XML 文書のバージョン管理とアクセス制御の統合モデルの要求機能と応用

中井　隆史† チャットウィチェンチャイソムチャイ†† 岩井原瑞穂†††

† 京都大学工学部情報学科　〒 606-8501 京都市左京区吉田本町
†† 長崎シーボルト大学国際情報学部情報メディア学科　〒 851-2195 長崎県西彼杵郡長与町まなび野 1-1-1
††† 京都大学大学院情報学研究科　〒 606-8501 京都市左京区吉田本町
E-mail: †nakai@db.soc.i.kyoto-u.ac.jp, ††somchaic@sun.ac.jp, †††iwaihara@i.kyoto-u.ac.jp

あらまし　現在 XML は様々な分野で利用されている。XML 文書は時間とともに蓄積されアーカイブなどのバージョン管理が利用されている。しかし現在のアクセス制御モデルではバージョン管理を考慮していないため、変更や派生関係などを考慮した複数のバージョンを範囲に含むアクセスポリシーを指定することはできない。そこでバージョン管理とアクセス制御の統合を提案する。変更履歴から対応するノードを見出してアクセスポリシーを指定する機能やバージョンの生成日時で制御する機能が必要で、他のバージョンを攻撃することによる情報漏洩といった問題にも対処する必要がある。この論文ではバージョン管理を考慮した基本的なアクセス制御のモデルについて考察する。
キーワード　XML，セキュリティ，コンテンツマネジメント

Functional Requirements and Applications for Integration of Access and Version Control of XML Documents

Takashi NAKAI†, Somchai CHATVICHIENCHAI††, and Mizuho IWAIHARA†††

† School of Informatics, Faculty of Engineering, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto, 606-8501 Japan
†† Department of Info-Media, Faculty of Global Communication, Siebold University of Nagasaki
Manabino, 1-1-1, Nagayo Cho, Nishisonogi Gun, Nagasaki, 851-2195 Japan
††† Department of Social Informatics, Graduate School of Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto, 606-8501 Japan
E-mail: †nakai@db.soc.i.kyoto-u.ac.jp, ††somchaic@sun.ac.jp, †††iwaihara@i.kyoto-u.ac.jp

Abstract　Recently, XML documents are used in various domains and accumulated gradually. For the effective use of these documents version control is necessary. However in known access control models, version control is not taken into consideration, so access policies cannot deal with documents of two or more versions in consideration of change, the derivation relation, etc. We propose an integrated model of version and access control which includes derivation dependencies and temporal constraints. We consider propagation of access control policies over derivation relationships, and mention several application domains suitable for our model.
Key words　XML，Security，Contents Management

1. はじめに

現在、XML は様々な分野においてデータ表現や交換のためのフォーマットとして利用されている。また XML 文書は、その利用の中で日々更新され情報の蓄積を行っている。このような蓄積された XML 文書の有効利用が重要な課題となってきている。例えば Web アーカイブや編集時のバックアップなどは最新バージョン以外のバージョンを有効に活用している例である。またニュース記事についても刻々と新しいバージョンの記事が生まれているとみなすことが可能である。複数のバージョンの XML 文書を参照できるだけでなく、バージョンの生成日時や変更内容などを情報として得られることもバージョン管理の有用性である。また XML の応用分野には電子商取引、医療、行政など情報を保護するべき分野が多いことも特徴である。こ

のような XML 文書を安全に利用するためには、ユーザやその役割によって XML 文書のアクセス権を制限する必要がある。

しかし XML 文書の利用にこれまでアクセス制御とバージョン管理は別々に研究されていた。そのため一般的なアクセス制御モデルではバージョン管理が考慮されていない。バージョン管理を行う XML 文書の場合、保護情報は複数のバージョンに含まれるがスキーマレベルでは文書内容に対応した細やかなアクセス制御ができない。またインスタンスレベルではバージョン間でアクセスポリシーを共有したり、保護情報を統一管理したりすることができないなどの問題を抱えている。この問題を解決するためバージョン管理とアクセス制御の統合が文献 [4] で提案されている。本稿ではアクセス制御とバージョン管理を別々に行うことによって生じる問題点を整理することによって統合モデルの新たな要求機能を明確化し、アクセスポリシーの拡張を提案する。具体的には文書の派生関係や変更内容からバージョン間の等しいノードを見出し利用することによってバージョンを跨ぐアクセス制御やバージョン情報に含まれる時間情報を用いたアクセス制御を提案する。バージョン間で対応するノードのグラフを作り、その中の 1 点を指定し伝達を行うことでバージョン間の比較による情報漏洩の防止を行う。またバージョン管理に深く関係するバージョンの生成日時や有効期間などの時間情報を用いたアクセス制御について考察を行う。その利点や実現のための問題点の考察を行い、応用例を示す。

以下 2 章ではバージョン管理とアクセス制御の個々の技術について説明する。3 章においてバージョン管理とアクセス制御を別個に扱った場合の問題点について考察することで統合モデルの要求機能を明確化する。4 章で統合モデルの要求機能についてモデル化を行い 5 章では統合モデルの応用例について考察を行う。6 章では関連研究について述べ、最後に 7 章で本論文のまとめと将来研究について述べる。

2. XML 文書のバージョン管理とアクセス制御

本章では XML 文書のバージョン管理とアクセス制御のそれぞれの技術の概要をまとめる。まず 2.1 でバージョン管理の概要と後に利用するバージョングラフの概念の説明を行い、2.2 で XML 文書の一般的なアクセス制御モデルについてまとめる。

2.1 XML 文書のバージョン管理

XML 文書は情報の変更に伴い編集によって更新が行われる。更新の生じる XML 文書の利用において最新のバージョンのみが有用であるとは限らない。例えば文書編集時のバックアップや Web アーカイブは過去のバージョンを利用するためのものである。またニュースサイトでは刻々と記事が更新されていくが過去の記事も有用であり、時間や記事の関連性を用いて参照できるようにすることは重要である。バージョン管理は文書の更新履歴を記録し、過去のバージョンの文書への参照や復帰を可能にする技術である。各バージョンの復元や参照すべきバージョンの特定などのため、変更内容や元になった文書、バージョンの生成日時、有効期間などの情報を管理する。

XML 文書はテキストファイルであるため RCS や SCCS のようなテキストベースのバージョン管理も利用可能である。しかしこれらの手法は行を差分の単位としているため XML 文書の持つ構造を考慮できない。そのため XML の構造を利用したバージョン管理手法が研究されている [1], [5], [7]。[7] は XPath データモデルを拡張し有向枝に有効期間を与えた時制データのためのモデルである。[1] は両方向の編集スクリプトである completeΔ の列と最新バージョンを保存することでバージョン管理を可能にした変更内容に注目した研究である。RBVM [5] は、元のバージョンの部分木を参照することで必要な容量を減少させる研究である。

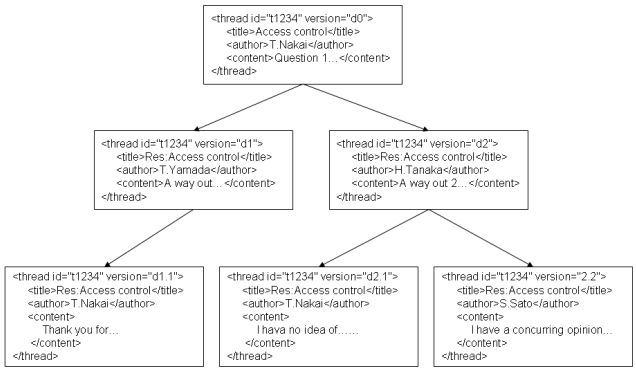

図 1　バージョン派生の例

本研究では分散環境や用途別文書の管理などを考慮する。この場合、一つのバージョンからの複数のバージョンの生成やマージによる統合、また複数のバージョンが同時に有効な状況が生じる。つまりバージョン間の派生関係が分岐したり合流したりすることから、バージョン間の関係を表現するには有向非巡回グラフである必要がある。このバージョン間の関係を表すグラフとしてバージョングラフを用いる。各文書をノードとし派生関係を有向枝で表すことでバージョン間の履歴関連を表すことが可能である。例としてスレッド型掲示板をあげる。図 1 では version ｄ 0 からｄ 1 とｄ 2、ｄ 1 からｄ 1.1、ｄ 2 からｄ 2.1 とｄ 2.2 の更新がおきている。

また XML 文書がノード単位で編集されることを考慮するとノードレベルでもバージョングラフを表すことができる。これをノードレベルのバージョングラフと呼び文書単位でのバージョングラフと区別する。

2.2 XML 文書のアクセス制御

2.2.1 概　　要

アクセス制御は XML 文書の閲覧や編集が可能であるユーザを制限し、保護すべき情報の漏洩や改竄を防ぐ技術である。また一つの文書をユーザに応じて様々な形式で見せるインターネット上での課金サービスやグループウェアでのユーザによる編集や閲覧の可能な範囲の制限にも用いることができる。

XML 文書のアクセス制御は、XPath データモデルを用いて要素または属性、部分木単位で実施する。またスキーマもしくは個々の文書に対して実施することが可能で前者をスキーマレベル、後者をインスタンスレベルと言う。このような XML 文書のアクセス制御は、多くの研究 [2], [3], [8] がなされている。[3] は基本的な XPath による XML 文書のアクセス制御の研究である。[2] は RDF と $XMLdeclarativeDescription(XDD)theory$ を用

いてアクセスポリシーの記述を行う研究であり、XACML [8] は OASIS の定めた標準である。

2.2.2 アクセス制御モデル

論文 [2] によるとアクセスポリシーは下記の 3 組で表現される情報によって定義する。

$< authorizations, conflict\_resolution, default\_policies >$

$authorizations$ は、権限を表すルールの集合である。$conflict\_resolution$ は、複数のルールが競合したときにどのルールを採用するかを決定する規則である。また $default\_policies$ は、ルールが指定されなかったノードに適用するルールである。以下でそれぞれについて詳しく述べる。

a) Authorization

各ルールは下の 5 組の情報によって表現される。

$< subject, object, privilege, type, sign >$

- $subject$:ユーザ個人またはグループ、ロールなど
- $object$: $< target, path >$ によって表現される。$target$ は対象の XML 文書またはそのスキーマを表す。$path$ は対象の要素もしくは属性ノードを表す XPath 式である。
- $privilege \in \{r, w\}$ r は閲覧、w は更新、編集の権限の指定であることを表す。
- $type \in \{no\_prop, cascade\}$ no_prop は $path$ で表されたノードのみを対象とする。$cascade$ は $path$ で表されたノードを根とする部分木を対象とすることを表す。
- $sign \in \{+, -\}$ + は許可、− は拒否のルールを表す。

この組により $subject$ で表されるユーザが $object$ の $target$ で表される XML 木またはスキーマ木の $path$ で表されるノード（$cascade$ の場合は部分木）に対して $privilege$ のアクセスを $sign$ が + の場合は許可、− の場合は拒否されることを表す。

b) Conflict Resolution

複数のルールが競合したときに優先順位を与える規則である。下のようなものがある。

- deny prevails 拒否のルールが存在するならば拒否する。
- permit prevails 許可のルールが存在するならば許可する。
- element-descendant prevails 木構造の祖先または自身の中でより低いレベルのノードに指定されているルールを適用する。
- instance prevails インスタンスレベルで指定されているルールをスキーマレベルのものより優先する。

上から 2 つは単独でも結果が決定する。しかし下の 2 つについては同じレベルで競合するルールが存在することがあるため補助的に上の 2 つを使用する必要がある。

c) Default Policies

適用されるルールが存在しない要素や属性に対して適用されるルールである。次の 2 種類が考えられる。

- Open-Policy ルールが存在しなければ許可する。
- Closed-Policy ルールが存在しなければ拒否する。

図 2 は複数の人間で文書を編集する例である。この例ではある組織のプロジェクトの報告書を共同執筆している。そのアクセスポリシーを図 3 で記述している。必要外のデータの破壊を防ぐために部門ごとに編集可能な範囲を制限している。開発部門のユーザは開発の section を編集可能で、運用の section 内の個人データの閲覧が競合解決により拒否される以外は閲覧可能である。運用部門のユーザは運用の section を編集可能でそれ以外は閲覧可能である。幹部は全体の編集が可能である。

```
<?xml version="1.0" encoding="shift_jis"?>
<record id="12345">
  <title>報告書</title>
  <description>報告書概要……</description>
  <section title="開発">
    ……
  </section>
  <section title="運用">
    ……
    <ptivateData>……</ptivateData>
    ……
  </section>
</record>
```

図 2　XML 文書の例（record.xml）

```
conflict_resolution = "deny prevails"
default_policies ="Closed-Policy"
<Developer,<redord.xml,/record,r,cascade,+>
<Developer,<record.xml,/record/section[@title="開発"]>,w,cascade,+>
<Developer,<record.xml,/record/section[@title="運用"]/privateDate,r,cascade,->
<Operater,<redord.xml,/record,r,cascade,+>
<Operater,<record.xml,/record/section[@title="運用"]>,w,cascade,+>
<Manager,<record.xml,/record>,w,cascade,+>
```

図 3　アクセスポリシーの例

3. 統合モデルの要求

XML 文書のバージョン管理とアクセス制御を個別に行うことによって生じる問題を整理し、統合モデルに要求される機能を明らかにする。特に複数のユーザが編集を行い、公開する協調作業文書管理システムを例にとって考察を行う。まず 3.1 節ではこれまでのアクセス制御モデルで更新のある文書を管理した場合の問題について述べる。3.2 節ではバージョンの生成日時や有効期間などを用いたアクセス制御についてまとめて 3 節で統合モデルに必要な機能について議論する。

3.1 更新を考慮しないアクセス制御の問題

更新が行われる XML 文書には、保護情報が複数のバージョンの文書に含まれる。そのためバージョングラフ上に存在する全てのバージョンの文書に適切なアクセスポリシーを設定しなければ、複数のバージョンの文書を参照することにより保護情報の漏洩が生じる可能性がある。また文書内で編集を行う部分を制限する場合も明示的な変更がない限りバージョン間で一貫性がある必要がある。この条件下でのこれまで提案されてきたアクセス制御の問題点について例を挙げながら考察を行う。

3.1.1 スキーマレベル

スキーマレベルでアクセス制御を行う場合、そのアクセスポリシーは個々の文書だけではなくそのスキーマに従う文書全体に影響を及ぼす。しかし XML 文書の持つ情報は文書ごとに異なりアクセスポリシーも非常に多岐に及ぶ。

例えば図 2 の例は報告書である。スキーマはこの組織で一般的な報告書を記述するスキーマであるとすると他にも多くのこのスキーマに従った報告書が存在するはずである。報告書の内容次第で編集や閲覧可能なユーザや範囲が異なるはずである。このようなアクセス制御をスキーマレベルで行うためには各文書を識別し膨大な可能性の中から適用するアクセスポリシーを

決める必要がある。文書内にアクセス制御のための属性や要素を埋め込む方法によって文書の識別を行う方法もあるがアクセスポリシーの記述や管理が複雑になる。また実質的にインスタンスレベルでの指定であり後に述べるインスタンスレベルでのアクセス制御と同じ問題点を持っている。スキーマレベルでアクセス制御を行う利点はスキーマ全体に一定のアクセスポリシーを適用できることであって、文書ごとのアクセスポリシーの差が大きいものには向かないのである。

しかし更新を考えた場合、更新の元になるバージョンと更新後のバージョンのアクセスポリシーは多くの場合似通っていることが予想される。例えば非公開で編集中の文書は完成するまで公開されない。しかし一度公開されたならば何らかの事情がなければ非公開にならないなどである。複数のバージョンを含むという点でスキーマレベルのアクセスポリシーに似ているがそれでは範囲が広すぎることは先に示した通りである。

以上のようにバージョン管理を考えた場合スキーマレベルより範囲が小さくバージョンの履歴関係を利用した範囲を指定してアクセスポリシーの記述を行う必要がある。

3.1.2 インスタンスレベル

次にインスタンスレベルで指定する場合を考える

a) 新たなバージョンの文書に対するアクセスポリシー

インスタンスレベルの場合、文書毎の必要に応じたアクセスポリシーの指定が可能である。一方で対象の文書の構造や内容に依存し、かつ他の文書には影響を及ぼさないアクセスポリシーでもある。そのため新たに生成されたバージョンの文書に対するアクセスポリシーを設定する必要が生じる。簡単な方法として更新前のアクセスポリシーをそのまま採用する方法がある。この方法の利点はスキーマレベルでのアクセスポリシーに近い。バージョングラフに含まれる文書の全体または一部に同じアクセス制御を実施できるという点で有用である。

しかし更新によって構造が変化するとノードのパスが変わる可能性やコピーによって複製が生じる可能性がある。スキーマに適合しなくなるような更新を禁止することも考えられるがXHTML のように要素の移動や複製が容易なスキーマや編集中のバックアップを考えるとこれらの可能性を考慮する必要がある。このような更新が起きた場合、更新前のアクセスポリシーをそのまま適用しても本来保護すべき情報の保護が可能であるとは限らない。そのためアクセス制限がかけられているノードについて、更新内容を考慮し移動や複製を追うことで更新後の文書内のパスを見出し元のノードに指定されていたアクセスポリシーを引き継ぐ機能が必要である。次のような例を考える。

図 4 の v1 → v2 は図 2 の更新を行い構造が変化した例である。新たに文書を構造化し Developer に対してアクセス拒否をしていた privateData 要素が subsection 要素の子になりパス式が変更になっている。この変更が起きると元の文書のアクセスポリシーに用いて情報を保護することはできない。そのためどのような変更が起こるか分からない将来生成されるバージョンに対しても予めアクセスポリシーを指定する機能が必要となる。これはノード単位でアクセスポリシーを引き継がせることが要求される。しかし単純にノードを追っただけでは問題が起こる

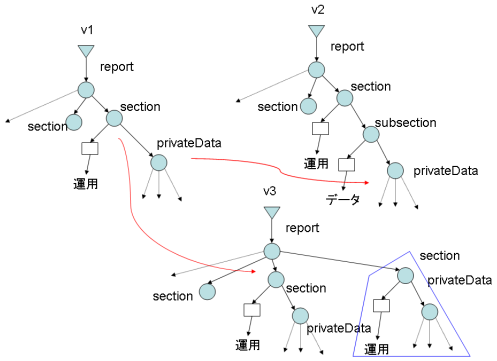

図 4　更新によって構造が変化する例

例もある。例えば図 4 の v1 → v3 は図 2 に更新が起こり運用に関する報告が一つ加えられ $<$ *section title* $=$ ”運用” $>$ 要素ができた場合である。運用部門のユーザはこの要素の編集が可能であることを期待されるが、元々許可されていたノードの直接的な子孫ではないため先の方法では編集を許可されない。このためバージョン間の関連性が高い範囲について同じパス式でアクセス制御を実施する機能が要求される。

b) 過去のバージョンのアクセスポリシーの変更

次に過去に生成されたバージョンのアクセス制御が持つ問題ついて述べる。過去に遡ってインスタンスレベルでアクセス制御を行った場合、既に更新が幾度も行われ文書のバージョングラフが発達した後にアクセスポリシーを変更したときに問題が生じる。先に出た図 2 の例で考える。

- 運用部門にアルバイトを雇うことになった。
- ある運用部門の報告書過去に遡って読む必要が生じた。
- 個人情報にはアクセスできないようにする。

アルバイトのアクセスポリシーが存在せず、Default Policies が拒否であるためそのままで報告書を見ることができない。そのため、これまでに生成された全バージョンついてアルバイト用のアクセスポリシーを加える必要がある。この作業はひどく手間が掛かるものである。さらに図 4 のような構造の変更が起きる場合は個々のバージョンに対して正しいパスを見出す必要がある。構造が複雑になり変化が大きくなると人の手による正しい指定は不可能と言ってよい。過去のバージョンの文書についてもノード同士の対応を管理しバージョングラフに含まれる文書全体を統一的に扱うアクセスポリシーを指定できる機能が必要である。

3.2 時間情報を用いたアクセス制御

バージョン管理の応用例を考えた場合時間情報は非常に重要である。先の報告書の例でいえば報告書の有効期限が終わると社員は読むことができなくするアクセス制御の例が挙げられる。また時間制約が特に顕著な応用例はニュースサイトなどの情報サイトで、更新から一ヶ月のみ情報を一般に公開しその後は有料会員のみが情報を閲覧可能にしたり、ある情報については会員に対して先行公開を行ったりといったアクセスポリシーが考えられる。バージョンの生成日時や有効期間はバージョン管理のシステムでは標準的な機能であるが、XML 文書がこれらの時間情報を含むことは仮定できない。またバージョン管理

との連携をしなければ取得する枠組みもない。そのためアクセス制御モデル単独でのスキーマや文書内容に依存しない実現は困難である。

またアクセスポリシーの有効な文書を限定する条件付けという点でも時間情報は重要である。例えば 2004 年 1 月 1 日以前に有効期限が終わった報告書については非公開とするといったアクセスポリシーの条件付けには時間情報が不可欠である。統合モデルではアクセス制御のための情報としてこれら時間情報をバージョン管理システムから取得できることを利用してアクセス制御モデルに有用な機能を加える必要がある。

3.3 統合モデルの要求機能

3.1 および 3.2 から以下の要求があることがわかった。

（1） 履歴関連性の高い複数のバージョンの文書に対して同じパス式でアクセス制御できること

（2） 新しいバージョンの情報保護

（3） バージョン間で対応するノードに対して同じアクセスポリシーを指定できること

（4） 過去のバージョンのアクセスポリシーの変更の容易さ

（5） 時間制約によるアクセス制御

これらの要求について我々はバージョン間の差分を利用してノードレベルのバージョングラフを求めることで解決できると考えた。１は文書レベル、３、４についてはノードレベルのバージョングラフを利用し、履歴関連性の強い複数のバージョンを一つのアクセスポリシーで情報保護を行う。対象のノードはバージョンとパスを指定することによってノードを特定する。また文書のバージョン間をパス式を変えずに伝播する方式とノードのバージョングラフをたどる方式の二通りを備える。新たに生じるバージョンもこの範囲に含めることで２を実現する。また更新日時などによる時間制約を行い、アクセスポリシーの有効範囲の制限としても用いる。

4. 統合モデル

ここでは 3 章で述べた要求を備えたバージョン管理とアクセス制御の統合モデルをモデル化する。まず 4.1 においてデータモデルについて議論し 4.2 で複数のバージョンを含むことが可能になるようなアクセスポリシーの定式化を行う。そして 4.3 ではバージョングラフ上の履歴関係を用いた範囲の指定方法について議論し、4.4 では時間制約を提案する。4.5 において拡張されたことによる新たな競合の解決手法について述べる。最後に 4.6 でここまでで提案したモデルを用いて 3 章で検討した例題のアクセスポリシーを記述し要求を満たしているか検討する。

4.1 データモデル

編集スクリプトの列と文章の親子関係を保存しバージョン管理を行う。編集スクリプトによってバージョン間のノードの移動や複製をより正確に取得するためである。XPath を用いて XML 文書のノードを指定する場合、パス式によって得られるノードは一般に集合である。しかし XML 文書は順序木であるためノードの一意の指定が可能であり、XML 木をラベル付順序木とみなすことができる。本研究においてもラベルを用いてノードレベルのバージョングラフと編集スクリプトを管理する。

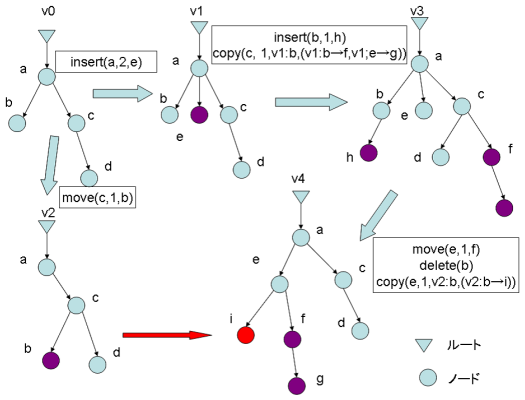

図 5 更新によるバージョングラフの成長

以後簡単のためにノードの型は区別しない。本研究では複数のバージョンから一つの文書を生成する状況を考える。主となる親はルートノードの元となった文書とする。そこに他の文書からの要素が copy することで文書のマージを実現する。そのため copy についてはどのバージョンから copy されたかを明示するためにバージョン v のラベル n のノードを v:n と表す。主となる文書に他の文書から必要な情報を組み込んでいく操作である。編集スクリプトは以下の 5 つを考える。

- delete(m) ノード m をルートとする XML 木を削除
- insert(n,k,m) ノード m を n の k 番目の子供として挿入
- update(m,v) ノード m の値を v に変更
- move(n,k,m) ノード m をルートとする XML 木をノード n の k 番目の子供の位置に移動
- copy(n,k,v:m,M) バージョン v のノード m をルートとする XML 木を n の k 番目の子供の位置に複製する。複製したノードには XML 文書内で一意にするために新たなラベルをふる。そのマッピングが M である。例えば更新により a が b、c が d に変更になった場合 M=(v:a → b,v:c → d) のように表す。

XML 文書のアクセス制御は属性、要素単位で行うため、構造の変化を起こさない update は無視する。copy でラベルが変えるのは複製されたノードが、同じアクセスポリシーを指定されるとは限らないため、別の物として処理する必要があるからである。この場合でも編集スクリプトを参照することで祖先のノードを確認する。他の編集スクリプトでは更新によるラベルの変更は行わない。このようにすると元のラベルと copy によって新たに分岐したラベルのノードは親子関係にあるノードであるとみなすことが可能である。こうしてバージョングラフの成長を表した図が図 5 である。ノードレベルのバージョングラフは各バージョンから同じラベルのノードとさらに copy によって分岐したノードを取得することで取得する。図 6 は図 5 から v3:f のノードレベルのバージョングラフを取得した例である。copy(c, 1,v1:b,(v1:b → f,v1;e → g)) と copy(e,1,v2:b,(v2:b → i)) から f と b と i が等しいことが分かる。

編集スクリプトの精度上記のモデルによってノードの移動や複製を取得する上での問題は編集スクリプトの精度である。本研究の目的は情報保護であるため、得られる編集スクリプトが本来ユーザが意図するノードの対応を持っている必要がある。

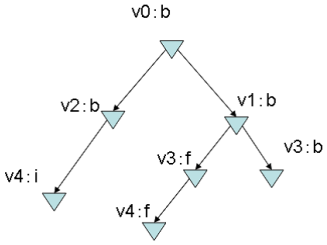

図 6　v3:f のバージョングラフ

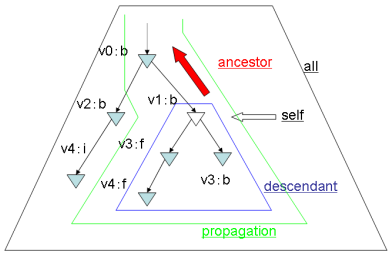

図 7　バージョングラフ上の関係

この編集スクリプトの取得方法の問題についてここで議論する。

編集スクリプトの取得方法には大きく二通りある。まずは要素、属性単位で操作を行うエディタやプログラムから直接得る場合である。次に XML 文書をテキストエディタで処理した場合や更新後の文書しか得られない場合である。前者についてはノードの移動や更新を保持していることを仮定できる。後者の場合、更新前後の文書を比較することで差分を発見し編集スクリプトを得ることになる。しかし似通ったノードが複数あり、その交換や更新が起きた場合、ノードの対応を正確に取得することは困難である。解決法の一つとしては文書内の全ての要素に一意となる ID 属性を持たせる方法がある。属性を要素の性質を決める物として位置づけ、属性単独での move、copy を行わないという制約を付けるによって要素の移動、削除の取得を解決する。ところが文書内で一意であることを遵守するならば ID 属性によって copy を管理することはできない。この解決策としては copy した要素に同じ ID をつけ、システムに入力された時点で片方の ID を変更し管理する方法が考えられる。しかしこの管理法は更新時の ID の管理をシステムではなくユーザが手作業で行うことになるため作業ミスなどによって十分な安全性の取得は困難であると思われる。以後本研究では前者を仮定し編集スクリプトの精度は十分にあるものとする。

4.2　アクセスポリシーの記述

要求機能をもつアクセスポリシーを定式化する。2 章で述べたアクセスポリシーの object の内容を変更し scope と duration を追加した以下の 7 つの情報の組で表す。

$$< subject, object, scope, duration, privilege, type, sign >$$

詳細は以下の通りである。

- $object$：$< file, version, path\_exp >$ の 3 組で表し、対象のバージョンを $version$ によって表す。
- $scope < ver\_scope, prop\_type >$ で表し、バージョングラフを利用したアクセスポリシーの有効範囲の指定を行う。$ver\_scope \in \{descendant, self, ancestor, propagation, all\}$ はバージョングラフ上の関係で、この範囲にアクセスポリシーが有効になる。$prop\_type \in \{node, path\}$ は文書とノードのどちらのバージョングラフを用いるか決定する。詳しくは 4.3 で説明する。
- $duration$ バージョンの生成日時や有効期間を利用した時間制約を記述する。この制約が真となる文書や操作にアクセスポリシーを適用する。詳しくは 4.4 で説明する。

$scope$ はバージョン間での範囲指定、$duration$ は時間による範囲指定である。両方が有効である範囲の文書に対してのみアクセスポリシーを適用する。

4.3　バージョングラフの関係を利用した範囲指定

図 7 のように祖先、子孫、自身といったバージョングラフ上の履歴関係を利用してアクセスポリシーを指定する。このとき、使うバージョングラフはノードレベルと文書レベルの二通り考える。前節で述べた定式化した $scope$ において、ver_scope で祖先や子孫の関係を示し $prop\_type$ でノードレベルか文書レベルかを決める。$descendant$ は祖先、$ancestor$ は子孫、$propagation$ は祖先と子孫を含み、all はバージョングラフ全体である。$self$ はそのバージョンのみである。$prop\_type$ については $node$ はノードレベルでバージョングラフをたどり $path$ は文書レベルでバージョングラフをたどる。この機能を用いるとすでにある保護情報について将来更新によって生成される文書のアクセス制御を事前に行うことができる。XPath 式を用いたノードレベルのアクセスポリシーを適切に他のバージョンに適用できるように次のような枠組みに従うこととする。

（1） 対象のノードはあるバージョンの文書のノードを XPath で指定することによって決める。

（2） ノードレベルで範囲指定を行う場合は、このノードのバージョングラフを対象とする。

図 7 のうち $descendant$ を含む範囲を指定したときには、将来生成されるバージョンに対してもアクセス制御を実施することが可能である。例をいくつかあげる。

- <ancestor,path> 文書レベルのバージョングラフの祖先を範囲に含む。このときパス式は変えない。
- <descendant,node> ノードレベルのバージョングラフの子孫を範囲に含む。
- <self,path> 他のバージョンに影響しない。
- <all,path> 全バージョンに適用する。

4.4　時 間 制 約

バージョン情報に含まれる時間情報を利用したアクセス制御を行う機能である。バージョンの生成日時や有効期間を用いた時間制約をアクセスポリシーに記述する。指定した期間に指定した日時が入っているかどうかで条件付けを行う。ただし更新時間以降や有効期間終了までのような片側の制限の無い物や時間制約をつけないことも可能とする。

また複雑な条件を記述するために and と or を用いて複数の制約が同時に成立するかどうかを表せるようにする。これらを目的に 4.2 のアクセスポリシー式の $duration$ の表し方を以下のように定義する。A と B は真または偽を表し and,or,in,out の

いずれかが入る。

- $and(A, B)$ A と B が共に真のときに真
- $or(A, B)$ A と B の少なくとも一方が真のときに真
- $in(t, interval)$ 日時 t が期間 interval に含まれるときに真
- $out(t, interval)$ 日時 t が期間 interval に含まれければ真
- 空欄 条件無し

$interval$ の表し方

- between(t1,t2) 日時 t1 から t2 の間
- between_m(m1,m2) m1 月から m2 月の間 m1>m2 の場合は次の年の m2 月まで
- between_d(d1,d2) d1 日から d2 日の間 d1>d2 の場合は次の月の d2 日まで
- between_t(t1,t2) 時刻 t1 から t2 の間 t1>t2 の場合は次の日の時刻 t2 まで
- before(t1,du) 日時 t1 で終わる期間 du
- after(t1,du) 日時 t2 から始まる期間 du

日時、期間の表記は ISO8601 に従う。日時には具体的な日付の指定のほかに以下のものを指定可能とする。

- $vf(v)$ バージョン v の有効期間始まりの日時
- $vt(v)$ バージョン v の有効期間終わりの日時
- $up(v)$ バージョン v の生成日時
- $access$ ユーザによるアクセス日時

バージョンを表す v には、具体的なバージョンの他に$v を指定可能とする。この$v にはアクセスの対象となった文書のバージョンが入る。日時には ∞ と $-\infty$ も指定可能とする。∞ や $-\infty$ と between を組み合わせることで片側の制限が無い時間制約を記述する。時間制約の例をあげる。

2004 年 1 月 1 日 0 時が有効期間に含まれない文書に対して適用

$$out(2004-01-01T00:00:00, between(vf(\$v), vt(\$v)))$$

2004 年 1 月 1 日 0 時までの一週間以内に有効期間が切れる文書に対して 2004 年 1 月 1 日 0 時以降のアクセスに適用

$$and(in(vt(\$v), before(2004-01-01T00:00:00, PW1)), in(ac, between(2004-01-01T00:00:00, \infty)))$$

4.5 競合解決

ここまで述べてきた統合モデルでは、新たなアクセスポリシーの競合問題が生じる。あるノードの対する操作の許可決定に影響を与えるものとして、これまでの文書内に指定されているアクセスポリシーに加えて他のバージョンから伝わるアクセスポリシーがあるからである。この問題を解決するために考えられる競合解決は以下のようなものがある。

- Deny-prevails 拒否のポリシーが存在するならば拒否
- Permit-prevails 許可のポリシーが存在するならば許可
- Element-descendant-prevails 文書構造上の祖先、もしくは自身のうち最も子孫にあたるものを適用する。
- Version-descendant-prevails バージョングラフ上の関係でより子孫に対して指定されたものを適用する。
- Newer-prevails より生成日時の新しいバージョンを対象としているポリシーを適用する。

下の 2 つについては単独では文書内での競合が解決せず、Element-descendant-prevails ではバージョン間の競合が解決しない。また同じバージョンの同じノードに異なる結果を返すアクセスポリシーがある場合、少なくとも Deny-prevails または Permit-prevails を補助的に用いる必要がある。情報保護の観点からは、一つでもアクセスを拒否するアクセスポリシーがあると拒否をする Deny-prevails が有効である。しかし非公開になっていた要素を公開にするためには元のアクセスポリシーを変更するしかない。またあるバージョンだけ公開したりあるバージョンから派生したバージョンだけを公開するといったアクセスポリシーの記述が複雑になる。そのため上で示したような競合解決手法が有効である。

4.6 検　　討

ここでは統合モデルを用いて 3 章において問題にした例についてアクセスポリシーを記述し要求を満たしているか検討する。

c) ノードの位置が変化する例

3.1.2 の図 4 の v1 → v2 で v1 の privateData 要素に対して Developer の権限を記述し、v2 の privateData 要素へのノードレベルで範囲を指定しアクセス拒否を行う。

<Developer,<record.xml,v1,
/report/section[@title=”運用”]/privateData>
,<descendant,node>,,r,cascade,- >

d) 要素が挿入される例

3.1.2 の図 4 の v1 → v3 で挿入された section 要素を v1 から Developer が編集できるようにする。パスが変わっていないため、文書レベルのバージョングラフで範囲を指定する。

<Developer,<record.xml,v1,
/report/section[@title=”運用”]/privateData>
,<descendant,path>,,w,cascade,+>

e) 過去のアクセスポリシーを変える例

3.1.2 の例において図 4 の v3 の文書の一つ目の section 要素で表されている文書をアルバイトが過去に遡って閲覧可能にする。そのとき privateData 要素は閲覧を拒否する。

<PartTimer,<record.xml,v3,
/report/section[@title=”運用”]/privateData>
,<propagation,node>,,r,cascade,->
<PartTimer,<record.xml,v3,
/report/section[@title=”運用”]/privateData>
,<propagation,node>,,r,cascade,+>

f) 時間制約の例

そのバージョンの有効期間が終わると社員は record.xml を読むことができなくなる。

<Member,<record.xml,v1,/report>,<all,path>
,in(access,between(vt($v),∞)),r,cascade,->

2004-01-01 以前に有効期間が終了しているバージョンを読むことはできない。

<Member,<record.xml,v1,/report>,<all,path>
,in(vt($v),between(−∞,2004-01-01T00:00:00)),r,cascade,->

このように 3 章で挙げた例のアクセスポリシーを簡潔に書くことができた。これまで個々の文書に対して記述するしかな

かったが、少ないアクセスポリシーでバージョン間の関連性を用いた記述することが可能であることが分かる。

5. 応　　用

4 章で提案したモデルではバージョングラフを有向非巡回グラフと想定し、また時間制約が可能である。このことによって様々な応用を考えることができる。例を挙げて統合モデルの利用方法を考察する。

g) グループウェア

wiki に代表されるように多くのユーザが編集できる環境は故意の改竄や不注意によるデータの破壊を防ぐことは難しい。そのためバックアップとしてバージョン管理が必要である。直線状のバージョン管理では過去のバージョンへの復帰や並行編集が生じた場合に、バージョン間の繋がりの管理が管理できなくなる。またコミュニティ内での使用を考えるとアクセス制御が有用である。そのため分岐や合流を許したバージョン管理とアクセス制御を行える統合モデルが有効である。

また 2 章で例示したようなスレッド型の掲示板の場合、各スレッドをバージョングラフであると見なすことが可能である。一つの掲示板内でスレッド毎に書き込みの可能なユーザや閲覧可能なユーザを制限することは有用である。またスレッドの途中からアクセスポリシーを変えるような利用方法も考えられる。例えば図 1 の場合、d1 から続く話題は公開するが d2 から続く話題については公開できない情報が入るために閲覧可能なユーザを制限するといった使い方である。

h) 有料情報サイト

情報の更新や内容の関連をバージョンとして扱うことができる。統合モデルを用いるとユーザを無料ユーザと有料ユーザに分け金額によって、更新後の閲覧可能な期間の限定や詳細度を分けるといった例が考えられる。

i) 電子カルテ

検査や治療により日々蓄積され患者毎にアクセスポリシーが異なる例である。転院や診療科が変るときなどに、それまでのデータのアクセスポリシーを変更する必要がある。このとき、複数のバージョンを持つデータを処理する必要があり、バージョン単位で管理できる統合モデルが有用となる。

6. 関 連 研 究

XML 文書のバージョン管理とアクセス制御の統合に関する研究に [4] がある。バージョン管理とアクセス制御の統合について考察し、一つのフレームワークで扱うことを目的とした研究である。バージョン情報やアクセスポリシーを RDF と RDF Declarative Description (RDD) theory を利用して記述し、関連付けを行うことで統一的な処理をすることを目的としている。この研究では各バージョンの文書に対しアクセスポリシーを割り当てているが、その更新に対応するアクセスポリシーの変更はユーザが更新の条件と共にアクセスポリシーを用意する手法をとっている。そのため将来生じる変更に対応したアクセス制限を行うためには、適当なアクセスポリシーと条件を用意する必要がある。あらかじめ変更を全て予測することは困難である。その場合は元のバージョンのアクセスポリシーを適用するがこのときに情報漏洩の可能性がある。またバージョン管理の重要な要素である時間情報を利用した拡張については解決していない。新たなバージョンに対して予めノード単位でアクセス制御を行える点と時間情報による制約をアクセスポリシーに含める点が本研究の優位点とである。

研究 [6] は時制 XML 文書のアクセス制御の研究である。時間が経つにつれて変化する XML 文書を X-Tree と呼ばれるデータモデルで表しアクセス制御を行っている。X-Tree は木構造で表した XML 文書の有向枝に時間情報を付加し値を管理することで文書の変化を保存しており時間による質問に強いモデルである。アクセス制御モデルも時間制約に注目して考えられており複雑な時間制約を持ったアクセスポリシーを記述可能である。しかし時制データはバージョングラフが直線状になる例である。バージョングラフが分岐や合流した場合にもアクセスポリシーの記述をできる点が本研究の優位点である。

7. お わ り に

本論文では、バージョン管理の行われる複数のバージョンを持つ XML 文書のアクセス制御をより確実に効率よく行うための統合モデルの機能のモデル化を試みた。

まず複数のバージョンのある XML 文書のアクセス制御をスキーマレベルやインスタンスレベルで行う場合の問題を整理し、バージョンを考慮した範囲指定が必要であることを述べた。またバージョンの生成日時などをアクセス制御で利用する利点について述べた。我々は要求を満たすための機能として文書レベルとノードレベルのバージョングラフを利用して範囲を指定する機能と時間による制約を持つ機能を提案した。次にこれらの機能を実現するデータモデルを提案し、アクセスポリシーの定式化を行った。また統合モデルの応用例を提示した。今後の課題としては、ノードの更新日時の利用や実装による評価がある。

文　　献

[1] A. Marian, S. Abuteboul, G. Cobena and L. Mignet. “Change-Centric Management of Versions in an XML Warehouse,” Proc. 27th Int. Conf. on Very Large Data Bases, pp.581–590, 2001.

[2] C. Anutariya, S. Chatvichienchai, M. Iwaihara, V. uwongse and Y. Kambayashi. “A Rule-Based Access Control Model,” Proc. 2nd Int. Workshop on Rules and Rule Markup Langauges for the Semantic Web, RuleML, pp.92–103, 2003.

[3] E. Bertino, S. Castano, E. Ferrari and M. Mesiti. “Specifying and enforcing access control policies for XML document sources,” World Wide Web,Vol.3, pp.139–151, 2000.

[4] S. Chatvichienchai, C. Anutariya, M. Iwaihara, V. Wuwongse and Y. Kambayashi. “Towards Integration of XML Document Access and Version Control,” Proc. 15th Int. Conf. on DEXA 2004, Springer LNCS3180, pp.801–810, 2004.

[5] S. Chien, V.J. Tsotras and C. Zaniolo. “Efficient Management of Multiversion Documents by Object Referencing,” Proc. 27th Int. Conference on Very Large Data Bases, pp.291–300, 2001.

[6] Sabrina De Capitani di Vimercati. “An authorization model for temporal XML documents,” Proc. 2002 ACM symposium on Applied computing, pp.1088–1093, 2002.

[7] 天笠 俊之, 吉川 正俊, 植村 俊亮. “時制 XML 文書のためのデータモデル,” 情報処理学会研究報告書, Vol. 122, pp.161–168, 2000.

[8] XACML1.0 Specification Set. OASIS Standard (Nov. 2002). http://www.oasis-open.org/committees/xacml/