差分 XSLT スタイルシート生成法の提案と実装

加藤 剛志　上野 英俊　石川 憲洋　高橋 修

（株）NTT ドコモ マルチメディア研究所 〒239-8536 横須賀市光の丘 3-5 NTT DoCoMo R&D センター

E-mail: {t_kato, ueno, ishikawa, osamu}@mml.yrp.nttdocomo.co.jp

あらまし 我々は、更新前後の XML 文書の差分を抽出し、その差分情報を XSLT スタイルシートにより表現する方式を提案する。本論文では、DOM ツリーを利用した差分抽出アルゴリズムと、XML の差分表現方式、及び差分情報を付加する差分 XSLT スタイルシートの生成方法について述べる。更新前後の XML 文書の差分情報から、差分 XSLT スタイルシートを生成し、更新前 XML 文書にそのスタイルシートを適用することにより、更新後の XML 文書を得ることができる。これにより、更新後の XML 文書そのものを送信するよりも、少ないデータ転送量での XML 文書の更新が可能となる。また、提案方式に基づいて実装した差分 XSLT スタイルシート生成ソフトウェアについて述べ、基本評価によりその有効性を示す。

キーワード XML(eXtensible Markup Language)，DOM(Document Object Model)，XPath，XSLT(eXtensible Stylesheet Language Transformation)，XML 差分符号化

Proposal and Implementation of Differential XSLT Stylesheet Generation Method

Takeshi KATO　Hidetoshi UENO　Norihiro ISHIKAWA　and　Osamu TAKAHASHI

Multimedia Laboratories NTT DoCoMo, Inc.　3-5 Hikarino-oka, Yokosuka, Kanagawa, 239-8536 Japan

E-mail: {t_kato, ueno, ishikawa, osamu}@mml.yrp.nttdocomo.co.jp

Abstract We propose the method to extract difference between the original XML document and its revision and to depict this differential data in XSLT stylesheets. This paper describes delta-extraction algorithm using DOM trees, XML delta-expression method and differential XSLT stylesheet file creation method. Consequently, it is possible to obtain the revised XML document by supplying the XSLT stylesheet with the differential data to the original XML document. Comparing with sending whole revised XML document, the original XML document can be updated by sending less information, the differential data. We also introduce prototype software implemented based on proposed method and evaluation result that shows the effectiveness of our method.

Keyword XML(eXtensible Markup Language)，DOM(Document Object Model)，XPath，XSLT(eXtensible Stylesheet Language Transformation)，XML delta encoding

1. はじめに

World Wide Web (WWW)は、SGML のアプリケーションであるマークアップ言語、HTML（Hyper Text Markup Language）の普及により、急速に発展してきた。これにより、特定の計算機システムに依存や制約のされない、インターネットを使った世界的な文書閲覧システムが実現された。しかし、HTML は文書を表示することが主な目的であることから、拡張性がなく、また終了タグの省略、属性の真偽値の省略なども許されている。さらに、多くの HTML ブラウザは、不正な HTML 文書であっても、ユーザの利便性を考慮して、ある程度の補正を行いレンダリングする設計となっている。そのため、インターネット上の多くの HTML 文書が妥当性検証されていないのが現状である。そこで W3C は、インターネット上で効率的に構造化文書を扱うマークアップ言語として SGML のサブセットである XML（eXtensible Markup Language）[1]を開発した。XML は、データをネットワーク経由で送受信するための汎用マークアップ言語であり、ユーザが独自のタグを拡張できるメタ言語の一種である。マークアップ言語とは、文書の内容である文字列や文字コード以外のメタ情報（文字列の制御情報、意味情報、構造情報など）を、別に定義する予約語を利用して文書中に記述していく方式の言語であり、様々な計算機処理の対象とすることを目的として開発されてきた。XML ベースのマークアップ言語としては、HTML を XML の仕様に準拠するように再定義した XHTML や、通信プロトコルの SOAP、Web 上でベクター画像の表現を行なう

SVG など、様々な応用が考えられている。

一方、携帯電話などのモバイル環境向けアプリケーションでも、Compact HTML や WML などの独自の HTML 仕様が開発されてきた。こうした中で、WAP (Wireless Application Protocol) フォーラム（現 OMA (Open Mobile Alliance)）では、記述言語の標準仕様として XHTML Basic[2]を採用するなど、XML 技術の移動体通信への応用が進められている。XHTML Basic は、携帯電話、PDA、テレビ、情報家電などの多様な端末での利用を想定し、それらで共通して使えるマークアップ言語の基盤となるものであり、モジュール化された XHTML から必要最小限のモジュール群だけを抽出し構成されている。

しかし、XML はこれまでの HTML やテキストファイル、バイナリファイルと比較してコンテンツサイズが大きくなる場合が多い。例えば XHTML では、HTML と比較して、XML 宣言や名前空間宣言、要素の終了を明示的に記す必要があるなど、一般的にそのファイルサイズが大きくなる傾向にある。また、移動通信網においても第三世代(3G)携帯電話方式の登場によりネットワークの高速化が行われてきたが、利用者の急増やコンテンツサイズの増加などに対処するため、限られた無線資源の利用効率の向上、Web サービス品質の向上は非常に重要な課題である。そこで筆者らは、携帯電話のプル型やプッシュ型配信において、XML コンテンツの差分情報を生成しその差分情報のみを送信することで転送データサイズの削減をする方式の検討を行ってきた[15][16]。

ネットワークを流れる転送データサイズを減らすための工夫としては様々な方式が存在する。例えば HTTP(Hypertext Transfer Protocol)では、クライアントのキャッシュを有効活用するため、キャッシュが無効の場合のみデータを転送する仕組みを備える。さらに、HTTP デルタエンコーディング[3]では、旧コンテンツからの差分情報のみを転送するための機能が提供されている。XML 文書においても、その差分情報を抽出し、HTTP デルタエンコーディング等を用いて差分情報のみを転送すれば、ネットワークを流れるデータ量の削減を図ることが可能である。

本論文では、XML のツリー構造に注目して任意の 2 つの XML 文書から差分情報を抽出し、その差分情報を転送する方式を提案し、その性能評価について述べる。2 節では、モバイル向け XML の差分情報生成法の課題と提案する解決法について述べる。3 節では提案方式の詳細について説明し、4 節で試作ソフトウェアの紹介とその性能評価について述べる。5 節では、提案方式に関する考察を述べ、最後に 6 節でまとめを行う。

2. モバイル向け XML 差分生成の課題

任意の 2 つの XML 文書からその差分情報を生成するには、1)XML 差分情報の抽出方法、2)XML 差分情報の表現フォーマット、3)差分情報による変更後 XML 文書の生成方法、という 3 つの課題がある。本研究では、これらの課題の解決法として、XSLT (eXtensible Stylesheet Language Transformations)[4]を用いることを試みた。XSLT は、XML 対応のスタイルシート言語である XSL (XML Stylesheet Language)から派生した言語であり、XSL は XML 文書を構造変換するための言語と、テキストをフォーマット化するための言語から構成されている。このうち構造変換用の言語の部分を独立させたものが XSLT であり、XHTML 文書をテキスト形式に変換することや、XML 文書の構成要素の並び替えを行うことが可能である。この XSLT を応用すると、XML の差分情報を XSLT による変換ルールとして記述することが可能となり、それを変換前 XML 文書に適用することで、自動的に変換後 XML 文書が生成できる。XSLT の採用により上記 3)の課題が解決できるため、本研究では課題 1)と 2)の解決法を検討する。

既存の差分生成法[8][9]と比較して、提案する XSLT を用いた差分生成の利点としては、XML として展開されているデータへの適用が容易、また既存の XSLT エンジンの利用が可能な点が上げられる。一方、XML のナビゲーションツリーを、メモリ上に展開する必要があるなど、携帯端末にとっては処理が重くなるという問題点も存在する。しかし、近年、TinyTree[10]や DTM(Document Table Model)[11]など、携帯端末等に適した超軽量 XML エンジンも開発が進められており、特にその場合、XML 文書は木構造や配列でメモリに展開されているため、XML 処理系で差分を処理した方が効率が良い。また提案方式では、携帯端末側には XSLT 処理のモジュールだけあればよく、[9]の DUL(Delta Update Language)ように、特別な差分言語処理モジュールを追加実装しなくて良いという利点がある。

3. 差分 XSLT スタイルシート生成法

任意の 2 つの XML 文書の差分情報を XSLT の文書変換ルールとして記述した XSLT スタイルシートについて、本研究では差分 XSLT スタイルシートと呼ぶ。差分 XSLT スタイルシートの生成法は、図 1 のように XML の差分を検出する 1)差分検出プロセス、差分情報を変換ルールとして整理する 2)差分抽出プロセス、差分 XSLT スタイルシートを生成する 3)差分表現プロセスからなる。これらのプロセスに分けることで、1)のプロセスは、XSLT に依存しない汎用的な XML 差分抽出アルゴリズムとして検討が可能となる。

また、差分生成方式としては、編集する XML 文書に対してユーザが行った編集操作の履歴を元に、差分

情報を抽出する方法が考えられる。しかし、この方式では XML 文書そのものと、その編集履歴を持つ必要があるため、利便性に欠ける。本研究では、任意に与えられた 2 つの XML 文書の差分生成法を提案する。これにより、編集履歴を保持する必要の無い汎用的な差分検出が可能となる。

以下、これらのプロセスについて詳細を述べる。

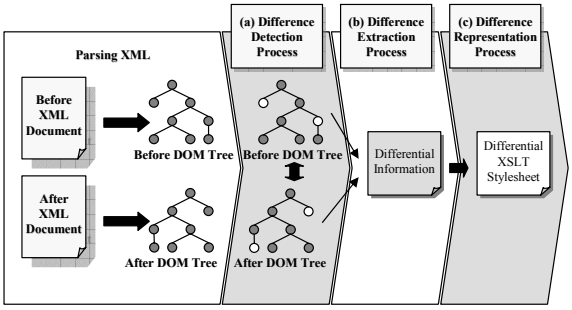

図 1 差分 XSLT 生成手順

3.1. 差分検出プロセス

提案方式では、XML データ構造をツリー（木）構造で処理することが可能な DOM (Document Object Model)[5]を用いて処理を行う。任意の 2 つの XML 文書の DOM ツリーをルート（根）から走査して比較することで、差分情報の検出が可能である。このプロセスでは、2 つの DOM ツリーの差分情報として、一致ノード(変換前後で共通に存在する要素)、削除ノード(変換後に存在しない要素)、追加ノード（変換後に出現する要素）の検出を行う。任意の 2 つのツリーから差分情報を抽出する方法としては簡易的な差分情報の検出法（既存方式であり 3.1.1 節で説明する）と最適化された差分を検出する方法（提案方式であり 3.1.2 節で説明する）が存在する。

3.1.1. 簡易差分検出法（既存方式）

簡易差分検出法は、2 つの任意のツリーのルートから、変更前ツリーの各ノードを基準に、変更後のツリーの対応ノードを順次比較し、変更前ツリーを走査し比較を進める過程で、ノードの不一致があったノード以下のサブツリー（部分木）を全て差分とみなす方式である。この場合、ブランチ（分岐枝）へのノードの挿入や削除を正確には検出できない。図 2 の例で説明すると、ノード"C"の上位にノード"1"が挿入されているが、簡易差分検出法ではノード"1"以下は全て差分とみなされる。文献[12][13]は、任意の XML 文書の差分生成について説明しているが、いずれも簡易差分検出法と同様に最適な差分情報の検出が不可能である。また、文献[14]では、2 つの有向順序木の頂点数を n_a、n_b とした場合の最大共通グラフ問題（連結グラフのうち、辺の数が最大であるものを求める）が $O(n_a n_b)$時間で計算できることを示しているが、その場合も同様に、変更や削除されたノード以下のサブツリーの正確な差分検出は不可能である。

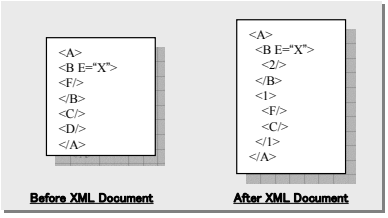

図 2 XML 文書の例

3.1.2. 最適化差分検出法（提案方式）

最適化差分検出法は、2 つの XML の差分情報を生成するために、初めに一致ノード・不一致ノードの検出を行う。検出アルゴリズムとして、DOM ツリー構造を変化させながら、一致ノードの最も多い形状を見つけて、残りを差分とする方法を提案する。アルゴリズムの前提条件は以下の通りである。

i) 実際上のルートノードの上位に仮想ノードを設定し、それをルートノードとする。ルートノードは必ず一致するとして扱う。

ii) 変更前後の DOM ツリーにおいて、検出した一致ノードが最大となった状態を最適とする。

iii) 一致とは変更前後の各ノードにおいて、要素名や属性名が一致することとする。

また、ノードとして属性、要素、テキストを同列で扱う場合、属性ノードと要素ノードは比較する必要がない。そこで、比較の効率化のため各ノード種別で比較を行う。以下において、図 2 を用いて最適化差分検出の方式について説明する。

1) DOM ツリーの要素比較を行い、一致する可能性のある最大ノードの組合せを調べる

まず、DOM ツリーの要素比較を行い、一致する可能性があるノードのみで組合せの作成を行う。これにより、組合せの比較回数を減少させる。(図 3(a))

2) ノードの組合せの作成及びその比較を繰り返す

順次、組合せを構成するノード数が多いものから比較を行う。また、一致ノードが最大となる組合せが複数存在する場合は変更後 DOM ツリーにおいて、ルートに近い方、ツリーの右側に近い方を一致ノードとする。ルートに近い部分を変更する場合、ツリーのリーフに近い部分を変更する場合よりも、XML 文書がツリー構成である性質上、により大きな影響を与えると考えられる。(図 3(b))

3) 得られた一致ノードから、出現ノード・消失ノードを洗い出す

一致ノードが最大数存在する DOM ツリーを構成するノードが最終的な一致ノードとなる。また、それを変更前、変更後 DOM ツリーに当てはめたとき、一致ノードに合わないノードが不一致ノードとなる。次に、一致ノード・出現ノード・消失ノードの区別を行う。

出現ノードは､変更後に存在する不一致ノードである。消失ノードは変更前に存在する不一致ノードである。それぞれのノードの情報を、図 3(c)の表にまとめる。

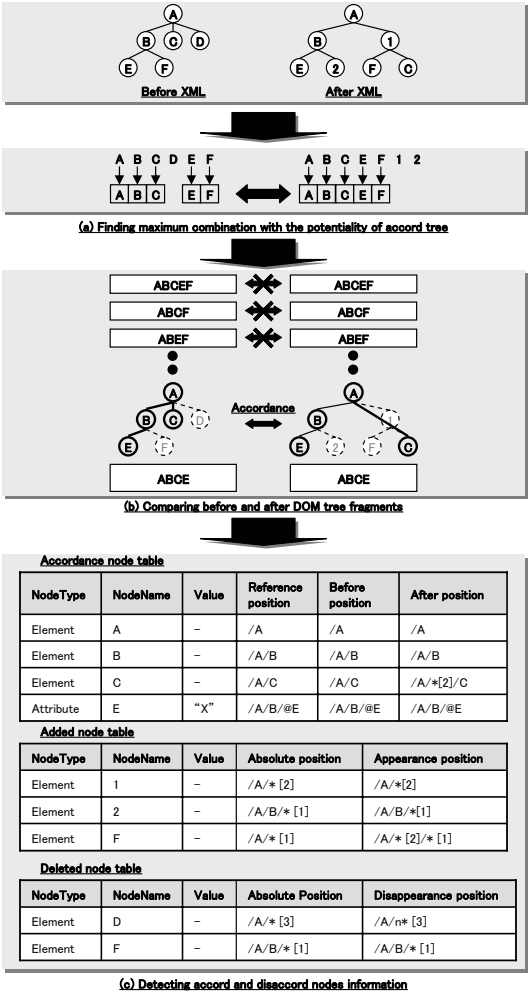

図 3 差分検出プロセス

一致ノードテーブルにおいて、一致ノードテーブルは一致 DOM ツリー上での位置を示す｡変更前位置は、変更前 DOM ツリーでの位置を示し、変更後位置は同様に変更後のノードの位置を示す。出現ノードテーブルにおいて、出現位置は変更後 DOM ツリーでの位置を表す。消失ノードテーブルでも同様に、消失位置は変更前 DOM ツリーでの該当ノードの位置を示す。それぞれ、一致ノード以外の追加や変更された要素ノードは**[n]*で表し、属性は@を付けて表している。*n* は DOM ツリーでの一番兄となるノードを基準とした順番である｡以上により､最適な差分情報が検出される。

3.2. 差分抽出プロセス

3.1 で述べたプロセスより得られた、一致ノード、出現ノード、消失ノードリストより、より正確な差分情報を抽出する。変更前のノードが消失する理由として、消去された場合、移動した場合、内容を変更された場合が考えられる。また、変更後に新たに出現したノードは、移動してきたか、変更されたか、追加されたかのどれかに当てはまる。提案する方式では、出現ノード及び消失ノードを、さらに以下に示したノードに分類する。

i) 変更ノード

出現ノードテーブルと消失ノードテーブルを比較して、同じ絶対位置と同じノードタイプであるノードを、変更ノードとみなす。図 3(c)において、出現ノードテーブルのノード"2"と消失ノードテーブルのノード"F"が変更ノードの対象となる。

ii) 追加ノード

出現ノードテーブルにおいて、変更ノードとみなされたノード以外が追加ノードとなる｡図 3(c)において、出現ノードテーブルのノード"1"とノード"F"が追加ノードである。

iii) 削除ノード

消失ノードにおいて、変更ノードとみなされたノード以外が削除ノードとなる。図 3(c)において、消失ノードテーブルのノード"D"が削除ノードである。

最終的に、差分情報は、表 1 のように表される。表 1 は、図 2 の例を用いた場合の差分情報テーブルの例である。

表 1 差分情報テーブル

Reference Position	NodeType	Classification of difference	Additional Data		
			NodeType	NodeName	NodeValue
/A/B/*[1]	Element	Change	Element	2	–
/A/*[2]	Element	Addition	Element	1	–
/A/*[2]/*[1]	Element	Addition	Element	F	–
/A/*[3]	Element	Deletion	–	–	–

3.3. 差分表現プロセス

このプロセスでは、前項の差分抽出プロセスで得た差分情報を､XSLT テンプレートへマッピングを行う。さらに、それらを用いて差分 XSLT スタイルシートを生成する。また、DOM と XSLT が持つデータモデルには、いくつかの相違点がある[7]。これは、XSLT が利用する XPath[6]は DOM と異なり、CDATA-section や ReferenceEntity がテキストノードとして扱われたり、属性ノードは属する要素ノードを親とするが、当該ノードの子ではない､等の相違点に起因する｡そのため、このプロセスではその相違点に関しても補正を行う必要がある。以下において、提案する差分表現方式の仕組みについて述べる。差分表現プロセスは XSLT テンプレートルール作成のための 1)写像位置の決定、2)同一の写像情報を利用する差分ノードの決定、3)XSLT テンプレートへの適用、最終的に 4)差分 XSLT スタイルシートの生成、という 4 つの手順となる。

1) XSLT テンプレートマッチのための写像位置決定

XSLT による XML データの構造変換の記述は、以下の 2 つを指定することである。

i)　XML 文書のどの位置を変換するのか（パターン（写像位置）の決定）

ii)　変換先の XML 文書にどのような処理を行うのか（差分情報の表現）

この指定を行うことを、XSLT ではテンプレートルールで表現し、*xsl:template* 要素で表す。XSLT テンプレートルールにおいて、パターン（写像位置）は変更前 DOM ツリーでのノード位置の XPath 表現となる。

XSLT で差分表現を実現するには、パターンとなりうる写像位置となるノードを指定する必要がある。つまり、例えば追加ノードであれば、変更後 XML 内にある、新たに出現したノードの位置情報を「あるノードの絶対位置」からの「相対位置」という 2 つの情報に分離し、「あるノードの絶対位置」をテンプレートルールのパターンに XPath として表現する。（図 4）

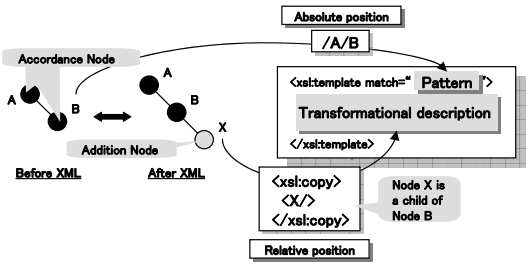

図 4 XSLT テンプレートルール

提案方式では、ノードのテンプレートのパターン（写像位置ノード）決定は、変更後 DOM ツリーの上方向から下方向（親から子へ）、且つ左方向から右方向（兄から弟へ）の順で行う事とする。XML 文書は完全なツリー構造のため、親から子に向かって分岐していく。パターンとなりうるノードは、差分ノードよりもルートに近いノードであるため、深さ優先でパターンの探索を行う。パターンは「差分情報が変更である要素ノード」または「一致する要素ノード」となる。

次に図 5 を参照して、テンプレートルールのパターン(写像位置ノード)決定方法について詳細を述べる。一致ノードと変更ノードは、自身の位置が写像位置となる。追加ノードの場合、以下に記述する順で処理を行い、写像位置を決定する。

i)　子ノード方向

図 5(a)のように、当該ノードの子孫ノード方向に対してパターンを検出する。検出した最も近いノードの位置をパターンとする。ノードが検出できない場合、一意に定まらない場合、ツリーが分岐している場合は次の処理へ移行する。

ii)　兄ノード方向

図 5(b)のように、当該ノードの兄ノード方向に対してパターンを検出する。検出した最も近い兄ノードの位置をパターンとする。ノードが検出できない場合は次の処理へ移行する。

iii)　弟ノード方向

図 5(c)のように、当該ノードの弟ノード方向に対してパターンを検出する。検出した最も近い弟ノードの位置をパターンとする。ノードが検出できない場合は次の処理へ移行する。

iv)　親ノード方向

図 5(d)のように、当該ノードの先祖ノード方向に対してパターンを検出する。検出した最も近い親ノードの位置をパターンとする。ノードが検出できない場合は、上位ノードで決定されたパターンを当該ノードのパターンとする。

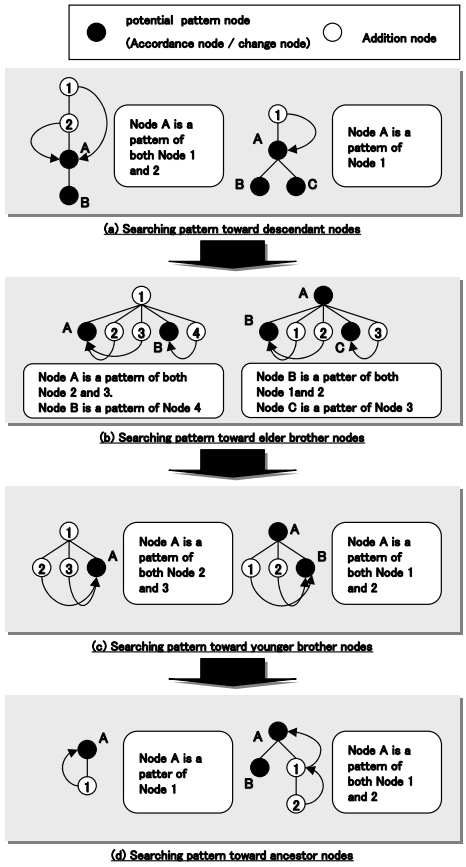

図 5 パターン探索

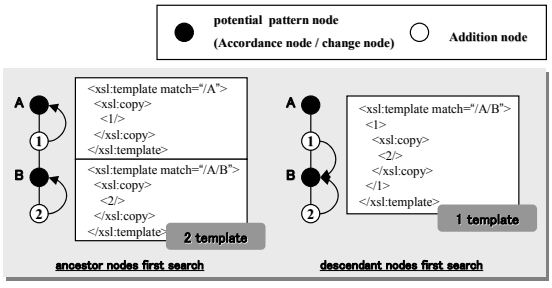

図 6 祖先優先探索と子孫優先探索

何故、子ノード方向を初めに処理を行い、親ノード方向が最後であるかは、図 6 に示した例のように、親ノード方向にパターンを生成すると、テンプレート数が大きくなり、最終的な XSLT スタイルシートが大き

くなってしまうからである。なぜなら、XML はツリー構造であるため、子要素の方が、より多くのノードと接する確率が高くなり、より多くの不一致ノードのパターンとして利用できるからである。

次に削除ノードのパターン探索を説明する。削除ノードの場合、基本的には親ノード方向にパターンを探す。図 7 のように、子孫ノードが全て削除ノードであれば、一番上位のノードをパターンとして、全てのノードが削除できる。そのため、子孫ノードに削除ノード以外を含まない先祖ノードを探して、それをパターンとする。ノードが検出できない場合は、自身の位置をパターンとする。また、属性ノードは、基本的に親ノードのみを持つため、追加ノードの(iv)と同様の処理を行う。

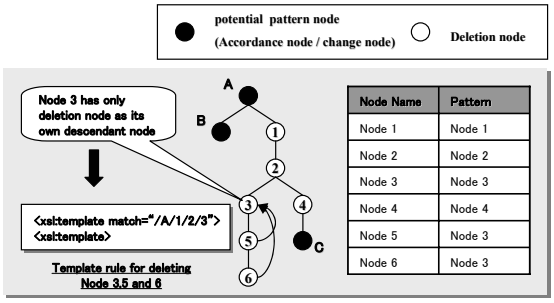

図 7 削除ノードのパターン探索

Pattern of template rule

Absolute position	NodeType	Classification of difference	Transformationa
/A	Element	Accordance	No
/A/B	Element	Accordance	Yes
/A/C	Element	Accordance	Yes
/A/D	Element	Deletion	Yes
/A/E	Attribute	Accordance	No
/A/B/F	Element	Change	Yes

Pattern node table

NodeName	Classification of difference	Pattern
/A/1	Addition	/A/B
/A/B/2	Change	/A/B/F
/A/1/F	Addition	/A/C
/A/D	Deletion	/A/D

Transformational description of template rule

Absolute position	Relative position	NodeType	Classification of difference	Addition Data		
				NodeType	NodeNAme	NodeValue
/A/B	Parent	Element	Addition	Element	1	-
/A/C	Elder brother	Element	Addition	Element	F	-
/A/B/F	itself	Element	Change	Element	2	-
/A/D	itself	Element	Deletion	-	-	-

図 8 パターンで整理した差分情報

2) 写像位置を基準とした差分情報の生成

前項で説明したパターンにより、差分情報を図 8 のように整理する。XSLT テンプレートを生成する場合、同じパターンを持つ差分ノードは、同じテンプレートに記述できる。したがって、パターンとなっている一致ノード毎に、差分ノードを整理する。図 8 は図 2 で示したサンプル XML 文書の差分情報である。

3) テンプレートへのマッピング

パターンとなるノード毎の情報に従って、以下の手順で XSLT テンプレートへのマッピングを行う。

i) パターンとなりうるノードのテンプレート化（一致ノード、変更ノード、削除ノード）

同じパターンを持つノードをまとめて、1 つのテンプレートに記述していく。付録 A で示すようなテンプレートへマッピングを行う。

ii) 追加ノードのテンプレート化

同じパターンを持つノードをまとめて、1 つのテンプレートに記述していく。付録 B のように、パターンとなるノードに対して、親ノード、子ノード、兄ノード、弟ノードを記述し、各追加ノードは付録 C に示したように記述する。図 9 は図 2 で示したサンプル XML の差分情報の例を示す。

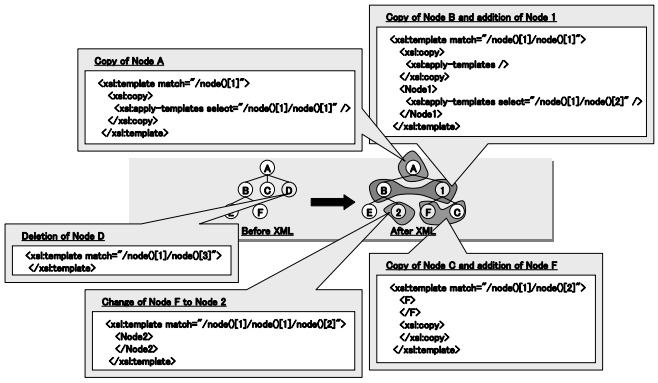

図 9 XSLT テンプレート例

4) 差分 XSLT スタイルシートの生成

最終的に、作成されたテンプレートを図 10 のように並べて、差分 XSLT スタイルシートを生成する。ヘッダ部分は、生成するスタイルシートの XML 宣言、及び各種設定値を表す。フッタ部分は、差分表現のテンプレートに現れなかった、その他一致ノードのコピーするテンプレートである。

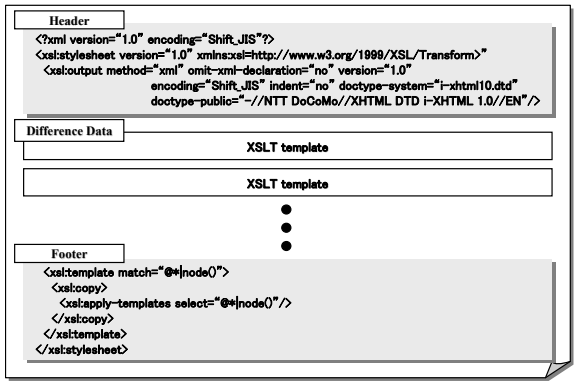

図 10 差分 XSLT スタイルシート

4. 実験

4.1. 試作ソフトウェア

本研究では、提案方式に基づいたソフトウェアを Java（J2SE 1.3.1）を用いて実装した。XML パーサーとして IBM XML Parser for Java を用い、作成した差分 XSLT スタイルシートの妥当性チェックのための XSLT エンジンとして Apache Xalan-Java 2.4.0 を用いた。本ソフトウェアは、任意の 2 つの XML 文書から差分 XSLT スタイルシートを生成し、さらに差分情報をグ

ラフィカルに表現する機能を持つ。このソフトウェアを用いて、提案方式の有効性を評価した。

4.2. 実験及び結果

まず、以下の株価情報ファイルを利用して、XML 文書の圧縮率と更新前後の XML 文書の差分割合の関係について測定を行った。図 11 に示す株価日足チャートは、毎分更新され、一番古い価格情報が消去され、新しい価格情報が追加される。このコンテンツを用いて、コンテンツ全体のサイズと差分割合を変化させたときに XML 文書の圧縮率を測定した。

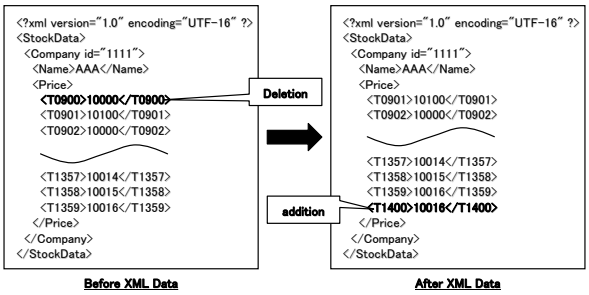

図 11 株価日足チャート

ここで、圧縮率は、更新前文書に対する差分 XSLT スタイルシートの大きさの割合を表す。(式(1))また、差分割合は、XML 文書を構成するノードに対する差分ノード数の割合を表す。(式(2))

$$\text{Compressibility ratio} = \frac{\text{Size of Generated XSLT Stylesheet}}{\text{Size of before XML data}} \times 100\,(\%) \quad (1)$$

$$\text{Difference ratio} = \frac{\text{Number of discordance nodes}}{\text{Number of all nodes in before XML data}} \times 100\,(\%) \quad (2)$$

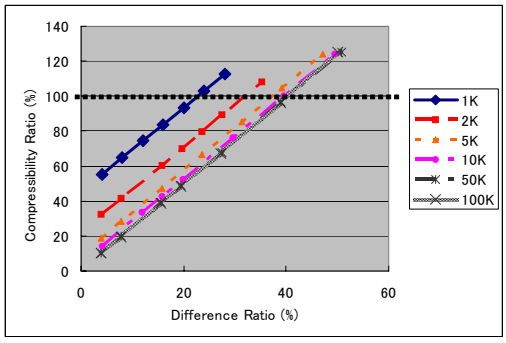

図 12 圧縮率と差分割合の関係

図 12 に結果を示す。XML 文書全体のサイズを 1Kbyte から 100Kbyte まで変化させ、差分割合を 4%から 50%まで変化させた。図に示したように、XML 文書サイズが大きくなると、圧縮率も高くなる。これは、コンテンツサイズが大きくなると、差分 XSLT スタイルシートのヘッダ部分とフッタ部分のオーバーヘッドが無視できるくらい小さくなっていくからである。しかし、1Kbyte 程度の小さな XML 文書でも、差分割合が 25%程度までは提案方式が有効に機能していることが分かる。提案方式はモバイル向けのような小さな XML 文書でも、変更箇所がある程度小さければ有効に機能することが分かる。

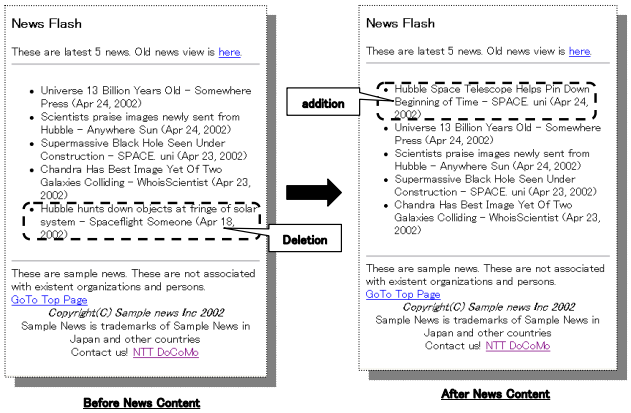

図 13 ニュースフラッシュコンテンツ

次に図 13 のような、ニュースフラッシュコンテンツを用いて、提案方式の性能評価を行った。図に示すコンテンツでは、更新の際に新しいニュース項目が追加され、一番古いニュース項目が削除される。これを用いて実際の差分 XSLT スタイルシートのサイズ(D-XSLT)、及びそれを gzip で圧縮したもの(D-XSLT+gzip)、さらに同じコンテンツ(但し、要素毎に改行したもの)を Diff で差分をとり、gzip で圧縮したもの(Diff+gzip)と差分割合の関係をそれぞれ測定した。また、変更後コンテンツを送信する場合との比較として、変更後コンテンツを gzip で圧縮したもの(HTML+gzip)も測定した。

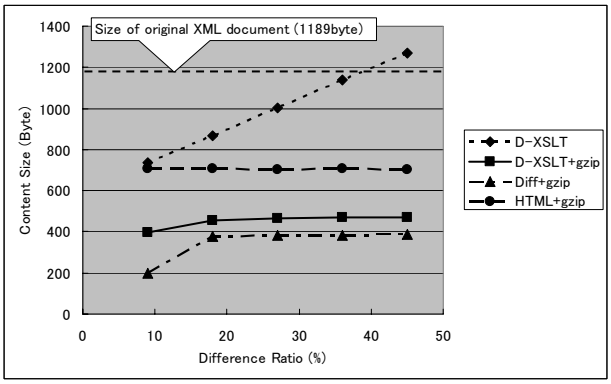

図 14 差分ファイルのサイズと差分割合の関係

図 14 は、その結果を示す。差分割合を 10%から 45%まで変化させたときの、差分 XSLT スタイルシートのサイズを示す。結果に示すように、差分 XSLT スタイルシートを gzip で圧縮したものは、変更後コンテンツを圧縮したものより、平均で 63%の大きさに圧縮されている。また、Diff を用いた差分ファイルを圧縮したものと比較すると、約 37%サイズ増となった。これにより、Diff には及ばないが、比較的、効率良く差分圧縮が実現されることが分かる。

以上、試作ソフトウェアの評価により、提案方式による差分 XSLT スタイルシートの生成、及び更新前

XML 文書と差分 XSLT スタイルシートからの更新後 XML 文書の生成が全て自動で行えることを確認し、さらにその圧縮効率も実用上問題がないことを確認した。

5. 考察

XSLT は必ずしも、差分情報を表現するのに最適ではない。そこで、差分 XSLT スタイルシート生成の過程において、特に有効であると思われる XSLT 機能として *xsl:copy-of* に関して新しい関数を提案する。*xsl:copy-of* は、サブツリーを指定するための XSLT 要素であるが、この場合、指定したノード以下全てのサブツリーが対象となってしまう。提案する *generation()*（図 15）は、この要素にサブツリーを基準ノードから n 世代まで指定するための関数である。

これにより、従来の XSLT よりも単純に、サブツリー単位のコピーや要素と属性をまとめたコピーの表現が可能となり、より効率的な差分情報を表現することができるようになる。

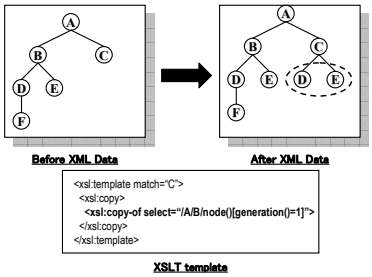

図 15 Generation 関数

6. まとめ

本研究では、任意の 2 つの XML 文書から、自動的に差分 XSLT スタイルシートを生成する方式について提案した。提案方式は、差分検出プロセス、差分抽出プロセス及び差分表現プロセスからなり、それぞれにおいてアルゴリズムの提案を行った。さらに、試作ソフトウェアについて述べ、その有効性を示した。さらに、XSLT において、差分情報を表現するために有効な新しい関数について提案した。

今後の研究では、さらに一般的な XML における提案方式の有効性の評価、新しい XSLT 関数の有効性評価、差分検出アルゴリズムの評価が課題となる。

TYPE	Mapping to XSLT (No continuing process to descendant of pattern)	Mapping to XSLT (Continuing process to descendant of pattern)
Accordance	No template	<xsl:template match="XPath of its own position"> <xsl:copy> <xsl:apply-templates/> </xsl:copy> </xsl:template>
Delete	<xsl:template match="XPath of its own position"> </xsl:template>	<xsl:template match="XPath of its own position"> <xsl:apply-templates/> </xsl:template>
Change	<xsl:template match="XPath of its own position"> <Data of Changed node/> </xsl:template>	<xsl:template match="XPath of its own position"> <Data of Changed node> <xsl:apply-templates/> </Data of Changed node> </xsl:template>

付録 A 一致ノード、削除ノード、変更ノードの XSLT テンプレートへのマッピング

TYPE	Mapping to XSLT (No continuing process to descendant of pattern)	Mapping to XSLT (Continuing process to descendant of pattern)
Addition node in Parent	<xsl:template match="XPath of a pattern node"> <Parent addition node> <xsl:copy/> </Parent addition node> </xsl:template>	<xsl:template match="XPath of its own position"> <Parent addition node> <xsl:copy> <xsl:apply-template> </xsl:copy> </Parent addition node> </xsl:template>
Addition node in Child	<xsl:template match="XPath of a pattern node"> <xsl:copy> <Child addition node/> </xsl:copy> </xsl:template>	Same as left template
Addition node in elder brother	<xsl:template match="XPath of a pattern node"> <Elder brother addition node/> <xsl:copy/> </xsl:template>	<xsl:template match="XPath of a pattern node"> <Elder brother addition node/> <xsl:copy> <xsl:apply-templates/> </xsl:copy> </xsl:template>
Addition node in younger brother	<xsl:template match="XPath of a pattern node"> <xsl:copy/> <Younger brother addition node/> </xsl:template>	<xsl:template match="XPath of a pattern node"> <xsl:copy> <xsl:apply-templates/> </xsl:copy> <Younger brother addition node/> </xsl:template>

付録 B ブランチへの追加ノードの XSLT テンプレートへのマッピング

TYPE	Mapping to XSLT (No continuing process to descendant of addition node)	Mapping to XSLT (Continuing process to descendant of addition node)
Addition node	<Addition node/>	<Addition node> <xsl:apply-template select="XPath of lower pattern"/> </Addition node>

付録 C ツリーの末端への追加ノードの XSLT テンプレートへのマッピング

文　献

[1] Tim Bray, et al., “Extensible Markup Language (XML) 1.0 (Second Edition),” W3C Recommendation, Oct 2000.

[2] Mark Baker, et al., “XHTML Basic,” W3C Recommendation, Dec 2000.

[3] Jeffrey C. Mogul, et al., “Delta Encoding in HTTP,” RFC3229, Jan 2002.

[4] James Clark, “XSL Transformations (XSLT) Version 1.0,” W3C Recommendation, Nov 1999.

[5] Arnaud Le Hors, et al., “Document Object Model (DOM) Level 2 Core Specification Version 1.0,” W3C Reccomendation, Nov 2000.

[6] James Clark, et al., “XML Path Language (XPath) Version 1.0,” W3C Reccomendation, Nov 1999.

[7] Ray Whitmer, “Document Object Model (DOM) Level 3 XPath Specification Version 1.0,” W3C Candidate Recommendation, March 2003.

[8] Diff, http://www.gnu.org/

[9] XMLDiff, http://diffxml.sourceforge.net/

[10] TinyTree, http://saxon.sourceforge.net/

[11] DTM, http://xml.apache.org/xalan-j/dtm.html

[12] Robin La Fontaine, “A Delta Format for XML: Identifying Changes in XML Files and Representing the Changes in XML,” XML Europe 2001, May 2001.

[13] F. P. Curbera, D. A. Epstein, “Fast Difference and Update of XML Documents”, XTech’99, March 1999.

[14] 増田澄男 他,“二つの木の最大共通部分グラフを求めるアルゴリズム,” 電子情報通信学会論文誌 (A), vol.J77-A, no.3, pp.460-470, March 1994.

[15] Norihiro Ishikawa, et al., “Automatic Generation of a Differential XSL Stylesheet From Two XML Documents,” WWW Conference 2002, May 2002

[16] 上野英俊 他, “XML コンテンツの差分生成法とプッシュ型配信への応用,”情報処理学会 MBL 研究会 第 20 回 研究報告会, March 2002.