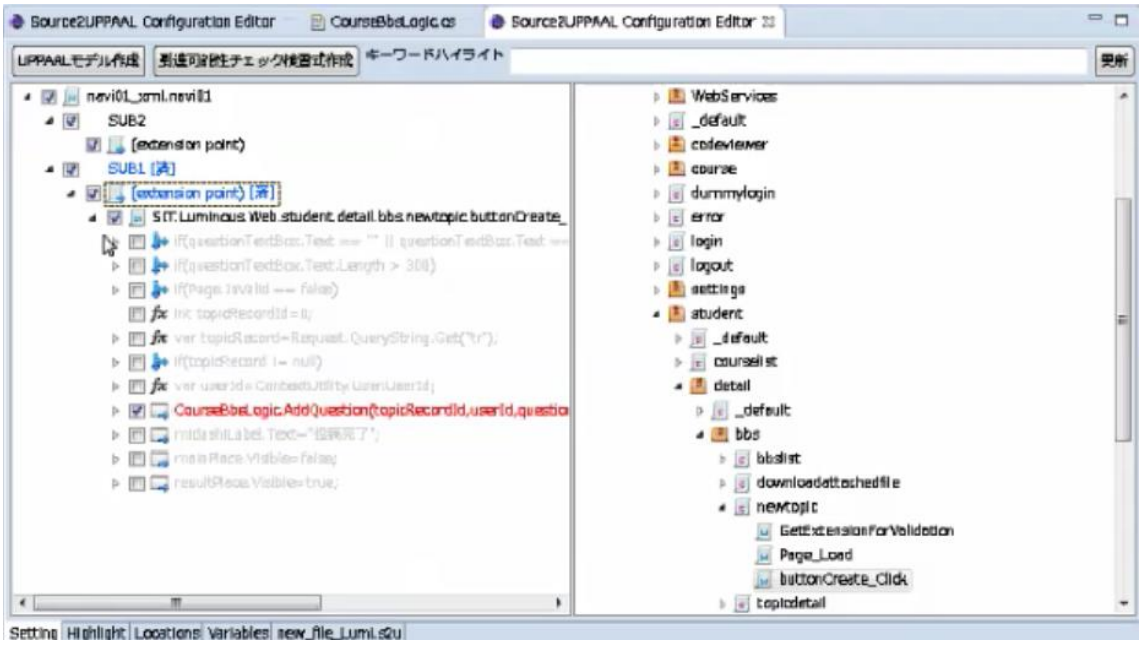


図 3.4-5 機能の割り当て

4) ルールを決定するセキュリティ属性 isPublic をキーワード検索し,これを含むメソッドを探す.この中のユースケースに対応するメソッド AddQuestions を図 3.4-6 のようにステートメントに割り当てる．さらに，ここでは図 3.4-7 のように，ステートメントに非決定値としての真偽値を割り当てる.

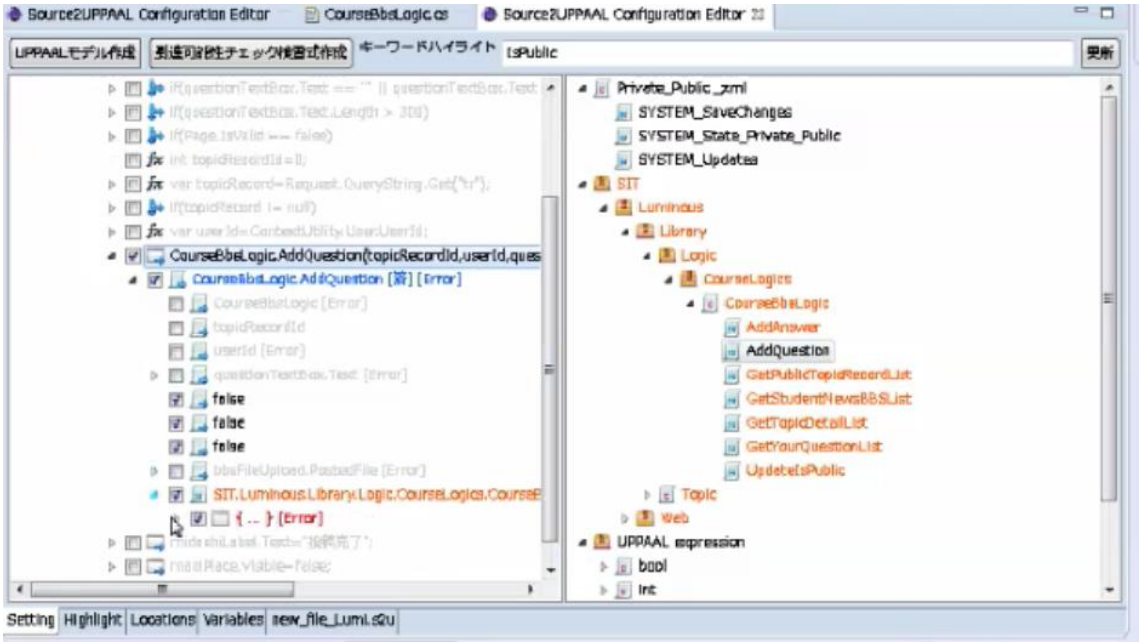


図 3.4-6　メソッドの割り当て

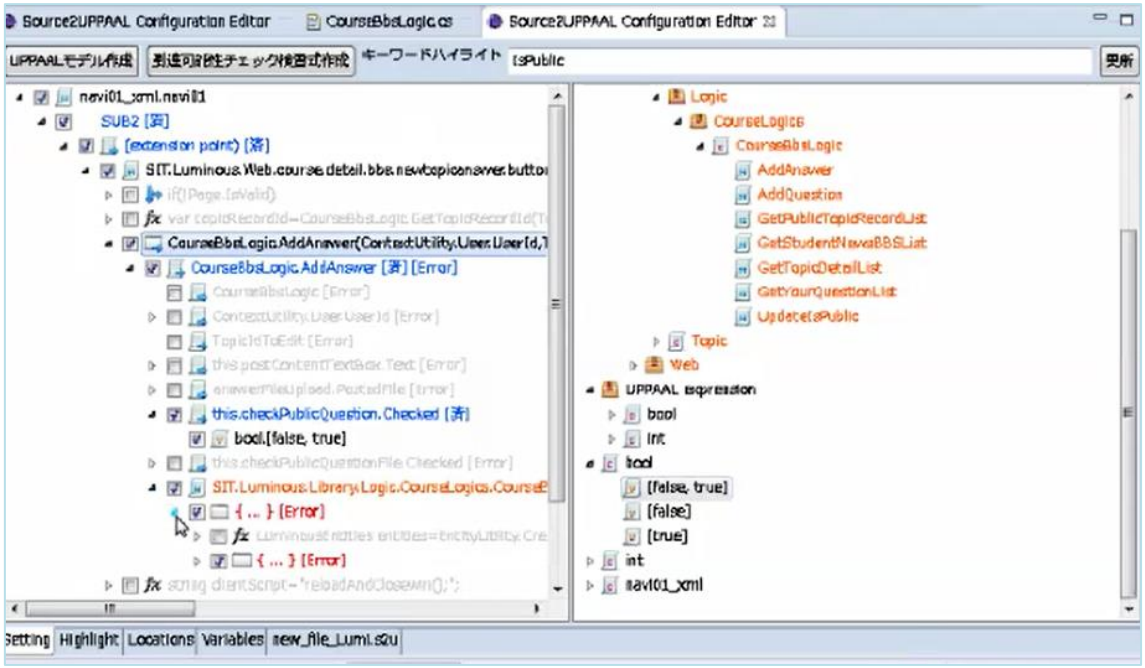


図 3.4-7　値の割り当て

5) セキュリティ属性の満たすべき性質を表したステートマシン図における状態遷移を引

き起こすメソッドのモデルを図 3.4-8 のように割り当てる．このモデルは，図 3.1-20 に示したステートマシン図とソースコードモデルの仲介を行うモデルであり，メソッドとしては C#のデータベース更新 API である．

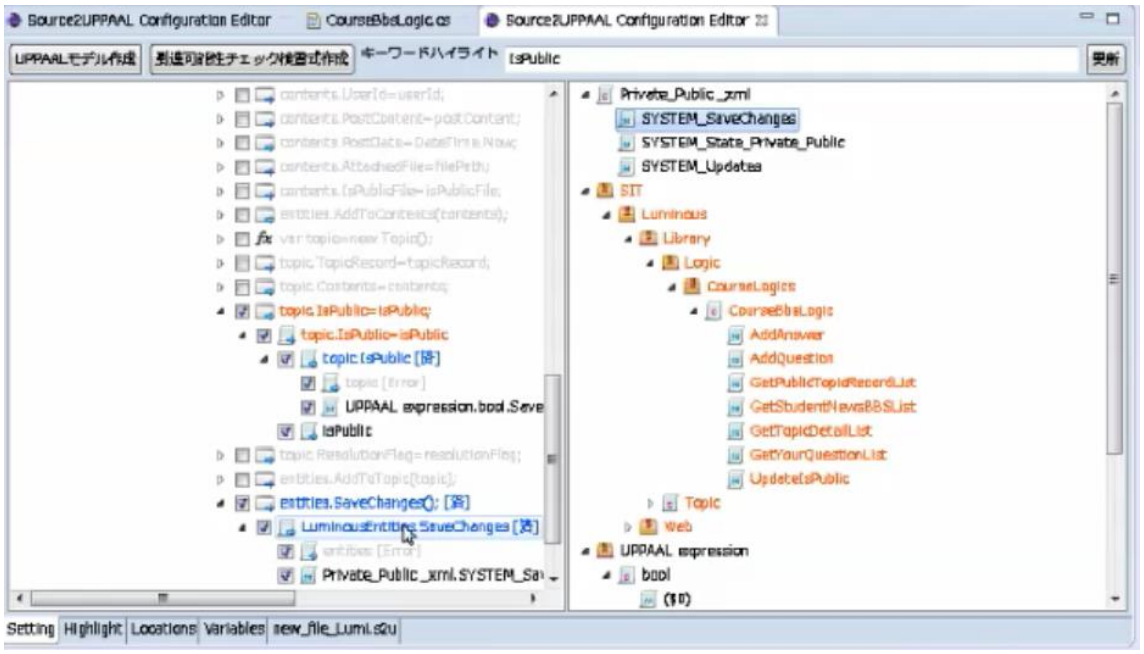


図 3.4-8　属性の状態遷移メソッドの割り当て

6)　割り当てが終了したならば，左上のボタン「UPPAAL モデル作成」により，モデルを作成し，ファイルを生成する．UPPAAL を起動し，このファイルを読み込み，図 3.4-9 のように，検査を実施する．

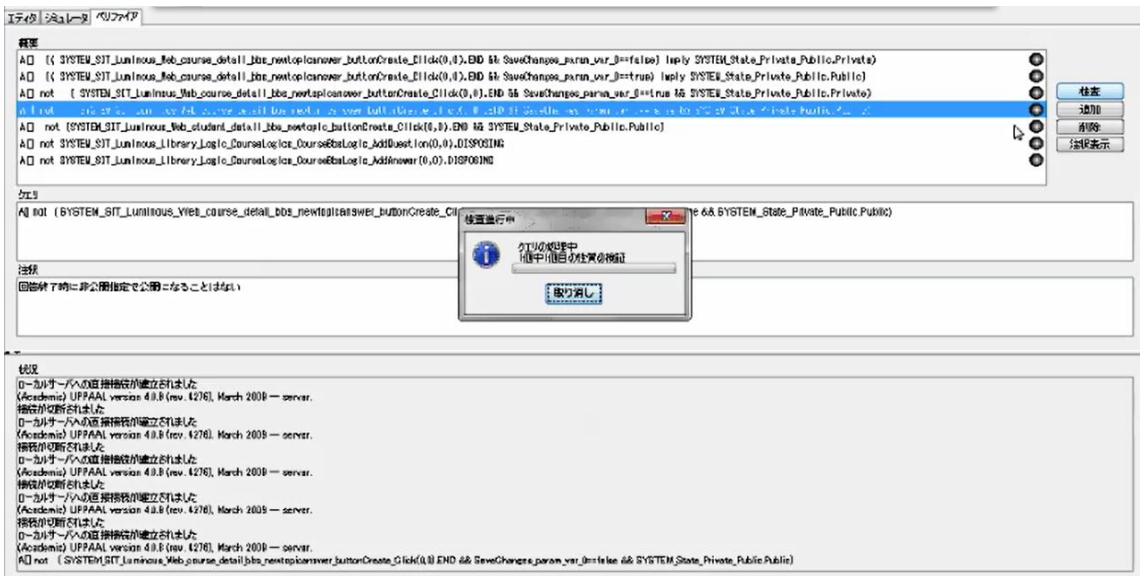


図 3.4-9　UPPAAL によるモデル検査の実施

SFP で定義した各ルールおよび，想定した検査シナリオはすべて満たされた．

3.4.3 実用化へ向けた課題と問題点

(1) 課題と問題点

現状では，仕様モデルから UPPAAL のモデルを検査者が手動で定義している．検査プロセスの前提である以下の項目においては，定義支援は構築されているが，よりその方法を精査し，現場の開発者が容易に利用できるように自動化可能な部分を実装する必要がある．

1) LUMINOUS の要求分析モデル（ユースケース図，各ユースケースのアクティビティ図，クラス図）
2) ユースケースの使用するエンティティデータとそのセキュリティ属性の特定
3) セキュリティ属性の満たすべき性質を表したステートマシン図
4) ソースコードの静的解析により抽出された UI 要素と UI コードの対応表
5) 事前条件に従い組み合わされた検査シナリオとそれに対応する SFP のルール定義

(2) 将来の応用方法

システムのマイグレーション時に，仕様の残されていないシステムの仕様の再構築と，その結果を用いた，ソースコードが仕様を満たしていることの検証により，システムがある特定の性質を満たしていることを保証することが期待される．

3.5 研究目標 5「反例解析支援ツールの設計・開発」

3.5.1 当初の想定

(1) 研究内容

①概要

検査結果で得られた反例解析による原因の絞り込みの支援方法を検討し，反例解析支援ツールを設計・開発する．

②想定される課題と解決策

モデル検査ツールから提示される反例は状態遷移の経路である．検査モデルの各状態はソースコードのステートメントと対応はしているが，そのまま見ても問題点を読み解くことは難しい．反例，すなわち失敗している経路と成功している経路の比較により，失敗している場合の通過点を限定しながら，再検査を行う方法を検討する．失敗経路の通過点を順次限定することで，失敗の原因となるソースコード上のステートメントを再検査により特定する．経路比較をグラフにより可視化する方法を検討し，ツールを試作する．再検査は手動で行い，自動化の方法を検討する．

(2) 当初の到達目標と期待される結果

検査者が確認したい情報の組み合わせを Excel 上に展開してグラフ化するツールを構築し，小事例で反例解析を実施する．

3.5.2 研究プロセスと成果

(1) 研究プロセス

① 小事例（大学授業課題・与信管理業務事例）を Source2UPPAAL で検査し、反例を解析し、ツールの設計を行う。

② ツールを試作する。

③ テストを行い、ツールの改善を検討する。

(2) 具体的な研究成果の内容

本節では (1) のプロセスに従って研究を行った成果を 2 つの反例解析事例により説明する．なお，ここでは UML の仕様を作成せずに．自然言語の仕様から，文章を解析して，デシジョンテーブルを作成して，検査モデルを生成する方式で，検査を行っている．

① 反映解析容易化支援

UPPAAL の検査で検査式に対する反例が得られた場合には，「想定される状態」になった場合と「想定されない状態」になった場合の両者の反例をグラフ化し，それらを比較して差異ができる部分を表示することにより反例解析を支援する．

検査において「想定されない状態」になるかの検査で反例がでた場合，それと対になる「想定される状態」になるかの検査を行う．単純に「想定される状態」の検査を行ってもまったく関係ない状態遷移になる可能性が高い．そのため「想定されない状態」の反例からシステムの仕様で特定できる識別子に着目して,その識別子に関係する UPPAAL の変数を検査式に取り込む．システムの振舞いを特定できる状態をそろえ，類似した状態遷移になるため比較が可能になり，図 3.5-1 に示すように「想定される状態」「想定されない状態」の分岐ポイントを見つけることができる．

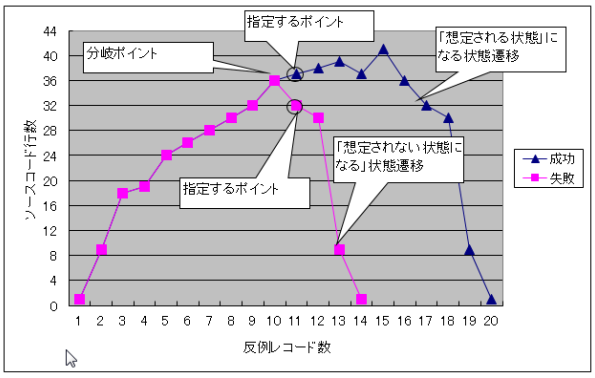


図 3.5-1 状態遷移のグラフによる比較例

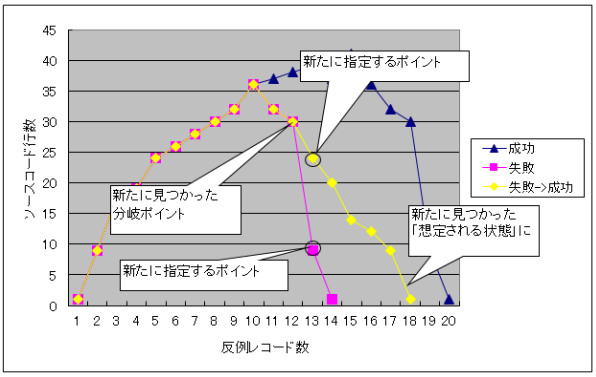


図 3.5-2 状態遷移のグラフによる比較例

しかし，その分岐後で最終的なシステムの振る舞いの状態が特定できるとは限らない．分岐ポイント以降の状態遷移を規定するため，分岐ポイントより先の通過指定ポイントを指定した式を追加した検査式を検査する．これを繰り返すことにより検査結果の精度が高められる．図 3.5-2 のように新たな分岐ポイントが見つかるかもしれず，最終的なシステムの振る舞いの状態が変化しない分岐ポイントが確定するまで検査を行う．

② 反例解析事例 1

1)適用事例概要

検証の対象は，流通の業務における販売システムの一部である受注伝票作成の登録処理において使用される，与信限度による伝票登録のブロック機能である．与信管理マスタに格納されている与信限度を超過する新しい受注を受けた場合，販売システムは受注伝票の登録をブロックする．ここでは，実際の業務で使用されたものに則した仕様とそのサンプルプログラムより不具合を検出した．

2)プログラムの仕様

提示された仕様は以下のとおりである．プログラム本体である「受注伝票登録」の仕様は(1)～(3) であり，(4)～(9) はその内部で使われる各メソッドの仕様である．

(1)新規受注登録拒否状態の場合は，伝票登録は行わず，エラーメッセージを表示する．

(2)新規受注登録拒否状態ではなく(受注新規登録が可能な状態),かつ仮の与信チェックの結果が OVER の場合は，受注伝票登録を行い，エラーメッセージを表示する．この時，与信ブロックフラグを ON にする．

(3)新規受注登録拒否状態ではなく，かつ仮の与信チェック結果が UNDER の場合は，受注伝票登録を行い，さらに与信ブロックフラグが ON の場合は OFF にする．

(4)新規受注伝票拒否状態のチェックは新規受注伝票拒否状態のチェックメソッドを利用する．与信状態が OVER かつブロックフラグが ON の場合 true を返し，それ以外は false を返す．与信状態には新規受注金額は含まれない．

(5)エラーメッセージの表示はエラーメッセージ表示メソッドを使用する．

(6)与信のチェックには仮の与信チェックメソッドを使用する．これは既存の債権と新規の受注金額の合計が与信限度額を超えているかどうかをチェックするメソッドである．与信限度額を OVER する場合は false を返し，UNDER な場合は true を返す．

(7)与信ブロックフラグのチェックには与信ブロックのフラグチェックメソッドを使用する．これは与信ブロックフラグが ON 場合は true を返し，OFF の場合は false を返すメソッドである．
(8)受注伝票の登録は受注伝票登録メソッドを使用する．受注伝票登録により，債権は増加する．
(9)与信ブロックフラグの変更は与信ブロックフラグ変更メソッドを使用する．引数が true の場合は ON に false の場合は OFF に変更するメソッドである．

3)デシジョンテーブル作成

仕様からシステムがとり得る状態遷移をモデル化するため，デシジョンテーブルを作成する．デシジョンテーブルは複数の判定条件とそれに伴うアクションを組み合せた表である．上記の仕様に対して形態素解析を行い，状態・アクション・条件を抽出してデシジョンテーブルを作成する．抽出した状態は表 3.5-1 のとおりである．

表 3.5-1 状態の一覧

状態名	取り得る状態
受注伝票登録許可状態	許可／拒否
与信状態	UNDER／OVER
与信ブロックフラグ	OFF／ON
債権	増加／ニュートラル

一方，抽出した条件は以下の表 3.5-2 の内容になる．

表 3.5-2 条件の一覧

種別	条件
プログラム本体仕様	受注伝票登録許可状態:拒否
プログラム本体仕様	受注伝票登録許可状態:許可 & 仮与信チェック:OVER
プログラム本体仕様	受注伝票登録許可状態:許可 & 仮与信チェック:UNDER
プログラム本体仕様	与信ブロックフラグ:ON
メソッド仕様	与信状態:OVER & 与信ブロックフラグ:ON
メソッド仕様	!(与信状態:OVER & 与信ブロックフラグ:ON)
メソッド仕様	与信状態:OVER
メソッド仕様	与信状態:UNDER
メソッド仕様	与信ブロックフラグ:ON
メソッド仕様	与信ブロックフラグ:OFF
メソッド仕様	引数:TRUE
メソッド仕様	引数:FALSE

抽出したアクションは以下の表 3.5-3 の内容になる．

表 3.5-3 アクションの一覧

アクション
エラーメッセージ表示
受注伝票登録
与信チェック
新規受注伝票拒否の状態のチェック
与信ブロックのチェック
与信ブロックフラグの変更

さらにアクションを実行した結果を表に付加する．“結果”自体が遷移の条件になる場合が考えらえる，その場合は“条件”と“状態“は同一になる．

プログラム本体部分に関するプログラム本体の仕様(1)(2)(3)を基にデシジョンテーブルを作成する(表 3.5-4)．このプログラムが取り得る受注伝票登録許可状態・与信状態・与信ブロックフラグの組合せが判明する．

表 3.5-4 プログラム本体　デシジョンテーブル

項目	種別	仕様(1)	仕様(2)	仕様(3)	仕様(3)
受注伝票登録	アクション	実行しない	実行する	実行する	実行する
エラーメッセージ表示	アクション	実行する	実行する	実行しない	実行しない
与信ブロックのチェック	アクション	実行しない	実行する	実行する	実行する
与信チェック	アクション	実行しない	実行する	実行する	実行する
新規受注伝票拒否の状態のチェック	アクション	実行する	実行する	実行する	実行する
与信ブロックフラグの変更	アクション	実行しない	実行する	実行しない	実行する
受注伝票登録許可状態	条件	拒否	許可	許可	許可
仮与信チェック	条件	Nothing	OVER	UNDER	UNDER
与信ブロックフラグ	条件	Nothing	Nothing	OFF	ON
与信ブロックフラグ	結果としての状態	ON	ON	OFF	OFF
受注伝票登録許可状態	結果としての状態	拒否	許可	許可	許可
与信状態	結果としての状態	OVER	OVER	UNDER	UNDER
債権	結果としての状態	変化なし	増加	増加	増加

メソッドについても同様にデシジョンテーブルの作成を行う．

4)仕様モデルの作成

与信ブロックフラグのモデルの作成を例に説明する．与信ブロックフラグの取り得る状態は表 3.5-1 より OFF/ON のどちらかということが判る．メソッドのデシジョンテーブル(表 3.5-3)よりどのアクションが関係するかは明確に判る．検査モデルへのアクションが及ぼす状態遷移の反映の方針はつぎのとおりである．

(1)アクションが直接状態変化を起こすならば，状態->状態の遷移を引く．

(2)アクション＋非決定性で状態が決まる場合は,アクションで遷移を開始し非決定性で状態を決定する遷移をひく．

(3)アクション＋他状態の組合せで状態が決まる場合は,アクションで遷移を開始し条件分岐で状態を決定する遷移を引く．

(4)アクションにより状態が変化しない場合は，元の状態に戻る遷移を引く．

与信ブロックフラグのモデル(図 3.5-3)は(1)と(4)が該当する．このモデルは ON と OFF の二つの状態を持ち,「与信ブロックフラグの変更」時の ON->OFF，OFF->ON の状態遷移と「与信ブロックフラグのチェック」時の ON->ON，OFF->OFF の状態遷移で構成される．

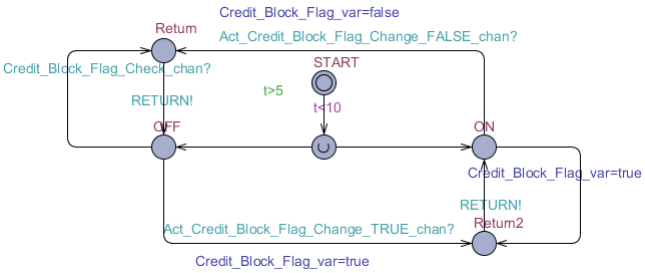


図 3.5-3　与信ブロックフラグのモデル

“アクション”のモデルも作成する．与信ブロックフラグ変更処理のデシシジョンテーブルは表 3.5-5 となる．そこから作成されるモデルは図 3.5-4 である．引数による条件分岐を持ち，与信ブロックフラグを ON/OFF させる同期呼び出しを送信する．

表 3.5-5 与信ブロックフラグ変更デシジョンテーブル

項目	種別	(1)	(2)
引数	条件	TRUE	FALSE
与信ブロックフラグ	条件	Nothing	Nothing
与信ブロックフラグ	結果としての状態	ON	OFF

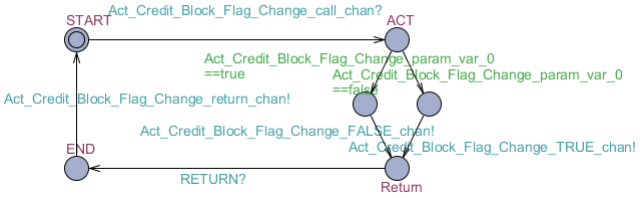


図 3.5-4　与信ブロック変更メソッドのモデル

5)モデル検査

上記で作成した仕様モデルをソースコードから制御フローを元に生成した検査モデルを結合させてモデル検査を行う．この作業には 2012 年度の本事業の成果である支援ツール Source2UPPAAL を使用する．

初期検査

最初に想定されない状態になるかどうかを検査する．この適用事例の仕様では，以下の 3 つの状態に必ずなるはずである．

・与信ブロックフラグ：ON かつ与信状態：OVER　かつ受注伝票登録許可状態：許可

・与信ブロックフラグ ON かつ与信状態 OVER　かつ受注伝票登録許可状態：拒否

・与信ブロックフラグ OFF かつ与信状態 UNDER　かつ受注伝票登録許可状態：許可

これ以外の状態になるならば，不具合があることになる．式の意味が“ソースコードモデルの終点にいる場合は常に上記 3 つの状態のいずれかになる”という検査式を作成して検査する．検査した結果は” Property is NOT satisfied.”となり想定されない状態になりうることが判明した．さらに想定されない状態になる場合と同じ UPPAAL の初期状態から始めても想定される状態になることはないかという検査をする．検査した結果は”

Property is NOT satisfied.”となり正常に終了する場合もあることが判明する.

制限を付加した再検査

想定される状態になる状態遷移と想定されない状態になる状態遷移を比較することによりソースコードのどの部分に成功と失敗の分岐点があるかわかる(図 3.5-5).

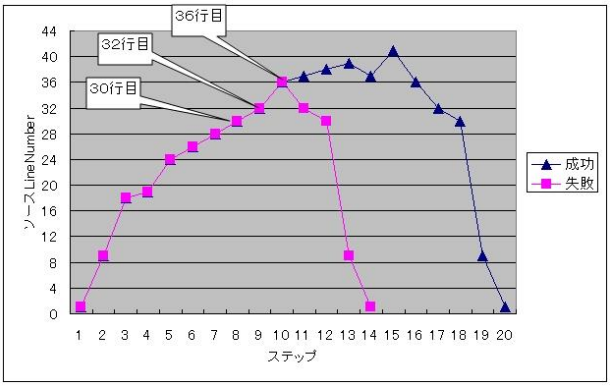


図 3.5-5　条件分岐で差異がでるポイント

しかし，これだけでは想定される状態になる状態遷移が分岐ポイント以降で想定されない状態になることはないのか，想定されない状態になる状態遷移が分岐ポイント以降で想定される状態になることはないのかの保証はできない.

分岐後のポイント(図 3.5-6①②)を通過したことを確認できるフラグを検査式に付加して再度検査を行う.

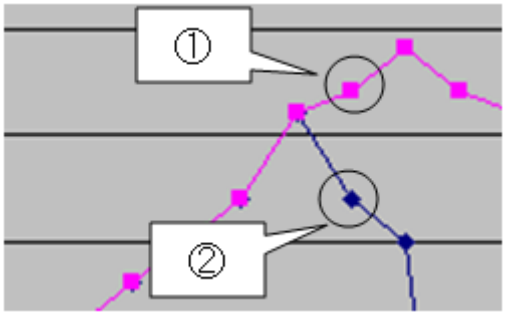


図 3.5-6　検査式指定するポイント

モデル上へ反映する方法は，UPPAAL 上で図 3.5-6 示した追加のポイントを通過するときに変数 checkFlg に true 設定することにより行う．図 3.5-7 は①の場合の例である.

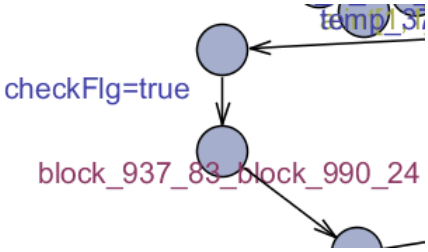


図 3.5-7　UPPAAL 上の指定ポイント

検査式は①通過後に「必ず 3 つの状態のいずれかになる」という式と，②通過後に「3 つの状態のいずれかにもならない」という式になる．

両式とも検査結果”Property is satisfied.”となり，分岐以降に最終的な状態が変わることはないことがわかる．したがって，調べるべき場所はソースコード 36 行目(図 3.5-8)の条件分岐であると確定できる．

```
30          if(rejectSalceOrderFlag==true ){
31              db.updateErrorTable("credit block on");
32          }else if(checkResult==false && blockFlag==false){
33              db.updateErrorTable("Credit line over");
34              db.changeBlockFlag(ON);
35              db.saveSalesOrder();
36          }else if(checkResult==true){
37              if(blockFlag==true){
38                  db.updateErrorTable("credit block off");
39                  db.changeBlockFlag(OFF);
40              }
41              db.saveSalesOrder();
42          }
```

図 3.5-8　ソースコードリスト

与信限度引き上げした直後の処理を行うと与信 UNDER かつ与信ブロックフラグ ON の状態が発生する(checkResult==false && blockFlag==true の状態)ため，if 文の処理を全てすり抜けてしまっていることが原因と判明する．

③　反例解析事例 2

1)適用事例概要

長方形エディタプログラムへ提案手法を適用した．これは芝浦工業大学のプログラミング演習で使われた課題である．具体的には学生に提示されたプログラムの仕様書とそれを基に作成された Java プログラムである．仕様書は 53 行の文章により構成されている．

プログラムの機能はつぎのとおりである．

- 機能制御：start()メソッド
- 長方形の作成：create()メソッド
- 長方形の移動：move()メソッド
- 長方形の削除：delete()メソッド
- 長方形の拡大・縮小：expand(),shrink()メソッド
- ２つの長方形の重なり部分から長方形の作成：intersect()メソッド

2)検査モデルの作成

モデル化すべき文章の特定を行う．前節で作成した文章を用いてシステムの振舞いが記述されている文章を抽出する．長方形エディタの仕様書から抽出した例を以下にあげる．

- プログラム（create）は幅、高さ、左上の位置（ｘ座標、ｙ座標）を設定して長方形を作成する。
- プログラム（move）は長方形 R をｘ方向の距離 x0，ｙ方向の距離 y0 だけ移動する。
- プログラム（expand）は長方形 R は幅の倍率 mx で幅を高さの倍率 my で高さをそれぞれ拡大する。
- プログラム（expand）は長方形 R は幅の倍率 mx で幅を高さの倍率 my で高さをそれぞれ拡大する。
- プログラム(delete)は長方形 R を削除する。
- プログラム(displayBoard)は長方形をボード上に表示する。

抽出した文章よりシステムの振舞いを起こす動作を抽出する．これをアクション(システムの動作)とし，アクションを起こすための条件とアクションを実行した後の結果(システムの振舞いの状態)を抽出する．

システムが取りうる状態を知るためデシジョンテーブルを作成する(表 3.5-6)．

デシジョンテーブルとは，プログラムの仕様を整理する際に，複数の判定条件の組み合わせと，それに対応する判定結果をまとめた表のことである．ここでは，仕様を構成するシステムの基本的な振舞い（これをアクションと呼ぶ）に対して，これらを実行した際の結果として，振舞いに関わるデータの変化を結果としてとらえる．結果の違いを生じさせる条件を仕様から抽出して，表を整理する．(振舞いの事前条件と事後条件を整理する)

表 3.5-6 デシジョンテーブル

種別	項目	1	2	3	4
アクション	長方形を作成する	実行する	実行する	実行する	実行する
アクション	長方形を移動する	実行しない	実行しない	実行しない	実行しない
アクション	長方形を縮小する	実行しない	実行しない	実行しない	実行しない
アクション	長方形を拡大する	実行しない	実行しない	実行しない	実行しない
アクション	長方形を削除する	実行しない	実行しない	実行しない	実行しない
アクション	長方形を全て削除する	実行しない	実行しない	実行しない	実行しない
条件	ボード	長方形の数0	長方形の数2以上10未満	長方形の数2以上10未満	長方形の数10
条件	追加する図形が点および線分ではない	満たす	満たす	満たす	Nothing
条件	追加・変化する図形と同じ長方形はない	満たす	満たす	満たす	Nothing
条件	ボードから長方形がはみ出していない	満たす	満たす	満たす	Nothing
結果	ボード	長方形の数1	長方形の数2以上10未満	長方形の数10	長方形の数10

表 3.5-6 より長方形の個数が永続的に変化する状態がわかる．長方形のはみ出し，重複については永続的に変化するものでない.特定した状態をモデル検査ツール UPPAAL のロケーションに置き換え，アクションをエッジに置き換える．ロケーション間を関係するアクションで結ぶことにより仕様モデルを作成できる．

表 3.5-6 よりこのシステムの「想定されない状態」を抽出する．今回の場合「重複した

長方形が存在する」「11 個以上の長方形が存在する」「長方形ははみ出していない」となる．「想定されない状態」については UPPAAL 上の変数にその状態を格納する．そのため検査式は以下のようになる．

A[]not(長方形重複判定変数==true)

A[]not(長方形の数>11)

A[]not(長方形はみ出し判定変数==false)

動作動詞のアクション(システムの動作)をソースコード上の該当部分(メソッドおよび API)に当てはめてモデル化対象として特定する．本事例では「作成する」「削除する」「移動する」「縮小する」「拡大する」「全て削除する」についてモデルを作成する．

ソースコードから制御フローを元に生成した検査モデルを作成する．この作業には Source2UPPAAL を使用する．アクション をソースコード上の該当するメソッドおよび API に当てはめてモデル化対象として特定する．

作成したアクションモデルとソースコードモデル上のアクションを呼び出す状態遷移のエッジと同期処理により紐づける．さらにアクションの結果生じる状態遷移がある場合，その状態遷移をアクションモデルから同期処理により呼び出す．本事例では仕様上の「長方形を作る」はソースコード上のメソッド create に紐付く．さらに「長方形を作成する」実行後は長方形の個数が増加するため，長方形の個数を表す仕様モデルの呼び出す．

3)モデル検査の実施

先にあげた 3 つの「想定されない状態」に決してならないという安全性の検査を行う．そのうちのひとつ「長方形ははみ出していない」の検査について反例が出力された．検査式は「機能制御プログラム(start()メソッド)の終点において長方形は,はみ出した状態(判定フラグ=false) になることは決してない」である．

```
A[] not(Act_isIn_return_val==false &&
voidstart.pst_While_1006_id__0__id__8 )
```

図 3.5-9　検査式

- Act_isIn_return_val
  ：長方形はみ出し判定フラグ はみ出しあり false
- voidstart.pst_While_1006_id__0__id__8
  ：start()メソッドの終点のロケーション

反例を解析して仕様モデルである長方形の個数の状態とアクション(システムの動作)を確認する．

- 長方形の数は１個
- コマンドの実行回数は２回
- 実行したコマンドは１回目　作成、２回目　移動

上記の状態は既知の状態として以降の検査で利用する．

つぎの検査では「機能制御プログラム(start()メソッド)の終点において長方形は，はみ出さない状態（判定フラグ=true）になることは決してない」を検査する．システムの振舞

いを再現するために検査式は先の検査で判明した既知の状態も追加する.

```
A[] not(Act_isIn_return_val==true &&
voidstart.pst_While_1006_id__0__id__8 &&
cmdCnt==2 && State_Board_Cnt==1 && ID==2)
```

図 3.5-10 検査式 2

- cmdCnt：コマンドの実行回数
- State_Board_Cnt：長方形の数
- ID：コマンド番号　1:create　2:move

この検査でも反例が出力され，長方形がはみ出す場合とはみ出さない場合の両方が存在することがわかる.

4)反例解析

UPPAAL のロケーションはソースコードのステートメントに紐付いているためソースコードの動きも反例を解析することにより判明する. 図 3.5-11 は上記の反例から Main メソッドから開始して「長方形がはみ出した」状態で処理が終了する過程をメソッド毎の状態遷移を一つのグラフを組み合せることにより表したものである. 反例は状態遷移のトレースである. 横軸は反例のトレースのレコード数、縦軸はロケーションの番号である.

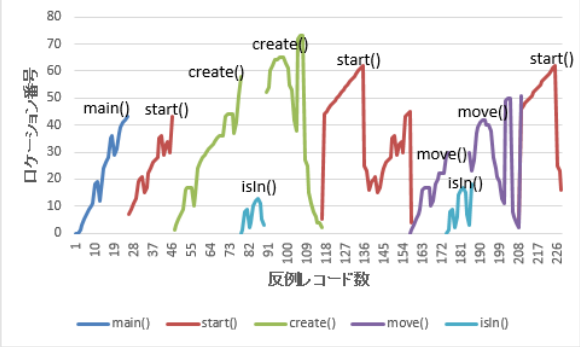


図 3.5-11 メソッド単位の状態遷移のグラフ

全てをつなぐと以下のシステム全体の状態遷移グラフになる(図 3.5-12).

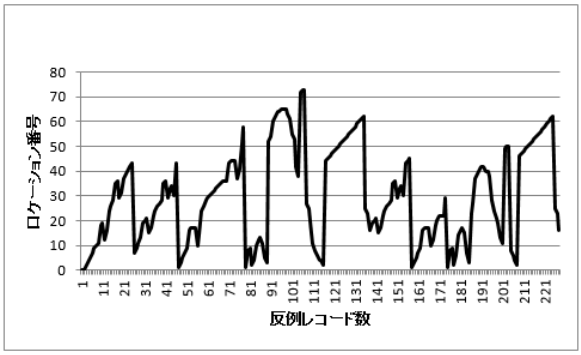


図 3.5-12　はみ出しが発生する場合の状態遷移のグラフ

同様に反例(はみ出しが発生しない)をグラフ化する.はみ出しが発生する場合(図 3.5-12)とこの発生しない場合のグラフを重ねると図 3.5-13 のグラフができる.途中から状態遷移が別れることが判る.

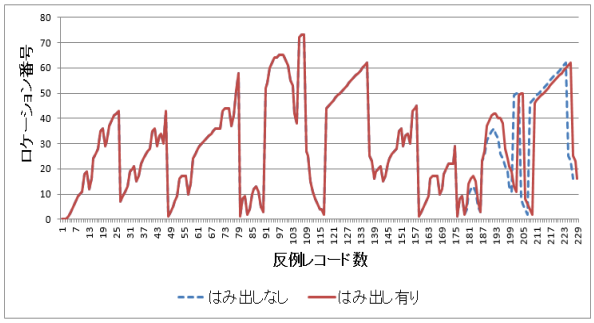


図 3.5-13 重ねたグラフ

分岐部分を拡大すると図 3.5-14 のグラフになる.

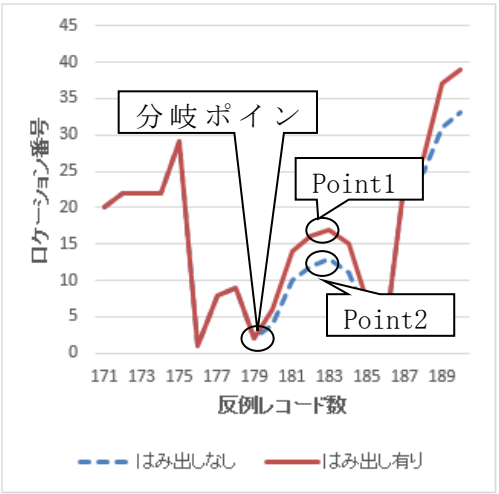


図 3.5-14 分岐ポイントのグラフ

長方形がはみ出す場合とはみ出さない場合の分岐ポイントがわかる．次にその分岐ポイント以降ではみ出しの判定が変わる場合があるかを検査する．検査式に分岐後のロケーション通過の有無の変数を付加する.図 3.5-14 にあるはみ出しが発生する場合のロケーションPoint1,はみ出しが発生しない場合のロケーションPoint2 に通過有無の変数を設ける．

Act_isIn_return_val==false ⇒trueに変換した場合の式は図3.5-15の検査式3になる．検査式 3 は検査式 1 を変換したものである．

```
A[] not(Act_isIn_return_val==true && voidstart.pst_While_1006_id__0__id__8
&& cmdCnt==2 && State_Board_Cnt==1 && ID==2 && checkPoint1=true)
```

図 3.5-15 検査式 3

- checkPoint1：図 3.5-14　Pint1 通過有無の変数

Act_isIn_return_val==true ⇒falseに変換した場合の式は図3.5-16の検査式4になる．検査式 4 は検査式 2 を変換したものである．

```
A[] not(Act_isIn_return_val==false &&
voidstart.pst_While_1006_id__0__id__8 && cmdCnt==2 && State_Board_Cnt==1 &&
ID==2 && checkPoint2=true)
```

図 3.5-16 検査式 4

- checkPoint2：図 3.5-14　Pint2 通過有無の変数

両検査とも反例が出力されないため，現在見つかっている分岐ポイントが最終的な分岐ポイントであることが判明する．この分岐ポイントに該当するソースコードの部分を調査すればよいということになる.ソースコードを調査すると条件判定式の図 3.5-17 の〇印の部分に等号が不足していることがわかった.

```
if(x >= 0 && y >= 0 && x + w <= board_w && y + h <= board_h){
```

図 3.5-17 条件判定式

3.5.3 実用化へ向けた課題と問題点

(1) 課題と問題点

「想定しない状態」を表す反例と「想定される状態」を表す反例をグラフ化して比較することにより複数のメソッドにまたがる場合でも，システムの振舞いを比較検討することができ，反例の理解を助けられることを示した．反例が出なくなるまで検査式を繰り返して精度を上げることができるため，不具合の原因となる分岐ポイントを見つける方法は有効であると考える．しかし，現状追加ポイントの設定を手作業で行っており，改善する必要がある.

(2) 将来の応用方法

検証支援ツールと連動したグラフ化支援と追加ポイントの設定支援を行うことで，ソースコードの検証結果をモデル検査ツールを直接操作せずに解析できることが期待される．ただし，可視化の方法には検討の余地がある．

4 考察

4.1 研究により判明した効果や問題点等

4.1.1 研究課題に対する解決方法

本研究では，開発現場におけるシステムのリリースの判断やマイグレーション時に，システムが「仕様」を満たしているかの確認作業を高効率・高品質で実施するために，モデル検査技術を用いたソースコード検証を，形式手法のスペシャリストでなくても、その恩恵が受けられるような検査方法を構築することを課題として取り組んできた．図 4-1 に 2.1.3 節で述べた目標に対する本研究での課題解決方法を示す．

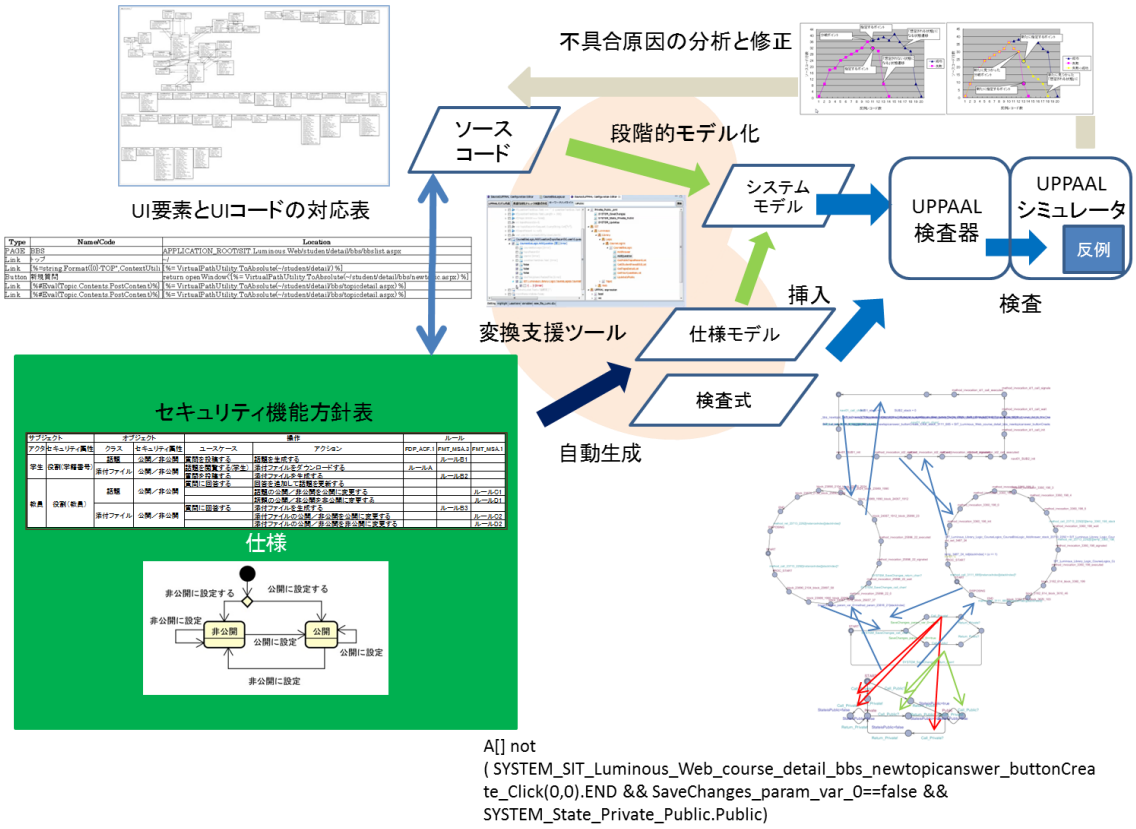


図 4-1 本研究での課題解決方法

1.2 節で述べた課題「検査モデルと検査式の定義方法」については，ユースケースの UI 要素とセキュリティ要件の対象となるエンティティクラスとその属性をソースコードと仕様の関連付けに用いることで，ソースコードの意味解析を行わずに，システムの開発者以外でも容易に対応付けを定義することができた．

「システムモデルへの変換方法」においても，2012 年度の成果である Source2UPPAAL の問題点を改善し，上述の対応表に基づき，容易に検査モデルを設定できた．さらに，設定した検査モデルから変換した UPPAAL モデルはエラーなく検査を行えた．LUMINOUS のソースコードはロジック 36,352 ステップであるが，検査シナリオに基づいて，不要なステートメントをモデル化しないことにより，問題なく検査を実施することができた．

「反例解析支援方法」については，反例のグラフ化により，段階的に問題個所を絞り込む方法を提案できた．絞り込みの操作が手動のため，作業負担軽減のため，自動化を行う必要がある．

ある程度の規模の仕様定義がないシステムから，システムの開発者ではない技術者が，システムの機能要件に対して，機能要件の振舞いと深くかかわるセキュリティ要件を検査することができた．以下に，問題点と今後の展望を記す．

4.1.2 既存システムの仕様定義についての考察

システムを操作しながらモデルを定義する作業においては,モデリングすることよりも,正確に必要な要素を拾えているかが問題である．本研究では，アクター・ユースケース・ユースケースの関連を用いて，トップの画面のメニュー項目に従い，ユースケースを列挙した．仕様としての品質を高めるために，網羅性と正確性を確認する作業の作業負荷を軽減する支援を考える必要がある．

4.1.3 ソースコード解析の問題点

仕様とソースコードの要素を対応付けのために，本研究では，ソースコードの静的解析により，対応付けの情報を取得する解析ツールを構築したが，3.1.3 節で述べた，実装プラットフォームの問題，開発者によるコーディング規約の不統一の問題，設計の欠如によるクラス構造の読み取りが困難になるという問題があった．

多様なプラットフォームや多様な実装スタイルに対する問題は，解析ツールの構築にコストをかけなければならず，検証ツールを構築する妨げとなる．開発プロジェクトにおけるソフトウェアアーキテクチャの設計，コーディング規約の統一，設計方針の明確化を行うことが，検証しやすいソースコード構築には不可欠である．

4.1.4 セキュリティ要件の検証についての考察

本研究で扱ったセキュリティ要件は，ユースケースの振舞いと密接にかかわる資産の機密性に関するものである．利用者の役割は，第一段階として，システムの資産へアクセスする機能の認可を定めることになる．こうしたシステムの利用者の属性による制御を行う場合，UI からの入力が正しくロジックに伝達されているかが問題となる．セッション管理等に不備があるために，想定外の値が機能のロジックに与えられることがあると，これはセキュリティ要件を満たさないケースとなる．しかし，この場合は，検査モデルにおいて，UI からの情報伝達もモデル化する必要がある．今回の検証では，ユースケースレベルの機能を認可した後に，特定の情報へのアクセスをその情報の属性によって制御しているケースをロジック部分のモデル化により検査することができた．UI コードはロジックとは異なる言語を用いているのみならず．複数の言語が用いられているため，このケースは扱っておらず．今後の課題である．

4.2 今後の課題と産業界への要望

4.2.1 今後の課題

形式手法のスペシャリストではない現場の開発者が，モデル検査の恩恵を受けられる方法を研究し，ユースケースにより定義した機能要件に対して，セキュリティ要件という非機能要件の１つをセキュリティ機能方針表により定義し，検査を実施するという方法を提案することができた．本方法の産業界における活用の場面としては、以下の場面が想定できる。

- システムのマイグレーション時に，本研究で示した仕様定義方法により，仕様の十分でないシステムの仕様を整理し，旧システムおよび新システムのソースコードを検証する.これにより,旧システムが満たす仕様を新システムが満たすことを保障できる.
- 最終的なリリース判断において，テストでは不十分であると思われる特定の仕様を満たしていることを低コストで確認する.

これらの過程において,仕様が定義されている場合でも,仕様定義が不十分な場合でも,副次的に，検査したいことの観点で仕様を再整理することで，仕様定義の教育が行え，手戻りの少ない開発に貢献することが期待される.

しかし，提案方法を実際に活用するためには，以下の課題を解決する必要がある.

(1) モデル検査とテストの切り分けの問題

必ずしもモデル検査を利用する必要がなく，実環境でのテストが必要な場合もあると考える．技術が低コストで適用可能な基準を明確にすることが，産業界へ普及を図る上で重要である.

(2) 仕様化観点の問題

システムにはセキュリティ要件だけでなく，ハードウェアや通信の条件，使用性，信頼性，効率性，保守性等の品質に関わる要求等，様々な要求がある，これらに関する要件は，どのように整理すれば,本提案のようにモデル検査を使って,その要件を検証できるのか,また，機能要件においてもテストでは発見しにくい問題に対して，どのように仕様を整理すれば，モデル検査を活用できるのかといった問題を整理する必要がある.

(3) 変換支援ツールの汎用性の問題

変換支援ツールそのものは実装技術の多様性から，提案ツールをそのまま各現場の開発に適用できるケースは少ない.そこで.抽象構文木の生成エンジンは既存のものを利用し，変換支援ツールへ組み込む等の作り込みを行う必要がある．対象とするソースコードも多言語を利用している場合には，それらの静的解析を自前で行う必要もある.

4.2.2 課題解決へのアプローチ

「モデル検査とテストの切り分けの問題」に対しては，モデル検査により検査できる仕様パターンの研究に基づき，機能要求，非機能要求の観点も取り入れて，技術が低コストで適用可能な基準を明確にすることを検討している.「仕様化観点の問題」については，再利用資産の開発と再利用資産の利用を体系的に行う開発スタイル，例えばプロダクトライン開発やモデル駆動開発をもとに仕様化の観点を整理することを検討する.「変換支援ツールの汎用性の問題」については，本ツールのフレームワーク化を検討する．個別の言語に依存する部分を埋め込むことにより，提案手法を支援するツールを開発したいと考える.

4.2.3 産業界への要望

こうした技術の普及を図るためには，提案手法を実際に試行する題材（システムまたは仕様定義とソースコード）の提供とその題材に適用可能な変換支援ツールを技術者の参加により共同して開発することへの投資を産業界または IPA へ要望する．提案技術が開発現場で活用できるためには，高品質な変換支援ツールの開発と技術者の教育が必須であると考える．

参考文献

[1] 中島 震, ソフトウェア工学の道具としての形式手法, コンピュータ ソフトウェア, NII Technical Report ISSN 1346-5597,2007.

[2] Aoki, Y., Matsuura, S., “Verifying Business Rules Using Model-Checking Techniques for Non-specialist in Model-Checking”, IEICE TRANSACTIONS on Information and Systems, Vol.E97-D, No.5, pp.1097-1108, 2014.

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

[4] 谷沢智史, 西村一彦, 青木善貴, 小形真平, 松浦佐江子,“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,” 19 回ソフトウェア工学の基礎ワークショップ, pp.241-242, 2012.

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

[6] 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

[7] 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.

[8] 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.

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

[11]野呂, 松浦, コモンクライテリアを用いたモデル駆動セキュリティ要求分析手法, ソフトウェアエンジニアリングシンポジウム (SES2013) ,2013

[12]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.

[13]青木善貴, 小形真平, 野呂惇, 松浦佐江子,“モデル検査技術を用いたセキュリティ要求の検証,” 第 20 回ソフトウェア工学の基礎ワークショップ, pp.209-214, 2013.

[14]Pathfinder,http://javapathfinder.sourceforge.net/, 2014.

[15]Markosian, L.Z.; Mansouri-Samani, M.; Mehlitz, P.C.; Pressburger, T., Program Model Checking Using Design-for-Verification: NASA Flight Software Case Study, 2007 IEEE Aerospace Conference, vol.1, no.9, pp.3-10, 2007.

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

[17]SPIN,http://spinroot.com/spin/whatispin.html/, 2015.

[18]NuSMV, http:// http://nusmv.fbk.eu/ ,2015.

[19]B. Dwyer, G.S.Avrunin, and J.C. Corbett:"Patterns in property specifications for finite-state verification,"Proceedings of the 1999 International Conference on Software Engineering(ICSE),pp.411--420,
[20].NET Compiler Platform Roslyn: http://msdn.microsoft.com/ja-jp/vstudio/roslyn.aspx
[21]ASP Parser, http://www.codeproject.com/Articles/ 6888/ASP-Parser