パターン間の関係を考慮した セキュリティパターン適用支援

城間祐輝[†] 久保淳人[††] 吉岡信和[††] 鷲崎弘宜[†] 深澤良彰[†]

セキュリティ専門家のノウハウを活用する方法としてセキュリティパターンがある．セキュリティパターンとは頻出するセキュリティ課題に対する典型的な解決法である．セキュリティパターン間には依存関係が存在することが多くあり，ソフトウェア開発者がパターン間の依存関係を考慮しないセキュリティパターン適用を行った場合，セキュリティパターンを誤って適用する可能性があり，その結果ソフトウェア全体のセキュリティが損なわれる．しかしながら，依存関係関係を考慮したパターン適用支援手法は現在提案されていない．本稿では，セキュリティパターンの適用をモデル駆動型開発における変換規則として定義し，自動適用する手法を提案する．変換規則としてのパターンは事前条件及び適用箇所に関する情報を含むため，誤った適用および誤った箇所への適用を防ぐことが可能である．さらに，ATL を用いて変換規則を記述することでセキュリティパターンの自動適用が可能になる．

Application Support for Security Patterns Considering Relationships among Patterns

Yuki Shiroma[†] Atsuto Kubo[††]
Nobukazu Yoshioka[††] Hironori Washizaki[†] Yoshiaki Fukazawa[†]

As open-software services through the Internet are spreading, importance of security is increasing. A security pattern is one of the techniques that developers utilize experts' knowledge. Security patterns contain typical solution about security problems. But There is possibility that developers may apply security patterns in inappropriate ways. However, application techniques of security patterns considering relationships among patterns have not been proposed yet. In this paper, we propose the automated application technique for security patterns under model driven architecture and define applications of security patterns as transformation rules under model driven architecture. Security patterns prevent application mistakes and application to wrong model elements. Moreover, security experts describe transformation rules using ATLAS Transformation Language. So developers can do automated application of security patterns

1. はじめに

インターネットを介したオープンなサービスの拡大に伴い，ソフトウェアセキュリティの重要性が高まっている．

セキュリティに関する問題は，要求，分析，設計，実装のいずれの開発工程においても発生する可能性があるため，開発の上流工程から一貫してセキュリティを作り込むことが欠かせないが，これは容易ではない．また，セキュリティの実現にあたっては，効率性などの他の品質特性とのトレードオフを考慮する必要があり，その適切な判断には多くの開発経験が求められる．

現在，セキュリティに関するノウハウがセキュリティパターン[1]として蓄積されつつある．セキュリティパターン(以下，SP)とは頻出するセキュリティ課題に対する典型的な解決法である．SP を用いることで，ソフトウェア開発者がセキュリティ専門家の知識を活用できる．SP はソフトウェア開発において，セキュリティの作り込みの指針となる．

また，SP 間に依存関係が存在することが多く，ある SP を適用した箇所に下流工程で他の SP を重ねて適用することが多い．このため，ソフトウェア開発者がパターン間の依存関係を考慮しない SP 適用を行った場合，誤って適用する可能性があり，その結果ソフトウェア全体のセキュリティが損なわれる．しかしながら，パターン間の依存関係を考慮した SP 適用支援手法は提案されていない．

そこで本稿では，パターン間の依存関係を考慮した SP の適用支援手法を提案する． 提案手法におけるパターン適用はモデル変換により実現される．提案手法では，パターン適用の際に適用の痕跡をモデル中に残し，以降に適用する SP はその痕跡を目印にして適用される．提案手法を用いることで，依存関係がある SP について誤りのない連続適用が可能になる．

† 早稲田大学理工学術院， Faculty of Science and Engineering Waseda University.
†† 国立情報学研究所， National Institute of Informatics.

2. モデル駆動型開発とセキュリティパターン

本章では，最初に，提案手法で用いるモデル駆動型開発および SP について述べ，その上で，SP 間の依存関係を考慮した適用における問題について議論する．

2.1 モデル駆動型開発

モデル駆動型開発とは，モデルを中心にソフトウェア開発を進めていく開発体系である[7]．開発者は，Unified Modeling Language(UML)などのモデル記述言語を用いて記述された抽象的なモデルを，より具体的なモデルへ変換する．変換は通常，自動化のためのツールを使って行われる．具体的なモデルへの変換を繰り返すことで開発者は自動的または半自動的にソースコードを得ることができる．

また，モデル変換方法は，モデルに対し付加的なモデルを指定し併合するモデル・マージ方式や，モデル内に残した痕跡をもとにモデル変換を行うマーキング方式など様々である．

2.2 セキュリティパターン

セキュリティとは，保護資産に対する攻撃への対策に関する概念のことである．セキュリティには大きく分けると可用性, 完全性, 機密性の 3 種類が存在する．可用性とは，システムが適切に稼働し，許可されたユーザに対してサービスが拒否されないことを保証することである．完全性とは，データや機能の正確性が維持されることである．機密性とは，承認されていない個人に個人情報または機密情報が開示されないことである．

SP とは, 頻出するセキュリティに関する問題に対する典型的な解決法である．開発における SP 適用のメリットは，ソフトウェア開発者がセキュリティ専門家の知識を活用できることである．セキュリティを満たすソフトウェア設計を最初から考えるのではなく，蓄積されている SP を開発時のモデルに適用することで実証済みの問題に対する解決法を活用できる. SP はソフトウェア開発において，可用性，完全性，機密性の作り込みの指針となる．SP の例を図 1 に示す．図 1 で示すように，SP の記述方式には，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はパターン間に依存関係を持つことが指摘されていて[6]，組み合わせて適用する際には SP 間の依存関係を意識した開発プロセス全体に

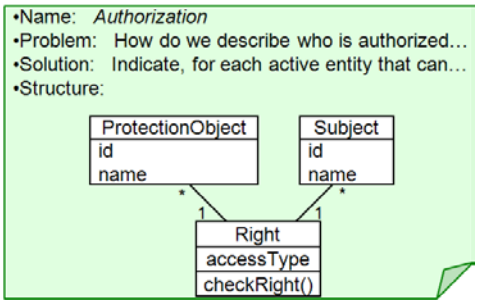

図 1　セキュリティパターンの例

おける一貫した適用が重要である．図 2 に SP 間の依存関係の例を示す．図 2 は，認証について記述されていて，機密性を向上させる Authenticator パターン，認可について記述されていて，完全性と機密性を向上させる Authorization パターン，ロールベースのアクセス制御に関する Role-Based Access Control(以下，RBAC)パターン，より具体的なアクセス制御に関する Reference Monitor パターンの依存関係の例を表している．ある SP，Px と Py が存在して，Px <- Py のとき，Py は Px に依存することを表している．例えば，Authneticator <- Authorization は，Authorization パターンの適用前に Authenticator パターンを適用すべきであることを表している．従って，セキュリティ要求に応じてとりうる適用の順序は複数存在する．4 つのパターンのとりうる適用順序は次の 3 通りである．

- Authneticator，Authorization，RBAC，Reference Monitor
- Authneticator，Authorization，Reference Monitor
- Authneticator，RBAC，Reference Monitor

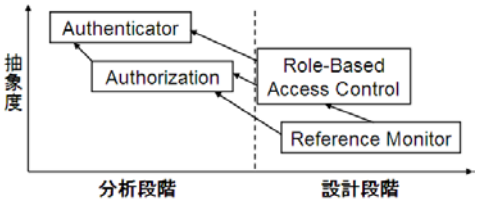

図 2　セキュリティパターン間の依存関係の例

適用する SP が依存するパターンを適用済みのモデルに対して適用することで，セキュリティを一貫して作りこむことができる．従って，識別された依存関係に基づく SP 適用を行わなければ，ソフトウェア全体のセキュリティが損なわれる可能性がある．

手動での SP の連続適用は次の 3 つの問題を持つ．

P1:　パターン間の依存関係を考慮しない誤った SP

適用の可能性

P2: 誤った箇所への SP 適用の可能性

P3: 時間や労力のコスト

しかし，SP 適用支援への取り組みは分類[2]や単体の SP の適用支援[3][5]にとどまり，パターン間の依存関係を考慮した SP 適用支援は試みられていない．

2.3 セキュリティパターン適用の失敗例

提案手法は，次の 2 通りの SP 適用における失敗を想定する．

- パターン間の依存関係を考慮しない SP 適用
- 誤った箇所への SP 適用

本章では，病院における患者の個人情報システムにおいて Authenticator パターンを用いず，Authorization パターンを適用することを例に用いて，パターン間の依存関係を考慮しない誤った SP 適用の例を述べる．

ある病院には患者の個人情報を管理するためのシステムが存在する．患者の個人情報管理システムのユースケース図を図 3 に示す．図 3 に示すように，この個人情報管理システムでは次の 2 つを扱う．

UC1: 患者の治療記録を見る

UC2: 患者の診察結果を記録する

また，個人情報管理システムには次の２つのセキュリティ要求が課せられているものとする．

SR1: 病院内の従業員のみが個人情報管理システムにアクセスできる．これにより患者の個人情報の機密性を維持する．

SR2: システムのユーザは割り当てられた権限があるユースケースを行うことができる．これにより患者の個人情報の機密性を維持する．

セキュリティ要求と照らし合わせると，個人情報管理システムは次の 2 つの問題を持つ．

第一の問題点は，現状の個人情報管理システムは，ユーザが病院の従業員かを判断する構造がないことである．つまり，悪意のある第三者が病院関係者になりすまし，患者の個人情報を盗み転売するなどのミスユースケース[11]が存在すると考えられる．従って，現状の個人情報管理システムは個人認証ができないという問題を持つ．

第二の問題点は，第一の問題を解決したと仮定しても，現状の個人情報管理システムでは病院関係者の誰もが患者の個人情報にアクセスして，読み込みや書き込みをすること全てが許可されていることである．従って，もともと権限のない病院関係者が悪意をもって，患者の診察結果を不正に書き換え，正しい情報が得られなくなるなどのミスユースケースが存在すると考えられる．従って現状の個人情報管理システムは権限をチェックできないという問題を持つ．

これらの問題が存在した時に Authenticator パターンを適用せず Authorization パターンを適用すると，認可するための構造ができるため，ユーザが医者であるか，看護師であるかなどは判別することができるが，認証する構造が存在しないため，誰でも自分が医者であると主張できてしまう．従って，依存関係を考慮した SP 適用を行わないと，もともとのセキュリティ要求を満たすことができない．

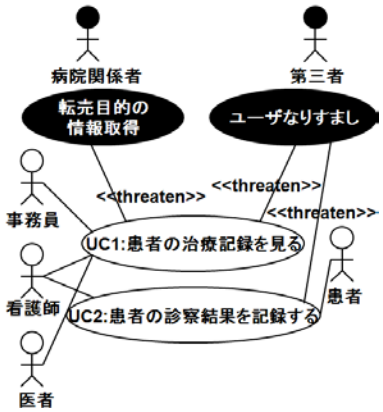

図 3 患者の個人情報管理システムのユースケース図

3. 提案手法

本章ではセキュリティパターン変換規則（以下，SP 変換規則）の記述方法，提案システムの手順，提案手法を用いることによる問題の解決について述べる．

提案手法における問題への解決を次の 3 つである．

S1: モデル変換の自動化

S2: SP 変換規則のライブラリ化

S3: 適用結果の痕跡を残し，以降の SP の適用に使用

前章の P1 の解決として S1 と S3，前章の P2 の解決として S1 と S3，前章の P3 の解決として S1 と S2 がそれぞれ対応する．

3.1 セキュリティパターン変換規則の記述方法

セキュリティ専門家が，SP をもとに ATL(ATLAS Transformation Language)[8]を用いて SP 変換規則を記述する．SP 変換規則は(1)事前条件，(2)引数，(3)操作の３つの要素から構成される．

(1)事前条件とは SP 適用の前提としていることがらである．適用する SP が依存するパターンの有無により事前条件の定義が異なる．

適用する SP が依存するパターンが存在しない場合は，適用する SP のロールに相当するクラスがモデル中に存在することを事前条件として定義する．

ある SP，Px と Py が存在して，Px が Py に依存する場合は，Py を適用した痕跡がモデル中に存在することを，Px を適用する事前条件として定義する．提案手法では過去に適用した SP の出力を事前条件とすることで，パターン間の関係を考慮した SP の連続適用を支援する．パターン間の依存関係はセキュリティ専門家が SP の中の関連パターンを参照することにより得る．また，SP の文脈や問題が，その SP が依存する SP の文脈や解決，結果と類似・包含関係にある場合にも，依存関係が得られる可能性がある．

例えば，[1]の中で Authenticator パターンの文脈には，”The authenticated user, represented by processes running on its behalf, is then allowed to access resources according to their rights.”(p. 323) と記述されている．一方，Authorization パターンの文脈には，”Any environment in which we have resources whose access needs to be controlled.”(p. 245)と記述されている．Authenticator パターンと Authorization パターンが互いに関連パターンであると明確に書かれてはいないものの，保護資産へのアクセスを制御するという点で類似していることから，２つのパターンには関係が存在すると判断できる．また，「認証されたユーザがその権限により保護資産にアクセスすることを許す」という記述は認証した後に，認可を行うべきであることを示している．従って，Authorization パターン適用前に Authenticator パターンを適用すべきであると判断することができる．

また，[1]の中で Authorization パターンの関連パターンには，”REFERENCE MONITOR complements this pattern by defining how to enforce the defined rights.”(p. 248)と記述されていることから，Reference Monitor パターン適用前に Authorization パターンを適用すべきであると判断することができる．

(2)引数とは適用する SP のロールに対応するクラスの名前など，ソフトウェア開発者が提案システムに入力する必要のあるパラメータのことである．

(3)操作とはパターン適用前のモデルに対して，SP 適用時に追加したクラスや関連の集合を適用後のモデルにマッピングすることである．操作では，適用済みのパターンと適用するパターンとの間に共通のロールが存在するかをパターンの構造の記述から判断する．

例えば，Authenticator パターンの Subject は，[1]に”A Subject, typically a user, requests access to system resources.”(p. 324)と記述されている．また，Authorization パターンの Subject は，1)に”The Subject class describes an active entity that attempts to access a resource (Protection Object) in some way.” (p. 246)と記述されている．２つの SP の Subject ロールの記述が非常に類似しているため，２つの SP の Subject ロールは共通であると判断できる．

つまり，Px<-Py であるとき，両者に共通に登場するロール r は，各パターンの変換規則導出のもととなる．具体的には，Px の変換規則において変換後の r やその周辺が有する性質を，Py の変換規則における変換前の r やその周辺が満たすべき性質として記述する．例えば，Authenticator パターン適用後の Subject ロールに相当するクラスやその周辺が有する性質が，Authorization パターン適用の条件となる．従って，セキュリティの専門家は Authenticator パターンの Subject ロールと Authorization パターンのロールが対応するように SP 変換規則を記述する．

以上のような SP 変換規則を用いてモデル変換することにより，SP 適用を実現する．

3.1.1 セキュリティパターン変換規則の記述の具体例

図 1 に示した Authorization パターンの適用を例として SP 変換規則の記述方法を述べる．

事前条件は，’AuthenticatorSubject’と記述されたステレオタイプを持つクラスが存在することである．このステレオタイプは Authorization パターンが依存する Authenticator パターンがモデル中に適用されている痕跡である．

引数は Authorization パターンの ProtectionObject ロールに相当するモデル中のクラス名である．

操作は次の 4 つである．

- 引数として開発者が入力した ProtectionObject ロールに相当するクラスに’ProtectionObject’と記述したステレオタイプを作成
- Right ロールに相当するクラスを追加
- Subject ロールに相当するクラスと Right ロールに相当するクラスとの関連，ProtectionObject ロールに相当するクラスと Right ロールに相当するクラスとの関連をそれぞれ追加
- ProtectionObject ロールに相当するクラスと Subject ロールに相当するクラスとの関連を削除

以上のマッピングを行うために，熟練開発者はモデル変換言語である ATL を用いて SP 変換規則を記述する．ATL を用いて記述した Authorization パターンの SP 変換規則を図 4 に示す．図 4 の 3 行目の isProtOb 関数が，ソフトウェア開発者が入力した ProtectionObject ロールに相当するクラス名の文字列とモデル中のクラス名が一致しているかを判定する．7 行目の hasStereotype 関数が，’AuthenticatorSubject’と記述されたステレオタ

イプを持つクラスかを判定する.

```
-- 指定クラスが ProtectionObject ロールに該当するか判定する
helper context UML!Class def:isProtOb() : Boolean =
if self.name = thisModule.ProtObName
 then true else false endif;
-- 指定ステレオタイプが引数と同じかどうか判定する
Helper context UML!Class def:hasStereotype(stereotype
 : String) : Boolean =
 self.stereotype->collect(s | s.name)->includes(stereotype)
・・・
-- ProtectionObject 変換
rule ProtectionObjectClass {
 from s : UML!Class (s.isProtectionObject()) ・・・
  stereotype <- stereotypePO),
  -- ステレオタイプ追加
  stereotypePO : UML!Stereotype (
   name <- 'ProtectionObject',  ・・・
-- Subject 変換
rule SubjectClass {
 from s : UML!Class (s.hasStereotype('Authenticator
 Subject')) ・・・
```

図 4　Authorization パターンの SP 変換規則の記述例（抜粋）

提案手法では，モデル変換の形式として，ステレオタイプによるマーキング方式を採用した．マーキング方式を採用した理由は，モデル変換の出力ごとに SP を適用した痕跡を残すため，ソフトウェア開発者が出力されたモデルを見てどの SP を適用したかを容易に判断できるからである．

また，モデル記述言語として UML，モデル変換言語として ATL を採用した．ATL を採用した理由は，ATL がモデル駆動型開発におけるモデル変換の標準である Queries/Views/Transformations(QVT)を実装していることから，ソフトウェア開発者への導入や今後のシステムの拡張が QVT を実装していないモデル変換言語と比較して容易になるからである．

3.2　提案システムの手順

提案手法では，XML Metadata Interchange(XMI)形式の UML モデルを提案システムへの入力としてモデル変換を行い，変換時に SP を適用する．

提案システムの全体像を図 5 に示す．提案システムを用いた SP の適用手順を以下に示す．

1. 開発者は，適用したい SP 変換規則を選択する．
2. 開発者は，パラメータと開発者が作成したモデルを提案システムに入力する．
3. システムがモデル変換を行い，SP 適用後のモデルを出力する．

提案システムは入力，出力モデルとして，システムの構造を定義するクラスモデルとシステムの振る舞いを定義するコラボレーションモデルの 2 つを扱う．入力モデルと出力モデルはともに XMI 形式である． また，一度記述された SP 変換規則は，以降の別の開発においても再利用できる．

得られた出力モデルを次の変換の入力モデルとして使用し，再びモデル変換することで，SP の連続適用が可能になる．SP 変換規則の多くは，他の SP の適用の痕跡を目印にして適用箇所を自動決定するので，SP 間の関係を考慮した適用が提案手法により実現される．

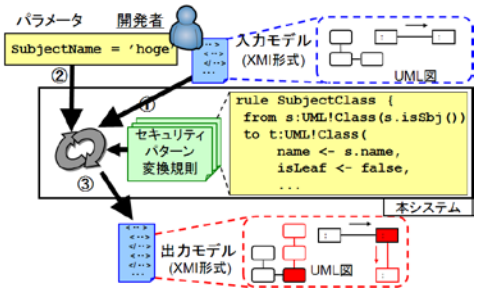

図 5　提案システムの全体像

3.3　手法の適用可能範囲

図 6 に扱える SP の分布を示す．

提案手法で扱える SP は 27 パターン存在する．提案手法は UML モデルでの構造が記述されている SP であれば，扱うことができる．19 パターンの提案手法で扱えない SP は，モデルの形で表わされていない．そのため，SP 変換規則を記述できないので本手法で扱うことができない．

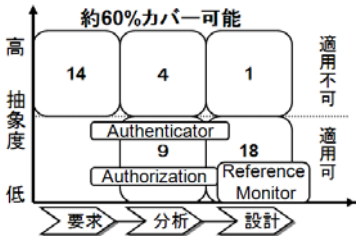

図 6　扱えるセキュリティパターンの分布

4. 具体例

本章では，提案手法の適用例として前述の病院の患者の個人情報管理システムをとりあげる．個人情報管理システムのクラス図を図 7 に，図 7 のクラス図の XMI の抜粋を図 8 に，個人情報システムの振る舞いを表すコラボレーション図を図 9 に示す．前述の図 3 に示した脅威に対する解決策として，2 つの SP を適用する．

まず，認証に関する Authenticator パターンを適用し，次に，認可に関する Authorization パターンを適用する．

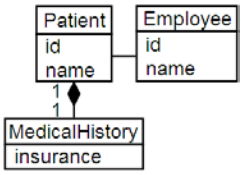

図 7　患者の個人情報管理システムのクラス図

```
<UML:Class xmi.id = 'a2' name ='Employee' ・・・
```

図 8　個人情報管理システムのクラス図の XMI（抜粋）

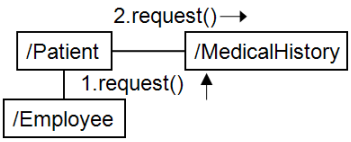

図 9　個人情報管理システムのコラボレーション図

4.1　Authenticator パターンの適用

開発者は Authenticator パターンが記述された ATL ファイルを選択後，システムに入力モデルとパラメータを入力する．sbjName = ‘Employee’と入力すると，提案システムは，Employee クラスが Authenticator パターンの Subject ロールに相当すると判断し Authenticator パターンを適用する．このときに，Employee クラスが Subject に相当すると判断して適用した痕跡を残すことを，ステレオタイプを追加することにより実現している．Authenticator パターンのクラス図を図 10 に，Authenticator パターン適用後のクラス図を図 11 に，Authenticator パターン適用後のクラス図の XMI を図 12 に示す．また，Authenticator パターンは振る舞いについても SP の中で触れられている．図 13 に Authenticator パターンのコラボレーション図を示す．

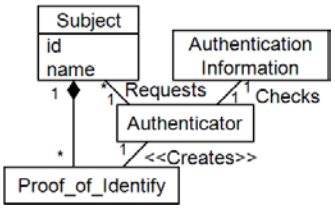

図 10　Authenticator パターンのクラス図

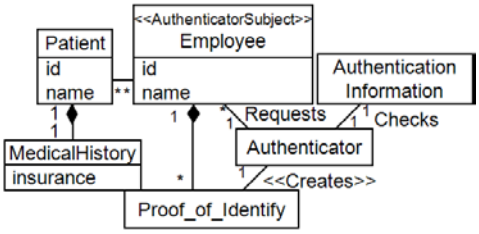

図 11　Authenticator パターン適用後のクラス図

```
<UML:Class xmi.id = 'a2' name ='Employee'　・・・
    <UML:Stereotype xmi.idref = 'a3'/>　・・・
  <UML:Stereotype xmi.id = 'a3' name = 'Subject'　・・・
<UML:Class xmi.id = 'a10' name = 'Authenticator'・・・
```

図 12　Authenticator パターン適用後のクラス図の XMI（抜粋）

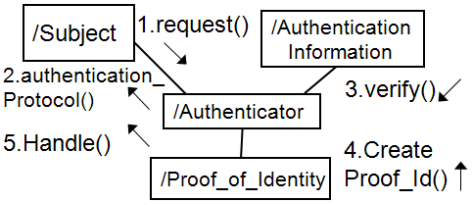

図 13　Authenticator パターンのコラボレーション図

クラスモデルの変換の際に sbjName = ‘Employee’と入力したので，提案システムは，Employee オブジェクトが Authenticator パターンの Subject ロールに相当すると判断し，Authenticator パターンを適用する．このときに，Employee クオブジェクトが Subject ロールに相当すると判断して適用した痕跡を残すことを，ステレオタイプを Employee オブジェクトに追加することにより実現している．Authenticator パターン適用後のコラボレーション図を図 14 に示す．Authenticator パターンを適用することにより，認証する構造ができたので，悪意のある第三者によるユーザなりすましへの対策を施すことができた．しかし，現状では病院の従業員であれば誰でも患者の個人情報にアクセスすることができるため，悪意のある従業員が患者の個人情報にアク

セスできてしまうことが問題である．そこで，認可について記述されている Authorization パターンを適用する．

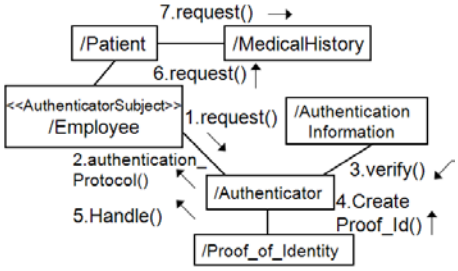

図 14　Authenticator パターン適用後のコラボレーション図

4.2　Authorization パターンの適用

開発者は Authorization パターンが記述された ATL ファイルを選択後，システムに入力モデルとパラメータを入力する．protObName = ‘Patient’と入力すると，提案システムは，Patient クラスが Authorization パターンの ProtectionObject ロールに相当すると判断する．また，Subject ロールに相当するクラスは，前回の Authenticator パターン適用時に Employee クラスに残した‘AuthenticatorSubject’と記述されているステレオタイプが存在するため，提案システムはステレオタイプから，Employee クラスが Authenticator パターンの Subject ロールかつ，Authorization パターンの Subject ロールに相当すると判断する．以上のプロセスを経て提案システムが Authorization パターンを適用する．

このときに，Patient クラスが ProtectionObject ロールに相当する，また Authorization パターンの Subject ロールと Employee クラスに相当すると判断して適用した痕跡を残すことを，ステレオタイプを追加することにより実現している．図 15 に Authorization パターンのクラス図を，図 16 に Authorization パターン適用後のクラス図を，図 17 にパターン適用後のユースケース図を示す．

図 17 が示すように，個人情報管理システムは認証する構造がなく，認可を行う構造がないという 2 つのセキュリティ上の問題を抱えていた．2 つの問題を解決するために，個人情報管理システムのモデルに Authenticator パターンと Authorization パターンを適用した．Authenticator パターンを正しくモデルに適用することで，ユーザなりすましへの対策を，Authorization パターンを正しくモデルに適用することで，権限なき利用への対策を正しく施すことができると考えられる．

4.2 節で行った SP の連続適用の様子を図 18 に示す．

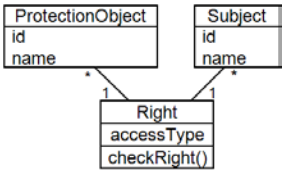

図 15　Authorization パターンのクラス図

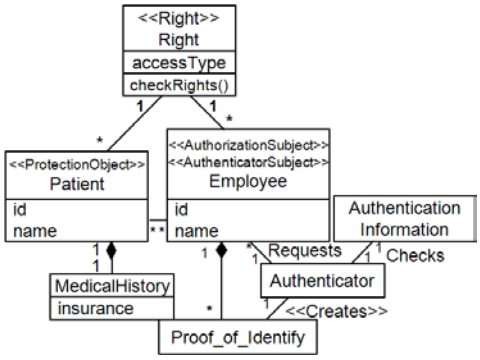

図 16　Authorization パターン適用後のクラス図

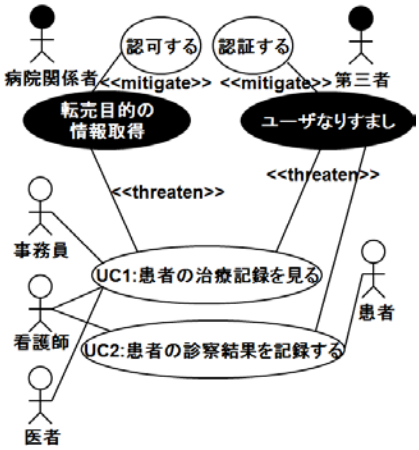

図 17　パターン適用後のユースケース図

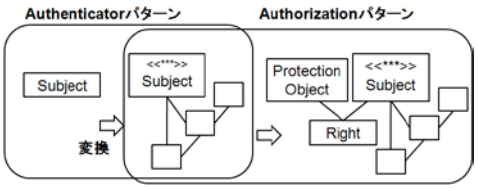

図 18 セキュリティパターンの連続適用の様子

5.　評価

本章では提案手法での SP 適用と手動での SP 適用との比較評価を行う．SP 適用における作業ステップ数と所要時間を用いて，提案手法および手動による SP の適用効率を評価した．

作業ステップ数は，(1)クラスの追加・削除，(2)関連の追加・削除，(3)クラス名の入力，(4)モデル変換（パターン適用）(5)モデル変換時に入力する引数の指定をそれぞれ 1 ステップとして算出した.

所要時間は，UML モデリングツールの使用経験がある学部 4 年生 6 人を対象とした利用実験で計測した SP の適用に要した時間の平均値とした. 提案手法および手動での SP 適用の作業ステップ数を表 1 に示す. また, 被験者 6 名が SP 適用の所要時間を図 19 に示す.

表 1　SP 適用時の作業ステップ数の比較

	手動	提案手法
Authenticator	7	2
Authorization	4	2
Reference Monitor	15	1

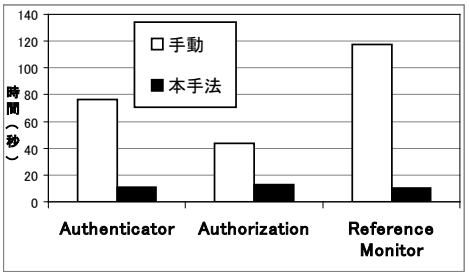

図 19　SP 適用の所要時間の比較

どの SP に対しても提案手法では作業ステップ数, 適用に要する時間ともに低減された．手動適用に対する提案手法の低減幅は，作業ステップ数が 50%から 93%，要した時間が 71%から 91%である．このことから提案手法を用いることで，開発者が SP を適用するコストが低減されたと言える.

6.　関連研究

Yu らは，i*モデルに対して ATL を用いたモデル変換を行うことにより, SP を適用する手法を提案した[3]. また，Horvath らはペトリネットを用いたモデル変換を行うことにより，SP を適用する手法を提案した[5].

提案手法では，ATL を利用して UML モデルに対してモデル変換を行うことにより, SP を適用する点で既存の研究と異なる.

7.　おわりに

本稿では, SP の適用をモデル駆動型開発における変換規則として定義し，自動適用する手法を提案した.

これまでは, SP 間の関係を考慮した自動適用が存在せず，開発者が SP 間の関係を把握し，手動で適用する以外の手法が存在しなかった．提案手法は，SP 変換規則の記述方法を具体的に定め，モデル変換の際に適用箇所の痕跡としてステレオタイプをモデル中に追加することで，パターン間の依存関係を考慮した SP の連続適用の自動化を実現した.

本稿では，Authenticator パターンと Authorization パターン，Authorization パターンと Reference Monitor パターンなど，依存関係のある SP の組に対して依存関係を考慮した連続適用手法を提案した．提案手法は他のパターンにも応用できる可能性がある. しかし, GoF デザインパターンのような他のパターンのパターン間には SP のような強い依存関係はみえにくかった.

今後の課題として以下の 4 つが挙げられる.

- 提案手法で扱うことのできる 27 のすべての SP についてパターン間の関係も考慮し，網羅する
- セキュリティ特性に関して厳密に記述して，SP を使うことによりシステム全体のセキュリティを保証する
- 久保らの手法[4]を応用し，パターン文書からパターン間の依存関係を導出する
- SP の適用の正確性を定量的に評価する

参考文献

1) Schumacher, M et al.:Security Patterns, Wiley, 2006
2) Yoshioka, N et al.: A Survey on Security Patterns, Progress in Informatics, No.5, pp. 35-47. 2008
3) Yu, Y et al.:Enforcing a Security Pattern in Stakeholder Goal Models, ACM Proc. ACM workshop on Quality of protection (QoP'08), 2008
4) Kubo, A et al.: Extracting Relations among Embedded Software Design Patterns, Journal of Integrated Design and Process Science (SDPS), pp.39-52, vol.9, no.3, 2005
5) Horvath, V. Dorges, T.:From Security Patterns to Implementation Using Petri Nets, ACM, 2008
6) Schumacher, M. Roeding, U.:Security Engineering with Patterns, LNCS2754, pp. 121 – 140,2003
7) Frankel, D.:Model Driven Architecture, Wiley,2003.
8) Eclipse.org. ATL Project, http://www.eclipse.org/m2m/atl/
9) Jurjens, J.: Secure Systems Development with UML, Springer, 2004
10) Lodderstedt, T et al.:SecureUML: A UML-Based Modeling Language for Model-Driven Security, In Proceedings of the 5th International Conference on the Unified Modeling Language, pp. 426-441,2002
11) Sindre, G et al.: Elicting security requirements with misuse cases, Requir.Eng., pp.34-44, vol 10, no.1, 2005