情報処理学会論文誌 Vol. 49 No. 7 2292–2303 (July 2008)

# モデル駆動型開発に基づくSOAのセキュリティ開発プロセス

佐藤史子[†1] 中村祐一[†2] 小野康一[†1]

WS-Securityを適用したSOAアプリケーションでは複雑なセキュリティ設定が必要となるが，現状のSOA開発プロセスでは，どのようにセキュリティ設定を行うかが明確ではない．そのため，開発の下流プロセスでセキュリティ要件を考慮することになり，一部の開発者に負担がかかったり，正しいセキュリティ設定が難しかったりするという問題がある．本研究では，SOAアプリケーションのセキュリティ開発プロセスを明確にし，各プロセスで開発者が行う役割を定義する．また，複雑なセキュリティ設定を簡単に行うためにセキュリティ設定を抽象化したセキュリティ注釈を導入し，モデル駆動型セキュリティを提案する．これにより，アプリケーションのセキュリティ要件を開発の上流プロセスから考慮でき，モデル変換によりセキュリティ設定を自動生成することが可能になる．

# A Model-Driven Security Development Process for SOA Applications

Fumiko Satoh,[†1] Yuichi Nakamura[†2] and Kouichi Ono[†1]

An SOA application with WS-Security requires a complex security configuration. The security requirements are typically configured at the end of the development process, since it is not clear how and when they should be considered during the current development process of an SOA application. This increases the complexity and difficulty of the security configuration, and some developers of the downstream processes are forced to manage these configurations. In this paper, we clarify the security development process for an SOA application, and define roles for developers in each step of the process. We introduce security intents which specify the security requirements abstractly and propose Model-Driven Security to generate a security configuration automatically. Our contribution is providing the mechanisms to specify security requirements in upstream processes and generate detailed security configurations in downstream process by model transformations.

## 1. はじめに

サービス指向アーキテクチャ（SOA）はサービスをビジネスプロセスに基づいて組み合わせることでアプリケーションを構成するフレームワークである．特に，サービスが実行されるプラットフォームを限定せず，Webサービスや EJB などさまざまな手段で実装することが可能であることがSOAの特徴である．そのため，SOAアプリケーションを構成する複数のサービスが，Kerberos や PKI など異なるセキュリティ技術をサポートするプラットフォームで実行されていることは容易に考えられる．SOAアプリケーションのセキュリティを確保するにはこのような異なるセキュリティ技術を連携させなければならない．これをセキュリティドメインの統合と呼ぶ．

Webサービスセキュリティ（WS-Security）[1)] では，セキュリティドメイン統合のためのモデルが提案されており，サービスごとに異なるセキュリティ技術を統合することが可能である．しかし，現在のSOAアプリケーションの開発プロセス（以後，SOA開発プロセスとする）では，WS-Securityを利用したアプリケーションの開発プロセスが明確でない．そのため，現状ではセキュリティなしのサービスを実装した後，コンポジット・サービスを組み立てるプロセスになってアプリケーションに要求されるセキュリティ要件を考慮するというケースが多い．また，WS-Securityのための設定（以後，セキュリティ設定とする）を正しく記述するためには，サービス組み立てプロセスで取得できる情報だけではなく，ビジネスプロセスの要求から規定されるセキュリティ要件やセキュリティプラットフォームの情報など複数のプロセスにまたがった情報が必要であるにもかかわらず，現状ではある1つの開発プロセスでセキュリティ設定を行っている点に問題がある．そこで本研究では，各開発プロセスで扱う情報の粒度に合わせてそのプロセスでの設定内容を定義し，SOA開発プロセスに基づくセキュリティ設定の方法（以後，SOAのセキュリティ開発プロセスとする）を提案する．さらに，複数のプロセスで行われた設定を統合して正しいセキュリティ設定を自動生成するためのモデル駆動型セキュリティを提案する．

モデル駆動型セキュリティでは，開発の上流プロセスでビジネス要求からくるセキュリ

†1 日本アイ・ビー・エム株式会社東京基礎研究所
IBM Tokyo Research Laboratory
†2 日本アイ・ビー・エム株式会社サービス事業部
IBM Global Services

ティ要件を指定する．SOA 開発の場合，上流プロセスではプラットフォームを考慮せずアプリケーションが設計できることが利点であるため，セキュリティ設定においても詳細なセキュリティ設定を抽象化し，プラットフォームに依存しないセキュリティ要件を表すセキュリティ注釈を導入した．セキュリティ注釈からのモデル変換により，サービスがデプロイされるプラットフォームに対応した詳細なセキュリティ設定が生成される．プラットフォーム非依存モデル（PIM）であるセキュリティ注釈から，プラットフォーム依存モデル（PSM）であるセキュリティ設定を生成するためには，プラットフォームの情報を取得する必要がある．そこで，プラットフォームの情報をあらかじめモデル化しておくためのセキュリティ基盤モデルを導入し，これを参照しながらモデル変換することで，詳細なセキュリティ設定の自動生成を可能にした．本稿では，旅行予約アプリケーションでのセキュリティ設定を例とし，モデル駆動型セキュリティを適用した SOA のセキュリティ開発プロセスを示す．また，プロトタイプ実装した提案手法と既存の設定手法との比較・考察を行う．

モデル駆動型セキュリティにより，各開発プロセスで扱うセキュリティ関連情報を分類し，開発者の役割分担を明確にしたことで，開発の上流プロセスからセキュリティ要件を考慮できるようになった．また，モデル変換によるセキュリティ設定の自動生成により，開発者が設定ファイルを直接記述する負担を大きく軽減することができる．本研究で提案する SOA のセキュリティ開発プロセスは，安全な SOA アプリケーションの開発に大きく貢献できると考えている．

2 章では，現状の SOA 開発プロセスと問題点について述べ，3 章では，モデル駆動型セキュリティを適用した SOA セキュリティ開発プロセスを提案する．4 章では，モデル変換によるセキュリティ設定の生成について説明し，5 章では旅行代理店アプリケーションへの適用例を示す．最後に関連研究，考察を述べる．

## 2. SOA のセキュリティ開発プロセス

本研究が対象とするセキュリティ技術である Web サービスセキュリティ（WS-Security）[1] とそのセキュリティ開発における問題を具体的なアプリケーション例を用いて示す．

### 2.1 Web サービスセキュリティ

WS-Security は SOAP メッセージそのものに XML 署名，XML 暗号化することでメッセージを認証可能にし，データ完全性，データ秘匿性を保証する技術である．特に，異なるセキュリティ技術を利用している複数のサービス間でも個々のセキュリティ技術に依存せず，メッセージ交換におけるセキュリティを確保する，セキュリティドメイン統合を目的としており[2]，SOA アプリケーションの利点を損なうことなくサービス間の安全なメッセージ交換を実現することができる．

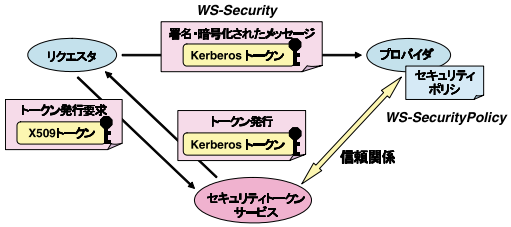


**図 1**　WS-Security によるセキュリティドメイン統合
Fig. 1　Security domain federation by WS-Security.

WS-Security による簡単なセキュリティドメイン統合の例を**図 1** に示す．ここでは，サービスリクエスタは PKI，サービスプロバイダは Kerberos という異なるセキュリティ技術で動作しているものとする．両者で交換されるメッセージに署名・暗号化するには，共有鍵などの秘密情報を抽象化して XML で表現したセキュリティトークンを共有する必要がある．一方，PKI では X509 トークン，Kerberos では Kerberos トークンなど，セキュリティ技術ごとに利用できるセキュリティトークンは限定されている．しかし，異なる種類のセキュリティトークンはセキュリティトークンサービスを介して交換することが可能であるため，リクエスタが持つ X509 トークンをプロバイダが要求する Kerberos トークンに変換し，メッセージに署名・暗号化することで，セキュリティドメインの統合が可能になる．プロバイダのセキュリティポリシには，受け取り可能なセキュリティトークンの種類や利用可能なセキュリティトークンサービスなどが記述されている．リクエスタはこのセキュリティポリシを参照することで，プロバイダが受け取り可能な署名・暗号化を行うことができる．

WS-Security では，ほかにも複数のセキュリティドメイン統合のモデルが提案されており，非常に柔軟かつ拡張可能な仕様になっている．そのため，WS-Security を適用するにはサービスごとに複雑な設定を正確に行う必要があり，そのセキュリティ設定の複雑さが問題となっている．次節で WS-Security の具体的なセキュリティ設定とその問題点を示す．

### 2.2 旅行予約アプリケーション

WS-Security によるセキュリティ設定の例を示すために，本稿では旅行予約アプリケーションを用いる．**図 2** に示した旅行予約アプリケーションは，旅行代理店（以後，代理店とする）と航空会社の 2 つのサービスからなるコンポジット・アプリケーションである．顧

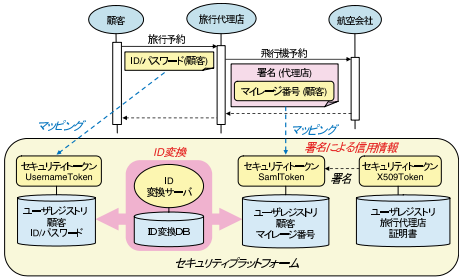


**図 2**　旅行予約アプリケーション
Fig. 2　Travel reservation application.

客が代理店に旅行予約を依頼すると，代理店は顧客の情報を航空会社へ転送し，フライトの予約を依頼する．航空会社はこの顧客に対しフライト予約を行う．ここでは以下のセキュリティ要件が求められているものとする．

- メッセージは署名・暗号化しなければならない．
- 代理店は顧客を ID・パスワードで認証する．
- 航空会社は顧客のマイレージ番号を要求する．
- 航空会社は信頼できる代理店からきた顧客の情報のみを信用する．

すなわち，航空会社は顧客の認証を中間サービスである代理店に委譲し，代理店が認証した顧客の ID をマイレージ番号に変換し伝播されたものを受け取る．このような顧客認証機構を ID 伝播と呼んでいる．航空会社は，信用できる代理店であることを確認する方法として代理店の署名を要求し，伝播された顧客情報を利用する．ここでは，顧客のマイレージ番号は Saml トークンとして要求されており，代理店は顧客の Saml トークンを自分の X509 トークンで署名する必要がある．これらを実現するために必要なセキュリティ技術を持ったプラットフォーム（以後，セキュリティプラットフォームとする）を図 2 に示す．

セキュリティ設定の例を示すために，代理店サービスを WebSphere Application Server（WAS）V6.1[3] で実行したときのアーキテクチャを**図 3** に示す．WS-Security が適用されたメッセージは，受信側 WSS ハンドラで署名や暗号化の検証が行われる．この例では，顧客の ID・パスワードを含む Username トークンが関連付けられたユーザレジストリにより認証される．また，代理店から航空会社へ送信するメッセージは送信側 WSS ハンドラで生成される．代理店は Username トークンを外部の ID 変換サーバに送り，変換された Saml トークンに署名を付加して航空会社に転送する．

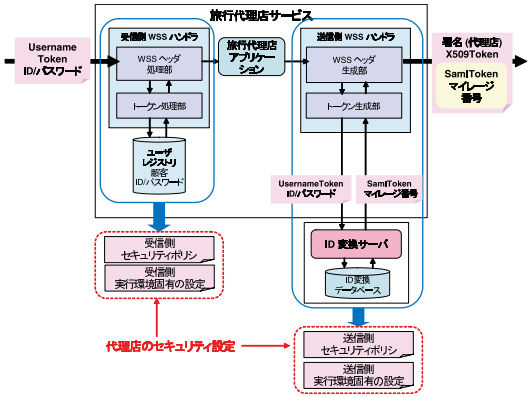


**図 3**　WebSpherer WS-Security ランタイムアーキテクチャ
Fig. 3　WebSphere WS-Security runtime architecture.

WS-Security のためのセキュリティ設定とは，WSS ハンドラが必要とするセキュリティポリシと実行環境固有の設定ファイルである．セキュリティポリシは OASIS で標準化されている WS-SecurityPolicy[4] である．実行環境固有の設定ファイルとは，この例では WAS の設定ファイルである Deployment Descriptor と Binding Configuration という 2 種類のファイルであり，セキュリティポリシをさらに詳細化した情報が記述されている．WS-Security を正しく適用するためには，セキュリティポリシと実行環境固有設定ファイルを正確に記述する必要があるが，問題となるのは設定ファイルの記述形式が複雑であるということではなく，正しい設定をするために必要な知識や情報が複雑かつ非常に多いということである．たとえば，代理店において Username トークンを認証するユーザレジストリを正しく知っておく必要がある．さらに，航空会社が要求するトークンの種類や署名形式などを理解しておかなければならない．また，代理店の実行環境の情報だけではなく，ID 変換サーバの URI や変換可能なセキュリティトークンの形式など外部の情報も必要となる．このように，正しいセキュリティ設定のためには，WS-Security の仕様，複雑なセキュリティプラットフォームの詳細や各サービスの要求などを理解しなければならず，必要な知識が多岐にわたるということが本研究で対象とする問題点である．次に，現状の SOA 開発プロセスにおけるセキュリティ設定の方法とそこでの問題点を示す．

### 2.3　セキュリティ開発の問題点

最初に，セキュリティを考慮しない場合の SOA 開発が，現状ではどのように行われてい

るかを示す．SOA 開発プロセスでは，次のように開発プロセスと開発者を定義できる．

- ビジネス要件分析（ビジネス・アナリスト）
  最終的なビジネス目標から，アプリケーションで実現したいビジネス要求を明確にし，ビジネスプロセスモデルを作成する．
- サービス設計（ソフトウェア・アーキテクト）
  ビジネスプロセスモデルから対応するサービスを設計し，サービスモデルを作成する．
- サービス実装（デベロッパ）
  サービスモデルに従い，単体サービスの実装・テストを行う．
- サービス組み立て（アセンブラ）
  単体サービスを組み合わせてコンポジット・アプリケーションを作成する．
- サービス配置（デプロイヤ）
  アプリケーションをプラットフォームに配置する．

上記の開発プロセスにおいて，WS-Security のセキュリティ設定はサービス組み立てプロセスにおいて行われれることが多い．すなわち，アプリケーションを構成する単体サービスをセキュリティなしで実装した後，サービス組み立てプロセスでアセンブラがサービス間で要求されるセキュリティ要件を考慮し，必要な設定を行う．このように，現状では開発の下流プロセスになって初めてセキュリティ要件が考慮されることが多い．しかし，現状の開発手法では次のような問題点がある．

- セキュリティ設定を誰がどのようなプロセスで行うかが明確ではないため，サービス配置直前まで考慮されず，アセンブラが行わざるをえなくなっている．
- セキュリティ設定そのものが複雑であることに加え，設定を正しく行うために必要な情報が散在しているためアセンブラが知らない情報が必要になる．したがって，必要な情報を得るために，ソフトウェア・アーキテクトやデプロイヤなどに問い合わせる必要が生じてしまう．
- ビジネスレベルでどのようなセキュリティが要求されており，それが正しく反映された設定になっているかどうかが明確でない．

以上の問題点を解決するために，セキュリティ設定の開発プロセスを明確にし，SOA 開発プロセスにあわせたセキュリティ設定プロセスを示す．また，非常に複雑なセキュリティ設定を自動生成し，開発者の負担を軽減するモデル駆動型セキュリティを提案する．

## 3. モデル駆動型セキュリティ

SOA 開発プロセスでは，各プロセスにおいて開発者が得ることのできる情報の種類や粒度が異なり，またそれらが互いに独立であることが特徴である．ビジネス要件分析プロセスでは，ビジネス全体の処理を表現するビジネスプロセスモデルを対象とするのに対し，サービス設計プロセスでは，ビジネスプロセスモデルを構成するコンポジット・サービスが対象となる．さらに下流のプロセスでは，コンポジット・サービスを構成する単体サービスやその実装が対象である．そこで，セキュリティに関しても，各プロセスでどのような情報を扱えるのかという視点に基づき，各プロセスで行うセキュリティ設定の内容を以下のように定義した．

- ビジネス要求分析：必要なビジネスレベルポリシをビジネス要件から定義する．
- サービス設計：各サービスが満たすべきセキュリティ要件をモデル化する．
- サービス組み立て：サービスのセキュリティ要件を実現するためのセキュリティ設定を行う．
- サービス配置：サービスを安全に実行するためのプラットフォームの設定をする．

ビジネス・アナリストは，ビジネス全体の要求を明確にするため，3.1 節で示すようなビジネスレベルのセキュリティポリシを明確にする．ソフトウェア・アーキテクトは，プロセスモデルに従い具体的なサービスモデルを定義する．したがって，ビジネスレベルのセキュリティポリシをサービスモデル上にモデル化し，サービスがどのようなセキュリティ要件を満たす必要があるかを指定するのが妥当である．アセンブラは，サービスに割り当てられたセキュリティ要件を，WS-Security のセキュリティポリシや設定ファイルへ変換し，デプロイヤがそれをプラットフォームにデプロイする．デプロイされたセキュリティポリシや設定が正しく動作するためのプラットフォームの設定はデプロイヤが行う．

ここで，サービス実装プロセスが含まれていないのは，次のような理由からである．WS-Security の場合，セキュリティなしのサービス実装とともにセキュリティ設定をデプロイすることで WSS ハンドラが呼ばれ，セキュアなサービスとして動作する．したがって，サービス実装そのものは WS-Security を適用するかどうかにかかわらず同じである．このためサービス実装を担当するデベロッパには，セキュリティ設定に関する役割はない．

本研究ではこのようなプロセスに基づいたセキュリティ開発のために，抽象的なセキュリティ要件から具体的なセキュリティ設定を自動生成するモデル駆動型セキュリティを提案する．モデル駆動型セキュリティによる WS-Security 設定生成の流れを**図 4** に示す．モデル

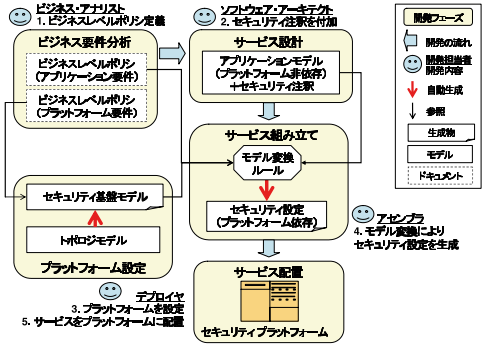


**図 4**　モデル駆動型セキュリティによる開発プロセス
Fig. 4　Development process based Model-Driven Security.

駆動型セキュリティでは，ビジネス・アナリストが定義したビジネスレベルのセキュリティ要件をソフトウェア・アーキテクトがサービスモデル上にセキュリティ注釈として付加し，モデル変換により具体的なセキュリティ設定を生成する．ここで，セキュリティ設定の生成には，デプロイヤが設定したプラットフォームの情報やビジネスレベルで定義されたプラットフォームへの要求内容が必要になる．そこで，プラットフォームの情報をあらかじめモデル化しておくためのセキュリティ基盤モデルを導入する．セキュリティプラットフォームは，WebSphere Application Server V6.1（WAS）[3] など，サービスがデプロイされる実行環境が動作している物理的なサーバ（以後，ノードとする）から構成され，セキュリティ基盤モデルはノードの物理的な接続関係をモデル化したトポロジモデルから半自動生成できる．ビジネスレベルポリシとセキュリティ基盤モデルを参照することで抽象的なセキュリティ要件からモデル変換による具体的なセキュリティ設定の自動生成が可能になる．

### 3.1　ビジネスレベルポリシ

ビジネスレベルポリシとは，ビジネス要件から決まるセキュリティ要件を指しており，Service Level Agreement（SLA）などのドキュメントに記述されていることを想定している．ビジネスレベルポリシは，ビジネス要件からくるアプリケーションのセキュリティ要件を指定するものであるが，アプリケーションの特徴からプラットフォームに対する制限が必要になる場合にはプラットフォームへの要件も記述する．したがって，本研究ではビジネスレベルポリシを以下の 2 種類の要件から構成されると考える．

アプリケーションへの要件

- サービス認証・署名・暗号化の必要性
- 署名・暗号化対象

プラットフォームへの要件

- 鍵の強度（鍵の形式・長さ）
- 署名・暗号化アルゴリズム
- 信頼する認証機関
- 認証に使うユーザレジストリ

プラットフォームへの要件例としては，ユーザの ID・パスワードを扱うアプリケーションでは高い安全性が要求されるため，256 bit AES で暗号化し，Verisign により認証された X509v3 証明書による署名を要求する，などが考えられる．プラットフォームの要件はセキュリティ基盤モデルの生成時に参照され，この要件が反映されたセキュリティ基盤モデルが作られる．さらに，アプリケーションへの要件とセキュリティ基盤モデルを参照して，具体的なセキュリティ設定を自動生成することで，ビジネスレベルポリシを満たすセキュリティ設定を生成することができる．

### 3.2　セキュリティ注釈

ソフトウェア・アーキテクトはサービスに要求されるセキュリティ要件を設定する．WS-Security を適用するには，使われるセキュリティトークンの種類，アルゴリズムや鍵の種類など，サービスがデプロイされるプラットフォームに依存する情報が必要になる．しかし，ソフトウェア・アーキテクトが扱うサービスモデルは単体サービスがデプロイされる環境にかかわらず，どのようなサービスが連携しているかを表すモデルであるため，プラットフォームの情報はモデル化されない．したがって，プラットフォームの詳細情報はソフトウェア・アーキテクトが扱う情報ではなく，デプロイヤが扱う情報であると考えた．このような理由から，ソフトウェア・アーキテクトがサービスモデル上で扱う情報として，抽象的なセキュリティ要件を表現するためのキーワードをセキュリティ注釈として定義した．

本研究では，ソフトウェア・アーキテクトが記述するサービスモデルは “ソフトウェア・サービスのための UML2.0 プロファイル” [5] で記述することを想定している．WS-Security で保証するセキュリティ要件は認証・健全性・完全性である [2] ので，これらをサービスモデル上で指定するためセキュリティ注釈を**表 1** のように定義した．ソフトウェア・アーキテクトは，認証を要求するサービスには ⟨⟨authentication⟩⟩ というステレオタイプを付加し，userType 属性に認証されるべき主体を指定する．これにより，このサービスでは主体を認証するた

**表 1**　セキュリティ注釈
Table 1　Security intents.

| セキュリティ要件 | ステレオタイプ | 属性 |
|---|---|---|
| 認証 | authentication | userType |
| 健全性 | integrity | userType |
| 完全性 | confidentiality | userType |

めの情報が要求されていることを示し，これは WS-Security ではセキュリティトークンが要求されていることを示す．同様に，健全性や完全性を要求するサービスには ⟨⟨integrity⟩⟩ や ⟨⟨confidentiality⟩⟩ というステレオタイプを付加する．これらは，WS-Security ではメッセージの署名や暗号化が必要とされていることを示す．

### 3.3　セキュリティ基盤モデル

WS-Security のセキュリティ設定のためには，ソフトウェア・アーキテクトが付加したセキュリティ注釈に加え，より詳細なプラットフォームの情報などが必要になる．セキュリティプラットフォーム全体の情報は，図 4 の開発プロセスにおいてデプロイヤが記述する，ノードの物理的な接続関係を表したトポロジモデルに対応する．そこで，トポロジモデルの情報から，WS-Security のセキュリティ設定に必要な情報だけを抽出しておくために，セキュリティ基盤モデルを導入した．セキュリティ基盤メタモデルを**図 5** に示す．デプロイヤは，各ノードに WS-Security 用のパラメータ設定を行うことを想定しており，セキュリティ基盤モデルはこのパラメータ設定から半自動生成することができる．ノードの実行環境ごとにパラメータ設定内容や記述方式が異なるため，セキュリティ基盤モデルの生成ルールはノードの実行環境ごとに異なる．しかし，ノードでのパラメータ設定内容とセキュリティ基盤モデルで記述する情報はほぼ同等であるため，セキュリティ基盤モデルの Mapping 要素と TrustMethod 要素以外の要素に関しては単純なモデル間マッピングを定義できる．Mapping 要素と TrustMethod 要素はビジネスレベルポリシのプラットフォームへの要件を参照してデプロイヤが指定し，セキュリティ基盤モデルを完成させる必要がある．以下にセキュリティ基盤モデルの中心的な要素をあげる．

- TokenAssertion 要素は，ノードがサポートしているセキュリティトークンの種類を示す抽象要素である．MachineNode 要素と TokenAssertion 要素間の関連には，セキュリティトークンの使用方法を表す role を指定する．role には，“inboundAuthentication”，“outboundAuthentication”，“inboundSignature” などがある．
- Mapping 要素は，ID 変換により変換される 2 種類のセキュリティトークンを接続する．セキュリティトークンを認証するためのユーザレジストリは，セキュリティトークンごとに決められており，Mapping 要素で接続されたセキュリティトークンのユーザレジストリの ID 変換が可能であることを示す．この要素は，ユーザレジストリの情報をもとにデプロイヤが指定する．
- TrustMethod 要素では，2.2 節で説明した信用方法をデプロイヤが指定する．TrustMethod 要素を持っている TokenAssertion 要素が転送される顧客のセキュリティトークンを示している場合，このトークンをどのような方法で信用するかを指定する．method 要素にはベーシック認証を表す “Basic” または署名を表す “Signature” が指定できる．
- TokenProcessing 要素は，TokenAssertion 要素で指定されたセキュリティトークンの処理に必要なパラメータを指定する．TokenProcessing 要素以下は，ノードの実行環境に依存する部分であり，図 5 は，WebSphere Application Server V6.1[3] をモデル化したものである．

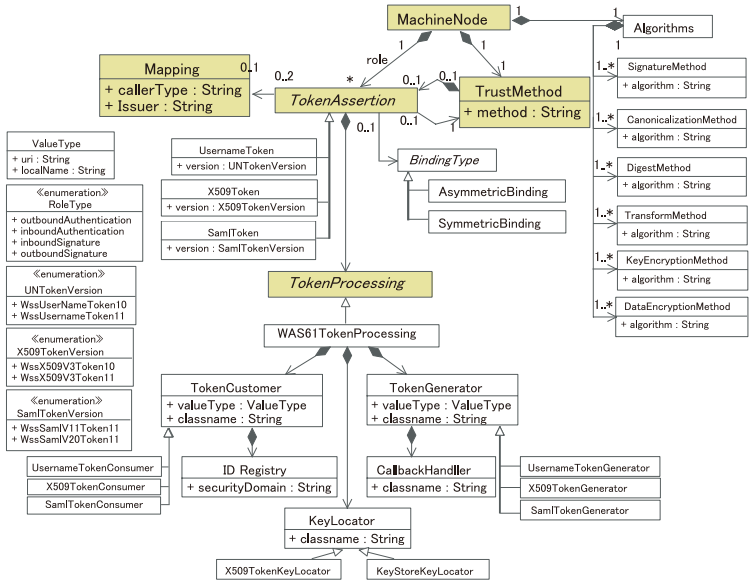


**図 5**　セキュリティ基盤メタモデル
Fig. 5　Security infrastructure meta model.

セキュリティ注釈からセキュリティポリシへのモデル変換において，プラットフォームの

情報を得るために各ノードの設定をそのまま参照するのではなく，セキュリティ基盤モデルという共通モデルを利用することで，ノードの実行環境に依存しないモデル変換ルールを定義することができる．セキュリティポリシからより詳細な設定ファイルへの変換は，ノードの実行環境に依存するが，4.2 節ではこの依存性を下げるための変換方法を示す．

## 4. モデル変換

ここでは，PIM であるセキュリティ注釈から PSM であるセキュリティポリシへのモデル変換と，より詳細な PSM である設定ファイルへのモデル変換の詳細を示す．

WS-Security のセキュリティポリシは，WS-SecurityPolicy [4] で記述されるので，セキュリティ注釈で表されたセキュリティ要件を WS-SecurityPolicy 記述に変換する必要がある．このポリシは，WS-Security をサポートしている WebSphere Application Server V6.1 [3]，Apache Rampart [6] など，複数の実行環境で利用できる．この点で，セキュリティポリシは実行環境の種類に依存しない．しかし，セキュリティポリシで指定した署名や暗号化を実行するには，実行環境ごとに異なるさらに詳細な設定ファイルが必要となる．したがって，セキュリティ注釈からポリシへの変換ルールは実行環境に依存しないが，ポリシから設定ファイルへの変換ルールは実行環境に依存する．

### 4.1 セキュリティ注釈からポリシへの変換

WS-SecurityPolicy では，セキュリティアサーションの組合せによりセキュリティポリシを表現する．セキュリティアサーションとは，署名・暗号化やそれに必要なアルゴリズムなどのセキュリティ要件を表現する XML 要素である．たとえば，署名・暗号化の方法を指定するためには Security Binding Assertion，署名・暗号化の対象を指定するためには，Protection Assertion，セキュリティトークンの種類を指定するためには Token Assertion が定義されている．

セキュリティポリシを生成するには，セキュリティ注釈から対応するセキュリティアサーションを適切に組み合わせる必要がある．セキュリティアサーションは，それぞれ並列に組み合わされるだけでなく，Security Binding Assertion の中に Token Assertion が指定されるなど入れ子構造をとるものもある．したがって，セキュリティ注釈とセキュリティアサーションの対応は単純なマッピングとして定義できないため，テンプレートを使ったモデル変換を定義した．

**図 6** にテンプレートベースのモデル変換を示す．セキュリティポリシテンプレートにおいて，他のテンプレートが指定されるべき部分は $[BindingAssertionType] のようにパラメータとして表されている．各セキュリティアサーションにもテンプレートが定義されており，$[BindingAssertionType] には，バインディングアサーションテンプレートにパラメータがセットされたものが入る．さらに，セキュリティアサーションのテンプレートも入れ子構造を持ち，バインディングアサーションテンプレートはパラメータとして $[TokenAssertionType] を持っている．

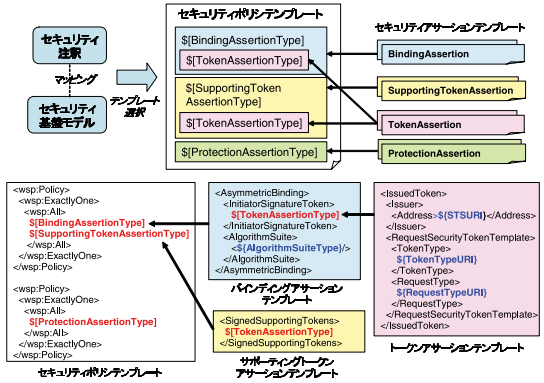


**図 6**　テンプレートベースのモデル変換
Fig. 6　Template-based model transformation.

セキュリティ注釈からセキュリティポリシへ変換するには，セキュリティ基盤モデルを参照し，対応する要素を参照して得られた値をテンプレートパラメータにセットする．テンプレートパラメータと対応するセキュリティ基盤モデルの要素は決まっており，この対応関係もモデル変換ルールとして実装されている．テンプレートは図 6 に示した 5 種類があるが，ここでは紙面の都合上その一部を示す．セキュリティ注釈として ⟨⟨authentication⟩⟩ が指定された場合，セキュリティポリシテンプレートの $[SupportingTokenAssertionType] パラメータを埋める必要がある．図 6 のサポーティングトークンアサーションテンプレートは，$[TokenAssertonType] パラメータを指定する必要があり，ここに認証で使われるセキュリティトークンの種類を指定する．認証に使われるセキュリティトークンの種類は，セキュリティ基盤モデルを参照すれば得られ，すなわち，authentication の role を持つセキュリティトークンが該当する．

同様に，⟨⟨integrity⟩⟩ に対しては，$[BindingAssertionType] と $[ProtectionAssertionType] を埋める必要がある．ここで，${AlgorithmSuiteType} はアサーションテンプレー

トではなく，アルゴリズムスイート名が入れられる．セキュリティ基盤モデルを参照すれば，プラットフォームがサポートしているアルゴリズムの種類が分かるので，対応するアルゴリズムスイート名が指定される．

このように，どのテンプレートが必要かということをセキュリティ注釈から決定し，テンプレートのパラメータの値をサービスがデプロイされるノードのセキュリティ基盤モデルから決定することで，モデル変換が可能になる．

### 4.2　ポリシから設定ファイルへの変換

セキュリティポリシからより詳細な実行環境依存の設定ファイルへのモデル変換では，実行環境ごとに変換ルールが異なるため，ポリシアサーションと設定ファイル内の対応する要素とのマッピングを考える必要がある．ただし，ここで問題となるのは，セキュリティポリシと設定ファイルの記述方法の違いである．WS-Security を適用したメッセージにおいて複数の署名や暗号化が行われている場合，これらの署名・暗号化はそれぞれ区別されず同等に扱われるため，実行環境依存の設定ファイルの記述では複数の署名は区別なく列挙される．しかし，セキュリティポリシでは SOAP Body への署名・暗号をメッセージ署名・暗号と定義し，その他の署名・暗号（たとえば，セキュリティトークンやメッセージ署名をさらに署名・暗号化したものなど）を追加署名と定義し区別をして扱う．すなわち，メッセージ署名・暗号とは SOAP Body に入っているアプリケーションのパラメータに対するセキュリティ要件であり，追加署名・暗号はセキュリティ強度を高めるために追加するセキュリティ要件であるという意味付けがなされている．そのため，設定ファイル上では同等に扱われる署名が，セキュリティポリシでは異なるポリシアサーションで記述されており，セキュリティポリシアサーションと設定ファイルの要素は 1 対 1 に対応しない．そのため，セキュリティポリシから設定ファイルへの変換ルールを考えると，複数のポリシアサーションの組合せに応じて設定ファイルの 1 つの要素が決定するという複雑なルール集合になってしまう．また，変換先の設定ファイルは，WebSphere Application Server V6.1[3]，Apache WSS4J[7] や Rampart[6] など実行環境の数だけ存在する．これらの設定ファイルへの変換ルールはすべて異なるため，アプリケーションが動作する実行環境が変更されるとセキュリティポリシから設定ファイルへの複雑な変換ルールをすべて修正する必要がある．ここでは紙面の都合上詳細は述べないが，詳しい議論は文献 8) で行っている．

この複雑かつ実行環境に依存したモデル変換ルールを扱いやすくするために，中間モデルを定義しセキュリティポリシから設定ファイルへの 2 段階変換を導入した（**図 7**）．セキュリティポリシと設定ファイルではセキュリティ要件の意味付けが異なるものの WS-Security

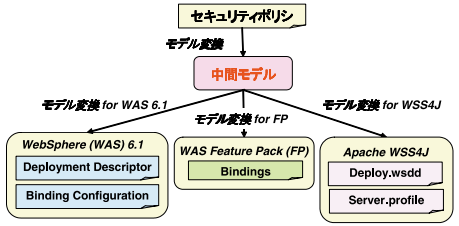


**図 7**　ポリシから設定ファイルへのモデル変換
Fig. 7　Model transformation into configurations.

を適用した場合のメッセージ構造を記述することを目的としている点では共通である．したがって，セキュリティポリシとメッセージ構造，また設定ファイルとメッセージ構造のマッピングは容易であるため，このメッセージ構造を反映した中間モデルを定義した．これにより，セキュリティポリシから設定ファイルへの複雑な直接変換ルールを記述する必要がなくなり，セキュリティポリシからメッセージ構造，メッセージ構造から設定ファイルへの変換ルールだけを考えればよくなる．モデル変換には 2 種類の変換ルールが必要になるがメッセージ構造との変換ルールのほうが単純になるため，モデル変換の実装が容易になる．

セキュリティポリシから設定ファイルへの変換では，セキュリティポリシに書かれているセキュリティ要件の記述を設定ファイルでの記述形式に変換するのに加え，プラットフォームで実行するために必要なバインディング情報を記述する必要がある．たとえば，WAS V6.1 の場合，セキュリティポリシと同等の内容は Deployment Descriptor（DD）に記述され，DD に対応するバインディング情報は Binding Configuration（Binding）に記述される．バインディング情報は，セキュリティ基盤モデルの TokenProcessing 要素以下の要素から得ることができ，セキュリティ基盤モデルから得られた情報が Binding に変換される．これら 2 つのファイルが正しく生成されてはじめて，WS-Security を正しく適用したアプリケーションが動作可能になる．

中間モデルを介した変換ルールの単純化による利点は，複数の実行環境に対応したモデル変換を実装する場合のコストを軽減することができるということである．図 7 に示したセキュリティポリシから中間モデルへの変換ルールは実行環境非依存になるため，モデル変換ルール全体としては実行環境依存の部分が半分になる．したがって，新しい実行環境が追加されても，図 7 の下半分のモデル変換ルールだけを変更すれば対応できる．

WAS V6.1 の場合について，セキュリティポリシから設定ファイルへ直接変換する場合

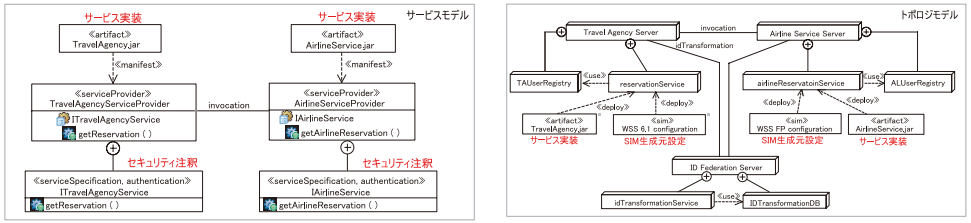


**図 8**　旅行予約アプリケーションのサービス，トポロジモデル
Fig. 8　Service and topology model for travel reservation application.

と，図 7 の 2 段階変換の場合について実装コストの比較を行った．コード行数は 2 段階変換の場合が直接変換の約 1.5 倍になるが，変換ルールそのものは単純になり実装も容易になる．また，Apache WSS4J [7] の設定ファイルへの変換を追加実装する場合にはコード行数は直接変換の約半分で済むことが分かった．

以上のことから，モデル変換ルールの実行環境に対する依存性が弱まるということができ，1 つのセキュリティポリシから異なる実行環境の設定ファイルへの変換が容易になる．これは，実行環境の設定ファイルは WS-Security のメッセージ構造を反映した形式で記述されているという特徴があるため，その特徴を生かした中間モデルの導入により変換ルールを単純化できた結果であり，複数の実行環境が異なる設定ファイルを必要とする現状では重要な利点であるといえる．

## 5. 適　用　例

3 章で提案したモデル駆動型セキュリティに基づきセキュリティ設定を行うためのツールのプロトタイプ実装を行い，2.2 節の旅行予約アプリケーションに適用した例を示す．

プロトタイプの開発環境としてモデル駆動型開発のプラットフォームである Rational Software Architect（RSA）V7.0 [9] を用い，RSA が提供しているモデル変換フレームワークを利用した．RSA 7.0 では，“サービスモデルのための UML2.0 プロファイル” [5] が提供されており，UML のクラス図でサービスをモデル化できる．そこで，このサービスモデルにセキュリティ注釈をつけるために，3.2 節で定義したステレオタイプと属性を WS-Security プロファイルとして実装した．また，文献 5) のサービスプロファイルでは，サービスがデプロイされるプラットフォームのトポロジモデルは考慮されていないため，デプロイヤが記述するトポロジモデルを UML のデプロイメント図で記述することとした．トポロジモデルには，各ノードに設定されている WS-Security 用のパラメータ設定ファイルがモデル化され，ここからセキュリティ基盤モデルを生成する．そこで，このパラメータ設定ファイルを示すためのステレオタイプとして ⟨⟨sim⟩⟩ を定義し，WS-Security プロファイルに追加した．

旅行予約アプリケーションでのサービスモデルとトポロジモデルを**図 8** に示す．サービスモデルは，文献 5) に基づいた代理店サービスと航空会社サービスのモデルである．ここで，TravelAgency.jar と AirlineService.jar はサービス実装を表す．ソフトウェア・アーキテクトはサービスモデルを記述し，ビジネスレベルのセキュリティ要件に従って，各サービスが要求するセキュリティ注釈を付加する．ここでは，TravelAgencyService と AirlineService に認証のためのセキュリティ注釈 ⟨⟨authentication⟩⟩ が指定されている．

デプロイヤが記述するトポロジモデルには，サービスがデプロイされるノードがモデル化される．デプロイヤが TravelAgency.jar と AirlineService.jar をデプロイするノードを決定すると，このデプロイ関係からサービスモデルとトポロジモデルが紐付けられることになる．デプロイヤが行う WS-Security のために必要なパラメータ設定は，図 8 の reservationService では “WSS 6.1 configuration” とモデル化されている．セキュリティ基盤モデルはこの設定から生成されるため，WSS 6.1 configuration にはステレオタイプ ⟨⟨sim⟩⟩ を指定する．ここから生成される代理店のセキュリティ基盤モデルの一部を**図 9** に示す．セキュリティ基盤モデルには，実行環境でサポートされているトークンとして，受信側（inboundAuthentication）には UsernameToken，送信側（outboundAuthentication）には SamlToken がモデル化されている．航空会社へは UsernameToken で受信した顧客の情報を SamlToken に変換して送る必要があるため，これらのトークンは Mapping 要素で紐付けられ，ID 変換が必要であることを示している．このようにサービスモデルとトポロジモデルが紐付けられた状態で，アセンブラがサービス組み立てプロセスにおいて ITravelAgencyService を選択しモデル変

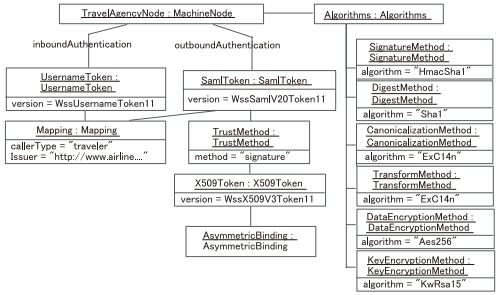


**図 9**　旅行代理店のセキュリティ基盤モデル
Fig. 9　Security infrastructure model for travel agency.

換を実行すると，セキュリティ注釈とセキュリティ基盤モデルから対応するセキュリティポリシが生成される．ここで重要な点は，サービスモデルにおいて代理店サービスと航空会社サービスに同じセキュリティ注釈をつけることができることである．実際の WS-Security での認証に必要なトークンは両サービスで異なるが，サービスモデル上では両サービスが顧客認証を必要としていることだけを同じセキュリティ注釈でモデル化することができる．実際のトークンの種類などの詳細設定は，デプロイヤが作成するトポロジモデルから作られるセキュリティ基盤モデルから得られ，ソフトウェア・アーキテクトはビジネス要求を満たすセキュリティ要件を指定することだけを行えばよいというのが，本研究の SOA のセキュリティ開発プロセスの利点である．

今までのセキュリティ開発では，デプロイヤが設定するトポロジの情報とそのトポロジ上で動作するアプリケーションの設定は別々に扱われ連携していなかった．そのため，セキュリティ設定はトポロジの情報から決まる部分が多くあるにもかかわらず，アセンブラが一から手で記述しなければならない状況になっていた．そこで，トポロジの情報から必要な情報を抽出し，不足している情報はデプロイヤやアセンブラが補ったうえで，セキュリティ設定を自動生成する仕組みを提案した．トポロジの情報から，セキュリティ設定に必要な情報を抽出してモデル化したものがセキュリティ基盤モデルである．セキュリティ基盤モデルの導入により，アセンブラとデプロイヤがビジネスレベルからの要求をモデル化すれば，アセンブラがすべて手で記述していたセキュリティ設定は自動生成できるようになった．

現状でも，セキュリティポリシや設定ファイルを書くエディタは存在する[10]が，アセンブラはセキュリティ設定内容を詳細に理解していないと利用することができない．しかし，WS-Security のセキュリティ設定の内容は，ビジネスレベルポリシで決定される内容から，サービスがデプロイされるプラットフォームの情報まで広範囲にわたっている．また，旅行予約アプリケーションで使われる設定ファイルを現在のエディタで記述するには，アセンブラは 50 以上のパラメータの値を手で指定する必要がある．一方，モデル駆動型セキュリティによるセキュリティ開発プロセスでは，アセンブラはサービスモデルにセキュリティ注釈を付加するだけでよく，旅行予約アプリケーションの例では 2 つのステレオタイプを追加するだけでよい．

デプロイヤにはトポロジモデルを記述する必要が生じるため，UML の配置図を記述するスキルは必要になるが，トポロジモデルの内容はノードの接続関係とノードの WS-Security パラメータ設定ファイルの場所だけであるため，スキルの習得は困難ではないと考えている．ノードの WS-Security のパラメータ設定はもともとデプロイヤが作成するものであり，これをモデル上に指定することにも問題はない．さらに，デプロイヤにはセキュリティ基盤モデルの生成も要求されるが，これはトポロジモデルから大部分が自動生成される．3.3 節で示したように，デプロイヤが手で指定しなければならない要素は 2 つだけであるので，大きな負担にはならないと考える．以上のことから，サービスモデルとトポロジモデルで定義された情報をつなぎ，モデル変換によりセキュリティ設定を自動生成することで，複雑な設定を手で記述する負担を大きく軽減することができるといえる．

このプロトタイプ実装では，ビジネスレベルポリシとセキュリティ注釈はどのように対応付けられるかは，ソフトウェア・アーキテクトが判断することを想定している．今後はビジネスレベルポリシとセキュリティ注釈の対応付けを自動的に行う方法を考えたい．たとえば，ビジネス・アナリストが使うビジネス要件分析のためのツールとして文献 11) などがあげられるが，ここにセキュリティポリシ定義のための機能を追加するなどが考えられる．デプロイヤの負担を軽減するためには，セキュリティドメイン統合のためのトポロジモデルパターンの導入が考えられる．デプロイヤは一からトポロジモデルを作成する代わりに，パターンを選択・修正することでトポロジモデルが作成できるため，デプロイヤの負担を軽減することが可能になると考えている．さらに，モデル駆動型開発の特徴としてモデル間のトレーサビリティがあげられる．これはセキュリティ設定においても非常に重要であり，セキュリティ基盤モデルからどのようなセキュリティ要件が実現できるか，といった検証が可能になる．また，実際のアプリケーション開発では，ビジネス要求からトップダウンにセキュリティ設定を導く開発手法だけではなく，すでにある実行環境のセキュリティ設定からボトムアップに実現できるセキュリティ要件を定義することも要求される．今後はこのよう

な点を含めて，モデル間の変換ルールを向上させていきたい．

## 6. 関 連 研 究

本研究は文献 8)，12)，13) の研究内容を発展させたものである．文献 12) ではモデル駆動型セキュリティの概要を提案し，文献 13) ではそれを複雑な認証設定を行う場合について議論した．文献 8) は，セキュリティポリシから詳細な設定ファイルを生成する際のモデル変換について議論した．本稿では，モデル駆動型セキュリティを SOA 開発プロセスでどのように適用するのかを示し，SOA のセキュリティ開発プロセスを明確にした．さらに，WS-Security の 3 つのセキュリティ要件とモデル変換の詳細を議論した．

アプリケーションのセキュリティ設定にモデル駆動型開発を適用する研究は今までにも行われてきている．SecureUML[14),15)] は，アクセス制御の設定をモデル駆動型開発で行っている．認証やアクセス制御の設定にフォーカスしたものは，ほかに文献 16)，17) などがある．また，UMLSec[18)–20)] では，分散システム環境における UML のセキュリティ拡張を提案し，その検証も行っている[21)]．文献 22) は，Web サービスのセキュリティに関してモデル駆動型開発を適用しているが，RBAC の設定から XACML を生成することを目的としており，WS-Security を対象としている本研究とは明確に異なる．文献 23) は，WS-Security の設定方法として Web サービスの非機能要件をモデル駆動で扱う手法を提案しており，WS-SecurityPolicy も対象としている．しかし，非常に単純なポリシのみを考慮しており，本研究のセキュリティドメインの統合のような複雑な例については考慮していない．また，文献 23) で PSM として出力するのは，WS-SecurityPolicy であるが，本研究では WS-SecurityPolicy と同時により詳細な実行環境固有の設定ファイルを対象としている．実際に WS-SecurityPolicy で規定されたセキュリティを適用するには，このような設定ファイルを生成することが不可欠である．本研究では，アプリケーションモデルとトポロジモデルを関連づけてモデル変換を適用することで，セキュリティポリシだけではなく対象とするプラットフォームに適した設定ファイルまでをモデル変換の対象としたことは，本研究の貢献であるといえる．

## 7. ま と め

SOA アプリケーションのセキュリティ開発では，開発の下流プロセスになってセキュリティ要件を考慮することが多く，必要な情報が分散してしまっているため設定が難しい．したがって，一部の開発者に負担がかかるという問題があった．そこで，SOA アプリケーションの開発プロセスに基づいたセキュリティ開発プロセスを定義し，各開発プロセスでの役割を明確にした．これにより開発者に負担をかけることなく開発の上流プロセスからセキュリティ要件を考慮することができる．上流モデルでセキュリティ要件を指定するためのセキュリティ注釈と，プラットフォームの情報をモデル化するセキュリティ基盤モデルを導入し，モデル駆動型開発の適用により実行環境に適したセキュリティ設定を自動生成する手法を提案した．また，モデル駆動型セキュリティを実現するためのツールをプロトタイプ実装し，提案するセキュリティ開発プロセスの効果を確認した．今後はより上流プロセスからのセキュリティ設定の自動生成やトレーサビリティの実現を目指していきたい．

## 参 考 文 献


1) OASIS: Web Services Security: SOAP Message Security 1.1. available from http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf (accessed 2008-04-16).
2) IBM and Microsoft: Security in a Web Services World: A Proposed Architecture and Roadmap. available from http://www.ibm.com/developerworks/library/specification/ws-secmap/ (accessed 2008-04-16).
3) IBM: WebSphere Application Server V6.1. available from http://www.ibm.com/software/webservers/appserv/was (accessed 2008-04-16).
4) OASIS: WS-SecurityPolicy 1.2. available from http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.doc (accessed 2008-04-16).
5) Johnston, S.: UML 2.0 Profile for Software Services. available from http://www.ibm.com/developerworks/rational/library/05/419_soa/ (accessed 2008-04-16).
6) Apache: Securing SOAP Messages with Rampart. available from http://ws.apache.org/axis2/modules/rampart/1_3/security-module.html (accessed 2008-04-16).
7) Apache: WSS4J. available from http://ws.apache.org/wss4j/ (accessed 2008-04-16).
8) Satoh, F. and Yamaguchi, Y.: Generic Security Policy Transformation Framework for WS-Security, *Proc. International Conference on Web Services*, pp.513–520 (2007).
9) IBM: Rational Software Architect. available from http://www.ibm.com/software/awdtools/architect/swarchitect/index.html (accessed 2008-04-16).
10) IBM: Rational Application Developer for WebSphere Software. available from http://www.ibm.com/software/awdtools/developer/application/index.html

(accessed 2008-04-16).
11) IBM: WebSphere Business Modeler. available from http://www.ibm.com/software/integration/wbimodeler/ (accessed 2008-04-16).
12) Nakamura, Y., Tatsubori, M., Imamura, T. and Ono, K.: Model-Driven Security Based on a Web Services Security Architecture, *Proc. IEEE International Conference on Services Computing*, pp.7–15 (2005).
13) Satoh, F., Nakamura, Y. and Ono, K.: Adding Authentication to Model Driven Security, *Proc. International Conference on Web Services*, pp.585–594 (2006).
14) Lodderstedt, T., Basin, D. and Doser, J.: SecureUML: A UML-Based Modeling Language for Model-Driven Security, *Proc. International Conference on The Unified Modeling Language*, pp.426–441 (2002).
15) Basin, D., Doser, J. and Lodderstedt, T.: Model Driven Security for Process-Oriented Systems, *Proc. ACM Symposium on Access Control Models and Technologies*, pp.100–109 (2003).
16) Fink, T., Koch, M. and Pauls, K.: An MDA approach to Access Control Specifications Using MOF and UML Profiles, *Proc. International Workshop on Views On Designing Complex Architectures*, pp.161–179 (2004).
17) Burt, C.C., Bryant, B.R., Raje, R.R., et al.: Model Driven Security: Unification of Authorization Models for Fine-Grain Access Control, *Proc. International Enterprise Distributed Object Computing Conference*, pp.159–171 (2003).
18) Jürjens, J.: UMLsec: Extending UML for Secure Systems Development, *Proc. International Conference on The Unified Modeling Language*, pp.412–425 (2002).
19) Jürjens, J.: Sound Methods and Effective Tools for Model-based Security Engineering with UML, *Proc. International Conference on Software Engineering*, pp.322–331 (2005).
20) Jürjens, J. and Fox, J.: Tools for model-based security engineering, *Proc. International Conference on Software Engineering*, pp.819–822 (2006).
21) Deubler, M., Grünbauer, J., Jürjens, J. and Wimmel, G.: Sound Development of Secure Service-based Systems, *Proc. International Conference on Service Oriented Computing*, pp.115–124 (2004).
22) Alam, M.M., Breu, R. and Breu, M.: Model driven security for Web services (MDS4WS), *Proc. International of Multitopic Conference*, pp.498–505 (2004).
23) Ortiz, G. and Hernandez, J.: Toward UML Profiles for Web Services and their Extra-Functional Properties, *Proc. International Conference on Web Services*, pp.889–892 (2006).

(平成 19 年 10 月 18 日受付)
(平成 20 年 4 月 8 日採録)

**佐藤　史子**（正会員）
2001 年東京工業大学大学院理工学研究科基礎物理学専攻修士課程修了．同年日本アイ・ビー・エム（株）入社．東京基礎研究所にて，ビデオオーサリング，モバイル Web サービス，Web サービスセキュリティに関する研究に従事．

**中村　祐一**（正会員）
1990 年大阪大学大学院工学研究科応用物理学専攻博士課程修了．同年日本アイ・ビー・エム（株）入社．東京基礎研究所にて，知識の再利用，オブジェクト指向プログラムの視覚化，移動エージェント，Web サービスに関する研究に従事．2008 年 1 月より，サービス事業部に勤務．工学博士．

**小野　康一**（正会員）
1994 年早稲田大学大学院理工学研究科後期博士課程単位取得退学．同年日本アイ・ビー・エム株式会社東京基礎研究所．移動エージェント技術，Web/XML アプリケーション開発支援技術，プログラム解析技術，モデル駆動開発技術，ソフトウェア検証技術等の研究に従事．日本ソフトウェア科学会，IEEE-CS，ACM 各会員．