DEWS2006 6B-i8

粒度が細かいアクセス制御のロールの扱いと高速化に関する研究

北川　直毅†　　吉川　正俊††

† †† 名古屋大学情報科学研究科　〒 464–8601 愛知県名古屋市千種区不老町名古屋大学情報科学研究科
E-mail: †kitagawa@dl.itc.nagoya-u.ac.jp, ††yosikawa@is.nagoya-u.ac.jp

あらまし　目的外の情報の使用を抑止するために，アクセス制御は必須である．データの交換性の高さなどから，XML 文書が利用されることが多く，XML 文書の要素単位での粒度の細かいアクセス制御が求められる．さらに，主体も，ID だけでなく，ロールやグループによるアクセス制御が求められる．しかし，粒度の細かいアクセス制御は，ポリシー記述の複雑さや，アクセス判定にかかる時間が大きくなってしまう問題がある．本研究では，医療電子カルテを例に，粒度の細かいロールの扱いと，アクセス判定の高速化に関して考察する．具体的に，複数のロールをグループに分けることで，ポリシー記述を単純化し，アクセス頻度を考慮することで，アクセス判定を高速化する．
キーワード　アクセス制御，XML，XACML

A Study on Role Management and Performance Optimization for Fine Grained Access Control

Naotake KITAHGAWA† and Masatoshi YOSHIKAWA††

†　Nagoya University　Furochou, chikusa-ku, aichi, 464–8601 Japan
E-mail: †kitagawa@dl.itc.nagoya-u.ac.jp, ††yosikawa@is.nagoya-u.ac.jp

Abstract　The access control is needed with the spread of Internet and enforcement of a law to avoid accessing the data for other purposes. In access control for XML (Extensible Markup Language) documents, the security level for particular XML nodes differs depending on users ID or role. In XML documents, fine grained access control is needed. In medical field, the care card system needs the control of access. The fine grained access control for care cards is required because care cards contain personal information and users who access care card have various roles. However, in fine grained access control, there are problems that policy description becomes complex and access decision time becomes longer. In this paper, we consider how to hold role to users to simplify the policy description, and generate the Policy Table using access frequency to reduce the access decision time.
Key words　Access control, XML, XACML

1. はじめに

近年，情報セキュリティは非常に重要な分野となっている．情報セキュリティが重要になった背景として，インターネットの普及など情報技術の向上や，個人情報保護法など法律の施行などが挙げられる．情報セキュリティの中で，アクセス制御は，目的外の情報の使用を抑止するために必要不可欠な技術である．アクセス制御は，Resource, Subject, Action の三つの要素で考える．本研究では，Resource として，XML 文書に対するアクセス制御を考える．XML 文書は，データの交換性に優れており，データにタグを付けることで，文書全体ではなく，要素単位のアクセス制御が可能である．アクセス制御には，Resource に対して，非常に粒度が細かいアクセス制御が求められる．Subject は，ID だけでなく，ロールといった，役割の集合を考える．Action は，最も基本的な read, write, delete の三つを考える．

しかし，Resource の粒度が細かくなったり，Subject であるロールの種類が多くなると，アクセス制御の許可，拒否の判定にかかるオーバーヘッドが大きくなり，さらにアクセス制御ポリシーの記述が複雑になる問題がある．

これまでもアクセス制御の高速化に関する研究は行われてきた [6] [5]．本論文では，アクセス制御の判定にかかるオーベーヘッドが大きくなる問題点に対して，アクセス頻度を利用することでアクセス制御を高速化する．さらに，アクセス制御ポリシーの記述が複雑になる問題点を解決するために，ロールの種類が非常に多くなった場合に，効率よくロールを扱う方法を考察する．

本研究では，医療分野の電子カルテを例に説明する．2 節で，

```
<Policy>
  <Rule>
    <Subjects>Alice</Subjects>
    <Resources>/home/Alice</Resources>
    <Action>read</Action>
  <Rule/>
  <Rule>
     ..........................
  </Rule>
</Policy>
```

図 1　XACML 基本記述例

粒度の細かいアクセス制御を可能にするポリシー記述言語である XACML [2] に関して説明する．さらに，医療分野での電子カルテ共通のデータ構造を規格化した，MML [1] について，特にアクセス制御に関して説明する．3 節で，ロールの種類が非常に多くなった場合のロールの扱い方に関して，医療電子カルテを例に説明する．4 節で，アクセス制御を高速化するために，アクセス頻度を考慮したテーブルの生成に関して述べる．

2. 関 連 研 究

以下にポリシー記述言語である XACML (Extensible Access Control Markup Language) の説明をし，医療分野で電子カルテの標準フォーマットとして規格化された MML (Medical Markup Language) の説明をする．

2.1 XACML

ポリシー記述言語である XACML [2] を紹介する．2005 年 2 月に XACML は OASIS 標準として承認された．XACML では，アクセス制御ルールの記述に関する部分 (ポリシー記述言語) と，認可処理要求とそのやり取りに関する部分 (コンテキスト) の大きく二つの機能がある．ここでは，ポリシー記述言語としての XACML を紹介する．XACML は XML [3] で記述され，ポリシーは複数のルールによって構成されている．ルールは主に以下の三つの要素で構成される．

- Subject: ルールの対象となる主体（ユーザ ID，グループ名，ロールなど）
- Resource: 対象となる資源の位置（XPath 式 [4] など）
- Action: 対象となる資源への処理内容（read，write，delete など）

図 1 は, XACML の基本記述例である．はじめのルールでは，Alice が，資源 /home/Alice に対して読み込みが可能であることを示す．以下に，複数のルールで構成されたポリシー記述を示す．

ルール：外科の医者はすべての情報を read 可能だが，内科医は，手術記録に関しては read できない（手術記録以外は read 可能）

```
<Policy rule_comb_algr = Deny_override>
  <Rule  Effect = Permit>
    <Subjects>外科，内科</Subjects>
    <Resources>all</Resources>
    <Actions>read</Actions>
  </Rule>
  <Rule  Effect = Deny>
    <Subjects>内科</Subjects>
    <Resources>カルテ/手術記録</Resources>
    <Actions>read</Actions>
   </Rule>
</Policy>
```

ポリシー記述で，XML 文書の要素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，一つの要素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．XACML は非常に柔軟な記述力を持ち，粒度が細かいアクセス制御に対しても対応することが可能である．しかし，粒度が細かいアクセス制御のポリシーを記述すると，記述が複雑になり，ルール数も多くなる．アクセス判定をするには，通常複数のルールを参照することが多く，ルールの衝突を結合アルゴリズムによって解消し評価する．XACML のポリシー記述によるアクセス制御では，ポリシー記述が複雑になる可能性があり，アクセス判定にかかるオーバーヘッドが大きくなる問題が考えられる．

2.2 電子カルテの現状

現在，電子カルテはさまざまな企業によって開発されている．電子カルテの導入により，医療コストの削減，保険点数の透明化，遠隔治療への可能性などさまざまな利点がある．現時点で電子カルテの普及率は中核病院で 20 %，開業医なども含めた病院全体 (約 9000 病院) では 10 %未満となる．厚生労働省は，2007 年までに，電子カルテの普及率を 60 %にする目標を掲げている．

電子カルテは，今までの紙面でのカルテと比較して，電子データであるので，権限さえ持っていれば院内の端末のどこからでもアクセスすることができる．患者が，今までと違った診療科で受診する際，紙面のカルテの場合，新しい診療科にカルテを持っていく必要があるが，電子カルテは必要ない．しかし，このような利便性がある反面，患者の個人情報を保護するために，セキュリティを強化する必要がある．2005 年に個人情報保護法が施行され，今まで以上に患者の個人情報の保護は急務である．

しかし，電子カルテのアクセス制御を強化するには，いくつかの問題がある．電子カルテの現状として，ある病院の電子カルテシステムのアクセス制御とそのの問題点を調査した．

ある病院の電子カルテシステムにアクセスする利用者は，医師に限らず，病院関係者全てである．アクションは大きく read, write, オーダーの大きく三種類である．はじめに read に関するアクセス制御を述べる．read は，医師，看護師，事務員といった役職に関わらず，病院関係者のほとんどがカルテ全体に対して許可されている．次に，write は，一部のロール，薬剤師，栄養士のみ薬剤療法，栄養指導記録の欄に情報を書き込むことが可能である．つまり，看護師や医学部学生であっても診療記録など本来医師が書き込むべき要素に情報を書き込むことができる．最後に，オーダーは，実際の抗がん剤投与など診療命令であるので，医師のみが許可されている．このようなアクセス制御では，患者の個人情報を保護できないどころか，悪意をもった利用者のカルテ改竄など，医療内容の低下にもつながる．これらの問題を解決するために，粒度が細かいアクセス制御が求められる．例えば，従来は医師としてロールが定義されていたのを，内科の医師，外科の医師とさらに細かく定義し，内科の医師は内科の患者のカルテしかアクセスが許可されないようなポリシーを構築することである．しかし，これには医療現場独自の問題がある．医療現場では，患者はさまざまな診療科に掛かることが多く，診療科毎のアクセス制御をするには，患者が掛かる診療科が変わるたびにポリシーを書き換えなければならない．これにより，管理者のシステムの運営が非常に困難になる．

アクセス制御の粒度を高くするには問題が多いが，患者の個人情報に関する認識も高くなり，今後は情報の保護に関する法律も強化されると考えられる．この点からも，粒度が細かいアクセス制御は必要不可欠であると考えられる．

2.3 MML

医療カルテの異なったドメイン間での交換性を高めるための規格として，MML (Medical Markup Language) [1] がある．VPN (Virtual Private Network) を用いることで異なった医療ドメイン同士のカルテの交換を可能にしている研究もある [7]．MML では電子カルテのデータ構造を定義し，共通のデータ構造を持つことで，異なった病院間でのカルテの交換性を高めている．ここでは，MML のアクセス権の定義について述べる．**図 2** は MML のデータ構造である．MML で定義できるアクセス権を以下に示す．

- Resource: 一文書（患者一人の情報）
- Subject: 施設単位，診療科単位，職種単位，個人単位
- Action: read (参照)，write (書き込み)，delete(削除)，all(参照，書き込み，削除の全て)

MML アクセスポリシー記述例

MML で記述するポリシー記述例を以下に示す (MML3.0 規格書より抜粋)．

ルール：宮崎医科大学附属病院の内科医師に 2001 年 10 月 1 日から 2001 年 12 月 31 日まで参照，修正，削除の権利を与える場合

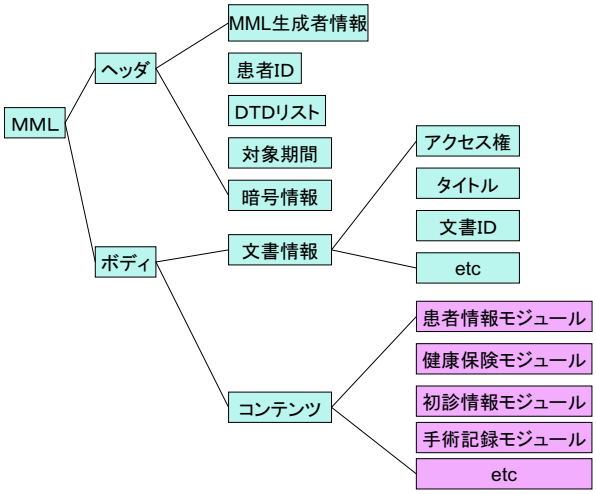


図 2　MML データ構造

Action と利用期間の記述

```
<securityLevel>
    <accessRight
        permit="all"
        startDate="2001-10-01"
        endDate="2001-12-31">
```

subject の記述

施設単位

```
<mmlSc:facility>
  <mmlSc:facilityName
    mmlSc:facilityCode="individual"
    mmlSc:tableId="MML0035"
    mmlSc:facilityId="JPN4530101000003"
    mmlSc:facilityIdType="JMARI">
    宮崎医科大学附属病院
  </mmlSc:facilityName>
</mmlSc:facility>
```

診療科単位

```
<mmlSc:department>
  <mmlSc:departmentName
    mmlSc:departmentCode="01"
    mmlSc:tableId="MML0028"/>
</mmlSc:department>
```

職種単位

```
    <mmlSc:license>
      <mmlSc:licenseName
       mmlSc:licenseCode="doctor"
       mmlSc:tableId="MML0026"/>
    </mmlSc:license>
  </accessRight>
</securityLevel>
```

MML でのアクセス権は，施設単位，診療科単位など粒度が高いアクセス権を定義することができるが，対象となる資源が一文書単位である．一文書単位では，職種単位などで分ける利点

```
<data>
  <Care_Card>
    <name>Bob</name>
    <address>Aichi chikusa</address>
    <tell>012345</tell>
    <health_insurance_number>777</health_insurance_number>
    <medical_department>surgery</medical_department>
    <doctorID>245786</doctorID>
    <medical_plan>.....</medical_plan>
    <medical_result>.....</medical_result>
    <drug_info>.....</drug_info>
    <X-ray_ picture >.....</X-ray_ picture >
    <operative_records>
      <operation_medical_department>surgery</operation_medical_department>
      <chairmanID>129876</chairmanID>
      <stuffID>098765</narcosis_>
      <narcosis_stuffID>987098</narcosis_stuffID>
      <operative_processs>.....</operative_process>
      <narcosis__record>.....</narcosis__record>
    </operative_records>
  </Care_Card>
  <Care_Card>
    <name>Alice</name>
    <address>Gifu gifushi</address>
    <tell>43586</tell>
    <health_insurance_number>333</health_insurance_number>
    <medical_department>internal</medical_department>
    ·····
    ·····
  </Care_Card>
</data>
```

図 3　XML 文書例

がほとんどなく，対象となる資源を要素単位で定義する必要がある．しかし，要素単位でアクセス制御しようとすると，ポリシー記述が非常に複雑になる．MML のように一文書毎にアクセス権を記述するのは非常に非効率的である．カルテ情報に対するアクセス権のポリシー記述は，一つにまとめて記述する方が効率的であると考えられる．ポリシー記述として XACML が考えられるが，施設単位，診療科単位，職種単位など，粒度が細かい XACML 記述は非常に複雑になる．以下に，ロールの扱いを考慮した，XACML 記述を考察する．**図 3** は，MML を参考にして記述した，電子カルテ XML 記述例である．以下，**図 3** の XML 文書を例に説明する．

3.　ロールの扱い

医療電子カルテにアクセスする利用者はさまざまな立場の人が存在する．一般的に，医療現場では大きく，医者，看護師，薬剤師，事務員など職種でロールを分けていることが多い．実際，ある病院では，職種を，医師，歯科医師，薬剤師，看護師，診療放射線技師等，検査技師等，管理栄養士，歯科衛生士，医学生，看護学生など，非常に細かく分けている．さらに，医師や看護師などが所属する診療科は，血液内科，消化器内科，血管外科，消化器外科１，産婦人科，小児科，皮膚科など２９種類に区分される．また，職種の医師においては，医師の中でさらに，常勤医師，医員・診療従事者，研修医などに区分される．役職においても，部長，副部長，一般医師などに区分されることが考えられる．このように，ロールの種類は非常に多い．実際の医療現場では，さらに細かく分かれているが，例として，以下のロールを定義する．

- 職種: 医師
- 診療科（医師の場合): 外科，内科，小児科，麻酔科
- 役職（医師の場合): 一般医師，研修医，外来医師

以下に，一人のユーザに対して一つのロールを割り当てた場合と，一人のユーザに複数のロールを割り当てた場合を考える．

```
<Rule Effect = "Premit">
  <Subject>外科一般医師</Subject>
  <Resource>/data/CareCard[medical_department="外科"]/operative_record/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>外科一般医師</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>write</Action>
</Rule>
<Rule Effect = "Premit">
  <Subject>内科一般医師</Subject>
  <Resource>/data/CareCard[medical_department="外科"]/operative_record/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>内科一般医師</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>write</Action>
</Rule>
<Rule Effect = "Premit">
  <Subject>小児科一般医師</Subject>
  <Resource>/data/CareCard[medical_department="外科"]/operative_record/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>小児科一般医師</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>write</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>外科研修医</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>外科研修医</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>write</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>内科研修医</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>内科研修医</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>write</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>小児科研修医</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>小児科研修医</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>write</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>麻酔科研修医</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>麻酔科研修医</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>write</Action>
</Rule>
```

図 4　利用者に一つのロールを与えた場合のポリシー記述

3.1　1 ユーザ：1 ロール

一人の利用者に一つのロールを割り当てる．例えば，“麻酔科研修医” というロールを割り当てる．このようにロールを割り当てると，ロールの種類は医師のみを考えても，診療科の４種類，役職の３種類で計 12 種類のロールを扱うことになる．例として以下のルールが与えられた場合のポリシー記述を**図 4** 示す．

（1）　外科の一般医師は，外科に所属する全ての患者の麻酔情報を read することができるが，どの患者の麻酔情報に対しても write できない．

（2）　外科の研修医は，全ての患者の麻酔情報を read も write もできない．

（3）　内科の一般医師は，内科に所属する全ての患者の麻酔情報を read することができるが，どの患者の麻酔情報に対しても write できない．

（4）　内科の研修医は，全ての患者の麻酔情報を read も write もできない．

（5）　小児科の一般医師は，小児科に所属する全ての患者の麻酔情報を read することができるが，どの患者の麻酔情報に対しても write できない．

（6）　小児科の研修医は，全ての患者の麻酔情報を read も write もできない．

（7）　麻酔科の一般医師は，全ての患者の麻酔情報を read も write もできる．

（8）　麻酔科の研修医は，全ての患者の麻酔情報を read も write もできない．

以上のような，ポリシー記述では，ロールの数がさらに増えたときにルールの数が，非常に多くなってしまう．利用者に複数のロールを与える場合のポリシー記述に関して考察する．

3.2 1 ユーザ：多ロール

一人のユーザに複数のロールを割り当てる．ここで，医師，看護師，外科，内科などを，ロールと呼ぶのに対して，ロールをまとめる，職種や診療科をグループロール名（gRole）とする．ユーザに対して，複数のロールを割り当てるとユーザは以下のようなユーザコンテキストを保持する．

UserContext (UserID, gRole1, gRole2, ····) =（ID, role1, role2, ····）

例えば，外科の一般医師ならば，

UserContext (UserID, 職種, 診療科, 役職) =（123, 医師, 外科, 一般）

となる．職種，診療科，役職の gRole に注目すると，3.1 節のルールは以下のように書き換えることができる．

（1） 外科に所属する医師は，外科に所属する全ての患者の麻酔情報を read することができるが，どの患者の麻酔情報に対しても write できない．

（2） 内科に所属する医師は，内科に所属する全ての患者の麻酔情報を read することができるが，どの患者の麻酔情報に対しても write できない．

（3） 小児科に所属する医師は，小児科に所属する全ての患者の麻酔情報を read することができるが，どの患者の麻酔情報に対しても write できない．

（4） 麻酔科に所属する医師は，全ての患者の麻酔情報を read も write もできる．

（5） 研修医は，所属している診療科が許可していても全ての患者の麻酔情報を read も write もできない．

（6） 一般医師は，所属している診療科が許可していれば，全ての患者の麻酔情報を read も write もできる．

上記のルールを反映させたポリシー記述を**図 5** に示す．利用者に複数のロールを与えることで，ロール数が多くなった場合でも，ポリシー記述が単純化できると考えられる．また，ポリシーにルールが追加された場合を考える．追加するルールは以下のルールである．

追加ルール：外来医師は，担当した患者の麻酔情報を read することができるが，どの患者の麻酔情報に対しても write できない．

上記のようなルールが追加された場合，利用者に一つのロールを割り当てる場合，診療科の数に応じてルールを記述しなければならない．しかし，複数のロールを割り当てることで，**図 5** に**図 6** のルールを追加するだけでよい．複数のロールを与えた場合，複数のルールを参照しなければならない．例えば，UserContext (UserID, 職種, 診療科, 役職) =（123, 医師, 外科, 一般) を持った利用者ならば，医師，外科，一般のそれぞれの Subject に対して評価しなければならない．従って，アクセス判定時間が大きくなる可能性がある．この問題に関して，アクセス頻度を考慮したテーブルの生成によってアクセス制御を高速化する手法を提案する．

```
<Rule Effect = "Premit">
  <Subject>外科</Subject>
  <Resource>/data/CareCard[medical_department="外科"]/operative_record/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>外科</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>write</Action>
</Rule>
<Rule Effect = "Permit">
  <Subject>内科</Subject>
  <Resource>/data/CareCard[medical_department="内科"]/operative_record/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>内科</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>write</Action>
</Rule>
<Rule Effect = "Permit">
  <Subject>小児科</Subject>
  <Resource>/data/CareCard[medical_department="小児科"]/operative_record/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>小児科</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Permit">
  <Subject>麻酔科</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Permit">
  <Subject>麻酔科</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>write</Action>
</Rule>
<Rule Effect = "Permit">
  <Subject>一般</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Permit">
  <Subject>一般</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>write</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>研修医</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>研修医</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>write</Action>
</Rule>
```

図 5 利用者に複数のロールを与えた場合のポリシー記述

```
<Rule Effect = "Permit">
  <Subject>外来医</Subject>
  <Resource>/data/CareCard/operative_record[doctorID='UserID']/narcosis_record</Subject>
  <Action>read</Action>
</Rule>
<Rule Effect = "Deny">
  <Subject>外来医</Subject>
  <Resource>/data/CareCard/operative_record/narcosis_record</Subject>
  <Action>write</Action>
</Rule>
```

図 6 追加ルールのポリシー記述

4. アクセス頻度を考慮したポリシーテーブルの生成

通常，アクセス判定は，2 節で述べたような XACML などのポリシー記述をもとに，ユーザが資源に対してアクセス可能かどうかを判定する．これまで述べた医療現場での電子カルテの例からも分かるように，アクセス制御には非常に粒度が細かい設定が求められる．アクセス制御の粒度が細かくなると，アクセス判定にかかる時間が大きくなってしまう問題がある．この節では，アクセス頻度と MRU (Most Recently Used) を考慮した，アクセス制御の高速化に関して考察する．

ユーザがアクセスする要素の情報がアクセス前にあれば，その要素に対する判定結果をアクセス前に出すことができる．アクセス時には，前もって出された判定結果を参照することでアクセス制御をすることができ，制御が高速化される．ユーザが高い可能性でアクセスすると予想される要素は以下の要素であると考える．

- 最近アクセスした要素
- アクセスする頻度が高い要素

以上の要素に対する判定結果を格納するために，Most-Recently-

Rank	Path	Action	NumOfAccess	TotalOfAccess	ValueOfFreq
0	/data/Care_card/medical_plan	read	5	21	5/21
1	/data/Care_card/tell	read	1	4	1/4
2	/data/Care_card/oprative_record/stuffID	write	21	98	21/98
N-2	/data/Care_card/oprative_record/narcosis_record	write	1	263	1/263
N-1	/data/Care_card/oprative_record/operative_process	read	1	291	1/291

図 7　MRU-Table 例

Used Table (以下 MRU-Table) と High-Frequency Table (以下 HF-Table) をロール毎に生成する．

4.1　MRU-Table

MRU-Table は，最近アクセスした要素を格納する．MRU-Table は以下の属性によって構成される．またテーブルのサイズは固定長で，本研究では N タプル格納できるテーブルとする．

MRU-Table = {Rank, path, Action, NumOfAccess，TotalOfAccess，ValueOfFreq}

それぞれの属性に関して説明する．

- Rank：0 から N-1 までの値をとり，最も最近にアクセスされた要素にもっとも小さな値を格納する．
- path：アクセスされた要素の位置で，XPath 式によって表す．
- Action：path によって特定された要素へのアクション（本研究では，read, write, delete のいずれか）
- NumOfAccess：path によって特定された要素へのアクセス回数を表す．
- TotalOfAccess：path によって特定された要素へのアクセス後全ての要素へのアクセス回数を表す．
- ValueOfFreq：NumOfAccess/TotalOfAccess によって表す．

図 7 は，MRU-Table の例である．MRU-Table にデータを挿入するときのアルゴリズムを**図 8** 示す．1 行目は，ユーザが資源である XML 文書の path がPである要素にアクセスした場合を表している．2〜20 行では，Pがすでに，MRU-Table に存在した場合である．この場合，Pが存在するタプルの Rank を 0 に更新し，さらに，P が存在したタプルの Rank よりも小さい Rank を 1 増やす．そして全ての TotalOfAccess を 1 増やす．最後に新しい，ValueOfFreq を計算し更新する．9〜19，29〜39 行では，ValueOfFreq が 1/P 以上で，且つ TotalOfAccess が N-1 以上のタプルの情報を HF-Table に格納し，MRU-Table からは，削除する．削除した後，全体の Rank を整える．21〜40 行目は，P が，MRU-Table に存在しなかった場合である．このアルゴリズムによって，最近にアクセスされた要素の情報は MRU-Table に存在し，アクセスする頻度が高い要素の情報は HF-Table に格納することができる．今回，HF-Table に格納する要素のアクセス ValueOfFreq は 1/10 より大きい値の要素とした．

以上の MRU-Table への挿入アルゴリズムを用いて，XML 文書へのアクセスが行われたときの例を**図 9** に示す．

```
01 if (User access the XML document where path=P){
02   if(SELECT path FROM MRU-Table = P){
03      UPDATE MRU-Table SET Rank = '0', NumOfAccess = 'NumOfAccess + 1'
04                                                  WHERE path = 'P';
05      UPDATE MRU-Table SET Rank = 'Rank + 1', WHERE Rank < Rank WHERE path=P; ≠ 'P';
06      UPDATE MRU-Table SET TotalOfAccess = 'TotalOfAccess + 1';
07      UPDATE MRU-Table SET ValueOfFreq = NumOfAccess/TotalOfAccess;
08      }
09          if(SELECT ValueOfFreq FROM MRU-Table >=1/P ∧
10          SELECT TotalOfAccess FROM MRU-Table > N-1){
11            INSERT INTO HF-Table VALUES *
12                          WHERE  MRU-Table.ValueOfFreq >=1/P ∧
13                                          MRU-Table.TotalOfAccess > N-1;
14            UPDATE MRU-Table SET Rank = 'Rank - 1'
15                          WHRER Rank < Rank (ValueOfFreq >=1/P ∧
16                                                  TotalOfAccess) > N-1;
17            DELETE FROM MRU-Table
18                          WHERE ValueOfFreq >=1/P ∧ TotalOfAccess > N-1;
19            }
20          else  end;
21   else{
22      DELETE FROM MRU-Table WHERE Rank = N-1;
23      INSERT INTO NRU-Table (Rank, path, Action, NumOfAccess, TotalOfAccess, ValueOfFreq)
24                                                          VALUES (0, P, A, 1, 1, 1/1);
25      UPDATE MRU-Table SET Rank = 'Rank + 1', WHERE path ≠ 'P';
26      UPDATE MRU-Table SET TotalOfAccess = 'TotalOfAccess + 1',
27                                                  WHERE path ≠ 'P';
28      }
29          if(SELECT ValueOfFreq FROM MRU-Table >=1/P ∧
30                              SELECT TotalOfAccess FROM MRU-Table > N-1){
31            INSERT INTO HF-Table VALUES *
32                           WHERE  MRU-Table.ValueOfFreq >=1/P ∧
33                                          MRU-Table.TotalOfAccess > N-1;
34            UPDATE MRU-Table SET Rank = 'Rank - 1'
35                           WHERE Rank < Rank (ValueOfFreq >=1/P ∧
36                                          TotalOfAccess > N-1);
37            DELETE FROM MRU-Table
38                           WHERE ValueOfFreq >=1/P ∧ TotalOfAccess > N-1;
39            }
40          else  end;
41 }
```

図 8　挿入アルゴリズム

4.2　HF-Table

HF-Table はアクセス頻度が高い要素を格納する．HF-Table の構成要素を以下に示す．

HF-Table = {Rank, path, Action, NumOfAccess，TotalOfAccess，ValueOfFreq}

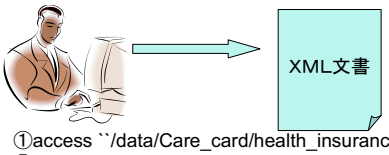
HF-Table に要素が挿入される場合は，ValueOfFreq が最小の要素がテーブルから削除される．Rank は，ValueOfFreq が大きい順に，0 から M-1 を割り当てる．

図 10 に HF-Table の例を示す．**図 10** から分かるように，HF-Table には，TotalOfAccess が N-1 以下の要素は格納されない．これは，MRU-Table から，アクセス頻度が高い要素を HT-Table に移す時に，TotalOfAccess が N-1 以上の要素と限定しているからである．TotalOfAccess が小さい場合，十分に信頼できるアクセス頻度値は出せない．例として，a/s/x の要素は，NumOfAccess = 1, TotalOfAccess = 3 であり，a/s/y の要素は，NumOfAccess = 230, TotalOfAccess = 2000 である場合を考える．

- a/s/x：ValueOfFreq = 1/3 で，非常に高い頻度値であるが，今後，利用者がどの要素にアクセスするかで，頻度値が大きく変わる．従って，信頼できる頻度値ではない．
- a/s/y：ValueOfFreq = 230/2000 で，a/s/x よりも頻度値は低いが，今後のアクセス状況による頻度値の変動が小さい．従って，信頼できる頻度値と考えられる．

5.　実　　験

ポリシーテーブルを用いた場合のアクセス判定時間と XML のポリシー記述を用いたアクセス判定時間を比較する．以下に，



①access ``/data/Care_card/health_insurance_number''
②access ``/data/Care_card/oprative_record/stuffID''

Rank	Path	Action	NumOfAccess	TotalOfAccess	ValueOfFreq
0	/data/Care_card/medical_plan	read	5	21	5/21
1	/data/Care_card/tell	read	1	4	1/4
2	/data/Care_card/oprative_record/stuffID	write	21	98	21/98
98	/data/Care_card/oprative_record/narcosis_record	write	1	263	1/263
99	/data/Care_card/oprative_record/operative_process	read	1	291	1/291

①

Rank	Path	Action	NumOf Access	TotalOfAccess	ValueOfFreq
0	/data/Care_card/health_insurance_number	read	1	1	1/1
1	/data/Care_card/medical_plan	read	5	22	5/22
2	/data/Care_card/tell	read	1	5	1/5
3	/data/Care_card/oprative_record/stuffID	write	21	99	21/99
98	/data/Care_card/X-ray_picture	write	1	134	1/134
99	/data/Care_card/oprative_record/narcosis_record	write	1	264	1/264

②

Rank	Path	Action	NumOf Access	TotalOfAccess	ValueOfFreq
0	/data/Care_card/health_insurance_number	read	6	23	5/23
1	/data/Care_card/medical_plan	read	1	2	1/2
2	/data/Care_card/tell	read	1	6	1/6
3	/data/Care_card/oprative_record/stuffID	write	21	100	21/100
98	/data/Care_card/X-ray_picture	write	1	135	1/135
99	/data/Care_card/oprative_record/narcosis_record	write	1	265	1/265

delete

Rank	Path	Action	NumOf Access	TotalOfAccess	ValueOfFreq
0	/data/Care_card/health_insurance_number	read	6	23	5/23
1	/data/Care_card/medical_plan	read	1	2	1/2
2	/data/Care_card/tell	read	1	6	1/6
97	/data/Care_card/X-ray_picture	write	1	135	1/135
98	/data/Care_card/oprative_record/narcosis_record	write	1	265	1/265
99	null	null	null	null	null

図 9　MRU-Table への挿入例

Rank	Path	Action	NumOfAccess	TotalOfAccess	ValueOfFreq
0	/data/Care_card/medical_result	read	92	345	92/345
1	/data/Care_card/operative_records/chairmanID	read	134	409	134/409
2	/data/Care_card/name	write	34	123	34/123
M-2	/data/Care_card/oprative_record/stuffID	write	21	100	21/100
M-1	/data/Care_card/address	read	12	200	12/200

図 10　HF-Table 例

実験環境を示し，ある病院のアクセス制御の例から目標となるアクセス判定時間を計算する．最後に実験結果を示す．

5.1　実験環境

実験に使用した計算機は，2.53GHz Pentium4 1.0GB RAM, OS は Microsoft Windows XP，実行プログラムは JAVA を用いて実装した．XML のポリシー記述では，それぞれ 100, 200, 300, 1000 個のルールを持つポリシーを準備した．また，ポリシーテーブルでは，それぞれ 100, 200, 1000 個のタプルを持つポリシーテーブルを準備した．

5.2　目標となるアクセス判定時間

アクセス判定時間は，小さいほど望ましい．しかし，粒度が細かいアクセス判定をすると，粒度が粗いアクセス判定よりもアクセス判定時間は長くなってしまう．粒度が細かいアクセス判定におけるアクセス判定時間は，粒度が粗いアクセス判定におけるアクセス判定時間に近づくことが目標となる．本研究では，2.2 節で述べたある病院のアクセス制御ポリシーを XML で記述し，そのポリシー記述を用いてアクセス判定した場合のアクセス判定時間を目標とする．その病院アクセス制御について簡単に説明する．Subject は，ロールによって定義され，61 種類のロールが存在する．Resource は，経路単位で定義されている．この病院のアクセス判定時に用いるアルゴリズムは，“Permit-overrides” アルゴリズムを使用した．それは，全ての経路に対して許可されているか，ほんの一部の経路に対して許可されているかのどちらかで，“Permit-overrides” を用いた方が他のアルゴリズムを用いるよりルール数が少なくなるからである．最後に Action は，read，write，ordering が定義可能であるが，本実験では, read と write とした．

以上のアクセス制御をポリシー記述に表すとルール数は 15 となった．さらに，このポリシー記述を用いてアクセス判定を行った時のアクセス判定時間は 16(ms) となった．この実験に用いたアクセス制御は非常に粒度が粗いアクセス制御であるので，粒度を細かくした場合には，16(ms) に近くなるようなアクセス判定時間が望まれる．

5.3　実験結果

図 11 は，XML で記述されたポリシーを用いたときの最長のアクセス判定時間を示している．また，**図 12** は，ポリシーテーブルを用いたときの最長のアクセス判定時間と平均のアクセス判定時間を示している．

XML のポリシー記述では，アクセス判定の大半が，全てのルールを参照しなければならない．それは，“Deny-overrides” アルゴリズムを用いた場合，どこに拒否を現すルールが存在するか分からないので，最終的に許可を返すルールは，全てのルールを参照する必要があるからである．また，一般的に，許可を返すアクセスの方が，拒否を返すルールより多いと考えられる．従って，アクセス判定時間が最大値より小さくなることは少ない．一方，ポリシーテーブルを用いたアクセス判定では，対応するタプルが見つかれば，即座にアクセス判定を出すことが可能である．さらに，4. 節で説明したアクセス頻度と MRU を考慮することで，対応するタプルが比較的早く見つかる場合が多いと考えられる．従って，最大値よりアクセス判定時間が小さくなることが多いと考えられる．

しかしながら，全てのアクセス判定をポリシーテーブルを用いて実施することはできない．ポリシーテーブルでは判定できない場合，まず，ポリシーテーブルで判定ができないことを確認する．確認後，XML のポリシー記述で判定するので，XML のポリシー記述のみを参照するよりアクセス判定時間は大きくなると考えられる．そこで，アクセス時の 80 %がポリシーテーブルで判定が可能と仮定し，その際のアクセス判定時間を計算する．その計算結果を，**図 13** に示す．**図 13** を見て分かるよう

Number of rules	Max Run-time for access decision (ms)
100	16
200	32
300	60
1000	110

図 11　XML ポリシー記述を用いたときのアクセス判定時間

Number of tuples	Max Run-time for access decision (ms)	Average Run-time for access decision (ms)
100	15	15
200	16	15
1000	78	62

図 12　ポリシーテーブルを用いたときのアクセス判定時間 1

Number of rules	Number of tuples	Run-time for access decision (ms)
100	100	18.2
100	200	18.2
100	1000	65.2
200	100	21.4
200	200	21.4
200	1000	68.4
300	100	27
300	200	27
300	1000	74
1000	100	37
1000	200	37
1000	1000	84

図 13　ポリシーテーブルを用いたときのアクセス判定時間 2

に，ルール数が多くなるほど，ポリシーテーブルの利用が有効であることが分かる．ポリシーテーブルのタプルの数が 100 個か 200 個の場合は，XML のポリシー記述のルール数が，200 以上のとき，XML ポリシー記述のみを用いる場合より有効であるといえる．

6. まとめと今後の課題

ロールの種類が多くなった場合のロールの扱い方とアクセス頻度を考慮したアクセス制御の高速化に関して述べた．アクセス頻度と MRU を考慮したポリシーテーブルの生成では，経路単位でのアクセス制御を仮定した．実験から，アクセス頻度に偏りがあり，XML のポリシー記述のルール数が多くなった場合は非常に有効であるといえる．しかし，経路単位でなく資源となる XML 文書の内部データに依存した判定など，要素単位でのアクセス制御を考える必要がある．また，ポリシーテーブルの資源の位置を示す記述として，XPath 式を用いた．XPath 式では，要素や属性の出現位置に依存せずにそれらを参照できる descendant-or-self 軸 (“//”) や，特定の条件を満たした要素をフィルターするための構文を使うことができる．このような構文を用いた場合，入力の XPath 記述とポリシーに記述されている XPath 式との包含関係を調べるためのコストを考慮する必要がある．さらに，本研究ではロール毎にポリシーテーブルを生成したが，ポリシーテーブルのもととなる，XACML のポリシー記述自体が変更された場合は，ポリシーテーブルも更新する必要がある．ポリシーの変更が多い場合，変更に伴うコストも考慮する必要がある．

ロールの扱いに関しては，医療分野を例に述べたが，医療分野独自の問題も存在した．それは，動的なロールの変更である．医療現場では，常に同じアクセス権限では障害が起こる可能性がある．それは，患者の様態が急変した場合などに，その患者情報にアクセスできる権限を持った医師が存在しなかった場合である．そのような場合には，その場に存在する医師にアクセス権を与える必要がある．また，患者は常に同じ診療科に掛かるのでなく，頻繁に変更したり，掛け持ちする場合もある．その場合のロールの変更などを管理していくのは非常に困難である．従って，動的なロールの扱いに関しては今後の課題である．また，医療分野では，さらに複雑なアクセス制御が求められている．例えば，患者自身が，自分のカルテ情報に対して自由にアクセス権を定義できるようにすることである．これは，カルテは患者のものであるという考え方があり，患者主体のアクセス制御が求められているからである．このことから，ポリシー記述方法などの専門知識がない患者などが，容易に粒度が細かいアクセス制御ポリシーを構築できる仕組みも必要であると考えられる．

文　献

[1] Medical Markup Language (MML)Version3.0 規格書 http://www.medxml.net/mml30/MMLV3Spec_050817.pdf

[2] OASIS.eXtensible Access Control Markup Laguage(XACML) Version2.0.OASIS Standard,1 February 2005 http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[3] Extensible Markup Language 1.0. W3C Recommendation 10-February-1998 http://www.w3.org/TR/1998/REC-xml-19980210.html

[4] XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999 http://www.w3.org/TR/xpath

[5] Notake Kitagawa, Masatoshi Yoshikawa. A Study Efficient Access Control for XML Documents. International Special Workshop on Databases For Next Generation Researchers (SWOD2005)

[6] Naishin Seki, Michiharu Kudo. 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)

[7] Mamoru Sato, Masuyoshi Yachida, Hiroyuki Suzuki, Takashi Obi, Masahiro Yamaguchi, Nagaaki Ohyama, Kouichi Kita 異なる医療情報ネットワークドメイン 間に属する機器の接続方法に関する研究. FIT2005