情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-SE-164 No.9
Vol.2009-EMB-13 No.9
Vol.2009-CSEC-45 No.9
2009/5/28

アスペクト指向による セキュアなアプリケーション開発環境の提案

大久保　隆夫[†1]

セキュアなアプリケーションの開発においては，対策が個々の工程に閉じたアドホックなものになっている，開発者に必要なセキュリティ知識が不足しているといった問題がある．本稿では，アスペクト指向の概念によりセキュリティ知識の必要な開発を分離し，工程全体通じてセキュアなアプリケーション開発を実現する開発環境「Security Injector」を提案する．

A proposal of aspect-oriented secure application development environment

Takao Okubo[†1]

There are several problems with secure application development: each methods are ad-hoc, and developers' security knowlege is not enough. This paper proposes a new framework called "Security Injector" for secure application development. Security Injector framework is composed of the aspect-oriented development process and elemental technologies in every development stages. This paper also evaluates the efficiency of the proposed framework with case studies.

1. はじめに

本稿では，セキュアなアプリケーションを開発するための手法として，開発工程を通じたアスペクト指向のプロセスと，各要素技術で構成される開発環境"Security Injector"を新規に提案する．また提案した開発環境のケーススタディによる評価を行い，その効果について確認する．

2. 従来技術の問題点

近年，IT 技術の進歩により，様々なサービスがインターネットを経由して提供されるようになった一方で，同じくインターネットを介したサービスへの攻撃による脅威も高まっている．アプリケーションが攻撃によって被害を受けてしまう原因の多くは，適切な対策をアプリケーションに実装しておけば攻撃は防げていたのに，現実には何らかの原因で対策がされず，結果として脆弱性が残ってしまったためと考えられる．情報処理推進機構 (IPA) が受け付けた日本国内の脆弱性届出件数が，2007 年度だけで 572 件，2004 年度からの累計では 1,749 件に達していることもその推測を裏付けている．脆弱性をなくし，攻撃に対しても安全なアプリケーションを実現するには，アプリケーション開発工程の早期段階からセキュリティを意識し，脅威への対策を意識した開発が必要になる．筆者らが行っている Web アプリケーションプログラムのシステム・インテグレーション (SI) におけるセキュリティ構築支援の経験によれば，現実のソフトウェア開発において，セキュリティの導入が成功していない主な要因としては，次の 3 点が挙げられる．

- ステークホルダ[⋆1]のセキュリティ知識不足
- 手段，人的資源不足による重いセキュリティ作業コスト
- 開発工程を通した有効なセキュリティ開発手法の欠如

セキュリティに関連したソフトウェアの開発手法としては，既に提案され，用いられている従来技術がいくつかある．しかし，従来技術の中で，従来のソフトウェア開発手法を基本にする手法 (ゴール指向分析[1)2)3)4)]，UML の拡張[5)]) は，既存の開発手法との整合性はよいが，セキュリティの観点が不足している．また，従来技術の中で，セキュリティ独自の開発手法を採用するもの (SDL[6)]，CC[7)] など) は，多くのセキュリティ有識者による作業を必要とし，また，従来のソフトウェア開発とは整合しないものが多い．従って，従来技術では上記に挙げた問題の解決は困難である．また，各技術は対象とする開発工程 (要求分析，設計，実装，テスト) に閉じており，各工程間の関連を応用した技術，手法はあまり見られない．このため，従来のソフトウェア開発手法で用いられるような，前工程 (例えば要求分析) を先に行うことにより，後工程 (例えば設計) の効率が向上するような効果が期待できない．

†1 富士通研究所
Fujitsu Laboratories ltd.

⋆1 顧客，エンドユーザ，開発担当者，プロジェクト管理者などを含む，対象システムに関わるすべての利害関係者

Vol.2009-SE-164 No.9
Vol.2009-EMB-13 No.9
Vol.2009-CSEC-45 No.9
2009/5/28

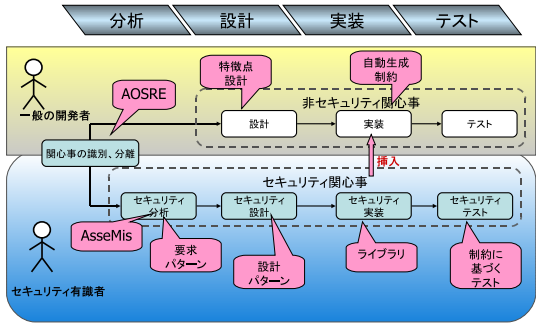

図 1 Security Injector の構成

3. アスペクト指向開発環境”Security Injector”の提案

本稿では上記に述べた問題を解決する手法として，アスペクト指向開発環境 (フレームワーク)(以下，”Security Injector”と呼ぶ) を提案する．

Security Injector は，アスペクト指向ソフトウェア開発 (AOSD) においては，ソフトウェアがその関心事別に実装される点に着目し，セキュリティ関心事と非セキュリティ関心事を分離して開発する AOSD の手法を導入する．また，依存性注入 (Dependency Injection) の考え方を用い，非セキュリティ関心事からセキュリティ関心事への依存性を可能な限り排除し，非セキュリティ関心事はセキュリティ知識に乏しい開発者でも従来のソフトウェア開発手法による開発が可能なようにする．また，従来の AOSD へのセキュリティへの適用は実装レベルのものが多かったが，Security Injector では要求分析段階からセキュリティ関心事を分離して別に開発を行うことにより，開発工程のすべての段階におけるセキュリティ知識不足の解消を目指す．Security Injector の構成を図 1 に示す．

Security Injector は下記の 2 種類の要素技術で構成される．

- セキュリティ関心事およびその実現を一般の開発者から分離して開発を行うアスペクト指向のプロセス
- 上記プロセスの効率的な実現を支援する各開発工程ごとの開発 (要求分析，設計，実装，テスト) 支援手法

前者のプロセスは開発工程の各工程に対応する次の手順で構成される．

要求分析 セキュリティ要求の確定，およびその段階での非セキュリティ関心事とセキュリティ関心事の分離．

設計 要求に従った各関心事の設計と，関心事の横断点の設計．また．脆弱性の排除，および確認容易性の向上を目的とした制約の導入．

実装 横断点を含む関心事の，制約に基づく実装．

テスト 制約の確認テスト．

次に，上記プロセスの実現を支援する各要素技術について述べる．

4. 要 素 技 術

4.1 要 求 分 析

要求分析段階では，Security Injector は次の 2 つを達成目標とする．

(1) 非セキュリティ関心事とセキュリティ関心事を分離し，前者を一般開発者，後者をセキュリティ有識者が担当し，各関心事に基づいた要求定義を可能にする．

(2) セキュリティ要求を定義するために行う脅威分析において，限られたセキュリティ知識で効率的な脅威分析を可能にする．

Security Injector では，(1) の目標を実現する要素技術として，関心事を分離するプロセス (アスペクト指向セキュリティ要求策定プロセス (AOSRE)) を，(2) の目標を実現する要素技術として，セキュリティ要求の記法ミスユースケース[8] を拡張した記法 (資産ベースミスユースケース (AsseMis))[9]，および Web アプリケーションにおける脅威，対策をパターン化した Web セキュリティ要求パターンを用いる[10]．

4.2 設　　計

設計段階においては，Security Injector は「アスペクト指向」と「セキュリティ」という 2 つの観点に立った設計を行う必要がある．すなわち，次の 4 つの条件を満たす必要がある．

(1) 非セキュリティ関心事について，一般の開発者はセキュリティを極力意識せずに設計できる．一方，セキュリティ有識者はセキュリティの設計に専念する

(2) 関心事は別々に実装されるにも関わらず，セキュリティは非セキュリティ関心事の中でセキュリティが必要とされる箇所に適切に挿入される

(3) 「セキュリティが必要とされるかどうか」は，要求分析段階で定義したセキュリティ要求に従う．

(4) セキュリティ有識者の数が限られていることを考慮すると，セキュリティ関心事の設

Vol.2009-SE-164 No.9
Vol.2009-EMB-13 No.9
Vol.2009-CSEC-45 No.9
2009/5/28

計は汎用的で，かつ再利用可能であるほど望ましい

上記の条件 (1)〜(3) を満たすために，要素技術として，依存性注入に基づく関心事横断点の設計手法と，制約の導入を用いる．また，(4) を満たすために，セキュリティ設計のパターンを用いて再利用を促進する[11)12)]．

4.2.1 依存性注入 (DI) に基づく関心事横断点の設計と，制約の導入

前項で述べたように，Security Injector では，非セキュリティ関心事からセキュリティ関心事への依存性を排除する形で開発を進める必要がある．そのため，非セキュリティ関心事上にセキュリティを挿入するべき特徴が存在する必要があり，設計においてはその特徴を定義し，設計しなければならない．そこで，非セキュリティ側のクラスを，AsseMis のユースケースよりセキュリティプロパティを継承する形で自動生成させることを提案する．

本提案では，クラス名，メソッド名，フィールド名等の命名規約および自動生成機能を利用する．ここで自動生成とは，ユースケース図からクラス図，シーケンス図の自動生成および，シーケンス図からソースコード (雛形) の自動生成をさす．次に，クラス図やシーケンス図に対して，自動生成を利用して上記特徴点を挿入する方法を提案する．

4.2.2 クラス図，シーケンス図の設計

- 自動生成による特徴点の挿入
  [9)] で提案した，ユースケースにデータ表記とセキュリティプロパティを追加した AsseMis 表記から，データクラス，およびユースケースに対応するクラスを自動生成する．自動生成は次の手順で行う．
  (1) 拡張ユースケース図に記述された各データ資産に対応するデータクラスを生成する．
  その際，データ資産に付与されたセキュリティプロパティのステレオタイプを，生成後のクラスにも記述する．
  (2) 拡張ユースケース図に記述された各ユースケースに対応するデータクラスを生成する．その際，ユースケースに付与されたセキュリティプロパティのステレオタイプを，生成後のクラスにも記述する．
- アスペクト (セキュリティ関心事) の設計
  アスペクトの設計においては，関心事横断点を発見するためのアルゴリズム (アスペクト指向プログラム (Aspect-oriented Programming: AOP) におけるポイントカット[⋆1]に相当) の設計，および挿入するセキュリティ処理 (AOP におけるアドバイス) の設計を行う．

ポイントカットの設計では，まずユースケースから自動生成されたクラス図に基づき，クラス図，シーケンス図の詳細設計が行われることを前提とし，その出力を利用する．その手順を次に示す．

(1) 要求分析で明確になった資産に対する脅威が，シーケンス図の中のどの時点に影響するかを識別する
(2) 脅威に対する対策として，要求分析で識別されている対策ユースケースが，シーケンスのどの時点で必要になるかを識別する．識別した点が横断点，すなわちその対策が挿入される点となる．
(3) 上記に基づき横断点を設計する．

分析結果に基づいて設計する例について述べる．

シーケンス図を用いた設計を図 2〜3 に示す．開発者は，自動生成により特徴を挿入されたクラス (図 2 におけるユースケース) を用いて図 2 に示すようにシーケンス図を作図する．シーケンス図完成後，セキュリティ有識者は，セキュリティ要求に従い，セキュリティ対策の設計を行う．この例では，AOSRE,AsseMis の分析結果により，可用性を要する「UC0」ユースケースには,「利用者識別認証」のセキュリティ対策を行うことになっている．図 2 において,「UC0」クラスには A(可用性) のセキュリティプロパティが付与されているため，セキュリティ有識者はこの「UC0」に対して認証をするセキュリティ処理を挿入する必要がある．そこで，次に図 2 の中でセキュリティ処理を挿入する箇所を決定する．一般に，ある機能が利用者識別認証処理を必要とする場合，その認証処理は利用者が機能を利用する「前」の行う必要がある．そこで，セキュリティ有識者は図 3 に示すように，UC0 において最初に機能が活性化される前に，利用者識別認証のアスペクトを挿入する[⋆2]．

4.3 実装・テスト

実装，テスト段階においては，次の 4 点が必要条件となる．

(1) 設計段階において，設計した関心事の横断点通りに横断点が実装される
(2) 実装したセキュリティ機能の再利用することにより，開発効率が向上する

⋆1 処理をプログラムのどこに挿入するかの指定．メソッドの呼び出し，実行，初期化，代入処理などのタイミングを用いて記述する

⋆2 シーケンス図におけるアスペクトの記述方法は文献[13)] に従う．

Vol.2009-SE-164 No.9
Vol.2009-EMB-13 No.9
Vol.2009-CSEC-45 No.9
2009/5/28

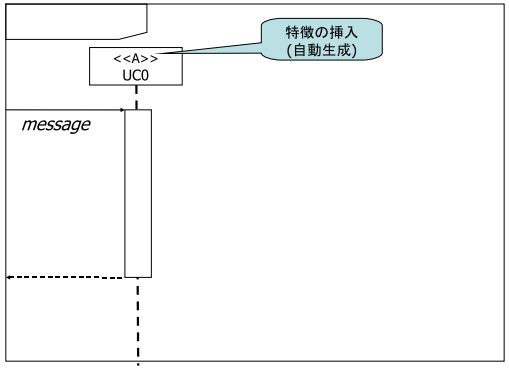

図 2 シーケンス図の例

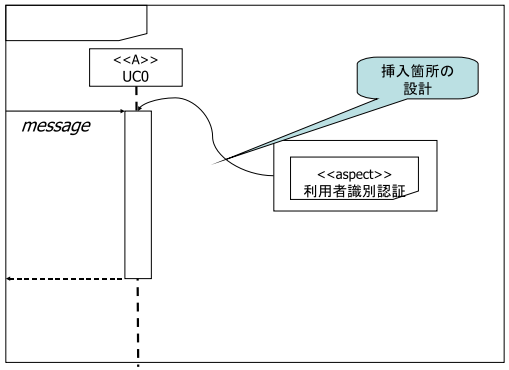

図 3 アスペクトを追加したシーケンス図

(3) セキュリティ機能が適切に，脆弱性のないように実装される
(4) テストにおいてセキュリティの確認を容易にする

(1)(4) を満たす方法として横断点を設計通りに実装するためのソースコードの自動生成手法を，(2)(3) を満たす方法として，ライブラリ化による再利用を用いる．また，セキュリティライブラリの実装例として，インジェクションを防止するライブラリ[14)15)] が，また，(4) を満たすテスト手法として，制約に基づくテスト手法が提供される．

5. ケーススタディ

Web アプリケーションを例に，要求分析段階から Security Injector を適用して開発するという想定でケーススタディを行った．対象のアプリケーションは，インターネット経由で本を検索，注文する Web のブックストアのアプリケーションとする．利用者は Web ブラウザを経由して，本を検索したり，注文を行うことができる．また，システムの管理機能も Web で提供され，管理者はアプリケーションを通して利用者情報の参照，変更を行うことができる．なお，開発には Java 言語および Servlet を利用し，データの格納には SQL データベースを用いるものとする．このアプリケーションの開発に Security Injecto のプロセスおよび提案した各要素技術を適用した．

5.1 要 求 分 析

対象のアプリケーションの要求分析を，4.1 で述べた AOSRE のプロセスに従い，セキュリティ要求を抽出してセキュリティ関心事と非セキュリティ関心事に分離する作業を行う．また，そのための脅威分析の過程で AsseMis 記法と，Web セキュリティ要求パターンを用いる．対策案を抽出した結果を図 4 に示す．

5.2 設　　計

設計では，前項の要求分析で抽出された非セキュリティ関心事およびセキュリティ関心事の設計を行う．アクセス制御が対策 (=セキュリティ関心事) として抽出されたので，アクセス制御を例として説明する．

5.2.1 アクセス制御の設計

対象とするアプリケーションでは，役割単位，および利用者単位でのアクセス制御が必要になるため，4.2 で述べた RBAC の設計パターンを利用する．RBAC の設計パターンでは，次の 3 つをアプリケーション設計時に用意する．

- 役割単位の権限定義ファイル (自動生成により取得)

Vol.2009-SE-164 No.9
Vol.2009-EMB-13 No.9
Vol.2009-CSEC-45 No.9
2009/5/28

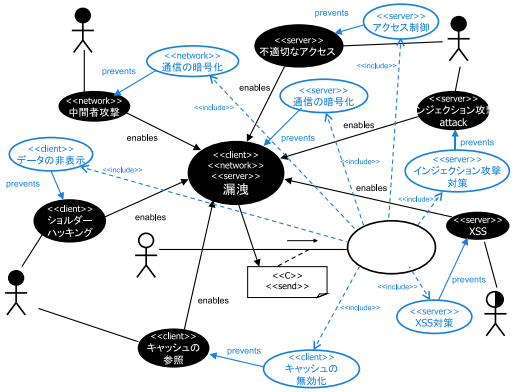

図 4 ケーススタディ：セキュリティ要求 (対策)

- データクラス (自動生成により取得)

- ユースケースに対応する機能クラス (自動生成により取得)

上記のうち，権限定義ファイルは，要求分析における AOSRE の「アクセス権限の定義」のステップにおいて既に自動生成されているのでそれを用いる．残りの 2 つについては，自動生成のアルゴリズムを用い，データクラスおよびユースケースに対応する機能クラスのクラス図を自動生成する．

もう一つのアクセス権検査に必要な権限検査ライブラリは，Security Injector フレームワークにより提供される．

クラスの自動生成後，非セキュリティ関心事の開発者は，クラスの詳細定義，および制御フロー (シーケンス) の定義などを行う．その際，自動生成で付与されたセキュリティプロパティのステレオタイプは変更しないようにする．

5.3 実　　装

5.3.1 アクセス制御の実装

RBAC のための自動生成のアルゴリズムを用いて Java クラスを自動生成する．その結果，例えば個人情報クラスについては以下に示すクラスが自動生成される．これらの中のセキュリティプロパティに関連するメソッド呼び出しは，予め定義したポイントカットにより，補足し，アクセス検査を行うことが可能になる．

5.3.2 SQL 呼び出しの実装

SQL 呼び出しに関しては，SQL インジェクション防止のライブラリの API を使っていれば，インジェクションは防止されるので，以下のコーディング制約を設ける．

- SQL 呼び出しには，必ず SafeStatement クラスの execute() を用い，他の API(Statement クラスや PreparedStetement クラス) は使わない．
- SafeStatement に SQL の文字列を追加する際，パラメータになる変数以外の文法要素は，必ず規定した定数の型を用いる．

5.4 テ　ス　ト

テストでは，採用したライブラリに基づき，規定された制約の遵守を確認するテストを行う．ケーススタディで例を提示したアクセス制御，SQL インジェクションのセキュリティテストについては，以下を実施する．

- 自動生成したデータクラス，機能クラスに付与された注釈が変更されていないか
- データクラスのデータに対するアクセスで，データクラスのメソッドを経由しないアクセスがされていないか
- Security Injector フレームワークの提供する SQL インジェクションライブラリの API 以外を使って，SQL 呼び出しを行っていないか

ケーススタディへの提案手法の適用により，非セキュリティ関心事とセキュリティを分離し，かつセキュリティが要求定義通りに実現されていることが確認できた．

6. 評　　価

提案した Security Injector フレームワークについて，評価を行う．評価は，次の問題点が解決されているかを問題ごとに検証する．

(1) 各開発工程における問題 (脆弱性の排除)
脆弱性の排除を直接扱った要素技術「インジェクション防止ライブラリ」について，脆弱性排除の効果を評価した結果，従来のライブラリでは防止が困難な場合でも，提案ライブラリを用いれば脆弱性を防止することが可能であることが確認できた．

(2) セキュリティ知識不足の問題
Security Injector ではアスペクト指向によりセキュリティ関心事を要求分析から分離し，以降それぞれの関心事ごとに開発させることにより，セキュリティ有識者が不足している開発体制でも開発を可能にしていることが確認できた．

Vol.2009-SE-164 No.9
Vol.2009-EMB-13 No.9
Vol.2009-CSEC-45 No.9
2009/5/28

(3) セキュリティ作業の効率の問題
AOSRE と AsseMis を組み合わせた手法，およびセキュリティ要求パターンを Web アプリケーション開発のセキュリティ要求策定に適用した結果について，従来作業とのセキュリティ作業量 (セキュリティ有識者が行った作業の作業量) の比較を行い，セキュリティの作業量が削減されることを確認した．

(4) 工程間の連携不足の問題
従来技術が工程ごとに独立しており，工程間が連携していなかったのに対して，Security Injector は，次のように前工程の結果を後工程が利用する連携によって，作業効率やセキュリティ向上，確認容易性の向上を実現していることを確認した．

7. おわりに

本稿では，セキュアなアプリケーション開発を困難にしている要因として，セキュリティ知識不足と，既存手法との不整合に着目した．これらの問題解決の方針としてセキュリティの関心事と非セキュリティ関心事を分離し，後者がセキュリティ知識によらずに開発可能なアスペクト指向の開発プロセスと，各開発工程においてその実践を支援する要素技術からなるフレームワーク”Security Injector”を新規に提案した．また，各要素技術の一部実装と評価を行うとともに，開発プロセス全体が有効に機能することをケーススタディにより確認した．開発者は本稿に示したプロセスに従い開発を行うことで，セキュリティ関心事を分離した開発が可能になる．開発プロセスのみを利用し，要素技術を用いなくても一定の効果は得られる．

また，上記開発プロセスを効率的に実行するための，開発工程に必要な要素技術を明確にし，実現方法を示した．一部 (ライブラリ) については，実際に実装を行った．その他の技術においてについては，ツールの実現方法を提示し，ツール化を行えば効率化が実現できることを示した．

今後の課題は，提案した手法をセキュアな統合開発環境として整備していくことである．例えば，AsseMis 作図ツールの整備と，パターン部品の提供，および自動適用およびクラス図，プログラムの自動生成を連携するなどして，1 つのライフサイクルに閉じて利用できる開発環境の実現をはかっていくことが考えられる．また，パターン，ライブラリの拡充や性能面からの検討，他の関心事 (セキュリティ，非セキュリティ) との競合，実際の開発への適用と評価も今後の課題である．

参 考 文 献

1) van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour.
2) Yu, E. S.-K.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering, *Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97)*.
3) Bresciani, P., Giorgini, P. and Mouratidis, H.: On Security Requirements Analysis for Multi-Agent Systems, *2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems SELMAS 2003 in conjunction with the 25th International Conference on Software Engineering (ICSE 2003)*, Portland, Oregon, USA (2003).
4) Gani, A., Giorgini, P., Manson, G. and Mouratidis, H.: Analysing Security Requirements of Information Systems Using Tropos, *the 5th International Conference on Enterprise Information Systems*, Angers, France (2003).
5) Jürjens, J.: *Secure Systems Development with UML*, Springer-Verlag (2004).
6) Howard, M. and Lipner, S.: *The Security Development Lifecycle*, Microsoft (2006).
7) CC: Common Criteria for Information Technology Security Evaluation v3.1 (2007). (accessed 2008-06-28).
8) Sindre, G. and Opdahl, A. L.I.: Eliciting Security Requirements by Misuse Cases, *Proc. TOOLS Pacific 2000*, pp.120–131 (2000).
9) Okubo, T. and Tanaka, H.: Identifying Security Aspects in Early Development Stages, *Proc. International Conference on Availability, Reliability and Security (ARES'08)*, Barcelona, Spain, pp.1148–1155 (2008).
10) Okubo, T. and Tanaka, H.: Web Security Patterns for Analysis and Design, *15th Cpmference on Pattern Languages of Programs (PLoP 2008)*, Nashville, TN, USA (2008).
11) 大久保隆夫，田中英彦：状態遷移に基づくアクセス制御のアプリケーション上での自動実装手法，コンピュータセキュリティシンポジウム (CSS)2008，pp.355–360 (2008).
12) 大久保隆夫，田中英彦：セキュアなアプリケーション開発のための要求・デザインパターンの提案，第 44 回 CSEC 研究会研究報告，pp.241–246 (2009).
13) Jacobson, I. and Ng, P.-W.: *Aspect-Oriented Software Development with UML*, Addison-Wesley (2004).
14) Okubo, T. and Tanaka, H.: Secure Software Development through Coding Conventions and Frameworks, *Proc. 2nd International Conference on Availability, Reliability and Security (ARES'07)*, Vienna, Austria, pp.1042–1051 (2007).
15) 大久保隆夫，田中英彦：インジェクション系攻撃防止ライブラリの評価，第 38 回 CSEC 研究会研究報告，pp.177–184 (2007).