# 時刻認証付き XML 文書のデータベースによる管理について

陳　明強†　　吉川　正俊‡
†名古屋大学大学院情報科学研究科　〒464-8601　名古屋市千種区不老町
‡名古屋大学情報連携基盤センター　〒464-8601　名古屋市千種区不老町
E-mail: †cmqiang@dl.itc.nagoya-u.ac.jp　‡yosikawa@itc.nagoya-u.ac.jp

**あらまし**　デジタル時代において,様々なデータが電子化され,他人から転送されてきたとき,そのデータがその時間に本当に存在したことを証明するための時刻認証というサービスが出て来た．一方で,XML は単純な構造と,人間にも機械にも理解しやすい特徴から,現在,電子化された情報の標準的な記述フォーマットの一つになっている．大量な XML データを処理するために,XML データベースを用いる．本論文では XML データ及びそれに関する時刻認証情報を格納した XML データベースの実現を提案する．

**キーワード**　時刻認証，XML ，データベース

# Managing XML Documents with Time Certification using Database

Mingqiang Chen†,　Masatoshi Yoshikawa‡
†Graduate School of Information Science, Nagoya University　Furoucyou, Chikusa-ku, Nagoya, 464-8601 Japan
‡Information Technology Center, Nagoya University　Furo-cho, Chikusa-ku, Nagoya, 464-8601 Japan
E-mail: †cmqiang@dl.itc.nagoya-u.ac.jp ‡ yosikawa@itc.nagoya-u.ac.jp

**Abstract**　In the digital era, when various data written in digital format is transmitted from others, there are time certificate service for proving the existence of the data at a specific time. Since XML is becoming one of the standard description formats for computerized information, we use XML database to process a large volume of XML data. In this paper, we propose to store XML data and its time certificate in database.

**Keywords**　Time certification, XML, Database

## 1．はじめに

現在,人間は電子技術のおかけで生活が便利になりつつあり,この反面,実社会と同じように仮想空間では,悪意を持った第三者が本人を偽って本人の行為を行い,電子文書等の重要な情報を不正に書き換える行為,ある事実を後になって否認する行為が起こる恐れがある．電子社会での不安を解消し安全性・利便性が高い社会を構築していくためには,技術面で安全性を保証しなければならない.

電子社会においては,電子データに PKI 技術を用いた電子署名などを用いることにより,「誰が」「何を」「どうしたか」という確認ができる．電子カルテにおける診察時間やオンライン取引時刻など「いつ」という時の痕跡を電子的に証明することに対して,

１）電子データの存在証明：その時刻に確かにその電子データが存在していたこと

２）電子データの完全性：その時刻以降にその電子データが不正に改ざんされていないこと

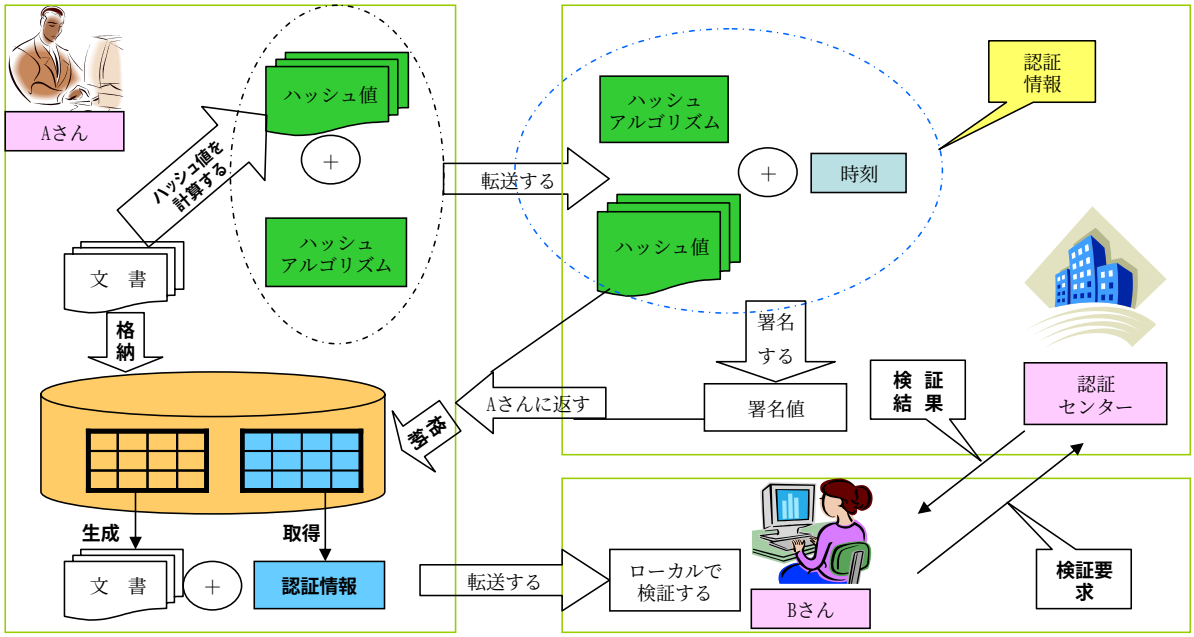



図１：時刻認証の仕組み

を証明することが可能になった．このような要求に応じた時刻認証を行うサービス機関が出現しつつある[1].そこでは対象データに利用者システムの時刻ではなく標準時刻を付与し,そのデータに対して利用者が自分の署名を行う．

1.1　時刻標準

世界の標準時刻とは,「UTC」と呼ばれる協定世界時のことである[2]．これは,各国に置かれたセシウム原子時計の時刻を集め,それらをパリの国際度量衡局(BIPM)で決めている．日本の場合は UTC に 9 時間を足したものが日本標準時となる．ただし日本国内でも計量研究所(NRLM)や国立天文台(NAOJ)など複数の機関による UTC が存在している．精度が最も高いといわれているのは独立行政法人の通信総合研究所(CRL)のものである．

1.2　時刻認証の仕組み

現在,様々な機構が時刻認証サービスを提供しているが,標準的な仕様はまだ存在しない．また特定の形式の文書に対して時刻認証はできるが,より汎用な電子データについての時刻認証モデルはない．本論文では,標準化の時刻認証を提案する．

図１のように,利用者が時刻認証を受けたい文書に対してあるハッシュ関数をかけ,生成されたハッシュ値に個人情報やハッシュ関数名などを付加し,時刻認証サービスを提供している認証センターに転送する．認証センターが送られてきたハッシュ値,ハッシュ関数名とその時点の標準時刻を合わせ,電子署名を行う．署名付きの時刻認証値（よく電子タイムスタンプと呼ばれる）を利用者に返す.利用者がもらった署名付きの時刻認証値と原文を第三者に渡す．第三者での文書検証がローカルでできない場合は,文書の検証請求を認証センターに転送する．認証を行うとき,まず認証センターの電子署名により,ハッシュ値やハッシュ関数名や認証時刻などの認証情報が改ざんされていないをチェックし,改ざんされていなければ,認証時刻の時点までそのデータが確かに存在したことを信じることができる．次は原文から信じられたハッシュ関数でもう一度ハッシュ値を計算し,得られたハッシュ値と認証情報におけるハッシュ値を比較し,結果が同じであれば原文が改ざんされていないと判定する．検証を認証センターで行う場合は,検証結果を請求者に返す.

利用者が図２のような XML で記述した認証請求を認証センターに転送する．認証請求での Reference タグでは,認証文書 URI だけではなく,認証範囲を指定した XPath を含めることもできる．従って,部分文書の認証も可能である.認証センターでは認証情報に対して XML 電子署名[3]を行う．全ての処理が標準化されたので，時刻認証

```
<Certificate_Request xmlns=`dl.itc.nagoya-u.ac.jp/`>
    <Requester>請求者の情報</ Requester >
    <Document name=``,size=``.....>
        <Reference> 請求文書のURI＋XPath< /Reference >
    </Document>
    <Hash value> .... </Hash value>
    <Algorithm> .... </ Algorithm>
    ..........
</Certificate_Request>
```

図２: 認証請求

が具体的な電子データの形式及び認証センターに依存しない利点でより広い範囲で使われる.

この時刻認証の仕組は既存の時刻認証処理と比較すると,以下の利点がある．１）原文ではなくハッシュ値だけを伝送するので,文書のサイズが大きい場合はインターネットでの転送時間が減少できる．また原文内容が漏れる可能性を避けることができる．２）現在,様々な形式の文書に対して時刻認証できるが,実際には時刻認証を提供した会社によって実現の手法が全く異なるため,文書の検証が特定の会社に依存する欠点がある.標準化すれば,具体的な文書形式やツールに依存しなく,標準をサポートしたツールがあればよい.３）認証情報が別に保存されるので,それと原文を別々にデータベースへ保存することが可能になった.

日本では,2005 年 4 月から施行される e-文書法[4]により,契約書など紙文書が電子データで表現できる．電子データの数量が増加するとともに,データベースで管理する必要が高くなっていくと思われる．一方で,電子契約書や電子カルテや特許・知的財産保護など数多くの場合でデータの時刻認証をしなければならない,もし電子データ及びそれの認証情報が一緒にデータベースに格納され,データの使用者はいつでも認証情報により既存のデータの信頼性が確認できるようにすれば,安心して使用できる．図１のように認証された文書及び認証情報をデータベースへ格納する.検証または第三者に転送するときには,またデータベースによる元の文書を生成し,認証情報を取得する.

様々な形式の電子データに対して時刻認証できるが,本論文では,XML 文書,または他の形式で記述され XML 形式に変換された電子データを認証対象として標準的に時刻認証を行ってから,原文と認証情報を格納する信頼性の高い XML データベースの実現を提案する.

## 2．XML データベース

W3C(World Wide Web Consortium) から仕様が勧告された XML(Extensible Markup Language) は様々な文書やデータを表現するためのメタ言語である．現在 XML を利用したアプリケーションが多数考えられており,今後 XML で記述された文書,データが増加していくと思われる．従って,大量な XML 文書をデータベースで効率的に管理する必要がある．このとき,関係データベースを用いる利点として次の三つをあげることができる．1) 関係データベースが既に多くの応用システムで用いられていること．2) 大量のデータが関係データベース上に蓄積されていること．3) 関係データベースに実現された問合せ最適化やトランザクション管理などが利用できること.

XRel[5]は我々が開発している XML データベースシステムである.XRel では XML 文書を分解して固定したスキーマを持つ 4 個の関係に格納する．関係スキーマは DTD や文書の論理構造に現れる要素型に依存しないので,文書の論理構造の変化が関係スキーマに影響を及ぼすことはない．XML 文書の各要素はラベルと XML 木における根からの経路によって管理される．XRel では XML 問合せ言語 XPath の問合せを SQL 問合せに変換し,実行している.従来の XRel では XML 文書の各ノードの開始タグと終了タグの物理的なバイト位置をデータとして格納することにより XML の木構造を保存していた.この方法では XML 文書にノード挿入が起きた際,挿入位置よりも後に出現する全てのノードの開始タグと終了タグのバイト位置を更新しなければならないため,文書更新のコストが高くなってしまう．この問題を解決するために,ORDPATH[6][7] ラベル付けを利用して XML 文書の木構造を保存する．ノード挿入が起きた際,既存ノードのラベルを更新する必要がない.またあるラベルが他のラベルに含まれるかどうか判断することにより,親子関係を判定できる利点がある.

実際には電子データはよく更新されるので,更新された文書から生成されたハッシュ値が元の文書のハッシュ値と異なるため,検証できなくなってしまう．従って,永

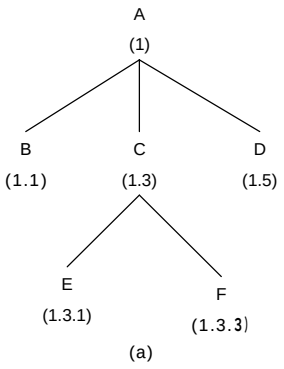
続的な検証を可能とするように原文及び認証情報だけではなく,更新情報も保存しなければならない.

## 3．ORDPATH ラベル付け

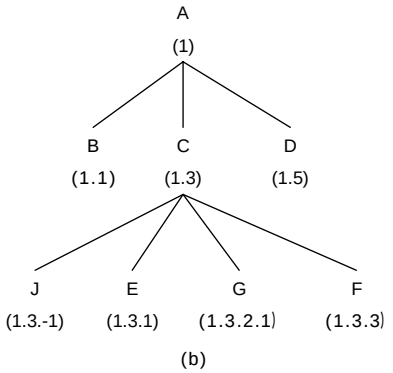
ORDPATH とは XML のラベリング手法の一つで,文書の構造が正確に表現できることが特徴である.またノードが挿入されても既存ノードに対してラベルを更新する必要がなく,かなり高速にノードの挿入を行うことができるラベリング方法である.ORDPATH では文書のスキーマに依存せずにラベルを生成する際,ある親ノードの子ノードには親ノードの ORDPATH から一つ構成要素の数字を付加すればよい．実際は,ORDPATH は数字とドットが並んでいるものである.たとえば,ORDPATH が 1.3 や 1.5 のノードの親ノードは 1 のノードとなる．また 1.3 は 1.5 より文書順において先であるというように数字を単純に比較するだけで判断できる.

ORDPATH が各ノードに対して割り振られているが,どのノードに対しても正の奇数だけが割り振られており,偶数や負数はノードの挿入があった場合のために使用する.あるノードを既存の XML 木に挿入したい時には挿入位置の左の兄弟と右の兄弟の間の数字を ORDPATH として取り,その後に奇数を割り振り,それが新たなノードの ORDPATH となる.

たとえば,ORDPATH が 1.3.1 と 1.3.3 のノードの間に G ノードを挿入する場合は,G ノードの ORDPATH は 1.3.2.1 になる．端に挿入したい場合は ORDPATH の値を拡張する.右端への挿入の場合は右端の兄弟の ORDPATH から数字を２プラスし．逆に左端の場合は数字を２マイナスする.たとえば,図 3.(b)のように E ノードの左に J ノードを挿



(a)



(b)

**図 3：ORDPATHによるノードの挿入**

入すれば,1 に 2 をマイナスすることによる 1.3.-1 の ORDPATH を新しく挿入するノードに割り振る.

## 4．XML データベースへの格納

時刻認証された文書およびそれに関する認証情報をデータベースへ格納する.文書が更新される場合は,更新情報をデータベースへ保存する.時刻検証を行うときには,更新情報を用いて元の文書または部分文書におけるノードをデータベースから取り出してから,ORDPATH による元の XML 文書を生成する.また生成された文書及び保存した認証情報による時刻検証を行う.

以上三つの情報を格納するのに,以下の四つの関係スキーマを定義する.

- XML 文書の格納用関係スキーマ

Path (pathID, pathExp)：経路を格納する関係.

Node_type(name, type)：ノードのタイプを格納する.様々なノードの格納が可能.

Node (node_ID, docID, index, pathID, type, label, value, insertion_time, deletion_time)：様々なノードを格納する関係.

- 認証情報の格納用関係キーマ

Certification ( certifi_ID, time_center, certifi_path, certified_time, signature）認証情報を格納する.

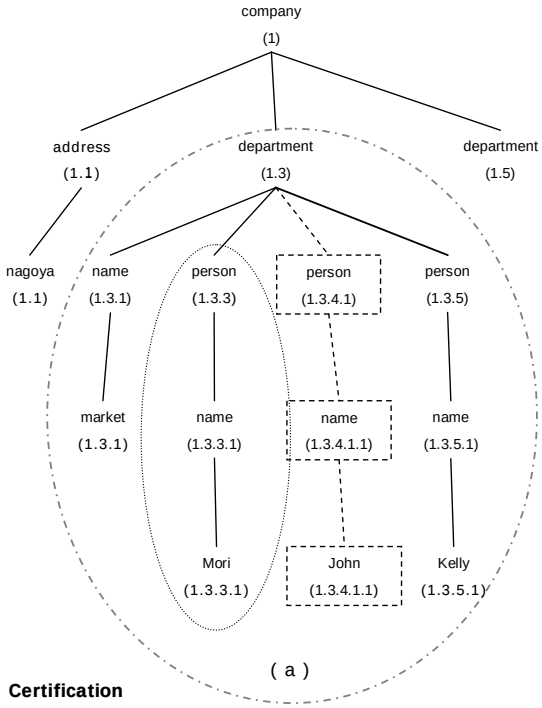
次に,それぞれの属性について述べる.

pathID：経路を識別するための ID である.

pathExp：実際の経路式を格納する.

name：ノードの種類名である.たとえば,text, element, attribute など.

type：ノードの種類名を識別するための ID である.




( a )

**Path**

| pathID | pathExp |
|---|---|
| 1 | /company |
| 2 | /company/@address |
| 3 | /company/department |
| 4 | /company/department /@name |
| 5 | /company/department /person |
| 6 | /company/department/person/name |

( b )

**Node_type**

| name | type |
|---|---|
| element | 1 |
| attribute | 2 |
| text | 3 |
| ... | ... |

( c )

**Node**

| node ID | doc ID | index | path ID | type | label | value | Insertion _time | Deletion _time |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | null | 04/10/8 | null |
| 2 | 1 | 1 | 2 | 2 | 1.1 | null | 04/10/8 | null |
| 3 | 1 | 1 | 2 | 3 | 1.1 | nagoya | 04/10/8 | null |
| 4 | 1 | 1 | 3 | 1 | 1.3 | null | 04/10/8 | null |
| 5 | 1 | 1 | 4 | 2 | 1.3.1 | null | 04/10/8 | null |
| 6 | 1 | 1 | 4 | 3 | 1.3.1 | market | 04/10/8 | null |
| 7 | 1 | 1 | 5 | 1 | 1.3.3 | null | 04/10/8 | null |
| 8 | 1 | 1 | 6 | 1 | 1.3.3.1 | null | 04/10/8 | null |
| 9 | 1 | 1 | 6 | 3 | 1.3.3.1 | Mori | 04/10/8 | null |
| 10 | 1 | 2 | 5 | 1 | 1.3.4.1 | null | 04/10/12 | null |
| 11 | 1 | 1 | 6 | 1 | 1.3.4.1.1 | null | 04/10/12 | null |
| 12 | 1 | 1 | 6 | 3 | 1.3.4.1.1 | John | 04/10/12 | null |
| 13 | 1 | 3 | 5 | 1 | 1.3.5 | null | 04/10/8 | 04/11/8 |
| 14 | 1 | 1 | 6 | 1 | 1.3.5.1 | null | 04/10/8 | 04/11/8 |
| 15 | 1 | 1 | 6 | 3 | 1.3.5.1 | Kelly | 04/10/8 | 04/11/8 |
| 16 | 1 | 2 | 3 | 1 | 1.5 | null | 04/10/8 | null |

(d)

**Certification**

| certifi _ID | time_ center | certifi_path | certified _time | signature |
|---|---|---|---|---|
| 1 | Nagoya center | /company/department/ person[/name=`Mori`] | 04/10/9 | 2947523... |
| 2 | Nagoya center | /company/department[ @name=`market`] | 04/10/10 | 34503245.. |

( e )

図４: XML文書を**RDB**に格納

node_ID: ノードを識別するための ID である.

docID: XML 文書を識別するための ID である.

index: 兄弟間の相対位置を格納する.

label:ノードに付与した ORDERPATH ラベルを格納する.

value: 要素内の文字列を格納する. テキストノード以外の場合は NULL にする.

insertion_time: ノードが XML 文書に挿入された時刻を格納する.

deletion_time:ノードが XML 文書から削除された時刻を格納する.

certifi_ID:認証情報を識別するための ID である.

time_center:時刻認証センターに関する情報を格納する.

certifi_path: 認証範囲を指定した XPath を格納する, 実際には認証された部分文書の最上ノードを指定する.

certified_time: 認証を行った時刻を格納する.

signature:認証された文書に関する基本情報やハッシュ値や認証センターの署名などの認証情報を格納する.

図４で実際の XML 文書及び認証情報を XML データベースへ格納し, 大円に含まれる部分文書が認証された時刻である 04/10/10 以後で John という人の情報が文書に挿入された例を表している. ここでは Mori という人の情報が二回認証された,図 4.(a)で小円に含まれた部分文書の認証情報を certifi_ID である 1 で,大円に含まれた部分文書の認証情報を certifi_ID である２で保存する.異なる認証を属性 certifi_path と certified_time で区別する. 従って,様々なノードに対して別に認証ができるので,証券オンライン取引の値段のような電子文書の一部が頻繁に更新される場合でも動的な時刻認証ができる.

4.1 XML 更新の処理

認証された XML 文書が更新されたときに以後検証できるように元の XML 文書の情報を保存しなければならない.

| nodeID | docID | index | pathID | type | label | value | Insertion_time | Deletion_time |
|---|---|---|---|---|---|---|---|---|
| .... | | | | | | | | |
| 7 | 1 | 1 | 5 | 1 | 1.3.3 | null | 04/10/8 | null |
| 8 | 1 | 1 | 6 | 1 | 1.3.3.1 | null | 04/10/8 | null |
| 9 | 1 | 1 | 6 | 3 | 1.3.3.1 | Mori | 04/10/8 | 04/11/8 |
| 10 | 1 | 2 | 5 | 1 | 1.3.4.1 | null | 04/10/12 | null |
| 11 | 1 | 1 | 6 | 1 | 1.3.4.1.1 | null | 04/10/12 | null |
| 12 | 1 | 1 | 6 | 3 | 1.3.4.1.1 | John | 04/10/12 | null |
| 13 | 1 | 3 | 5 | 1 | 1.3.5 | null | 04/10/8 | 04/11/8 |
| 14 | 1 | 1 | 6 | 1 | 1.3.5.1 | null | 04/10/8 | 04/11/8 |
| 15 | 1 | 1 | 6 | 3 | 1.3.5.1 | Kelly | 04/10/8 | 04/11/8 |
| 16 | 1 | 2 | 3 | 3 | 1.5 | null | 04/10/8 | null |
| 17 | 1 | 1 | 6 | 3 | 1.3.3.1 | Tanaka | 04/11/8 | null |

図５: ノードの更新の処理

- ノードの挿入

挿入したいノードを Node 表に挿入すればよい．図 4.(d)で,挿入された John の情報は node_ID が 10 から 12 までの組に保存する．

- ノードの削除

Node 表において,削除したいノード及び子孫ノードの組の label は変更せずに削除された時刻を属性 deletion_time に保存する．図 4.(d )の node_ID が 13 から 15 までの組は 04/11/8 に削除されたノードを表す．

- テキストノードの更新

元のノードを削除し,新たなノード挿入する．削除の処理は上と一緒にすればよい．たとえば,Mori という人の名前が Tanaka に変換される場合は,ORDPATH である 1.3.3.1 の Mori ノードに関する組の属性 deletion_time に削除時刻である 04/11/08 を保存し,新たなノード Tanaka を Node 表に挿入し，図 4.(d)の Node 表が図５のようになる．

## 5．データベースから元の XML 文書の生成

何回か更新された XML 文書に対して検証を行いたいときには，元の XML 文書の生成をしなければならない．従って,データベースから全ての認証されたノードを取得し,元の XML 文書を生成する．

```
SELECT N3.value, N3.type, N3.label
FROM path P1, node N1, path P2, node N2,node N3, certification C
WHERE  P1.pathExp= `/company/department/person`
   AND  P1.pathID=N1.pathID
   AND  P2.pathExp= `/company/department/person/name`
   AND  P2.pathID=N2.pathID
   AND N2.value=`Mori`
   AND N1.label=substring(N2.label,length(N1.label)+1)
   AND N1.label=substring(N3.label,length(N1.label))
   AND N3.insertion_time<C.certified_time
   AND ((N3.deletion_time>C.certified_time) OR(N3.deletion_time  IS NULL))
```

図6: データベースからの認証範囲ノードの選択

- 認証されたノードの取得

Node 表から認証範囲を指定した XPath を満足するノードを取り出す.一度認証され,その後 XML 文書が更新された場合は時刻認証を行った時刻である certified_time を使用する.動的な更新される場合は,ノードが挿入されてから,時刻認証を受け,その後削除されたノードは insertion_time <certified_time <deletion_time という条件で区別できる．たとえば，Certification 表の certifi_ID が１の認証範囲のノードを図５の Node 表から取得したいとき,まず certifi_path に格納された XPath 式の結果を取得し,次はその子孫ノードを取得する(図６の SQL 文).SQL 文の出力としては属性 value，type 及び label が必要である．XML 文書の要素と属性ノードをデータベースへ格納する領域を節約するために，ノード名を属性 value に格納していないので，Path 表からノード名を算出する.あるノードに関する pathExp で一番後の`/`からの文字を取り出せばよい,ただ属性の場合は`@`を削除する.得られたノード名を value に保存する．

XML 文書は内容が同一であっても XML 構文に空白文字や改行などが任意に加わると表現が異なるため,ハッシュ関数を計算した結果であるハッシュ値が変わってしまう．従って,ハッシュ値を計算する前にその XML 文書の正規化（Canonicalization）を行う必要がある．本論文では,W3C のデジタル署名ワークグループ（WG）と IETF で標

準化された XML 文書の正規化処理[8]（以下,正規化処理標準と呼ぶ）を用いる．正規化処理標準によるハッシュ値はあるノードの属性ノードの順序とは関係がないが,要素ノードの順序が変わると,ハッシュ値が変化する．ORDPATH 手法では,認証された XML 文書が何回更新されても最初のラベルが変わらないので,親子関係と兄弟ノードの相対順序が保持できる．

5.1　ORDPATH ラベルによる文書の生成アルゴリズム

ノードをラベルの辞書順でソートし．ソートされたノードから生成された XML 木に基づき,XML 文書を生成する．

- ノードのソート

ノードをラベルの左からドットごとに比較し,同じならば次のドットまでの比較を行う．空値は他の数値より小さいとする.たとえば 1< 1.-1< 1.1 となる．ラベルが全く同じの場合はテキストノードが他のノードより大きいとする．ソートされたノードの順序は XML 木に対する深さ優先探索結果の順序と同じである．ソートされたノードをノードリスト（図7）に保存する．

Certification 表の certifi_ID が 2 の認証範囲のノードをノードに対してソートされたノードリストを図7に示す．一般的なソート手法（たとえばマージソート）で約 T*(logT)回の比較を行うが,ここでは二つのラベルの大小の比較が一回でできなく,レベルの平均ドット段数ぐらい回数が必要である.従ってソートのコストが約{T*(logT)＊（レベルの平均ドット段数)}となる．

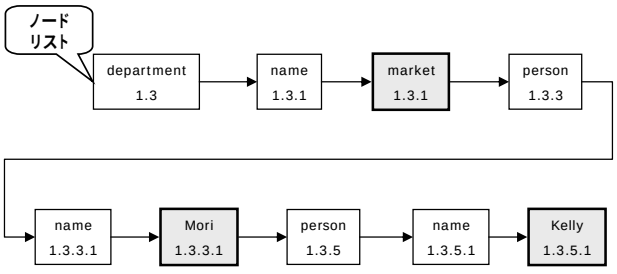



図7：ノードリスト

- XML 木の構築

ノードリストから XML 木を構築するのに,以下のデータ構造を定義する．

1）　木： XML 木を格納する．そのノードはデータエリアとポインタエリアから構成する．データエリアにはデータベースから取得した組の value, node_type, label 属性を格納する．ポインタエリアには子供,次の兄弟及び親を指すポインタを格納する．

2）　ノードリスト：ソートされたノードを格納する．

3）　ノードリストにおいて処理されているノードを指すポインタをリストポインタと呼び,木にいて処理されているノードを指すポインタを木ポインタと呼び，リストポインタまたは木ポインタで指されているノードを current ノードと呼ぶこととする．

- 木の生成

ステップ１：木ポインタを NULL にし,リストポインタは一番前のリストノードを指す．

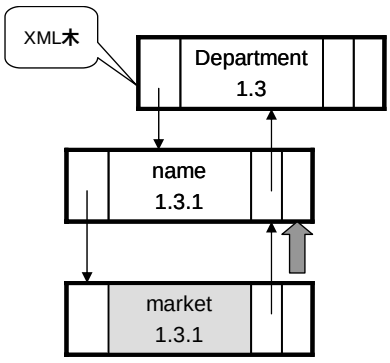
ステップ２：ノードリストの前から順番にリストノードを取り,リストポインタが次のリストノードを指す．ノードリストから取ったノードを木ポインタが指している木ノードの子ノードとして,木に挿入する（木ポインタが NULL であれば,ルートノードとする）．type によるノードのタイプを判断し,テキストノード以外の場合は木ポインタが挿入されたノードを指し,ステップ２を繰り返す．

テキストノードの場合は木ポインタの指し先が変わらずに,次のリストノードを取り,指された木ノードの兄弟かどうか判断する．木において current ノードの親ノードのラベルが比較対象の二つノードのラベルに含まれるかどうかにより,兄弟の判断を行う．兄弟であれば,木において current ノードの兄弟として木に挿入する．木ポインターが新たなノードを指す．兄弟でなければ,木ポインタが current ノードの親ノードを指し,また兄弟を判断する．

たとえば，図 8.(a)のようにノードリストから department や name ノードを一つずつ取り,XML 木に挿入する．続いて market ノードがテキストノードなので，XML 木に挿入してから木ポインタ（太矢印）の指し先が変わらずに,ノードリストから person ノードを取ってくる(図 8.(b))．それはテキストノードではないので,木ポインタに指されている name ノードの兄弟の判断を行う．ラ

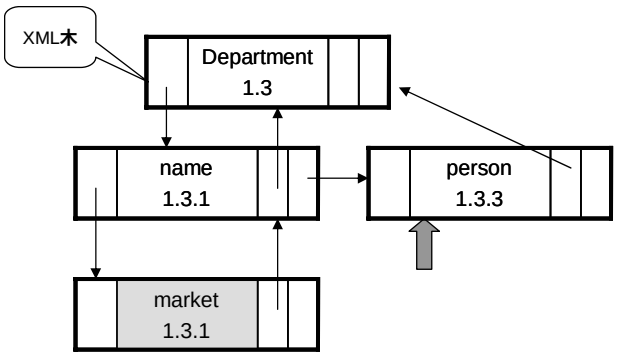
ベル1.3.1と1.3.3 はいずれもその中にnameノードの親ノードのラベルである 1.3 を含むまめ，二つのノードが兄弟であると判断し,personノードをnameノードの兄弟としてXML木に挿入する.

- XML木をもとにXML文書の生成




（a）

.




（b）

図8：XML木の構築

構築された XML 木に対して深さ優先探索すれば,元のXML 文書が生成できる．紙幅の都合上,具体的なプログラムは省略する.

## 6．まとめ

本論文の手法で認証された文書が何回更新されても検証が可能である.だたし,大量更新の場合は,全ての更新情報をデータベースに保存するのは非常に時間がかかる.また株価のような頻繁に更新されるデータの場合は,数多くの更新情報が存在するため,検索の効率が低下する可能性がある.従って,認証されていない更新はデータベースに保存しないなどの対策が必要となる.

## 7．今後の課題

大量のXMLデータを格納するデータベースから元のXML文書を生成するのに非常に時間がかかることが予想されるので,現在のアルゴリズムの性能を評価すると同時に他のXML文書管理手法も考察する予定である.

## 参考文献

[1] www.e-timing.ne.jp

[2] 最新キーワード解説_時刻認証サービス http://www.keyman.or.jp/search/name/name_file/n_30000112_1.html

[3] XML Signature　http://www.w3.org/Signature.

[4] 民間事業者等が行う書面の保存等における情報通信の技術の利用に関する法律案　http://www.cas.go.jp/jp/houan/index.html

[5]Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, Shunsuke Uemura: XRel: a path-based approach to storage and retrieval of XML documents using relational databases. ACM Trans. Internet Techn. 1(1): 110-141 (2001).

[6] Shankar Pal, Istvan Cseri, Oliver Seeliger, Gideon Schaller, Leo Giakoumakis, Vasili Zolotov Indexing XML Data Stored in a Relational VLDB　2004

[7] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller. ORDPATHs: Insert-Friendly XML Node Labels. SIGMOD2004.

[8] Canonical XML Version 1.0　http://www.w3.org/TR/xml-c14n

[9] Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/REC-xml.