XML 文書のアクセス制御におけるポリシー簡略化の定量的評価

王 波† チャットウィチェンチャイ ソムチャイ‡ 岩井原 瑞穂†

†京都大学大学院情報学研究科社会情報学専攻 〒606-8501 京都市左京区吉田本町
‡県立長崎シーボルト大学国際情報学部 〒851-2195 長崎県西彼杵郡長与町まなび野 1-1-1
E-mail: †{wang, iwaihara}@db.soc.i.kyoto-u.ac.jp, ‡somchaic@sun.ac.jp

あらまし 本稿では，XML 文書のアクセス制御におけるポリシー簡略化のアルゴリズムとその定量的評価について報告する．HL7 患者情報記録，TravelXML 予約情報レポート，および VISA 送り状の 3 種類の DTD（アクセス制御情報なし）を使って，実験用のインスタンスを生成し，これらのインスタンスにアクセス制御ルールをランダムに生成し，われわれの提案した First-applicable，Deny-overrides および Permit-overrides による Cascade 最適化のアルゴリズムをそれぞれ適用してポリシーの簡略化を行い，その定量的評価を行った．最適化によりルール数が大幅に減り，アクセス判定時間も短縮することが分かった．実験の結果を踏まえた，アルゴリズムの改良も検討する．

キーワード XML， e-commerce，セキュリティ，アクセス制御

A Quantitative Evaluation of the Simplification of Access Authorization Policies for XML Documents

Bo WANG† Somchai CHATVICHIENCHAI‡ and Mizuho IWAIHARA†

† Department of Social Informatics, Kyoto University

Honmachi,Yoshida, Sakyo-ku, Kyoto, 606-8501 Japan
‡ Department of Info-Media, Siebold University of Nagasaki

1-1-1 Manabino, Nagayo-cho, Nishisonogi-gun, Nagasaki, 851-2195 Japan
E-mail: †{wang, iwaihara}@db.soc.i.kyoto-u.ac.jp, ‡somchaic@sun.ac.jp

Abstract In this paper, we propose three cascade-based algorithms to simplify access authorization policies for XML documents and report a quantitative evaluation based on the algorithms. We used three real-world DTDs without built-in access control information to do experiments: HL7 patient records, TravelXML allotment booking reports, and VISA invoices. We randomly generated 1,000 XML documents from each of these DTDs, and randomly inserted an access authorization rule to each open tag of each document with a probability of 5%~95%. Then, we simplified the policies and quantitatively evaluated the simplification, using First-applicable, Deny-overrides, and Permit-overrides algorithm respectively. The evaluation result demonstrates that the algorithms are fairly effective in reducing both access authorization rules and access time. We conclude by discussing further improvement to the algorithms based on the evaluation.

Keyword XML, e-Commerce, Security， Access Authorization Control

1. はじめに

近年，XML による情報流通が活発になるにつれ，XML 文書の情報を保護するためのアクセス制御が重要となってきている. XML 文書内のデータにおけるきめの細かいアクセス制御は，XML 文書の構造・内容に直接的にアクセスの制約をかける．ゆえに，アクセス権限判定ポリシーは XML 文書の構造・内容に深く関わる．

XML 文書の構造は，アプリケーションの拡張や組織間のデータ交換など様々な理由で変わる傾向があるので[1][2][3]，新しい構造に適合するように権限判定を修正しなければならないが，新しいポリシーは冗長な場合が多い．また，ユーザが個人情報に関するアクセス制御情報を自ら定義する場合，個々のユーザの定義したポリシーをまとめて簡略化する必要もあるが，ポリシーの簡略化問題はこれまで検討されていなかった．

われわれは， Deny-overrides, Permit-overrides, および First-applicable による cascade 最適化のアルゴリズ

ムを提案しているが[4], 本論文では計算機実験によるポリシー簡略化の定量的評価を検討する.

以下, 本稿は第 2 節において XML のアクセス制御言語である XACML を述べ, 第 3 節においてアクセス制御ポリシーの簡略化アルゴリズムについて述べ, 第 4 節においてポリシー簡略化の定量的評価を述べる. 第 5 節はまとめと今後の課題である.

2. XML 文書のアクセス制御

XML 文書[5]のアクセス制御は, 文書の構造・内容によって定義される. 値に依存する権限付与（value-dependent authorization）の場合は, パス式で要素あるいは属性の値を比較し, アクセスの可否を決める. 値に依存しない権限付与（value-independent authorization）の場合もパス式を使うが, 要素と属性の値を比較しない.

2.1. XACML (eXtensible Access Control Markup Language)

XACML[6]は, XML 文書に対するアクセス権を設定するための仕様である. セキュリティ管理者は XACML 言語を利用して, ポリシー（あるリソースに対して,「だれ」がどのような「権限」で「どこ」にアクセスできるのか）を定義することができる. 主要なトップレベル要素は PolicySet（ポリシー集合）で, これは他の PolicySet 要素や Policy（ポリシー）要素を 1 つにまとめたものである. Policy 要素は主に Target（対象）, Rule （ルール）, Obligation（責務）の各要素から構成される.

Target 要素は, 要求されたリソースと適用可能な Policy を関連付けるのに使用される. アクセス制御の対象である主体<Subjects>, 資源<Resources>, 動作<Actions>の 3 つの体<Subjects>, 資源<Resources>, 動作<Actions>の 3 つの要素に対する規則を定める. ルールはだれ<Subjects>に対して, どのような資源<Resources>を, どのように動作<Actions>させるかを記述する. PolicySet, Policy, または Rule をリソースに適用できるためには, 要求の Subject （サブジェクト）, Resource（リソース）, または Action（アクション）が一定の条件を満たす必要があり, Target 要素にはそうした条件が記述される. Target 要素には, Policy のインデックス付け/ルックアップを効率化するための組み込みメカニズムが用意されている.

アクセス決定に適用できるポリシーが複数見つかる場合（また, 単一のポリシーに複数の Rule が含まれている場合）があるので, 結合アルゴリズムを用いて, 複数の判定結果が単一の決定に一本化される. ルール結合アルゴリズムとしては以下の 3 つのアルゴリズムが規定されている.

・Deny-overrides（拒否優先）

ポリシー内のすべてのルールについて, どれか 1 つのルールの Effect(判定値)が deny であれば結果は deny とする. すべてのルールが permit, あるいは幾つかのルールが permit で残りのルールすべてが NotApplicable の場合のみ結果を permit とする.

・Permit-overrides（許可優先）

ポリシー内のすべてのルールについて, どれか 1 つのルールの Effect が permit であれば結果は permit とする. すべてのルールが deny, あるいは幾つかのルールが deny で残りのルールすべてが NotApplicable の場合のみ結果を deny とする.

・First-applicable（先行優先）

ポリシー内のすべてのルールについて順番に評価して, 対象<Target>がマッチした場合, 以後の<Rule>の評価を中断し<Target>の評価結果を Match とし, 次に Condition を評価し, これが True なら結果は Effect で指定された permit または deny とする.

First-applicable(先行優先)
(permit, deny) = permit; (deny, permit) = deny
Deny-overrides(拒否優先)
(permit, deny) = deny; (deny, permit) = deny
Permit-overrides(許可優先)
(permit, deny) = permit; (deny, permit) = permit

図 1. ルール結合アルゴリズム

ルール結合アルゴリズムはルールを評価するとき, 上記の permit または deny の決定以外に, 要求 Context が用意されたルールにすべてマッチしなければ評価結果は NotApplicable を, 評価の過程で構文エラーとなった場合 Indeterminate を返す.

3. アクセス制御ポリシーの簡略化

3.1. Cascade 最適化による簡略化

アクセス制御ルールに用いる XPath 式において, ターゲットとなる要素とそれを根とする部分木全体に同じ権限判定値を設定することは有用であり, 多くのアクセス制御モデルで採用されており[7][8], cascade と呼ばれることが多い. 本稿では cascade の設定を最適化することにより, ポリシー をできるだけ簡略化されたものにするアルゴリズムについて検討する. cascade を付加するとは, XPath 式においてターゲットとなる要素に descendant-or-self を設定することに相当する. そして cascade を持たないポリシーを入力とし, ルール数が最小となるように cascade を設定することを目

標とする．さらに以下を仮定する．

(仮定１) 取り扱う文書 D はすべて，アクセス制御ポリシーP を適用することにより，D の全要素について deny または permit の評価結果が得られる（NotApplicable となる要素は存在しない）．

（仮定２）ルール数最小化として，cascade の設定を各ノードで行うかどうかのみを判断するものとする．

XPath 式集合の簡略化としては，他にスキーマを考慮する方法なども考えられるが，今後の課題とする．ある部分木全体が deny あるいは permit のいずれか片方のみならば，１つのルールですむが，権限判定が子孫に含まれるときは，別のより詳細なルールが必要である．

記号	意味	ルール数
n	ノード e にはルールを設定しない．	0
+	ノード e に権限判定値が a(e)で cascade を行うルールを設定する．	1
-	ノード e に権限判定値が a(e)で cascade を行わないルールを設定する．	1
±	ノード e に権限判定値が a(e)で cascade を行わないルール，次に権限判定値が a(e)の反対で cascade を行うルールの２つを設定する（e 自身には前者，e の子孫には後者のルールが適用される）．	2

表１．ノードに設定するルールの記号

表１は，ノードごとに設定するアクセス制御ルールを４つの場合に分けたものである．ノード e にはあらかじめ権限判定値 a(e) ∈{permit, deny}が与えられているものとする．n は e にルールを設定しない（祖先から cascade されるものを利用する），+は a(e)の値を e に設定し，しかもそれを子孫に cascade する，-は a(e)の値を e に設定するが子孫には cascade しない，±は a(e)の値を e に設定するが，子孫には a(e)とは反転させた権限判定値を cascade させるというものである．１つのノードに対するルールの設定方法は，冗長なものを除くとこの４通りのみである．

3.2. First-applicable における cascade 最適化

われわれは，cascade 最適化アルゴリズムとして，First-applicable, Deny-overrides, および Permit-overrides の 3 つを提案している．First-applicable アルゴリズムは以下である．

アルゴリズム 1.

入力：各ノード e に 0(deny)または 1(permit)の権限判定値 a(e)が与えられた XML 文書 D

出力：D に対するポリシーP_D

1. D の根を e とする．e が子を持たないときは，rule (e) = “-“として終了する．e が子を持つ場合は，e の子の集合を$\{e_1,\ldots,e_k\}$とする．各子 e_i について，アルゴリズム 1 を再帰的に適用し，e_i を根とする部分木 D_i に対するポリシーP_{Di} を求める．e_i に対するルールは rule(e_i)に格納される．
2. $cost^+ = 0$, $cost^- = 0$, $cost^{\pm} = 1$ とする．
3. for each e_i in $\{e_1,\ldots,e_k\}$ {
   if ($a(e_i) = a(e)$) {
   switch rule(e_i) {
   case “+” : $cost^-$ += 1; $cost^{\pm}$+= 1;
   case “-” : $cost^-$ += 1; $cost^{\pm}$+= 1;
   case “±” : $cost^+$ += 2; $cost^-$ += 2; $cost^{\pm}$+= 1;
   }
   }
   else {
   switch rule(e_i) {
   case “+” : $cost^+$ += 1; $cost^-$ += 1;
   case “-” : $cost^+$ += 1; $cost^-$ += 1;
   case “±” : $cost^+$ += 1; $cost^-$ += 2; $cost^{\pm}$+= 2;
   }
   }
4. if ($cost^+ = \min(cost^+, cost^-, cost^{\pm})$) {
   rule(e) = “+”;
   for each e_i in $\{e_1,\ldots,e_k\}$ {
   if ($a(e_i) = a(e)$) {
   if (rule(e_i) = “+”) OR (rule(e_i) = “-”) {
   rule(e_i) = “n”;
   }
   }
   else {
   if (rule(e_i) = “±”) {
   rule(e_i) = “-”;
   }
   }
   }
   }
   else if ($cost^{\pm} = \min(cost^+, cost^-, cost^{\pm})$) {
   rule(e) = “±”;
   for each e_i in $\{e_1,\ldots,e_k\}$ {
   if ($a(e_i) \neq a(e)$) {
   if (rule(e_i) = “+”) OR (rule(e_i) = “-”) {
   rule(e_i) = “n”;
   }
   }
   }
   }
   else {
   rule(e) = “−”;
   }
   }
return;

［定理 1］ アルゴリズム 1 は入力の XML 文書 D のノード数|D|について O(|D|)時間で停止し，First-applicable の衝突解消アルゴリズムを用いる場合のルール数最小のポリシーP_Dを求める．

［証明］ アルゴリズム 1 において，各ノード e について高々１回のみ部分木の根として再起呼び出しが行われ，１回の呼び出しでは e とその子の

数に比例した計算時間であるため，全体で O(|D|) 時間であることがわかる．

ルールの合計数については，表 1 に示す各ノードのルールの数を合計したものである．以下，ルール数の最小性について D のノード数に関する帰納法を用いる．

(1) D が 1 つのノードのみからなる場合は，1 つのルール“-“がアルゴリズムにより得られ，また実際にこのルールが必要であるため，最小性は成り立つ．

(2) D よりノード数が小さい任意の文書 D’について，アルゴリズム 1 は最小のルール集合を求めると仮定する．このとき，D の根 e の子 $e_1,\ldots,e_k$ それぞれを根とする部分木を $D_1,\ldots,D_k$ とする．アルゴリズム 1 は，各部分木のルール集合をステップ 1 で再帰的に求める．ステップ 3 では，e と $e_1,\ldots,e_k$ の間で，e のルールを”+”，”-”，”±”のそれぞれのルールを設定したときに，省略できるルールを除いたルール数の合計を計算している．ステップ 4 では，ステップ 3 で求めたルールの合計数を最小とするルールを e に設定する．そして部分木の根 $e_1,\ldots,e_k$ それぞれについて省略できるルールを 1 つ除くかまたはそのままにする．ここで，もしアルゴリズム 1 よりもルール数が小さいポリシー P'_D が存在したとする．もし，P'_D の各部分木 $D_1,\ldots,D_k$ が P_D と一致するならば，根 e のルール数が P_D より小さいはずであるが，rule(e)は e と $e_1,\ldots,e_k$ の間で最小に選んであるためそれはあり得ない．すると，部分木 $D_1,\ldots,D_k$ のいずれかで P'_D は P_D より小さな解を求めているはずである．ところが帰納法の仮定により P_D は $D_1,\ldots,D_k$ の部分で最小のルール数を求めて，これからさらに部分木の根 $e_1,\ldots,e_k$ のルールを削減しているか同じであるため，これもあり得ない．よって P_D はルール数が最小である．□

3.3. Deny-overrides および Permit-overrides における cascade 最適化

Permit-overrides は Deny-overrides と同様な議論が可能であるため，一般性を失うことなく Deny-overrides について述べる．まず，Deny-overrides を First-applicable で模倣することについて検討する．First-applicable はルールの順序により，先行するルールの判定値が優先するというものであるから，Deny-overrides を模倣するには，Deny を行なうルールを先行させるように，ルールの順序付けを行なえばよいと考えられる．より一般的には，以下の性質が成り立つ．

［定理 2］ 与えられた XML 文書 D およびポリシー P_D について，ルール数が P_D と等しい次の条件を満たすポリシー P'_D が存在する．すなわち任意のアクセス要求について Deny-overrides (同様に Permit-overrides)アルゴリズムで P_D を評価した権限判定値と，First-applicable で P'_D を評価した権限判定値は常に一致する．

［証明］ P_D のルール集合について，D の各ノード e について，permit を与えるルール集合を p(e)，deny を与えるルール集合を d(e)とする．このとき，以下のアルゴリズムで出力されるポリシーを P'_D とする．

1. D の文書木を後順(post-order)で走査する．そして各要素 e が走査されたとき，d(e)の各ルールを任意の順で出力する．
2. 再び D の文書木を後順(post-order)で走査する．そして，各要素 e が走査されたとき，p(e)の各ルールを任意の順で出力する．

上記で出力されるルールの順序は，最初に deny を与えるルールが葉から根の順序で出力され，次に permit を与えるルールが葉から根の順序で出力される．また P_D を並べ替えただけであるからルール数は変わらない．ここでもしある権限判定が P_D において deny であるならば，P_D の中のあるルール r が deny の値を与えるはずである．すると，P'_D においても r は deny の値を持つ．P'_D においては，r よりも前に permit を与えるルールは存在しないため，P'_D においても First-applicable のアルゴリズムにより，deny の判定が行なわれる．P'_D は後順でルールが出力されているため，cascade の処理も正しく行なわれる．P'_D における判定値が permit の場合も同様な議論で P'_D も permit を出力することが確かめられる．Permit-overrides についても，上記の d(e)と p(e)，deny と permit をそれぞれ入れ替えればよい． □

定理 2 より，ルール数の観点からは，First-applicable を用いた方が Deny-overrides や Permit-overrides よりも常に等しいか少ないルール数が得られることが分かった．しかし，権限判定を行なうコストの観点からは，例えば Deny-overrides の場合に文書木を根から判定対象のノードに降下しながら各ルールの評価値を求めていった場合，途中でも deny の値を与えるルールが見つかれば，そこで木の探索を終了して，権限判定は deny であると結論することができる．このため祖先で deny を行なうルールが多い場合など，Deny-overrides が有利な局面も考えられる．以下では，Deny-overrides（および Permit-override）でのルールの最小化を行なうアルゴリズムについて考察する．

Deny-overrides では，deny をあるノードから cascade させた場合，そのノードからの部分木全体が deny になる．そのため，もしその部分木中に permit のノードが 1 つでもあるならば，cascade は使用できないことになる．また，アルゴリズム 1 で用いた±については，a(e)=permit のノード e については，deny が優先されるため±では deny

に判定されてしまい，用いることができない．以上を考慮すると以下のアルゴリズム 2 を構成することができる．ここでは permit と deny を交換すれば Permit-overrides 用に変更できる．

アルゴリズム 2.
入力:各ノード e に 0(deny)または 1(permit)の権限判定値 a(e)が与えられた XML 文書 D
出力：D に対するポリシーP_D

```
1. D の根を e とする. e が子を持たないときは, rule (e) =  "-"とする. さらに a (e)= deny のとき, 部分木全体が deny であることを示すフラグとして uniformity (e)="yes"とし, a (e)=permit のときは uniformity (e)="no"と設定して終了する. e が子を持つ場合は, e の子の集合を {e1...ek} とする. 各子 ei について, アルゴリズム 2 を再帰的に適用し, ei を根とする部分木 Di に対するポリシーPDi を求める. ei に対するルールは rule (ei)に格納される.
2. if (a(e) = deny AND
     uniformity(ei)="yes" holds for every ei in {e1,...,ek})
   {
      rule(e) = "+";   uniformity(e)="yes";
      return;
   }
3. uniformity(e) = "no";
   cost+ = 0;   cost- = 0;   cost± = 1;
4. for each ei in {e1,...,ek} {
     if (a(ei) = a(e)) then {
       switch rule(ei) {
         case "+" :   cost-  += 1;   cost±+= 1;
         case "-"   : cost-  += 1;   cost±+= 1;
         case "±" : cost+  += 1; cost-  += 2; cost±+= 1;
       }
     }
     else {
       switch rule(ei) {
         case "+" : cost+  += 1;   cost-  += 1;
         case "-" : cost+  += 1;   cost-  += 1;
         case "±" : cost+ += 1; cost- += 2; cost±+= 2;
       }
     }
   }
5. if ((cost- = min(cost+ , cost- , cost±)) OR
     (a(e)=deny AND cost+ = min(cost+, cost-, cost±)) OR
     (a(e)=permit AND cost± = min(cost+ , cost- , cost±)))
   {
      rule(e) = "−";
      return;
   }
6. /*±使用不可，+使用可*/
   if (a(e)=permit) {
     rule(e) = "+";
     for each ei in {e1,...,ek} {
       if (a(ei) = a(e)) then {
         if (rule(ei) = "+") OR ( rule(ei) = "-") {
           rule(ei) = "n";
         }
       }
       else {
         if (rule(ei) = "±") {
           rule(ei) = "-";
         }
       }
     }
     return;
 }
7. /*±使用可，+使用不可*/
   rule(e) = "±";
   for each ei in {e1,...,ek} {
     if (a(ei)  ≠  a(e)) {
       if ((rule(ei) = "+") OR (rule(ei) = "-" )) {
           rule(ei) = "n";
       }
     }
   }
   return;
```

［定理 3］ アルゴリズム 2 は入力の XML 文書 D のノード数|D|について O(|D|)時間で停止し，Deny-overrides の衝突解消アルゴリズムを用いる場合のルール数最小のポリシーP_Dを求める．

［証明］ 定理 1 とほぼ同様な議論で証明することができる．□

文書木を根から判定対象のノードに降下しながら各ルールの評価値を求めていった場合は，根要素に部分木全体が deny/ permit かを示すフラグの属性（例えば uniformity という属性名とする）があるかどうかによって，権限判定の効率が変わる．属性 uniformity があれば，どのアルゴリズムも時間的なコストがあまり変わらないことが考えられる．しかし，属性 uniformity のない場合は，判定の途中でも deny の値を与える cascade のルールが見つかれば，そこで木の探索を終了して，権限判定が deny であると結論付けることができるという，Deny-overrides が有利になる場合がある．

このように，3 つのアルゴリズムはそれぞれ利点・欠点を持っている．応用の場合は，ユーザによる選択が必要である．一般的には First-applicable の設定が最も効率的である．セキュリティを厳しくするには，Deny-overrides の設定が必要である．場合によっては，Permit-overrides の設定も考えられる．

4. ポリシー簡略化の定量的評価

大量の XML 文書を処理する計算機実験により，ポリシー簡略化の定量的な評価を行った．実験の結果，3 つの実用的な DTD から XML ファイルを 240,415 個生成した．データの合計サイズは 3.77GB である．

4.1. 実験の環境

実験の前半は Pentium 4 CPU 1.49GHz，256MB RAM の PC を使った．OS は Microsoft Windows XP で，実行プログラムは Java j2sdk_1.4.2_05 を用いて実装した．実験の後半は OS と実行プログラムが同じであるが，Pentium 4 CPU 3.20GHz，2GB RAM の PC を使った．

4.2. 実験用インスタンスの生成

HL7 の患者記録[9]，TravelXML の注文内容明細書[10]，および VISA 送り状[11]の 3 種類の DTD ファイル（アクセス制御情報なし）を集めて，ランダムにそれぞれ 1,000 個のインスタンスを作った．インスタンスのサイズは，4KB～20KB である．HL7 の場合は，要素がおよそ 150 種類で，属性が約 200 種類で，文書木の深さは最大 7 層くらいである．TravelXML の場合は，要素がおよそ 210 種類で，属性が 0 種類で，文書木の深さは最大 5 層くらいである．VISA の場合は，要素がおよそ 350 種類で，属性が約 610 種類で，文書木の深さが最大 5 層くらいである．図 2 は，VISA 送り状のインスタンスを簡略化したもので，アクセス制御情報は含まれていない．

```
<?xml version="1.0" encoding="UTF-8" ?>
<Invoice>
  <Party>
      <Name>Bob </Name>
 </Party>
 <Price>$100</Payment >
 <Tax>$5 </Tax >
 <Date>2005-01-05</Date>
</Invoice>
```

図 2. VISA 送り状（アクセス制御情報なし）

4.3. アクセス制御情報の添加

すべての文書において，すべての開始タグに対し，当該要素を制御するための情報を新しい属性として添加した．環境変数から属性名を読み取るのが望ましいが，今回は簡略化のため，属性 access と cascade のペアとし，1 つの開始タグに 1 ペアずつ入れた．cascade の値はすべて“－”としたが，access が 0 の確率を 5%～95%の 19 段階に設定し，1 つのインスタンスから新しいインスタンスを 19 個ランダムに作った．その結果，ルール数は開始タグ数と等しくなった（図 3）．

```
<?xml version="1.0" encoding="UTF-8" ?>
<Invoice access="1" cascade="-">
  <Party access="0" cascade="-">
      <Name access="0" cascade="-">Bob </Name>
 </Party>
 <Price access="1" cascade="-">$100</Payment >
 <Tax access="1" cascade="-">$5 </Tax >
 <Date access="1" cascade="-">2005-01-05</Date>
</Invoice>
```

図 3. アクセス制御情報を入れた VISA 送り状

4.4. アクセス制御ポリシーの簡略化

すべての文書に対して，First-applicable，Deny-overrides，および Permit-overrides の 3 つのアルゴリズムにおけるポリシー最適化の演算を行い，その結果を新しい文書に保存した．根要素の属性 cascade が”+”あるいは”±”の場合は，部分木全体が deny/ permit になっているかを示すために，新しい属性 uniformity を加えた．部分木全体が deny/ permit であれば uniformity の値を“yes”とし，さもなければ “no”とした．図 4 は，図 3 の VISA 送り状を First-applicable による cascade 最適化をしたものである．

```
<?xml version="1.0" encoding="UTF-8" ?>
<Invoice access="1" cascade="+" uniformity="no">
  <Party access="0" cascade="+" uniformity="yes">
      <Name access="0" cascade="n">Bob </Name>
 </Party>
 <Price access="1" cascade="n">$100</Payment >
 <Tax access="1" cascade="n">$5 </Tax >
 <Date access="1" cascade="n">2005-01-05</Date>
</Invoice>
```

図 4. First-applicable 簡略化後の VISA 送り状

4.5. ルール数の削減の評価

簡略化前後のルール数を計算し，ルール数をどれだけ減らしたかを計測した．図 5，図 6 および図 7 は，図 3 に示したような，HL7，TravelXML，および Visa のインスタンスを，それぞれ cascade 最適化により，簡略化した場合，ルール数がどれだけ減ったかを示している．

横軸は，アクセス制御情報をランダムに入れたとき，ルールが deny (属性 access= “0”) の確率（%）を表しており，当該インスタンスにおける deny のルールの占める割合とみなすことができる．縦軸は，簡略化前のルール数に対して，簡略化後のルール数が占める割合(%)である．

HL7，TravelXML および VISA の三種類の DTD のインスタンスを処理したが，VISA を例に説明する．横軸はルールが deny の確率が 5%から 95%まで 19 の段階となっているが，1 つの段階は 1,000 個のインスタンスを処理した平均値を表すので，全部で 19,000 個の VISA 送り状を簡略化した．

いずれも First-applicable において,ルール数の削減効果がもっとも顕著である．ルールが deny の確率が低いときは， permit の cascade が多く，一方 deny の確率が高いときは，deny の cascade が多いので，図の両端は簡略化によりルール数が大幅に減ったことを示している．

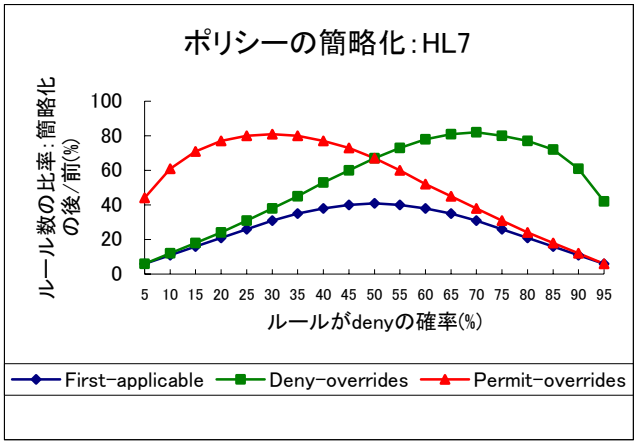

図 5. ポリシーの簡略化：HL7

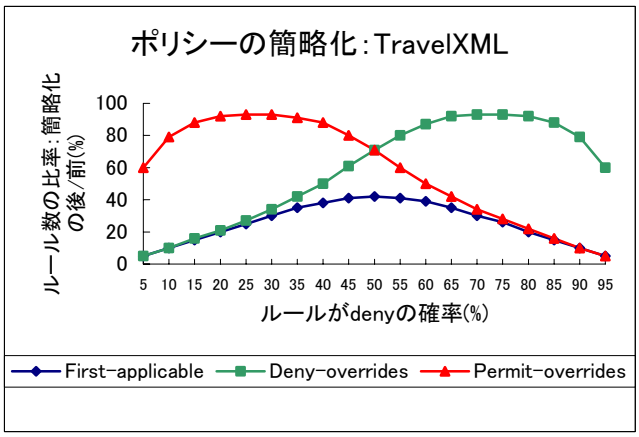

図 6. ポリシーの簡略化：TravelXML

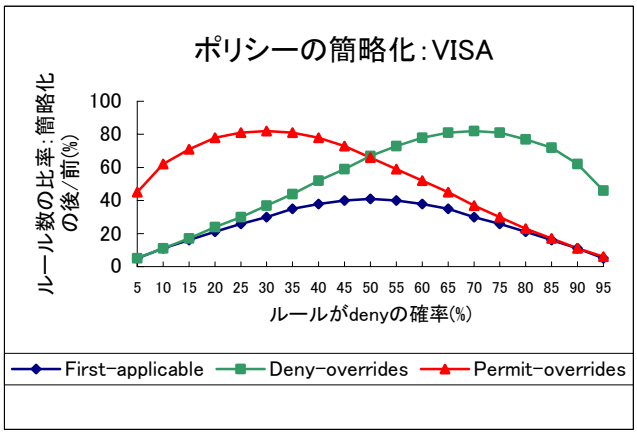

図 7. ポリシーの簡略化：VISA

Deny-overrides は，First-applicable ほどルール数が減少していなかった．ルールが deny の確率が低いとき，permit の cascade が多く，折れ線が First-applicable のそれとほぼ重なっている．ルールが deny の確率が高くなるにつれ，permit の cascade が少なくなるが，（子孫に 1 つでも permit のルールがあれば deny の cascade が使えないので）deny の cascade があまり増えないので，折れ線が右上がりになっている．また，deny の確率が 95%に近づくと，permit のルールが極端に減るので，deny の cascade がだんだん使えるようになり，折れ線が下がると考えられる．Permit-overrides の折れ線は，Deny-overrides のそれと対称的になっている．

4.6. 判定時間の短縮の評価

アクセス判定時間を評価するには，様々な方法があると考えられるが，実験の連続性に配慮し，ポリシーの簡略化されたインスタンスの全てのノードに対するアクセス権判定にかかる時間を計測してみた．これは，アクセス権を考慮したユーザ・ビューの生成時間に相当する[12]．

図 8 は，図 4 のインスタンスのユーザ・ビューである．図 4 の文書については，ユーザには属性 access が 0 つまり deny である要素を見せてはいけない．図 4 の文書木から属性 access が 1 つまり permit である要素のみ新しい文書木にコピーしたら，図 8 の文書になる．この新しい文書木のことをユーザ・ビューという．

```
<?xml version="1.0" encoding="UTF-8" ?>
<Invoice>
  <Price>$100</Payment >
  <Tax>$5 </Tax >
  <Date>2005-01-05</Date>
</Invoice>
```

図 8. VISA 送り状のユーザ・ビュー

ユーザ・ビューを生成するときに，図 4 のようなインスタンスを，1 個ずつメモリに読み込んで DOM ツリーを築き，アクセス判定を行って新たにユーザ・ビューのツリーを作り，その生成時間を計る．DOM ツリーを 2 回築くわけであるが，元のツリーの生成時間はユーザ・ビューの生成時間に含めない．

図 9，図 10 および図 11 は，図 4 に示したような，簡略化後の HL7, TravelXML, VISA インスタンスから，ユーザ・ビューを生成するのに，それぞれかかった平均時間を示す．横軸は，ルールが deny の確率を表す．縦軸は，ユーザ・ユーの生成時間である．比較のために，簡略化前のインスタンスからもユーザ・ビューを生成しておいた．

簡略化前のインスタンスは，cascade がまったく行われておらず，ユーザ・ビューを作るには，1 つずつノードのルールをチェックし，権限判定をしなければならないので，ルールが deny である確率が変わっても，権限判定にかかった時間が同じである．しかし，ルールが deny の確率が増えると，ユーザ・ビューのツリーへのコピー作業が減るので，それだけにユーザ・ビューの生成時間が減ると考えられる．要するに，ユーザ・ビューを作るのにかかる時間は，権限判定のための時間と，permit のノードを新しいツリーにコピーするための時間との合計である．したがって，簡略化前の生

成時間だけでなく，いずれの生成時間も，deny の確率が増えるにつれ，一定のペースで下がる傾向にあると考えられる．

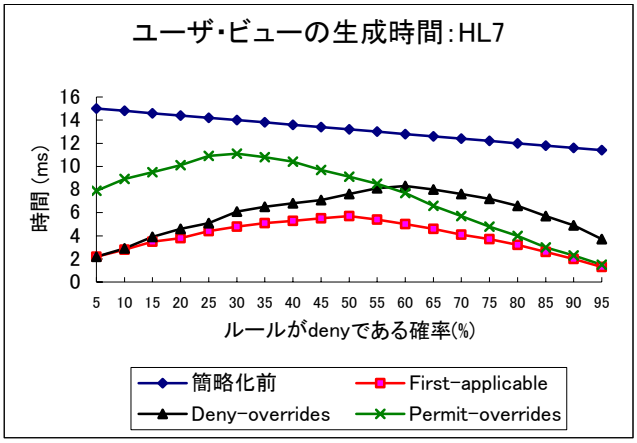

図 9．ユーザ・ビューの生成時間：HL7

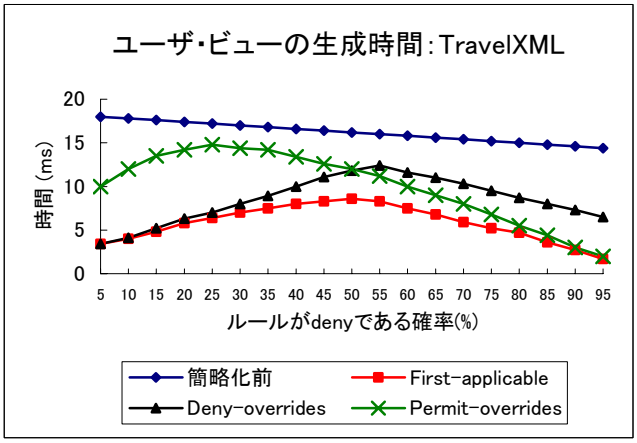

図 10．ユーザ・ビューの生成時間：TravelXML

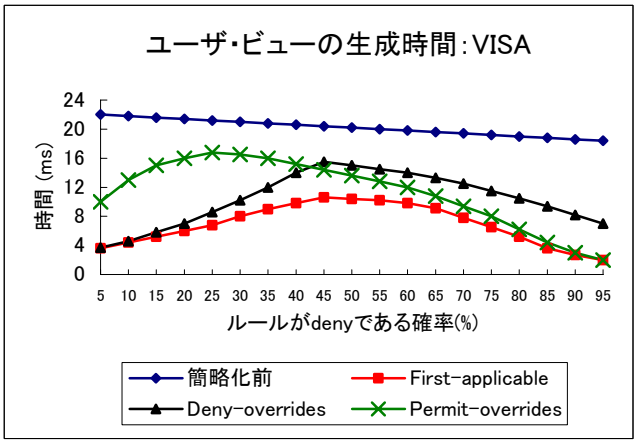

図 11．ユーザ・ビューの生成時間：VISA

簡略化後の 3 つの生成時間は，属性 uniformity が簡略化の段階で入れられたので，ユーザ・ビューの生成時間とルール数とがプラスの比例関係にあり，折れ線の形が図 5，図 6 および図 7 のそれと似ていて，3.3 節で議論した Deny-overrides の優位性は存在しない．

5. おわりに

われわれの提案した cascade 最適化のアルゴリズムを実装し，ポリシー簡略化の定量的評価を行った．最適化によりルール数が大幅に減り（First-applicable の場合，最大 94%，平均 71%のルールが削減できる），アクセス判定時間も（First-applicable の場合，最大 7 分の 1，平均 5 分の 1 まで）短縮されることが分かった．

今後は，スキーマを考慮した場合など，より広い XPath のクラスの簡略化について考察する予定である．また，次の課題としてマルチロールを導入し，新しいアルゴリズムの開発を検討する予定である．

文　献

[1] Chatvichienchai, S., Iwaihara, M., and Kambayashi, Y. Towards translating authorizations for transformed XML documents. In *Proc. of the 3rd International Conference on Web Information Systems Engineering* (WISE 2002), IEEE CS, pp. 291–300, Dec. 2002.

[2] Chatvichienchai, S., Iwaihara, M., and Kambayashi, Y. Translating content-based authorizations for XML documents. In *Proc. of the 4th Int. Conference on Web Information Systems Engineering (WISE 2003)*, IEEE CS, pp.103–112, Roma, Italy, Dec. 2003.

[3] Chatvichienchai, S., Iwaihara, M., and Kambayashi, Y. Authorization translation for XML document transformation. *Int. Journal of World Wide Web: Internet and Web Information Systems*, Kluwer, Vol. 7, No.1, pp. 111-138, Mar 2004.

[4] 王波，チャットウィチェンチャイ・ソムチャイ，岩井原瑞穂，“XML 文書のアクセス制御ポリシーの簡略化について”，DBWS2004，信学技報 Vol. 104 No. 176，pp. 103-108，July 2004.

[5] http://www.w3.org/XML/

[6] OASIS eXtensible Access Control Markup Language Technical Committee. XACML 1.0 Committee Specification Set. http://www.oasis-open.org/committees/ xacml/, Nov. 2002.

[7] Bertino, E., Castano, S., Ferrari, S., and Mesiti, M. Specifying and enforcing access control policies for XML document sources. *World Wide Web*, Baltzer Science Publishers, Netherlands, 3, 2000.

[8] Hitchens, M. and Varadharajan, V. RBAC for XML document stores. In *Information and Communications Security, 3rd Int. Conference*, ICICS 2001, LNCS2229, pp.121–135, Nov. 2001.

[9] http://www.hl7.org/library/standards_non1.htm

[10] http://www.xmlconsortium.org/wg/TravelXML/TravelXML_index.html

[11] http://international.visa.com/fb/downloads/commprod/visaxmlinvoice/main.jsp?jsessionid=2624421085118079244

[12] 戚乃箴，工藤道治，“アクセス条件テーブルを用いた XML アクセス制御”，DBWS2004，信学技報 Vol. 104 No. 176，pp. 109-114，July 2004.