サービス合成におけるセキュリティポリシーの抽象化による整合性の検証

Verifying the Consistency of Security Policies by Abstraction for Web Service Composition

小野 康一† 中村 祐一† 佐藤 史子† 立石 孝彰†
Kouichi ONO Yuichi NAKAMURA Fumiko SATOH Takaaki TATEISHI

†日本アイ・ビー・エム株式会社 東京基礎研究所
Tokyo Research Laboratory, IBM Research
{onono,nakamury,sfumiko,tate}@jp.ibm.com

既存サービスの組合せによるアプリケーション合成では，合成可能性の判断が難しい．合成可能性は，機能的性質の整合性だけではなく，非機能的性質の整合性も判断する必要があるからである．ここでは，非機能的性質としてメッセージ防護に関するセキュリティポリシーを対象とする．本稿では，セキュリティポリシーを抽象化して security qualifier を求め，それを用いて情報フロー解析することで整合性を検証する手法を提案する．また，コンプライアンスルールに対しても，同様の方法によって整合性を検証する．

1 はじめに

Service Oriented Architecture (SOA) に基づくソフトウェア開発が注目されている [1]．SOA によってインフラストラクチャの技術に依存せずにアプリケーションソフトウェアを開発することができる．その一方で，ネットワークの発達により，アプリケーションソフトウェアが稼動する環境はより複雑化しつつある．このため，セキュリティなどの非機能的側面の設定には，その複雑な環境の深い理解が必要となる．したがって，ソフトウェア開発工程の初期段階からセキュリティは考慮されるべきであり，セキュリティ・エンジニアリング [2][3] は重要である．

Component Business Modeling (CBM) などのボトムアップ開発手法を用いて，求めるアプリケーションソフトウェアを，既存のサービスコンポーネントの組合せによって構築できるかどうかを判断するのは難しい．既存のサービスコンポーネント間の機能的な整合性だけでなく，セキュリティのような非機能的な整合性も検証される必要があるからである．個々のサービスは，サービス間で交換するメッセージの防護についてのセキュリティポリシーをあらかじめ与えられている．セキュリティポリシーは，Web サービスセキュリティポリシー言語 (WS-SecurityPolicy) [4] で記述されるが，セキュリティポリシーの記述をそのまま整合性検証するのは困難である．

この論文では，セキュリティポリシーから，それを抽象化した security qualifier を得て，それを元に整合性を検証する方法を提案する．抽象化の方法はあらかじめセキュリティ設計者が与えると仮定するが，その抽象化方法は，WS-SecurityPolicy 言語仕様で定義している暗号の強度などにしたがった制約を満たす必要がある．本稿ではその制約がどのようになるかについても述べる．プロセスフローを用いて，security qualifier をメッセージ防護のセキュリティレベルとみなし，型検査にもとづく情報フロー解析によって，セキュリティポリシーの整合性を検証する．

また，サービスを利用する組織においては，情報の利用などに関して組織内で遵守すべき規則，いわゆるコンプライアンスルール (Compliance Rule) が定められていることがある．本手法では，このコンプライアンスルールに対しても，同様の方法によって整合性を検証する．すなわち，コンプライアンスルールのうち，組織内で取り扱う情報についての遵守規則をもとに，サービスで交換されるデータについてのセキュリティ要件/ポリシーを取り出して，整合性の検証に用いる．

2 節では SOA とセキュリティポリシーについて述べる．3 節ではセキュリティポリシーの抽象化による整合性検証手法について述べる．4 節では関連研究について触れ，5 節でまとめる．

2 SOA とセキュリティ

2.1 Service-Oriented Architecture

本稿では，SOA に基づくアプリケーションソフトウェア開発として，ボトムアップ開発手法を想定する．CBM などのボトムアップ開発手法では，既存のサービスコンポーネントを組み合わせてアプリケーションソフトウェアを開発する．以下では，図 1 に示す例を用いる．この例は，Web Services Interoperability Organization (WS-I) によって定義されているサプライチェインマネージメント (SCM) のアプリケーションである [5]．この SCM アプリケーションは，3 つのサブシステムから構成されている: デモシステム，リテイラーシステム，マニュファクチャラーシステムである．また，この SCM アプリケーションは，いくつかの既存のサービスを組み合わせている: ロギングサービス，リテイラーサービス，ウェアハウスサービス，ウェアハウスコールバックサービス，マニュファクチャラーサービスである．

顧客 (Customer) から小売業者のリテイラーサービスに商品が注文されて，リテイラーサービスは卸業者のウェアハウスサービスに在庫を問い合わせる．ウェアハウスに当該商品の在庫があれば出荷可能の返信を行ない，在庫がなければ生産業者のマニュファクチャラーサービスに発注する．

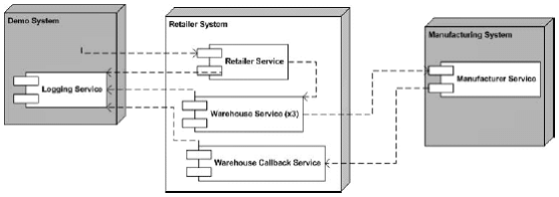

図 1: サプライチェインマネージメント (WS-I)

2.2 Security Policy

個々のサービスについて，必要とされるメッセージ防護のためのセキュリティポリシーが，図 2 に示すように，個々のサービスにあらかじめ与えられていると仮定する．セキュリティポリシーは，そのサービスが交換するメッセージがどのように防護されるかについて，定義する．主に，暗号技術や署名技術を用いたメッセージ防護のポリシーが，WS-SecurityPolicy によって記述される．暗号技術はメッセージの秘匿性 (confidentiality) に対応し，署名技術はメッセージの健全性 (integrity) に対応している．

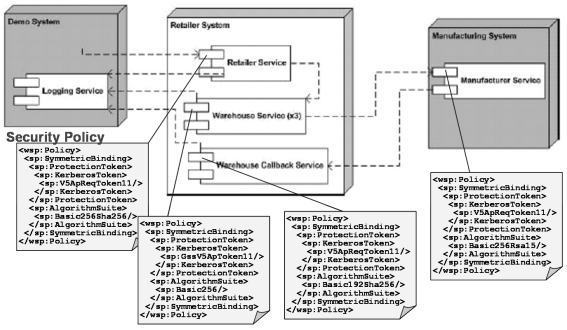

図 2: SCM とセキュリティポリシー

サービスの組合せでは，そのサービス間で交換されるメッセージのどのようなデータがどのように処理されるかのフローを定義する．したがって，サービスの組合せでは，サービス間で交換されるメッセージ中のデータの保護についてのセキュリティポリシーの整合性を検証する必要がある．たとえば，SCM アプリケーションのリテイラーサービスにおいて，注文情報が強い暗号によって暗号化されるというセキュリティポリシーが記述されていると仮定する．同様に，ウェアハウスサービスにおいて，注文情報が弱い暗号によって暗号化されるというセキュリティポリシーが記述されていると仮定する．リテイラーサービスへ発注するリクエスタは，リテイラーサービスのセキュリティポリシーにしたがい，注文情報を強い暗号で保護して送る必要がある．しかしながら，リテイラーサービスからウェアハウスサービスへ注文情報を送る場合，ウェアハウスサービスのセキュリティポリシーにしたがい，その注文情報は弱い暗号で保護される．この時，リテイラーサービスに与えられたセキュリティポリシーが要請している強い暗号による保護は満たされないことになる．つまり，リテイラーサービスのセキュリティポリシーと，ウェアハウスサービスのセキュリティポリシーが整合していないことになる．CBM などのボトムアップ開発では，既存のサービスがサービスコンポーネントとして，あらかじめセキュリティポリシーを与えられている．したがって，あるサービスの組合せにおいて，セキュリティポリシー間の整合性を開発時点で検証することが可能であることになる．

3 セキュリティポリシーの整合性検証

この節では，個々の Web サービスに与えられたセキュリティポリシー間の整合性を検証する方法について述べる．基本的なアイデアは，秘匿性や健全性に関するセキュリティレベルを表す security qualifier を考え，セキュリティポリシーから security qualifier へのマッピングをあらかじめ定義しておき，個々のセキュリティポリシーをそのマッピングにしたがって security qualifier へ変換した上で，サービスの個々の機能を定義するプロセスフローを解析して，security qualifier の整合性を検証する，というものである．security qualifier は一種のセキュリティ型を表し，秘匿性や健全性について，それぞれセキュリティレベルを持つ．それぞれのセキュリティレベルは全順序 (total order)，もしくは束 (lattice) になっているとする．したがって，秘匿性のセキュリティレベルと健全性のセキュリティレベルの組である security qualifier は束になる．本稿では，セキュリティポリシーから security qualifier への変換を「セキュリティポリシーの抽象化」と呼ぶ．図 3 は，図 2 に示した SCM アプリケーションの個々のサービスに与えられたセキュリティポリシーの抽象化を表している．図 3 において，“high”, “middle”, “strong”, “normal” は，それぞれ，セキュリティレベルを表している．さらに，セキュリティレベルから security qualifier へのマッピングを「セキュリティポリシーの抽象化方法」と呼ぶ．

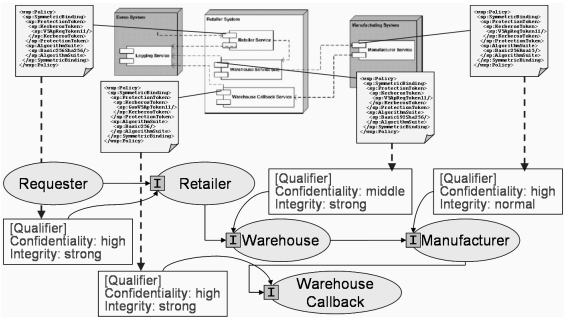

図 3: セキュリティポリシーの抽象化

このようにして得られる security qualifier の整合性は，プロセスフローに対して情報フロー解析技術 [6][7] を応用することで検証することができる．本稿では，プロセスフローを BPEL4WS (以降では BPEL と省略する) [8][9] で記述すると仮定する．プロセスフローに記述されている個々の操作のパラメタ変数は，security qualifier によるセキュリティ型が型付けされている．同様に，外部のサービス呼出で使用される変数も security qualifier によるセキュリティ型が型付けされている．プロセスフローを用いた情報フロー解析は，それらの security qualifier による型付けを元にした型検査により，静的に行なわれる．

図 4 に本手法のシステム構成を示す．セキュリティ設計者によって，前もって抽象化手法が定義されている必要がある．抽象化手法は，WS-SecurityPolicy のセマンティックスに基づいて定められる制約を満たすように定義されなければならない．

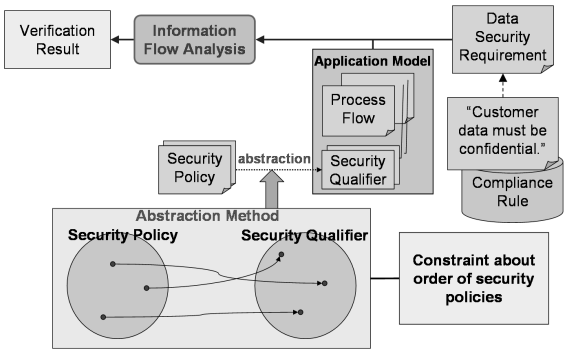

図 4: システム構成

3.1 抽象化手法と制約

WS-SecurityPolicy の言語仕様は，セキュリティポリシーの構成要素である security qualifier の構文とセマンティックスを定義している．たとえば，`<sp:AlgorithmSuite>` assertion は，対象とする SOAP メッセージの要素やパートに適用されるアルゴリズムスイートを指定する．`<sp:AlgorithmSuite>` assertion には，適用するアルゴリズムスイートの名前を記述する．アルゴリズムスイートの名前は，ディジタル署名のためのメッセージダイジェスト関数や，暗号，対称鍵/非対称鍵の key wrap アルゴリズム，などの組合せを表している．これらのアルゴリズムは本質的な強度の違いがあり，このため，任意のアルゴリズムスイートの間には，秘匿性および健全性について，それぞれ，順序関係 (order) が存在しうる．表 1 に WS-SecurityPolicy 仕様で定義されているアルゴリズムスイートのリストを示す (仕様から抜粋)．

これらのアルゴリズムは，メッセージ防護の強度に影響を与える．メッセージダイジェスト関数は健全性のセキュリティレベルに影響を与え，暗号は秘匿性のセキュリティレベルに影響を与え，key

表 1: アルゴリズムスイート (抜粋)

Algorithm	Dig	Enc	Sym KW	Asym KW
Basic256	Sha1	Aes256	KwAes256	KwRsaOaep
Basic192	Sha1	Aes192	KwAes192	KwRsaOaep
Basic128	Sha1	Aes128	KwAes128	KwRsaOaep
TripleDes	Sha1	TripleDes	KwTripleDes	KwRsaOaep
Basic256Rsa15	Sha1	Aes256	KwAes256	KwRsa15
Basic192Rsa15	Sha1	Aes192	KwAes192	KwRsa15
Basic128Rsa15	Sha1	Aes128	KwAes128	KwRsa15
TripleDesRsa15	Sha1	TripleDes	KwTripleDes	KwRsa15
Basic256Sha256	Sha256	Aes256	KwAes256	KwRsaOaep
Basic192Sha256	Sha256	Aes192	KwAes192	KwRsaOaep
Basic128Sha256	Sha256	Aes128	KwAes128	KwRsaOaep
TripleDesSha256	Sha256	TripleDes	KwTripleDes	KwRsaOaep
Basic256Sha256Rsa15	Sha256	Aes256	KwAes256	KwRsa15
Basic192Sha256Rsa15	Sha256	Aes192	KwAes192	KwRsa15
Basic128Sha256Rsa15	Sha256	Aes128	KwAes128	KwRsa15
TripleDesSha256Rsa15	Sha256	TripleDes	KwTripleDes	KwRsa15

wrap アルゴリズムは双方に影響を与える．ただし，<sp:AlgorithmSuite> assertion が記述されている<wsp:Policy>要素の中に<sp:SignedParts> assertions も<sp:SignedElements> assertions もない場合，key wrap アルゴリズムは健全性のセキュリティレベルに影響を与えない．同様に，<sp:AlgorithmSuite> assertion が記述されている<wsp:Policy>要素の中に<sp:EncryptedParts> assertions も<sp:EncryptedElements> assertions もない場合，key wrap アルゴリズムは秘匿性のセキュリティレベルに影響を与えない．

個々のアルゴリズムスイートにおいて，メッセージダイジェスト関数は SHA-1 (表 1 における “Sha1”) か SHA-256 (同 “Sha256”) である．一般的には，SHA-256 の方が SHA-1 よりも強度が高い．したがって，メッセージダイジェスト関数の健全性に関する強度の観点から，表 1 中のアルゴリズムスイートは以下のように，強度の順序が制約される．

$$
\begin{aligned}
& \text{Basic256} = \text{Basic192} \\
&= \text{Basic128} = \text{TripleDes} \\
&= \text{Basic256Rsa15} = \text{Basic192Rsa15} \\
&= \text{Basic128Rsa15} = \text{TripleDesRsa15} \\
\sqsubseteq\ & \text{Basic256Sha256} = \text{Basic192Sha256} \\
&= \text{Basic128Sha256} = \text{TripleDesSha256} \\
&= \text{Basic256Sha256Rsa15} \\
&= \text{Basic192Sha256Rsa15} \\
&= \text{Basic128Sha256Rsa15} \\
&= \text{TripleDesSha256Rsa15}
\end{aligned}
$$

また，個々のアルゴリズムスイートにおいて，暗号アルゴリズムは，AES-256 (表 1 における “Aes256”), AES-192 (同 “Aes192”), AES-128 (同 “Aes128”), Triple-DES (同 “TripleDes”) のいずれかである．一般的には，AES-256, AES-192, AES-128, Triple-DES の順で強度が高い．したがって，暗号アルゴリズムの秘匿性に関する強度の観点から，表 1 中のアルゴリズムスイートは以下のように強度の順序が制約される．

$$
\begin{aligned}
& \text{Basic256} = \text{Basic256Rsa15} \\
&= \text{Basic256Sha256} = \text{Basic256Sha256Rsa15} \\
\sqsupseteq\ & \text{Basic192} = \text{Basic192Rsa15} \\
&= \text{Basic192Sha256} = \text{Basic192Sha256Rsa15} \\
\sqsupseteq\ & \text{Basic128} = \text{Basic128Rsa15} \\
&= \text{Basic128Sha256} = \text{Basic128Sha256Rsa15} \\
\sqsupseteq\ & \text{TripleDes} = \text{TripleDesRsa15} \\
&= \text{TripleDesSha256} = \text{TripleDesSha256Rsa15}
\end{aligned}
$$

同様に，個々のアルゴリズムスイートにおいて，対象鍵の key wrap アルゴリズムは，AES-256 (表 1 における “KwAes256”), AES-192 (同 “KwAes192”), AES-128 (同 “KwAes128”), Triple-DES (同 “KwTripleDes”) のいずれかである．対象鍵の key wrap アルゴリズムの健全性と秘匿性に関する強度の観点から，表 1 中のアルゴリズムスイートは以下のように強度の順序が制約される．

$$
\text{Basic256} = \text{Basic256Rsa15}
$$

$$
\begin{aligned}
&= \text{Basic256Sha256} = \text{Basic256Sha256Rsa15} \\
\sqsupseteq\ &\text{Basic192} = \text{Basic192Rsa15} \\
&= \text{Basic192Sha256} = \text{Basic192Sha256Rsa15} \\
\sqsupseteq\ &\text{Basic128} = \text{Basic128Rsa15} \\
&= \text{Basic128Sha256} = \text{Basic128Sha256Rsa15} \\
\sqsupseteq\ &\text{TripleDes} = \text{TripleDesRsa15} \\
&= \text{TripleDesSha256} = \text{TripleDesSha256Rsa15}
\end{aligned}
$$

また，非対象鍵の key wrap アルゴリズムは，RSA-OAEP (表 1 における “KwRsaOaep”), RSA-1.5 (同 “KwRsa15”) のいずれかである．一般的には，RSA-OAEP, RSA-1.5 の順で強度が高い．したがって，非対象鍵の key wrap アルゴリズムの健全性と秘匿性に関する強度の観点から，表 1 中のアルゴリズムスイートは以下のように強度の順序が制約される．

$$
\begin{aligned}
&\text{Basic256} = \text{Basic192} \\
&= \text{Basic128} = \text{TripleDes} \\
&= \text{Basic256Sha256} = \text{Basic192Sha256} \\
&= \text{Basic128Sha256} = \text{TripleDesSha256} \\
\sqsupseteq\ &\text{Basic256Rsa15} = \text{Basic192Rsa15} \\
&= \text{Basic128Rsa15} = \text{TripleDesRsa15} \\
&= \text{Basic256Sha256Rsa15} \\
&= \text{Basic192Sha256Rsa15} \\
&= \text{Basic128Sha256Rsa15} \\
&= \text{TripleDesSha256Rsa15}
\end{aligned}
$$

さらに，以下の assertion

```
<sp:ProtectTokens>
<sp:OnlySignEntireHeadersAndBody>
<sp:IncludeTimestamp>
<sp:EncryptSignature>
```

は，健全性のレベルに影響を与える．個々の assertion はオプショナルであり，子ノードを持たない．個々の assertion について，その assertion が与えられたセキュリティポリシーと，その assertion 以外は同一のセキュリティポリシーでは，前者が後者より強度が高い (もしくは同等である)．

上述の security assertion は個々に独立であり，相互には健全性の強度の順序が存在しない．たとえば，<sp:IncludeTimestamp> assertion が与えられたセキュリティポリシーと，<sp:IncludeTimestamp> assertion の代わりに<sp:ProtectTokens> assertion が与えられたセキュリティポリシーの間には，健全性のレベルの順序関係に関する制約は存在しない．

サポーティングトークンアサーション (supporting tokens assertions)，つまり，以下の assertion

```
<sp:SupportingTokens>
<sp:SignedSupportingTokens>
<sp:EndorsingSupportingTokens>
<sp:SignedEndorsingSupportingTokens>
```

は，追加のセキュリティトークンによるメッセージ防護について記述するために用いられる．したがって，あるサポーティングトークンアサーションが記述されているとき，それが持つ<wsp:Policy>要素中に記述されている<sp:AlgorithmSuite> assertion は，健全性および秘匿性のレベルに影響を与える．また，endorsing に関するサポーティングトークンアサーション，つまり，以下の assertion

```
<sp:EndorsingSupportingTokens>
<sp:SignedEndorsingSupportingTokens>
```

は，健全性の強度を高める．なぜなら，それらの assertion は，メッセージのディジタルシグニチャに対してさらに署名するからである．また，署名つきサポーティングトークンアサーション，つまり，以下の assertion

```
<sp:SignedSupportingTokens>
<sp:SignedEndorsingSupportingTokens>
```

は，署名のために用いられるので健全性の強度を高める．なぜなら，その assertion のセキュリティトークンは，そのメッセージの署名鍵で署名されるからである．

セキュリティ設計者は，これまで述べたそれぞれの制約の全てを満たすようにセキュリティポリシーの抽象化方法を定義しなければならない．これらの制約は，セキュリティポリシーの部分的な assertion の順序関係を定義しているだけなので，制約がない部分については自由にマッピングを定義してもよい．

仮に，以下の 2 つのセキュリティポリシーが個々のサービスに与えられているとする．ここでは，Policy 1 がリテイラーサービスに，Policy 2 がウェアハウスサービスに与えられているとする．なお，以下のセキュリティポリシーは，説明を容易にするために，いくつかの要素を省いて単純化している．

```
[Policy 1]
<wsp:Policy>
  <sp:SymmetricBinding>
    <wsp:Policy>
      <sp:AlgorithmSuite>
        <wsp:Policy>
          <sp:Basic256Sha256/>
        </wsp:Policy>
      </sp:AlgorithmSuite>
      <sp:IncludeTimestamp/>
      <sp:ProtectTokens/>
    </wsp:Policy>
  </sp:SymmetricBinding>
  <sp:SignedParts>
    <sp:Body/>
  </sp:SignedParts>
  <sp:EncryptedParts>
    <sp:Body/>
  </sp:EncryptedParts>
</wsp:Policy>

[Policy 2]
<wsp:Policy>
  <sp:SymmetricBinding>
    <wsp:Policy>
      <sp:AlgorithmSuite>
        <wsp:Policy>
          <sp:Basic256/>
        </wsp:Policy>
      </sp:AlgorithmSuite>
    </wsp:Policy>
  </sp:SymmetricBinding>
  <sp:SignedParts>
    <sp:Body/>
  </sp:SignedParts>
  <sp:EncryptedParts>
    <sp:Body/>
  </sp:EncryptedParts>
</wsp:Policy>
```

これらのセキュリティポリシーに関して，秘匿性のセキュリティレベルは同一になる．なぜなら，いずれのセキュリティポリシーのアルゴリズムスイート (Basic256Sha256 と Basic256) においても，同じ暗号アルゴリズムと同じ対象鍵 key wrap アルゴリズムが用いられているためである．一方で，これらのセキュリティポリシーに関して，健全性のセキュリティレベルは異なる．なぜなら，Basic256Sha256 のメッセージダイジェスト関数は SHA-256 であり，Basic256 のメッセージダイジェスト関数は SHA-1 だからである．このため，Policy 1 の健全性に関するセキュリティレベルは，Policy 2 のそれよりも強い．

セキュリティ設計者が，秘匿性に関するセキュリティレベルとして，**high**, **middle**, **low** を，また同様に，健全性に関するセキュリティレベルとして，**strong**, **normal**, **poor** を，それぞれ定義しているとする．また，セキュリティ設計者はセキュリティポリシーの抽象化方法を定義してあり，それにしたがうと，Policy 1 は [confidentiality: **high**, integrity: **strong**] (SQ1) に，Policy 2 は [confidentiality: **high**, integrity: **normal**] (SQ2) に，それぞれマップすると仮定する．このように，個々のセキュリティポリシーに対応する security qualifier を求めてから，それを用いて整合性を検証する方法を次節で述べる．

3.2 コンプライアンスルールとデータのセキュリティ要件

抽象化によって得られる security qualifier は，サービスの各操作の引数に代入されるデータが通信路上で交換されるときの秘匿性と健全性の要件として扱われる．そのように操作の引数に対してセキュリティ要件を与えるだけでは，システム全体のセキュリティ要件を定義するのに十分ではない．全体のサービスは，それを利用する組織のコンプライアンスルールに従う必要があるからである．本稿では，コンプライアンスルールから，データに関するセキュリティ要件を取り出し，それと security qualifier との整合性を判断することも併せて提案する．たとえば，コンプライアンスルールが以下のように与えられているとする．

> 「顧客 (Customer) のデータは高い機密性と健全性をもって扱われなければならない．ここで『顧客のデータ』とは，顧客の個人的な情報のことを意味する．たとえば，顧客の個人名，住所，電話番号，メイルアドレス，性別，生年月日，年齢などである．顧客のデータを含むいかなるメッセージも，それがネットワークで転送される場合は高い機密性と健全性によって防護されなければならない．顧客のデータの暗号化には，AES-256 が用いられなければならない．顧客のデータのディジタル署名には，メッセージダイジェスト関数として SHA-256 かそれ以上の強度を持つ関数が用いられなければならない．」

アプリケーションモデルでは，サービスで使用するデータの構造が定義されている．それらのデータ構造の表記法は，UML のクラス図などを用いることができる．たとえば，UML のクラス図を用いて図 5 のようにデータ構造を定義することができる．

このようなデータ構造定義があるとき，コンプライアンスルールが対象とするデータに対するセキュリティ要件を導く．この例では，コンプライアンス

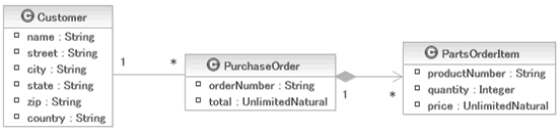

図 5: WS-I SCM データ構造 (一部)

ルールが対象とするデータである『顧客のデータ』は，クラス Customer として定義されているとする．

前述したセキュリティポリシーの抽象化の方法の決定による健全性と秘匿性についての順序関係と，このコンプライアンスルールをもとに，セキュリティ設計者は，コンプライアンスルールから導かれるデータ (クラス Customer) に関するセキュリティ要件が，どの順序に相当するかを決定する．この例では，AES-256 による暗号化を施せるのは前述のセキュリティポリシー Policy 1 および Policy 2 であり，SHA-256 を署名のためのメッセージダイジェスト関数に用いているのは Policy 1 である．よって，コンプライアンスルールから導かれるデータ (クラス Customer) に関するセキュリティ要件は，Policy 1 に対応する security qualifier で与えられるとみなすことができる．これを図 5 の UML クラス図に対する security qualifier として付与すると，図 6 のように表現することができる．

図 6: データのセキュリティ要件例

この図では，注文者である顧客クラス Customer に security qualifier が付与されており，この Customer クラスのインスタンスであるオブジェクト (つまりデータ) は，どのサービス間で受け渡される場合でも，その security qualifier のもとで扱われることを要求している．

3.3 Security qualifier 間の整合性検証

個々のサービスのセキュリティポリシーを security qualifier に変換した様子を図 3 に示す．得られた security qualifier 間の整合性は，プロセスフローで解析することで検証される．リテイラーサービスの操作 `submitOrder` についてのプロセスフローを図 7 に示す．この操作は，顧客 `Customer` が発行した注文 `PartsOrder` を受け取り，3 つのウェアハウスサービス A, B, C の操作 `shipGoods` を呼び出す．あるウェアハウスサービスに，その注文に対して十分な在庫がある場合は，操作 `submitOrder` は，在庫がある旨の応答 `OrderResponse` を返す．もし，どのウェアハウスサービスにも十分な在庫がない場合は，その操作は，0 を与えた応答 `OrderResponse` を返す．

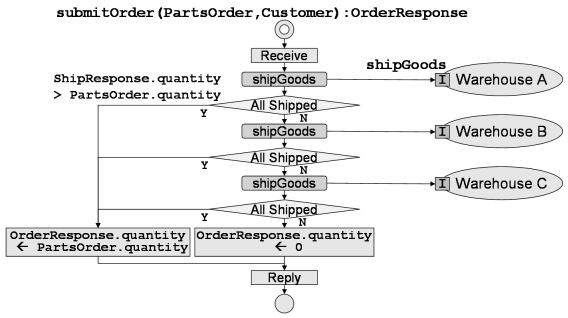

図 7: プロセスフロー

プロセスフローの変数は，security qualifier をセキュリティ型として型付けされている．前述のように，リテイラーサービスには security qualifier SQ1 が与えられ，ウェアハウスサービスには security qualifier SQ2 が与えられる．したがって，変数 `PartsOrder`, `Customer`, `OrderResponse` は SQ1 に型付けされ，`ShipResponse` (ウェアハウスサービスの操作 `shipGoods` の戻り値) は SQ2 に型付けされる．この型付けとプロセスフローに対して，基本的に文献 [7] に示される typing rule にもとづいて，型検査を用いた情報フロー解析を行なうことで，整合性を検証する．なお，文献中ではセキュリティ型が $high, low$ の二種類だけで typing rule が定義されているが，以下のように一部の rule を拡張することで任意の種類に対応させることができる．この場合，$high$ を束の上限，low を束の下限と再定義しておく．

$$C7' \qquad \frac{[T_1] \vdash C \quad T_1 \sqsubseteq T_2}{[T_2] \vdash C}$$

また，上記文献では秘匿性を想定して typing rule を定義してあるが，健全性の場合でも同じ rule を援用できる．ただし，健全性の場合はセキュリティ型の順序を反転して rule を扱う必要がある．$high$ を下限，

low を上限とみなすのは直感的にわかりにくい上に，秘匿性と健全性はそれぞれ別のセキュリティ型として扱うので，以下に，健全性のセキュリティ型に応じて書き改めた typing rule を示す．なお，秘匿性と区別するために，$high, low$ ではなく，$string, weak$ を，それぞれ，健全性のセキュリティ型の上限と下限とし，それぞれに対応する変数を，s, w とする．この typing rule のうち，$C1_I, C4_I, C5_I, C6_I$ は，上記文献の $C1, C4, C5, C6$ と同一になる．

$$E1_I \quad \vdash : weak$$

$$E2_I \quad \frac{w\ is\_not\_member\_of\ Vars(exp)}{\vdash exp : strong}$$

$$C1_I \quad [pc] \vdash skip$$

$$C2_I \quad [pc] \vdash w := exp$$

$$C3_I \quad \frac{\vdash exp : strong}{[strong] \vdash s := exp}$$

$$C4_I \quad \frac{[pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash C_1; C_2}$$

$$C5_I \quad \frac{\vdash exp : pc \quad [pc] \vdash C}{[pc] \vdash while\ exp\ do\ C}$$

$$C6_I \quad \frac{\vdash exp : pc \quad [pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash if\ exp\ then\ C_1\ else\ C_2}$$

$$C7_I \quad \frac{[T_1] \vdash C \quad T_1 \sqsupseteq T_2}{[T_2] \vdash C}$$

このように定義された typing rule のもとで，プロセスフロー中の変数のセキュリティ型を静的に検査する．プロセスフロー中の条件式

```
ShipResponse.quantity > PartsOrder.quantity
```

の，健全性についてのセキュリティ型は **normal** である．なぜなら，変数 `PartsOrder` の健全性についてのセキュリティ型は **strong** だが，変数 `ShipResponse` の健全性についてのセキュリティ型は **normal** となるからである．この条件式のセキュリティ型が **normal** となるため，この条件分岐の健全性に関するプログラムコンテキストは **normal** となる．これは implicit flow である．この条件分岐は以下の代入

```
OrderResponse.quantity <-- PartsOrder.quantity
```

に到達する．この代入の健全性についてのセキュリティ型は **strong** である．条件分岐のプログラムコンテキストと，条件分岐先の文のセキュリティ型が一致しないため，健全性に関して不整合であることが導かれる．

また，ウェアハウスサービスの操作 `shipGoods` には `Customer` のデータが渡される．セキュリティポリシー Policy 2 で指定されているアルゴリズムスイート Basic256 のメッセージダイジェスト関数は SHA-1 であるため，コンプライアンスルールによって定義されるデータのセキュリティ要件を満たせないことが検出される．

4 関連研究

抽象的なリンク仕様記述言語を導入することでセキュリティポリシーを検証する方法が提案されている [10]．高水準なリンク仕様によって，SOAP プロセッサ間を流通するメッセージについての対象とする機密性と認証のゴールを記述する．リンク仕様は WS-SecurityPolicy によるセキュリティポリシーファイルにコンパイルされる．その上で，そのリンク仕様のセキュリティのゴールが，与えられた WS-SecurityPolicy ファイルの集合によって満たされるかどうかを，π-calculus を用いて検証する．この手法はセキュリティ設計者が直接記述した WS-SecurityPolicy ファイルではなく，抽象的なリンクしようから生成された WS-SecurityPolicy ファイルを対象としている．これに対して我々の手法は，セキュリティ設計者が直接記述した WS-SecurityPolicy ファイルを対象としている．

アスペクト指向技術を用いて，BPEL による Web サービス合成をセキュアに行なうためのフレームワークが提案されている [11]．このフレームワークは 3 つのコンポーネントから構成される: セキュリティサービス，プロセスコンテナ，デプロイメントディスクリプタである．プロセスコンテナは，AO4BPEL [12] で記述されるアスペクトの集合として実装される．AO4BPEL は，高いモジュール性と適用可能性を与える BPEL のアスペクト指向拡張である．これらのアスペクトは一般的なアスペクトライブラリから開発時にデプロイメントディスクリプタにしたがって生成される．デプロイメントディスクリプタは，BPEL のアクティビティとそのパラメータについてのセキュリティ要件を記述する．この手法は，BPEL の解釈系の拡張であり，セキュリティは生成されたあすぺうとによって実行時に取り扱われる．これに対して我々の手法は，セキュリティポリシーを設計時に静的に検証する．

5 まとめ

本稿では，セキュリティポリシー間の整合性を，ポリシーを抽象化した security qualifier を求めることで検証する方法を提案した．セキュリティ設計者が事前に抽象化方法を定義しておくが，WS-SecurityPolicy の記述言語の仕様で定義されているセマンティックスに基づいた制約を満たすように，抽象化方法は定義される必要がある．その抽象化方法にしたがって，個々のセキュリティポリシーから対応する security qualifier を得る．求めた security qualifier はアプリケーションモデルにおける対応するサービスコンポーネントに割り当てられ，BPEL で記述されたプロセスフローをもとに情報フロー解析技術を用いて整合性が検証される．

SOA 環境においてもアプリケーションはさらに複雑化しているため，その開発は困難になりつつある．CBM のようなボトムアップ開発方法論は，既存のサービスコンポーネントの再利用によって開発を容易にすることを目的としているが，セキュリティなどの非機能的側面についての支援はあまりない．SOA に基づくアプリケーションがより広く適用されるにともない，そのセキュリティは重要度を増していく．我々が提案した整合性検証手法は，既存のサービスコンポーネントを組み合わせてアプリケーションソフトウェアを開発する際の困難を減らすことができると考えている．

参考文献

[1] CBDI Forum Ltd., A CBDI Report Series - Guiding the Transition to Web Services and SOA, `http://www.cbdiforum.com/bronze/downloads/ws_roadmap_guide.pdf` (2003).

[2] Premkumar T. Devanbu and Stuart Stubblebine, Software Engineering for Security: a Roadmap, In *Proc. of the 22nd International Conference on Software Engineering (ICSE 2000)*, Limerick, Ireland, 2000, pp. 227–239,

[3] Ross J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, John Wiley & Sons, Inc. (2001).

[4] Giovanni Della-Libera, Martin Gudgin, Phillip Hallam-Baker, Maryann Hondo, Hans Granqvist, Chris Kaler, Hiroshi Maruyama, Michael McIntosh, Anthony Nadalin, Nataraj Nagaratnam, Rob Philpott, Hemma Prafullchandra, John Shewchuk, Doug Walter and Riaz Zolfonoon, Web Services Security Policy Language (WS-SecurityPolicy) Version 1.1, `http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf` (2005).

[5] Web Services Interoperability Organization (WS-I), Supply Chain Management Sample Application Architecture, `http://www.ws-i.org/SampleApplications/SupplyChainManagement/2003-12/SCMArchitecture1.01.pdf` (2003).

[6] Dennis Volpano, Cynthia Irvine and Geoffrey Smith, A Sound Type System for Secure Flow Analysis, *Journal of Computer Security*, Vol. 4, No. 2-3 (1996), pp. 167–187.

[7] Andrei Sabelfeld and Andrew C. Myers, Language-Based Information-Flow Security, *IEEE Journal on Selected Areas in Communications*, Vol. 21, No. 1 (2003), pp. 5–19.

[8] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic and Sanjiva Weerawarana, Business Process Execution Language for Web Services (BPEL4WS) Version 1.1, `http://www-128.ibm.com/developerworks/library/specification/ws-bpel/` (2005).

[9] OASIS Open, Web Services Business Process Execution Language Technical Committee (WS-BPEL TC), `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`

[10] Karthikeyan Bhargavan, Cédric Fournet and Andrew D. Gordon, Verifying Policy-Based Security for Web Services, In *Proc. of the 11th ACM Conference on Computer and Communications Security (CCS'04)*, Washington DC, U.S.A., 2004, pp. 268–277.

[11] Anis Charfi and Mira Mezini, Using Aspects for Security Engineering of Web Service Compositions, In *Proc. of the IEEE International Conference on Web Services (ICWS'05)*, Orlando, Florida, U.S.A., 2005, pp. 59–66.

[12] Anis Charfi and Mira Mezini, Aspect-Oriented Web Service Composition with AO4BPEL, In *Proc. of the European Conference on Web Services (ECOWS'04)*, LNCS 3250, Springer-Verlag, Erfurt, Germany, 2004, pp. 168–182.