社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－DBS－134 (I) (19)
2004／7／13

アクセス条件テーブルを用いたXMLアクセス制御

戚 乃箴† 工藤 道治†

† 日本アイ・ビー・エム 東京基礎研究所 〒242-8502 神奈川県大和市下鶴間 1623-14

あらまし XML文書にはセキュリティレベルの異なる要素や属性が含まれていることが多く、ノードレベルのアクセス制御が必要である。XPathを使ってノードレベルのアクセス制御を実現する場合、アクセス制御ポリシーの記述力と実行時のアクセス判定時間とのトレードオフを考慮することが重要になる。本稿で提案するアクセス条件テーブル方式を用いると、'//'や述語式などを使った柔軟なポリシー記述が可能となる。計算機実験により、実行時のアクセス判定がポリシーのサイズやXML文書の数に依存せずに高速に実行できることを示す。

XML Access Control Using Access-condition-table

Naishin SEKI† Michiharu KUDO†

† Tokyo Research Laboratory, IBM Japan 1623-14, Shimotsuruma Yamato-shi 1-2-3, Kanagawa-ken, 242-8502 Japan

Abstract Access control policies for XML requires fine-grained access control that works on each node belonging to different security level. XPath-based node-level access control needs to address the trade-off problem between expressiveness of the policy specification and the required time for making access decision. Access-condition-table based scheme enables flexible policy specification using '//' and predicate expression. This paper also includes the result of the computer experiments that shows the proposed scheme enables fast access decision making independent of the size of the policy and XML documents.

1. はじめに

XML[1]は汎用的な木構造データモデルであり、ビジネス文書、受発注処理、医療カルテ、遺伝子情報などの幅広い分野で使われている。一つのXML文書の中には、セキュリティレベルの異なる内部データ(受付日と認証コードなど)が含まれることが多く、文書レベルのアクセス制御ではセキュリティの観点から不十分である。要素や属性単位にアクセス制御を行う「ノードレベルのアクセス制御」が必要であり、これまで多くの研究が行われてきた[2,4-7]。

ノードレベルのアクセス制御では、XPath[3]を利用して対象ノードを指定してアクセス制御ポリシーを記述する。予め定められた位置に出現する要素や属性に対するアクセス制御は比較的に容易に実現できる。しかし、任意の要素の中に出現可能な属性(例えばグローバル属性)や、任意の位置・深さに現れる要素(例えばコンテンツを修飾するために使われる要素)などに対するアクセス制御が必要な場合もある。XPath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 axis ('//')や、特定の条件を満たしたノードをフィルターするための構文を使うことができる。しかし、複雑なXPath式を扱うと処理のオーバーヘッドが問題となる[2,4,5]。そのために、アクセス対象のXML文書の値を利用してアクセス判定機構の最適化を行う方法などが考えられている[9,11-14]。しかし、XML文書が頻繁に更新されるようなアプリケーション[16]ではそのような最適化は一般には適用できない。

本論文では、柔軟なポリシー記述を許すと同時

に高速なアクセス判定ができ、XML データベースにも適用可能な汎用的なアクセス制御手法の実現を目指した。提案方式は、与えられたアクセス制御ポリシーを変換してアクセス条件テーブル(Access condition table, ACT)という表構造を生成し、その表を用いて実行時に任意の深さに対する要求パスのアクセス判定を行う。ACT には、検索のキーとして使われるパスと、パスに結び付けられた二種類のアクセス条件を格納する。アクセス判定時には、該当するパスを検索し、付随したアクセス条件を評価することでアクセス可能、禁止を判定する。本方式を用いると、XPath の柔軟な表現力を満たしながら、XML 文書やスキーマ情報の更新に影響されずに高速なアクセス判定が可能となる。

2. 準備

2.1. XML

XML 文書は要素、属性、テキストノードから構成される木構造である。要素の内容は、要素またはテキストノードのシーケンスであり、任意の個数の属性(名前と値のペア)を持つことができる。図 1 に XML 文書の例を示す。

```
<a>
  <b>
    <e><i/><j/></e>
    <f><k/><l/></f>
  </b>
  <c><g/></c>
  <d><h/></d>
</a>
```

図 1　サンプル XML 文書

2.2. XPath

XPath[3]は XSLT や XQuery[10]などで用いられる正規パス表現の一つである。XPath では、"axis" を用いてノード間の構造関係を表現する。例えば、p//a の'//'は descendant-or-self という axis であり、ノード p とノード a の間の構造的な階層関係(子孫関係)を表す。XPath には複数の axis が定義されているが、本論文では descendant-or-self ('//'), child ('/'), attribute ('@')の三つの axis と述語式を扱う。述語式とは、個々のノードが満たすべき条件を付加的に記述するものであり、例えば p[@bar="abc"]は、ノード p が満たすべき条件 @bar="abc"を指定している。

2.3. アクセス制御ポリシーの概要

本論文では、著者らが ACM CCS の論文で提案したアクセス制御ポリシーのモデル[6]を採用する[1]。アクセス制御ポリシーは、任意の数のアクセス制御ルールから構成される。アクセス制御ルールは、主体(subject)、アクセスモード(access mode)、客体(object)の三つ組みから構成され、(subject, access mode, object)によって表現する。アクセス制御ルールのパラメータを表 1 に示す。

表 1　アクセス制御ルールの構成要素

パラメータ	意味
主体 (subject)	利用者の ID やロール、計算機プロセスなど
アクセスモード (access mode)	許可フラグと動作の組み合わせ '+r'や'-R'など
許可フラグ (permission)	許可('+')または禁止('-')
動作 (action)	単ノード読込み('r')、サブツリー読込み('R')
客体 (object)	アクセス制御の対象となる XML のノード。XPath 式で記述。

subject は、プリフィックスとして uid や role などの型を指定できる。action における単ノード読込み('r')とは、XPath で指定されたノードのみに適用される読込み動作を意味する。一方、サブツリー読込み('R')とは、XPath で指定されたノードおよびその子孫の全てのノードに対して適用される読込み動作を意味する。例えば (uid:Alice, +r, /record)の場合、ユーザ ID:'Alice'のユーザは、/record で参照されるノードにのみ読込みアクセス権が与えられることを意味する。一方、(role:nurse, +R, /record/diagnosis)の場合は、

[1] 標準化されたアクセス制御ポリシー言語として XACML[8]を使っても記述可能。

nurse というロールが割り当てられたユーザは、/record/diagnosis で参照されるノードおよびその子孫の全てのノードに対して、読込みアクセス権を持つことを意味する[2]。

複数のアクセス制御ルール同士は'or'で結合される。二つの異なるルールが相矛盾する場合(一方がアクセス許可で他方がアクセス禁止の場合)、アクセス禁止のルールを優先する。他の矛盾解消ポリシーを適用することも可能である[15]。

3. アクセス条件テーブル

3.1. 要件

本論文では、汎用的なアクセス判定機構に対して次ぎの三つの要件を満たす必要があると考える。

- XML 文書独立：アクセス判定機構は XML 文書と独立して定義できること。XML 文書に対する変更や追加があった場合でも判定機構の構成に影響を与えないこと。
- スキーマ独立：アクセス判定機構は XML のスキーマ定義と独立して定義できること。スキーマに対する変更があった場合でも判定機構の構成に影響を与えないこと。
- 問い合わせ言語独立：アクセス判定機構は問い合わせ言語の種類に依存しないこと。複数の操作言語に対して中立でありどの言語にも適用可能であること。

上記三つの要件の前提条件としては、ポリシー変更の頻度は XML 文書やスキーマの変更頻度より圧倒的に少ないことである。これは、通常の XML データの利用形態から妥当であると考えられる。

3.2. ACT の概要

ここでは ACT の概要を簡単に説明する。ACT はアクセス制御ポリシーの各ルール記述から生成される。例として、R1 から R4 の四つのルールを考える。

R1: (role:manager, +r, /a)
R2: (role:manager, +R, /a/b)
R3: (role:manager, +r, /a/c[g>1])
R4: (role:manager, -R, /a/b//e)

図 2 は、図 1 の XML 文書上で R1-R4 のルールがもたらすアクセス制御の効果を示している[3]。黒丸のノードはアクセスが許可されるノード、黒丸が二重のノードは、ノードのアクセスに何らかの条件式が付加されているノード、白丸のノードは明示的にアクセスが禁止されているノード、四角のノードは該当するルールが存在しないノードを示す。矢印は、アクセス制御ルールが子孫のノードに伝播していることを意味する。

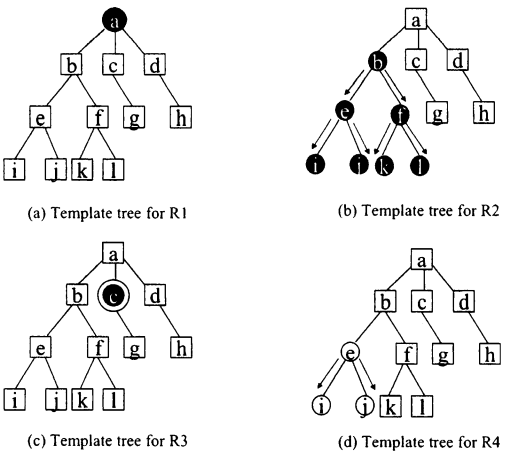

図 2　アクセス制御ルールの効果

図 3 は、図 2 の各効果を一つの図の上に重ね合わせたものである。

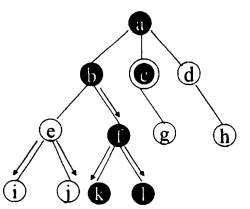

図 3　効果の重ね合わせ

[2]読込み以外の動作として、ノードの更新、生成、削除などにも本論文の手法は適用可能。

[3] ACT の定義は XML 文書に依存しない。ここでは、説明のために文書上の効果を示している。

効果の重ね合わせの結果を表形式で表現したものがACTである。図3を表現するACTを表2に示す。ACTは、対象パス、ローカルアクセス条件、サブツリーアクセス条件の三つのカラムから構成される。対象パスは、アクセス判定時の要求パスに対するキーとして用いる。ローカルアクセス条件は、対象パスの末端のノードに適用される条件を記述する。サブツリーアクセス条件は、対象パスの末端ノードの子孫に対して適用される条件を記述する。表2の場合、第一行はR1から、第二行はR2とR4から、第三行はR3から生成される。図2から図3を生成する際には、各ルールの伝播、矛盾解消、デフォルトの判定ポリシーなどを考慮して重ね合わせるが、ACTを作成する時にも同様の処理を行う。

表2　ACTの例

対象パス	ローカル アクセス条件 (LAC)	サブツリー アクセス条件(SAC)
/a	true	false
/a/b	true	not (ancestor-or-self::e)
/a/c	g>1	false

3.3. ACT実行時の動作

ACTを使って実行時判定を行う方法について述べる。データベースに保存されているXML文書にアクセスが発生し、XML文書全体あるいは特定のノードに対してアクセスが発生したとする。ここで、個々のノードに対するアクセスは、全てルート要素からのパスに基づくアクセスと考える。これを要求パスと呼ぶ。要求パスがACTの対象パスに存在すれば、該当するローカルアクセス条件を使ってアクセス権を判定する。もし要求パスが対象パスに存在しなければ、要求パスの各々のプリフィックスに対して処理を繰り返す。プリフィックスのパスが対象パスに存在すれば、該当するサブツリーアクセス条件を使ってアクセス権を判定する。全てのプリフィックスに該当する対象パスが存在しなければアクセスは拒否する。表2のACTを用いて四つのアクセス判定を行う例を表3に示す。

表3 アクセス判定の例

要求 パス	該当 対象パス	評価される アクセス条件	アクセス判定
/a	/a	true　(LAC)	許可
/a/c	/a/c	g>1　(LAC)	g>1ならば許可、 g≦1なら禁止
/a/d/h	/a	false　(SAC)	禁止
/a/b/e/i	/a/b	not (ancestor-or-self:: e)　(SAC)	禁止

3.4. XPathの制限

ACTでは、objectを参照するために用いるXPathとして、'//'、'*'、述語式を使うことができるが、利用できる構文には以下のような制限を設けている。

- '//'は参照式の中で一度しか現われない.(例えば/a//c)
- '*'は'//'または'@'と組み合わせて指定する(例えば/a//*)
- '//'は**述語の中では使用しない**
- '//'に続く相対パスの長さは最大1。(例えば/a//c)

上記の制限はあるが、任意の位置・深さに出現する可能性のある要素を指定したり(//comment)、特定の条件を満たす要素を指定したりできるため(//*[@private="yes"])、柔軟性の高いポリシーを記述することが可能になる。

4. ACTの構成方法

ACTを構成する方法について説明する。ACTは、1:対象パスの生成、2:ローカルアクセス条件の生成、3:サブツリーアクセス条件の生成、4: 効果の伝播、5: 矛盾解消と条件式の結合の5つのステップで構成される。ACTはアクセス制御ポリシーの値からのみ生成され、XML文書やスキーマの値には一切依存しない。説明の中では、3.2節のR1-R4のポリシーを参照する。

4.1. ステップ1: 対象パスの生成

XPath式に'//'が含まれている場合は、'//'のプ

リフィックスが対象パスになる。XPath 式に述語が含まれている場合は、XPath 式から述語を除いたものが対象パスになる。'//'も述語も含まれていない場合、XPath 式そのものが対象パスになる。R2 は/a/b、R3 は/a/c、R4 は/a/b がそれぞれ対象パスになり、ACT に該当するエントリーが生成される。

4.2. ステップ 2: ローカルアクセス条件の生成

ローカルアクセス条件(LAC)は、XPath 式とアクセスモードから生成される。XPath 式が'//'と述語を含まない場合、許可フラグが'+'ならば'true'、'-'ならば'false'を生成する。XPath 式が'//'は含まないが述語を含む場合、述語式を生成する(ただし述語内の要素は対象パスを基準とした相対パスに変換する)。一つの XPath 式に述語が複数存在する場合、変換後の述語式を'and'で結合した論理式を生成する。R2 は'true'、R3 は'g>1'となる。

4.3. ステップ 3: サブツリーアクセス条件の生成

サブツリーアクセス条件(SAC)は、ルートから対象パスに至るアクセス条件と、対象パスからある階層関係を満たす子孫に至るまでのアクセス条件を組み合わせて生成する。図 4 にサブツリーアクセス条件生成アルゴリズムを示す。T は XPath 式('/e_1//e_n'と仮定)、A はアクセスモードを表す。R4 の場合、not(ancestor-or-self::e)となる。

```
If (T contains '//') then
  If (A is '+R') then SAC:= ancestor-or-self::en or
                            descendant-or-self:en
  Else if (A is '+r') then
    SAC := descendant-or-self::en
  Else if (A is '-R') then
    SAC := not(ancestor-or-self::en)
  End if
Else
  If (A is '+R') then
    SAC := true
  Else if (A is '-R') then
    SAC := false
  Else
    Return    // SAC is N/A
  End if
End if
```

図 4　サブツリーアクセス条件生成アルゴリズム

4.4. ステップ 4: アクセス式の伝播

このステップでは、サブツリーアクセス条件が記述されている対象パスを基準として、その子孫にあたる全ての対象パスに対して同じサブツリーアクセス条件を伝播させる。この処理により、上流のアクセス条件を必ず満たすことがことができる。

4.5. ステップ 5: アクセス式の結合と矛盾解消

ステップ 4 により複数の条件式が一つのアクセス条件(LAC または SAC)に生成された場合、それらを"and"で結合した後、アクセス不許可優先ポリシーに従って矛盾解消を行う。3.2 節の例の場合、/a/b の対象パスの LAC に R2 から'true'が、R4 から'not(ancestor-or-self::e)'が生成され、結果として'true and not(ancestor-or-self:e)'になる。このステップで'not(ancestor-or-self::e)'に変換される。

5. 実験

ACT を使った計算機実験を行い、アクセス判定速度を計測した。本実験では、アクセス制御ポリシーの典型的な記述方法を二種類設定し、各々の記述方法(パターン)に対してポリシー記述のサイズと判定時間との関係を調べた。実験の単純化のためにポリシー記述の対象をユーザ一人の場合に限定したが、ユーザが複数の場合は ACT を複数生成すればよい。述語式の評価のためには XML 文書の特定のノード値を取り出す必要があるが、それにかかる時間は ACT 特有のものではないので本実験では全て無視できる一定時間と仮定している。

5.1. 実験環境

実験に使用した P C は 3GHz Xeon 2.5GB RAM、OS は Microsoft Windows XP、実行プログラムは Java JDK1.3 を用いて実装した。実験に使用した XML 文書はファイルサイズが 200KB、768 個のパス、4,727 個のノードが存在するもので、xmlspec-v20.dtd から生成した。

本実験では、アクセス制御ルールの個数と判定

時間との関連性を調べる。アクセス制御ルールをランダムに生成すると、単一の要素や属性に対して複数のポリシーが重複して生成されてしまう恐れがある。実際にセキュリティ管理者がアクセス制御ルールを記述する場合、ルールの重複記述は行わない。従って、本実験ではルールをランダムに生成するのではなく、パターンaとパターンbの二種類の記述パターンに従ってアクセス制御ルールを生成することにした。パターンaは、アクセスモードとして"+r"のみを使って記述する。パターンbは、ルート要素に対してのみ"+R"を記述し、他のノードには全て"-R"を使って記述する。図4は、四つの要素a,b,c,dを持つXML文書に対して同じ効果を持つ2種類の記述例を示す。

パターンa
(uid:Seki, +r, /a)
(uid:Seki, +r, /a/b)
(uid:Seki, +r, /a/c)
パターンb
(uid:Seki, +R, /a)
(uid:Seki, -R, /a/d)

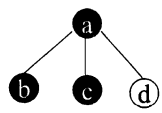

図4　二パターンによるアクセス制御ポリシー

5.2. ACTのサイズ

各々のパターンとACTのサイズとの関係について説明する。ACTのサイズは対象パスのエントリー数とする。図5は、アクセスが許可されるノードの個数と必要なルールの個数との関係を示している。

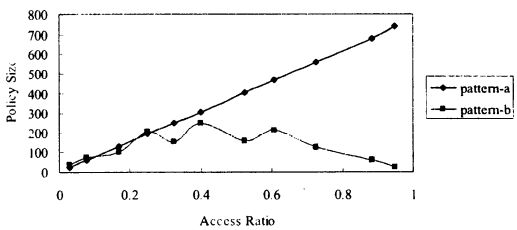

図5　ルールの個数とアクセス可能ノード数の関係

X軸のAccess Ratioは、XML文書中でアクセス可能と判定されるパス数の、文書全体のパス数に対する割合を表している。Y軸のPolicy Sizeとは、該当するアクセス制御ルールの個数を表している。例えばAccess Ratio 0.6の場合、全体の6割のノードにアクセス許可するために必要なポリシーの個数が、パターンaの場合は約460個(768パスの約6割)、パターンbの場合は約200個のポリシーが必要になることを示している。パターンaの場合は、アクセス可能なパスには必ず該当するルールが一つ必要になるので、Access Ratioが1に近づくにつれACTのサイズも線形に増加していく。一方、パターンbの場合は"+R"のルールによりパスの多くをカバーすることができるため、Access Ratioが増えても必要となるルール数は増えない。従ってパターンbの方がACTのサイズを小さく保つことができる。

5.3. 実験結果

Access Ratioを0.03から0.95まで変化させた時に、XML文書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た。これはアクセス権を考慮したユーザービュー生成時間に相当する。

A) '//'のないポリシーの場合

はじめに、ルールに"//"が含まれていない場合を計測した。図6は、パターンaとパターンbに対してダイレクトアクセス権チェック方式(Direct方式)とACT方式による経過時間を示す。

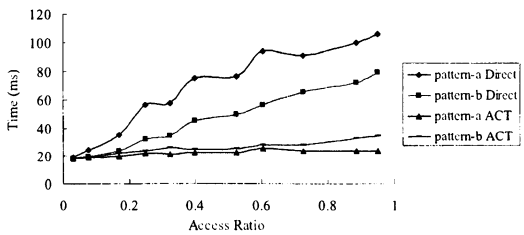

図6　ユーザービュー生成時間の比較

Direct方式とは、ACTのような構造を用いずに、要求パスとルールとを比較して適合するルールを検出し、アクセス判定を行う方式を意味する。

図からわかるように、Access Ratio が 1 に近づくにつれて、Direct 方式は判定に必要な時間が増えていくのに対して、ACT 方式では Access Ratio によらずほぼ一定であることがわかる。特に Access Ratio が 0.95 の時は、パターン a の場合は ACT 方式が 4 倍高速に、パターン b の場合は ACT 方式が 2 倍強高速にユーザービューを構築できる。

B) '//'のあるポリシーの場合

次に、ルールに"//"が含まれている場合を計測した。実験 A で生成したパターン b のルールの中から任意の 20%のルールを選び、その中の'/'の一つを"//"に変換してルールを生成した。実験結果を図 7 に示す。

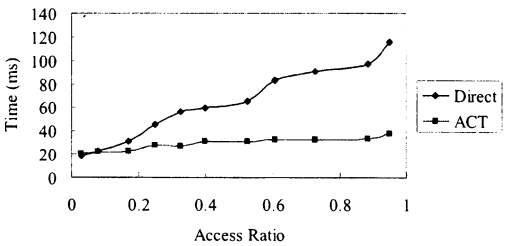

図 7　ユーザービュー生成時間の比較

実験結果から、Direct 方式は Access Ratio が増えるにつれて経過時間が増えていくが、ACT 方式はほぼ一定であることがわかる。特に、Access Ratio が 1 に近い時は、3 倍以上高速にユーザービューを構築できる。

以上の実験結果により、'//'や述語式のような計算コストのかかる XPath 構文をサポートしながら、ACT のサイズによらずほぼ一定の時間でアクセス判定を行うことができることが示された。これは、ACT 方式がアクセス制御ルールの相互関係や伝播、矛盾解消などの処理を構築時に予め処理しておくことにより、アクセス判定時にかかる時間を最小限にしたことにより実現されたと考えられる。

6. 結論

XML 文書に対するノードレベルのアクセス判定を高速に実行する方式として、アクセス条件テーブル(Access Condition Table)に基づく ACT 方式を提案した。ACT のテーブルは、アクセス制御ポリシーからのみ生成されるため、データベースに保管される XML 文書の変更やスキーマの変更には影響されないという特長を持つ。アクセス制御の対象ノードを指定する XPath 式では、'//'、'*'、述語式などの柔軟な書式を使うことが可能である。プロトタイプ実装により、実行時のアクセス判定にかかる時間が ACT のサイズに依存せずほぼ一定であるという結果が得られた。今後の課題は、より柔軟な XPath 式を処理できるような ACT 方式の拡張と、静的なアクセス判定を行う機構との連携に関する研究である。

文　献

[1] T.Bray, J.Paoli, and C.M.Sperberg-McQueen, "Extensible Markup Language (XML) 1.0. W3C Recommendation," available at http://www.w3g.org/TR/REC-xml, Feb. 1998.

[2] M.Kudo and S.Hada, "XML Document Security based on Provisional Authorization," in ACM CCS, 2000.

[3] J.Clarkand S.DeRose, "XML Path Language (XPath) version 1.0. W3C Recommendation," available at http://www.w3g.org/TR/xpath, 1999.

[4] E.Bertino and E.Ferrari, "Secure and selective dissemination of XML documents," in TISSEC, 2002

[5] E.Damiani, S.De Capitani di Vimercati, S.Paraboschi, and P.Samarati, "A Fine-Grained Access Control System for XML Documents," in ACM TISSEC, 2002.

[6] M. Murata, A. Tozawa, M. Kudo and S. Hada, "XML Access Control Using Static Analysis", in ACM CCS, 2003.

[7] S. Hada, M. Kudo, "XML Access Control Language:Provisional Authorization for XML Documents", http://www.trl.ibm.com/projects/xml/xacl/xacl-spec.html, 2000.

[8] S. Godik and T. Moses, "eXtensible Access Control Markup Language", OASIS Open Standard, XACML 1.0, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, 2003.

[9] T.Yu, D.Srivastava, L.V.S.Lakshmanan, H.V.Jagadish, "Compressed Accessibility Map: Efficient Access Control for XML", in VLDB,

2002.

[10] S.Boag, D.Chamberlin, M.F.Fernandez, D.Florescu, J.Robie, and J.Simeon, "XQuery 1.0: An XML query language, W3C Working Draft 12 November 2003," available at http://www.w3.org/TR/xquery/, 2003.

[11] Q.Li and B.Moon, "Indexing and Querying XML Data for Regular Path Expressions," in VLDB, 2001.

[12] R.Kaushik, P.Bohannon, J.F.Naughton, and H.F.Korth, "Covering indexes for branching path queries," in ACM SIGMOD, 2002.

[13] D.D.Kha, M.Yoshikawa, and S.Uemura "An XML Indexing Structure with Relative Region Coordinate, " in ICDE, 2001.

[14] M.F.Fernandez and D.Suciu, "Optimizing regular path expressions using graph schemas," in ICDE, 1998.

[15] Pierangela Samarati and S. De Capitani di Vimercati, "Access Control: Policies, Models, and Mechanisms", Foundations of Security Analysis and Design, R. Focardi and R. Gorrieri (eds) LNCS 2172, Springer Verlag, pp. 137-196, 2001.

[16] E. Cecchet, J. Marguerite, and W. Zwaenepoel, "Performance and scalability of EJB applications, OOPSLA 2002.