モデルテストによるセキュリティ分析・設計パターンの適用支援

小橋孝紀† 大久保隆夫‡ 海谷治彦†† 吉岡信和‡‡ 伊永祥太† 鷲崎弘宜† 深澤良彰†

†早稲田大学
169-8555 東京都新宿区大久保 3-4-1
kobashi@akane.waseda.jp

‡株式会社 富士通研究所
211-8588 川崎市中原区上小田中 4-1-1
okubo@jp.fujitsu.com

††信州大学
690-0826 長野市若里 4-17-1
kaiya@cs.shinshu-u.ac.jo

‡‡国立情報学研究所
101-8430 東京都千代田区一ツ橋 2-1-2
nobukazu@nii.ac.jp

あらまし 現在のソフトウェア開発では開発者が必ずしもセキュリティの専門家ではなく,システムに起こりうる脅威を確認,検討することができていない.本稿ではUMLモデルのシミュレーション環境を用いたモデルテストによる,セキュリティパターンの適用支援を提案する. 本手法を用いる事によりセキュリティに精通しない開発者であっても分析・設計段階でのシステムにおける脆弱性の把握が可能になり,セキュアな設計が可能になる.更に本稿では既存のセキュリティパターンに対し構造・振る舞いを定義し新たに加え用意しておく事で,パターン毎に異なっていた構造の抽象度や表現の仕方を統一し,形式的なパターンの適用を可能にした.

Model-Driven Security Analysis-Design Patterns Application Using Model Testing

Takanori Kobashi† Takao Okubo‡ Haruhiko Kaiya †† Nobukazu Yoshioka‡‡
Shota Inaga† Hironori Washizaki† Yoshiaki Hukazawa†

†Waseda University
Computer Science and Engineering Department
Tokyo Japan

‡Fujitsu Laboratories limited
Secure Computing Department
Kawasaki Japan

††Shinshu University
Computer Science Department
Nagano Pref Japan

‡‡National Institute of Informatics
Architecture Research Division
Tokyo Japan

Abstract Current software developer is not necessarily an expert in security. So check the possible threat to the system, have not been able to examine. In this paper, we propose Model-Driven security analysis-design patterns application using model testing. By using this method, secure design of analysis and design stages is possible even developers who are not familiar with the security. In addition, to define structure and behavior for existing security patterns uniformity, developers can application patterns formally.

1 はじめに

近年,オープンなネットワークを利用したサービスが社会に浸透し,ビジネスとして利用されるようになってきている.また,それに伴い,セキュリティ対策を施したソフトウェアを構築,提供することは重要であり,現状のネット社会ではある種義務となってきている.

セキュリティの関心事はソフトウェア開発の要求から

設計,実装,テスト,運用の全てのフェーズで扱えるようにしなければならない[1].

しかしながら,現在の開発では開発者が必ずしもセキュリティの専門家ではなく,開発の早い段階でシステムにおけるセキュリティ項目について十分に考慮されていない.特にセキュリティの設計は,機能レベルの要求の実現とは異なる.その為,設計のノウハウが蓄積されておらず,特に,セキュリティ知識に乏しい開発者では,標準的な設計手法やベストプラクティスの不足により,要求仕様に従った適切な設計の実現は困難である.更に設計仕様が本当にセキュリティ要求仕様を満たしているかの検証は十分におこなわれていない[2].

そこで本稿では,UML モデルのシミュレーション環境を用いたモデルテストによる,セキュリティパターンの適用支援を提案する.本手法を用いることによって,まずパターン適用前に要求段階で特定した脅威によって用いられ被害を引き起こされる可能性のある設計上の脆弱性の有無を検出する.次にパターン適用によりセキュリティ対策を施すことで設計上の脆弱性を解消し,セキュアな設計を可能とする.

2　背景と関連研究

ここではセキュリティ設計における代表的な設計手法とその問題点について述べる.

2.1　UML のセキュリティ拡張

近年では,設計の為のモデリング言語として UML が標準になってきている.セキュリティをモデル化する為の拡張として,UMLsec[3]や SecureUML[4]が提案されている.UMLsec は UML の拡張記法を利用し,セキュリティ要素を UML 拡張であるプロファイルに反映させることにより,モデルベース開発を行う手法である.UMLsec ではクラス図やアクティビティ図の要素に対して,セキュリティ的な特性や前提条件などを,ステレオタイプやタグ付き値として記述する.

しかしセキュリティに乏しい開発者が,新規に UMLsec のモデルを記述する事は容易ではなく,導入には教育期間やコストが必要である.

2.2　セキュリティパターン

現在,セキュリティ要求仕様を満たすソフトウェア設計の具体的な設計手法の一つにセキュリティパターンの利用がある.セキュリティパターンとは,セキュリティに関する特定の状況下で幾らか抽象化された問題およびその解決法のことであり,ソフトウェア開発者はセキュリティパターンを用いることでセキュリティの専門家と同様のセキュリティレベルの開発や再利用を効率的に行うことができる.例えば[5]では,46 個のセキュリティパターンが示されている.

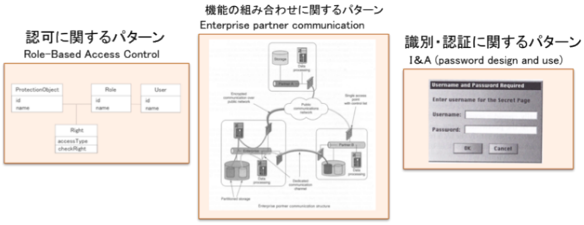

図1　セキュリティパターンの構造の例

セキュリティパターンの記述項目には,名前などのほかに,状況(Context),問題(Problem),解法(Solution),解法の構造(Structure),結果(Consequences),関連パターン(See Also)などが含まれる.パターンの記述は,複数のシステムに再利用可能であるように考慮されている.

セキュリティパターンの構造の例を図1に示す.セキュリティパターンも構造の表記に UML 等を使って表現しているパターンが多くある.しかし,図 1 にあるように現状では,これらのモデル表記にはモデルの抽象度や構造・振る舞いの表し方に統一性がない.

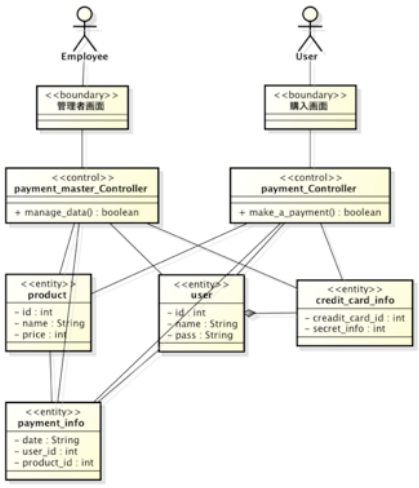

図 2　セキュリティを考慮していない分析クラス図

例として図2にWeb上での購買を実現するためのソフトウェアの分析モデルを示す.このモデルは特にセキュリティを考慮していない.現状ではこれらのモデル

に対し特定済みの脅威に対しセキュリティ対策として図1のようなパターンを適用しようとしても,パターン毎の統一性の欠如により,一貫した適切かつ効率的な適用は困難である.更にパターンを適用した際,パターンを適切に適用できたか,更に脅威によって用いられ被害を引き起こされる可能性のある脆弱性が解消できたか,これらを検証することは十分おこなわれていない.

2.3 セキュリティ要求パターン

ソフトウエアシステムにおける脅威とその対策の特定をする手法として Security requirements patterns（以下,SRP）[6] がある. セキュリティパターンには要求分析時に用いる要求パターンもあれば,設計時に適用する設計パターンもある.前者については脅威の特定にそのまま利用可能である.後者については2.2節で述べているような問題があるため,本稿では制限を与えて具体化した結果を用いる.

SRP ではパターンの構造に The Misuse cases with Asset and Security Goal（以下,MASG）を使うことによって,要求段階でシステムにおけるアセット,脅威,対策の関係性を表現している. このモデルはミスユースケース[7]の拡張であり,システムにおける攻撃者,脅威,対策を通常のユースケースに加えモデル化したものである.

MASG モデルは通常のミスユースケースに加え,以下の3つの要素を持っている.

- Data assets：守られるべきアセット
- Usecase assets：アセットに関連した機能
- Security goals：アセットを保護する為に満たされるべき目標

アセットの特定は脅威の特定を可能にし,Security goals の特定はシステムにおいて満たされるべきセキュアなユースケースを明確にする.MASG モデルもこの分析プロセスの流れを組み込んでいる.最初にシステムにおけるアセットを特定し,Security goals を定義する.次にアセットを脅かす脅威を特定し,その脅威を低減する対策を定義する.最後に Security goals を満たすセキュアなユースケースを確定する.

図3に Web 上での購買を例に MASG モデルを使った SRP の構造を示す.アセットである[make a payment：購買処理]や[manage data：データ管理処理]に対して[Unintented use：予期しない利用],[Spoofing：なりすまし],[Unauthorized use：権限無き利用]の脅威がある事が分かり,その対策として[I & A：認証と識別],[Prvent CSRF：CSRF 対策],[RBAC：Role-Based-Access-Control]が有効である事がモデル化されている.

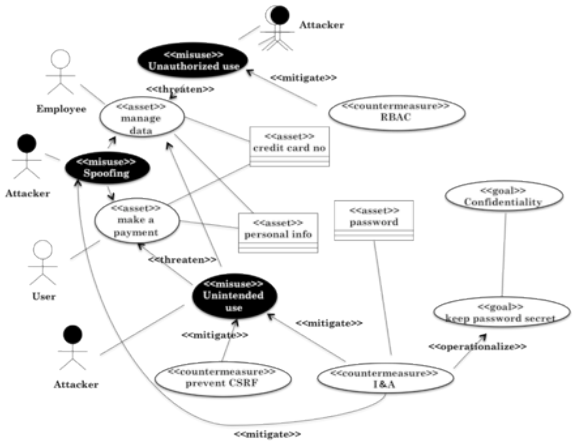

図 3 Web の購買における MASG モデル

3 モデルテストによるセキュリティ分析・設計パターンの適用

本章では,本手法について述べる. 先の背景と関連研究を踏まえ本稿で扱う Research Questions（研究課題）を以下に示す.

RQ1 セキュリティパターンの一貫した適用支援を実現できるか.

RQ2 モデル上でのセキュリティパターンの適切な適用を検証できるか.

RQ3 セキュリティパターンの適用前や適用後におけるモデル上で,特定済みの脅威に対する脆弱性の有無を正確に検出できるか.

本手法では UML モデルのシミュレーション環境を用いたモデルテストによって RQ1,RQ2 ,RQ3 に対して解放策を示す.以下にテスト実行の際に用いる考え方であるテスト駆動開発について説明する.

3.1 テスト駆動開発(TDD)

テスト駆動開発 (TDD：Test Driven Development) とは,プログラム開発手法の一種で,プログラムに必要な各機能について,最初にテストを書き(テストファースト),そのテストが動作する必要最低限な実装をとりあえず行った後, コードを洗練させる,という短い工程を繰り返すスタイルである[8].

本研究では,UML モデルのシミュレーション環境 USE[9]を使用して,モデルのテスト実行を行う.USE はドイツのブレーメン大学で Java 開発されたオープンソフトで,UML,OCL の記述とシミュレーションによる OCL の評価を行うことができる.オブジェクト制約言語 OCL(Object Constraint Language)[10]はモデル記述のための形式言語であり,モデルが整合的であるためにモデル要素や関係の間で満たすべき命題を式として定義できる.

本稿では,セキュリティを考慮していない分析・設計モデルを USE にてテスト実行をし,最初に要求段階で特定した脅威によって用いられ被害を引き起こされる可能性のある設計上の脆弱性を検出させた上で(テストファースト),その後セキュリティパターンを適用したモデルをテスト実行し,脆弱性が解消された事を確認する.

3.2 本手法の説明

まず本手法の全体像を図 4 に示す.本手法の流れは以下のようである.

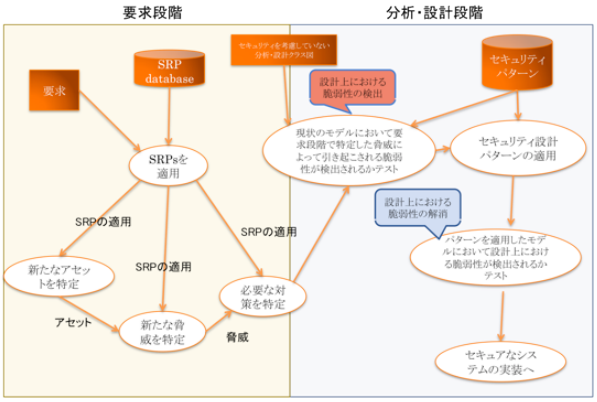

図4 本手法の全体像

1. 新たな要求に対し,SRP を適用し,要求段階において開発中のソフトウェアにどういったアセット(資産),脅威,対策があるかを特定しその関連性を図式化する.
2. セキュリティを考慮していない分析レベルクラス図を入力とし USE にてモデルテストを行い,パターン適用前に要求段階で特定した脅威によってもたらされる脆弱性が検出される,つまりセキュリティ要求を満たしていない事を確認する.
3. セキュリティパターンを適用する.具体的には入力として与えられた分析レベルクラス図に対しセキュリティ機能を持つ構造・振る舞いを加える.パターンの扱いに関しては後述する.パターン適用の際は OCL 記述で制約したパターンの構造を満たすようにする.
4. パターンの適用によってパターン適用前に検出されていた脆弱性が解消された,つまりセキュリティ要求を満たしたことと,パターンの構造を満たしていることを USE で確認する.

このように USE でパターン適用前ではセキュリティ要求を表す OCL 式が満たされないとなっていること確認した上で,パターンの解決を適用することによってセキュリティ要求を表す OCL 式が満たすなることでセキュリティ要求仕様を満たしているかを検証する.これは RQ3 の解決策である.

3.3 新たな定義を加えたセキュリティパターン

本手法ではまず,既存のセキュリティパターン毎に,パターンにおけるセキュリティ要求と統一的な表記で表された構造・振る舞いを加えておく.パターンの構造とセキュリティ要求は OCL 記述で表現する.これらによってシステムにおける脅威(問題)と対策(解決)の構造と振る舞いの把握と複数のパターンの適用が容易になる.また OCL 記述でパターンに制約を加えることによってパターンを厳密に定義しパターン適用時にセキュリティ要求を満たし,かつパターンが適切に適用できたかを検証することが可能となる.これらは RQ1,RQ2 の解決策になる.

作成方法としては,まず既存のセキュリティパターンの問題,構造,解決を参照し,そのパターンがどのような脅威に対しどのような構造で解決をするのかを把握する.次にロバストネス分析を行い,分析・設計レベルの構造に統一させた上で振る舞いを定義する.

以下に例として I&A(password design and use) と Prevent CSRF(Validation of the secret string in the hidden parameter)における OCL 記述で表現したセキュリティ要求とパターン適用前の構造・振る舞い(問題),パターン適用後の構造・振る舞い(解決)を示す.

3.3.1 I&A(password design and use)

図 5 は「アセットにアクセスできるのであるならばユーザはシステムの正規ユーザである」という I&A におけるセキュリティ要求が OCL で記述されている.

```
context subject_controller
  inv  Security Requirement  :
   self.access_asset_data = true  implies
      self.subject_UI.actor.registerd_user = true
```

図 5 I&A におけるセキュリティ要求

次にパターン適用前と後の構造を図 6,パターンの構造が満たすべき OCL 記述を図 7 に示す.

パターン適用前はシステムの非正規ユーザ(registered user = false)でもアセットにアクセス可能である異常系が存在しており,図 5 のセキュリティ要求を満たしていないが,適用後にセキュリティ機能を持つ構造・振る舞いを加ええ,図 7 に示すような OCL 記述の制約を満たすことによってシステムの正規ユーザでしかアセットにアクセスできないようになる.

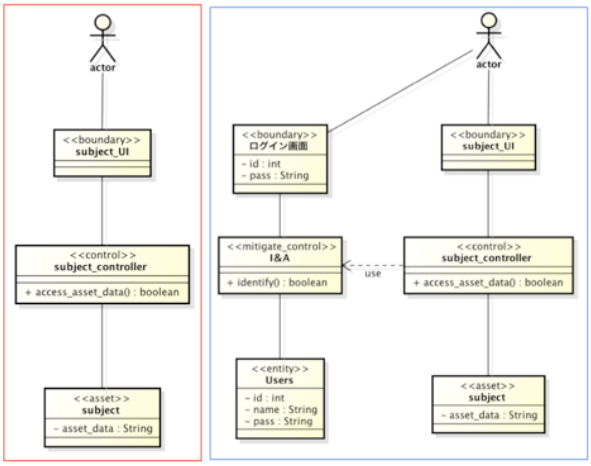

図 6 パターン適用前と後の構造(問題と解決)

```
context subject_controller
  inv check_id_and_pass:
    if self.IandA.users->exists(p | p.id = self.IandA.login_UI.id and
                                     p.pass = self.IandA.login_UI.pass
                                  )
                               and  self.subject_UI.actor.registerd_user = true
  then
                self.asscess_asset_data = true
  else
                self.asscess_asset_data = false
  endif
```

図 7 パターンの構造が満たすべき OCL 記述

振る舞いを加えることによりシステムの非正規ユーザの場合 I&A の機能によってエラーを出力しアセットにアクセスできないようになる.つまりパターン適用後は図 5 のセキュリティ要求を満たしている.

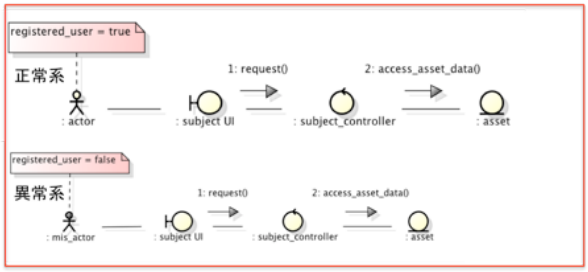

図 8 パターン適用前の振る舞い(問題)

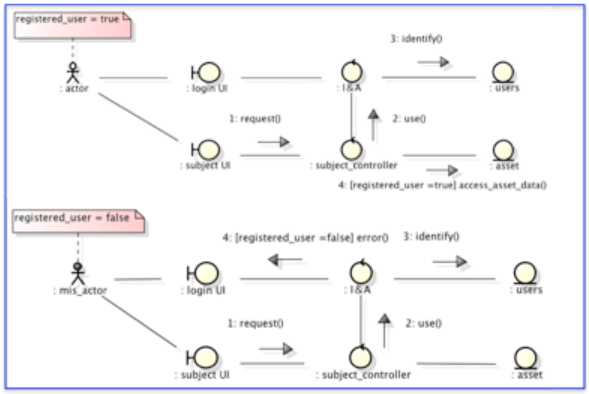

図 9 パターン適用後の振る舞い(解決)

3.3.2 Prevent CSRF(Validation of the secret string in the hidden parameter)

Prevent CSRF も I&A と同様に OCL 記述で表現したセキュリティ要求とパターン適用前の構造・振る舞い(問題),パターン適用後の構造・振る舞い(解決)を持つ.

図 10 は「アセットにアクセスできるのであるならば受け取っているリクエストは安全なリクエストである」というPrevent CSRF におけるセキュリティ要求が OCL で記述してある.

```
Context subject_controller
  inv Security Requirement :
      self.access_asset_data = true implies
            self.get_safe_request = true
```

図 10 Prevent CSRF におけるセキュリティ要求

次にパターン適用前と後の構造を図 11,パターンの構造が満たすべき OCL 記述を図 12 に示す. パターン適用前は脅威の存在しうる不正なリクエスト(get_safe_request = false)でもアセットにアクセス可

能である異常系が存在しており,図 10 のセキュリティ要求を満たしていないが,適用後にセセキュリティ機能を持つ構造・振る舞いを加え,図 12 に示すような OCL 記述の制約を満たすことによって不正なリクエストを受け取った場合,アセットにはアクセスできなくなる.

図 12 の OCL 記述は「実行画面で受け取ったトークンが実行直前画面で埋め込まれたトークと一致するのであれば安全なリクエストと判断されアセットへのアクセスが可能であり,それ以外であるならば不正なリクエストと判断されアセットにアクセス不可能である」という制約が記述されている.これは図 10 のセキュリティ要求を満たしている.

最後にパターン適用前の振る舞い(問題)と適用後の振る舞い(解決)を図 13 と図 14 に示す.

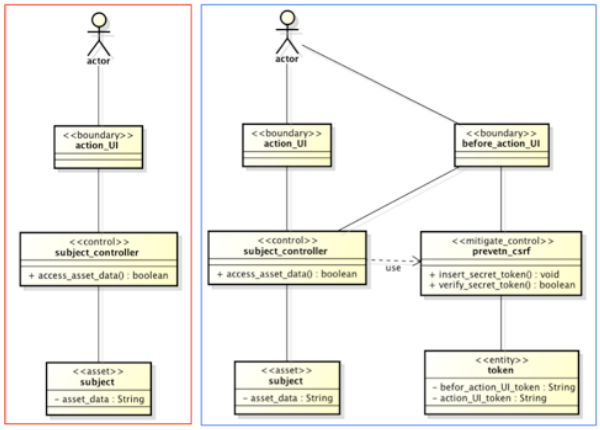

図 11 パターン適用前と後の構造(問題と解決)

```
context subject_controller
 inv verify_secret_token :
  if  self.prevent_CSRF.token->exists(p | p.before_action_UI_token = p.action_UI_token)
                                        and self.get_safe_request = true  then
     self.get_safe_request = true
  else
     self.get_safe_request = false
  endif
```

図 12 パターンの構造が満たすべき OCL 記述

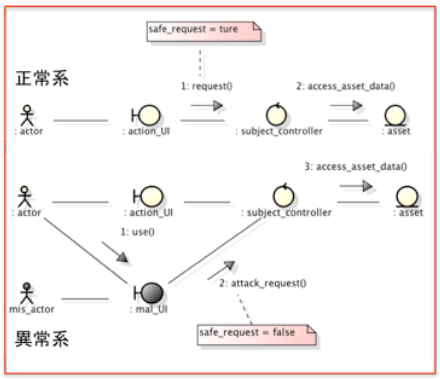

図 13 パターン適用前の振る舞い(問題)

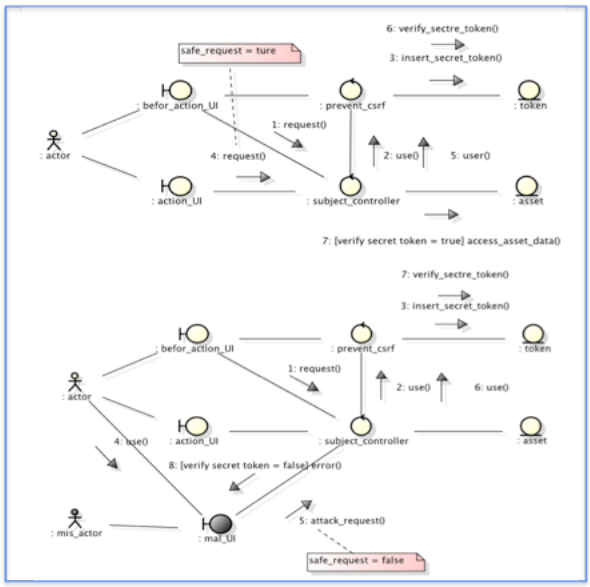

図 14 パターン適用後の振る舞い(解決)

4 具体例を用いた本手法の説明

本章では Web 上での購買において本手法の流れに沿ってセキュアな設計を実現する.具体的には「Web 上での購買」という要求に対して SRP を参照することによってシステムにおける資産,脅威,対策をモデル化し特定する.次に要求段階で特定した脅威によってもたらされる脆弱性が実際に存在するのかを USE にてモデルテストを行い脆弱性が存在する事を確認した上でセキュリティパターンの解決を適用する.最後に対象となる脆弱性がパターン適用により解決されているかについて再度モデルテストを行い脆弱性が解消された事を確認する.

具体例における SRP は図 3 のモデルを構造として使用する.つまり「Web 上での商取引」という要求に対してこのシステムにはアセットである(make a payment:購買処理)や(manage data:データ管理処理)に対して(Unintented use:予期しない利用),(Spoofing:なりすまし),(Unauthorized use:権限無き利用)の脅威があり,その対策として(I&A:認証と識別),(Prvent CSRF:CSRF 対策),(RBAC:Role-Based-Access-Control)が有効である,というケースを設定する. 本手法を図 2 に設定したセキュリティを考慮していない分析・設計クラス図に対し適用していく.

今回は図 3 にある[make a payment:購買処理]を例にとる.購買処理には(Unintented use:予期しない利用),(Spoofing:なりすまし)の脅威が存在し,その対策として(I&A:認証と識別),(Prvent CSRF:CSRF 対策)

がある.このことから購買処理において満たされるべきセキュリティ要求を図 15 に示す.アセットにアクセスできるのはシステムの正規ユーザ,かつ安全なリクエストを受け取る必要がある.

```
context payment_controller
 inv Security Requirement :
   self.make_a_payment = true implies
      self.payment_UI.user.registerd_user = true and self.get_safe_request = true
```

図 15 購買処理におけるセキュリティ要求

次に図 2 において図 15 のセキュリティ要求が満たされているかどうか USE にてモデルシミュレーションを行った結果を図 16 に示す.セキュリティを考慮していないモデルはオブジェクト制約が無く,システムの正規ユーザまたは不正なリクエストを受け取るような異常系でも[make_a_payment = true],つまり,アセットにアクセス可能となっている.つまり図 2 のモデルはセキュリティ要求を満たさず,モデルシミュレーションの結果 OCL 記述の評価が false となっている.

次にセキュリティパターンの解決を参照し, セキュリティ機能を持つ構造・振る舞いを加える.図 17 に全通りのシミュレーション結果を示す.システムの非正規ユーザ,または不正なリクエストを受け取る場合ではパターン適用前はもちろん I&A のみ適用時や Prevent CSRF のみ適用の場合でも図 15 のセキュリティ要求を満たしていない事が分かる.I&A と Prevent CSRF の構造が満たすべき OCL 記述を両方満たす事によってセキュリティ要素を満たすことが可能となる.

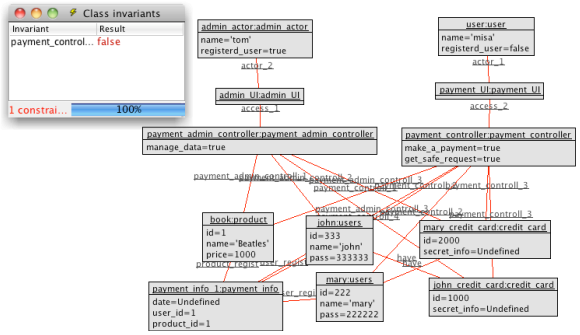

図 16 パターン適用前のモデルシミュレーション結果

Context	パターン適用前	I&Aのみ適用	Prevent CSRFのみ適用	両方適用
システムの正規ユーザ、かつ安全なリクエスト	セキュリティ要求:true	セキュリティ要求:true	セキュリティ要求:true	セキュリティ要求:true
システムの非正規ユーザ、または不正なリクエスト	セキュリティ要求:false	セキュリティ要求:false	セキュリティ要求:false	セキュリティ要求:true

図 17 モデルシミュレーションの結果

このようにシミュレーションを行うことによって要求段階で特定した脅威によって用いられ被害を引き起こされる可能性のある設計上の脆弱性がパターンの構造が満たすべき OCL 記述を満たすことによってセキュリティ要求を満たし,セキュアな設計になった事を確認,検証する事が可能とした.図 18 に購買処理に対してパターンを適用し,最終的に得られたクラス図を示す.同様な手法で[manage data:データ管理処理]にもパターンを適用する.

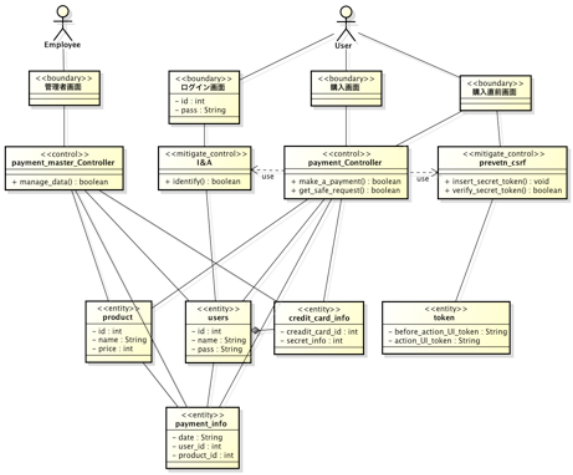

図 18 購買処理に対しパターンを適用したクラス図

5 制限

本章では本手法の適用範囲,制限を述べる.本手法では要求段階で特定した脅威・対策を基にモデルシミュレーションを行う為,要求段階で特定できなかった脅威によって用いられ被害を引き起こされる可能性のある設計上の脆弱性の検出,その対策は範囲外である.

6 評価

本章では本手法をホテル予約システムに対して適用することで,同等の結果が得られるかを比較・検証した.このシステムは NII/TopSE プロジェクト「アスペクト指向講座」[11]で扱われたシステムである.SRP を参照し予約処理に対する脅威を(Unintented use:予期しない利用),(Spoofing:なりすまし)とし,その対策として(I&A:認証と識別),(Prvent CSRF:CSRF 対策)としてケースを想定した.つまり,このシステムには予約画面から宿泊予定日を入力するユーザがシステムの正規ユーザであるか,また予約画面から受け取るリクエストが安全なリクエストであるかを判断する機能がない.図 19 に設定したセキュリティを考慮していない分析・設計クラス図を示す. また予約処理において満たされるべきセ

キュリティ要求を図 20 に示す.

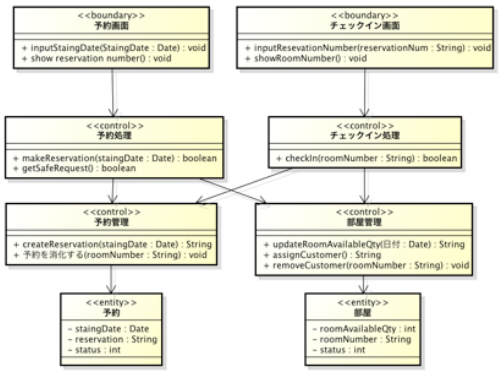

図 19 セキュリティを考慮していない分析クラス図

```
Context reservation_controller
  inv Security Requirement :
    self.makeReservation = true implies
        self.reservation_UI.user.registerd_user = true and self.get_safe_request = true
```

図 20 予約処理におけるセキュリティ要求

次にモデルシミュレーションによってセキュリティ要求が満たされているかどうか検証し,パターン適用によって脆弱性が解消される事を確認した.

購買処理と同様に本手法でホテル予約処理に対してもパターン適用しモデルシミュレーションによりセキュリティ要求を満たしセキュアな設計の実現が可能となった.図 21 に予約処理に対し,パターン適用により最終的に得られたクラス図を示す.

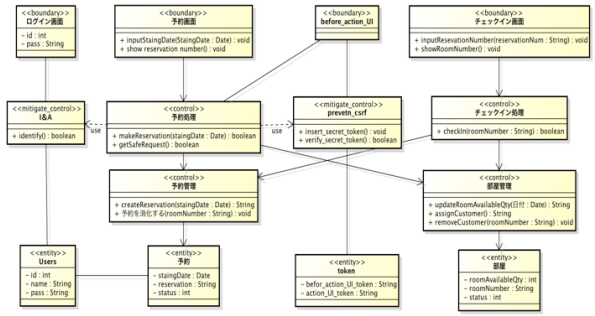

図 21 予約処理に対しパターンを適用したクラス図

このようにホテル予約システムにも本手法を適用し,あらたな定義を加えたセキュリティパターンを用いモデルテストを行うことによって 3 章で説明した RQ が解決されたことを確認した.

7 おわりに

本稿ではUMLモデルのシミュレーション環境を用いたモデルテストによる,セキュリティパターンの適用支援を提案した. システムにおける設計上の脆弱性というのは,漠然と考えていても考えつくものではないため,本稿では既存の問題とその対策が記述されているセキュリティパターンを用い,モデルのシミュレーション実行という形でテストを行った.更に既存のセキュリティパターンにセキュリティ要求,統一的な構造・振る舞いを新たに定義することで,厳密にパターンを定義し,システムにおける脆弱性(問題)と対策(解決)の把握と複数のパターンの適用を容易にした.

今後の課題としては XMI 形式のファイルからの自動変換によるパターンの自動がある. USE では現在,XML 形式のファイルに対応していない為,手動でモデル記述を行っているが,自動変換が可能となることにより実用の幅が広がると思われる.

参考文献

[1] N.Yoshioka, H.Washizaki, K.Maruyama “A Survey on Security Patterns”, Progress in Informatics, No.5, pp. 35-47. 2008

[2]吉岡信和 大久保隆夫 宗藤誠治 セキュリティソフトウエア工学の研究動向 コンピュータソフトウエア No.5 pp. 78-83 (2011)

[3]Jan Jurijens “Secure Systems Development with UML” 2004

[4] Torsten Lodderstedt David A. Basin Jürgen Doser “SecureUML: A UML-Based Modeling Language for Model-Driven Security” 2002

[5]M.Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, Peter Sommerlad. “SECURITY PATTERNS”. Wikey. 2006

[6]T.Okubo, H.Kaiya, N.Yoshikawa Effective Security Impact Analysis with Patters for SoftwareEnhancement IJSSE 3(1): 37-61 (2012)

[7] Guttorm Sindre and Andreas L. Opdahl. Eliciting security requirements by misuse cases. In TOOLS (37), pages 120–131. IEEE Computer Society, 2000.

[8]ケント・ベック. “テスト駆動開発入門”. ピアソン・エデュケーション. 2003

[9]USEhttp://www.db.informatik.uni-bremen.de/projects/USE/index.html

[10] ヨシュヴァルメル,アーネク・クレッペ. “UML/MDA のためのオブジェク ト制約言語 OCL”. 星雲社. 2004 ww.iwsec.org/css/2012/

[11] NII/TopSE プロジェクト「アスペクト指向講座」http://topse.jp