(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2005-18672
(P2005-18672A)
(43)公開日　平成17年1月20日(2005.1.20)

(51)Int.Cl.[7]	F I		テーマコード（参考）
G06F 5/00	G06F 5/00	H	5B009
G06F 12/00	G06F 12/00	511A	5B082
G06F 17/21	G06F 17/21	501T	
	G06F 17/21	570G	

審査請求 未請求　請求項の数 19　OL　（全 59 頁）

(21)出願番号　特願2003-186103(P2003-186103)
(22)出願日　平成15年6月30日(2003.6.30)

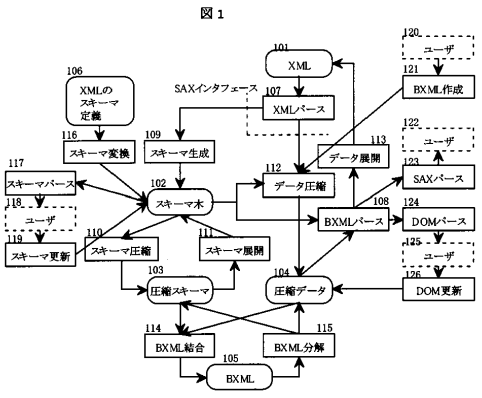
(71)出願人　000005108
株式会社日立製作所
東京都千代田区丸の内一丁目６番６号
(74)代理人　100075096
弁理士　作田　康夫
(72)発明者　室　啓朗
東京都国分寺市東恋ケ窪一丁目２８０番地
株式会社日立製作所中央研究所内
(72)発明者　舟生　幸雄
神奈川県横浜市戸塚区戸塚町２１６番地
株式会社日立製作所ディフェンスシステム
事業部内

Fターム(参考)　5B009 QA06　SA08
5B082 AA11　GA01

(54)【発明の名称】構造化文書の圧縮方法

(57)【要約】
【課題】多量の図形情報が含まれる情報格納件数やデータサイズの巨大な構造化文書を、コンパクトに格納し、高速に解析・更新・作成するための方法を提供すること。
【解決手段】構造化文書を圧縮スキーマ部と圧縮データ部に分離し、構造化文書の個別情報であるテキスト要素、属性値、重複数のみを圧縮データに、それ以外を圧縮スキーマ部に格納する。さらにテキスト要素の辞書化、可変長数値の適用、相対値管理を行うことでサイズを削減する。また標準インタフェースであるＳＡＸ、ＤＯＭを利用して構造化文書を解析・更新・作成する手段を提供すると共に、圧縮スキーマ更新により大量の圧縮データの更新なしで構造の一括変更手段を提供する。
【効果】拡張性の高い構造化文書の作成、及び高速なデータ解析を可能とする。
【選択図】　図１

図１



【特許請求の範囲】
【請求項１】
多量の図形情報が含まれる地図情報のように情報格納件数やデータサイズの巨大な構造化文書をコンパクトに格納し高速に通信するための構造化文書圧縮方法であり、構造化文書を圧縮スキーマ部と圧縮データ部に分離し、構造化文書のタグ名称、タグの性質を示すスキーマ型、タグが保有する子要素の数、タグが保有する属性の数を一つのタグに関する情報とし、構造化文書の属性キー名称、属性の性質を示すスキーマ型を一つの属性キーに関する情報とし、属性や子タグの情報をそれが所属する親タグの情報の直後に連続して再起的に記述する、という方法で構造化文書に出現する全種類のタグ情報・属性キー情報を圧縮スキーマ部に記述し、また構造化文書のタグや属性キーがその位置で何個連続して出現するかを示す重複数、テキスト要素、属性値を、圧縮スキーマで記述した順番通りに圧縮データ部に記述することにより構造化文書をその論理関係を保ったまま変形し、これにより構造化文書のサイズを削減し、また圧縮データ部の値を圧縮スキーマ部のタグ階層関係、タグと属性の関係、タグの重複関係、圧縮データ部のタグ重複数から類推することにより復元することを可能とする構造化文書圧縮方法。
【請求項２】
請求項１の構造化文書圧縮方法において、圧縮スキーマ部のタグに関する情報として、タグが重複して存在するかを示す重複フラグを設定し、通常は重複フラグに「重複」と記述し、圧縮データ部にタグや属性キーがその位置で何個連続して出現するかを示す重複数を記載し、また必ず一回のみ出現するという事が保証されているタグや属性キーについては重複フラグに「単独」と記述し、圧縮データ部の重複数の記載を省略することにより、構造化文書のサイズを削減することを特徴とする構造化文書圧縮方法。
【請求項３】
請求項１の構造化文書圧縮方法において、特定タグの下に複数の指定タグが順不同に複数回出現する可能性がある構造化文書を圧縮する場合において、圧縮スキーマ部における該特定タグのスキーマ型を重複選択型と記述し、圧縮データ部の対応する部分に子タグ総数を記述し、それぞれの子タグの情報として種類を示す識別子、子タグのテキスト要素をペアで記述し、あるいはまた特定タグの下に複数の指定タグのどれか一つのみが出現することが保証されている構造化文書を圧縮する場合において、圧縮スキーマ部における該特定タグのスキーマ型を選択型と記述し、圧縮データ部の対応する部分の子タグ総数の記述を省略し、それぞれの子タグの情報として種類を示す識別子、子タグのテキスト要素のみをペアで記述することにより、構造化文書のサイズを削減することを特徴とする構造化文書圧縮方法。
【請求項４】
請求項１の構造化文書圧縮方法において、繰り返し出現する冗長なテキスト要素や属性値を識別子により代替することにより構造化文書のサイズを削減する方式であり、圧縮スキーマ部の特定タグの付加情報として、該特定タグに繰り返し出現するテキスト要素や属性キーの全部または断片の文字列を辞書として格納し、圧縮データ部の対応するテキスト要素・属性キーの格納位置に、文字列が識別子で代替表現されている事を示すフラグと識別子のペア、文字列が識別子で代替表現されていない事を示すフラグと文字列のペア、の順不同の組み合わせを記述する事により、構造化文書のサイズを削減することを特徴とする構造化文書圧縮方法。
【請求項５】
請求項１の構造化文書圧縮方法において、テキスト要素や属性キーが汎用的な文字列ではなく、整数型や実数型、日付型など、特定の規則で表現される特殊な値であることが保証されている場合、圧縮スキーマ部においてスキーマ型を整数型や実数型、日付型など、該特定型と指定し、圧縮データにおいて該特定型に対応した記述を行うことにより、構造化文書のサイズを削減することを特徴とする構造化文書圧縮方法。
【請求項６】
請求項１の構造化文書圧縮方法において、テキスト要素や属性キーが汎用的な文字列では

なく、整数型や実数型、日付型など、特定の規則で表現される数値であることが保証されており、かつ該数値が緯経度座標など巨大な数値であり、その変動値が該数値の絶対値と比較して微少である場合、圧縮スキーマ部においてスキーマ型を整数型や実数型、日付型など、該特定型と指定し、また圧縮スキーマ部において該数値の基準値を記載し、圧縮データにおいて該特定型に対応した該基準値からの相対値を可変長数値表現で記述し、圧縮データの数値記載部分のサイズを削減することにより、構造化文書のサイズを削減することを特徴とする構造化文書圧縮方法。

【請求項７】

構造化文書を請求項１の構造化文書圧縮方法に従って効率よく圧縮するための方法であり、現在作成中のタグとその子要素に当たる次に到来すると期待されるタグを管理し、構造化文書を解析して開始タグ・終了タグ・テキスト要素・属性をその出現順に取得し、次に到来すると期待されるタグと該開始タグ、現在作成中のタグと該終了タグと比較することによりタグの親子関係・重複関係を決定する方式と、タグのテキスト要素と過去に登録したタグのスキーマ型、属性と過去に登録した属性のスキーマ型とを上書規則を用いて比較することによりスキーマ型を決定する方式からなるスキーマ生成ブロックと、要素単位でそのタグの重複数と該重複数の圧縮データ挿入位置を管理し、テキスト要素を圧縮データに書き込みつつ構造化文書を解析してその開始タグ、終了タグと比較することにより、重複数を勘定して、事後に圧縮データに書き込む方式からなるデータ圧縮ブロックから構成される構造化文書圧縮方法。

【請求項８】

請求項１の構造化文書圧縮方法において圧縮された構造化文書を効率よく解析するための方法であり、現在参照しているタグに関する情報として、圧縮スキーマ部分の該タグに対応するスキーマ要素への参照、該タグと同名のタグが連続して何個出現しているかを示す重複数、該タグがそのうち何番目のタグであるかを示す重複カウンタ、圧縮データにおける該タグに対応する部分へのポインタを管理し、現在参照しているタグが重複型であれば圧縮データから重複数を取得し、スキーマ要素のスキーマ型に従った型のデータを圧縮データから取得してテキスト要素とし、重複カウンタが重複数よりも少なければ次の要素も同じスキーマ型として繰り返し、スキーマ要素が子要素を保持する場合は現在参照しているタグを該子要素に移動して再起的に処理を行い、全ての子要素の解析を終了した時点で現在参照しているタグを親要素に移動して再起的に処理を行うことにより圧縮された構造化文書をパースし、また圧縮データにおいて重複数や各スキーマ型に従ったデータが出現した時に外部プログラムにその出現情報を伝えるメッセージを発行することにより処理を委譲できる構造化文書解析方法。

【請求項９】

請求項１の構造化文書圧縮方法において圧縮された構造化文書を効率よく解析するための方法であり、現在参照しているタグを示すノードポインタとして、圧縮スキーマ部分の該タグに対応するスキーマ要素への参照、該タグと同名のタグが連続して何個出現しているかを示す重複数、該タグがそのうち何番目のタグであるかを示す重複カウンタ、圧縮データにおける該タグに対応する部分へのポインタを保有し、子ノードへのポインタ移動を行う際には圧縮データの属性値部分をスキップし、該子要素へのスキーマ要素を返戻することにより実施し、弟ノードへのポインタ移動を行う際には圧縮データの自分自身並びに子要素部分を再起的にスキップし、該弟要素のスキーマ要素を返戻することにより実施し、また該ノードポインタの情報をその親ノード、祖父ノードと根ノードまで遡り、リストとして管理することにより、親ノードへのポインタ移動を可能とすることにより提供されるノード移動手段と、該圧縮データを該スキーマ要素のスキーマ型に従って復元することによるノードのテキスト要素参照手段を備えることにより、構造化文書の解析を高速に、かつ少ない記憶容量で行うことを特長とする構造化文書解析方法。

【請求項１０】

請求項９の構造化文書解析方法において、指定したタグ名の弟要素・子要素を高速に検索するための方法ならびに装置であり、圧縮スキーマ部分にスキーマ要素を一意に識別する

ための識別子である数値を記述し、該ノードポインタの情報としてさらに検索対象のタグ名を表す識別子である数値を管理し、弟ノードへのポインタ移動を行う際に請求項９の方法に加え、弟候補ノードの識別子と該検索対象の識別子を比較し不一致の場合は次の弟候補ノードを再起的に検索する比較検索方法を適用し、子ノードへのポインタ移動を行う際には請求項９の方法に加え、該比較検索方法を適用することにより、指定したタグ名の弟要素・子要素の高速検索を可能とすることを特徴とする構造化文書解析方法。

【請求項１１】

請求項９の構造化文書解析方法において、タグを示すノードポインタを特定の分野に特化することにより利用者の利便性を向上させる方法であり、地理情報分野における構造化文書の解析手段として地物を示す地物ノード、図形を示す図形ノードを設け、これらのノードの内部情報として請求項９で示した汎用ノードを保有することにより汎用のノード移動手段・テキスト要素参照手段を提供し、更に固定個数の図形ノードのタグ名をあらかじめ記憶しておき、該タグ名と検索対象のタグ名を比較することにより、地物ノードからタグ名を指定しないで図形ノードへポインタ移動を行うことを可能とすることにより、利用者の利便性を向上させる事を特徴とする構造化文書解析方法。

【請求項１２】

請求項１の構造化文書圧縮方法において圧縮された構造化文書を効率よく更新するための方法であり、要素群の挿入時には挿入対象要素群の圧縮データ挿入に先立ち挿入トークンと挿入要素のスキーマ要素を識別する識別子を挿入し、また要素群の削除時には削除対処要素群に先立ち削除トークンを挿入し、また更新された構造化文書の解析時には挿入トークン出現時には重複カウンタの増加を抑制しつつ挿入要素の解析を実行し、また削除トークン出現時には重複カウンタの増加のみを行い削除要素部分の解析をスキップすることにより、圧縮データの重複要素部分の修正を行わずに要素群の挿入・削除を実現することにより、高速な更新を行えることを特徴とし、さらに更新部分に挿入トークンと削除トークンを挿入することにより更新部分が判定できるため、更新の取り消しや更新部分のみの差分通信を可能とすることを特長とする構造化文書更新方法。

【請求項１３】

請求項１の構造化文書圧縮方法において圧縮された構造化文書を効率よく更新するための方法であり、圧縮スキーマ部のタグに関する情報として、さらにタグを隠蔽するかどうかを識別する隠蔽フラグを設定し、該請求項８の構造化文書解析方法において隠蔽されたタグおよびその子要素については解析を中止し、スキップすることにより利用者に対して該タグを隠蔽する方法であり、オリジナルの構造化文書の圧縮データ部分を作り直すことなく利用者の要求する部分の抽出機能を実現することを特長とする構造化文書更新方法。

【請求項１４】

請求項１の圧縮された構造化文書を作成するための方法であり、利用者へのインタフェースとして子要素作成メソッド、弟要素作成メソッド、親へ移動メソッドから構成されるノード作成メソッド群、属性の作成、テキスト要素の作成から構成されるデータ作成メソッド群を提供し、これに対して開始タグ、終了タグ、属性、テキスト要素を請求項７で示した構造化文書圧縮装置に提供することにより、利用者に簡易なインタフェースで、効率的に構造化文書を作成することを可能とする構造化文書作成方法。

【請求項１５】

請求項１４の構造化文章作成方法において、子要素作成メソッド、弟要素作成メソッド、親へ移動メソッドから構成されるノード作成メソッド群を一時的なバッファに記憶しておき、子要素作成メソッド発行後に属性の作成、テキスト要素の作成から構成されるデータ作成メソッド群が一度も発行されない状態で親へ移動メソッドが発行された場合に前回発行された子要素作成メソッドをキャンセルすることにより、０個の子要素を作成する際に不要な子要素を作成することをせず、これにより利用者の構造化文章作成手順を簡易化することを特長とする構造化文書作成方法。

【請求項１６】

請求項１の圧縮された構造化文書同士を結合するための方法であり、結合対象の構造化文

書の圧縮スキーマが同一の場合は、該圧縮スキーマを持つ圧縮された構造化文書を請求項１４の方法で新たに作成し、結合対象のノードを示す圧縮データをコピーし、また結合対象の構造化文書の圧縮スキーマが異なる場合は、双方の構造化文書の圧縮スキーマを含む圧縮スキーマを作成し、結合対象のノードを示す圧縮データを新たに作成した圧縮スキーマに適合する構造に変換したのちにコピーすることにより構造化文書同士を結合するための方法。

【請求項１７】

クライアントとサーバ間の低速な通信回線を通じて効率よく大量の構造化文書を交換するための方法であり、クライアントとサーバ間のクライアント装置側に請求項７で示した構造化文書圧縮装置と請求項８で示した構造化文書解析装置を持つ中継装置を設置し、クライアントの発行する構造化文書を圧縮してサーバに転送し、サーバの発行する構造化文書を解析して元の構造化文書に復元し、クライアントに転送することにより、本発明の構造化文書圧縮装置・解析装置を保有しない一般のクライアントに対し、効率的に大量の構造化文書を交換することを可能とする中継装置。

【請求項１８】

多量の構造化文書を格納し、クライアントの検索要求に従って構造化文書の部分集合を返戻するデータベース装置であり、表形式のデータベース装置を保有し、データベースの要素の一つとして請求項１で示した圧縮データを格納し、また請求項１６で示した構造化文書結合装置を保有し、検索に該当した圧縮データを結合してクライアントに返戻することにより、高速な応答速度を実現することの可能なデータベース装置。

【請求項１９】

空間データを表現する多量の構造化文書を格納し、クライアントの検索要求に従って構造化文書の部分集合を返戻するデータベース装置であり、領域で分割された空間データを表現する請求項１で示した圧縮された構造化文書の集合を保有し、クライアントが要求する空間範囲に該当する圧縮された構造化文書の部分集合を収集し、請求項１３で示すタグ隠蔽方法によりクライアントが要求していない種類の空間データを隠蔽し、また請求項１６で示した構造化文書結合装置を保有し、検索に該当した圧縮データを結合してクライアントに返戻することにより、高速な応答速度を実現することの可能なデータベース装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、ＸＭＬ（ｅＸｔｅｎｓｉｂｌｅ Ｍａｒｋｕｐ Ｌａｎｇｕａｇｅ）やＳＧＭＬ（Ｓｔａｎｄａｒｄ Ｇｅｎｅｒａｌｉｚｅｄ Ｍａｒｋｕｐ Ｌａｎｇｕａｇｅ）等の構造化文書の圧縮・復元・解析・更新・作成に関わる技術に関する。

【０００２】

【従来の技術】

近年、インターネットやイントラネット等、計算機間を接続するネットワークが普及するに従い、ネットワークを用いて計算機を相互接続する動きがある。

【０００３】

複数の異種システム間でデータやプロトコルを統一して利用するための規格の一つとして、ＸＭＬが１９９８年２月にＷ３Ｃ（Ｗｏｒｌｄ Ｗｉｄｅ Ｗｅｂ Ｃｏｎｓｏｒｔｉｕｍ）によって勧告された。ＸＭＬは、同様の規格ＳＧＭＬのサブセットとなっており、文書の中にタグを埋め込む形で、その文書の構造が記述される。ＸＭＬやＳＧＭＬにより記述された文書は、一般に構造化文書と呼ばれる。

【０００４】

ＸＭＬでは、「〈」と「〉」とで囲まれた領域をタグ、「〈タグ名〉」を開始タグ、「〈／タグ名〉」を終了タグ、開始タグと終了タグで囲まれた領域をテキスト要素と呼ぶ。また開始タグの内部に「キー＝”値”」の形式で属性を記述することができる。開始タグと終了タグでテキスト要素を囲むことにより、そのテキスト要素が何についての記述なのかという意味づけを行うことができる。また、開始タグと終了タグの間に再帰的に別の開始

タグ・終了タグを含むことにより階層化されたデータ構造を表現することができる。

【０００５】

ＸＭＬの利用はＷＷＷ（Ｗｏｒｌｄ　Ｗｉｄｅ　Ｗｅｂ）の分野を中心に広がっており、特にＥＤＩ（Ｅｌｅｃｔｒｏｎｉｃ　Ｄａｔａ　Ｉｎｔｅｒｃｈａｎｇｅ）、ＥＣ（Ｅｌｅｃｔｒｉｃ　Ｃｏｍｍｅｒｃｅ）等で広く利用されつつある。ＸＭＬは階層化されたデータ構造を表現することのできる汎用的な仕組みであり、データの表現方式としても、またデータ要求・応答を実現する通信プロトコルとしても利用されている。

【０００６】

ＸＭＬを解析・更新するためのインタフェースも標準化されている。たとえばＸＭＬをイベント駆動形式で走査するＳＡＸ（Ｓｉｍｐｌｅ　ＡＰＩ　ｆｏｒ　ＸＭＬ）、ＸＭＬを木状に展開して管理し、検索・更新を行うＤＯＭ（Ｄｏｃｕｍｅｎｔ　Ｏｂｊｅｃｔ　Ｍｏｄｅｌ）が存在する。これらの実装としては、米国Ｍｉｃｒｏｓｏｆｔ社の　ＭＳＸＭＬ、Ａｐａｃｈｅ　Ｓｏｆｔｗａｒｅ　ＦｏｕｎｄａｔｉｏｎのＤＯＭ／ＳＡＸパーサであるＸｅｒｃｅｓなど、多数のメーカや団体から有償・無償で公開されている。

【０００７】

本発明では，ＸＭＬ文書を圧縮する際にＳＡＸインタフェースを利用している。また本発明では，本発明により圧縮したＸＭＬ文書に対する標準のインタフェースとして，ＳＡＸインタフェース，ＤＯＭインタフェースを提供している。そのため，以下にＳＡＸインタフェースとＤＯＭインタフェースの簡単な説明を行う。

【０００８】

図２を用いてＳＡＸパーサのアルゴリズムを示す。ＳＡＸパーサの目的は、ＸＭＬ文書を走査して、外部のプログラムに対しＸＭＬ文書構造やテキスト要素の情報を提供することにある。ＳＡＸパーサは、ＸＭＬ文書を走査し、ＸＭＬ文書の構造に対応した複数のイベントをユーザプログラムに通知する。具体的には、ＸＭＬ文書を走査する直前に文書開始処理２０１を、ＸＭＬ文書の捜査が終了した時点で文書終了処理２０２を呼び出す。また、開始タグを発見した時には開始タグ処理２０３を、終了タグを発見した時には終了タグ処理２０４を、テキスト要素を発見した時にはテキスト要素処理２０５を、属性を発見した時には属性処理２０６を呼び出す。

ユーザは、上記２０１～２０６までのイベントに対応したイベント処理プログラムを実装することにより、ＸＭＬ文書に対する様々な操作を行うことができる。

【０００９】

図３を用いて、ＤＯＭパーサについて説明する。ＤＯＭパーサの目的は、ＸＭＬ文書をメモリ空間上に木状に展開して、それに対する様々なアクセス方式をユーザに提供することによりＸＭＬ文書の解析・更新機能を提供する。

【００１０】

ＤＯＭパーサが作成する木を一般にはＤＯＭ木と呼ぶが、ここではＸＭＬ文書のインスタンスをそのまま木状に表現することから、インスタンス木と呼ぶことにする。図３のＸＭＬ文書３０１を木状に展開することにより、インスタンス木３０２を得る。インスタンス木３０２内部の円で囲まれる部分はノードと呼ばれ、それぞれＸＭＬ文書のタグが囲む領域に相当する。ＸＭＬ文書３０１では、タグＡの内部に２つのタグＢと１つのタグＣを保有する。また２番目のタグＢの内部には、タグＤ、タグＥが含まれる。タグＢにはキー「ｉｄ」で指定される属性が格納され、タグＤ、タグＥ、タグＣにはテキスト要素が格納されている。インスタンス木３０２は、この階層構造を木状に表現しており、ノードＡに２つのノードＢと１つのノードＣが接続し、また２番目のノードＢに、ノードＤ、ノードＥが接続するという構造を取る。

【００１１】

ユーザにはインタフェースとして、インスタンス木の一つのノードを指すノードポインタが与えられる。ノードポインタはＸＭＬ文書の階層構造を移動することができ、ノードに属するテキスト要素や属性を取得することができる。

【００１２】

例えばノードポインタがノードＢ（３０４）を指しているとする。親へ移動（３０７）することにより、ノードポインタはノードＡ（３０３）に移動する。また弟へ移動（３０８）することにより、ノードポインタはノードＣ（３０５）に移動する。子へ移動（３０９）することにより、ノードポインタはノードＤ（３０６）に移動する。またノードＢ（３０４）で、キー「ｉｄ」の属性を取得する（３１１）と、属性値「ｂ２」が取得でき、またノードＤ（３０６）でテキスト要素を取得する（３１０）と、テキスト要素「ｄ１」が取得できる。　従来のＤＯＭパーサは、メモリ空間上に展開されるインスタンス木３０２のサイズが巨大であった。インスタンス木３０２全体が占めるメモリ空間は、元のＸＭＬ文書３０１の数倍のサイズとなる場合がある。小さなサイズのＸＭＬ文書では問題はないが、本発明が対象とする地理情報分野などの、ノード数が多くサイズも巨大なＸＭＬ文書に対しては、大量のメモリ空間を占有し、かつメモリ空間への展開やＸＭＬ文書への復元に時間がかかる。本発明では、ＸＭＬ文書よりサイズの小さな圧縮ＸＭＬ文書に対するＤＯＭインタフェースを提供し、かつインスタンス木をメモリ空間に展開しない方式を用いることにより、メモリ空間の節約を図る。

【００１３】

ＸＭＬ圧縮技術については何度も繰り返される長いタグ名をタグ辞書に格納し、その識別子をタグの代替として使用することにより、ＸＭＬ文書のサイズを削減するものが、「ＷＡＰ　Ｂｉｎａｒｙ　ＸＭＬ　（ＷＢＸＭＬ）　Ｅｎｃｏｄｉｎｇ　Ｓｐｅｃｉｆｉｃａｔｉｏｎ」として１９９９年にエリクソン・ＩＢＭ・モトローラ・Ｐｈｏｎｅ．ｃｏｍから構成されるＷＡＰフォーラムによりＷ３Ｃに標準化案として提出されている。又、図５に開示する、ＸＭＬ文書の階層構造を示すタグ部分と、実質のデータを示すテキスト要素部分を分離するものがある（特許文献１）。

【特許文献１】特開２００２－１６３２４８号公報

【発明が解決しようとする課題】

ＸＭＬのメリットは次の３点である。（１）データ構造が高い柔軟性を有するため、ベンダに中立なデータ構造となり、ベンダ間の容易な相互接続を可能とする。（２）データ構造が高い拡張性を有するため、ユーザの自由な拡張を可能とする。（３）テキストベースであるため、人間に対し高い視認性を有する。

又、ＸＭＬはテキストベースであることが原因となり、従来フォーマットに比べてデータサイズが大きくなる。又、　テキストベースのため計算機が扱いにくい。例えばＸＭＬ文書で記述された地図を画面上に描画するのは、従来フォーマットで記述された地図を描画するよりはるかに時間がかかる。特に地理情報処理分野におけるＸＭＬ文書は　（１）項目数・サイズ共に桁違いに巨大である、（２）データのほとんどの項目が図形形状を表現する座標数値である。よって、従来のＸＭＬ文書を直接管理する方式では、容量・処理時間ともに実用に耐える時間では利用できず、本発明による圧縮・復元・解析方法が必要となる。

【００１４】

本発明では、上記の課題を解消するため、ＸＭＬ等の構造化文書の意味的構造を保ったままバイナリ構造に圧縮することにより、データサイズを削減する。また圧縮ＸＭＬの圧縮した状態での解析・更新機能を提供することにより、計算機が扱いやすい、従ってＸＭＬ文書と比較して高速な解析処理・加工処理を実現する機能を提供する。また圧縮ＸＭＬを直接作成する機能を提供することにより、高速な構造化文書作成機能を提供する。

【００１５】

【課題を解決するための手段】

本願の開示する発明の概要を説明すると以下の通りである。

構造化文書を圧縮する方法であって、タグとデータを分離し、構造の型に分類し、該型に応じたデータ構造へ変換する。さらに、テキストデータ中の頻出単語を識別子で代替しする。又、複数の数値表示型を採用し、座標をある基準点からの相対的な位置で表示する。さらに、通常の構造化文書を上記圧縮方法で作成される圧縮データ構造へと圧縮する方法、圧縮データ構造から通常の構造化文書への復元方法、通常の構造化文書の復元を行わず

に圧縮データ構造を直接解析する方法、直接更新する方法、圧縮データ構造を直接作成する方法について開示する。
構造化文書の圧縮は、構造化文書を解析してタグ間の関連やタグ内に格納されるデータの型を抽出するスキーマ生成工程と、生成したスキーマに従って構造化文書を適切な圧縮データ構造に変換するデータ圧縮行程で実現する。
構造化文書の復元は、スキーマ構造に従って圧縮データを解析し、重複・省略・条件選択されるタグを復元し、タグのデータ型に従ってデータを復元することにより実現する。
構造化文書の解析は、現在解析しているタグのスキーマ構造の位置・圧縮データの位置を管理するノードポインタを用意し、タグの親・子・弟への移動手段、タグが保有するデータの取得手段を提供することにより実現する。
構造化文書の更新は、構造化文書の部分集合の削除と他構造化文書の挿入で行い、それぞれ削除位置に削除トークンを挿入し、挿入位置に挿入トークンを挿入し、トークンに付随するデータは全体の重複タグ数から無視するという方式で効率的に実現する。またスキーマ構造を修正することにより、圧縮データの修正なしに全体の構造を一括変換することを実現する。
構造化文書の作成は、子要素作成、弟要素作成、親へ移動、属性作成、テキスト要素作成というメソッドを用意し、これに対して開始タグ、終了タグ、属性、テキスト要素を発行することにより実現し、かつ一時バッファを用意してやり直しを許容する機能を実現する。

【００１６】
【発明の実施の形態】
以下本願ではＸＭＬを例にとって説明する。しかし、ＸＭＬに限らず、データが階層的に管理できる構造化言語であれば本願の技術を採用できる。
図６に、地理情報システムの分野で利用されるＸＭＬ文書の例を示す。図６は全体をＭａｐタグで囲み、Ｍａｐタグの子要素として複数のｆｅａｔｕｒｅＭｅｍｂｅｒタグが記述されているという構造を取る。Ｍａｐタグは複数の地物を含む地物集合を表し、具体的には地図を表現する。ｆｅａｔｕｒｅＭｅｍｂｅｒタグの中にはＲｏａｄやＨｏｕｓｅなど複数種類の地物を表すタグが記述され、これらが地図を構成する個々の道路や家屋情報を表現する。例えば図６のＸＭＬ文書は、１／２５、０００縮尺の地図１０ｋｍ四方で、ｆｅａｔｕｒｅＭｅｍｂｅｒタグが数千個から数万個になり、データサイズも１０ＭＢ以上になる場合もある。個々の地物に着目すると、道路地物を表現するＲｏａｄタグの中には、道路名を表すＮａｍｅタグ、車線数を示すＬａｎｅタグ、舗装状態を示すＰａｖｅｍｅｎｔタグ、道路中心線の形状情報を示すｃｅｎｔｅｒＬｉｎｅＯｆタグが記述されている。ｃｅｎｔｅｒＬｉｎｅＯｆタグ内には、折線形状を表すＬｉｎｅＳｔｒｉｎｇタグが記述され、さらにその中には座標系を示すＳＲＳ属性と、折線の構成点を示す多数のｃｏｏｒｄタグが記述されている。ｃｏｏｒｄタグ内にはさらに構成点のＸ座標、Ｙ座標を示すＸタグ、Ｙタグが記述されている。例えば道路、河川、等高線など、複雑な形状の図形を表現する場合、ｃｏｏｒｄタグが１００点から１０００点以上となる場合もある。本発明によるＸＭＬの圧縮構造をＢＸＭＬ（Ｂｉｎａｒｙ　ＸＭＬ）と呼ぶことにする。初めにＢＸＭＬの圧縮方式とデータ構造の説明を行う。その後、（１）通常のＸＭＬ文書をＢＸＭＬ文書に圧縮するための手段、（２）ＢＸＭＬ文書を通常のＸＭＬ文書に展開するための手段、（３）ＢＸＭＬ文書を解析するための手段、　（４）ＢＸＭＬ文書を更新するための手段、（５）ＢＸＭＬ文書を直接作成するための手段、について順番に説明する。

【００１７】
初めに、ＢＸＭＬの圧縮方式とデータ構造について説明する。
ＢＸＭＬは、図７に示す次の５種類の方式を取ることによりＸＭＬ文書を圧縮する。すなわち、「タグとデータの分離」圧縮７０１、「データ型の適用」圧縮７０２、「文字列の辞書化」圧縮７０３、「可変長数値の適用」圧縮７０４、「相対値の適用」圧縮７０５、である。これらのうち、「タグとデータの分離」７０１が必須の圧縮方式であり、残りの方式はオプションである。オプションの各圧縮方式は、単独あるいは図７に示す依存関係

に従った組合せで適用することができる。矢印の矢尻に記された方式を適用するためには、矢印の先頭に記された方式を適用する必要がある。どの組合せを適用するかは、圧縮対象のＸＭＬ文書の特徴や、圧縮速度と圧縮サイズのどちらを優先するかなどのユーザ目的、ＸＭＬを圧縮するかＢＸＭＬを直接作成するかなどの使用条件に応じ、ＢＸＭＬ作成ユーザが決定することができる。

「タグとデータの分離」圧縮７０１は、ＢＸＭＬ文書を圧縮スキーマと圧縮データの二つの部分で構成し、ＸＭＬ文書で何度も繰り返し記述される冗長なタグ名やタグ同士の階層関係を圧縮スキーマにまとめて記述し、個別に必要なテキスト要素や属性値、タグの重複回数などを圧縮データに記述するという方法である。

「データ型の適用」圧縮７０２は、テキスト要素や属性値を単なる文字列として格納するのではなく、整数型・実数型・日付型などの多数の型に分類し、それぞれの型に応じて最適な格納方式を適用することにより、圧縮データのサイズを削減し、また検索速度を向上させる方式である。

「文字列の辞書化」圧縮７０３は、テキスト要素を単語に分解し、頻出単語を圧縮スキーマ部分で辞書化し、圧縮データ部分で頻出単語を識別子で代替する方式である。頻出する長い単語が、短い識別子で代替されることにより、圧縮データのサイズを削減することができる。

「可変長数値の適用」圧縮７０４は、上記の整数型・実数型のデータを、例えばＪａｖａ言語（Ｊａｖａは登録商標である）のｓｈｏｒｔ型整数（２ｂｙｔｅ）、ｌｏｎｇ型整数（４ｂｙｔｅ）、ｄｏｕｂｌｅ型浮動小数点数（８ｂｙｔｅ）のように固定長数値を利用するのではなく、０から１００までは１ｂｙｔｅ、１０１から１００００までは２ｂｙｔｅのように、数値の大きさにより格納サイズが可変となる数値格納方式を適用することにより、圧縮データのサイズを削減する方式である。

「相対値の適用」圧縮７０５は、特に多数の地図図形の座標をＸＭＬ文書で管理する場合に適用し、座標をある起点からの相対値で表現する方式である。前述の「可変長数値の適用」７０４と併用し、相対値表現により小さくなった数値を可変長数値により少ないサイズで記述することにより、圧縮データのサイズを削減することができる。

【００１８】

図６を用いて１番目の圧縮方式である「タグとデータの分離」圧縮７０１について説明する。

【００１９】

図６のＸＭＬ文書は、＜Ｒｏａｄ＞などの開始タグ、＜／Ｒｏａｄ＞などの終了タグなどで構成されるタグ部分と、「Ｒｏｕｔｅ　６」など、タグとタグの間に挟まれたデータ部分に分けられる。ここで、＜Ｒｏａｄ＞タグが多数回繰り返される場合を考えると、データ部分はそれぞれ異なるが、タグ部分は同じ構造が繰り返されるという冗長な構造となっていることが分かる。一般にＸＭＬ文書は、このような冗長なタグ部分が全体サイズの半分近くを占めている。「タグとデータの分離」圧縮７０１では、タグ部分とデータ部分を分離し、最低限必要なデータ部分は個別に記述するが、冗長なタグ部分は１回しか記述しない、という方式を取ることにより、ＸＭＬ文書のサイズを削減する。

【００２０】

図８を用いて、「タグとデータの分離」圧縮７０１について詳細に説明する。

【００２１】

ＢＸＭＬ文書８０３において、分離したタグ部分を表現する構造を圧縮スキーマ８０４、データ部分を表現する構造を圧縮データ８１０と呼ぶことにする。ＸＭＬ文書を構成する情報のうち、圧縮スキーマに記述する情報と、圧縮データに記述する情報は次のように分類される。

【００２２】

圧縮スキーマに記述される情報は次の７つである。（１）タグ名称８０５、（２）重複フラグ８０６、（３）スキーマ型８０７、（４）子要素保有数８０８、（５）保有子要素一覧、（６）属性保有数８０９、（７）保有属性一覧。

（１）タグ名称８０５は、ＸＭＬ文書に出現するタグの名称である。これを圧縮スキーマに記述することにより、ＸＭＬ文書の半分近くを占める開始タグ・終了タグを削減することができ、圧縮サイズは約半分になる。
（２）重複フラグ８０６は、そのタグが重複するか、すなわち連続した兄弟の位置に同じタグが複数並ぶ可能性があるかどうかを示すフラグである。これは「重複」と「単独」の２つの値のいずれかを取り、「単独」の場合はそのタグが必ず１回のみ出現する、「重複」はそれ以外の場合であり、そのタグが０回以上複数回出現する可能性があることを示す。なお、タグが省略される可能性がある場合も「重複」となる。一方、個々のタグが何個出現したかという重複数の情報はデータに依存するため、圧縮データ部分に記述し、圧縮スキーマには記述しない。なお、重複フラグ８０６が「単独」と指定されたタグは、重複数は必ず１回であることが保証されているため、圧縮データ部分に記述する必要はない。
（３）スキーマ型８０７は、そのタグがどのような性質を持つのかを記述する。例えばそのタグが子要素を持つか、あるいは子要素を持たず、テキスト要素を持つかを指定する。さらに子要素を持つ場合、指定した子要素群を順番通りに持つか（ＳＥＱＵＥＮＣＥ型）、順不同に持つか（ＭＣＨＯＩＣＥ型）、いずれか一つのみを持つか（ＣＨＯＩＣＥ型）、あるいはテキスト要素を持つ場合、任意文字列が入るか、整数や実数のみしか入らないか、などを細かく指定することができる。これはＸＭＬ文書に明示的に記述されていない情報であるが、本発明により効率のいい圧縮を行うため、本発明の圧縮工程で付与される情報である。なお、ＸＭＬ　Ｓｃｈｅｍａ等のＸＭＬ文書定義言語で圧縮対象ＸＭＬ文書の構造が提供される場合は、ここからスキーマ型８０７を取得することも可能である。
（４）子要素保有数８０８は、そのタグに出現する可能性のある全子要素の種類数である。これは種類の数であり、実際の子要素数ではないため、個々のタグで子要素が省略されたり子要素が重複して出現したりして、実際の子要素数と異なっても構わない。個々のタグにおいてどの子要素が省略されたか、またどの子要素が何個重複して出現したかという情報はデータに依存するため、圧縮データ部分に記述し、圧縮スキーマには記述しない。
（５）保有子要素一覧は、子要素保有数８０８で指定した数の、子要素情報の集合である。それぞれの子要素情報も再帰的に、上述した（１）タグ名称、（２）重複フラグ、（３）スキーマ型、（４）子要素保有数、（５）保有子要素一覧、（６）属性保有数、（７）保有属性一覧を保有する。
（６）属性保有数８０９は、そのタグに出現する可能性のある全属性の種類数である。これは種類の数であり、実際の属性数ではないため、個々のタグで属性が省略されてこの属性保有数よりも少ない属性を持つ場合があっても構わない。個々のタグにおいてどの属性が省略されたかという情報はデータに依存するため、圧縮データ部分に記述し、圧縮スキーマには記述しない。
（７）保有属性一覧は、属性保有数８０９で指定した数の、属性情報の集合である。属性情報とは、属性キーと属性値であるが、属性キーをタグ名称、属性値をテキスト要素と置き換えることにより、特殊な子要素と考えることができる。従って属性情報は、上述した子要素情報と同様に（１）キー名称、（２）重複フラグ、（３）スキーマ型、を持つ。属性は階層構造を持たないため子要素保有数は必ず０、属性は属性を持たないため、属性保有数も必ず０である。そのため、子要素保有数８０８、保有子要素一覧、属性保有数８０９、保有属性一覧は省略可能である。
【００２３】
圧縮データ８１０には、次の４つの情報が記述される。（１）テキスト要素、（２）属性値、（３）タグ重複数、（４）選択タグ識別子。
（１）テキスト要素は、ＸＭＬ文書においてタグとタグの間に囲まれた部分である。ＸＭＬ文書のテキスト要素がそのまま記述されたり、図７に示す他の圧縮方式７０２～７０５により圧縮されたテキスト要素が記述されたりする。
（２）属性値はＸＭＬ文書の保有する属性の値である。これもテキスト要素と同様、ＸＭＬ文書の属性値がそのまま記述されたり、図７に示す他の圧縮方式７０２～７０５により圧縮された属性値が記述されたりする。

（３）タグ重複数は、ＸＭＬ文書の兄弟の位置に存在する、同名のタグがいくつ連続で続くかを示す数値である。これはＸＭＬ文書により変化し、また一つのＸＭＬ文書内でも場所により変化するため、圧縮スキーマではなく圧縮データに直接記述する。
（４）選択タグ識別子は、子要素として複数種類のタグのどれか一つが出現する可能性がある場合、どのタグが出現したかを示すための識別子である。圧縮スキーマ８０４の子要素一覧として、あるタグの子要素として出現する可能性のあるタグ一覧を記述するが、その記述順の通し番号を識別子とすることができる。属性の場合も同様に、複数種類のキーのどれか一つが出現する可能性がある場合、どのキーが出現したかを選択タグ識別子として記述しても構わない。

【００２４】

ＸＭＬ文書は、図８に示す７個のパターンの組み合わせで表現できる。（１）子要素パターン８１１、（２）兄弟要素パターン８１２、（３）重複要素パターン８１３、（４）省略要素パターン８１４、（５）選択要素パターン８１５、（６）重複選択要素パターン８１６、（７）属性パターン８１７、である。以下、上記７個のパターンを例に取り、ＸＭＬ文書がどのような構造の圧縮スキーマ、圧縮データで表現されるかを説明する。
（１）子要素パターン８１１は、タグの中に子要素が存在するパターンである。ＸＭＬサンプル８０２では、タグＡの子要素としてタグＢが、タグＢの子要素としてタグＣが格納されている。タグＣは子要素を持たず、その代わりテキスト要素として文字列ｔｅｘｔを持つ。

【００２５】

圧縮スキーマでは、最初にルートタグＡの情報を記述する。タグＡは１回のみ出現するため重複タグ８０６は「単独」となる。また、タグＢのみを子要素として持つため、スキーマ型８０７は「ＳＥＱＵＥＮＣＥ」、子要素保有数８０８は１個となる。タグＢ、タグＣの情報はタグＡの情報に連続して記述する。タグＢも１回のみ出現するため重複タグ８０６は「単独」、また、タグＣのみを子要素として持つため、スキーマ型８０７は「ＳＥＱＵＥＮＣＥ」、子要素保有数８０８は１個となる。タグＣも１回のみ出現するため重複タグ８０６は「単独」、また、タグＣは子要素を持たず、テキスト要素のみを持つため、スキーマ型８０７は「文字列」となる。子要素パターンの圧縮データは、タグＣのテキスト要素であるｔｅｘｔ１のみを記述する。これは、圧縮スキーマでの定義で、タグＡ、タグＢがテキスト要素を持たないことを示しているため、最初に出現するテキスト要素がタグＣのものであるということが自明だからである。
（２）兄弟要素パターン８１２は、一つのタグの中に複数の異なるタグが順番に格納されるパターンである。ＸＭＬサンプル８０２では、タグＡの中にタグＢとタグＣが順番に格納されている。　兄弟要素パターンを示す圧縮スキーマでは、その兄弟タグの親タグのスキーマ文に続けて、兄弟タグをその出現順に記述する。タグＡは１回のみ出現するため重複タグ８０６は「単独」となる。また、タグＢ、タグＣを子要素として持つため、スキーマ型８０７は「ＳＥＱＵＥＮＣＥ」、子要素保有数８０８は２個となる。タグＢ、タグＣの情報はタグＡの情報に連続して記述する。タグＢも１回のみ出現するため重複タグ８０６は「単独」、子要素を持たず、テキスト要素のみを持つため、スキーマ型８０７は「文字列」となる。タグＣも１回のみ出現するため重複タグ８０６は「単独」、子要素を持たず、テキスト要素のみを持つため、スキーマ型８０７は「文字列」となる。

【００２６】

兄弟要素パターンを示す圧縮データでは、テキスト要素が出現順に記述される。圧縮スキーマによれば各テキスト要素がどのスキーマに対応するか自明だからである。
（３）重複要素パターン８１３は、一つのタグの中に複数の同じタグが格納されるパターンである。ＸＭＬサンプル８０２では、タグＡの中に２個のタグＢが重複要素として格納され、その後にタグＣが兄弟要素として格納されている。

【００２７】

重複要素パターンを示す圧縮スキーマでは、重複する要素の重複フラグに「重複」を設定する。タグＡは１回のみ出現するため重複タグ８０６は「単独」となる。また、タグＢ、

タグＣを子要素として持つため、スキーマ型８０７は「ＳＥＱＵＥＮＣＥ」、子要素保有数８０８は２個となる。タグＢは２回出現するため重複タグ８０６は「重複」、子要素を持たず、テキスト要素のみを持つため、スキーマ型８０７は「文字列」となる。タグＣは１回のみ出現するため重複タグ８０６は「単独」、子要素を持たず、テキスト要素のみを持つため、スキーマ型８０７は「文字列」となる。

【００２８】

重複要素パターンを示す圧縮データでは、重複するタグの位置に重複数を格納する。本例の場合、タグＢが２回重複して出現しているため、重複数として２を格納する。テキスト要素は出現順に記述する。スキーマの解釈によれば、タグＢの重複フラグは「重複」であるため、圧縮データ先頭の整数「２」はタグＢの重複数であることが分かる。それに続く文字列ｔｅｘｔ１、ｔｅｘｔ２は、タグＢのスキーマ型が「文字列」であり、重複数が２であることから、それぞれ重複する２つのタグＢのテキスト要素であることが分かる。さらに続く文字列ｔｅｘｔ３は、タグＢの次に定義されているタグＣのテキスト要素であることが分かる。

（４）省略要素パターン８１４は、あるタグの子要素が、ある場所では存在するが、別の場所では省略される場合があるパターンである。これは、省略されたタグは重複フラグが「重複」のタグであり、重複数が０であるとみなすことにより、重複要素パターンと同一パターンとみなすことができる。ＸＭＬサンプル８０２では、最初のタグＡの子要素としてタグＢ、タグＣが存在するが、最初のタグＡではタグＣが省略されており、次のタグＡではタグＣが省略されていない。

【００２９】

省略要素パターンを示す圧縮スキーマでは、省略する要素の重複フラグに「重複」を設定する。ＸＭＬサンプルでは、タグＣが省略される場合があるため、タグＣのスキーマ文に「重複」を設定する。

タグＡは２回のみ出現するため重複タグ８０６は「重複」となる。また、タグＢ、タグＣを子要素として持つため、スキーマ型８０７は「ＳＥＱＵＥＮＣＥ」、子要素保有数８０８は２個となる。タグＢは必ず１回のみ出現するため重複タグ８０６は「単独」、子要素を持たず、テキスト要素のみを持つため、スキーマ型５０７は「文字列」となる。タグＣは省略される場合もあるため、重複タグ８０６は「重複」、子要素を持たず、テキスト要素のみを持つため、スキーマ型８０７は「文字列」となる。

【００３０】

省略要素パターンを示す圧縮データでは、最初にタグＡの重複数である２を格納する。次に最初のタグＡの子要素であるタグＢのテキスト要素「ｔｅｘｔ１」を記述する。次にタグＣの重複数を記述する。この場合タグＣは省略されているため、重複数は０である。次に２番目のタグＡの子要素であるタグＢのテキスト要素「ｔｅｘｔ２」を記述する。次にタグＣの重複数を記述する。この場合タグＣは省略されていないため、重複数は１である。引き続いてタグＣのテキスト要素「ｔｅｘｔ３」を記述する。スキーマの解釈によれば、タグＡの重複フラグは「重複」であるため、最初の整数「２」はタグＡの重複数であることが分かる。タグＡの最初の子要素であるタグＢの重複フラグは「単独」であるため、次の文字列ｔｅｘｔ１はタグＢの保有である。タグＡの次の子要素であるタグＣの重複フラグは「重複」であるため、次の整数「０」はタグＣの重複数である。次のテキスト要素ｔｅｘｔ２は二番目のタグＡの先頭子タグＢのテキスト要素である。次の整数「１」は、タグＣの重複フラグであり、値が１であるため続く文字列ｔｅｘｔ３はタグＣのテキスト要素であることが分かる。

（５）選択要素パターン８１５は、あるタグの子要素が複数の候補のうちの一つであるというパターンである。ＸＭＬサンプル８０２では、最初のタグＡの子要素としてタグＢが来ており、次のタグＡの子要素としてタグＣが来ている。

【００３１】

選択要素パターンを示す圧縮スキーマでは、子要素が選択要素となるタグに対し、スキーマ型を「ＣＨＯＩＣＥ」と設定する。そして選択される可能性のある子要素情報を連続し

て記述する。なお、選択される子要素の重複数は必ず「単独」とする。
タグＡは２回のみ出現するため重複タグ８０６は「重複」となる。また、タグＢ、タグＣのいずれか一つを子要素として持つ可能性があるため、スキーマ型８０７は「ＣＨＯＩＣＥ」、子要素保有数８０８は２個となる。タグＢは選択される子要素のため重複タグ８０６は「単独」、子要素を持たず、テキスト要素のみを持つため、スキーマ型５０７は「文字列」となる。タグＣも同様に、重複タグ８０６は「単独」、子要素を持たず、テキスト要素のみを持つため、スキーマ型８０７は「文字列」となる。
【００３２】
選択要素パターンを示す圧縮データでは、タグＡの重複数の後に、実際に到来したタグの識別子を記述し、その後に到来したタグのデータを記述する。ここでいうタグの識別子は、同じ階層のタグ群から選択できればよいため、圧縮スキーマ部で定義したタグの出現順序を０で始まる通し番号として記述するので構わない。例えばＸＭＬサンプルでは、Ａの重複数の後、タグＢが出現したことを示すタグＢの識別子「０」を格納し、タグＢのテキスト要素を続いて格納する。その後、次にタグＣが出現したことを示す識別子「１」を格納し、続いてタグＣのデータを記述する。
（６）重複選択要素パターン８１６は、重複要素と選択要素が同時に生じるパターンである。ＸＭＬサンプル８０２では、タグＡの子要素として、タグＢとタグＣが順不同で複数回出現している。子要素が重複選択要素となるタグに対し、スキーマ型を「ＭＣＨＯＩＣＥ」と設定する。そして選択される可能性のある子要素情報を連続して記述する。なお、選択される子要素の重複数は必ず「単独」とする。
タグＡは１回のみ出現するため重複タグ８０６は「単独」となる。また、タグＢ、タグＣが順不同で複数回出現する可能性があるため、スキーマ型８０７は「ＭＣＨＯＩＣＥ」、子要素保有数８０８は２個となる。タグＢは選択される子要素のため重複タグ８０６は「単独」、子要素を持たず、テキスト要素のみを持つため、スキーマ型５０７は「文字列」となる。タグＣも同様に、重複タグ８０６は「単独」、子要素を持たず、テキスト要素のみを持つため、スキーマ型８０７は「文字列」となる。
【００３３】
重複選択要素パターンを示す圧縮データでは、タグＡの保有する子タグ群の総数を記述したあとに、選択要素パターンと同様、実際に到来したタグの識別子と、タグのデータのペアを、子タグ群の総数と同じ回数、連続して記述する。例えばＸＭＬサンプルでは、タグＡの子要素であるタグＢ、タグＣを合わせた重複数「４」を記述し、最初に到来したタグＢの識別子「０」、タグＢのテキスト要素「ｔｅｘｔ１」、２番目に到来したタグＣの識別子「１」、タグＣのテキスト要素ｔｅｘｔ２、３番目に到来したタグＢの識別子「０」、タグＢのテキスト要素「ｔｅｘｔ３」、４番目に到来したタグＣの識別子「１」、タグＣのテキスト要素ｔｅｘｔ４、を順に記述する。
（７）属性パターン８１７は、タグに属性が格納されるパターンである。ＸＭＬサンプル８０２では、タグＡの属性としてキーａ、値ｔｅｘｔ１という属性が格納されている。
【００３４】
属性パターンを示す圧縮スキーマでは、属性を含むタグのスキーマ文に、属性保有数８０９として格納される属性の数を記述し、さらに保有する属性の情報を連続して記述する。属性を示すスキーマ文は、子要素を示すスキーマ文とほとんど同じ構造を持つ。タグ名として、属性のキーを持つ。属性は省略可能だが同じキーの属性が複数来ることはない、すなわち重複数は必ず０か１であるため、重複フラグ５０６には特別に「属性」という値を設定しても構わない。また、属性は子要素や属性を持たないため、子要素保有数５０８、属性保有数５０９は省略して構わない。なお、属性を示すスキーマ文は、子要素を示すスキーマ文に先行して記述する。
【００３５】
タグＡは１回のみ出現するため重複タグ８０６は「単独」となる。タグＢのみを子要素として持つため、スキーマ型８０７は「ＳＥＱＵＥＮＣＥ」、子要素保有数８０８は１個となる。また、属性ａを持つため、属性保有数８０９は１個となる。属性ａの重複タグ８０

６は「属性」、属性のスキーマ型８０７は「文字列」となる。タグＢは１回のみ出現するため重複タグ８０６は「単独」、また、タグＢは子要素を持たず、テキスト要素のみを持つため、スキーマ型８０７は「文字列」となる。

【００３６】

属性パターンを示す圧縮データでは、属性の値とテキスト要素を、その出現順に記述する。属性が出現する可能性があるかどうかは圧縮スキーマに記述され、属性が省略されたかどうかは圧縮データに記述され、また属性は子要素に先行して記述するというルールから、属性の値を一意に取得することができる。

図８の圧縮データ８１０の構造について、図９のサンプルＸＭＬ文書９０１～９０５を用いて詳細に説明する。

【００３７】

ＸＭＬ文書９０１では、複数のタグＢが存在し、さらにタグＢの中に複数のタグＣが存在する。これに対する圧縮データは９０６の構造を取る。最初にタグＢの重複数Ｎｂ（この場合は重複数２）を記述し、その後にタグＢの子要素の圧縮データ９１３、９１４を記述する。圧縮データ９１３は重複する二つのタグＣを表現するため、再起的に最初のタグＣの重複数Ｎｃ１（この場合は重複数２）を記述し、その後に最初のタグＣのテキスト要素であるｃ１、次のタグＣのテキスト要素であるｃ２を記述する。ｃ１、ｃ２は単なる文字列が記述される場合もあるし、図７の圧縮方式で説明したデータ型の利用７０２、文字列の辞書化７０３、可変長数値の適用７０４、相対値の適用７０５を適用し、さらに圧縮された構造が格納される場合がある。ｃ１、ｃ２がどの形式の構造であるかはテキスト要素が含まれるタグの圧縮スキーマ８０４のスキーマ型８０７を参照することにより判断できる。また圧縮データのある位置の情報がＮｂなどの重複数であるか、ｃ１、ｃ２などのテキスト要素であるかの判断も、圧縮スキーマ８０４の重複フラグ８０６を参照することにより判断できる。

【００３８】

ＸＭＬ文書９０２は、ＸＭＬ文書９０１に対し、タグＣの兄弟要素としてタグＤが追加された構造を持つ。これに対する圧縮データは９０７の構造を取る。最初にタグＢの重複数Ｎｂ（この場合は重複数２）を記述し、その後にタグＢの子要素の圧縮データ９１５、９１６を記述する。圧縮データ９１５はさらにタグＣとタグＤから構成されているため、出現順番通りタグＣの圧縮データ９１７を記述した後、タグＤの圧縮データ９１８を記述する。圧縮データ９１７は、タグＣが重複要素であるため、最初にタグＣの重複数Ｎｃ１を記述し、２つのタグＣのテキスト要素ｃ１、ｃ２を記述する。同様に圧縮データ９１８は、タグＤの重複数Ｎｄ１を記述した後　、２つのタグＤのテキスト要素ｄ１、ｄ２を記述する。圧縮データ９１６も同様である。

【００３９】

ＸＭＬ文書９０３は、ＸＭＬ文書９０２の特別な構造であり、タグＣ、タグＤが重複せず、単独で出現する。これに対する圧縮データは、圧縮データ９０７と同様な考え方で９０８の構造を取ってもよいが、タグＣ、タグＤは必ず１個しか出現しないことが明らかになっている場合、重複数Ｎｃ１、Ｎｄ１などは全て１個となり、冗長である。そのため９０９の構造に示すように、重複数を取り除いた構造を取ることができる。あるタグが重複数を取るかどうかの判断は、図８におけるそのタグに対応する圧縮スキーマ８０４において、重複フラグ８０６が「単独」の場合重複数は省略され、「重複」の場合重複数が記述されるという方法で判定できる。圧縮データ９０９の構造は、最初のタグＢの重複数Ｎｂを除き、全てがＸＭＬ文書のテキスト要素のみであり、ＸＭＬ文書のタグ階層構造を示す情報がほとんど除去された構造となっている。多くのＸＭＬ文書の特徴として、単独要素の数が重複要素の数よりも多いため、重複フラグ８０６が「単独」の場合重複数を省略する、という圧縮方式は、効率のいい圧縮が期待できる。

【００４０】

ＸＭＬ文書９０４では、タグＢの中にタグＣ、タグＤ、タグＥのいずれか一つが格納される。これに対する圧縮データは、圧縮データ９０７と同様な考えかたで９１０の構造を取

ってもよいが、Ｎｃｉ、Ｎｄｉ、Ｎｅｉ（ｉ＝１、２、３．．．）の全ての組において、Ｎｃｉ、Ｎｄｉ、Ｎｅｉのいずれか一つを除いて全て０個であることが保証されるため、冗長である。そのため９１１の構造に示すように、どの種類のタグが出現するかという識別子Ｋｃ、Ｋｄ、Ｋｅの何れかを記述し、その後にその種類のタグに関するデータのみを記述する、という方法を取ることにより、圧縮データのサイズを低減することができる。ここで識別子は、圧縮スキーマにおいて定義されている子要素Ｃ、Ｄ、Ｅの出現順の通し番号を付ければよい。あるタグＢの子要素を表現する圧縮データが９１１の構造になっていることを判定するためには、タグＢに対応する圧縮スキーマにおいて、スキーマ型８０７がＳＥＱＵＥＮＣＥ型の場合９１０の構造、ＣＨＯＩＣＥ型の場合９１１の構造である、という形で判定できる。

【００４１】

ＸＭＬ文書９０５では、タグＢの中にタグＣ、タグＤ、タグＥが順不同に０個以上格納される。これに対する圧縮データは、タグＢの子要素の圧縮データとして、最初に順不同に出現するタグＣ、タグＤ、タグＥの総数Ｎｃｄｅ（この場合６）を記述し、その後に圧縮データ９１１と同様な方式で出現するタグの識別子、テキスト要素の組み合わせを記述する、という方式で記述できる。圧縮データ９１２の構造を取ることを判定するためには、タグＢに対応する圧縮スキーマにおいて、スキーマ型８０７がＭＣＨＯＩＣＥ型の場合９１２の構造である、という形で判定できる。

【００４２】

次に、図１０を用いて、ＸＭＬ圧縮構造の２番目の方法であり、最初のオプションである「文字列の辞書化」方式について説明する。本圧縮方式は、圧縮データ領域を対象とする。　前述したように、圧縮データ領域は主にＸＭＬ文書のテキスト要素を記述する部分である。ただし図８の７番目のパターン「属性」で示したとおり、本発明の圧縮方式では属性部分をテキスト要素を持つ子要素と同一の方式で管理している。そのため本発明におけるテキスト要素の圧縮方式は、属性の値部分の圧縮にも容易に適用することができる。

【００４３】

ＸＭＬ文書のデータ部にも何度も繰り返し出現する冗長な部分が存在する。例えば図１０のＸＭＬ文書１００１の例では、＜Ｔｙｐｅ＞タグの中に道路区分として、「国道」「県道」の２種類のいずれかが記述されている。一般に、同じ名称のタグが多量に存在し、そのタグの持つテキスト要素として取りうる値の種類がタグの総数よりも十分小さければ、そのテキスト要素を表す識別子を用意し、その識別子でテキスト要素を置き換えることにより冗長性が減少し、圧縮効率が向上する。

【００４４】

ＢＸＭＬ文書１００２では、圧縮スキーマ１００３において、指定したタグのテキスト要素として含まれる文字列の候補を辞書として持っておき、圧縮データ１００４において文字列を直接記述する代わりに、圧縮スキーマで保持している辞書データの識別子を持たせることができる。

【００４５】

テキスト要素を列挙させる場合において、例外的なテキスト要素が存在する場合がある。たとえば１００件のテキスト要素が存在し、その中の９０件は５種類の値のいずれかしか取らないが、残りの１０件はまったく異なる値を取る、という場合がある。指定したタグの全てのテキスト要素を辞書に登録し、その識別子でしか参照しない場合、このような例外的なテキスト要素を圧縮スキーマに登録するのは無駄である。通常のテキスト要素と辞書への識別子参照を混在することにより、この問題を解決することができる。

【００４６】

テキスト要素の種類がそのタグの総数に比べ十分に小さくない場合でも、そのテキスト要素の部分を切り出せば十分まとめることが可能な場合がある。この例の代表例としては、ＸＭＬの識別子（ＩＤ）が挙げられる。ＩＤが世界中で完全にユニークであるための仕掛けとして、Ｗ３ＣではＵＲＩ（Ｕｎｉｆｏｒｍ　Ｒｅｓｏｕｒｃｅ　Ｉｄｅｎｔｉｆｉｅｒ）という形式を提唱している。これはある組織がＩＤを決定する場合において、ＩＤを

その組織の保有するＵＲＬ（Ｕｎｉｆｏｒｍ　Ｒｅｓｏｕｒｃｅ　Ｌｏｃａｔｅｒ）にその組織内で固有のＩＤを組み合わせるという方式である。例えば、「ｈｔｔｐ：／／ｃｒｌ．ｈｉｔａｃｈｉ．ｃｏ．ｊｐ／ｇｉｓ／」というＵＲＬを持つ組織が「ｆ００００１」という固有ＩＤを持つ時、ＵＲＩは「ｈｔｔｐ：／／ｃｒｌ．ｈｉｔａｃｈｉ．ｃｏ．ｊｐ／ｇｉｓ／ｆ００００１」となる。この方式の欠点として、ＩＤが非常に長いものになるという点が挙げられる。例えば地図上の道路や家屋など、数千個の地物全てにＵＲＩ形式でＩＤを付加すると、ＩＤの大部分を占めるＵＲＬ部分が同じという冗長な構造でデータ量が肥大する、という問題がある。ここでＩＤをＵＲＬの部分と組織内固有ＩＤの部分に分割し、ＵＲＬの部分を辞書に登録することによりデータ量の圧縮が実現できる。また、ＢＸＭＬ内部のデータをＩＤをキーとして高速に検索する場合、一般に文字列比較よりも数値比較の方が高速なため、ＩＤは文字列ではなく、数値であるほうが都合がよい。このような場合もＵＲＬの部分を辞書化しておき、組織固有ＩＤを数値で管理しておけば、ＩＤの高速検索が実現できる。

【００４７】

本発明による「文字列の辞書化」では、テキスト要素を辞書化しないタグ、テキスト要素を全て辞書化するタグ、部分的に辞書化するタグが混在するという構造を取る。図１１に辞書化のパターンを示す。

【００４８】

パターン１「インライン文字列」では、辞書を用いず、入力のテキスト要素をそのまま記載する。圧縮データには、辞書に登録されたＩＤか文字列かを判定するため、「次のデータは文字列であり、終端する」ことを示すトークン「Ｔ＿ＷＯＲＤ」を記述し、その後文字列を記載する。

【００４９】

パターン２「辞書ＩＤ」では、テキスト要素を全て辞書に格納する例である。辞書に識別子と登録文字列のペアを格納しておき、圧縮データには、「次のデータは辞書に登録された文字列の識別子であり、終端する」ことを示すトークン「Ｔ＿ＩＤ」を記述し、その後識別子である数値「１」を記載する。

【００５０】

パターン３「辞書ＩＤ　＋　インライン文字列」では、最初の文字列「国道」を辞書に格納し、それ以降の文字列「２０号線」を圧縮データに記載する。それを表現するため、圧縮データには、「次のデータは辞書に登録された文字列の識別子であり、テキスト要素は継続する」ことを示すトークン「Ｃ＿ＩＤ」、辞書の識別子である数値「１」、「次のデータは文字列であり、終端する」ことを示すトークン「Ｔ＿ＷＯＲＤ」を記述し、その後文字列を記載する。

【００５１】

パターン４「辞書ＩＤ　＋　数値」では、「Ｔ＿ＷＯＲＤ」の代わりに「次のデータは数値であり、終端する」ことを示すトークン「Ｔ＿ＮＵＭ」を記述し、その後数値を記載する。

【００５２】

パターン５「インライン文字列　＋　辞書ＩＤ」では、「次のデータは文字列であり、テキスト要素は継続する」ことを示すトークン「Ｃ＿ＷＯＲＤ」、文字列、「次のデータは辞書に登録された文字列の識別子であり、終端する」ことを示すトークン「Ｔ＿ＩＤ」を記述し、その後識別子である数値「１」を記載する。

３番目の圧縮方式である、「データ型の適用」方式について説明する。本圧縮方式は、圧縮データ領域を対象とする。図４に示すように、「タグとデータの分離」方式では、テキスト要素しか持たないタグのスキーマ型を「文字列型」と設定していた。本圧縮方式では、従来の「文字列型」に加え、さらに７種類に拡張する。すなわち、「整数型」、「実数型」、「２次元整数型」、「２次元実数型」、「２次元整数配列型」、「２次元実数配列型」、「日付型」である。

整数型や実数型は、テキスト要素に整数や実数のみしか格納されていない場合に適用する

。「文字列型」に対応する圧縮データが、ＡＳＣＩＩやＳＨＩＦＴ－ＪＩＳ、ＵＮＩＣＯＤＥ等の文字コードで記載されているのに対し、たとえば計算機のメモリ上での整数格納形式や、不動小数点数格納形式（ＩＥＥＥ７５４）などの、より計算機に利用しやすい形で格納する。これは圧縮データのサイズ削減に貢献する。
例えば、整数３２７６７を文字列として格納する場合、「３」「２」「７」「６」「７」「￥０」の６つの文字となる。最後の￥０は文字列の終端を表すヌル文字である。ＡＳＣＩＩ文字セットでは文字は１ＢＹＴＥで表現されるため、整数３２７６７は６ＢＹＴＥの領域を占める。ＵＮＩＣＯＤＥ文字セットでは文字は２ＢＹＴＥで表現されるため、整数３２７６７は１２ＢＹＴＥの領域を占める。一方、Ｊａｖａ言語のｓｈｏｒｔ型では、数値３２７６７は０ｘ７Ｆ、０ｘＦＦの２ＢＹＴＥで表現される。同様に実数３．１４１５９２６は、「３」「．」「１」「４」「１」「５」「９」「２」「６」「￥０」となり、ＡＳＣＩＩ文字セットで１０ＢＹＴＥ、ＵＮＩＣＯＤＥ文字セットで２０ＢＹＴＥの領域を占めるが、Ｊａｖａ言語のｆｌｏａｔ型では４ＢＹＴＥで表現される。

【００５３】

また本方式は、ＸＭＬの解析処理速度を向上させるという効果もある。ＸＭＬ文書を利用するＸＭＬアプリケーションは、例えば数値の大小判定を行う、四則演算を行うなど、数値を利用する場合、数値を計算機のメモリ上で理解できる数値型で保持しておく必要がある。ＸＭＬでは数値は文字列として格納されているため、内部処理として文字列を数値型に変換する処理が必要となり、処理時間が必要となる。一方、本方式で圧縮したＢＸＭＬをそのまま解析することにより、文字列・数値変換が不要となるため、処理速度が向上する。
例えば地図をあらわしたＸＭＬ文書は、その大部分が図形を示す点・折れ線・ポリゴンの座標となる。データの大部分を占める数値情報を文字列ではなく数値として管理することにより、大きなデータ量削減効果が得られる。また用途として、地図を示すＸＭＬ文書全体を解析して全ての座標を取得して画面上に描画する、ＸＭＬ文書全体を解析して、指定した矩形領域内に存在する地物を抽出する、などの処理を行うにあたり、数値情報を文字列ではなく数値として管理することにより、大きな処理速度向上効果が得られる。
タグのスキーマ型として、整数型、実数型について説明したが、残る「２次元整数型」、「２次元実数型」、「２次元整数配列型」、「２次元実数配列型」「日付型」について説明する。
「２次元整数型」は二つの整数の組、「２次元実数型」とは二つの実数の組を表現する構造である。これは、圧縮元となるＸＭＬ文書において、例えば＜座標＞ｘ、ｙ＜／座標＞のように、数値がテキスト要素として分離していない場合に利用する。上記のような表記のＸＭＬ文書は、「整数型」「実数型」では表現できない。これを文字列型として表現するとデータサイズ、解析速度が悪化する。このような構造は座標を管理する地図用ＸＭＬでは頻出する可能性があるため、特別な型として定義する。これは圧縮スキーマにおいて「２次元整数型」あるいは「２次元実数型」と指定し、付加情報として二つの数値を分離するセパレータ文字（空白やカンマ文字）を定義し、圧縮データにおいて数値を連続して並べることにより実現できる。２次元整数、２次元実数は、他にも地物の生成時間と消滅時間のような時間的な期間を表すなど、様々なデータに利用できる。同様に、立体図形を表現するための座標（ｘ、ｙ、ｚ）、あるいは空間上の移動量・移動方向を表すベクトル（ｘ、ｙ、ｚ）を表現する「３次元整数型」、「３次元実数型」への拡張も可能である。
「２次元整数配列型」は２次元整数型の０個以上の組、「２次元実数配列型」は２次元実数型の０個以上の組を表す。これは例えば＜座標列＞ｘ１、ｙ１　ｘ２、ｙ２　ｘ３、ｙ３＜／座標列＞のように、数値がテキスト要素として分離していない場合に利用する。このような構造は座標を管理する地図用ＸＭＬでは頻出する可能性があるため、特別な型として定義する。これは圧縮スキーマにおいて「２次元整数配列型」あるいは「２次元実数配列型」と指定し、付加情報とし座標ｘ、ｙを分離するセパレータ文字、座標と座標を分離するセパレータ文字（上記の例ではカンマと空白）を定義し、圧縮データにおいて最初に配列の個数を、その後に数値を連続して並べることにより実現できる。これは同様に「

３次元整数配列型」、「３次元実数配列型」への拡張も可能であり、＜数値列＞ａ１、ａ２、ａ３、ａ４　＜／数値列＞のように整数や実数の配列である「１次元整数配列型」、「１次元実数配列型」も可能である。

【００５４】

「日付型」は日付を指定するために利用する。Ｗ３ＣではＷ３Ｃ－ＤＴＦとして、またＩＳＯ８６０１で日付の書式が定義されており、例えば２００２年１２月１日０時０分０秒を、「２００２－１２－０１Ｔ００：００：００」という文字列で表現する。一方、計算機内部で広く使われる日付の管理方式として「１９７０年１月１日午前０時（ＵＴＣ）を起算点とする通算の秒数」で日付を管理する場合がある。この方式は日付のデータに４ｂｙｔｅしか利用せず、また日付同士の比較、日付と期間の加算が容易であるというという特徴を持つ。では、圧縮スキーマにおいて「日付型」と指定し、圧縮データにおいてこの通算秒などよりコンパクトな日付の格納を行うことにより実現できる。

４番目の圧縮方式である、「可変長数値の適用」方式について説明する。本圧縮方式は、圧縮データ領域を対象とする。本圧縮方式では、整数と実数を、本発明で独自に定義した、ＭＢＩ（Ｍｕｌｔｉ　Ｂｙｔｅ　Ｉｎｔｅｇｅｒ：可変長整数）、　ＭＢＦ（Ｍｕｌｔｉ　Ｂｙｔｅ　Ｆｌｏａｔ：可変長実数）で表現する。

従来の計算機内で利用されている整数・実数の管理方式は、ｃｈａｒ型整数は１ｂｙｔｅ、ｓｈｏｒｔ型整数は２ｂｙｔｅ、ｌｏｎｇ型整数は４ｂｙｔｅ、ｆｌｏａｔ型実数は４ｂｙｔｅ、ｄｏｕｂｌｅ型実数は８ｂｙｔｅと固定されている、固定長数値である。本方式の可変長数値では、例えば数値１０は１ｂｙｔｅ、１００は２ｂｙｔｅ、１０、０００は３ｂｙｔｅ、１、０００、０００は４ｂｙｔｅと可変長で管理できる。

【００５５】

固定長数値の欠点は、表現できる桁数が限られるという点にある。例えばｃｈａｒ型整数は１２７まで、ｓｈｏｒｔ型整数は３２、７６７まで、ｌｏｎｇ型整数は２、１４７、４８３、６４６までしか表現できない。ｆｌｏａｔ型実数は仮数部の精度７桁、ｄｏｕｂｌｅ型実数は精度１５桁までしか表現できない。ＸＭＬ文書を「データ型の適用」を用いて圧縮する時、固定長数値を利用する場合、どの数値型を選択する必要があるかを判定する必要がある。全ての整数をｌｏｎｇ型の４ｂｙｔｅで管理するとデータ圧縮上効率が悪い。利用者が出現する整数の取りうる範囲を考慮して適切な数値型を割り当てるのは手間がかかり、また汎用的ではない。また例外的に極端に大きな数値が出現した時に破綻する可能性がある。本発明の可変長数値を利用することにより、小さな整数や精度の低い実数は必要最低限の領域で格納でき、またメモリ領域やファイルシステム領域の制限の許す限り大きな整数、精度の高い実数を格納することができる。

【００５６】

図１２に本発明の可変長整数であるＭＢＩの構造を示す。ＭＢＩは、Ｗ３Ｃの標準規格の一つであるＷＡＰ　Ｂｉｎａｒｙ　ＸＭＬで定義されるＭＢＩ（以降ＷＡＰ－ＭＢＩと呼ぶ）を、負数が表現できるように拡張したものである。

【００５７】

図１２（ａ）は、ＷＡＰ－ＭＢＩにおける、整数を表現する複数のバイト列の、一つのバイトに着目した構造である。最上位ビットは継続フラグであり、１の場合は後続バイトにデータが続くことを示し、０の場合はそのバイトでデータが終端することを示す。残りの７ｂｉｔに数値の情報が格納される。数値の情報を７ｂｉｔづつ区切り、格納する。

【００５８】

図１２（ｂ）は、本発明によるＭＢＩであり、ＷＡＰ－ＭＢＩを、負数が表現できるように拡張したものである。

最上位ビットは継続フラグというのはＷＡＰ－ＭＢＩと同様である。先頭バイトは例外的にデータ領域が５ｂｉｔであり、継続フラグとデータ領域の間に余分の２ｂｉｔが存在する。最上位ビットの次のビットは常に０が格納され、ＭＢＩであることを識別するために存在する。次のビットは符号フラグであり、０または正数の場合は０、負数の場合は１が格納される。データ領域は正数の絶対値が２進数表現で格納される。

【００５９】
本発明の可変長実数であるＭＢＦは、実数を仮数部と指数部の二つの整数に分割し、それぞれ二つのＭＢＩとして連続して格納することにより表現する。ＭＢＦと二つの連続したＭＢＩをそれ自体で区別する方法はない。圧縮スキーマ部でスキーマ型が「整数型」の場合、圧縮データから１つのＭＢＩを取得して、これを整数として認識する。スキーマ型が「実数型」の場合、圧縮データから連続する２つのＭＢＩを取得して、これをＭＢＦとして認識する、という方式で区別を行う。「２次元整数型」、「２次元実数型」、「２次元整数配列型」、「２次元実数配列型」等の型も同様な処理を行う。
【００６０】
５番目の圧縮方式である、「相対値管理」について説明する。本圧縮方式は、圧縮データ領域を対象とする。
特に地図図形の座標を多数ＸＭＬ文書で管理する場合に適用され、座標をある起点からの相対値で表現する方式である。
地図上の地物の位置や形状を一意に表現するためには、緯度経度などの絶対座標が用いられる。しかし地図上の多数の座標を全て絶対座標で表現すると、データ量が増大する。ある領域内に限定した地図をみると、その上位桁はほとんど同じで、下位桁のみ変化しているという冗長性のある構造を取っている。
従来の地図管理フォーマットでは、図面という概念を導入して、座標を図面原点からの相対座標で管理するという方式を取っている。この目的は、（１）座標格納領域のデータ量を削減する、（２）座標をｓｈｏｒｔ型など比較的小さいサイズの固定長数値で表現できるようにする、という二点である。図面管理方式の欠点として、座標を固定長数値で表現するため、図面のサイズがある程度固定されてしまう点にある。図面として規定されたサイズを超える図形や、図面をまたがる図形は、複数の図形断片に分断されてしまうことになる。
文字列ベースであるＸＭＬでは固定長数値の制約はないため、自由に大きな桁数の座標が記述できる。そこでＸＭＬをベースにした空間データフォーマットは、座標を緯経度などの絶対座標で格納し、図面という制約を仕様から外しているものが多い。ただしこれがデータサイズ増大の原因となっている。
【００６１】
本発明の圧縮方法の一つである「相対値管理」は、このような絶対座標で記述された空間データをＢＸＭＬ内部で図面と同様の相対座標管理を行うことができる。相対座標管理はＢＸＭＬの内部構造で行っており、空間データフォーマット上では相対座標であることが隠蔽されている。これにより、従来の図面管理方式と同様に、座標情報のデータ量削減が実現できる。また、４番目の圧縮方法の「可変長数値の適用」で示したとおり、本発明のデータ構造は固定長数値の制約が存在しないため、相対座標の原点から極端に外れている座標も問題なく管理できる。
【００６２】
相対値管理は、圧縮スキーマ部において、特定のタグに起点となる数値を記述し、圧縮データ部においてその起点数値からの相対座標を記述することにより実現できる。
【００６３】
次に、図１３を用いて、本発明によるＢＸＭＬ文書に関連し、本発明が提供する機能について説明する。本発明が提供する機能は、圧縮処理１３０１、展開処理１３０２、解析処理１３０３、更新処理１３０４、作成処理１３０５に分類される。
【００６４】
圧縮処理１３０１は、ＸＭＬ文書１３０６を読み込み、これをバイナリ構造に圧縮したＢＸＭＬ文書１３０７に変換する。展開処理１３０２は、ＢＸＭＬ文書１３０７をＸＭＬ文書１３０６に展開する。
【００６５】
解析処理１３０３は、ＢＸＭＬ文書１３０９のタグ構造の移動と、ＢＸＭＬ文書の任意の位置での情報取得機能を、ユーザの作成するプログラム１３０８に提供する。解析処理１

３０３はＸＭＬ文書の解析のための標準インタフェースであるＳＡＸとＤＯＭに準拠したインタフェースをユーザプログラム１３０８に提供する。

【００６６】

更新処理１３０４は、ＢＸＭＬ文書の任意の位置でのデータの更新機能を、ユーザプログラム１３０８に提供する。更新処理１３０４はＸＭＬ文書の更新のための標準インタフェースであるＤＯＭに準拠したインタフェースをユーザプログラム１３０８に提供する。

【００６７】

作成処理１３０５は、ＢＸＭＬ文書１３１１を直接作成する機能をユーザプログラム１３１０に提供する。ユーザプログラム１３１０は、ＸＭＬ文書を作成することなく、直接ＢＸＭＬ文書１３１１を作成することができる。

図１に、図１３の機能ブロックを実現するためのシステムブロック図を示す。ここで、矩形はそれぞれの機能を実現する各処理ブロックを示す。また破線で構成される矩形は、ユーザが作成するプログラムを示す。角の取れた矩形はデータを示す。また矢印の方向は、矢印の始点から終点方向にデータが参照され、あるいはデータが変換されることを示す。

最初に、図１に記載されているデータ構造群について説明する。ＸＭＬ文書１０１は本発明が圧縮対象とするＸＭＬ文書であり、ＢＸＭＬ文書１０５は本発明によりバイナリ構造に圧縮されたＸＭＬ文書である。

ＸＭＬのスキーマ定義１０６は、ＸＭＬの構造を定義した文書である。一般に、ＸＭＬ文書の構造を定義する標準的な形式として、ＤＴＤ（Ｄｏｃｕｍｅｎｔ Ｔｙｐｅ Ｄｅｆｉｎｉｔｉｏｎ）、ＸＭＬ Ｓｃｈｅｍａ、ＲＥＬＡＸ ＮＧなどが規定されているが、スキーマ定義１０６はこれらの形式の文書に相当する。これは圧縮処理１３０２や生成処理１３０５に利用される場合がある。

圧縮スキーマ１０３および圧縮データ１０４は、本発明の処理で一時的に利用されるデータであり、この二つを組み合わせることによりＢＸＭＬ文書１０５が構成される。圧縮スキーマ１０３にはＸＭＬ文書のタグ名やタグ間の階層関係など、ＸＭＬ文書全体で共通に利用される情報が、圧縮データ１０４にはテキスト要素や属性値、重複数など、データ固有の情報が格納される。

スキーマ木１０２は圧縮スキーマ１０３を計算機のメモリ空間に計算機に処理しやすいように、木状に展開したものである。

図１４にスキーマ木の構造を示す。スキーマ木は、ＸＭＬ文書のルート要素を根とする木構造を取り、木の要素としてＸＭＬのタグを表すスキーマ要素１４０１を持つ。重複するタグは同じスキーマ要素となる。スキーマ要素に含まれる変数は、図８に示す圧縮スキーマ８０４に記述される情報とほぼ同じであり、スキーマ要素識別子１４０４、タグ名１４０５、重複フラグ１４０６、スキーマ型１４０７、属性保有数１４０８、子要素保有数１４０９を持つ。また、各子要素を示すスキーマ要素には、０個以上の属性を示すスキーマ要素１４０２が接続し、また０個以上の子要素を示すスキーマ要素１４０３が接続する。これを実現するため、各スキーマ要素には、属性を表すスキーマ要素１４０２への０個以上のポインタ、子要素を表すスキーマ要素１４０３への０個以上のポインタ、親要素を表すスキーマ要素への１個のポインタを保持する。

【００６８】

図８の圧縮スキーマ８０４に存在しない新たな情報としてスキーマ要素識別子１４０４がある。これは、スキーマ要素を識別するための情報であり、例えば数値で記載される。この識別子の決定方法は様々な方法が考えられるが、例えばスキーマ木で出現する全てのタグ、属性を示すスキーマ要素群の出現順の通し番号としてもよい。

属性を示すスキーマ要素１４０２の構造は、タグを表すスキーマ要素１４０１と同じ構造を持つ。属性のキー名１４１１はタグ名１４０５で表現する。スキーマ要素識別子１４１０、重複フラグ１４１２、スキーマ型１４１３はスキーマ要素１４０１と同じである。属性に付属する属性は存在せず、また属性は階層構造を持たないため、属性を示すスキーマ要素１４０２には属性保有数１４０８、子要素保有数１４０９は存在しない。同様に、属性を示すスキーマ要素には、属性を表すスキーマ要素へのポインタ、子要素を表すスキー

マ要素へポインタは持たず、親要素を表すスキーマ要素への１個のポインタのみを保持する。
図１５を用いてスキーマ木について、より詳細に説明する。図３において、従来のＤＯＭで使用される木構造であるインスタンス木３０２について説明したが、スキーマ木は、インスタンス木と違い、重複したタグそれぞれについてのノードを持たない。重複したタグのスキーマ構造は全て同じであり、これを持つことは冗長であるためである。例えば図１５の最初の例１５０１では、タグＲの中にタグＡが３個、タグＢが２個格納されている。インスタンス木ではそれぞれのタグについて全て木の構成要素を持つが、スキーマ木ではタグＡ、タグＢの種類につき一つだけの構成要素しか持たない。図１５の２番目の例１５０２では、タグＲの中にタグＡが３個存在し、最初のタグＡにはタグＢが、次のタグＡにはタグＣが、３番目のタグＡにはタグＤが格納されている。インスタンス木ではそれをそのまま格納しているが、スキーマ木ではタグＡの下にタグＢ、タグＣ、タグＤが格納される、という構造を取る。これによりタグＡのスキーマ構造を重複せず持つことができる。図１５の３番目の例１５０３では、タグＣがタグＡ、タグＢの両方の下に出現している。これを表現するスキーマ木としては、スキーマ木１５０４、スキーマ木１５０５の２通りの構造が取れる。スキーマ木１５０４の方がタグＣのスキーマ構造を重複せず持つことができるので効率的である。ただしスキーマ木構造にループができてしまうため、圧縮方式が複雑になり、圧縮処理やスキーマ構造の変更処理における計算速度が低下する可能性がある。そのため、スキーマ木を表現する圧縮スキーマのサイズが圧縮データのサイズと比較して無視できるほど十分に小さい場合、冗長であるが簡単なスキーマ木１５０５の構造を取っても構わない。

【００６９】

次に、図１のシステムブロック図を用いて、図１３の機能ブロック図の各機能を実現する手段について説明する。図１３の機能ブロック図に示す各機能は、図１のシステムブロック図に示す各処理ブロックの組合せにより実現される。見易さのため、図１のシステムブロック図を、図１３の機能ブロック図に示す５つの機能に合わせて抜粋した図を、図１６、図１７、図１８、図１９、図２０に示す。

【００７０】

図１６は、図１３の圧縮処理１３０１を実現する処理ブロック群を、図１から抜粋したブロック抜粋図である。図１６を用いてＸＭＬからＢＸＭＬへの圧縮の処理フローについて説明する。ＸＭＬのＢＸＭＬへの圧縮は、３種類の方法がある。すなわち、（１）ＸＭＬ文書のみを利用した圧縮、（２）スキーマ定義を利用したＸＭＬ文書の圧縮、（３）ＢＸＭＬ文書を利用したＸＭＬ文書の圧縮である。
（１）「ＸＭＬ文書のみを利用した圧縮」では、ＸＭＬ文書をＢＸＭＬ文書に圧縮する際、圧縮対象のＸＭＬ文書以外の情報を利用しないで行う圧縮方法である。最初にＸＭＬ文書１０１全体をＸＭＬパースブロック１０７によりパースし、スキーマ生成ブロック１０９により、スキーマ木１０２を作成する。そして再度ＸＭＬ文書１０１全体をＸＭＬパースブロック１０７によりパースし、データ圧縮ブロック１１２により既に作成したスキーマ木１０２を利用しながら圧縮データ１０４を作成する。次にスキーマ圧縮ブロック１１０によりスキーマ木１０２を圧縮スキーマ１０３に圧縮し、ＢＸＭＬ結合ブロック１１４により圧縮スキーマ１０３と圧縮データ１０４を結合し、最終成果であるＢＸＭＬ文書１０５を作成する。
（２）「スキーマ定義を利用した圧縮」では、処理時間のかかるスキーマ生成ブロック１０９を省略することにより圧縮時間を短縮する方式である。ＸＭＬ　Ｓｃｈｅｍａなどの ＸＭＬスキーマ定義文書１０６から直接スキーマ木１０２を作成することによりスキーマ生成ブロック１０９を省略することができる。スキーマ変換ブロック１１６により、ＸＭＬのスキーマ定義１０６からスキーマ木１０２を作成する。後は「ＸＭＬ文書のみを利用した圧縮」で説明したのと同様な流れでＢＸＭＬ文書１０５を作成する。
（３）「ＢＸＭＬを利用した圧縮」も上記と同様に、処理時間のかかるスキーマ生成ブロック１０９を省略することにより圧縮時間を短縮する方式である。同じ構造のＸＭＬ文書

を繰り返し圧縮する場合に、スキーマ木を作成する方法として、以前に作成した同じスキーマ構造であることが分かっているＢＸＭＬからスキーマ木を取得する。
図１において、あらかじめ作成してあるＢＸＭＬ文書１０５を用意し、ＢＸＭＬ分解ブロック１１５によりＢＸＭＬ文書１０５から圧縮スキーマ１０３を取り出す。圧縮スキーマ１０３をスキーマ展開ブロック１１１により変換することでスキーマ木１０２を得る。あとは「ＸＭＬ文書のみを利用した圧縮」で説明したのと同様な流れでＢＸＭＬ文書１０５を作成する。上記説明において、ＢＸＭＬ文書１０５ではなく圧縮スキーマ１０３をあらかじめ保持しておくこともできる。
図１７は、図１３の展開処理１３０２を実現する処理ブロック群を、図１から抜粋したブロック抜粋図である。

【００７１】
ＢＸＭＬからＸＭＬへの展開は、２種類の方法がある。すなわち、（１）ＢＸＭＬ文書の展開、（２）既存圧縮スキーマを利用した圧縮データの展開、である。
（１）「ＢＸＭＬ文書の展開」では、ＢＸＭＬ文書１０５から圧縮スキーマ１０３と圧縮データ１０４を取り出し、それを用いてＸＭＬ文書に展開する。ＢＸＭＬ分解ブロック１１５によりＢＸＭＬ文書１０５から圧縮スキーマ１０３、圧縮データ１０４を取得し、スキーマ展開ブロック１１１により圧縮スキーマ１０３からスキーマ木１０２を取得する。そしてＢＸＭＬパースブロック１０８でスキーマ木１０２を利用して圧縮データ１０４をパースし、データ展開ブロック１１３によりＸＭＬ文書１０１を取得する。
（２）「既存圧縮スキーマを利用した圧縮データの展開」では、同じ構造のＢＸＭＬ文書を繰り返し展開する時に利用する事ができる方式である。同じ構造のＢＸＭＬ文書１０５が多数存在する場合、その圧縮スキーマ１０３は全て共通であり、従って圧縮スキーマ１０３の部分が冗長である。そこで、複数のＢＸＭＬ文書１０５を管理するのではなく、単独の圧縮スキーマ１０３と複数の圧縮データ１０４を管理する方が、データサイズの削減となり、その結果、データ管理領域の削減やデータ転送速度の向上が実現できる。例えば、クライアント・サーバシステムにおいて、あらかじめクライアントに共通の圧縮スキーマ１０３を送信しておく。そしてクライアントの要求に応じて、圧縮データ１０４を複数回クライアントに送信する。クライアントではあらかじめ、取得した圧縮スキーマ１０３をスキーマ展開ブロック１１１により圧縮スキーマ１０３をスキーマ木１０２に展開しておき、圧縮データ１０４が繰り返し送信されてきた時に、あらかじめ取得しておいたスキーマ木１０２を用いて繰り返し送付される圧縮データ１０４をＢＸＭＬパース１０８でパースし、データ展開ブロック１１３により、ＸＭＬ文書１０１を取得する。
図１８は、図１３の解析処理１３０３を実現する処理ブロック群を、図１から抜粋したブロック抜粋図である。

【００７２】
一般に、ＸＭＬ文書の解析を行うための標準インタフェースとして、ＳＡＸ、ＤＯＭが規定されている。解析処理１３０３では、ＢＸＭＬ文書を解析するインタフェースとして標準インタフェースであるＳＡＸ、ＤＯＭを提供する。これにより、ユーザはＸＭＬ文書を解析すると同様の方式でＢＸＭＬ文書を解析することができる。

【００７３】
ＢＸＭＬ文書１０５の解析に先立ち、まずＢＸＭＬ分解ブロック１１５によりＢＸＭＬ文書１０５から圧縮スキーマ１０３、圧縮データ１０４を取得し、スキーマ展開ブロック１１１により圧縮スキーマ１０３からスキーマ木１０２を取得しておく。そしてＢＸＭＬパースブロック１０８でスキーマ木１０２を利用して圧縮データ１０４をパースする。

【００７４】
ユーザプログラム１２２が標準インタフェースであるＳＡＸでＢＸＭＬ文書１０５を解析する場合、ＢＸＭＬパースブロック１０８はＳＡＸパースブロック１２３を呼び出し、ＳＡＸパースブロック１２３がユーザプログラム１２２に対しＸＭＬを構成する部品、すなわち開始タグ、終了タグ、テキスト要素、属性の情報などを提供する。

【００７５】

ユーザプログラム１２５が標準インタフェースであるＤＯＭでＢＸＭＬ文書１０５を解析する場合、ＤＯＭパースブロック１２４はＢＸＭＬパースブロック１０８を利用してＸＭＬ文書のノード間の移動を行い、また任意のノードにおけるタグ情報、テキスト要素、属性情報をユーザプログラム１２５に提供する。
図１９は、図１３の更新処理１３０４を実現する処理ブロック群を、図１から抜粋したブロック抜粋図である。

【００７６】
ＢＸＭＬ文書１０５の更新に先立ち、まずＢＸＭＬ分解ブロック１１５によりＢＸＭＬ文書１０５から圧縮スキーマ１０３、圧縮データ１０４を取得し、スキーマ展開ブロック１１１により圧縮スキーマ１０３からスキーマ木１０２を取得しておく。そしてＢＸＭＬパースブロック１０８でスキーマ木１０２を利用して圧縮データ１０４をパースする。

【００７７】
更新処理９０４では、ＢＸＭＬ文書１０５の更新機能として３つの機能を提供する。すなわち、（１）インスタンス変更機能、（２）部分ＢＸＭＬ文書抽出機能、（３）スキーマ変更機能である。

【００７８】
（１）インスタンス変更機能では、ＢＸＭＬ文書１０５の任意の位置に存在するテキスト要素や属性値の追加・削除・置換、任意のノードの挿入・削除・置換を行う。ユーザがテキスト要素や属性、ノードを指定する方法として、前に説明した解析機能１３０３のＤＯＭインタフェースを用いる。ＤＯＭパースブロック１２４はＢＸＭＬパースブロック１０８を利用してＸＭＬ文書のノード間の移動を行う。ＤＯＭ更新ブロックはユーザのテキスト要素、属性情報の追加・削除・置換、ノードの挿入・削除・置換処理を受け付け、圧縮データ１０４に書き込む。そして更新した圧縮データ１０４をＢＸＭＬ結合ブロック１１４により圧縮スキーマ１０３と結合し、ＢＸＭＬ文書１０５を出力する。

【００７９】
（２）部分ＢＸＭＬ文書抽出機能では、ＢＸＭＬ文書１０５のユーザの指定した任意のノードにおいて、そのノード以下の部分ＢＸＭＬ文書を抽出し出力する。ＤＯＭパースブロック１２４はＢＸＭＬパースブロック１０８を利用してユーザが抽出したいノードへの移動を行う。ＤＯＭ更新ブロックは圧縮データの現在参照している位置と、現在参照しているスキーマ木の位置とを利用して、　ＢＸＭＬ結合ブロック１１４を実行することにより、部分ＢＸＭＬ文書１０５を取得する。

【００８０】
（３）スキーマ変更機能は、スキーマ木を変更することにより、ＢＸＭＬ文書１０５のユーザが指定したタグ名称を一括変更する機能、ユーザが指定したノード以下の部分ＸＭＬ文書を一括削除する機能などを提供する。スキーマパースブロック１１７は、スキーマ木に対してユーザが変更したいスキーマ要素へ移動する機能を提供する。スキーマ更新ブロック１１９は、ユーザのタグ名一括変更、ノード一括削除要求を受付け、スキーマ木の変更を行う。スキーマ圧縮ブロック１１０により変更されたスキーマ木１０２を圧縮スキーマ１０３に変換し、ＢＸＭＬ結合ブロック１１４により圧縮データと結合し、ＢＸＭＬ文書１０５を出力する。

【００８１】
図２０は、図１３に示すＢＸＭＬ文書作成処理１３０５を実現する処理ブロック群を、図１から抜粋したブロック抜粋図である。生成処理１３０５ではＢＸＭＬ文書を新規に作成する機能を提供する。

【００８２】
ＸＭＬ文書を新規に作成する従来方法としては、（１）専用のプログラムを作成して、テキストのＸＭＬ文書を直接記述する、（２）標準のＤＯＭでＸＭＬ文書を作成し、完成したインスタンス木からＸＭＬ文書を生成する、という２通りの方法が存在する。しかし最初の方法では、ユーザプログラムのミスで開始タグと終了タグの対応付けが正しく記述されないというエラーが、ＸＭＬ文書が完成して整合性の確認を行うまで発見できない。ま

たテキストベースのＸＭＬ文書を直接作成するため、巨大なＸＭＬ文書を作成するには大量の処理時間とメモリ空間を必要とする。２番目の方法は、従来のＤＯＭパーサでは、作成するインスタンス木は目的のＸＭＬ文書より数倍のサイズのメモリ空間を必要とするため、巨大なＸＭＬ文書を作成するには大量のメモリ空間を必要とする。本発明の作成処理１３０５は、ＸＭＬ文書より小さなサイズのＢＸＭＬ文書を直接作成すること、従来のＤＯＭパーサの作成するインスタンス木のような大量のメモリ空間を必要とする中間生成物を必要としないことにより、従来の上記２方法と比較し占有メモリ空間が少なく、かつ高速にＢＸＭＬ文書を生成することができる。

【００８３】

ＢＸＭＬ文書を新規に作成する手順は、（１）スキーマ木１０２の作成、（２）圧縮データ１０４の作成、の２手順で行う。この二つのデータが完成すると、圧縮処理１３０１で説明したとおり、スキーマ圧縮ブロック１１０、ＢＸＭＬ結合ブロック１１４を経由して最終結果であるＢＸＭＬ文書１０５を作成することができる。

（１）スキーマ木１０２の作成方法は、次に示す４つの方法が存在する。（１）作成対象のＢＸＭＬ文書と同じ構成のＸＭＬ文書やテンプレート１０１から、スキーマ生成ブロック１０９を経てスキーマ木１０２を作成する。（２）　ＸＭＬ　Ｓｃｈｅｍａなどのスキーマ定義文書１０６からスキーマ変換ブロック１１６を経てスキーマ木１０２を作成する。（３）あらかじめ用意しておいたＢＸＭＬ文書１０５や圧縮スキーマ１０３からスキーマ展開ブロック１１１を経てスキーマ木を作成する。（４）スキーマパースブロック１１７、スキーマ更新ブロック１１９を用いてスキーマ木を最初から構築する。また、これら４つの方法のいずれかで作成したスキーマ木は、スキーマパースブロック１１７、スキーマ更新ブロック１１９を用いたユーザプログラム１１８でスキーマ要素の追加・置換・削除を行ったり、別のＢＸＭＬ文書が持つスキーマ木の一部を挿入したりすることにより更新することもできる。

（２）圧縮データ１０４の作成は、圧縮行程１３０１で説明したデータ圧縮ブロック１１３をそのまま利用する。圧縮処理１３０１で説明した通り、データ圧縮ブロック１１３はＸＭＬパースブロック１０７からＳＡＸインタフェースで呼び出される。すなわち、ＸＭＬパースブロック１０７はＸＭＬ文書を走査し、「開始タグ」、「終了タグ」、「属性」、「テキスト要素」などのイベントをデータ圧縮ブロック１１３に発行し、データ圧縮ブロック１１３はそれに対応して圧縮データ１１２を作成していく。

図２０のＢＸＭＬ作成ブロック１２１はＸＭＬパースブロック１０７と同様に、ＳＡＸインタフェースを用いてデータ圧縮ブロック１１３を呼び出す。ユーザプログラム１２２が、「開始タグ」、「終了タグ」、「属性」、「テキスト要素」などのＳＡＸイベントを直接データ圧縮ブロック１１３に発行すると圧縮データ１０４を作成することができるが、これはユーザにとって使いにくい。そのためＢＸＭＬ作成ブロックは、ユーザプログラム１２２に対し、「親に移動」「子に移動」「弟に移動」「テキスト要素の挿入」などのインタフェースを提供し、これをＳＡＸイベントに変換してデータ圧縮ブロック１１３を呼び出す、という方式をとる。また、「テキスト要素の挿入」のインタフェースの派生として、「整数の挿入」「実数の挿入」など、図７の「データ型の適用」圧縮７０２で説明した複数のデータ型を直接挿入するインタフェースを提供することにより、ユーザプログラム１２２が持つデータ型をＸＭＬの文字列に変換し、ＢＸＭＬ文書の内部構造へ変換する、という手間が省け、その結果処理速度を向上させることができる。

【００８４】

ＢＸＭＬ作成ブロック１２１のもう一つの特長として、他のＢＸＭＬ文書の指定したノード以下の部分集合を一括挿入することができる、という点がある。挿入元のＢＸＭＬ文書の圧縮データを一括して挿入先の圧縮データにコピーすることにより、タグ単位で挿入する方式より高速な挿入が実現できる。またこの機能を利用して、複数のＢＸＭＬ文書の高速な結合処理が実現できる。

本発明の実施形態として、図１に示す本発明を構成するシステムブロック図の各処理ブロックについて詳細に説明する。その後に、本発明の実施例として、本発明を利用したアプ

リケーションについて説明する。
最初に、図１に示す本発明を構成するシステムブロック図の各処理ブロックについて、（１）ＸＭＬパースブロック１０７、（２）スキーマ生成ブロック１０９、（３）スキーマ変換ブロック１１６、（４）データ圧縮ブロック１１２、（５）スキーマ圧縮ブロック１１４、（６）スキーマ展開ブロック１１１、（７）ＢＸＭＬ結合ブロック１１４、（８）ＢＸＭＬ分解ブロック１１５、（９）ＢＸＭＬパースブロック１０８、（１０）データ展開ブロック１１３、（１１）ＳＡＸパースブロック１２３、（１２）ＤＯＭパースブロック１２４、（１３）ＤＯＭ更新ブロック１２６、（１４）ＢＸＭＬ作成ブロック１２１、（１５）スキーマパースブロック１１７、（１６）スキーマ更新ブロック１１７、の順番で説明を行う。
最初に、図１のＸＭＬパースブロック１０７について詳細に説明する。
ＸＭＬパースブロック１０７の目的は、入力で与えられたＸＭＬ文書１０１の初めから最後までを走査し、外部のプログラムに対しＸＭＬ文書構造やテキスト要素の情報を提供することにある。ＸＭＬパースブロック１０７は、図２で説明した、標準のＸＭＬ解析インタフェースであるＳＡＸを利用することにより実現する。
【００８５】
次に、図１のスキーマ生成ブロック１０９について詳細に説明する。
【００８６】
図１のスキーマ生成ブロック１０９の目的は、ＸＭＬ文書１０１からスキーマ木１０２を作成することである。前述の６つの文書圧縮方式、すなわち（１）タグとデータの分離、（２）文字列の辞書化、（３）データ型の適用、（４）可変長数値の適用、（５）相対値の適用、に必要な情報を同時に作成し、スキーマ木１０１に登録する。
「タグとデータの分離」を行うため、図２１に示すアルゴリズムを用いて処理を行う。
［ＳＴＥＰ　１］　最初にＸＭＬ文書をメモリ空間に読み込む。またスキーマ木のルートスキーマ要素を設定する。ここでルートスキーマ要素とは、ＸＭＬ文書のルート要素ではなく、その親の位置にある仮想的なものである。図１４に示すように、スキーマ要素には属性を示すスキーマ要素が０個以上、子タグを示すスキーマ要素が０個以上接続される構造を取るが、初期状態ではいずれも０個とする。また各スキーマ要素には、処理のための一時的な変数として、「期待タグ」として、次に来ると期待される子タグのスキーマ要素を指すポインタ、「期待キー」として、次に来ると期待される属性キーのスキーマ要素を指すポインタを持つ。またスキーマ生成部が保持する一時変数「参照要素」として、スキーマ木の現在作成中のスキーマ要素を記憶しておく。参照要素の初期状態は、ルートスキーマ要素を指している。
［ＳＴＥＰ　２］　文書終端に達するまで、［ＳＴＥＰ　３］以降を繰り返す。
［ＳＴＥＰ　３］　ＸＭＬ文書の部品の一つを取得する。ここでＸＭＬ文書の部品とは、開始タグ、終了タグ、テキスト要素、属性である。これは標準のＳＡＸでの動作と同様である。
［ＳＴＥＰ　４］　取得した部品である開始タグ、終了タグ、テキスト要素、属性に対し、それぞれ処理を分岐させる。ここで分岐する処理は開始タグ、終了タグ、属性である。実際はテキスト要素に対しても処理を行うが、これは「タグとデータの分離」の圧縮のためではなく、その他のオプションの圧縮方針を実現するために利用するため、ここでは割愛する。
［ＳＴＥＰ　５］　部品が開始タグである場合、本処理を実行する。以降、参照要素の期待タグが指すスキーマ要素のことを期待要素と命名する。期待要素のタグ名称と、今回取得した開始タグのタグ名称とを比較し、同一であればＳＴＥＰ　６を実行する。それ以外の場合、参照要素の期待タグを、子タグ一覧の次に指すタグにして、ＳＴＥＰ　５の先頭に戻る。子タグ一覧が空である場合、または子タグ一覧に継続する要素が存在しない場合、ＳＴＥＰ　８を実行する。
［ＳＴＥＰ　６］　取得した開始タグが期待要素のタグ名称と一致している場合、本処理を実行する。期待要素の重複数を１増加させる。重複数は初期設定時点では０となってい

る。この時点で重複数が２以上になっている場合、その期待要素は図４の重複要素パターンであることが確定されるため、重複フラグを「重複」に設定する。
［ＳＴＥＰ　７］　参照要素を現在の参照要素の期待要素に移動する。
［ＳＴＥＰ　８］　取得した開始タグが期待要素のタグ名称と一致していない場合、あるいは期待要素が存在しない場合、本処理を実行する。新しいスキーマ要素を作成し、参照要素の持つ子タグ一覧の末尾に挿入する。参照要素の期待タグはここで作成したスキーマ要素を指すようにする。
［ＳＴＥＰ　９］参照要素には一時変数として処理済フラグというフラグを設け、一度処理を行ったスキーマ要素にはＴＲＵＥが、処理を行っていない要素にはＦＡＬＳＥが設定されるようにしておく。そして処理済フラグがＴＲＵＥの場合、ＳＴＥＰ　１０を実行する。これはＳＴＥＰ　８で到来したタグが、以前は省略されていたことを示す。具体例を図１５の例１５０２を用いて説明する。最初のタグＡに存在するタグＢが到来した時、タグＡの処理済フラグはＦＡＬＳＥであり、タグＢの重複フラグは「単独」となる。２番目のタグＡの中に存在するタグＣが到来した時、タグＡの処理済フラグはＴＲＵＥであるため、タグＣの重複度は「重複」となる。これは、タグＡが２回目以降の処理であり、前回の処理でタグＣが省略された、すなわちタグＣは省略可能なタグである、と判断したことに基づく。
［ＳＴＥＰ　１０］　ＳＴＥＰ　８で新規に挿入したスキーマ要素の重複フラグを「重複」に設定する。
［ＳＴＥＰ　１１］　参照要素の要素型を「ＳＥＱＵＥＮＣＥ」に設定する。これは参照要素に子要素が存在しているためである。
［ＳＴＥＰ　１２］　部品が終了タグである場合、本処理を実行する。参照タグの全ての子要素について、ＳＴＥＰ　１３を実行する。
［ＳＴＥＰ　１３］　重複数が１ではない子要素について、子要素の重複フラグを「重複」に設定する。重複数が０の子要素は、参照タグ内においてその子要素が省略されたことを示し、重複数が２以上の子要素は、参照タグ内においてその子要素が２回以上出現したことを示す。いずれも「子要素が必ず１回のみ出現する」という条件から外れるため、重複フラグは「重複」となる。
［ＳＴＥＰ　１５］　参照要素の一時変数である処理済フラグをＴＲＵＥにする。
［ＳＴＥＰ　１６］　部品が属性キーである場合、本処理を実行する。属性キーが参照要素の属性として登録済みの場合、何も処理は行わない。属性キーが登録されていない場合、新たな属性を生成し、属性一覧の末尾に挿入する。属性の重複フラグは必ず「属性」とし、これは属性が省略されるか、１回出現するかのいずれかを示す。ＸＭＬの定義によると属性が２回以上重複することはないため、子要素のような重複フラグ判定処理は省略可能である。

【００８７】

なおここでは簡単のため、図８に示すパターンのうち兄弟要素パターン８１２でのスキーマ木生成方式について示したが、選択要素パターン８１５、重複選択要素８１６の生成も可能である。具体的には、ＳＴＥＰ　１２の終了タグ処理において、図８パターン８１６のＸＭＬサンプル８０２のタグＢのように、登録された子要素のタグのうち、同名のタグが連続せずに存在する場合、重複選択要素８１６のパターンであることが判定でき、その場合参照要素Ａの要素型を「ＳＥＱＵＥＮＣＥ」から「ＭＣＨＯＩＣＥ」に変更する。また、図８パターン８１５のＸＭＬサンプル８０２のタグＢ、タグＣのように、子要素数は必ず１個であり、そのタグ名は参照要素Ａによって異なる場合、選択要素８１５のパターンが判定でき、その場合、参照要素Ａの要素型を「ＳＥＱＵＥＮＣＥ」から「ＣＨＯＩＣＥ」に変更する。
残りの圧縮方式、すなわち、（２）文字列の辞書化、（３）データ型の適用、（４）可変長数値の適用、（５）相対値の適用、を行うため、図２２に示すアルゴリズムを用いて処理を行う。これは図２１で示した「タグとデータの分離」と同時に行われるが、説明を簡単にするため、図を分けている。

［ＳＴＥＰ　１］～［ＳＴＥＰ　４］までは、前述した、図２１の説明と同じである。ここでは、前回割愛した、テキスト要素についての説明のみを抜き出して示す。
［ＳＴＥＰ　５］　部品がテキスト要素である場合、本処理を実行する。ここでは、テキスト要素が前述した、「整数型」、「実数型」、「２次元整数型」、「２次元実数型」、「２次元整数配列型」、「２次元実数配列型」、「日付型」、「文字列型」のいずれかであるかを判定する。
【００８８】
整数型の判定規則としては、その記述形式（先頭のプラスマイナス記号と０から９までの文字が１個以上でのみ構成されるなど）、記述範囲（ｌｏｎｇ値を超える極端に大きな数値は整数ではなく実数や文字列で管理するなど）の規則を適用できる。また「００２」のような特殊な記述がある場合、「２」として整数で記述するか、文字列で記述するか、また他の拡張形式（例えば固定長３桁の整数など）で記述するかをユーザ指定で方針を切り替えても構わない。
【００８９】
実数型の判定規則としては、その記述形式（小数点１個を含んでもよい、Ｅ＋０２などの指数部表現を許可するなど）、記述範囲（指数部の桁数や仮数部の精度桁数がｄｏｕｂｌｅ値を超えるものは文字列で管理するなど）の規則を適用できる。
２次元整数型、２次元実数型、ならびにそれを拡張した整数配列型、実数配列型は、テキスト要素が整数や実数と、それを区切る区切り文字でできていることを判定する。区切り文字はあらかじめカンマ、空白、タブなどのいずれかで判定してもよく、テキスト要素を解析することで区切り文字を導出してもいい。２次元整数配列型、２次元実数配列型のように区切り文字が二種類存在する場合も同様である。
【００９０】
日付型は、その記述形式（ＩＳＯ８６０１に従うことなど）、記述範囲（日付を１９７０年１月１日午前０時（ＵＴＣ）を起算点とする通算の秒数で管理する場合、その起算点からｌｏｎｇ値の限界まで）の規則を適用できる。
【００９１】
これらの判定規則には優先順位を設ける。たとえば「２」という文字列は整数型でも実数型でも文字列型でもあるが、整数型とした方が圧縮効率がいい。そのため「整数型でも実数型でもある場合は整数型にする」、という優先順位を設ける。優先順位は、「整数型」、「実数型」、「２次元整数型」、「２次元実数型」、「２次元整数配列型」、「２次元実数配列型」、「日付型」、「文字列型」の順となる。
［ＳＴＥＰ　６］　現在の参照要素の期待タグが示すスキーマ要素（以降は期待要素と呼ぶ）に対し、スキーマ型としてＳＴＥＰ　５で判定した型を代入する。ただし上書規則を設定する必要がある。例えば２個の重複要素が存在し、最初のタグのテキスト要素に整数、次のタグのテキスト要素に実数が入っている場合、そのタグのスキーマ要素の型は「実数」となる。最初のタグのテキスト要素に実数、次のタグのテキスト要素に整数が入っている場合も、そのタグのスキーマ要素の型は「実数」となる。
【００９２】
図２０に上書規則を示す。あるスキーマ要素のスキーマ型が、既に図２０の列に示す型として判定されている状態で、新たに左端に示すスキーマ型で上書しようとする場合の規則を示す。ここで、「ＯＫ」は上書可能、「ＮＧ」は上書不可能、「文字列」は元の型や上書する型にかかわらず文字列型に置換されてしまうことを示す。
［ＳＴＥＰ　７］　文字列辞書化圧縮方式のオプションが選択されており、かつ期待要素のスキーマ型が文字列型と判定された場合、本処理を実行する。本処理では、頻出する文字列を抜き出し、スキーマ木のスキーマ要素毎に存在する辞書に、文字列と、それに対応する識別子を登録することにより、データサイズの削減を図る。
【００９３】
文字列辞書化のアルゴリズムは多数考えられる。例えば、（１）単純辞書化、（２）頻出単語登録、（３）事前辞書準備、（４）数値分離、の方式が考えられる。

（１）単純辞書化は、全てのテキスト要素を分割せずにそのまま辞書登録を行う。辞書登録量の爆発を防止するため、辞書の登録件数が閾値以上（例えば１００件以上）の場合、そのスキーマ要素は辞書化が困難であると判定して辞書登録を停止する、という戦略が可能である。
（２）頻出単語登録は、テキスト要素を単語に分割して、その出現数と共に一時バッファに登録する。単語の分割は既存の形態素解析等を利用することで行う。そして単語の出現数が閾値以上（例えば上位１０件）のみを登録することにより、辞書の肥大化を防止することができる。
（３）事前辞書準備は、あらかじめ外部から辞書を提供する方式である。一般に、ＸＭＬ文書を作成するユーザはそのＸＭＬ文書内に頻出する用語を理解している場合があり、この場合には本方式が有効となる。
（４）数値分離は、テキスト要素を共通の文字列と固有の数値に分離できると仮定して、テキスト要素を文字データの前半部分と数値の後半部分に分離し、前半部分を辞書化する方法である。例えば前述のＵＲＩのように、識別子の前半がＵＲＬのような文字データ、後半が組織固有の数値というように分離できる場合に有効である。
［ＳＴＥＰ　８］　相対座標管理方式のオプションが選択されており、かつ期待要素のスキーマ型が整数型、実数型などの数値型であると判定された場合、本処理を実行する。本処理では、数値の相対値管理を行うための、起点となる数値を取得するために実行する。
【００９４】
相対値管理のアルゴリズムは多数考えられる。例えば次の３つが考えられる。すなわち、（１）差分管理、（２）起点計算、（３）起点入力、である。
（１）差分管理は、テキスト要素として、同じスキーマ要素の前出したテキスト要素からの相対値管理を行う方式である。例えば３つのテキスト要素、１００００、１００００１、１００００２がある場合、格納される相対値としては１００００、１、１となる。
（２）起点計算は、同じスキーマ要素に対し、起点を計算する方式である。同じスキーマ要素の全てのタグに対して最小となる数値を計算し、その数値を起点とする。あるいは同じスキーマ要素の全てのタグに対して平均となる数値を計算し、その数値を起点とする。後者の方が、極端に大きいあるいは小さい数値を含む場合に全体として安定している。例えば３つのテキスト要素、１００００、１００００１、１００００２がある場合、起点を平均値１００００１とすると、圧縮データに格納される相対値は、－１、０、１となる。起点情報は図２９の２９３２のように、圧縮スキーマに記述する。
（３）起点入力は、あらかじめ外部から起点を入力する方式である。例えばユーザの設定により起点を入力する。また、ある種のＸＭＬ文書には数値範囲を記録している場合がある。たとえば地理情報分野に置ける多数の地物を表現するＸＭＬ文書の中には、地物の位置を示す座標群の他に、全ての地物を外接する矩形を格納している場合がある。この矩形の端点、あるいはこの矩形の中心点を相対値管理の起点として採用することもできる。
【００９５】
次に、図１のスキーマ変換ブロック１１６について詳細に説明する。スキーマ変換ブロック１１６の目的は、スキーマ生成ブロック８０９の代替ブロックとして、ＸＭＬ　Ｓｃｈｅｍａなどのスキーマ定義文書１０６からスキーマ木１０２を作成することである。
【００９６】
たとえばＸＭＬ　Ｓｃｈｅｍａでは、あるタグの中にどのタグが含まれるか、タグの最小出現回数と最大出現回数、テキスト要素の型を定義することができるため、スキーマ木への変換を行うことができる。また頻出単語や相対値など、ＸＭＬ　Ｓｃｈｅｍａで表現できない項目については、その部分だけ別の形式で管理することもでき、またＸＭＬ　Ｓｃｈｅｍａの拡張を行い、上記の情報を格納するようにすることもできる。
【００９７】
逆に、ＸＭＬ　Ｓｃｈｅｍａでは表現可能であるが、ＢＸＭＬの圧縮スキーマでは表現できない構造も存在する。本発明の圧縮ＸＭＬの目的は、ＸＭＬのサイズを削減し、高速な

処理を実現することであり、ＸＭＬ　Ｓｃｈｅｍａで示される構造のチェック（ｖａｒｉｄａｔｉｏｎ）を行う必要は必ずしもない。そのため、以下のようなＸＭＬ構造の簡略化を行う。

（１）ＸＭＬ　Ｓｃｈｅｍａでは、「正の整数型」「単精度３２ｂｉｔ浮動小数点型」など、あらかじめ４４種類のデータ型が定められている。本発明では、「正の整数型」を「整数型」に変換するなど、型変換規則を適用することにより、型の違いを吸収する。

（２）ＸＭＬ　Ｓｃｈｅｍａでは、あるタグの子要素として出現する組み合わせを自由に定義できる。例えば、タグＡの子要素としてタグＢ、タグＣ、タグＤ、タグＥが存在する可能性がある場合、タグＢ、タグＣ、タグＤ、タグＥが連続して１回だけ来るのか、タグＢ、タグＣ、タグＤ、タグＥの何れかが１回だけ来るかなど、ＸＭＬ　Ｓｃｈｅｍａでは様々な組み合わせが定義できる。一方、本発明では、図８の兄弟要素パターン８１２、選択要素パターン８１５、重複選択要素パターン８１６の三種類のみを定義している。本発明では、兄弟要素パターン８１２、選択要素パターン８１５に当てはまらないパターンを全て重複選択要素パターン８１６とすることにより組み合わせの違いを吸収する。

図２４に、ＸＭＬ　Ｓｃｈｅｍａで定義可能な出現組み合わせの例をＢＮＦ（Ｂａｃｋｕｓ　Ｎａｕｒ　Ｆｏｒｍ）記法で示したもの２４０１、そのサンプル２４０２、圧縮スキーマでのＢＮＦ記法２４０３、それに対応する要素型２４０４を示す。

【００９８】

次に、図１のデータ圧縮ブロック１１２について詳細に説明する。データ圧縮ブロック１１３の目的は、スキーマ木１０２を利用してＸＭＬ文書１０１をパースし、圧縮データ１０４を作成することである。

【００９９】

初めに、「タグとデータの分離」圧縮７０１に対する圧縮データの作成方法について説明する。その後、「文字列辞書の適用」圧縮７０３、「相対値の適用」圧縮７０５に対する圧縮データの作成方法について説明する。

【０１００】

「タグとデータの分離」圧縮７０１に対する圧縮データの作成方法について説明する。本方式では、スキーマ木を参照しながら、ＸＭＬ文書を解析し、目的の圧縮データを作成する。

【０１０１】

データ圧縮ブロック１１３は、ＳＡＸパーサであるＸＭＬパースブロック１０７を利用して実現する。データ圧縮ブロック１１３では、各ＳＡＸのイベントに対し、ＸＭＬ文書で現在参照している位置がスキーマ木のどの部分であるかを把握しておく必要がある。そのため、ＳＡＸイベントに対し、スキーマ木を移動していく手段が必要となる。

【０１０２】

初めに、図２５を用いて、ＸＭＬ文書をパースする際、現在参照中のタグやテキスト要素が、スキーマ木のどのスキーマ要素に対応するかを判定する方法について説明する。

【０１０３】

ＸＭＬ文書２５０１と、そのスキーマ構造を表すスキーマ木２５０２が存在する場合を考える。スキーマ木２５０２の矩形Ａ、Ｂ、ＣなどはＸＭＬ文書２５０１の同名のタグが囲む領域を表すスキーマ要素である。一時変数として、スキーマ木外部に、現在参照中のスキーマ要素を表す参照要素２５０４を１つ設定する。また、全てのスキーマ要素に対し、現在参照している子要素を示す期待タグ（２５０３、２５０５）と、現在参照している属性を示す期待キーを持つ。以降の説明では簡単のため、期待タグの指すスキーマ要素を期待要素と呼ぶことにする。

【０１０４】

スキーマ木移動のアルゴリズムを図２６に示す。これを図２５の具体例、すなわちサンプルＸＭＬ文書２５０１と、そのスキーマ構造を示すスキーマ木２５０２を用いて説明する。

［ＳＴＥＰ　１］　ＸＭＬ文書２５０１の終端になるまで、ＳＴＥＰ　２以降を繰り返す

。
［ＳＴＥＰ　２］　ＸＭＬ文書を構成する部品、すなわち開始タグ、終了タグ、テキスト要素、属性を一つづつ取り出す。
［ＳＴＥＰ　３］　部品の種類により、ＳＴＥＰ　４、ＳＴＥＰ　６、ＳＴＥＰ　７、ＳＴＥＰ　８のいずれかに分岐する。
［ＳＴＥＰ　４］　本ステップは開始タグが到来した時に実行される。本ステップでは、到来したタグが、現在の参照要素の指すスキーマ要素が持つ期待要素と一致するかを判定し、一致しない場合期待要素を弟要素に進めていく。
本ステップは、重複したり省略されたりするＸＭＬ文書のタグを正しく処理するために存在する。タグの出現パターンは、（１）単独、（２）重複、（３）省略、（４）エラータグの４つに分類される。ＸＭＬ文書２５０１において、次の４つのパターンで説明する。
（１）　開始タグＤ（２５０６）が到来した時、参照要素２１０４はスキーマ要素Ｂ（２５１０）を指しており、スキーマ要素Ｂ（２５１０）の期待要素は最初の子要素であるスキーマ要素Ｄ（２５１１）を指している。到来したタグと期待要素のタグ名が一致するため、そのままＳＴＥＰ　５に移行する。
（２）　開始タグＥ（２５０７）が到来した時、スキーマ要素Ｂ（２５１０）の期待要素は最初の子要素であるスキーマ要素Ｄ（２５１１）を指しており、到来したタグのタグ名と一致しない。そこで期待要素を一つ進めてスキーマ要素Ｅ（２５１２）に移動する。この結果タグ名が一致したため、ＳＴＥＰ　５に移行する。これは単独タグＤ　２５０６を正しく処理したことを示す。
（３）　２番目の開始タグＥ（２５０８）が到来した時、スキーマ要素Ｂ（２５１０）の期待要素はスキーマ要素Ｅ（２５１２）を指しており、到来したタグと期待要素のタグ名が一致するため、そのままＳＴＥＰ　５に移行する。これは重複タグＥ（２５０７、２５０８）を正しく処理したことを示す。
（４）　ＸＭＬ文書２５０１において仮に２つのタグＥ（２５０７、２５０８）が省略され、終了タグＤ（２５１４）の直後に開始タグＦ（２５０９）が来た場合を考える。スキーマ要素Ｂ（２５１０）の期待要素は最初の子要素であるスキーマ要素Ｄ（２５１１）を指しており、到来した開始タグＦ（２５０９）と一致しない。期待要素を次のスキーマ要素Ｅ（２５１１）にするが、まだ開始タグＦ（２５０９）と一致しない。期待要素をさらに次のスキーマ要素Ｆ（２５１３）にすると、開始タグＦ（２５０９）と一致するため、ＳＴＥＰ　５に移行する。これはタグＥ（２５０７、２５０８）が省略された状況を正しく処理したことを示す。
（５）スキーマ木に存在しないタグＸが来た時、期待タグは全ての子要素を探索するが、到来タグと一致する要素を発見することができない。この場合本ステップは例外を発行し、ＸＭＬ文書２５０１が正しい構文ではないことを報告する。
［ＳＴＥＰ　５］　参照要素を、現在の参照要素の期待要素に移動する。例えば上記の例のパターン（１）で開始タグＤ（２５０６）が到来した時、スキーマ要素Ｂ（２５１０）を指していた参照要素２１０４をスキーマ要素Ｄ（２５１１）に移動する。
［ＳＴＥＰ　６］　本ステップは終了タグが到来した時に実行される。本ステップでは、現在の参照要素をその参照要素の親要素に移動する。例えば、終了タグＤ（２５１４）が到来した時、スキーマ要素Ｄ（２５１１）を指していた参照要素２５０４をスキーマ要素Ｂ（２５１０）に移動する。
【０１０５】
上記説明したとおり、ＳＴＥＰ　４、ＳＴＥＰ　５とＳＴＥＰ　６をＸＭＬ文書の開始タグ・終了タグに順次適用することにより、重複タグ、省略タグを含む任意のＸＭＬ文書に対し、参照要素２１０４をスキーマ木の適切な位置に移動することができる。
［ＳＴＥＰ　７］　本ステップはテキスト要素が到来した時に実行される。スキーマ木移動アルゴリズムでは、本ステップでは何も行わない。後述するデータ圧縮ブロック１１３では、本ステップでテキスト要素圧縮処理を行う。本ステップにおいて、参照要素２５０４はそのテキスト要素を含むタグのスキーマ要素を指している。例えばテキスト要素”ｄ

１”（２５１５）が到来した時の参照要素２５０４はスキーマ要素Ｄ（２５１１）を指している。
［ＳＴＥＰ　８］　本ステップは属性が到来した時に実行される。図２１では簡単のため省略したが、図１４に示すように、スキーマ要素１４０１は、子要素を示す０個以上のスキーマ要素１４０３への接続を持つと同様に、属性を示す０個以上のスキーマ要素１４０２への接続を持つ。属性のスキーマ木移動も上述した子要素のスキーマ木と同様にできる。本ステップでは、到来した属性キーが、現在の参照要素の指すスキーマ要素が持つ期待キーと一致するかを判定し、一致しない場合期待キーを弟要素に進めていく。
【０１０６】
図２７、図２８、図２９を用いて「タグとデータの分離」による圧縮について説明する。
【０１０７】
図２７は、図８で示したＸＭＬ文書のパターンのうち、子要素パターン８１１、兄弟要素パターン８１２のみを用いた重複・省略なしの単純なＸＭＬ文書に対するデータ圧縮ブロック１１３のアルゴリズムである。
【０１０８】
ＸＭＬ文書２７０１から前に説明したスキーマ生成ブロック１１０を用いて、スキーマ木２７０２を得る。タグＡはタグＢ、タグＣを保有し、タグＢ、タグＣはテキスト要素を保有する。タグＡ、タグＢ、タグＣはいずれも、必ず１回のみ出現するため、重複フラグは「単独」である。圧縮データ２７０３はテキスト要素ｂ１、ｂ２が連続する構造を取る。
【０１０９】
図２６に示すスキーマ木移動のアルゴリズムをベースに、データ圧縮を実現する。具体的には、図２６のＳＴＥＰ　７のテキスト要素処理においてテキスト文字列を取得し、これを圧縮データに追加していく。ＳＴＥＰ　７で説明したとおり、ＳＴＥＰ　７を実行する時点でそのテキスト要素が含まれるタグのスキーマ要素は、参照要素２５０４からアクセスすることができる。そのスキーマ要素のスキーマ型に対し、以下のルールを適用することにより圧縮データを作成する。
［ＲＵＬＥ　１］　スキーマ要素が複合型（ＳＥＱＵＥＮＣＥ型、ＣＨＯＩＣＥ型、ＭＣＨＯＩＣＥ型）の場合、テキスト要素は無視する。これは具体的には、ＸＭＬ文書２５０１のタグ＜Ａ＞と＜Ｂ＞、＜／Ｂ＞と＜Ｃ＞、＜／Ｃ＞と＜／Ａ＞の間のテキスト要素を無視するということを意味する。上に上げたテキスト要素は全てスキーマ要素Ａ（２７０４）のテキスト要素であり、スキーマ要素Ａのスキーマ型はＳＥＱＵＥＮＣＥ型であるため無視される。
［ＲＵＬＥ　２］　スキーマ型が文字列型の場合、圧縮データに文字データを格納する。
【０１１０】
ＢＸＭＬ圧縮方式として、いくつかのオプションである圧縮方式を前述している。これらの圧縮方式により、以下の拡張ルールを適用する。
「文字列の辞書化」による圧縮を行う場合、次のルールを適用する。
［ＲＵＬＥ　３］　スキーマ型が文字列型で語彙辞書が存在する場合、テキスト要素をこの辞書と比較して、図１１に示したようなテキスト要素と識別子の組合せを圧縮データに格納する。
「データ型の適用」「可変長数値の適用」圧縮を行う場合は次のルールを適用する。
［ＲＵＬＥ　４］　スキーマ型が整数の場合、テキスト要素を図１２で示したＭＢＩ形式整数に変換して圧縮データに追加する。
［ＲＵＬＥ　５］　スキーマ型が実数の場合、テキスト要素を図１２で示したＭＢＦ形式実数に変換して圧縮データに追加する。
［ＲＵＬＥ　６］　スキーマ型が２次元整数の場合、テキスト要素を連続する２つのＭＢＩ形式整数に変換して圧縮データに追加する。
［ＲＵＬＥ　７］　スキーマ型が２次元実数の場合、テキスト要素を連続する２つのＭＢＦ形式実数に変換して圧縮データに追加する。
【０１１１】

同様に他のデータ型に対しても、それに特有の格納方法で格納することができる。

【０１１２】

また、「相対値の適用」圧縮を行う場合、スキーマ木の各スキーマ要素に相対値の起点が格納されている。そして次のルールを適用する。

［ＲＵＬＥ　８］　スキーマ型が数値（整数・実数等）であり、相対値の起点が格納されている場合、テキスト要素の数値から起点までの差分を圧縮データに追加する。

【０１１３】

図２８は、図２７に対し、図８で示したＸＭＬ文書のパターンのうち、さらに重複要素８１３が追加されたＸＭＬ文書に対するデータ圧縮ブロック１１３のアルゴリズムである。

【０１１４】

ＸＭＬ文書２８０１から前に説明したスキーマ生成ブロック１１０を用いて、スキーマ木２８０２を得る。タグＡはタグＢ、タグＣを保有し、タグＢ、タグＣはテキスト要素を保有する。タグＡ、タグＢ、タグＣはいずれも、複数回出現するため、スキーマ要素２８０５、２８０６の重複フラグは「重複」である。圧縮データ２８０３は、タグＢの重複数Ｎｂ、タグＢのテキスト要素ｂ１、ｂ２、タグＣの重複数Ｎｃ、タグＣのテキスト要素ｃ１、ｃ２が連続する構造を取る。

【０１１５】

図２７と異なる点は、圧縮データに重複要素の重複数を記述するという点である。圧縮データ２４０３において、タグＢの重複数ＮｂはタグＢのテキスト要素ｂ１、ｂ２より先に記述する。重複数Ｎｂはテキスト要素ｂ１、ｂ２の処理が終わった時点で判明するため、先に仮の重複数を記述しておき、テキスト要素の処理が終了した後に正しい重複数を記述する、という方式を取る。

【０１１６】

図２６に示すスキーマ木移動のアルゴリズムをベースに、データ圧縮を実現する。初めに、スキーマ木の各スキーマ要素が持つ一時変数として、重複カウンタを設置する。重複カウンタはそのスキーマ要素が同じ階層で何回出現したかを計測するカウンタであり、開始タグが出現する度に１づつ増加し、その親タグの終了タグが出現した時に上述した重複数を設定するために用いられる。

具体的には、図２６で説明したステップに、図２７で説明した処理に加え、以下の処理を加える。

［ＳＴＥＰ　４］　開始タグ処理の拡張：到来した開始タグが期待要素と一致する場合、期待要素の重複カウンタを１つ増加させる。期待要素の重複フラグが「重複」であり、重複カウンタが１（初めて出現したタグ）の場合、圧縮データに重複数を追加する。ただしこの段階では重複数の値は不明であるため、仮の値を追加しておき、後で変更を行うため、スキーマ要素の一時変数として「重複数登録位置」を格納しておく。

［ＳＴＥＰ　６］　終了タグ処理の拡張：現在の参照要素の持つ全ての子要素について、その重複カウンタを取得し、これを最終的な重複数として圧縮データに記述する。記述する場所は、それぞれのスキーマ要素の一時変数「重複数登録位置」に記録されている値を利用する。

【０１１７】

図２９は、図２７に対し、図８で示したＸＭＬ文書のパターンのうち、省略要素パターン８１４が追加されたＸＭＬ文書に対するデータ圧縮ブロック１１３のアルゴリズムである。

【０１１８】

ＸＭＬ文書２９０１から前に説明したスキーマ生成ブロック１１０を用いて、スキーマ木２９０２を得る。タグＲは３個のタグＡを保有する。タグＡ以下のスキーマ構造は、図２８と同様である。最初のタグＡは子要素としてタグＢ、タグＣを持つが、２番目のタグＡは子要素のタグＢが省略されている。３番目のタグＡは子要素のタグＣが省略されている。

圧縮データ２８０３は、タグＡの重複数Ｎａ、最初のタグＡにおけるタグＢの重複数Ｎｂ

、タグＢのテキスト要素ｂ１、タグＣの重複数Ｎｃ、タグＣのテキスト要素ｃ１、そして２番目のタグＡにおけるタグＢの重複数Ｎｂ２、ここで２番目のタグＡにおけるタグＢは省略されているのでタグＢのテキスト要素はなく、次にタグＣの重複数Ｎｃ２、タグＣのテキスト要素ｃ２、そして３番目のタグＡにおけるタグＢの重複数Ｎｂ３、タグＢのテキスト要素ｂ３、タグＣの重複数Ｎｃ３、３番目のタグＡにおけるタグＣは省略されているのでタグＣのテキスト要素はなし、というデータが連続する構造を取る。

【０１１９】

図２７と異なる点は、省略により開始タグが出現しない場合がある、という点である。ＳＴＥＰ　６の終了タグ処理で、「重複数設定位置」に子要素の重複数を記述するが、省略されたタグについてはＳＴＥＰ　４の開始タグ処理を通らないため、「重複数設定位置」が設定されていない。

これに対応するため、図２６で説明したステップに、図２７、図２８で説明した処理に加え、以下の処理を加える。

［ＳＴＥＰ　４］　開始タグ処理の拡張：到来した開始タグが期待要素と一致しない場合、期待要素を弟要素に進めていくが、その前に一致しなかった期待要素の重複数０を圧縮データに追加する。

［ＳＴＥＰ　６］　終了タグ処理の拡張：終了タグの子要素の内、期待要素より後に並ぶ要素群は、その終了タグの子要素として出現しなかったことになる。出現しなかった子要素の重複数０を圧縮データに追加する。

【０１２０】

次に、図１のスキーマ圧縮ブロック１１０について詳細に説明する。スキーマ圧縮ブロック１１０は、計算機のメモリ空間上に展開されているスキーマ木１０２を、ファイルとして保管・通信できるような形である圧縮スキーマ１０３に変換する。

【０１２１】

図３０の圧縮スキーマ構造を用いてスキーマ圧縮ブロック１１０について説明する。

【０１２２】

圧縮スキーマ１０３は、「タグとデータの分離」圧縮７０１を行う場合に使用する、スキーマ木の構造を表す構造スキーマ３００１、「文字列の辞書化」圧縮７０３を行う場合に使用する辞書スキーマ３００２、「相対値の適用」圧縮７０５を行う場合に使用する相対値スキーマ３００３から構成される。各スキーマはスキーマ開始トークン３００５、３００６、３００７より開始し、圧縮データ３００４の開始を表すデータトークン３００８と区別する。またそれぞれのスキーマの種類を判定するため、構造スキーマ３００１には構造トークン３００９、辞書スキーマ３００２には辞書トークン３０１０、相対値スキーマには相対値トークン３０１１を格納する。

【０１２３】

構造スキーマ３００１の説明を行う。図１４のスキーマ木を構成するスキーマ要素１４０１は、タグ名１４０５、重複フラグ１４０６、スキーマ型１４０７、属性保有数１４０８、子要素保有数１４０９、で構成され、属性保有数で示される個数（０個以上）の、属性を示すスキーマ要素１４０２へのリンクポインタ、子要素保有数で示される個数（０個以上）の、子要素タグを示すスキーマ要素１４０３へのリンクポインタから構成されている。属性を示すスキーマ要素１４０２も同じ形式で表現でき、タグ名をキー名１４１１として使用し、属性保有数や子要素保有数が０個、属性や子要素へのリンクポインタが存在しない構造で表現できる。

スキーマ木のタグを示すスキーマ要素１４０１は次のような手順で圧縮スキーマ３０１９に変換することができる。まずタグ名３０１２は文字列であるため、０で終端する文字列として格納する。重複フラグ３０１３は「重複」「単独」のいずれかを取るため、それぞれの列挙に識別子を割り振ることにより登録する。スキーマ型３０１５は前述の「ＳＥＱＵＥＮＣＥ」や「ＣＨＯＩＣＥ」、「ＭＣＨＯＩＣＥ」、「整数型」、「実数型」などと列挙される項目のいずれかを取るため、それぞれの列挙に識別子を割り振ることにより登録する。子要素保有数３０１４や属性保有数３０１６は整数であるため、前述のＭＢＩ形

式の整数で格納する。また、スキーマ型が子要素を含まないことが自明である場合、具体的にはＳＥＱＵＥＮＣＥ型、ＣＨＯＩＣＥ型、ＭＣＨＯＩＣＥ型以外、すなわち文字列型、整数型、実数型、ならびに「データ型の適用」７０２において述べた各種スキーマ型である場合、子要素保有数３０１４は必ず０個であり、省略できる。
属性を示すスキーマ要素１４０２は、次のような手順で圧縮スキーマ３０２０に変換することができる。キー名３０１７、スキーマ型３０１８を、タグ名３０１２、スキーマ型３０１５と同様な方法で格納する。属性保有数、子要素保有数は必ず０個であり、省略できる。また重複フラグ１４１２も必ず「属性」であるため、省略できる。

【０１２４】

スキーマ木の、スキーマ要素間のリンク関係は次のような手順で圧縮スキーマに変換することができる。まずルート要素のスキーマ要素３０１９を圧縮スキーマに記述する。属性保有数３０１６を１個以上指定した場合、属性を示す１個以上のスキーマ要素３０２０を次に連続して記述する。次に子要素保有数を１個以上指定した場合、子要素タグを示す１個以上のスキーマ要素３０２１を次に連続して記述する。子要素タグを示すスキーマ要素がそれぞれ更に属性や子要素を持つ場合、そのスキーマ要素３０２１の弟を記述する前に再帰的に属性、子要素を記述する。この方式では、全てのスキーマ要素に対し、自分自身の情報として属性保有数３０１６、子要素保有数３０１４を持ち、その後に属性・子要素が順番に記述される、という構造を持つため、リンク関係は完全に保存される。

【０１２５】

次に、辞書スキーマ３００２の説明を行う。「文字列の辞書化」の圧縮を行う場合、圧縮スキーマには、辞書情報を格納する必要がある。辞書情報としては、スキーマトークン３００６、辞書トークン３０１０の後に辞書の総数３０２２を格納する。その後、辞書情報３０２３を、辞書総数３０２２と同数繰り返して記述する。辞書情報３０２３は、スキーマ要素識別子３０２４、登録単語数３０２５、を格納する。その後、単語情報３０２６を、登録単語数３０２５と同数繰り返して記述する。単語情報３０２６は、単語を識別する識別子３０２７、単語の文字列３０２８を格納する。

【０１２６】

スキーマ要素識別子３０２４は、辞書が対象とするスキーマ要素を示す識別子を記述する。この識別子は、図１４のスキーマ要素識別子１４０４を記載する。図３０の構造スキーマ３００１にスキーマ要素識別子１４０４を明示的に記述してもよいが、「スキーマ要素識別子はスキーマ木の出現順の通し番号とする」などの決定方法が明確になっていれば、構造スキーマ３００１への記載は省略することが出来る。

【０１２７】

次に、相対値スキーマ３００３の説明を行う。「相対値の適用」の圧縮を行う場合、圧縮スキーマには相対値の起点となる数値を格納する必要がある場合がある。起点情報としては、スキーマトークン３００７、辞書トークン３０１１の後に相対値格納数３０２９を格納する。その後、相対値情報３０３０を、相対値格納数３０２９と同数繰り返して記述する。相対値情報３０３０は、スキーマ要素識別子３０３１、起点数値３０３２、を格納する。

【０１２８】

スキーマ要素識別子３０３１は、相対値管理を行うスキーマ要素を示す識別子であり、辞書スキーマにおけるスキーマ要素識別子３０２４と同様、図１４のスキーマ要素識別子１４０４を記載する。また起点数値３０３２は、整数、実数、２次元整数、すなわち座標（ｘ、ｙ）を示す二つの数値、２次元実数のいずれかを格納する。起点数値がどの形式で格納されるかは、そのスキーマ要素識別子３０３１が示すスキーマ要素のスキーマ型３０１５により決定することができる。

【０１２９】

次に、図１のスキーマ展開ブロック１１１について詳細に説明する。スキーマ展開ブロック１１１は、スキーマ圧縮ブロック１１０の逆の機能ブロックであり、ファイルや通信で取得した圧縮スキーマ１０３をスキーマ木１０２として計算機のメモリ空間上に展開する

。これは、スキーマ圧縮ブロック１１０で説明した手順の逆操作を行うことにより実現できる。

【０１３０】

スキーマ展開ブロックでは、初めにスキーマトークン３００５、３００６、３００７を識別子、それに続く構造トークン３００９、辞書トークン３０１０、相対値トークン３０１１を判別し、それぞれに従った展開を行う。本発明の圧縮構造を拡張してあらたなスキーマ構造を追加する場合、スキーマトークンの後に拡張したスキーマを識別するトークンを記述することにより容易に拡張することが出来る。そしてデータトークンが出現した位置で圧縮データが開始されると判定することが出来る。

【０１３１】

次に、図１のＢＸＭＬ結合ブロック１１４について詳細に説明する。ＢＸＭＬ結合ブロック１１４では、圧縮スキーマ１０３および圧縮データ１０４を結合することにより、ＢＸＭＬ文書１０５を取得する。ＢＸＭＬ結合ブロック１１４は、単純に圧縮スキーマ１０３の後に圧縮データ１０４を連結させることにより実現できる。

【０１３２】

次に、図１のＢＸＭＬ分解ブロック１１５について詳細に説明する。ＢＸＭＬ分解ブロック１１５では、ＢＸＭ文書１０５を圧縮スキーマ１０３と圧縮データ１０４に分解する。ＢＸＭＬ分解ブロック１１５は、ＢＸＭＬ文書１０５において、圧縮スキーマ１０３の開始位置と終了位置、圧縮データ１０４の開始位置と終了位置を調査し、開始位置から終了位置までを別データとして保存することにより実現できる。実際の利用として、文字通り分解しなくても、圧縮データの開始位置を取得するだけで十分な場合がある。なお、圧縮スキーマの開始位置はＢＸＭＬ文書の開始位置と同じである。圧縮データの開始位置は、スキーマ展開ブロック１１１で説明した通り、データトークン３００８を判定することにより確定できる。

【０１３３】

次に、図１のＢＸＭＬパースブロック１０８について詳細に説明する。ＸＭＬパースブロック１０７がＸＭＬ文書をパースし、外部ハンドラルーチンに対しＸＭＬ文書の構造を表すイベントを発行すると同様、ＢＸＭＬパースブロック１０８は、ＢＸＭＬ文書をパースし、外部ハンドラルーチンにたいしＢＸＭＬ文書の構造を表すイベントを発行する。外部ハンドラルーチンとしてデータ展開ブロック１１３を利用することによりＢＸＭＬ文書をＸＭＬ文書に展開することができる。またＳＡＸパースブロック１２３を利用することによりＢＸＭＬ文書の標準ＳＡＸインタフェースを実現することができる。

【０１３４】

図３１に、ＢＸＭＬパースブロックのアルゴリズムを示す。一時変数として、図１４に示すスキーマ木において現在参照しているスキーマ要素１４０１を指す「参照要素」、圧縮データにおいて現在参照している位置を指す「参照ポインタ」を用意する。「参照要素」の初期位置はＸＭＬ文書のルート要素を指すスキーマ要素を、「参照ポインタ」の初期位置は圧縮データの先頭を指しておく。

［ＳＴＥＰ　１］　参照要素の重複フラグ１４０６を取得し、「単独」の場合はＳＴＥＰ　３以降のノード処理を実行する。重複フラグが「重複」の場合はＳＴＥＰ　２に進む。

［ＳＴＥＰ　２］　圧縮データから子要素の重複数を取得し、重複数と同じ回数、ＳＴＥＰ　３以降のノード処理を実行する。

［ＳＴＥＰ　３］　開始タグ名を引数として、ユーザの定義する開始タグ処理を実行する。開始タグ名は参照要素のタグ名１４０５から取得できる。

［ＳＴＥＰ　４］　参照要素から属性保有数１４０８を取得し、属性保有数と同じ回数、ＳＴＥＰ　５以降を繰り返す。

［ＳＴＥＰ　５］　圧縮データから属性の重複数を取得し、重複数が１の場合のみ、ＳＴＥＰ　６以降を実行する。

［ＳＴＥＰ　６］　圧縮データから属性値として文字列を取得する。「文字列の辞書化」圧縮７０３を行っている場合は、図１１の方法を用いて、単語識別子３０２７から文字列

３０２８への復元を行う。
［ＳＴＥＰ　７］　属性キーと属性値を引数として、ユーザの定義する属性処理を実行する。属性キーは参照要素のキー名１４１１から取得できる。
［ＳＴＥＰ　８］　参照要素のスキーマ型１４０７により、処理を分岐する。簡単のため、スキーマ型がＳＥＱＵＥＮＣＥ型の場合と、スキーマ型が文字列型の場合について説明するが、ＣＨＯＩＣＥ型、ＭＣＨＯＩＣＥ型、整数型、実数型等、他のスキーマ型でも同様である。
［ＳＴＥＰ　９］　本ステップは、スキーマ型が「ＳＥＱＵＥＮＣＥ型」の場合に実行する。参照要素から子要素保有数１４０９を取得し、子要素保有数と同じ回数、ＳＴＥＰ　１以降を再帰的に繰り返す。ここで繰り返されるＳＴＥＰ　１以降では、参照要素は対象とする子要素に移動している。
［ＳＴＥＰ　１０］　本ステップは、スキーマ型が「文字列型」の場合に実行する。圧縮データからテキスト要素として文字列を取得する。「文字列の辞書化」圧縮を行っている場合は、図１１の方法を用いて、単語識別子３０２７から文字列３０２８への復元を行う。
［ＳＴＥＰ　１１］　文字列型のテキスト要素を引数に、スキーマ型が「文字列型」のテキスト要素に対するユーザ定義処理を実行する。
［ＳＴＥＰ　１２］　本ステップは、スキーマ型が「整数型」の場合に実行する。圧縮データからテキスト要素として整数を取得する。「相対値の適用」圧縮７０５を行っている場合は、ここで相対値から元の数値への復元を行う。
［ＳＴＥＰ　１３］　整数型のテキスト要素を引数に、スキーマ型が「整数型」のテキスト要素に対するユーザ定義処理を実行する。
［ＳＴＥＰ　１４］　終了タグ名を引数として、ユーザの定義する終了タグ処理を実行する。終了タグ名は参照要素のタグ名１４０５から取得できる。
【０１３５】
次に、図１のデータ展開ブロック１１３について詳細に説明する。データ展開ブロック１１３の目的は、スキーマ木１０２を利用して圧縮データ１０４をＸＭＬ文書１０１に展開することである。データ展開ブロック１１３は、図３１に示したＢＸＭＬパースブロック１０８を利用することにより実装できる。具体的には、図３１の、各ユーザ定義ステップで以下の処理を実現する。
［ＳＴＥＰ　３］　開始タグ処理：前回の開始タグの閉じ括弧が出力されていない場合、閉じ括弧「＞」を出力する。また、開始タグの括弧「＜」を出力し、開始タグ名を取得し、ＸＭＬ文書の開始タグ名として出力する。
［ＳＴＥＰ　７］　属性処理：属性キーと属性値を取得し、ＸＭＬ文書の属性部分を出力する。
［ＳＴＥＰ　１１］　テキスト処理（文字列）：前回の開始タグの閉じ括弧が出力されていない場合、閉じ括弧「＞」を出力する。また、文字列を取得し、ＸＭＬ文書のテキスト要素として出力する。
［ＳＴＥＰ　１３］　テキスト処理（整数）：前回の開始タグの閉じ括弧が出力されていない場合、閉じ括弧「＞」を出力する。また、整数を取得し、文字列に変換して、ＸＭＬ文書のテキスト要素として出力する。
［ＳＴＥＰ　１４］　終了タグ処理：前回の開始タグの閉じ括弧が出力されていない場合、閉じ括弧「＞」を出力する。また、終了タグ名を取得し、ＸＭＬ文書の終了タグ部分を出力する。
【０１３６】
次に、図１のＳＡＸパースブロック１２３について詳細に説明する。ＳＡＸパースブロック１２３の目的は、ＢＸＭＬ文書に対して標準のＸＭＬ文書解析インタフェースの一つであるＳＡＸに準拠したインタフェースをユーザに提供することである。
ＳＡＸパースブロック１２３は、図３１に示したＢＸＭＬパースブロック１０８を利用することにより実装できる。具体的には、図３１の、各ユーザ定義ステップで以下の処理を

実現する。
［ＳＴＥＰ　３］　開始タグ処理：前回の開始タグが未処理の場合、ユーザにＳＡＸ開始タグイベントを発行する。また開始タグ名を記憶する。
［ＳＴＥＰ　７］　属性処理：属性キーと属性値を記憶する。
［ＳＴＥＰ　１１］　テキスト処理（文字列）：前回の開始タグが未処理の場合、ＳＡＸ開始タグイベントを発行する。また、文字列を取得し、ＳＡＸテキスト要素イベントを発行する。
［ＳＴＥＰ　１３］　テキスト処理（整数）：前回の開始タグが未処理の場合、ＳＡＸ開始タグイベントを発行する。また、整数を取得し、文字列に変換して、ＳＡＸテキスト要素イベントを発行する。
［ＳＴＥＰ　１４］　終了タグ処理：前回の開始タグが未処理の場合、ＳＡＸ開始タグイベントを発行する。また、終了タグ名を取得し、ＳＡＸ終了タグイベントを発行する。
【０１３７】
次に、図１のＤＯＭパースブロック１２４について詳細に説明する。ＤＯＭパースブロック１２４の目的は、ＸＭＬ文書の圧縮構造であるＢＸＭＬ文書に対して、図３で説明した標準ＸＭＬ文書解析インタフェースの一つであるＤＯＭに準拠したインタフェースを提供することにより、ＢＸＭＬ文書の解析・更新手段をユーザに提供することである。本発明のＤＯＭパースブロックの特徴は、従来の方式と比較し、ＤＯＭパースに必要なメモリ空間が小さいこと、実行速度が速いことが挙げられる。メモリ空間を小さくする手段としては、（１）解析する対象がＸＭＬ文書ではなく、本発明により圧縮されたＢＸＭＬ文書であること、（２）従来のＤＯＭパーサのようにインスタンス木をメモリ上に展開しない、という点が挙げられる。実行速度を高速化するための手段としては、文字列をできるだけ使用しない方式を提供することが挙げられる。たとえばＢＸＭＬ文書において整数や実数は文字列で格納されておらず、整数や実数を表す適切な型で格納されている。本方式では文字列と整数・実数変換を介することなく取得し、また更新する手段を提供する。またＸＭＬ文書のタグの判定や移動も、文字列ではなく適切な識別子で判定できる手段を提供する。
【０１３８】
図３２を用いて、ＤＯＭパースブロック１２４の実装方式について説明する。ＸＭＬ文書３２０１は、本発明ではスキーマ木３２０２と圧縮データ３２０３で表現される圧縮構造を取る。本発明では、図３で示したインスタンス木３０２をメモリ空間に展開しない。インスタンス木のノード３０４に相当する本発明のノードとして、ノード３２０７のようなＢＸＭＬノード構造を定義する。例えば、ノードＢ１（３２０７）はＸＭＬ文書３２０１の最初のタグＢを、ノードＢ２（３２０８）は、２番目のタグＢを、ノードＣ１（３２０９）は最初のタグＣを表すノードである。
【０１３９】
ＢＸＭＬノードは、内部の変数として、参照スキーマ要素３２１１、重複数３２１２、重複カウンタ３２１３、圧縮データ位置３２１４を持つ。
【０１４０】
参照スキーマ要素３２１１は、そのノードが対応するスキーマ木３２０２のスキーマ要素を指す。ノードＢ１（３２０７）の参照スキーマ要素３２１１は、スキーマ木３２０２の対応するスキーマ要素３２０５を指している。
【０１４１】
重複数３２１２は、ノード３２０７と連続する兄弟の位置に存在する、同名のタグの個数である。ＸＭＬ文書３２０１にはタグＢが２つ連続して存在するため、重複数３２１２は２である。これは、圧縮スキーマ３２０３のＮｂから取得される値である。
【０１４２】
重複カウンタ３２１３は、ノード３２０７が同名の連続するタグの、０から始まる何番目のタグかを示すカウンタである。ノード３２０７は２つあるタグＢの最初のタグであるため、重複カウンタは０である。一方、ノード３２０８の重複カウンタは、２番目のタグで

あるため、重複カウンタは１である。
【０１４３】
圧縮データ位置３２１４は、そのノードの情報が圧縮データ３２０３のどの位置に存在するかを示すポインタである。ノードＢ１（３２０７）の圧縮データ位置３２１４は、圧縮データ３２０３のｂ１の位置、ノードＢ２（３２０８）の圧縮データ位置３２１４は、圧縮データ３２０３のｂ２の位置を指している。
【０１４４】
図３２の構成で、図３に示す標準ＤＯＭインタフェースの５つのメソッド、すなわち（１）属性値の取得、（２）テキスト要素の取得、（３）子へ移動、（４）弟へ移動、（５）親へ移動、の実装アルゴリズムを以下に示す。
図３３を用いて、ＤＯＭパースブロック１２４における「属性値取得」のアルゴリズムについて説明する。「属性値取得」は、引数として属性のキーを入力し、そのノードにおけるキーに対応する属性値を返す処理である。
［ＳＴＥＰ　１］　ノードの参照スキーマ要素３２１１から、そのノードに対応したスキーマ要素を取得し、それに接続する全属性についてＳＴＥＰ　２以降を実行する。
［ＳＴＥＰ　２］　圧縮データから重複数を取得する。
［ＳＴＥＰ　３］　重複数が０の場合はその属性は省略されていることになる。ＳＴＥＰ　１に戻り、次の属性を探索する。全ての属性を探索し尽くせば、全ての属性が省略されていることになり、「属性値取得」は失敗する。重複数が１の場合は、属性が発見されたことになる。現在の属性のキー名と引数キーとを比較し、一致する場合、圧縮データから文字列を取得し、終了する。
図３４を用いて、　ＤＯＭパースブロック１２４における「テキスト要素取得」のアルゴリズムについて説明する。
［ＳＴＥＰ　１］　ノードの変数である圧縮データ位置３２１４は、ノードを示す圧縮データの先頭位置を指している。データ先頭にはそのノードの属性情報が記述されているため、これを読み飛ばす。属性スキップ処理は、図３３を利用することにより実現できる。
［ＳＴＥＰ　２］　ノードの参照スキーマ要素３２１１から、そのノードに対応したスキーマ要素を取得し、そのスキーマ型に応じた方法で圧縮データからテキスト要素を取得する。
図３５を用いて、　ＤＯＭパースブロックにおける「子に移動」のアルゴリズムについて説明する。
［ＳＴＥＰ　１］　ノードの変数である圧縮データ位置３２１４は、ノードを示す圧縮データの先頭位置を指している。ノードの先頭にはそのノードの属性情報が記述されているため、これを読み飛ばす。
［ＳＴＥＰ　２］　ノードの参照スキーマ要素３２１１から、そのノードに対応したスキーマ要素を取得し、それに接続する全子要素についてＳＴＥＰ　３以降を実行する。子要素が存在しない場合、「子への移動」は失敗する。
［ＳＴＥＰ　３］　子要素の重複数を取得する。子要素の重複フラグが「単独」の場合、重複数は１である。子要素の重複フラグが「重複」の場合、圧縮データから子要素の重複数を取得する。
［ＳＴＥＰ　４］　重複数が０の場合、その子要素は省略されているため、ＳＴＥＰ　２に戻り、次の子要素を探索する。全ての子要素を探索し尽くせば、全ての子要素が省略されていることになり、「子への移動」は失敗する。一方、重複数が０より大きい場合、子要素が発見されたことになる。パラメータを設定し、「子への移動」は成功終了する。
図３６を用いて、ＤＯＭパースブロック１２４における「弟に移動」のアルゴリズムについて説明する。
［ＳＴＥＰ　１］　ノードの変数である圧縮データ位置３２１４を、自ノードの最後までパースした状態に移動する。自ノードが子要素を含む場合、子要素の情報もスキップする。これは図１のＢＸＭＬパースブロック１０８を利用することにより実現できる。
［ＳＴＥＰ　２］　ノードの変数である重複カウンタ３２１３が、同じノードの変数であ

る重複数３２１２より小さい場合、同名の弟タグが残っていることになる。パラメータを設定し、「弟への移動」は成功終了する。
［ＳＴＥＰ　３］　ノードの参照スキーマ要素３２１１から、そのノードに対応したスキーマ要素を取得し、そのスキーマ要素の後に存在する全弟要素についてＳＴＥＰ　４以降を実行する。弟要素が存在しない場合、「弟への移動」は失敗する。
［ＳＴＥＰ　３］　弟要素の重複数を取得する。弟要素の重複フラグが「単独」の場合、重複数は１である。弟要素の重複フラグが「重複」の場合、圧縮データから弟要素の重複数を取得する。
［ＳＴＥＰ　４］　重複数が０の場合、その弟要素は省略されているため、ＳＴＥＰ　３に戻り、次の弟要素を探索する。全ての弟要素を探索し尽くせば、全ての弟要素が省略されていることになり、「弟への移動」は失敗する。一方、重複数が０より大きい場合、弟要素が発見されたことになる。パラメータを設定し、「弟への移動」は成功終了する。
図３２を用いて、ＤＯＭパースブロック１２４における「親に移動」のアルゴリズムについて説明する。
【０１４５】
「親へ移動」の機能を実装するためには、たとえば図３２のノード３２０７において、その親ノードに対する状態変数、すなわち参照スキーマ要素３２１１、重複数３２１２、重複カウンタ３２１３、圧縮データ位置３２１４を取得する必要がある。そこで、全てのノードにおいて、ルート要素からそのノードに到るまでの全ての状態変数を可変長配列として記憶する。その結果、配列の先頭にはルート要素の状態変数が、配列の最後尾には自分のノードの状態変数が格納されるという構造になる。「親へ移動」の機能は、状態変数配列の最後尾の状態変数を取り除き、最後から２番目の状態変数を自分のノードの状態変数とすることで実現できる。また前述した「子への移動」機能では、子への移動の結果の状態変数を状態変数配列の最後尾に追加する。「弟への移動」機能では、配列の最後尾の状態変数を弟への移動の結果の状態変数に置き換える。これにより「親へ移動」機能が実現できる。
【０１４６】
このような状態変数配列をノードに持たせると速度低下の原因となる。そこで、「親へ移動」の機能が必要ない場合は、状態変数配列を持たないバージョンのノードを利用することにより高速化を図ることができる。
【０１４７】
また、「子に移動」、「弟に移動」のアルゴリズムを利用して、「指定タグ名を持つ最初の子に移動」、「指定タグ名を持つ最初の弟に移動」、「指定タグ名を持つ次の弟に移動」という高機能ノード移動機能が実現できる。
この実現手段としては、ノード３２０７の保有する情報として、さらに検索対象タグ名を付加する。「指定タグ名を持つ最初の子に移動」を実現するには、「子に移動」を実行し、検索対象タグ名と移動したノードのタグ名を比較し、それが一致するまで繰り返し「弟に移動」を実行すればよい。「指定タグ名を持つ最初の弟に移動」を実現するには、「弟に移動」を実行し、検索対象タグ名と移動したノードのタグ名を比較し、それが一致するまで繰り返し「弟に移動」を実行すればよい。「指定タグ名を持つ次の弟に移動」を実現するには、前回の検索で設定した検索対象タグ名を保持しておき、検索対象タグ名とノードのタグ名が一致するまで繰り返し「弟に移動」を実行すればよい。
【０１４８】
同様に、「指定タグ名を持つ最初の孫に移動」、「指定タグ名を親に持つ最初の孫に移動」、「指定タグ名を親に、別の指定タグ名を持つ最初の孫に移動」、「指定条件の次の孫に移動」という二階層のノード移動機能も実現できる。この実現手段としては、ノード３２０７の保有する情報として、検索対象タグ名に加え、検索対象親タグ名を付加し、判定対象ノードのタグ名・親タグ名と検索対象タグ名・検索対象親タグ名を比較すればよい。
【０１４９】
次に、図３７を用いて、ＸＭＬ文書のタグ名の判定を文字列ではなく識別子で判定する手

段について説明する。例えばＸＭＬ文書３７０１で、タグ＜Ａ＞が多数存在する状況において、タグ＜Ａ＞の下のタグ＜Ｂ＞を検索し、そのテキスト要素を取得し、何らかのユーザ処理を行う場合を考える。従来の検索方式では、タグ名文字列による判定３７０３を行っていた。すなわち、ブロック３７０５において、ユーザの指定したタグ名の文字列と、ＸＭＬ文書３７０１に記述された文字列との文字列比較を行う。本発明が提供する識別子による判定方式３６０４では、あらかじめ検索したいタグ名のスキーマ要素識別子を取得しておき（３６０６）、ループの中のブロック３６０７において、ユーザの取得したスキーマ要素識別子と、ＢＸＭＬ文書３６０１の参照スキーマに記述されるスキーマ要素識別子とを比較する。スキーマ要素識別子は、図１４のスキーマ要素識別子１４０４の説明で示したとおり数値で表現できるため、文字列比較より高速な比較が可能となる。

【０１５０】

ＸＭＬの名前空間に対応すると、速度差はより顕著となる。図３６０２に示す名前空間に対応したＸＭＬ文書において、タグ＜Ｂ＞を判定する場合を考える。タグ＜Ｘ：Ｂ＞　３７１１、タグ＜Ｙ：Ｂ＞　３７１２、タグ＜Ｚ：Ｂ＞　３７１３の三つのタグにおいて、タグ＜Ｘ：Ｂ＞　３７１１とタグ＜Ｙ：Ｂ＞　３７１２は異なるタグ、タグ＜Ｘ：Ｂ＞　３７１１、タグ＜Ｚ：Ｂ＞　３７１３は同じタグであると判定しなければならない。従来の判定手段は、タグ＜Ｘ：Ｂ＞　３７１１の修飾子「Ｘ」の名前空間定義３７０８とタグ＜Ｙ：Ｂ＞　３７１２の修飾子「Ｙ」の名前空間定義３７０９が異なるためタグ＜Ｘ：Ｂ＞　３７１１とタグ＜Ｙ：Ｂ＞　３７１２は異なるタグ、タグ＜Ｘ：Ｂ＞　３７１１、タグ＜Ｚ：Ｂ＞　３７１３は修飾子「Ｘ」「Ｚ」が異なるが、その名前空間定義３７０８と３７１０は同じであるため同じタグ、と判定していた。このような処理をブロック３７０５で多数回行うと時間がかかる。

一方、本方式では、タグ＜Ｘ：Ｂ＞　３７１１とタグ＜Ｚ：Ｂ＞３７１３のスキーマ要素識別子は同じ値、＜Ｙ：Ｂ＞　３７１２のみ異なる値となる。ループの外の処理ブロック３７０６で１回のみ名前空間の判定を行い、多数回繰り返されるループ　内の処理ブロック３７０７でスキーマ要素識別子の比較だけを行うことにより、高速なタグ名判定が実現できる。

【０１５１】

また、ＤＯＭパースブロック１２４の応用例として、図３で説明した標準ＸＭＬ文書解析インタフェースであるＤＯＭインタフェースの他に、特定の分野に特化した高機能なインタフェースを提供することも考えられる。高機能なインタフェースの実現方式として、図３のノード３０４のように、ＸＭＬ文書の一つのタグに対応したノードではなく、意味を持ったタグの集合を一つのノードとして管理し、ノード移動機能を提供することが考えられる。

【０１５２】

図３８を用いて、地理情報分野に特化したノードの説明を行う。図３８は、図６と同様、地理情報分野のＸＭＬ文書の例であり、一枚の地図を表す。地図である地物集合を表すＭａｐタグの孫の位置に、個々の地物を表すＲｏａｄタグ３８０１、３８０７、Ｈｏｕｓｅタグ３８０８が存在する。Ｒｏａｄタグ３８０１に着目すると、孫の位置に道路中心線（ｃｅｎｔｅｒＬｉｎｅＯｆ）を表す折線図形（ＬｉｎｅＳｔｒｉｎｇ）タグ３８０２、交差点（ｊｕｎｃｔｉｏｎ）を表す点集合（ＭｕｌｔｉＰｏｉｎｔ）タグ３８０３が存在する。さらに点集合タグ３８０３の孫の位置には、点集合を構成する個々の点（Ｐｏｉｎｔ）タグ３８０４、３８０５、３８０６が存在する。

【０１５３】

ここで、地物を表す地物ノード、図形を表す図形ノードを定義する事が出来る。ノード３８０１、３８０７、３８０８は地物ノードである。また、ノード３８０２、３８０３、３８０４、３８０５、３８０６は図形ノードである。地物ノードの機能として、「最初の地物ノードに移動」、「最初の道路地物ノードに移動」、「次の地物ノードに移動」、「地物に含まれる最初の道路中心線を表す折線を示す図形ノードを取得」、「地物に含まれる最初の折線図形ノードを取得」、「次の折線図形ノードに移動」などのインタフェースが

実現できる。また図形３８０３のような複数の図形を含む図形に対し、「図形に含まれる最初の図形ノードを取得」などが定義できる。これらのインタフェースは、前述の「子に移動」「孫に移動」等の基本インタフェースを組み合わせることにより実現できる。
さらに地物ノード、図形ノードのように、地理情報分野に特化したノードでは、汎用の構造化文書が持たない特徴を利用したインタフェースが構築できる。例えば地物ノード３８０１は、道路・家屋・河川など利用者が自由に定義できるが、図形ノード３８０２は、点・折線・ポリゴンなど限られた固定個数の種類に限られる。この性質を利用して、「地物に含まれる最初の図形ノードに移動」というインタフェースが実現できる。汎用の構造化文書に対するノードではタグ名を指定する必要があったが、「地物に含まれる最初の図形ノードに移動」というインタフェースは、あらかじめ固定個数の図形タグを記憶しておき、判定時にその全ての図形タグと比較を行うことにより実現できる。また、「地物の属性を取得する」、「図形の座標系を取得する」、「図形の重心を取得する」等の地理情報分野に特有なインタフェースを追加することにより、地理情報分野のＸＭＬ文書を効率的に処理することの出来るインタフェースを提供することが出来る。

【０１５４】
次に、図１のＤＯＭ更新ブロック１２６について詳細に説明する。ＤＯＭ更新ブロック１２６の目的は、既存のＢＸＭＬ文書の更新を行うことである。

【０１５５】
ＢＸＭＬ文書の更新は、図３２で説明したＤＯＭパースブロック１２４のインタフェースを用いて更新対象ノードに移動し、ＤＯＭ更新ブロック１２６が提供するインタフェースである、（１）属性値変更、（２）テキスト要素変更、（３）属性の挿入、（４）属性の削除、（５）子要素の挿入、（６）子要素の削除、（７）子要素の置換を実行することにより行う。基本的に、テキスト要素変更、子要素の置換は、子要素の削除・子要素の挿入で表現することが出来る。また属性値変更、属性の挿入、属性の削除は、本発明では属性は子要素の特別な形としているため、同様に子要素の削除・子要素の挿入で表現することが出来る。そこで、子要素の挿入、子要素の削除方式について説明する。

【０１５６】
更新ブロックの実装は様々な方式が考えられるが、更新機能に必要な条件として、次の３点が考えられる。（１）多数のノードを高速に更新できること、（２）更新のやりなおし（ＵＮＤＯ）ができること、（３）更新情報を差分として抽出できること。特に三番目の更新差分は、地理情報サーバが地図を端末に配信し、端末側で地図を編集・更新し、それを地理情報サーバに反映するという利用を考えると、端末が更新差分のみをサーバに送信することにより通信コストが大きく低減するため、重要である。以下では、これらの条件を満たす更新方式について説明する。

【０１５７】
図３９を用いてＤＯＭ更新ブロック１２６の説明を行う。初めに、「子要素の挿入」の例として、ＸＭＬ文書３９０１をＸＭＬ文書３９０２に更新する場合を考える。これは二つのタグＣの間に新たなタグＣが挿入された状態を示す。

【０１５８】
「子要素の挿入」機能の実装例を２つ示す。第１の実装例を３９０７に示す。挿入されたタグＣのテキスト要素３９２２が挿入され、タグＣの重複数３９２１が元のデータ３９０６と比べ１増加している。しかしこの方式では、更新時に更新データ実体３９２２の挿入と重複数３９２１の更新と二カ所の変更が必要となり、複数のノードを同時更新する場合に処理が煩雑となり、処理速度が落ちる。また更新後にやりなおしや差分抽出が出来ない。

【０１５９】
「子要素の挿入」機能の第２の実装例を３９０８に示す。３９０８は更新されたＸＭＬ文書３９０２を表現する圧縮データである。挿入されたタグＣの部分に挿入データ３９１５を挿入する。挿入データ３９１５は、挿入トークン３９２３、挿入タグのスキーマ要素識別子３９２４、挿入データ実体３９２５から構成される。挿入トークン３９２３は、デー

タが挿入されたことを示すトークンである。スキーマ要素識別子３９２４は、挿入されたデータがタグＣであることを示す識別子であり、図１４のスキーマ要素識別子１４０４と同じものである。挿入データ実体３９２５は挿入されたノードの圧縮データであり、３９２２と同じものである。

【０１６０】

スキーマ要素識別子３９２４を設ける理由は、挿入されたノードが子要素群の最後に存在する場合への対処である。ＸＭＬ文書３９０３はタグ群Ｃの最後に新しいタグＣを挿入している。これを表す圧縮データは、圧縮データ３９０９における挿入データ３９１６である。また、ＸＭＬ文書３９０４は、タグＣの親タグＢの最後に新しいタグＢを挿入している。これを表す圧縮データは、圧縮データ３９１０における挿入データ３９１７である。挿入データ３９１６と挿入データ３９１７は、スキーマ要素識別子がないと挿入位置の区別が付かない。これを区別するために挿入タグ識別子３９２４を設けている。

【０１６１】

次に、「子要素の削除」の例として、ＸＭＬ文書３９０１をＸＭＬ文書３９０５に更新する場合を考える。これは、二番目のタグが削除された状態を示す。

【０１６２】

「子要素の削除」機能の実装例を４つ示す。第１の実装例を３９１１に示す。元のデータ３９０６と比較し、重複数Ｎｃが１減少しており、また２番目のタグの情報ｃ２が削除されている。これも挿入の場合と同様な問題を持つため、別の方式が必要となる。

【０１６３】

「子要素の削除」機能の第２の実装例を３９１２に示す。これは削除ノードの情報全体を削除トークン３９２６で置き換えるという方式である。この方式は、データがコンパクトになり、また重複数Ｎｃを更新する必要がないので高速な削除処理を行うことが出来る。

【０１６４】

「子要素の削除」機能の第３の実装例を３９１２に示す。これは削除ノードの前方に削除トークン３９２７を挿入するという方式である。削除対象のデータｃ２は、データ３９２６のように実際は削除されていない。この方式は、削除データ情報全体を削除トークン３９２６で置き換えるという方法である。この方式の利点は、削除されるノードの情報が残されるため、「更新のやりなおし」や更新差分抽出が行えることである。

【０１６５】

「子要素の削除」機能の第４の実装例を３９１４に示す。これは削除ノードの位置に削除トークン３９２８と、削除ノードのサイズ３９２９を記述するという方式である。削除ノードの残りの情報はごみ３９３０として残るが、利用者は削除トークン３９２８が出現したら３９２９で示されるサイズだけデータの読み込みをスキップすることでごみ３９３０を無視することが出来る。この方式の利点は、更新処理が高速となるという点である。実装例３９１１、３９１２、３９１３はいずれも、「子要素の削除」処理により削除データ部分のサイズ変更が発生し、削除データより後方の圧縮データの位置をシフトするためにコピーが発生する。第４の実装例では、削除トークン３９２８と削除サイズ３９２９の合計が元のデータよりも大きくなるというわずかな例外を除いて、圧縮データの位置シフトが発生しないため、全体的な更新パフォーマンスの向上につながる。

【０１６６】

上記に示した手法で更新を行う場合、圧縮データに挿入トークン・削除トークン等の余分なデータが蓄積されていく。そのため定期的にガベージコレクションを実施することにより挿入トークン・削除トークンを削除する。

【０１６７】

図４０を用いて、挿入・削除されたノードに対応したＢＸＭＬパースブロック１０８のアルゴリズムを説明する。これは図３１で説明したＢＸＭＬパースブロック１０８を拡張したものである。

【０１６８】

ＳＴＥＰ　１～ＳＴＥＰ　１４までの処理は図３１のＢＸＭＬパースブロック１０８と同

一であるため、説明を割愛する。ここでは、新たに挿入されたＳＴＥＰ１５～ＳＴＥＰ１８について説明を行う。
［ＳＴＥＰ　１５］　本ステップでは、図３９のＸＭＬ文書３９０２のように、新しいノードが重複する子要素群の最後尾を除く任意の位置に挿入された場合の処理を行う。ＳＴＥＰ　２で子要素の重複数だけノード処理を実行するが、挿入された子要素は重複数には含まれないため、重複カウンタを進めてはならない。圧縮データからデータを取得し、それが挿入トークン３９２３である場合、重複カウンタを進めずに繰り返しノード処理を実行する。
［ＳＴＥＰ　１６］　図３９のＸＭＬ文書３９０３のように、新しいノードが重複する子要素群の最後尾に挿入された場合の処理を行う。圧縮データからデータを取得し、それが挿入トークン３９２３であり、スキーマ識別子３９２４が現在の参照要素と同一である限り、繰り返しノード処理を実行する。
［ＳＴＥＰ　１７］　本ステップは、子要素が削除された場合に対応する。圧縮データからデータを取得し、それが削除トークン３９２６の場合、圧縮データの参照ポインタを削除トークンの次に合わせ、ノード処理を終了する。
［ＳＴＥＰ　１８］　本ステップは、属性が削除された場合に対応する。圧縮データからデータを取得し、それが削除トークン３９２６でない場合のみ、ＳＴＥＰ　６以降を実行する。
【０１６９】
次に、図１のＢＸＭＬ作成ブロック１２１について詳細に説明する。ＢＸＭＬ作成ブロック１２１の目的は、ＢＸＭＬ文書を直接作成することである。ＢＸＭＬ作成ブロック１２１は、図２０で説明したとおり、データ圧縮ブロック１１２に対し図２に示したＳＡＸイベントを発行することにより、圧縮データを直接作成する。ＢＸＭＬ作成ブロック１２１の機能は、ユーザプログラム１２０にＢＸＭＬ作成のためのインタフェースを提供し、これをデータ圧縮ブロック１１２が受け付けるＳＡＸイベントに変更することである。
【０１７０】
図４１に、ＢＸＭＬ作成ブロック１２１がユーザプログラム１２０に提供するインタフェースを示す。ＢＸＭＬ作成ブロック１２１のインタフェースは、以下の１２個のメソッド、すなわち、文書作成系の（１）文書開始メソッド、（２）文書終了メソッド、ノード作成系の（３）子要素作成メソッド、（４）弟要素作成メソッド、（５）親へ移動メソッド、（６）先祖へ移動メソッド、データ作成系の（７）属性の作成メソッド、（８）文字列の作成メソッド、（９）整数の作成メソッド、（１０）実数の作成メソッド、（１１）空文字の作成メソッド、（１２）ノードのコピー、から構成される。
（１）文書開始メソッド４１０８は、ＢＸＭＬ文書を作成開始する時に実行する。文書開始メソッドは、ＢＸＭＬ文書作成のための初期処理を行う。これはデータ圧縮ブロック１１２に対し、図２の文書開始処理２０１を呼び出すことで実現する。
（２）文書終了メソッド４１０９は、ＢＸＭＬ文書の作成を完了する時に実行する。文書終了メソッドは、ＢＸＭＬ文書の終了タグのないタグ全てに終了タグを付加し、ＢＸＭＬ文書作成の終了処理を行う。これは、スキーマ木の参照要素から親要素をさかのぼり、ルート要素に到るまで繰り返しデータ圧縮ブロック１１２に対し終了タグ処理２０４を呼び出し、その後文書終了処理２０２を呼び出すことで実現する。
（３）子要素作成メソッド４１１０は、作成途中のＢＸＭＬ文書における現在の位置に、新たな子要素を追加する時に実行する。子要素作成メソッドは、引数で指定したタグ名の子要素をＢＸＭＬ文書に追加する。これは、データ圧縮ブロック１１２に対し、引数で指定されるタグ名を用いて開始タグ処理２０３を呼び出すことにより実現できる。ここで、作成途中のＢＸＭＬ文書は、図４１の４１０４と４１０５に示す２つの状態がある。これをそれぞれ開状態、閉状態と呼ぶことにする。開状態４１０４は、最後に開始タグが記述されている状態であり、閉状態４１０５は最後に終了タグが記述されている状態である。子要素作成メソッドは開状態４１０４で実行されることを前提としており、閉状態４１０５で実行するとエラーを返す。また別の実装方式として、閉状態４１０５で実行された時

に、現在のノードＢと同じ弟ノードＢを作成し、その弟ノードＢに対して子ノードＣを追加しても構わない。この場合、スキーマ木の参照要素のタグ名を用いて開始タグ処理２０３を呼び出し、その後引数で指定されるタグ名Ｃを用いて開始タグ処理２０３を呼び出すことにより実現できる。

（４）弟要素作成メソッド４１１１は、作成途中のＢＸＭＬ文書におけるの現在の位置に、新たな弟要素を追加する時に実行する。弟要素作成メソッドは、引数で指定したタグ名の弟要素をＢＸＭＬ文書に追加する。これは、閉状態４１０７において、データ圧縮ブロック１１２に対し、引数で指定されるタグ名を用いて開始タグ処理２０３を呼び出すことにより実現できる。開状態４１０６で実行するとエラーを返す。別の実装として、開状態４１０６で実行された時に、現在のノードを空タグとして閉じ、引数で指定したタグ名の弟要素をＢＸＭＬ文書に追加しても構わない。この場合、スキーマ木の参照要素のタグ名を用いて終了タグ処理２０４を呼び出し、その後引数で指定されるタグ名Ｃを用いて開始タグ処理２０３を呼び出すことにより実現できる。

（５）親へ移動メソッド４１１２は、現在のノードの書き込みを終了し、現在のノードから見て親の弟、あるいは親の親の弟など、上位階層のノードの書き込みを開始する直前に実行する。親へ移動メソッドは引数を持たない。閉状態４１０９では親のノードを閉じる。これはスキーマ木の参照要素における親のタグ名を用いて終了タグ処理２０４を呼び出素ことにより実現できる。また開状態４１０８では、現在のノードを空タグとして閉じ、その後親のノードを閉じる。これはスキーマ木の参照要素のタグ名を用いて終了タグ処理２０４を呼び出し、その後に親のタグ名を用いて終了タグ処理２０４を呼び出素ことにより実現できる。

（６）先祖へ移動メソッド４１１３は、指定したノードまで親へ移動メソッドを繰り返し発行する。これはスキーマ木の参照要素から指定したタグ名のノードまで繰り返し遡って検索し、遡った回数を利用して親へ移動メソッドを繰り返し実行することにより実現できる。

【０１７１】

また、図４１のＢＸＭＬ作成ブロック１２１が提供するインタフェースの異なる実装として、子要素作成メソッド４１１０、弟要素作成メソッド４１１１、親へ移動メソッド４１１２、先祖へ移動メソッド４１１３で示されるノード作成メソッド群に対して直接データ圧縮ブロック１１２に対して処理を実行するのではなく、一時的なバッファにメソッドを記憶しておき、属性の作成メソッド４１１４、文字列の作成メソッド４１１５等のデータ作成メソッド群が発行された時点で一度に記憶しておいたノード作成メソッド群を発行するという方式が考えられる。例えば０個以上の子要素を作成したい利用者は、最初に子要素作成メソッド４１１０を発行し、文字列の作成メソッド４１１５などのデータ作成メソッドと弟要素作成メソッド４１１１を作成したい子要素数と同数回発行し、最後に親へ移動メソッド４１１２を発行する。この際、作成すべき子要素が０個であった場合、本発明の最初の実装では、子要素作成メソッド４１１０が必ず発行されるため、不要な子要素が出現してしまう。ここでノード作成メソッド群を一時的なバッファに記憶しておき、子要素作成メソッド４１１０を発行後、データ作成メソッド群が一度も発行されることなく親へ移動メソッド４１１２が発行された場合、バッファに記憶された最後の子要素作成メソッド４１１０をキャンセルする動作を実現すれば、上記のような不要な子要素の出現を抑制することができる。

（７）属性の作成メソッド４１１４は、作成途中のＢＸＭＬ文書におけるの現在のノードに、指定したキーと値を持つ属性を挿入する。これは指定したキーと値を用いてデータ圧縮ブロック１１２に対し、属性処理２０６を呼び出すことにより実現できる。属性の作成メソッドは開状態４１１２でのみ許可され、閉状態ではエラーとなる。

（８）文字列の作成メソッド４１１５は、作成途中のＢＸＭＬ文書におけるの現在のノードに、テキスト要素として指定した文字列を挿入する時に使用する。これは指定した文字列を用いてデータ圧縮ブロック１１２に対し、テキスト要素処理２０５を呼び出し、その後にスキーマ木の参照要素のタグ名を用いて終了タグ処理２０４を呼び出すことにより実

現できる。文字列の作成メソッドは開状態４１１２でのみ許可され、閉状態ではエラーとなる。
（９）整数の作成メソッド４１１６は、作成途中のＢＸＭＬ文書におけるの現在のノードに、テキスト要素として指定した整数を挿入する。これは文字列の作成メソッドにおけるテキスト要素処理２０５の呼び出しの代わりに、指定した整数を用いて整数要素処理を呼び出すことにより実現できる。整数要素処理とテキスト要素処理２０５の内部処理は全く同じであり、引数として整数を用いることにより余分な文字列変換を実行しない、高速化のためのメソッドである。
（１０）実数の作成メソッド４１１７は作成途中のＢＸＭＬ文書におけるの現在のノードに、テキスト要素として指定した整数を挿入する。実現手段は整数の作成メソッドと同様である。またここでは整数と実数の場合のみを示したが、図４の「データ型の適用」圧縮４０２で説明した多数のデータ型についても同様なメソッドが定義できる。
（１１）空タグの作成メソッド４１１８は、現在のタグを閉じる時に使用する。これは例えば属性のみを持ち、子要素やテキスト要素を持たないタグを記述する場合に利用する。これはスキーマ木の参照要素のタグ名を用いて終了タグ処理２０４を呼び出すことにより実現できる。
（１２）ノードのコピーメソッド４１１９は、現在のタグ位置に別のＢＸＭＬ文書のノード群を挿入する時に利用する。
【０１７２】
図４２を用いてノードのコピーメソッド４１１９を詳細に説明する。作成中文書４２０１に対し、挿入元文書４２０３の中の、２つのタグＢで構成されるノード群４２０４を挿入し、文書４２０２にする例を考える。図２８で説明したとおり、データ圧縮ブロック１１２ではスキーマ木２８０２を利用して圧縮データ２８０３を作成していく。そのためコピーメソッド４１１９の実装は、スキーマ木の挿入対象タグＢのスキーマ要素４２０５、作成中の圧縮データ４２０８に対し、挿入元圧縮データ４２１０の２つのタグＢで構成されるノード群を示す圧縮データ４２１４を挿入し、スキーマ要素４２０６、圧縮データ４２０９にする操作ということになる。これは以下のステップで実行する。
［ＳＴＥＰ　１］　挿入するノード数（本例の場合２）を、挿入先スキーマ要素４２０５の一時変数である重複カウンタに加える。
［ＳＴＥＰ　２］　圧縮データ４２１４を作成中の圧縮データにコピーする。
【０１７３】
本例では、挿入元圧縮データ４２１４であるタグＢは単純な文字列型としたが、タグＢがさらに子要素を持つ複雑な構造を持っていても同様にコピーすることが出来る。また、挿入元文書４２０３のタグＢのスキーマ木が挿入先文書のタグＢのスキーマ木と異なる場合、挿入元圧縮データ４２１４を単純にコピーするのではなく、挿入先文書のタグＢのスキーマ木に一致する構造に変換を行った後にコピーを行うことになる。このような構造変換は、図１のＢＸＭＬパースブロック１０８を用い、挿入元文書のスキーマ木１０２と挿入先文書のスキーマ木型変換ブロックを実装することにより実現できる。
【０１７４】
またノードのコピーメソッド４１１９を利用して、複数のＢＸＭＬ文書を結合する処理を実現することができる。結合対象の複数のＢＸＭＬ文書の圧縮スキーマが等しい場合、図２０で示した生成処理１３０５を用いて同じ圧縮スキーマを持つ新たなＢＸＭＬ文書を作成する。スキーマ木１０２はあらかじめ用意しておいたＢＸＭＬ文書１０５からスキーマ展開ブロック１１１を経て作成する。そして結合したいノード群に対し、ノードのコピーメソッド４１１９を利用して圧縮データ４２１４を作成中の圧縮データにコピーすればよい。結合対象の複数のＢＸＭＬ文書の圧縮スキーマが異なる場合、図２０で示したスキーマパース処理１１７、スキーマ更新処理１１９を利用して結合する圧縮スキーマの双方の構造を含んだ最小公倍数的な圧縮スキーマを作成し、さらに結合したいノード群に対し、ノードのコピーメソッド４１１９を利用して、挿入先文書のスキーマ木に一致する構造に変換を行った後にコピーを行うことにより実現できる。

【０１７５】
次に、図１のスキーマパースブロック１１７について詳細に説明する。スキーマパースブロック１１７の目的は、スキーマ木の移動インタフェースをユーザに提供することである。
【０１７６】
スキーマパースブロック１１７は、ユーザプログラム１１８に対し、以下の３つの移動系メソッドを提供する。（１）親へ移動、（２）長男に移動、（３）次の弟に移動。これは図２５に示す参照要素２５０４を用いて実装する。例えば「親へ移動」メソッドは、図１４で説明した「親スキーマ要素へのポインタ」を利用し、参照要素２５０４を親に移動させる。「長男に移動」メソッドは、「子スキーマ要素へのポインタ」を利用し、参照要素２５０４を最初の子に移動させる。「次の弟に移動」メソッドは、親要素の持つ期待要素２５０５を一つ後ろの要素に移動させ、これを参照要素とする。また、スキーマパースブロック１１７は、図１４に示すスキーマ木の各変数を参照する機能を提供する。これにより、「指定したタグ名の子に移動」「指定したタグ名の弟に移動」メソッドは、スキーマ要素のタグ名１４０５が一致するまで「次の弟へ移動」メソッドを実行することにより実現できる。
【０１７７】
次に、図１のスキーマ更新ブロック１１９について詳細に説明する。スキーマ更新ブロック１１９の目的は、スキーマの変更を行うことにより、ＢＸＭＬ文書の一括変換を実現することである。
【０１７８】
スキーマ更新ブロック１１９は、上述したスキーマパースブロック１１７を用いて、更新の必要なスキーマ要素に移動し、図１４に示すスキーマ要素の状態変数を修正する機能を提供する。
【０１７９】
スキーマ要素１４０１の、タグ名１４０５を変更することにより、ＸＭＬ文書全体の指定した開始タグ・終了タグを一括変更することができる。ＸＭＬ文書内に同じ名称のタグが多数存在していても、本機能によりスキーマ木の１箇所を修正するだけで全てのタグを変更することができるため、高速な変換を実現することができる。
【０１８０】
これは例えば名前空間の変換に利用できる。図３７のＸＭＬ文書３７０２において、タグ＜Ｘ：Ｂ＞　３７１１に対し、名前空間修飾子のない形のタグ＜Ｂ＞に変換する場合、また異なる修飾子＜Ｚ：Ｂ＞に変換する場合、本機能によりスキーマ木の１箇所を変更することで全てのタグ＜Ｘ：Ｂ＞の変更ができる。
【０１８１】
また、特定のスキーマ要素に隠蔽の設定を行うことにより、ＸＭＬ文書の一部分を隠蔽することができる。図１４のスキーマ要素１４０１において、新たに隠蔽フラグを格納する。スキーマ更新ブロック１１９により隠蔽フラグをＴＲＵＥにしておき、ＢＸＭＬパースブロック１０８において隠蔽されたスキーマ要素を持つタグをスキップすることにより隠蔽機能は実装できる。それにより例えば、図４３のＸＭＬ文書４３０１において、Ｒｏａｄタグ以外を隠蔽し、ＸＭＬ文書４３０２に変更することができる。またＸＭＬ文書４３０３において、Ｒｏａｄタグの子要素であるＬａｎｅタグ、Ｐａｖｅｍｅｎｔタグを隠蔽し、ＸＭＬ文書４３０４に変更することが出来る。
【０１８２】
隠蔽機能により、ＸＭＬ文書の高速な加工処理が行える。例えば地理情報サーバがＸＭＬ文書４３０１を保有している時、地理情報クライアントが地理情報サーバに対し、道路を示すＲｏａｄのみを要求する場合を考える。地理情報サーバがＸＭＬ文書４３０１を加工し、ＸＭＬ文書４３０１を作成するには時間がかかるが、隠蔽機能を用いることにより、ＸＭＬ文書内に同じ名称のタグが多数存在していても、スキーマ木の１箇所を修正するだけで全ての関連タグを隠蔽することができるため、高速な加工を実現することができる。

【０１８３】
図４４を用いて、隠蔽されたノードに対応したＢＸＭＬパースブロック１０８のアルゴリズムを説明する。これは図３１で説明したＢＸＭＬパースブロック１０８を拡張したものである。
【０１８４】
ＳＴＥＰ　１、ＳＴＥＰ　２、並びにノード処理は図３１と同一であるため、説明を割愛する。ここでは、新たに挿入されたＳＴＥＰ　３では、現在参照しているスキーマ要素の隠蔽フラグを確認し、隠蔽状態であればノードのスキップを行う。
【０１８５】
以上により、図１の全処理ブロックの詳細な説明を終了する。次に、本発明の実施例として、圧縮ＸＭＬを利用した地理情報システムについて説明する。
【０１８６】
図４５を用いて、本発明の実施形態について説明する。図４５は一般的なインターネットのＷｅｂサービスの構成を示す三階層モデルで構成された地理情報システムである。
三階層モデルは、ユーザが利用する端末であるクライアント層４５０１、データを加工するアプリケーション層４５０２、データを格納するデータベース層４５０３、から構成される。
【０１８７】
クライアント層４５０１は、多数の一般ユーザが利用する汎用クライアント４５０４と、少数の特定ユーザが利用する専用クライアント４５０６に分類できる。汎用クライアントでは主に、地図の参照や利用を行う。汎用クライアントは多数の一般ユーザが利用するため、複数のベンダ製品での実装が前提となる。そのため、汎用クライアントとサーバ間のデータ形式や通信プロトコルは標準化がされている必要がある。従来の技術で説明したとおり、標準化されたデータ形式として、ＪＩＳ規格のＧ－ＸＭＬやＯＧＣのＧＭＬ、標準化された通信プロトコルとして、ＯＧＣのＷＦＳ、ＷＭＳが存在する。一方専用クライアント４５０６では主に、地図の保守や更新を行う。専用クライアントのユーザはデータベース層４５０３やアプリケーション層４５０２の管理者や、それに近い組織のユーザとなるため、専用クライアントはデータベース層４５０３やアプリケーション層４５０２と同じベンダ製品での実装も可能であり、必ずしも標準化がされている必要はない。
通信環境としては、一般に、アプリケーション層４５０２とクライアント層４５０１の間は電話回線や無線回線などの低速回線４５０９で、アプリケーション層４５０２とデータベース層４５０３はＬＡＮ（Ｌｏｃａｌ　Ａｒｅａ　Ｎｅｔｗｏｒｋ）などの高速回線４５１２で接続される。ただし例えば県庁向け地理情報システムにおいて、地図データが各市町村のデータベースに分散され、県庁がゲートウェイとなり、各市町村のデータベース層を統合して検索する場合など、データベース層４５０３とアプリケーション層４５０２が異なる組織に分散し、組織間が低速回線４５１３で接続されている場合もある。
【０１８８】
本発明によるＸＭＬ文書の圧縮構造であるＢＸＭＬ文書をデータベース層、アプリケーション層、クライアント層に適用する方法について説明する。
【０１８９】
本発明は、アプリケーション層４５０２に適用できる。一般にＷｅｂサービスでは、データベース層・アプリケーション層で大量のデータのやり取りを行い、大量の複雑な計算を行い、その結果として少量のデータをクライアント層に返戻する、というパターンを取る。データベース４５１５からＢＸＭＬ文書を取得し、アプリケーション４５１０において本発明の解析処理１３０３を用いることにより、大量のデータを用いた大量の複雑な計算処理を高速に実現することが出来る。例えば地図検索サービスでは、ユーザが指定する領域の地図をデータベース４５１５から検索し、アプリケーション４５１０において「道路は赤色で描画する」などのユーザが指定する描画規則で地図を描画し、地図画像として汎用クライアント４５０４に返戻する方式が考えられる。
多数の一般ユーザが利用する汎用クライアント４５０４と異なり、データベース層とアプ

リケーション層は同一のベンダでシステム構築を行うことが比較的容易である。そのため標準化されたＸＭＬ文書ではなく、独自仕様である本発明のＢＸＭＬ文書を用いてもかまわない。またＢＸＭＬ文書はＸＭＬ文書を圧縮した構造であるため、ＸＭＬ文書のメリットである自由度や拡張性を全て継承し、また可逆圧縮であるため変換により情報が劣化することはない。上述した県庁ゲートウェイと市町村データベースの場合のようにデータベース層４５０３とアプリケーション層４５０２が低速回線の場合も、コンパクトなＢＸＭＬ文書を利用することにより通信時間を削減することが出来る。

データベース層４５０３とアプリケーション層４５０２が低速回線４５１３で接続されており、かつデータベース４５１５やアプリケーション４５１０が標準のＸＭＬ文書の入出力しか出来ない場合、ブリッジ４５１１、４５１４を利用することにより、途中の低速通信回線４５１３をコンパクトなＢＸＭＬ文書で通信を行うことにより通信時間を削減することが出来る。ブリッジは、たとえばＨＴＴＰ（Ｈｙｐｅｒ　Ｔｅｘｔ　Ｔｒａｎｓｆｅｒ　Ｐｒｏｔｏｃｏｌ）通信におけるＰＲＯＸＹサーバとして実装される。データベース４５１５が標準のＸＭＬ文書を出力する時、ブリッジ４５１４は圧縮処理１３０１を実行することにより入力文書をＢＸＭＬ文書に圧縮し、低速回線４５１３において高速に通信を行う。ＢＸＭＬ文書は、ブリッジ４５１１で展開処理１３０２を実行することにより元のＸＭＬ文書に展開され、ＸＭＬ文書のみを受け付けるアプリケーション４５１０に入力される。

【０１９０】

また本発明は、専用クライアント４５０６とアプリケーション４５１０との間の通信に利用できる。クライアントとアプリケーション層の間が電話回線や無線通信など低速回線の場合、サイズのコンパクトな本発明のＢＸＭＬ文書を通信に利用することにより、通信時間の短縮を実現することが出来る。また専用クライアント４５０６は保守・更新など、大量のデータを利用する場合が多く、本発明の解析処理１３０３、更新処理１３０４により高速にデータ解析・更新を実現することが出来る。多数の一般ユーザが利用する汎用クライアント４５０４と異なり、専用クライアント４５０６はデータベース層やアプリケーション層と同一のベンダでシステム構築を行うことが比較的容易である。そのため標準化されたＸＭＬ文書ではなく、独自仕様である本発明のＢＸＭＬ文書を用いることもできる。また現在Ｗｅｂブラウザなどにおいて、Ｊａｖａ　やＭｉｃｒｏｓｏｆｔ社のＡｃｔｉｖｅＸなど、クライアント側で実行するソフトウェアをサーバ側から動的にダウンロードして使用する方式が利用されている。このようなクライアントでは、標準化されたＸＭＬ文書ではなく、独自仕様である本発明のＢＸＭＬ文書を用いてもよい。

【０１９１】

また本発明は、クライアント層４５０１がモバイル端末の場合に利用できる。モバイル端末とアプリケーション層との間の通信は無線通信となり、低速回線となる。また通常のデスクトップ端末に対し、モバイル端末自身のメモリ容量は小さく、処理能力も低い。サイズのコンパクトな本発明のＢＸＭＬ文書を通信に利用することにより、通信時間の短縮を実現することが出来る。またモバイル端末に本発明の解析処理１３０３を組み込むことにより、小メモリ容量・低処理能力のモバイル端末でも地図表示・利用が可能となる。

【０１９２】

また本発明を、汎用クライアント４５０５に適用する方法について説明する。第１の方法は、本発明の装置４５０７を汎用クライアント４５０５に組み込み、標準板フェースを用いて接続する方法である。本発明は図１８で説明したとおり、解析処理１３０３として標準のＸＭＬ文書解析用インタフェースであるＳＡＸ、ＤＯＭを提供する。そのためＳＡＸ、ＤＯＭを利用した汎用クライアントでは、ＳＡＸ、ＤＯＭのパーサ部品を本発明の解析処理１３０３と入れ替えることにより本発明のＢＸＭＬ文書を解析する事が可能となる。

【０１９３】

本発明を汎用クライアント４５０５に適用する第２の方法は、本発明の装置をクライアント側のデコーダ４５０８として組み込む方法である。デコーダは、たとえばＨＴＴＰ通信におけるクライアント側のＰＲＯＸＹサーバとして実装される。アプリケーション４５１

０がＢＸＭＬ文書を出力する時、デコーダ４５０８は展開処理１３０２を実行することにより元のＸＭＬ文書に展開され、ＸＭＬ文書のみを受け付ける汎用クライアント４５０６に入力される。

【０１９４】

次に、図４５のデータベース層４５１５に対する本発明の実施例について説明する。

【０１９５】

データベース層４５１５の実装方式は、ＲＤＢ（Ｒｅｌａｔｉｏｎａｌ Ｄａｔａｂａｓｅ）を利用した方式と、図面を利用した方式の二種類に分類される。図４６を用いて、ＲＤＢを利用した方式に対し、本発明を適用した実施例について説明する。

データベース層４５１５の目的は、ＸＭＬ文書４６０２のような構造を持つデータを多数、ＲＤＢに格納し、検索条件に合致するデータを抽出して返戻することである。

ＸＭＬ文書のデータ管理方式は木構造であり、ＲＤＢのデータ管理方式は表構造である。そのため、木構造のＸＭＬ文書を表構造のＲＤＢに格納する場合何らかの対応を行う必要がある。例えばＸＭＬ文書４６０２のタグＢ、タグＣ、タグＤのように、一回しか出現しないことが保証されているタグは、テーブル４６０３の列Ｂ、Ｃ、Ｄのように列に展開することが出来る。しかしタグＥのように複数回重複して出現するタグや、さらにタグＦ、タグＧのように、重複タグの中に子要素が存在する場合、単一のテーブル４６０３では表現できない。そのためテーブル４６０３の子供である別テーブル４６０４を用意し、重複するタグを別テーブル４６０４の行として管理する。テーブル４６０３とテーブル４６０４の関連付けは、テーブル４６０３とテーブル４６０４において、列ｉｄで示される共通のキーを持ち、テーブル４６０４ではさらに、ｉｄ２で示される、重複数だけ存在するキーを設けることで実現できる。

【０１９６】

しかしこの方式は検索時間がかかる。ＸＭＬ文書が単一のテーブル４６０３のみで管理される場合、１回のＳＱＬ検索要求のみで条件に該当する全てのＸＭＬ文書４６０２を作成するのに必要な情報が収集できる。しかし重複タグＥを含むＸＭＬ文書４６０２のような例では、初めにテーブル４６０３の検索を行い、さらに検索結果の回数分テーブル４６０４の検索を行う必要がある。ＸＭＬ文書４６０２の構造が複雑化し、テーブル４６０４にさらに子供である別テーブルが関連するなどの結果、検索時間は加速度的に悪化する。

【０１９７】

ＲＤＢ４６０５の構成は、ＲＤＢ４６０１を改良した構成であり、本発明を利用して検索時間の高速化を行っている。子テーブル４６０８はテーブル４６０４と同一である。主テーブル４６０７は主テーブル４６０３に対し、新たな列として圧縮データ列４６０９が付加される構造を取る。圧縮データ列４６０９にはテーブル４６０７、４６０８で管理される情報をあらかじめＢＸＭＬの圧縮データ１０４としたものを格納する。情報の二重管理となるためＲＤＢ４６０６のデータサイズは増加するが、検索を高速化することが出来る。また圧縮データ列４６０９に格納するものは本発明によるコンパクトな構造を取っているため、ＸＭＬ文書をそのまま格納する場合と比較して全体のデータ量を削減することが出来る。

【０１９８】

ユーザからＸＭＬ文書４６０６の取得要求があった場合、テーブル４６０７の圧縮データ列４６０９を取得し、検索結果の取得された複数の圧縮データ４６０９を、図４２のノードコピー方式で結合し、一つのＢＸＭＬ文書として返戻することができる。なおその時に必要となる圧縮スキーマは別管理しておく。圧縮スキーマの数は、テーブル４６０７の数と同じ数だけ存在する。

【０１９９】

次に、図４７を用いて、図面を利用したデータベース層４５０３の実装方式に対し、本発明を適用した実施例について説明する。

【０２００】

クライアント層４５０１が地理情報を要求するパターンは、地物検索と地図検索の２つに

分類できる。
地物検索パターンは、例えば「指定した二点を結ぶ最短経路を取得せよ」「指定した道路の、騒音の恐れのある周辺５０ｍの範囲の領域に存在する全ての病院や幼稚園を抽出せよ」というように、ユーザが空間データの解析をデータベース層に依頼する場合に生じる。この検索パターンは検索条件が複雑であり、かつ返戻結果が比較的少ない、という特徴を持つ。また、比較的長い検索時間も許容される。これは図４６で説明したＲＤＢベースのデータベース層４５１５が得意とするパターンである。
一方、地図検索パターンは、「指定した矩形領域に含まれる地図を取得せよ」というように、データベース層に複雑な解析を依頼しない場合に生じる。取得した地図はクライアント層で地図として描画されたり、クライアント層側の機能で解析されたりする。この検索パターンは検索条件が比較的単純であり、かつ返戻結果が膨大となる、という特徴を持つ。またユーザがストレスなく地図の様々な領域をスクロールしたり拡大縮小したりするために、高速な検索時間が要求される。これは図面ベースのデータベース層４５１５が得意とするパターンである。
【０２０１】
図面ベースのデータベース層で地図を管理するためには、地図を図面に分割する必要がある。従来の方式では、地図を図面に分割する際、図面端の図形を分断していた。例えば図４７において、地図４７０１を図面４７０４と図面４７０５に分割する際、図面端の家屋図形４７０２は家屋４７０６と家屋４７０７に分断され、道路４７０３は道路４７０８と道路４７０９に分断されていた。これらの図面をクライアント層に取り込み、例えば家屋数を数える場合や道路ネットワーク網の解析を行う場合、分断図形の再結合という複雑な処理を必要とした。
【０２０２】
従来の方式で図面を分割していた理由の一つとして、図形構成点の座標の容量を低減するため、相対値管理を行いたいという点がある。座標を緯度経度などの絶対座標で管理すると、表現したい精度に応じて例えば６４ｂｉｔ整数などの大サイズの固定長整数で管理する必要があり、データ量が増大する。図面内の座標をローカル座標で管理することにより例えば１６ｂｉｔ整数などの小サイズの固定長整数で管理することが出来る。この副作用として、例えば図面４７１０の道路４７１３のように、図面枠を大きくはずれる座標は小サイズの固定長整数でオーバーフローする可能性があるため、図面枠を超える図形を分断する必要があった。
【０２０３】
本発明では、図１２に示したとおり、座標値を可変長数値で管理することが出来るため、図面枠を大きくはずれる座標のオーバーフローの心配はない。また図７の相対値の適用７０５で示したとおり、圧縮方式として相対値管理を行っており、図面レベルで明示的に相対値管理を行う必要はない。
そこで本発明を適用した図面ベースデータベース層４５１５では、図面分割の際、図形を分断しない。地図４７０１を図面４７１０と図面４７１１に分割する際、図面端の家屋図形４７０２は家屋４７１２、道路４７０３は道路４７１３としていずれかの図面に格納する。また座標は絶対座標をそのまま管理する。各図面は外接矩形４７１４を属性として持ち、図面内の全ての図形が外接矩形４７１４に含まれることを保証する。
【０２０４】
【効果】
本発明の効果は以下の通りである。
（１）本発明の構造化文書圧縮処理ならびに展開処理を利用することにより、巨大なサイズの構造化文書をその意味的構造を完全に保ったままサイズを約１／１０に削減できる。これにより、多量かつ巨大なサイズの構造化文書をデータベースシステムやファイルシステムに格納する場合、格納容量を低減することができる。またモバイル環境などの低容量通信路において巨大なサイズの構造化文書を伝送する場合、通信負荷を軽減することができ、通信時間を軽減することができる。また利用者の許容できる応答時間でより大量な情

報を伝達することができる。またモバイル機器など、記憶装置や二次記憶装置のサイズが小さいクライアントに対し、クライアントの記憶装置の負荷を軽減できる。
（２）本発明の構造化文書解析処理を利用することにより、巨大なサイズの構造化文書を従来の非圧縮構造化文書解析処理の約１０倍の速度で解析できる。また本発明の構造化文書解析処理は従来のＤＯＭ構造のように構造化文書を不要にメモリに展開しないため、巨大なサイズの構造化文書をより少ない記憶容量で解析できる。
（３）本発明の構造化文書更新処理を利用することにより、巨大なサイズの構造化文書を従来の非圧縮構造化文書更新処理より高速に更新できる。また本発明の更新処理では更新部分に挿入トークン、削除トークンが挿入されるため、更新後に更新位置を特定することができる。これにより、更新の取り消しを実現することができる。また、クライアント・サーバシステムにおいて、サーバから取得した巨大な構造化文書をクライアントにおいて更新した場合、更新部分のみを差分情報としてサーバに転送することにより、クライアント・サーバ間の通信負荷を軽減できる。
（４）本発明の構造化文書作成処理を利用することにより、巨大なサイズの構造化文書を非圧縮の形を経由せずに直接作成できる。そのため巨大なサイズの構造化文書を従来の非圧縮構造化文書の作成処理より高速かつ記憶装置の負荷をかけずに更新できる。
（５）昨今の地理情報の標準化により、空間データを構造化文書で表現する方法が提案されてきている。しかし地理情報分野では街の地図など多量の空間データを扱う場合が多く、従来の非圧縮の構造化文書ではサイズが巨大であり解析処理も遅く、実用的に使用することはできなかった。そのため構造化文書による標準方式は、処理時間を問わないオフラインでのデータベース間のインポートに利用されるのみで、ネットワークを介したクライアント・サーバ間では利用されなかった。本発明による構造化文書圧縮処理・解析処理を利用することにより、利用者が許容できる時間で構造化文書の伝送・解析が可能となる。従来の地理情報システムでは、クライアント単体の機能としては、地図の表示と交差判定や包含判定などの空間演算しか行えなかった。本発明により、クライアントで構造化文書が利用可能となることから、「道路周辺の、騒音が到達する２０ｍ範囲での、防音設備のない病院や幼稚園を検索せよ」などの属性と空間データを交えた検索や、シミュレーション等さまざまなアプリケーションが開発可能となり、地理情報分野の応用範囲が大きく広がる。

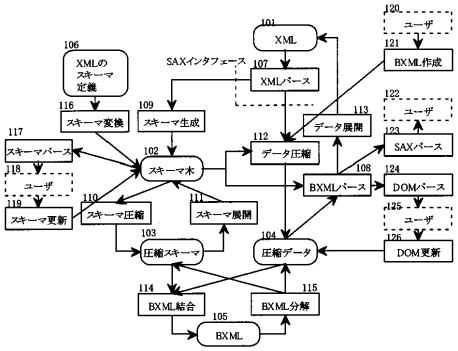
【図面の簡単な説明】

【図１】本発明を構成するシステムブロック図である。
【図２】ＳＡＸパーサのアルゴリズムである。
【図３】ＤＯＭパースブロックの基本インタフェースである。
【図４】ＷＡＰ　Ｂｉｎａｒｙ　ＸＭＬの圧縮方式である。
【図５】特開２００２－１６３２４８の圧縮方式である。
【図６】地物集合を表すＸＭＬ文書の例である。
【図７】ＢＸＭＬの圧縮方式である。
【図８】ＸＭＬ構造のパターンと圧縮方式である。
【図９】圧縮データのサンプルである。
【図１０】「文字列の辞書化」の概念的な構造である。
【図１１】辞書化のパターンである。
【図１２】可変長整数の構造である。
【図１３】本発明の機能ブロック図である。
【図１４】スキーマ木の構造である。
【図１５】スキーマ木のサンプルである。
【図１６】圧縮処理１３０１を実現するブロック抜粋図である。
【図１７】展開処理１３０２を実現するブロック抜粋図である。
【図１８】解析処理１３０３を実現するブロック抜粋図である。
【図１９】更新処理１３０４を実現するブロック抜粋図である。
【図２０】生成処理１３０５を実現するブロック抜粋図である。

【図２１】スキーマ生成ブロックのアルゴリズム（タグとデータの分離）である。
【図２２】スキーマ生成ブロックのアルゴリズム（要素型、文字列辞書、相対値）である。
【図２３】スキーマ型の上書規則である。
【図２４】ＸＭＬ　Ｓｃｈｅｍａの構造定義例と、その変換規則である。
【図２５】スキーマ木移動を説明する図である。
【図２６】スキーマ木移動のアルゴリズムである。
【図２７】データ圧縮ブロックのアルゴリズム（単独型）である。
【図２８】データ圧縮ブロックのアルゴリズム（重複型）である。
【図２９】データ圧縮ブロックのアルゴリズム（省略型）である。
【図３０】圧縮スキーマの構造である。
【図３１】ＢＸＭＬパースブロック１０８のアルゴリズムである。
【図３２】ＤＯＭパースブロックにおけるノード情報である。
【図３３】ＤＯＭパースブロック１２４における「属性値取得」アルゴリズムである。
【図３４】ＤＯＭパースブロック１２４における「テキスト要素取得」アルゴリズムである。
【図３５】ＤＯＭパースブロック１２４における「子に移動」アルゴリズムである。
【図３６】ＤＯＭパースブロック１２４における「弟に移動」アルゴリズムである。
【図３７】スキーマ要素識別子を用いた高速タグ判定方式である。
【図３８】地理情報分野に特化したノード例である。
【図３９】更新された圧縮データのデータ構造である。
【図４０】更新されたＢＸＭＬ文書に対するＢＸＭＬパースブロック１０８のアルゴリズムである。
【図４１】ＢＸＭＬ作成ブロックが提供するインタフェースである。
【図４２】ＢＸＭＬ部分ノードのコピー方式である。
【図４３】スキーマの隠蔽機能を説明する図である。
【図４４】隠蔽されたＢＸＭＬ文書に対するＢＸＭＬパースブロック１０８のアルゴリズムである。
【図４５】三階層モデルとＸＭＬ圧縮を説明する図である。
【図４６】ＲＤＢをベースとしたデータベース層４５１５の実装例である。
【図４７】図面をベースとしたデータベース層４５１５の実装例である。

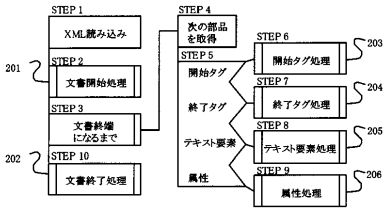
【図１】

図１



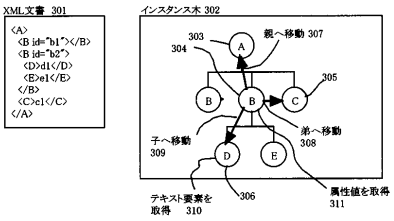
【図２】

図２



【図３】

図３



【図４】

図４

XML 文書	圧縮 XML 文書
<RoadGroup> <Road> <Name>国道 6 号線</Name> <Type>国道</Type> </Road> <Road> <Name>県道 20 号線</Name> <Type>県道</Type> </Road> </RoadGroup>	(タグ辞書) <RoadGroup> ::= 1 <Road> ::= 2 <Name> ::= 3 <Type> ::= 4 (圧縮 XML) <1> <2> <3>国道 6 号線</> <4>国道</> </> <2> <3>県道 20 号線</> <4>県道</> </> </>

【図５】

図５

XML 文書	圧縮 XML 文書
<RoadGroup> <Road> <Name>国道 6 号線</Name> <Type>国道</Type> </Road> <Road> <Name>県道 20 号線</Name> <Type>県道</Type> </Road> </RoadGroup>	(圧縮スキーマ) <RoadGroup> <Road> <Name></Name> <Type></Type> </Road> <Road> <Name></Name> <Type></Type> </Road> </RoadGroup> (圧縮データ) ,,"国道 6 号線",,"国道",,,,"県道 20 号線",,"県道",,

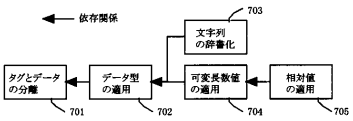
【図６】

図６

```
<Map>
  <featureMember>
    <Road fid="r001">
      <Name>Route 6</Name>
      <Lane>2</Lane>
      <Pavement>Asphalt</Pavement>
      <centerLineOf>
        <LineString SRS="WGS84">
          <coord><X>100</X><Y>100</Y></coord>
          <coord><X>220</X><Y>150</Y></coord>
          <coord><X>300</X><Y>130</Y></coord>
          ...
        </LineString>
      </centerLineOf>
    </Road>
  </featureMember>
  <featureMember><Road>...</Road></featureMemeber>
  <featureMember><House>...</House></featureMemeber>
  <featureMember><House>...</House></featureMemeber>
  ...
</Map>
```

【図７】

図７



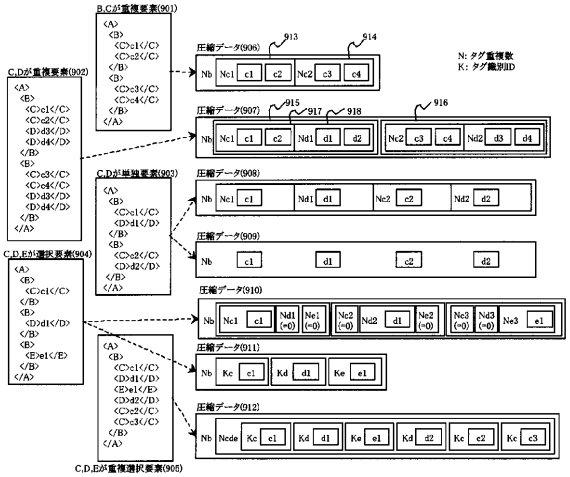
【図８】

図８

#	パターン 801	XML サンプル 802	BXML 文書 803					圧縮データ 810
			圧縮スキーマ 804					
			タグ名前 805	重複フラグ 806	スキーマ型 807	子要素保有数 808	属性保有数 809	
1	子要素 811	<A> <B> <C>text1</C> </B> </A>	A B C	単独 単独 単独	SEQUENCE SEQUENCE 文字列	1 1 -	0 0 0	text1
2	兄弟要素 812	<A> <B>text1</B> <C>text2</C> </A>	A B C	単独 単独 単独	SEQUENCE 文字列 文字列	2 - -	0 0 0	text1, text2
3	重複要素 813	<A> <B>text1</B> <B>text2</B> <C>text3</C> </A>	A B C	単独 重複 単独	SEQUENCE 文字列 文字列	2 - -	0 0 0	B の重複数=2 text1 text2 text3
4	省略要素 814	<R> <A> <B>text1</B> </A> <A> <B>text2</B> <C>text3</C> </A> </R>	R A B C	単独 重複 単独 重複	SEQUENCE SEQUENCE 文字列 文字列	1 2 - -	0 0 0 0	A の重複数=2 text1 C の重複数=0 text2 C の重複数=1 text3
5	選択要素 815	<R> <A> <B>text1</B> </A> <A> <C>text2</C> </A> </R>	R A B C	単独 重複 単独 単独	SEQUENCE CHOICE 文字列 文字列	1 2 - -	0 0 0 0	A の重複数=2 B が出現 text1 C が出現 text2
6	重複選択要素 816	<A> <B>text1</B> <C>text2</C> <B>text3</B> <C>text4</C> </A>	A B C	単独 単独 単独	MCHOICE 文字列 文字列	2 - -	0 0 0	B または C の重複数=4 B が出現 text1 C が出現 text2 B が出現 text3 C が出現 text4
7	属性 817	<A a='text1'> <B>text2</B> </A>	A a B	単独 属性 単独	SEQUENCE 文字列 文字列	1 - -	1 - 0	a の重複数=1 text1 text2

【図９】

図 9



【図１０】

図 10

XML 文書 1001	BXML 文書 1002	
	圧縮スキーマ 1003	圧縮データ 1004
<RoadGroup> <Road><Type>国道</Type></Road> <Road><Type>県道</Type></Road> <Road><Type>県道</Type></Road> <Road><Type>県道</Type></Road> </RoadGroup>	<Type>に含まれる文字列 ::= 1:国道, 2:県道	<RoadGroup> <Road><Type>1</Type></Road> <Road><Type>2</Type></Road> <Road><Type>2</Type></Road> <Road><Type>2</Type></Road> </RoadGroup>

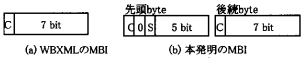
【図１１】

図 11

#	辞書化パターン	入力例	辞書	圧縮例
1	インライン文字列	"国道"	なし	T_WORD, "国道"
2	辞書 ID	"国道"	1="国道"	T_ID, 1
3	辞書 ID ＋ インライン文字列	"国道 20 号線"	1="国道"	C_ID, 1, T_WORD, "20 号線"
4	辞書 ID ＋ 数値	"http://a.com/f2256 "	1=" http://a.com/f "	C_ID, 1, T_NUM, 2256
5	インライン文字列 ＋ 辞書 ID	"user@crl.hitachi.co.jp"	1="@crl.hitachi.co.jp"	C_WORD, "user", T_ID, 1

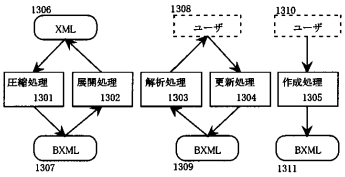
【図１２】

図 12



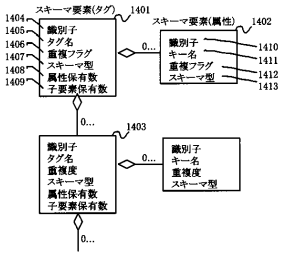
【図１３】

図 13



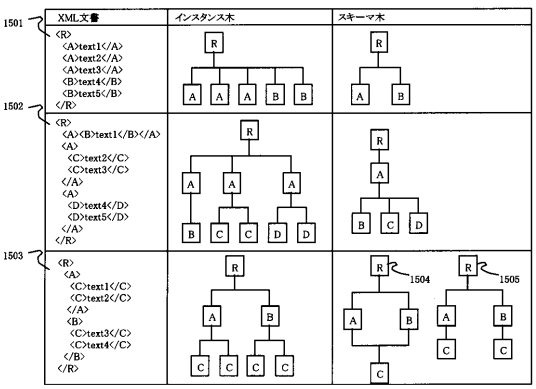
【図１４】

図 14



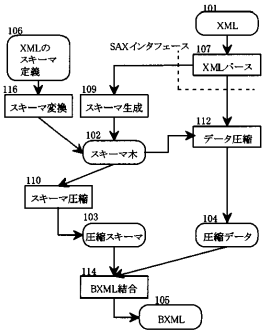
【図１５】

図 15



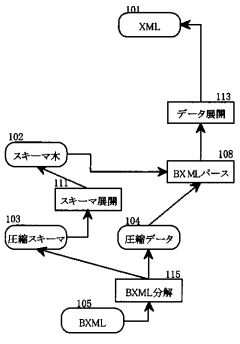
【図１６】

図 16



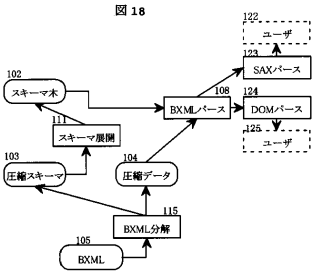
【図１７】

図 17



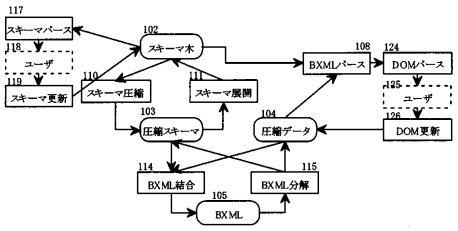
【図１８】

図 18



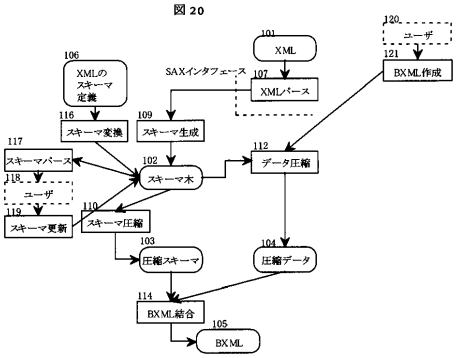
【図１９】

図 19



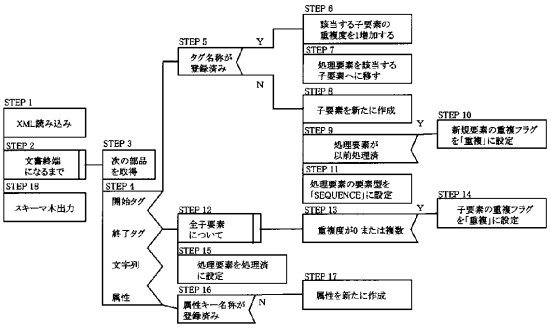
【図２０】

図 20



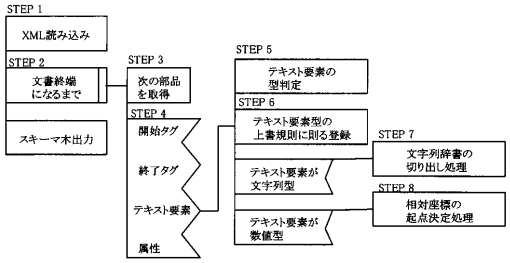
【図２１】

図 21



【図２２】

図 22



【図２３】

図 23

上書＼元	未定義	SEQUENCE CHOICE MCHOICE	文字列	実数	整数	2次元実数	2次元整数
未定義	OK	NG	NG	NG	NG	NG	NG
SEQUENCE CHOICE MCHOICE	OK	OK	OK	OK	OK	OK	OK
文字列	OK	NG	OK	OK	OK	OK	OK
実数	OK	NG	NG	OK	OK	文字列	文字列
整数	OK	NG	NG	NG	OK	文字列	文字列
2次元実数	OK	NG	NG	文字列	文字列	OK	NG
2次元整数	OK	NG	NG	文字列	文字列	OK	OK

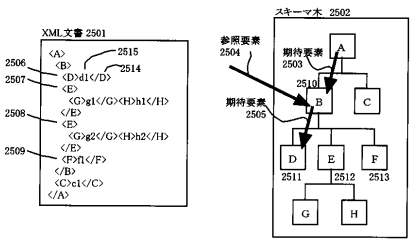
【図２４】

図 24

#	XML Schema		圧縮スキーマ	
	BNF 記法 2401	サンプル 2402	BNF 記法 2403	要素型 2404
1	BCDE	BCDE 固定	BCDE	SEQUENCE
2	B|C|D|E	B or C or D or E	B|C|D|E	CHOICE
3	*(BCDE)	BCDEBCDE	*(B|C|D|E)	MCHOICE
4	*(B|C|D|E)	BCDE 任意組合せ		
5	*(B|C|D|E) 但し各出現回数=1	EDCB		
6	(BC)(DE)	BCDE	BCDE	SEQUENCE
7	(BC)|(DE)	BC or DE	*(B|C|D|E)	MCHOICE
8	(B|C)(D|E)	BD or BE or CD or CE		
9	(B|C)|(D|E)	B or C or D or E		
10	*(BC)*(DE)	BCBCDEDE		
11	(*(BC))|(*(DE))	BCBC or DEDE		
12	*(B|C)*(D|E)	BBBDDD or BBBEEE or CCCDDD or CCCEEE		
13	(*(B|C))|(*(D|E))	BC 任意組合せ or DE 任意組合せ		
14	(BC)(D|E)	BCD BCE		
15	(BC)|(D|E)	BC D E		
16	(*(BC))|(D|E)	BCBC D E		
17	(BC)|(*(D|E))	BC DEDEDE		
18	(*(BC))|(*(D|E))	BCBC DEDE		
19	*(*(BC)|*(DE))	BCDEBCDE		
20	*(*(BC)*(DE))	BCBCDEDEBCBCDEDE		
21	*(BC)(D|E)	BCBCD BCBCE		
22	BC*(D|E)	BCCCD BCCCE		
23	*(BC)*(D|E)	BCBCDDD BCBCEEE		
	...			

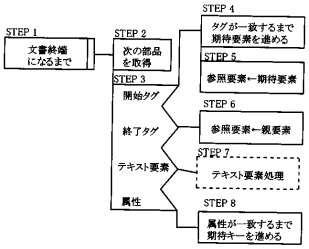
【図２５】

図 25



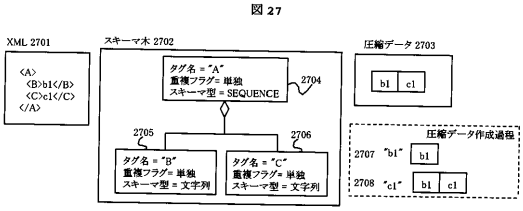
【図２６】

図 26



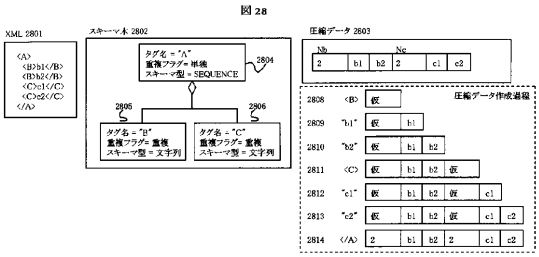
【図２７】

図 27



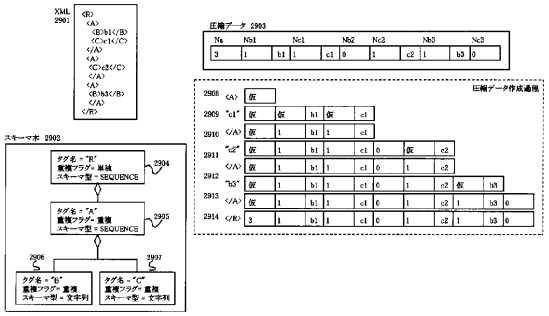
【図２８】

図 28



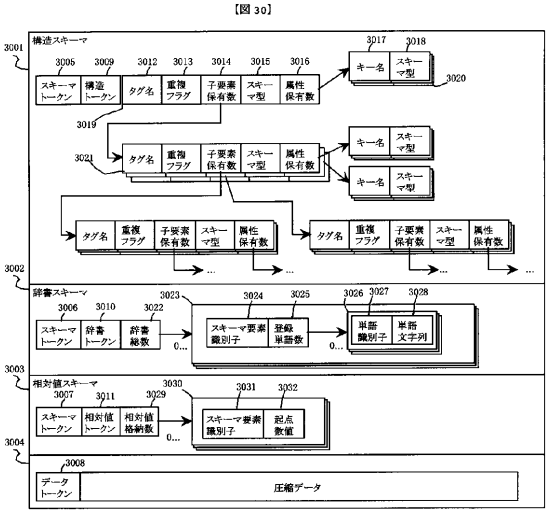
【図２９】

図 29



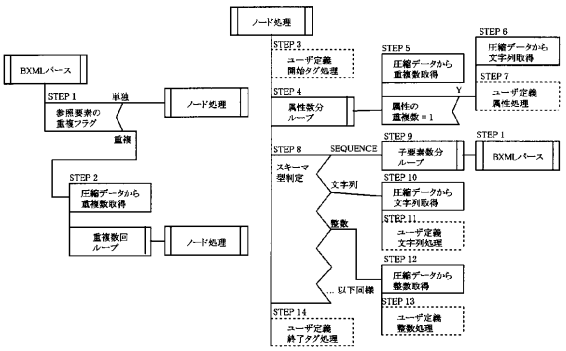
【図３０】

【図 30】



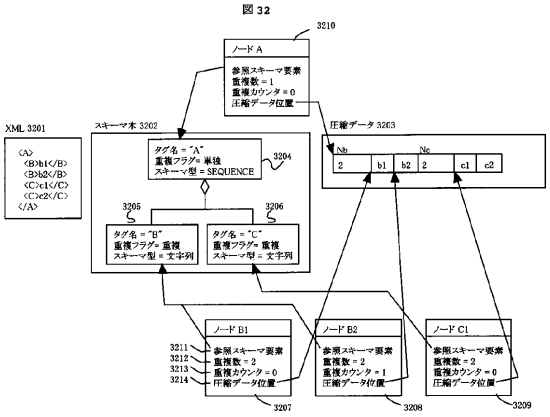
【図３１】

図 31



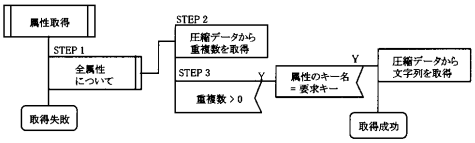
【図３２】

図 32



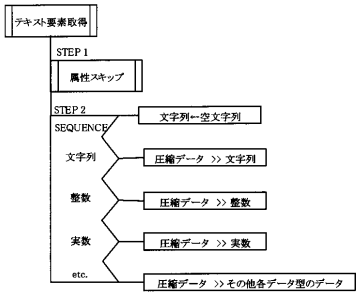
【図３３】

図 33



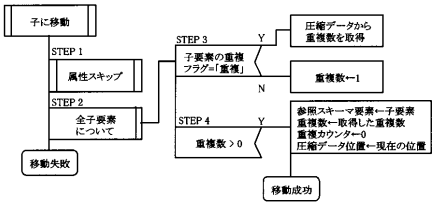
【図３４】

図 34



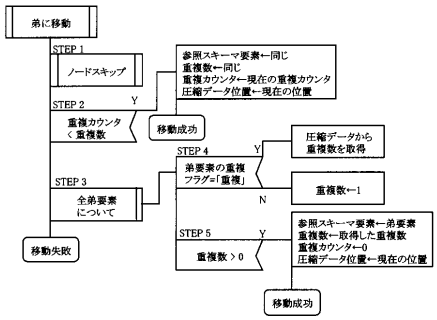
【図３５】

図 35



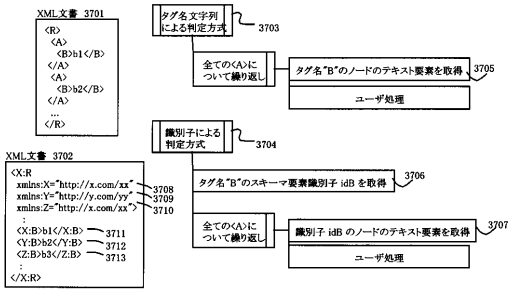
【図３６】

図 36



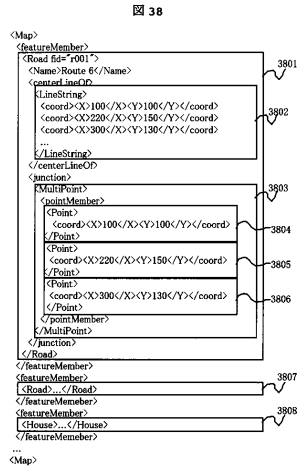
【図３７】

図 37



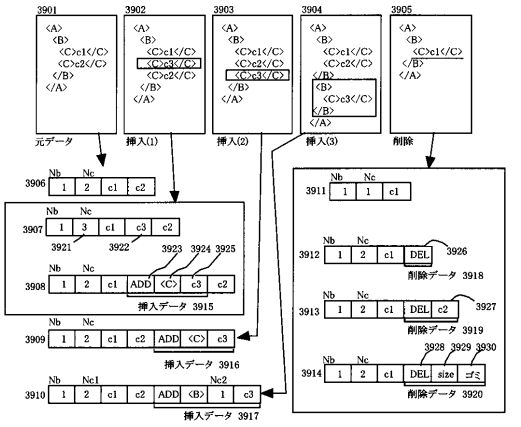
【図３８】

図 38



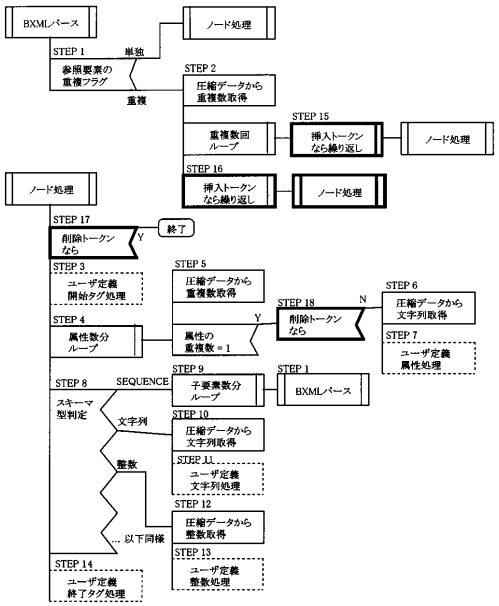
【図３９】

図 39



【図４０】

図 40



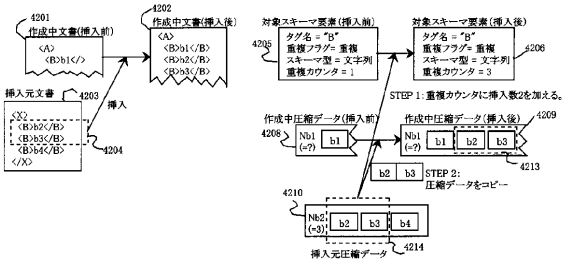
【図４１】

図 41

#	分野	メソッド	書式	開始状態		終了状態
1	文書作成	文書開始(4108)	startDocument();	―(4101)		<A>
2		文書終了(4109)	endDocument();	開(4102)	<A><B><C>	<A><B><C></C></B></A>
				開(4103)	<A><B><C></C>	
3	ノード作成	子要素作成(4110)	moveToChild("C");	開(4104)	<A><B>	<A><B><C>
				開(4105)	<A><B></B>	<A><B></B><B><C>
4		弟要素作成(4111)	moveToSibling("C");	開(4106)	<A><B>	<A><B></B><C>
				開(4107)	<A><B></B>	
5		親へ移動(4112)	moveToParent();	開(4108)	<A><B>	<A><B></B></A>
				開(4109)	<A><B></B>	
6		先祖へ移動(4113)	moveToAncestor("B");	開(4110)	<A><B><C><D>	<A><B><C><D></D></C></B>
				開(4111)	<A><B><C><D></D>	
7	データ作成	属性の作成(4114)	setAttr("id", "1");	開(4112)	<A>	<A id="1">
8		文字列の作成(4115)	setText("text");	開(4113)	<A>	<A>text</A>
9		整数の作成(4116)	setInteger(8);	開(4114)	<A>	<A>8</A>
10		実数の作成(4117)	setDouble(3.14);	開(4115)	<A>	<A>3.14</A>
11		空タグの作成(4118)	setNull();	開(4116)	<A>	<A></A>
12		ノードのコピー(4119)	setNode(node);	開(4117)	<A>	<A><B>...</B></A>

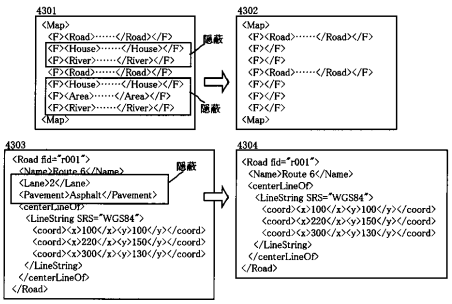
【図４２】

図 42



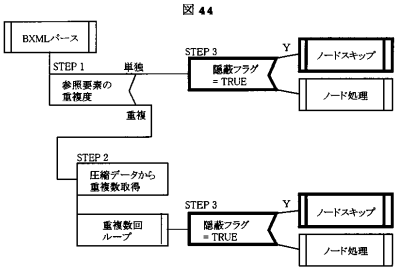
【図４３】

図 43



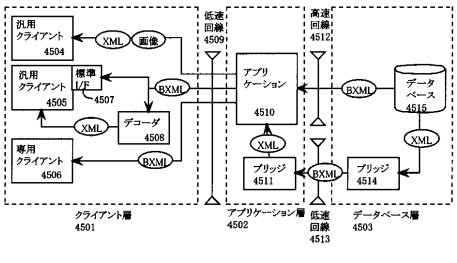
【図４４】

図 44



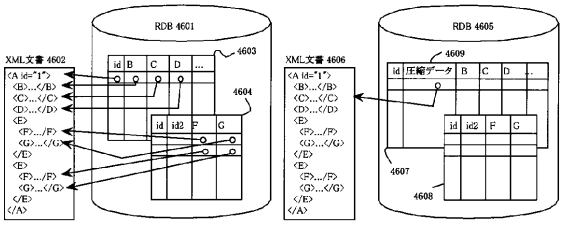
【図４５】

図 45



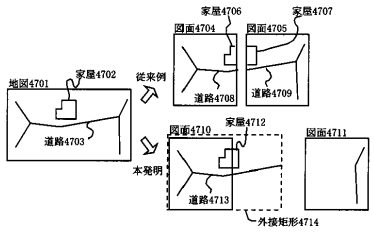
【図４６】

図 46



【図４７】

図47


家屋4706
家屋4707
図面4704
図面4705
地図4701
家屋4702
従来例
道路4708
道路4709
道路4703
図面4710
家屋4712
図面4711
本発明
道路4713
外接矩形4714