XML 文書のアクセス制御の効率化に関する研究

北川　直毅† 　　吉川　正俊††

† 名古屋大学情報科学研究科
†† 名古屋大学情報基盤連携センター
名古屋市千種区不老町
E-mail: †kitagawa@dl.itc.nagoya-u.ac.jp, ††yosikawa@itc.nagoya-u.ac.jp

あらまし　XML 文書に対するアクセス制御は，利用者の ID やロールによって各ノードに対するセキュリティレベルが異なる．本研究では，ロールごとのアクセス判定のテーブルを作成し，その最適化を考える．一般的に，アクセス制御に伴うポリシーの記述力とアクセス判定時間はトレードオフの関係である．そこで，ロールごとのアクセス判定テーブルを作成することで判定時にかかる時間を削減し，さらにテーブルの簡略化を考察する．具体的には，XML 文書の要素に対する同じアクセス判定の連続性を利用しテーブルの簡略化をする．

キーワード　XML, アクセスコントロール，セキュリティ

A Study on Efficient Access Control for XML Ducuments

Naotake KITAGAWA† and Masatosi YOSHIKAWA††

† Graduate School of Information Science, Nagoya Univercity
†† Information Technology Center, Nagoya Univercity
Furo-cho Chikusa Naogya 464-8601
E-mail: †kitagawa@dl.itc.nagoya-u.ac.jp, ††yosikawa@itc.nagoya-u.ac.jp

Abstract　In access control for XML documents, the security level for particular XML nodes differs depending on user's id or role. In this paper, we propose an introduction of Policy Tables for each role, and discuss optimization of the tables. In general, there is a trade-off between the power of policy description and the time required for access judgment. We discuss two approaches to the reduction of time for access judgment. The first one is the creation of Policy Tables beforehand. The second approach is a simplification of Policy Tables by exploiting the consecutive property of judgment result for XML elements.

Key words　XML,Access Control,Security

1. はじめに

近年，情報セキュリティの重要性が高まっており，XML の普及と相まって，XML 文書に対するアクセス制御が重要となっている [1] [2] [6]. XML 文書に対するアクセス制御は，XML のノード単位でのアクセス制御が可能であり，セキュリティレベルはノード単位で異なる．また，XML 文書中のテキストノードの値や属性値に依存した判定をすることも可能である．アクセス制御におけるポリシーとは一般的に XACML(eXtensible Access Control Language) [4] や XACL(XML Access Control Language) [5] などのポリシー記述言語で定義される．ポリシー内の構成は，利用者 ID などのサブジェクト，対象となるノードのオブジェクト，read，write などのアクションで定義される．ポリシー記述の問題点は，複雑なポリシー記述が可能である反面，ポリシー記述が複雑になるとアクセス判定時間が長くかかることである．ポリシーをフィルタリングすることでロールごとの，アクセス判定テーブルを生成する研究はこれまでも多く行われてきた．しかし，この手法は，アクセス判定テーブル自体の冗長性と，ロールの数に比例して，アクセス判定テーブルを用意しなければならない問題がある．

本研究の目的は，アクセス判定実行時のオーバーヘッドを軽減するとともに, ポリシー記述を簡略化することである．

本研究では, アクセス制御ポリシーをフィルタリングすることでポリシーテーブルを生成する．フィルタリングすることで, ポリシーテーブルが一度生成されれば, 実行時に複雑なアクセス判定を行う必要がなくなる．ポリシーテーブルは利用者のロールごとに生成する [3]. ポリシー記述は, 対象となる XML 文書の要素数に応じて大きくなる．そこで, ポリシーテーブル内でのポリシー記述の簡略化を考察する．具体的には，XML 文書の

```
<Karte>
  <patient>
    <patient_name>Bob</patient_name>
    <doctor_name>Sam</doctor_name>
    <age>24</age>
    <comment>
      <disease_name>・・・・・・・・</ disease_name>
      <condition_for_patient>・・・・・・</condition_for_patient>
      <condition_for_doctor>
        <plan>・・・・・・・</plan>
        <effect>・・・・・・・・</effect>
      </condition_for_doctor>
    </comment>
  </patient>
</Karte>
```

図 1　XML 文書例

要素に対する同じアクセス判定の連続性を用いてポリシーテーブルの冗長性を解決する．
また，アクセス制御ポリシーをフィルタリングすることで生成する，ポリシーテーブルを融合することで，ロールごとに必要なポリシーテーブルの数を大幅に少なくする．

以下，２節，３節でポリシー記述言語を XACML を例にして紹介し，ポリシーのフィルタリングを考える．４節，５節でポリシーのフィルタリングからポリシーテーブルを生成し，その簡略化を考える．６節でポリシーテーブルの融合を行い，最後に考察を行う．

2.　ポリシー記述言語

ポリシー記述言語として XACML を紹介する．XACML ではポリシーを定義することができる．ポリシーは対象となる資源に対して, 誰が (subjects), どこに (resources), どのような (actions) アクセスを許可されるのかということを定義したものである．以上の三つの要素に関して以下に説明する。

- Subjects：複数の Subject を指定することができる．アクセスを要求する主体の属性と，ルール内に定める主体の条件を比較できるように，主体のロールなどを記述する．要求する主体の属性とルールに記述されている属性を比較するための比較関数も用意され，それも指定しなければならない．
- Resources：複数の Resource を指定することができる．アクセスを要求する資源の位置と，ルール内に定める資源の位置を比較できるように，資源の位置を記述する．比較するための比較関数も指定しなければならない．
- Actions：複数の Action が指定できる．要求するアクセスに対する動作とルール内に定める資源に対する動作を比較できるように，資源に対する動作を記述する．XACL では，read,write など限られた動作しか記述できなかったが，XACML では，動作に対する記述の制約はない．しかし，本研究では，Action は read のみと仮定する．

本研究で用いる XML 文書例とポリシーの簡単な XML 記述を**図 1** と**図 2** に示す．
ポリシー記述で，XML 文書のノードに対するアクセス制御を定義することが可能である．しかし，一つのノードに対して複数のルールが見つかる場合がある．このようなルールの衝突が起こった場合の結合アルゴリズム (rule-combining algorithm) が

```
<Policy>
  <Target>
    このポリシーが適用される範囲
  </Target>
  <Rule>
    <Target>このルールが適用される範囲
      <Subjects>
        誰が
      </Subjects>
      <Resources>
        資源のどこに
      </Resources>
      <Actions>
        どんなことを
      <Actions>
    </Target>
  </Rule>
  <Rule>
    :
    :
  </Rule>
</Policy>
```

図 2　ポリシーの XML 記述

以下のようにある．

- Permit-overrides(許可優先)：

ポリシー内のルールについて一つでもルールの Effect が Permit ならばアクセスを許可する．

- Deny-overrides(拒否優先)：

ポリシー内のルールについて一つでもルールの Effect が Deny ならばアクセスを拒否する．

- First-applicable(先行優先)：

ポリシー内のルールを順番に評価して，はじめに出てきたルールを適用する．

以上のような結合アルゴリズムが定義され，これらの結合アルゴリズムに従ってポリシーを評価する．**図 1** に対するポリシー例を以下のように定める．

- ルール１：ロール名が patient である subject は，patient 要素の子要素である patient_name , doctor_name, age の各要素に対してアクセスできる．
- ルール２：ロール名が patient である subject は，comment 要素の子要素である disease_name, condition_for_patient の各要素に対して，age 要素のテキストの値が 18 以上ならばアクセスできる．
- ルール３：ロール名が patient である subject は，comment 要素の子要素である condition_for_doctor 要素以下の部分木の要素に対してアクセスできない．

上記のルール３に対しての XACML 記述を**図 3** に示す．
ルール 1，ルール２に対しても**図 3** のように XACML 記述が可能である．一般的には複数のルール間の衝突が起こるので，ポリシーの全てのルールを見なければ判定できない．アクセス判定実行時間はポリシーのルール数に比例して大きくなる．このような問題点の解決策として次節で，ポリシーのフィルタリングを挙げる．

3.　ポリシーのフィルタリング

リソースに対するアクセス判定実行時のオーバーヘッドには厳しい要求がある．2 節で述べたようなポリシー記述を用いての

```
<?xml version=1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
http://www.oasis-open.org/tc/xacml/1.0/cs-xacml-schema-policy-01.xsd"
PolicyId="identifier:example:SimplePolicy"
RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-overrides">
<Description>
   Karte access control policy
</Description>
<Target>
   <Subjects>
         <Anysubject/>
   </Subject>
   <Resources>
         <AnyResources/>
   </Resources>
   <Action>
         <AnyAction/>
   </Action>
</Target>
<Rule
  RuleId="urn:oasis:names:tc:xacml:1.0:example:SimpleRule"
  Effect="Deny">
<Description>
  An patient cannot read element of condition-for-doctor.
</Description>
<Target>
  <Subjects>
    <!--mach role subject attribute -->
    <SubjectMatch
     MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue
        DataType="http://www.w3c.org/2001/XMLSchema#string">patient
       </AttributeValue>
       <SubjectAttributeDesignator AttributeId=
        "urn:oasis:names.tc:xacml:1.0:example:attribute:role"
         DateType="http://www.w3.org/2001/XMLSchema#string"/>
    </SubjectMatch>
  </Subjects>
  <Resources>
   <Resource>
     <ResourceMatch
        MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-mach">
          <AttributeValue>
          /Karte/patient/comment/condition_for_doctor </ AttributeValue>
           <ResourceAttributeDesignator AttributeId=
            "urn:oasis:names:tc:xaml:1.0:resource:xpath"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
     </ResourceMatch>
    </Resource>
  <Resources>
  <Actions>
   <Action>
     <!--match 'read' action-->
     <ActionMatch
      MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
         DateType="http://www.w3.org/2001/XMLSchema#string">
           read
        </AttributeValue>
        <ActionAttributeDesignator AttributeId=
        "urn:oasis:names:tc:xacml:1.0:function:action:action-id"
        DataType="http://www.w3.org/2001/XMLScema#string/">
     </ActionMatch>
   </Action>
  </Actions>
 </Target>
</Rule>
```

図 3　ポリシーの XACML 記述例

アクセス判定では実行時のオーバーヘッドが大きくなってしまう．そこで，ポリシーを実行時前にフィルタリングしておくことで，アクセス判定実行時間は減少できると考えられる．

図 4 は，ポリシーのフィルタリングを表したアーキテクチャである．元のポリシーから，利用者のロール (**図 4** では，A,B,C ・・・) ごとのテーブルを生成する．従って，ロールの数に比例した

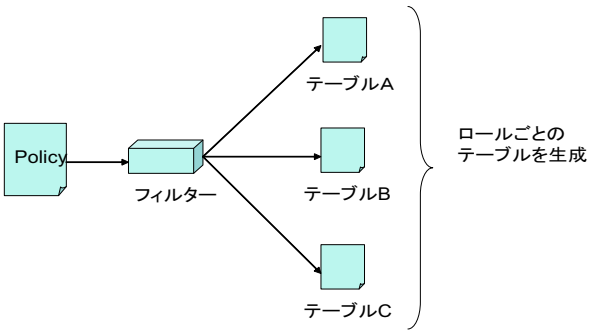

図 4　ポリシーのフィルタリング

経路	経路番号
/Karte	1
/karte/patient	2
/karte/patient/patient_name	3
/karte/patient/patient_name/text	4
/karte/patient/docotor_name	5
/karte/patient/docotor_name/text	6
/karte/patient/age	7
/karte/patient/age/text	8
/karte/patient/comment	9
/karte/patient/comment/disease_name	10
/karte/patient/comment/disease_name/text	11
/karte/patient/comment/condition_for_patient	12
/karte/patient/comment/condition_for_patient/text	13
/karte/patient/comment/condition_for_doctor	14
/karte/patient/comment/condition_for_doctor/plan	15
/karte/patient/comment/condition_for_doctor/plan/text	16
/karte/patient/comment/condition_for_doctor/effect	17
/karte/patient/comment/condition_for_doctor/effect/text	18

図 5　経路情報テーブル

テーブルが必要になる．本研究では生成するテーブルをポリシーテーブルと呼ぶ．

4.　ポリシーテーブルの生成

アクセス制御を定義したポリシーをフィルタリングすることで，ポリシーテーブルを生成する．ポリシーテーブルは，XML 文書のノードを表す"pathID"，ノードに対する認可である"判定"，対象となる XML 文書の内部情報による判定の場合の"条件"を属性としてもつ．

pathID

アクセス対象となる XML 文書の各ノードを特定するために，DTD などのスキーマ情報から経路情報を得て，それぞれの経路に番号をつける．そのテーブルを**図 5** に示す．その番号を pathID とする．**図 1** の XML 文書をラベリングした木の状態を**図 6** に示す．

判　　定

アクセス要求されたノードに対する許可，拒否などの判定を表す．判定には以下の三つが考えられる．

（1）　許可 (+) アクセス要求されたノードに対してアクセスを許可する．

（2）　拒否 (-) アクセス要求されたノードに対してアクセスを拒否する．

（3）　条件付許可 (?) ポリシーテーブルの条件属性の条件に適合すれば許可する．

XML 文書に対するアクセス制御では，文書の内部情報に依存した判定が可能である．例えば**図 1** では，age 要素の値が age ＞

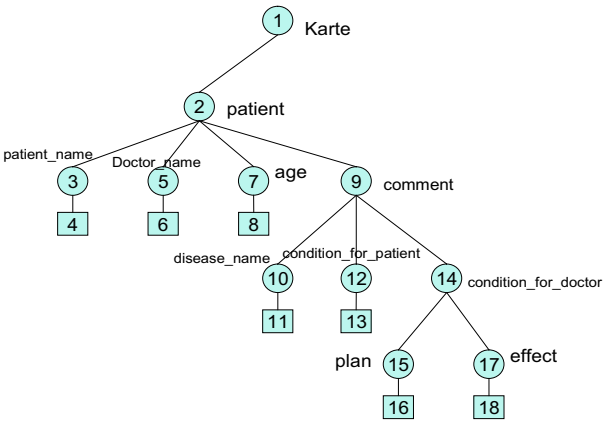

図 6　ノードのラベリング

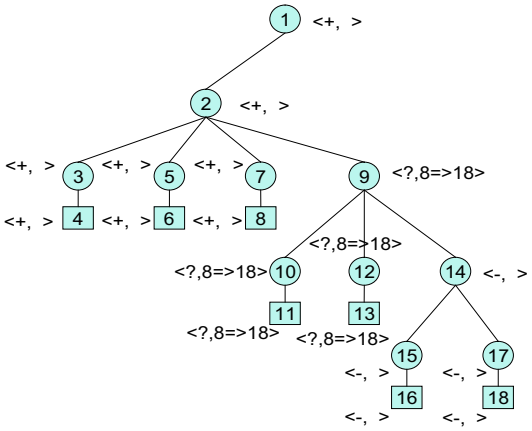

図 7　ノードに対するアクセス判定

=18 ならばアクセスを許可するといったことである．上記に挙げた条件付許可は，ポリシーをフィルタリングする時に同時に条件に適合するか否か調べることも可能である．しかし，XML 文書の内部情報は変更なども多く変更の度にフィルタリングしてポリシーテーブルを生成しなければならない．そこで，判定に条件付許可を加えることで文書のテキストノードなどの変更であれば，新しくポリシーテーブルを生成する必要がなくなる．

条　　件

アクセス判定において判定に条件が加わる場合がある．例えば，アクセス対象となる XML 文書の内部情報に依存した判定，アクセスする時刻による判定，アクセスする場所による判定などが考えられる．本研究では，XML 文書の内部情報に依存した判定について考える．ポリシーテーブルの条件属性には，条件の対象となるノードの番号と条件の比較となる値で構成する．”比較対象となるノードの番号 op 条件キー”(op は演算子) と表す．例えば，**図 1** の XML 文書で，年齢が 18 以上と表す条件ならば，”8 ＞=18”と表す．

4.1　ポリシーテーブル

図 1 の XML 文書のノードに対するアクセス判定を**図 7** に示す．**図 7** では，**図 1** での各ノードに対する，アクセスの許可，拒否，条件付許可を表し，条件付許可の場合はその条件も表している．各ノードに対して”＜判定, 条件＞” で表す．判定結果をポリシーテーブルに格納する．判定が格納されたポリシーテーブルを**図 8** に示す．

pathID	判定	条件
1	+	
2	+	
3	+	
4	+	
5	+	
6	+	
7	+	
8	+	
9	?	8>=18
10	?	8>=18
11	?	8>=18
12	?	8>=18
13	?	8>=18
14	-	
15	-	
16	-	
17	-	
18	-	

図 8　ポリシーテーブル

pathID	判定	条件
1	+	
9	?	8>=18
14	-	

図 9　簡略化されたポリシーテーブル

5.　ポリシーテーブルの簡略化

４節で示したポリシーテーブルは，利用者のロール数に比例して多くなる．また，各ロールのポリシーテーブルも対象となる XML 文書のノード数に比例して大きくなる．そこで，ポリシーテーブルの簡略化を考える．

XML 文書のノードを深さ優先でたどると，同じ判定結果が連続することが多い．それは，対象となる要素とそれを根とする部分木が同じ判定結果を持つことが多いからである．**図 8** でも，pathID の番号が１～８まで，９～13 まで，14～18 までは，それぞれ許可，条件付許可，拒否と同じ判定が続く．同じ判定結果が続くことを利用してポリシーテーブルの簡略化をする．

簡略化は，同じ判定結果が続く pathID は，番号が最小の pathID を残し，残った pathID 以下のタプルは省略する．同じ判定結果が連続する場合は，ポリシーテーブルの大きさを大幅に削減できる．**図 8** のポリシーテーブルを簡略化したポリシーテーブルを**図 9** に示す．

図 9 において，ノードのアクセス要求として pathID の番号が７，11 のノードが要求された場合を考える．

- ７が要求された場合･･･ pathID には７がないので，pathID の値が７よりも小さい最大の値を探す．そのような pathID は１であるので，pathID が１である組の判定に従って判定する．判定結果は許可である．
- 11 が要求された場合･･･ pathID には 11 がないので，pathID 値が 11 よりも小さい最大の値を探す．そのような pathID は 9 であるので，pathID が 9 である組の判定に従って判定する．判定結果は条件付許可である．

これまでは，各ロールごとのテーブルを作成し，その簡略化について考えてきた．しかし，問題点として，**図 4** を見ても分かるように，ロールの数に比例した，ポリシーテーブルが必要になる．各ルールの一つ一つのテーブルだけではデータの大きさも大きくないが，特にロールの数が多い場合，ロール対応する全てのポリシーテーブルのデータの大きさは非常に大きくなる．

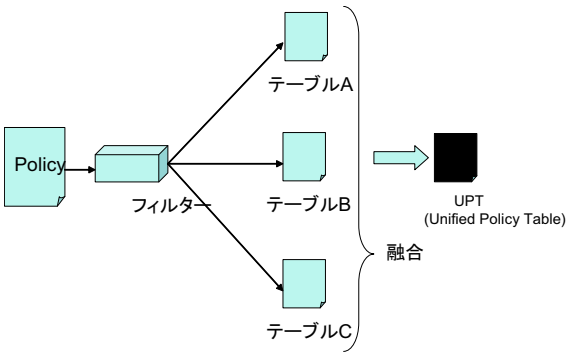

図 10　ポリシーテーブルの融合

pathID	判定
1	+
14	-

ポリシーテーブル : patient

pathID	判定
1	+
10	-
14	+

ポリシーテーブル : doctor

pathID	判定
1	+
9	-

ポリシーテーブル : receptionist

pathID	判定
1	+
5	-
7	+
12	-
14	+
17	-

ポリシーテーブル : druggist

図 11　各ロールのポリシーテーブル

このような問題点の解決策として，ロールに対応したポリシーテーブルの融合を次節で考える．

6. ポリシーテーブルの融合

アクセス制御ポリシーをフィルタリングしたときの，ロールの数に比例してテーブルの数が多くなる問題を解決する．上記でも述べてきたポリシーのフィルタリングで生成された各ロールのポリシーテーブルを一つのテーブルに融合する．融合してできたポリシーテーブルを UPT(Unified Policy Table) とする．ポリシーテーブルの融合を**図 10** に示す．

図 1 の XML 文書にアクセスするロールを，patient, doctor, receptionist, druggist の以上４種類とする．各ロールに対するポリシーテーブルを**図 11** に示す．ただし，融合されるポリシーテーブル内のアクセス判定は，許可 (+)or 拒否 (-) とする．内部情報に依存した，条件付判定 (?) を UPT に取り入れることは今後の課題にする．

はじめに，ロールテーブルを生成し，生成されたロールテーブルと各ロールに対応したポリシーテーブルを用いて UPT を生成する．

以下に，ロールテーブルと UPT の生成を説明し，UPT の実行時の動作を説明する．

6.1 ロールテーブルの生成

各ロールに対応したそれぞれのポリシーテーブルを融合するために，ロールテーブルを生成する．ロールテーブルのスキーマを以下に定義する．

```
role_table(role_name, rID)
```

role_name	rID
patient	2
doctor	3
receptionist	5
druggist	7

図 12　role_table 例

pathID	アクセス可能なロール
1	patient, doctor, receptionist, druggist
2	patient, doctor, receptionist, druggist
3	patient, doctor, receptionist, druggist
4	patient, doctor, receptionist, druggist
5	patient, doctor, receptionist
6	patient, doctor, receptionist
7	patient, doctor, receptionist, druggist
8	patient, doctor, receptionist, druggist
9	patient, doctor, druggist
10	patient, druggist
11	patient, druggist
12	patient
13	patient
14	doctor, druggist
15	doctor, druggist
16	doctor, druggist
17	doctor
18	doctor

図 13　ポリシーテーブルの重ね合わせ

ロールテーブルの，role_name,rID の二つの属性について説明する．

- role_name :XML 文書に対してアクセスを要求する利用者の各ロールのロール名である．本研究では，**図 1** の XML 文書にアクセスするロールとして，patient, doctor, receptionist, druggist を用いる．
- rID :ロール ID として，数字を入力する．使用する数字は素数で，一意に定まるものとする．

ロールが，patient, doctor, receptionist, druggist のロールテーブルを**図 12** に示す．

6.2 UPT の生成

各ロールのポリシーテーブルと，ロールテーブルから UPT を生成する．はじめに，**図 11** の各ポリシーテーブルをまとめると**図 13** のようになる．次に，UPT のスキーマを以下に定義する．

```
UPT(pathID, アクセス判定番号)
```

UPT で定めた，二つの属性について説明する．

- pathID :XML 文書のノードを示す番号.
- アクセス判定番号 :アクセスの許可が与えられた，ロールの rID の積で表す.

例１：**図 11** のテーブルで，pathID が 1 であるノードは，patient, doctor, receptionist, druggist にアクセスが許可されている．**図 12** で，patient, doctor, receptionist, druggist のそれぞれの rID は，2, 3, 5, 7 なので，アクセス判定番号は，2*3*5*7=210 となる．

例２：pathID が 10 であるノードは，patient と druggist にアクセスが許可されているので，それぞれの rID の，2 と 7 の積をして, アクセス判定番号は 21 となる．

生成された UPT 例を**図 14** に示す．

pathID	アクセス判定番号
1	210
2	210
3	210
4	210
5	30
6	30
7	210
8	210
9	42
10	14
11	14
12	2
13	2
14	21
15	21
16	21
17	3
18	3

図 14　ＵＰＴの例

6.3　UPT 実行時の動作

UPT を用いた実行時のアクセス判定の方法に関して述べる．ある利用者が XML 文書のある特定のノードに対して，アクセスが発生したとする．利用者の情報としては，利用者自身のロールとアクセスしたいノード番号が与えられる．与えられたロールを user_role とする．まず，**図 12** のようなロールテーブルの role_name に user_role が存在するか確かめる．存在した場合の role_name に対応する rID をロール ID とする．次に，利用者が要求するノード番号が，UPT の pathID に存在するか調べる．存在すれば，その pathID に対応する，アクセス判定番号を accessNum とする．最後に，ロール ID と accessNum を用いて以下のようにアクセス判定を判定する．

（1） アクセス許可の場合：accessNum mod ロール ID = 0 となる場合

（2） アクセス拒否の場合：accessNum mod ロール ID ≠ 0 となる場合

例として，ロールが patient の利用者が pathID が 4 と 16 のノードに対してアクセス要求があった場合を考える．また，この例を**図 15** に示す．

- pathID=4 が要求された場合

ロールテーブルの role_name が存在するので，rID=2 とする．

UPT の pathID に，利用者が要求する pathID=4 が存在するので，accessNum=210 とする．

210 mod 2 = 0 なので，アクセスを許可する．

- pathID=16 が要求された場合

ロールテーブルの role_name が存在するので，rID=2 とする．

UPT の pathID に，利用者が要求する pathID=16 が存在するので，accessNum=21 とする．

21 mod 2 = 2 なので，アクセスを拒否する．

7.　考　　察

本研究で考案したポリシーテーブルは，リソースとなる XML 文書の内部情報であるテキストノードの値変更では，テーブルを変更する必要がない．しかし，リソースとなる XML 文書の

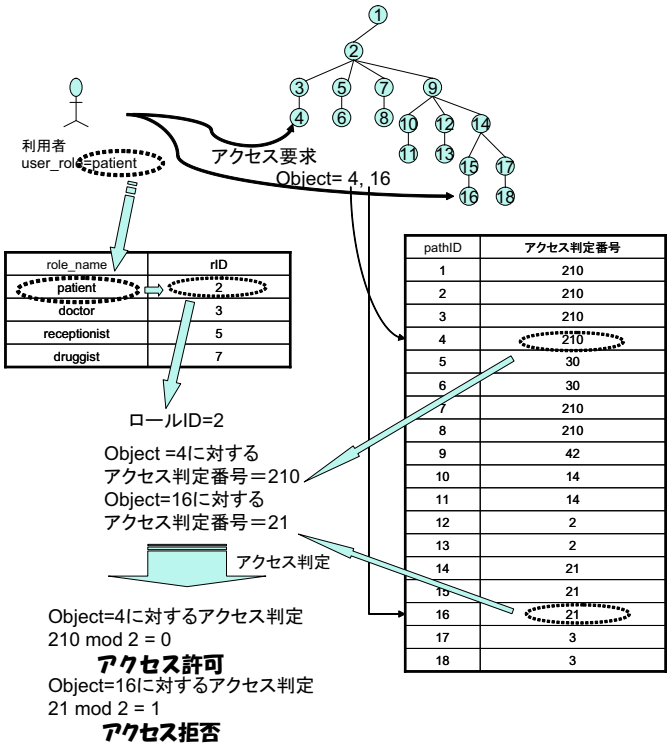

図 15　HPT 実行例

構造的変更は，同時にポリシーテーブルも変更しなければならない．本稿では，ラベリングを XML 文書のノードの経路順に番号をつけたが，番号のつけ方でポリシーテーブルの変更を最小限なものにすることも可能であると思われる．この問題は今後の課題である．

8.　ま　と　め

本稿では，アクセス判定時間を削減するために，ポリシーテーブルを生成し，ポリシーテーブルの簡略化を考察した．資源となる XML 文書の要素のアクセス判定結果を深さ優先調べると連続することが多い．連続する場合にはポリシーテーブルが簡略化されることを示した．

また，生成されたポリシーテーブルを融合することで，フィルタリングする際の問題点である，ロールの数に比例してポリシーテーブルが必要になることを解決した．今後は考察でも挙げたように，ポリシーテーブルの再利用性の向上と，さらに効率的な簡略化についても考察する予定である．

文　　献

[1] 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．電子情報通信学会技術研究報告.DE2004-27(2004-07)

[2] 戚 乃，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．電子情報通信学会技術研究報告.DE2004-28(2004-07)

[3] Davit F Ferraiolo, Janet A Cugini, D Rivhard Kuhn. Role-Based Access Control(RBAC):Features and Motivations. National Institute of Standard and Technology U.S.Department of Commerce Gaithersburg MD 20899

[4] OASIS.eXtensible Access Control Markup Laguage(XACML)Version 1.0.OASIS Standard,18 February2003 http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf

[5] XML Access Control Language:Provisional Authorization for XML Documents 2002 http://www.trl.ibm.com/projects/xml/xacl/xacl-spec

.html
[6] 鈴木 優一.Web サービスとセキュリティ-PKI と PMI を融合させる次世代言語 XACML. http://www.atmarkit.co.jp/fsecurity/rensai/webserv05/webserv01.html 2002