A6-1 並行編集環境におけるXML文書の一貫性維持方式*

鳥井 修　　木村 哲郎　　瀬川 淳一
(株) 東芝 研究開発センター コンピュータ・ネットワークラボラトリー
{osamu.torii, tetsuro2.kimura, junichi.segawa}@toshiba.co.jp

サーバ上で一元管理しているXML文書を複数のユーザが各自のクライアント上で並行して編集するシステムで, XML文書の一貫性を維持しながら, ユーザの編集意図を豊富に, 柔軟に取り込むことは難しい. 本研究では, ユーザがXML文書に対して行った編集全体を二段階の手続きを用いて分割し, 分割した単位ごとにサーバに反映する手法を確立した. この手法は従来の一貫性維持方式と比較して, より適切な編集の単位でコンフリクト判定, サーバへの反映を行うため, XML文書の一貫性を維持しながら, すべてのユーザの編集意図をより豊富に, 柔軟に取り込むことが可能である.

1　はじめに

サーバ上で一元管理しているXML文書 ([1]) を, 複数のユーザが各自のクライアント上で並行して編集するシステムでは, ユーザがつじつまが合わない編集を行った場合であっても, システムがコンフリクトしないと判定してサーバに反映した結果, XML文書の一貫性を壊すことや, 逆にユーザがつじつまが合う編集を行った場合であっても, システムがコンフリクトすると判定した結果, サーバに反映することが不可能なことが多い.

例えば, 同一の論文を複数の執筆者が並行して編集する時, ある執筆者が他の執筆者の編集内容とつじつまが合わない編集を行った場合であっても, 常に論文全体の一貫性を維持するシステムを実現することや, お互い他の執筆者の編集内容とつじつまが合う編集を行っている場合には, 常にすべての執筆者の編集内容を忠実に反映するシステムを実現することは困難である.

上記の問題は, ユーザがクライアントで行った編集を適切な単位に分割し, この単位ごとにコンフリクト判定し, サーバへ反映することで解決するが, XML文書が持つ文字情報と構造情報のうち, 構造情報に関する編集を適切な単位に分割することは, 実際には難しい.

同一の文書を複数のユーザが並行して編集する際は, 現在 CVS を用いて文書管理する方法が一般的に用いられている. しかしながら, CVS を XML 文書を管理する際に用いると, CVS は文書の行単位でコンフリクト判定を行うために, XML文書が持っている構造情報を反映した単位でのコンフリクト判定を行わず, 望ましい結果が得られない.

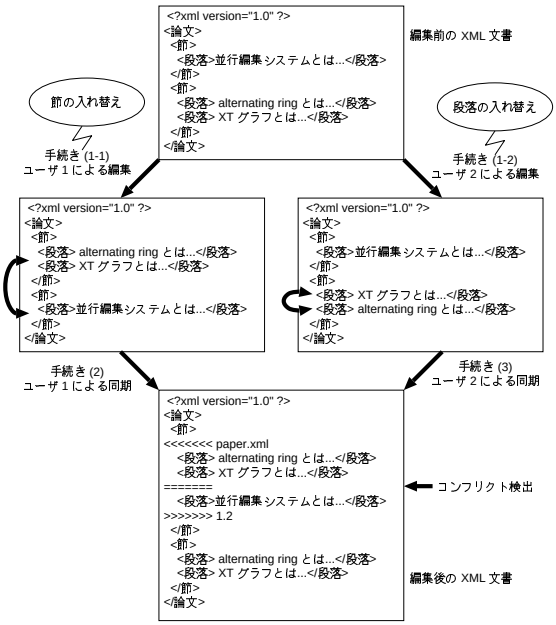

図 1: CVS を用いた XML 文書管理

例えば, CVS を使って XML 文書を管理するシステムで, 図 1 上に示した XML 文書をユーザ 1 とユーザ 2 が並行して編集する場合を考える. 手続き (1-1) でユーザ 1 が図 1 中左に示した節の順番を入れ替える編集を行い, 手続き (1-2) でユーザ 2 が図 1 中右に示した段落の順番を入れ替える編集を行ったとする. 節の順番を入れ替える編集と, 段落の順番を入れ替える編集とは独立であるので, 二人のユーザが行った編集内容は両方とも反映されるべきである. しかし, 編集結果を第一に手続き (2) においてユーザ 1 が, 続いて手続き (3) においてユーザ 2 が同期を取ると, システム

*本研究は通信・放送機構の委託研究『スーパーインターネットプラットフォーム技術の研究開発』の一部として行った.

はコンフリクトが発生したと判断し, 結果は図 1 下に示した文書になる.

本研究では, ユーザが XML 文書に対して行った編集全体を二段階の手続きを用いて分割し, この単位ごとにサーバに反映する手法を確立した. この手法は従来の一貫性維持方式と比較して, より適切な編集の単位でコンフリクト判定, サーバへの反映を行うため, XML 文書の一貫性を維持しながら, すべてのユーザの編集意図をより豊富に, 柔軟に取り込むことが可能である.

本稿は以下に示す構成になっている. 第 2 節において, 我々の XML 文書の一貫性維持方式を用いた並行編集環境システムの説明をする. 第 2 節で説明する通り, 我々の XML 文書の一貫性維持方式のメインは, XML 文書の構造情報に関する編集全体を適切な編集の単位に分割する方法である. 第 3 節において, 構造情報に関する編集を分割する方法を説明する. 第 4 節において, 本研究の関連研究の説明を行い, 第 5 節において, 結論を述べる.

2 XML 文書の並行編集システム

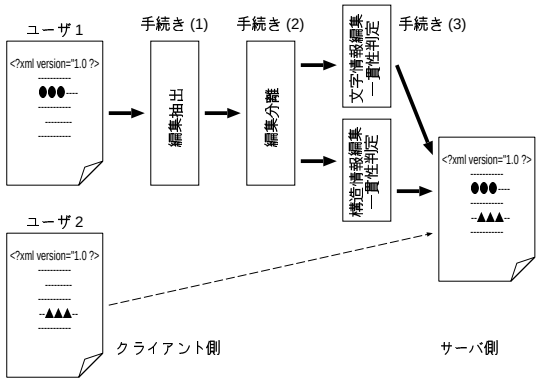

図 2: XML 文書の並行編集システム

図 2 に示した通り, クライアントで編集を行った XML 文書は, (1) 編集前後の XML 文書を比較してどの部分にどんな編集が行われたかを求める, (2) 求められた編集全体を適切な編集の単位に分割する. (3) 分割された編集の単位ごとに一貫性判定を行い, 一貫性を維持すると判定された編集の単位のみをサーバに反映する, という三つの手続きを経て, サーバに反映される. 以下, 個々の手続きに関して簡単に説明を行う.

手続き (2) は, クライアントで行った編集全体を, コンフリクト判定・サーバへの反映を行うのに適切な編集の単位に分割する. ここで XML 文書の編集が適切な単位に分割されているとは, 具体的には以下の二つの性質を満たしていることを指す.

独立性 : 編集の単位間の関連がうすい. より厳密には, 編集の単位全体のうち任意の一部を省略し, さらに編集の単位を任意の順番に入れ替えてサーバに反映した場合, 得られる文書が必ず (well-formed な) XML 文書になる.

極小性 : 独立性を満たしながらこれ以上小さな編集の単位に分割することが困難である.

前述の通り, 編集の単位のうち実際にサーバに反映されるのは, 一貫性を維持すると判定されるもののみである. 編集の単位のうちサーバに反映されないものが存在しても, 結果として得られる文書が必ず (well-formed な) XML 文書になるための条件が『独立性』である. また, 編集の単位が一貫性を維持すると判定される確率をなるべく小さく抑えるためには, 個々の編集の単位がなるべく小さい方がよい. このために満たさなければならない条件が『極小性』である.

我々の XML 文書の一貫性維持方式は, ユーザが行った編集全体を適切な編集の単位に分割するために, 手続き (2) をさらに以下に示す二段階の手続きに分割する.

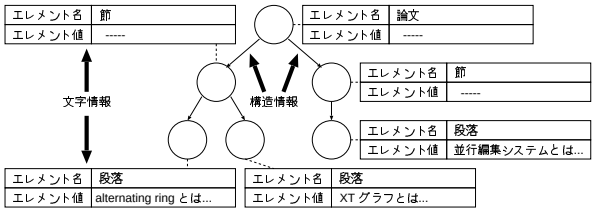

図 3: 文字情報と構造情報とへの分割

手続き (2-1) – 文字情報と構造情報への分割 :

手続き (2) の前半で, 編集全体を二つに分割する.

図 3 に示した通り, XML 文書の持つ情報には, 文字情報と構造情報の二種類があるが, これら二種類の情報が多くの場合において互いに独立であるととらえると, これに対応して, XML 文書の編集には, 文字情報に関する編集と構造情報に関する編集の二種類があり, これら二種類の編集も互いに独立であるととらえることが可能である. 論文の編集の例で言うと, ある段落の内容を変更すること (文字情報の編集) とこの段落の位置を移動すること (構造情報の編集) は互いに独立であるととらえる. この性質を利用して, XML 文書の編

集全体を文字情報の編集と構造情報の編集に分割する.

手続き (2-2) – 文字情報と構造情報の分割 :

手続き (2) の後半で, 二つに分割された編集をそれぞれ適切な, つまり『独立性』と『極小性』を満たす編集の単位に分割する. ただし, 文字情報の編集は自明で適切な分割を持つので本稿では説明を行わない. 構造情報に関する編集の分割方法が我々の XML 文書の一貫性維持方式のメインである.

XML 文書の構造情報の編集を, 以下の三つの手順で適切な単位に分割する.

第一に, 編集前後の XML 文書の構造情報を表現する木を重ね合わせたグラフを作成する. このグラフは, XML 文書に対して行われた構造情報に関する編集全体を表現する. 第二に, 上記グラフを構造情報に関する編集の最小単位に分割する. この手続きは, XML 文書に対して行われた構造情報に関する編集を小さな単位に分割することに相当する. 第三に, 構造情報に関する編集の最小単位をまとめて, 新しい編集の単位を作成する. この手続きは, 小さく分割し過ぎてしまった結果『独立性』を満たさない編集の小さな単位をまとめ適切な, つまり『独立性』と『極小性』を満たす編集の単位を作成することに相当する.

上記二段階の分割を用いると, 従来の一貫性維持方式と比較して, より適切な編集の単位に分割することが可能である. その結果, 我々の XML 文書の一貫性維持方式は従来の一貫性維持方式と比較して, より適切な編集の単位でコンフリクト判定, サーバへの反映を行うため, ユーザの編集意図をより豊富に柔軟に取り込むことが可能である.

手続き (1) は, 編集を行う前後二つの XML 文書において, どのエレメント (PCDATA) とどのエレメント (PCDATA) が対応するかを見付けるという難しい問題を含んでいる. しかし, 本稿では問題を簡単にするために XML 文書中のすべてのエレメントは編集の前後で値が変わらない id を持っていると仮定し, mixed content は認めないものとする. この id を利用すればエレメント (PCDATA) の対応を見付けるのは容易であり, XML 文書のどの部分にどんな編集が行われたかを簡単に求めることができる.

手続き (3) において, 一貫性を保つと判定された編集の単位はサーバに反映し, 一貫性を崩すと判断された編集の単位はサーバに反映せず, サーバ上のログに残す, またはユーザに通知する, サーバに反映し, 一貫性が崩れていることをコメントとして付加するなど, システムによって何通りかの処理が考えられる. しかし, 本稿の主旨とは異なること, 紙面の関係からこれ以上の議論は行わない.

以上簡単に説明を行った手続き (1), (2), (3) の三つの手続きを経て, クライアントで行った編集内容が, サーバの XML 文書に反映される. 本稿の以下において, 手続き全体の中でもっとも難しい技術が必要とされる手続き (2) の後半部分 (手続き (2-2)) に関してのみ, 詳細説明を行う. より具体的には, XML 文書の構造情報に関する編集全体を適切な, つまり独立性と極小性を満たす編集の単位に分割する方法のみにフォーカスを絞って説明を行う.

3 XML 文書の構造情報編集の分割

本節では, XML 文書の構造情報に関する編集を適切な編集の単位に分割する方法にフォーカスを絞って説明する.

本節の構成は以下の通りである. 第 3.1 節において, 本研究の目的に合った XML 文書の構造情報を表現するデータ構造を導入する. 第 3.2 節において, XML 文書の構造情報を表現するデータ構造を用いて, XML 文書の構造情報に関する『編集』を表現 (シミュレート) するデータ構造を導入する. 第 3.3 節において, このデータ構造を用いて, XML 文書の構造情報に関する編集の分割を定義する. 第 3.4 節において, XML 文書の構造情報に関する編集の分割の定義を用いて, XML 文書の構造情報に関する編集の『適切な』分割を定義する. 第 3.5 節において, XML 文書の構造情報に関する編集を適切な編集の単位に分割するアルゴリズムを説明する.

3.1 XT グラフ

本節において, 本研究の目的に合った XML 文書の構造情報を表現するデータ構造を導入する.

XML 文書のエレメントのローカルな構造情報は, このエレメントの左隣のエレメント, 右隣のエレメント, 親エレメントによって決まる. よって XML 文書の編集によって, XML 文書中のエレメントに関するローカルな構造情報が編集されたかどうかは, 編集の前後に

おいてこのエレ メントの左隣のエレ メント, 右隣のエレ メント, 親エレ メントが変化したかどうかによって決まる.

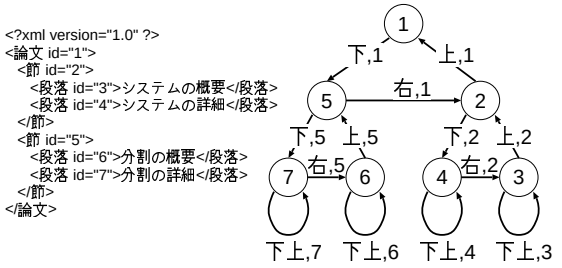

図 4: XT グラフ

図 4 左に示した XML 文書の構造情報を表現する方法は何通りか考えられるが, 我々の XML 文書の一貫性維持方式では, 編集の前後において XML 文書のエレ メントに関するローカルな構造情報が編集されたかどうかを検出する目的に適した, 図 4 右に示したグラフを用いる.

グラフの各頂点は, XML 文書の各エレ メントに一対一に対応する. もしエレ メントが右側に兄弟エレ メントを持つ場合には, グラフはこのエレ メントを開始頂点とし右隣の兄弟エレ メントを終了頂点とする右向き辺 (この辺ラベルを二つの頂点の親頂点の id とする) を持ち, 右側に兄弟エレ メントを持たない場合には, このエレ メントを開始頂点とし親エレ メントを終了頂点とする上向き辺 (この辺ラベルを終了頂点の id とする) を持つ. また, もしエレ メントが子エレ メントを持つ場合には, このエレ メントを開始頂点とし一番左の子エレ メントを終了頂点とする下向き辺 (この辺ラベルを開始頂点の id とする) を持ち, エレ メントが子エレ メントを持たない場合には, このエレ メントを開始頂点とし自分自身を終了頂点とする下上向き辺 (この辺ラベルを開始頂点の id とする) を持つ.

上記グラフが XT グラフ (XML tree graph, XML topology graph) である. 編集を行う前後の XML 文書を XT グラフで表すと, 各頂点に接続する辺が変化したかどうかをチェックするだけで, この頂点に関するローカルな構造情報が編集されたかどうかを判定することが可能である.

ここで XT グラフの厳密な定義を行う. 頂点集合 N, 辺集合 E (ただし, E の各辺は, 開始頂点, 終了頂点に加えて, 辺向き, 辺ラベルを保持するものとする), ルート頂点 r の組から定義されるグラフ $G = (N, E, r)$ において, 同一のラベルを保持する辺のみから構成されるループを 水平 ループと呼び, それ以外のループを 垂直 ループと呼ぶ. 特に, 同一の頂点を高々一回のみ通過するループのことを プリミティブ ループと呼ぶ. グラフ $G = (N, E, r)$ が以下に示す四つの条件を満たすとき, このグラフのことを XT グラフと定義する.

条件一

1. r 以外の任意の頂点 n を終了頂点とする右向きまたは下向き辺が, 合計一つ存在する (今後この辺 e を n の 左辺 と, e の開始頂点を n の 左頂点 と呼ぶ). また, r を終了頂点とする右向き辺, 下向き辺のいずれも存在しない.
2. r 以外の任意の頂点 n を開始頂点とする右向きまたは上向き辺が, 合計一つ存在する (今後この辺 e を n の 右辺 と, e の終了頂点を n の 右頂点 と呼ぶ). また, r を開始頂点とする右向き辺, 上向き辺のいずれも存在しない.
3. 任意の頂点 n を開始頂点とする下向きまたは下上向き辺が, 合計一つ存在する (今後この辺 e を n の 左下辺 と, e の終了頂点を n の 左下頂点 と呼ぶ).
4. 任意の頂点 n を終了頂点とする上向きまたは下上向き辺が, 合計一つ存在する (今後この辺 e を n の 右下辺 と, e の開始頂点を n の 右下頂点 と呼ぶ).

条件二　r 以外の任意の頂点の左辺と右辺のラベルは同一であり, 任意の頂点 n の左下辺と右下辺のラベルは n の頂点 id と同一である.

条件三　水平ループは, このループ中の辺ラベルと同一の頂点 id を保持する頂点を必ず通過する.

条件四　プリミティブな垂直ループは存在しない.

XML 文書の構造情報と XT グラフが一対一に対応することが証明可能であるが, 本稿では紙面の関係から省略する. あるグラフが与えられた場合, このグラフが XT グラフである場合には XML 文書の構造情報を表現していると言え, 逆にこのグラフが XT グラフでない場合には XML 文書の構造情報を表現していないと言える.

3.2　編集グラフ, スイッチ

本節において, 第 3.1 節で説明した XT グラフを用いて, XML 文書の構造情報に関する『編集』を表現 (シミュレート) するデータ構造を導入する.

実際に XML 文書のどの部分にどんな編集がどの順番に行われたかは, 編集開始前と終了後の XML 文書を比較しただけでは分からない. そこで, 編集前後の XML 文書を比較することにより編集をシミュレートする.

編集前の XML 文書と編集後の XML 文書から以下に示す手順でグラフを作成する. 第一に, 編集前の XT グラフ $G_b = (N, E_b, r)$ を作成する. ただしここで, グラフ中のすべての辺は黒で彩色する (E_b は (開始頂点, 終了頂点, 辺向き, 辺ラベル, 辺色) の五つ組の集合とする). 第二に, 編集後の XT グラフ $G_r = (N, E_r, r)$ を作成する. ただしここで, グラフ中のすべての辺は赤で彩色する. 第三に, G_b, G_r から新しいグラフ $G_c = (N, E_b \cup E_r, r)$ を作成する. この結果得られたグラフ G_c を 編集グラフ と呼ぶ. ここでは, 編集の前後を通じてルート頂点r と頂点集合が不変であるという前提を置いている. ルート頂点の変更や, 頂点集合の変更を扱うための拡張は, 付録第 A 節で説明を行う.

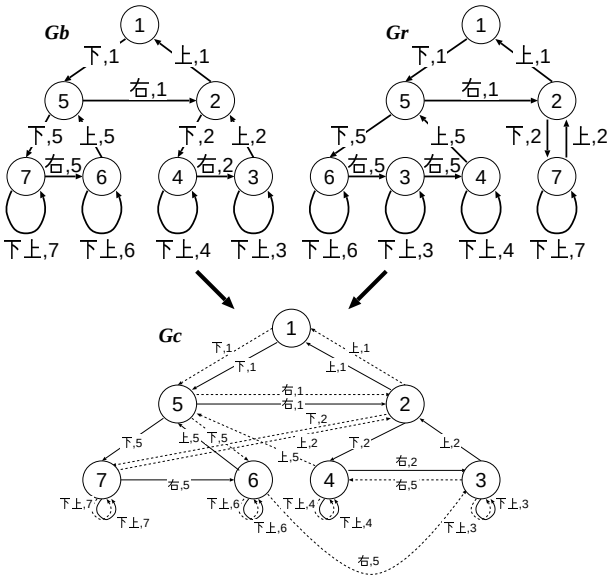

図 5: 編集グラフ

図 5 に, 編集前の XT グラフ G_b と編集後の XT グラフ G_r から作成される編集グラフ G_c を示す. ただし, 黒辺を実線で, 赤辺を点線で表した.

ここで, 辺色を現在と逆の色で再彩色する (現在黒の場合には赤で再彩色する, 逆に現在赤の場合には黒で再彩色する) 操作のことを スイッチ と呼び, 編集グラフの辺のうち一部をスイッチした結果得られるグラフのこともまた編集グラフと呼ぶことにする. さらに, 編集グラフ中の黒辺のみから構成される部分グラフと赤辺のみから構成される部分グラフをそれぞれ編集グラフの 黒部分グラフ, 赤部分グラフ と呼ぶ.

図 5 に示した編集グラフ G_c の黒部分グラフは, 編集前の XT グラフ G_b に等しい. また, 編集グラフ G_c においてすべての辺をスイッチした結果得られる編集グラフ G'_c の黒部分グラフは, (辺色の違いを除いて) 編集後の XT グラフ G_r に等しい. したがって XML 文書の構造情報の編集は, 編集グラフ G_c における辺のスイッチによってシミュレートすることが可能であり, 編集全体は編集グラフ G_c 中すべての辺のスイッチによってシミュレートされる.

3.3 編集グラフ分割

本節において, XML 文書の構造情報に関する編集の分割を定義する. XML 文書の構造情報に関する編集の分割は, XML 文書の構造情報に関する編集全体を互いに関連がうすい (第 2 節で説明した『独立性』を満たす) 編集の単位に分割することである. 定義の際には, 第 3.2 節で説明した編集グラフを用いる.

第 3.2 節において, XML 文書の構造情報に関する編集全体は, 編集グラフ中のすべての辺のスイッチによってシミュレートされることを説明したので, XML 文書の構造情報に関する編集の分割は, 編集グラフ中のすべての辺を (より厳密には, 編集グラフ中のすべての辺のスイッチを) 第 2 節で説明した『独立性』を満たすグループに分割することに帰着される.

編集グラフ中のすべての辺をいくつかのグループに分割したもののうち, グループ間の関連がうすいもの (第 2 節で説明した『独立性』を満たしているもの) を 編集グラフ分割 と呼ぶ[1].

編集グラフ $G_c = (N, E, r)$ の編集グラフ分割は一般に複数あるが, $\{E\}$ は自明な編集グラフ分割である.

[1] より厳密には, 編集グラフ $G_c = (N, E, r)$ において, 以下の条件を満たす E の分割 $\{E_0, E_1, \ldots, E_{m-1}\}$ を G_c の編集グラフ分割と定義する.

$$\forall \sigma \in \Sigma,\ G_i \quad = \quad S(G_{i-1}, E_{\sigma(i-1)}) \in \mathcal{FG} \\ (1 \le i \le m, G_0 = G_c).$$

ただしここで, Σ は自然数 $0, 1, \ldots, m-1$ の並べ換えの集合, つまり m 次の置換の集合であるとし, また, 編集グラフの黒部分グラフが XML 文書の構造情報を表現している (XT グラフになっている) 場合, この編集グラフは フィージブル であると呼び, $\mathcal{FG}$ はフィージブルな編集グラフ全体の集合であるとし, さらに, 編集グラフ $G_c = (N, E, r)$ の部分辺 E' をスイッチする関数を $S(G_c, E')$ で表し, この関数の返り値をスイッチ実行後の編集グラフとする.

3.4 適切な編集グラフ分割

本節において, XML 文書の構造情報に関する編集の分割の定義を用いて, XML 文書の構造情報に関する編集の『適切な』分割を定義する. XML 文書の構造情報に関する編集の『適切な』分割は, XML 文書の構造情報に関する編集全体を互いに関連がうすくなるべく細かい (第 2 節で説明した『独立性』と『極小性』を満たす) 編集の単位に分割することである. 定義の際には, 第 3.3 節で説明した編集グラフ分割を用いる.

第 3.3 節で説明した編集グラフ分割 (編集グラフ中のすべての辺を『独立性』を満たすグループに分割したもの) のうち, これ以上小さな辺グループに分割することが困難であるもの (第 2 節で説明した『極小性』を満たしているもの) を 適切な編集グラフ分割 と呼ぶ.

3.5 適切な編集グラフ分割を求めるアルゴリズム

本節において, XML 文書の構造情報に関する編集を適切な編集の単位に分割するアルゴリズム, つまり第 3.4 節で説明した適切な編集グラフ分割を求めるアルゴリズムを説明する.

アルゴリズムは第 2 節の手続き (2-2) で説明した通り, 以下の三つの手順で行われる. (1) XML 文書に対して行われた構造情報に関する編集全体を表現するグラフを作成する. (2) 編集全体を表現するグラフを, 構造情報に関する編集の最小単位に分割する. ただし, この単位は『独立性』や『極小性』を満たしていない. (3) 構造情報に関する編集の最小単位をまとめて, 『独立性』と『極小性』を満たす新しい編集の単位を作成する.

これを第 3.1 節から第 3.4 節で導入したデータ構造を用いて言い直すと以下の通りになる. (1) 編集前後の XML 文書の構造情報を表現する二つの『XT グラフ』を作成し, これら二つの XT グラフを重ね合わせて, XML 文書に対して行われた構造情報に関する編集全体を表現する『編集グラフ』を作成する. (2) 構造情報に関する編集の最小単位は, 編集グラフにおける個々の辺の『スイッチ』である. そこで, 編集グラフのすべての辺を, 一つの辺のみから構成される (編集グラフの辺と同数の) 辺グループに分割する. ただし, 一つの辺のみから構成される辺グループは, 『独立性』を満たさない. つまり, 編集グラフにおいて, 任意の辺グループを任意の順番でスイッチした結果得られる編集グラフは『フィージブル』でない (得られる編集グラフの『黒部分グラフ』は XT グラフではない). (3) 編集グラフにおいて, 任意の辺グループを任意の順番でスイッチした結果得られる編集グラフの黒部分グラフを XT グラフにするために, 一つの辺のみから構成される辺グループをまとめて『独立性』を満たす新たな辺グループを作成する. 編集グラフの黒部分グラフが XT グラフであるためには, 第 3.1 節で説明した通り, この黒部分グラフが条件一から条件四を満たさなければならない.

本節の以下において, 黒部分グラフが条件一から条件四を満たすためには, どうやって辺グループをまとめればよいかに関して, 個々の条件ごとに説明する.

3.5.1 条件一に対応するまとめあげ

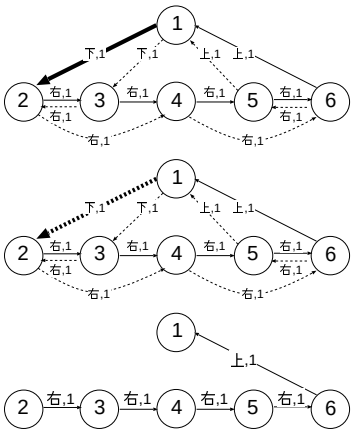

図 6: 条件一を満たさないスイッチ

図 6 上に示した編集グラフにおいて (ただし, グラフが煩雑にならないように下上向き辺は省略している), 辺 (1, 2, 下向き, 1, 黒) をスイッチした結果得られる編集グラフ (図 6 中) の黒部分グラフ (図 6 下) は『条件一』を満たしていないので XT グラフではない[2]. これは, 辺 (1, 2, 下向き, 1, 黒) をスイッチした結果, 黒部分グラフから頂点 1 の左下辺と頂点 2 の左辺がなくなったからであり, 黒部分グラフが条件一を満たし続けるためには, 辺 (1, 2, 下向き, 1, 黒) をスイッチするのと同時に辺 (1, 3, 下向き, 1, 赤) もスイッチしなければならない. 辺 (1, 3, 下向き, 1, 赤) をスイッチした結果, 今度は黒部分グラフにおいて, 頂点 3 の左辺が二本になったので, 黒部分グラフが条件一を満たし続けるためには, 辺 (1, 3, 下向き, 1, 赤) をスイッチするのと同時に辺 (2, 3, 右向き, 1, 黒) もスイッチしなけ

[2] フィージブルな編集グラフにおいて, ただ一つの辺のスイッチを行った結果得られる編集グラフは, 常にフィージブルではない.

ればならない. 上記スイッチの連鎖は, 黒部分グラフが条件一を満たし続けるためには, 辺 (3, 2, 右向き, 1, 赤) をスイッチするのと同時に辺 (1, 2, 下向き, 1, 黒) もスイッチしなければならない, と一番目にスイッチした辺に戻るまで続く.

上記の通り, 黒部分グラフが条件一を満たし続けるためには, 常に同時にスイッチしなければならない辺の組が存在することが分かる. 上記辺の組のことを 交互輪 と呼び, 同一の交互輪中の辺は, すべて同一の辺グループにまとめなければならない. なお, 常に同時にスイッチしなければならない辺の組は, 黒辺と赤辺が交互に現われる輪になることから上記命名を行った.

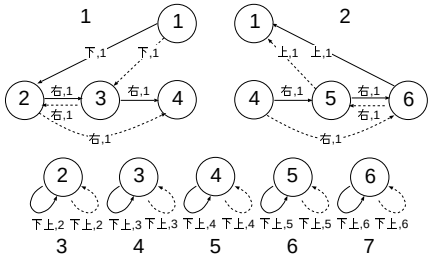

図 7: 交互輪

図 6 上の編集グラフのすべての辺を七つの交互輪にまとめ上げた結果を図 7 に示す.

色以外が同一な二つの辺 (開始頂点, 終了頂点, 辺向き, 辺ラベルが同一な二つの辺) からなる交互輪を特に自明な交互輪と呼ぶ. 自明な交互輪をスイッチする前後において, 編集グラフは不変である. 図 7 には五つの自明な交互輪が含まれている.

上記まとめ上げに要する計算時間は $O(N)$ である. このまとめ上げを行うことにより, 条件一を満たす辺グループを得ることができる.

3.5.2 条件二に対応するまとめあげ

図 5 に示した編集グラフは, 交互輪 (1) {(4, 3, 右向き, 2, 黒), (6, 3, 右向き, 5, 赤), (6, 5, 上向き, 5, 黒), (4, 5, 上向き, 2, 赤)} と交互輪 (2) {(2, 4, 下向き, 2, 黒), (2, 7, 下向き, 2, 赤), (5, 7, 下向き, 5, 黒), (5, 6, 下向き, 5, 赤), (7, 6, 右向き, 5, 黒), (7, 2, 上向き, 2, 赤), (3, 2, 上向き, 2, 黒), (3, 4, 右向き, 5, 赤)} の二つの自明でない交互輪を持つ. 交互輪 (1) をスイッチした結果得られる編集グラフを図 8 上に, 図 8 上の編集グラフの黒部分グラフを図 8 下に示した. 黒部分グラフの頂点 4 に注目すると, 左辺のラベルは 2 であり, 右辺のラベルは 5 であることから『条件二』を満たし

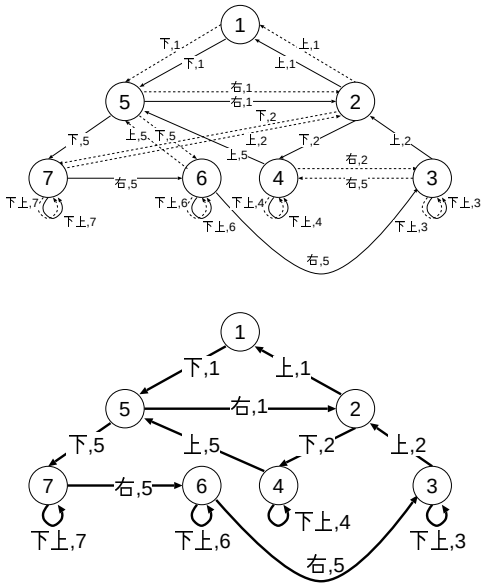

図 8: 条件二を満たさないスイッチ

ていないので, このグラフは XT グラフではない.

黒部分グラフがスイッチ後も条件二を満たし続けるためには, 編集グラフのすべての頂点について, 左辺と右辺を比較し, 赤の左辺と黒の右辺 (または, 黒の左辺と赤の右辺) の辺ラベルが異なる場合には両者を同一の辺グループにまとめなければならない.

上記まとめ上げに要する計算時間は $O(N)$ である. このまとめ上げを行うことにより, 条件二を満たす辺グループを得ることができる.

3.5.3 条件三に対応するまとめあげ

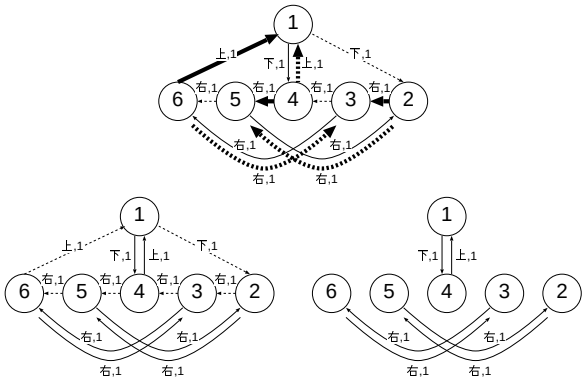

図 9: 条件三を満たさないスイッチ

図 9 上に示した編集グラフは (グラフが煩雑にならないように下上向き辺は省略した), 交互輪 (1) { (4, 1, 上向き, 1, 赤), (6, 1, 上向き, 1, 黒), (6, 3, 右向き, 1, 赤), (2, 3, 右向き, 1, 黒), (2, 5, 右向き, 1, 赤), (4, 5,

右向き, 1, 黒), } と交互輪 (2) { (1, 4, 下向き, 1, 黒), (1, 2, 下向き, 1, 赤), (5, 2, 右向き, 1, 黒), (5, 6, 右向き, 1, 赤), (3, 6, 右向き, 1, 黒), (3, 4, 右向き, 1, 赤), } の二つの自明でない交互輪を持つ. 交互輪 (1) をスイッチした結果得られる編集グラフを図 9 左下に, 図 9 左下の編集グラフの黒部分グラフを図 9 右下に示した. 黒部分グラフの水平ループ {(3, 6, 右向き, 1), (6, 3, 右向き, 1)} と水平ループ {(2, 5, 右向き, 1), (5, 2, 右向き, 1)} はともに条件三を満たさないため. 黒部分グラフは XT グラフではない. この場合には, 下記の方法を用いて複数の交互輪を同一グループにまとめる.

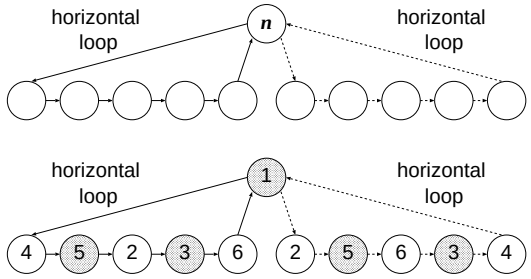

図 10: 条件三に対応するまとめあげ (1)

第一に, 図 10 上に示した通り, フィージブルな編集グラフの各頂点 n について, (1) 頂点 n を開始頂点とする下向き辺を含み黒の辺のみから構成される水平ループ, (2) 頂点 n を開始頂点とする下向き辺を含み赤の辺のみから構成される水平ループの二つのループを探し出す. 例えば, 図 9 上に示した編集グラフの頂点 1 は, 図 10 下に示した二つの水平ループを持つ. 第二に, 二つの水平ループ各々に関して, 頂点を順番に並べた頂点列を求め (ただし, 頂点列の第一頂点と最終頂点はともにを上記頂点 n とする) 二つの文字列の最大共通部分列を求める. 例えば, 図 10 下に示した二つの水平ループ 1, 4, 5, 2, 3, 6, 1 と 1, 2, 5, 6, 3, 4, 1 の最大共通部分列には 1, 5, 3, 1 がある.

最大共通部分列の隣り合う二つの頂点 n_{i-1}, n_i に注目すると, 図 11 上に示した通り, (1) 第一の水平ループの辺のみを使って結ぶパスと (2) 第二の水平ループの辺のみを使って結ぶパスの二つのパスが存在する. 例えば, 図 10 下の最大共通部分列 1, 5, 3, 1 の隣り合う二つの頂点 5, 6 は, 図 11 下に示した二つのパスを持つ. フィージブルな編集グラフの交互輪をスイッチした結果得られる編集グラフの黒部分グラフにおいて, 頂点 n_{i-1} と頂点 n_i を結ぶパスが上記パス (1) またはパス (2) のいずれにもなっていない場合には, スイッチした結果得られる編集グラフは必ずしもフィージブルになっているとは限らない. そこで第三に, 上記二つのパス上の辺を含む交互輪を常に同時にスイッチさせる目的ですべて同一グループにまとめる.

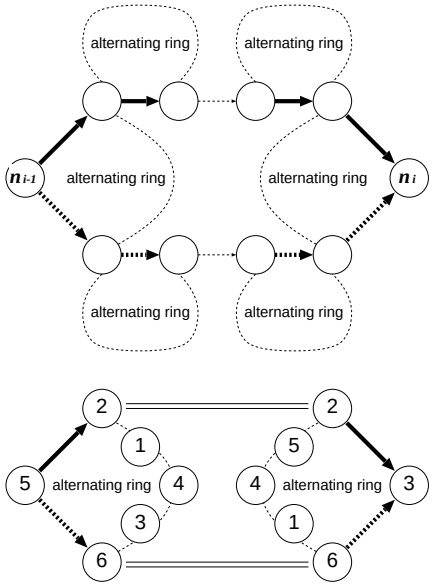

図 11: 条件三に対応するまとめあげ (2)

最大共通部分列問題を解く計算時間は $O(N^2)$ であるので, 上記まとめ上げに要する計算時間は $O(N^2)$ である. このまとめ上げを行うことにより, 条件三を満たす辺グループを得ることができる.

3.5.4 条件四に対応するまとめあげ

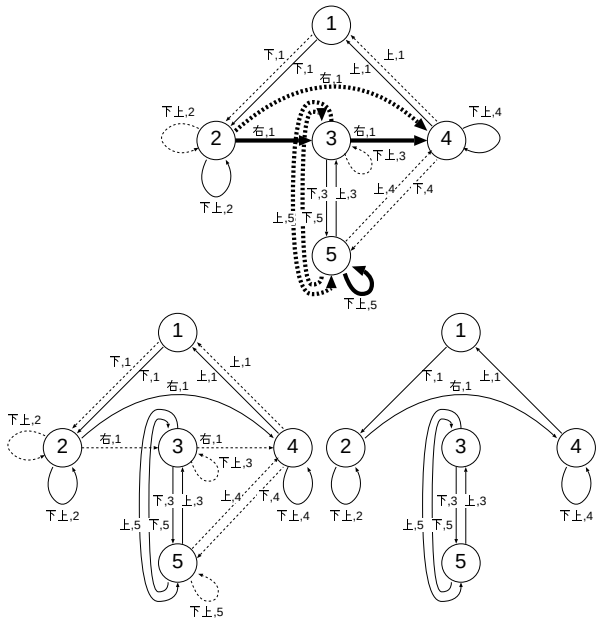

図 12: 条件四を満たさないスイッチ

図 12 上に示した編集グラフは, 交互輪 (1) {(3, 5, 上向き, 5, 赤), (5, 5, 下上向き, 5, 黒), (5, 3, 下向き, 5, 赤), (2, 3, 右向き, 1, 黒), (2, 4, 右向き, 1, 赤), (3, 4, 右向き, 1, 黒)} と交互輪 (2) {(3, 5, 下向き, 3, 黒), (4, 5, 下向き, 4, 赤), (4, 4, 下上向き, 4, 黒), (5, 4, 上向き, 4, 赤), (5, 3, 上向き, 3, 黒), (3, 3, 下上向き, 3, 赤)} の二つの自明でない交互輪を持つ. 交互輪 (1) をスイッチした結果得られる編集グラフを図 12 左下に, 図 12 左下の編集グラフの黒部分グラフを図 12 右下に示した. 黒部分グラフの垂直ループ {(3, 5, 下向き, 3), (5, 3, 下向き, 5)} は条件四を満たさないため, 黒部分グラフは XT グラフではない. この場合には, 下記の方法を用いて複数の交互輪を同一グループにまとめる.

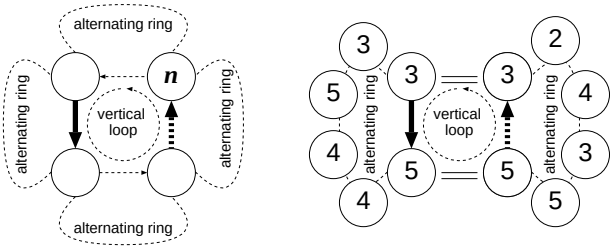

図 13: 条件四に対応するまとめあげ

第一に, フィージブルな編集グラフにおいて, プリミティブな垂直ループ (図 13 左) をすべて探し出す. 例えば, 図 12 上に示した編集グラフは, 図 13 右に示したループを持つ. 編集グラフがフィージブルであることからこの垂直ループは黒辺と赤辺の両方を含んでいる. もし, この垂直ループ中のすべての赤辺がスイッチされすべての辺が黒になった場合は, この編集グラフはフィージブルではなくなる. そこで第二に, 交互輪のスイッチを行った結果, この垂直ループ中の辺がすべて黒になることを避けるために, 垂直ループ上の辺を含む交互輪を常に同時にスイッチさせる目的ですべて同一グループにまとめる.

ただし, グラフ中のすべてのプリミティブな垂直ループを数え挙げるには, 多くの計算時間を要するので, 我々のアルゴリズムでは以下に示す近似解法を用いる. グラフを強連結成分に分解し, 各強連結成分ごとに, (同一強連結成分中に開始頂点, 終了頂点を持つ) 辺ラベルを調べる. 二種類以上の辺ラベルが含まれている場合には, 各強連結成分中の辺すべて同一のグループにまとめる.

上記まとめ上げに要する計算時間は $O(N)$ である. このまとめ上げを行うことにより, 条件四を満たす辺グループを得ることができる.

以上四通りのまとめ上げを行うことにより, 『独立性』, 『極小性』を満たす適切な編集グラフ分割を求めることが可能である.

4 関連研究

同一の XML 文書を複数のユーザが並行して編集する際に, XML 文書を管理するために様々な方法が用いられる.

CVS は最も一般に用いられるツールである. しかしながら, CVS は文書の行単位という XML 文書が持っている構造情報を無視した単位でコンフリクト判定を行うために, 望ましい結果が得られない.

二つの XML 文書の比較して差分を演算するツールに, IBM alphaWorks の XML TreeDiff ([2]) や XML Diff and Merge Tool ([3]) がある. これらツールを用いれは XML 文書に対して行われた編集全体をシミュレート, 分割することが可能である. XML TreeDiff は [4] で提案された方法を拡張し, 部分木の作成, 削除, 移動を追加しているが, 頂点単独の移動を考慮していないので, 編集の適切な分割を得ることができない. また, XML Diff and Merge Tool は頂点の移動を最初から考慮していない. 他にも, XML 文書の差分を演算するツールには diffmk ([5]), Dommitt ([6]), deltaXML ([7]) などが存在するが, 頂点単独の移動を考慮していないという点で XML TreeDiff や XML Diff and Merge Tool と同様である.

5 結論

サーバ上で一元管理されている XML 文書を, 複数のユーザが各自のクライアント上で並行して編集するシステムにおいて, XML 文書の一貫性を維持しながら, ユーザの編集意図をより豊富に, 柔軟に取り込むことを実現するための新しい XML 文書の一貫性維持方式を説明した.

具体的には, 編集前後の XML 文書を表現する木を重ね合わせたグラフを構成し, そのグラフ中から構造情報に関する編集の適切な単位を抽出し, この編集の単位ごとにユーザの編集内容をサーバに反映する手法を確立した. 本手法の計算時間は $O(N^2)$ である.

現在, 本方式を用いたシステムを実装中である. 今後は, 実証実験を通じて問題点の洗い出しを行う予定である.

参考文献

[1] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, "Extensible Markup Language (XML) 1.0 (Second Edition)", http://www.w3.org/TR/2000/REC-xml-20001006, W3C Recommendation, 6 October 2000.

[2] IBM alphaWorks, "XML TreeDiff", http://www.alphaworks.ibm.com/tech/xmltreediff .

[3] IBM alphaWorks, "XML Diff and Merge Tool", http://www.alphaworks.ibm.com/tech/xmldiffmerge .

[4] Kuo-Chung Tai, "The tree-to-tree correction problem", J. Assoc. Comput. Mach., 26, 487-495, 3 July 1979.

[5] Sun Microsystems, "diffmk", http://www.sun.com/xml/developers/diffmk .

[6] Dommitt, "XML Diff and Merge Tool", http://www.dommitt.com .

[7] Monsell EDM Ltd, "deltaXML", http://www.deltaxml.com .

付録

A　XTグラフの拡張

第 1 節や第 3 節において XML 文書の構造情報に関する編集例をいくつか挙げたが, すべての例において, XML 文書のルートエレ メントの構造情報は変更していない. XML 文書のルートエレ メントは親エレ メントを持たないためルートエレ メントの構造情報は変更する場合には本稿で説明した XML 文書の一貫性維持方式をそのまま適用することができないからである. XT グラフの拡張を拡張し, XML 文書のルートエレ メントが仮想的に親エレ メントを持つとみなすことにより, はじめて本稿で説明した XML 文書の一貫性維持方式を適用すること可能である.

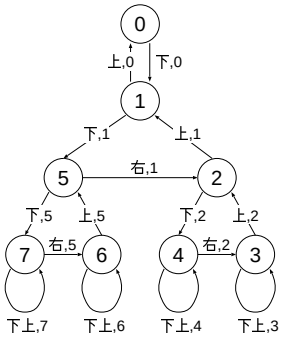

図 14: XT グラフへの仮想ルート頂点の追加

図 4 の XT グラフに仮想ルート頂点を追加する拡張を行った結果を図 14 に示す. 頂点 0 が仮想ルート頂点である.

また, 第 1 節や第 3 節において XML 文書の構造情報に関する編集例をいくつか挙げた. XML 文書の構造情報に関する編集の主なものには, 作成, 削除, 移動の三つがあるが, 本稿で挙げた例では, XML 文書の構造情報に関する編集としてエレ メントの移動のみを扱っている. なぜなら, XT グラフの拡張を行うことで, 作成, 削除は移動に帰着することが可能であるからである.

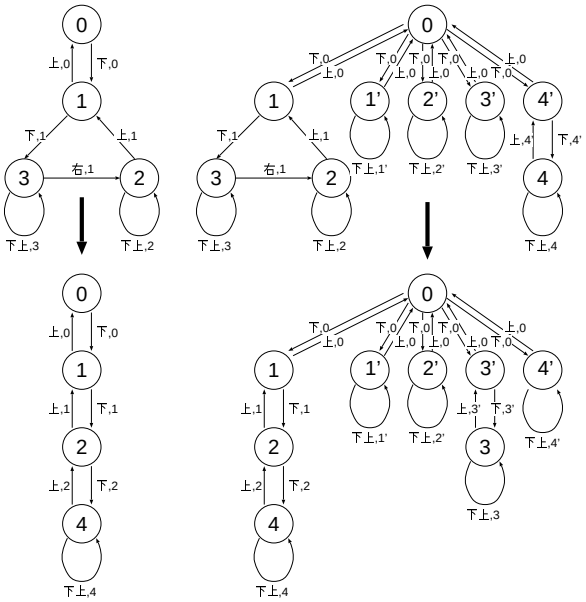

図 15: XT グラフへの仮想頂点の追加

図 15 左上に示した XT グラフにおいて, 頂点 4 を作成し, 頂点 3 を削除する編集を行って図 15 左下に示した XT グラフが得られたとする. このとき, 編集前後の XT グラフに対して以下に説明する拡張を行う. 第一に, XML 文書のすべてのエレ メントに一対一に対応する仮想頂点 1', 2', 3', 4' を編集前後の XT グラフの仮想ルート頂点 0 の子頂点として追加する. 第二に, XML 文書の編集によって作成された頂点 4 を編集前の XT グラフの仮想頂点 4' の子頂点として追加する. 第三に, XML 文書の編集によって削除された頂点 3 を編集前の XT グラフの仮想頂点 3' の子頂点として追加する.

XT グラフに対して以上の拡張を行うことにより, 作成, 削除は移動に帰着することが可能であることは明らかである.