(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2009-59215
(P2009-59215A)
(43)公開日　平成21年3月19日(2009.3.19)

(51)Int.Cl.		F I			テーマコード（参考）
G06F 12/00	(2006.01)	G06F	12/00	511C	5B009
G06F 17/21	(2006.01)	G06F	17/21	501T	5B075
G06F 17/30	(2006.01)	G06F	12/00	547Z	5B082
		G06F	17/30	140	5B109

審査請求 未請求　請求項の数 6　ＯＬ　（全 21 頁）

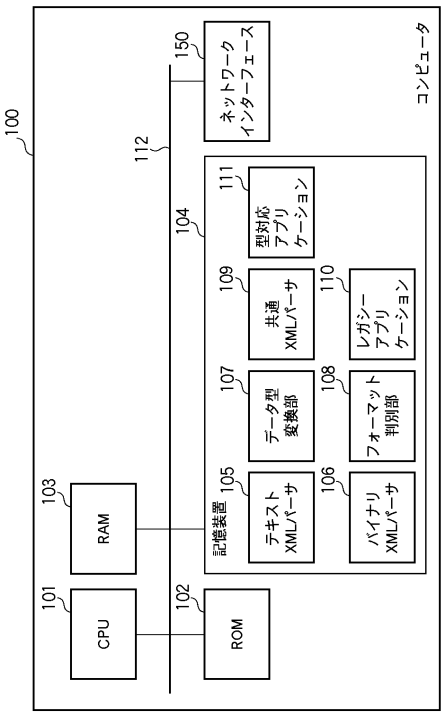
(21)出願番号　特願2007-226694 (P2007-226694)
(22)出願日　平成19年8月31日 (2007.8.31)

(71)出願人　000001007
キヤノン株式会社
東京都大田区下丸子３丁目３０番２号
(74)代理人　100076428
弁理士　大塚　康徳
(74)代理人　100112508
弁理士　高柳　司郎
(74)代理人　100115071
弁理士　大塚　康弘
(74)代理人　100116894
弁理士　木村　秀二
(74)代理人　100130409
弁理士　下山　治
(74)代理人　100134175
弁理士　永川　行光

最終頁に続く

(54)【発明の名称】構造化文書処理装置、構造化文書処理方法

(57)【要約】
【課題】　一つのアプリケーションで複数種のフォーマットのＸＭＬ文書を扱うことを可能にするための技術を提供すること。更に、データ型に応じたエンコーディングで記述されたバイナリＸＭＬの文書を効率的に扱う為の技術を提供すること。
【解決手段】　ＸＭＬ文書のフォーマットに応じて、テキストＸＭＬパーサ１０５、バイナリＸＭＬパーサ１０６の何れかにより、ＸＭＬ文書を解析する。型対応アプリケーション１１１は、ＸＭＬ文書中に記述されている要素を、指定された型で取得する要求を受け付ける。型対応アプリケーション１１１は、解析した型と、指定された型とが一致している場合には、要素を要求元に出力し、一致していない場合には、要素の型を指定された型に変換してから要求元に出力する。
【選択図】　図１



【特許請求の範囲】
【請求項１】
構造化文書を処理する構造化文書処理装置であって、
構造化文書のフォーマットを取得する取得手段と、
前記取得手段が取得したフォーマットに応じた解析方法で、前記構造化文書を解析する解析手段と、
前記構造化文書中に記されている要素を、指定された型で取得する要求を受け付ける手段と、
前記要素について前記解析手段が解析した型と、前記指定された型とが一致しているか否かを判断する判断手段と、
前記判断手段が一致していると判断した場合には前記要素を要求元に出力し、前記判断手段が一致していないと判断した場合には前記要素の型を前記指定された型に変換してから当該要素を前記要求元に出力する出力手段と
を備えることを特徴とする構造化文書処理装置。
【請求項２】
前記解析手段は、バイナリＸＭＬパーサと、テキストＸＭＬパーサと、で構成されており、
前記取得手段が取得したフォーマットがバイナリＸＭＬである場合には、前記バイナリＸＭＬパーサによって前記構造化文書を解析し、
前記取得手段が取得したフォーマットがテキストＸＭＬである場合には、前記テキストＸＭＬパーサによって前記構造化文書を解析する
ことを特徴とする請求項１に記載の構造化文書処理装置。
【請求項３】
前記解析手段は、Ｆａｓｔ　Ｉｎｆｏｓｅｔパーサと、テキストＸＭＬパーサと、で構成されており、
前記取得手段が取得したフォーマットがＦａｓｔ　Ｉｎｆｏｓｅｔ形式である場合には、前記Ｆａｓｔ　Ｉｎｆｏｓｅｔパーサによって前記構造化文書を解析し、
前記取得手段が取得したフォーマットがテキストＸＭＬである場合には、前記テキストＸＭＬパーサによって前記構造化文書を解析する
ことを特徴とする請求項１に記載の構造化文書処理装置。
【請求項４】
構造化文書を処理する構造化文書処理装置が行う構造化文書処理方法であって、
構造化文書のフォーマットを取得する取得工程と、
前記取得工程で取得したフォーマットに応じた解析方法で、前記構造化文書を解析する解析工程と、
前記構造化文書中に記されている要素を、指定された型で取得する要求を受け付ける工程と、
前記要素について前記解析工程で解析した型と、前記指定された型とが一致しているか否かを判断する判断工程と、
前記判断工程で一致していると判断した場合には前記要素を要求元に出力し、前記判断工程で一致していないと判断した場合には前記要素の型を前記指定された型に変換してから当該要素を前記要求元に出力する出力工程と
を備えることを特徴とする構造化文書処理方法。
【請求項５】
コンピュータに請求項４に記載の構造化文書処理方法を実行させるためのプログラム。
【請求項６】
請求項５に記載のプログラムを格納した、コンピュータ読み取り可能な記憶媒体。
【発明の詳細な説明】
【技術分野】
【０００１】

本発明は、構造化文書を処理する技術に関するものである。

【背景技術】

【０００２】

現在、コンピュータ上で扱う様々なデータのフォーマットとしてＸＭＬ（Ｅｘｔｅｎｓｉｂｌｅ　Ｍａｒｋｕｐ　Ｌａｎｇｕａｇｅ：http:／／www.w3.org／TR／2004／REC-xml-20040204／）が使用されている。ＸＭＬは、コンピュータやオペレーティングシステムなどに依存しないという特徴を持っている。これにより、ネットワーク上の異なる種類のコンピュータや機器間での通信が容易になるため、特に、ネットワーク上での通信データとして広く普及している。

【０００３】

また、最近は携帯電話や複写機、デジタルカメラなどといった、パーソナルコンピュータやサーバ以外のさまざまな機器のネットワーク化が進んでいる。このため、これらの機器でもＸＭＬを扱うことが増えている。

【０００４】

このような中で、ＸＭＬの処理速度や効率性が大きな問題となっている。ＸＭＬのフォーマットは、処理速度の向上を優先させた書式にはなっていないため、解析処理に時間がかかる。また、記述に冗長性があるために、そのデータサイズが大きくなる。これらの問題は、処理速度が遅くメモリ資源の少ない小型機器では大きな問題となる。また、サーバなどリソースの多い機器であっても、非常に多くのＸＭＬ形式の文書を処理する場合には、ＸＭＬの解析時間が大きな問題となっている。

【０００５】

そのため、ＸＭＬフォーマットと意味的に等価で、かつ、より効率的な処理が可能なフォーマットが使われるようになってきた。このようなフォーマットは一般に「バイナリＸＭＬ」と呼ばれている。これに対し、ＸＭＬの仕様に従った、テキスト形式のＸＭＬのことを本明細書では「テキストＸＭＬ」と呼ぶ。

【０００６】

バイナリＸＭＬのフォーマット仕様は一つではなく、いくつかある。多くのフォーマットは、冗長性の排除とデータ型に応じたエンコーディングを行うことによってサイズの低減や処理の効率化を図っている。

【０００７】

冗長性の排除とは、終了タグ名を省略したり、頻繁に登場する要素名や属性名、属性値などの文字列を整数に置き換えることである。終了タグは必ず直前に記述された開始タグと同じ名前でなければならないため、終了タグの名前は省くことが可能である。また、例えば画像を多く含むＸＨＴＭＬ文書では、「ｉｍｇ」という文字列が頻繁に出現する。これら頻出文字列をできるだけ小さな整数に置き換えることで、文書サイズの削減が行われている。

【０００８】

データ型に応じたエンコーディングとは、要素の内容や属性値等に対するエンコーディング方法を、その型（整数、浮動小数、日付など）に応じて変えることである。例えば、テキストＸＭＬでは、＜ｘ＞１２３４５＜／ｘ＞という要素における”１２３４５”が整数の１２３４５を表していたとしても、文書中には”１２３４５”という文字列として記述される。よって、文書の文字エンコーディングがＵＴＦ－８の場合、０ｘ３０、０ｘ３１、０ｘ３２、０ｘ３３、０ｘ４５というデータになる。

【０００９】

このように、ＸＭＬ文書中に記述される書式とコンピュータ内部で扱うための書式が異なるため、ＸＭＬ文書を読み込んでコンピュータ内部で処理する際に、書式の変換を行わなければならない。例えば、ある構造化文書処理装置の内部で、整数がビッグエンディアンの４バイトで扱われている場合、整数１２３４５は０ｘ００、０ｘ００、０ｘ３０、０ｘ２Ｅというバイト列になる。このような型の変換は、特に浮動小数の場合に多くの時間を要する。

【００１０】
これに対しバイナリＸＭＬでは、整数や浮動小数の値を、コンピュータ内部で扱う書式と同じ形式で記述する。このため、書式の変換を行う必要が無く、より高速に処理することが可能である。
【００１１】
インデックス化による冗長性の除去とデータ型に応じたエンコーディングを行う例としては、Ｆａｓｔ　Ｉｎｆｏｓｅｔ（ITU-T Rec. X.891 | ISO／IEC24824-1）が挙げられる。
【００１２】
バイナリＸＭＬを扱う場合は通常、バイナリＸＭＬデータ専用の解析器（以下バイナリＸＭＬパーサ）を使うのが普通である。また、バイナリＸＭＬパーサのインターフェースはテキストＸＭＬパーサと同じものになっていることが多い。なぜなら、インターフェースを揃えておくことで、テキストＸＭＬパーサを使用するアプリケーションを改変せずにバイナリＸＭＬパーサに対応させることができるからである。テキストＸＭＬパーサと同じインターフェースを持つバイナリＸＭＬパーサとして、Ｓｕｎ　Ｍｉｃｒｏｓｙｓｔｅｍｓ　Ｉｎｃ．のＦａｓｔ　Ｉｎｆｏｓｅｔ　Ｐｒｏｊｅｃｔのパーサがある。
【００１３】
特許文献１には、XMLデータを利用したシステムと、レガシーファイルデータを利用したシステムの、どちらでもデータを処理することができるように、それぞれをデータ変換することが記載されている。
【非特許文献１】Ｆａｓｔ　Ｉｎｆｏｓｅｔ　Ｐｒｏｊｅｃｔ（https:／／fi.dev.java.net／）
【特許文献１】特開2004-318420号公報
【発明の開示】
【発明が解決しようとする課題】
【００１４】
しかしながら、テキストＸＭＬパーサと同じインターフェースを持つバイナリＸＭＬパーサでは、データ型に応じたエンコーディングを行うバイナリＸＭＬフォーマットのメリットを生かすことができないという欠点があった。
【００１５】
なぜなら、テキストＸＭＬパーサのインターフェースでは、データはすべて文字列型として受け渡しが行われるため、インターフェースを同じにすると文字列型のデータしか扱えなくなってしまうからである。このため、バイナリＸＭＬ文書中にＩＥＥＥ７５４形式の浮動小数データがある場合も、バイナリＸＭＬパーサが文字列型に変換して渡し、さらにアプリケーションがまたＩＥＥＥ７５４形式に戻す、という無駄な変換が行われてしまう。
【００１６】
また、バイナリＸＭＬパーサのインターフェースをテキストＸＭＬパーサと異なるものにしてしまうと、バイナリＸＭＬパーサ用アプリケーションはテキストＸＭＬパーサを扱えないことになる。つまり、テキストＸＭＬ文書に対応できなくなるという問題が起きてしまう。
【００１７】
本発明は以上の問題に鑑みてなされたものであり、一つのアプリケーションで複数種のフォーマットのＸＭＬ文書を扱うことを可能にするための技術を提供することを目的とする。
【００１８】
更に、データ型に応じたエンコーディングで記述されたバイナリＸＭＬの文書を効率的に扱う為の技術を提供することも目的とする。
【課題を解決するための手段】
【００１９】

本発明の目的を達成するために、例えば、本発明の構造化文書処理装置は以下の構成を備える。

【００２０】

即ち、構造化文書を処理する構造化文書処理装置であって、

構造化文書のフォーマットを取得する取得手段と、

前記取得手段が取得したフォーマットに応じた解析方法で、前記構造化文書を解析する解析手段と、

前記構造化文書中に記されている要素を、指定された型で取得する要求を受け付ける手段と、

前記要素について前記解析手段が解析した型と、前記指定された型とが一致しているか否かを判断する判断手段と、

前記判断手段が一致していると判断した場合には前記要素を要求元に出力し、前記判断手段が一致していないと判断した場合には前記要素の型を前記指定された型に変換してから当該要素を前記要求元に出力する出力手段と

を備えることを特徴とする。

【００２１】

本発明の目的を達成するために、例えば、本発明の構造化文書処理方法は以下の構成を備える。

【００２２】

即ち、構造化文書を処理する構造化文書処理装置が行う構造化文書処理方法であって、

構造化文書のフォーマットを取得する取得工程と、

前記取得工程で取得したフォーマットに応じた解析方法で、前記構造化文書を解析する解析工程と、

前記構造化文書中に記されている要素を、指定された型で取得する要求を受け付ける工程と、

前記要素について前記解析工程で解析した型と、前記指定された型とが一致しているか否かを判断する判断工程と、

前記判断工程で一致していると判断した場合には前記要素を要求元に出力し、前記判断工程で一致していないと判断した場合には前記要素の型を前記指定された型に変換してから当該要素を前記要求元に出力する出力工程と

を備えることを特徴とする。

【発明の効果】

【００２３】

本発明の構成によれば、一つのアプリケーションで複数種のフォーマットのＸＭＬ文書を扱うことを可能にする。

【００２４】

更に、データ型に応じたエンコーディングで記述されたバイナリＸＭＬの文書を効率的に扱うことができる。

【発明を実施するための最良の形態】

【００２５】

以下、添付図面を参照し、本発明の好適な実施形態について詳細に説明する。

【００２６】

［第１の実施形態］

図１は、本実施形態に係る構造化文書処理装置に適用可能なコンピュータのハードウェア構成例を示すブロック図である。なお、本実施形態に係る構造化文書処理装置に適用可能な装置が有する構成は、図１に示した構成に限定するものではなく、当業者であれば、種種の変形例が考え得る。更に、本実施形態に係る構造化文書処理装置を１台の装置で実現させることに限定するものではなく、複数台の装置による協調動作でもって、本実施形態に係る構造化文書処理装置を実現させても良い。この場合、複数台の装置間は、ＬＡＮなどのネットワークを介して接続されていることになる。

【００２７】
図１において、ＣＰＵ１０１は、ＲＯＭ１０２やＲＡＭ１０３に格納されているプログラムやデータを用いて、コンピュータ１００全体の制御を行うと共に、コンピュータ１００が行うものとして説明する後述の各処理を実行する。
【００２８】
ＲＯＭ１０２には、コンピュータ１００の設定データやブートプログラム、変更を必要としないパラメータのデータなどが格納されている。
【００２９】
ＲＡＭ１０３は、記憶装置１０４からロードされたプログラムやデータ、ネットワークインターフェース１５０を介して外部から受信したデータなどを一時的に記憶するためのエリアを有する。更には、ＲＡＭ１０３は、ＣＰＵ１０１が各種の処理を実行する際に用いるワークエリアも有する。
【００３０】
記憶装置１０４は、ハードディスクドライブ装置に代表される大容量情報記憶装置である。記憶装置１０４には、ＯＳ（オペレーティングシステム）や、コンピュータ１００が行うものとして説明する後述の各処理をＣＰＵ１０１に実行させるためのプログラムやデータが保存されている。また、記憶装置１０４には、後述する処理の対象となる構造化文書としてのＸＭＬ文書のデータが、ファイルとして保存されている。記憶装置１０４に保存されているプログラムやデータは、ＣＰＵ１０１による制御に従って適宜ＲＡＭ１０３にロードされ、ＣＰＵ１０１による処理対象となる。
【００３１】
以下に、記憶装置１０４に保存されている各ソフトウェアについて説明する。
【００３２】
テキストＸＭＬパーサ１０５は、テキスト形式のＸＭＬ文書（以下、テキストＸＭＬ文書と呼称する）の解析依頼を受けると、このＸＭＬ文書に対する解析処理を行い、その結果を返す。
【００３３】
バイナリＸＭＬパーサ１０６は、バイナリ形式のＸＭＬ文書（以下、バイナリＸＭＬ文書と呼称する）の解析依頼を受けると、このＸＭＬ文書に対する解析処理を行い、その結果を返す。
【００３４】
データ型変換部１０７は、変換前のデータの型、変換後のデータの型、変換対象データ、の３つを指定すると、変換対象データの型を変換後の型に変換して返す。
【００３５】
フォーマット判別部１０８は、与えられたデータのフォーマットを判別する。
【００３６】
共通ＸＭＬパーサ１０９は、テキストＸＭＬパーサ１０５やバイナリＸＭＬパーサ１０６を使い分けて、テキストＸＭＬ文書やバイナリＸＭＬ文書の解析処理を実現するものである。
【００３７】
レガシーアプリケーション１１０は、テキストＸＭＬパーサ１０５のＡＰＩ（Ａｐｐｌｉｃａｔｉｏｎ　Ｐｒｏｇｒａｍｍｉｎｇ　Ｉｎｔｅｒｆａｃｅ）を使用して処理を行う。
【００３８】
型対応アプリケーション１１１は、共通ＸＭＬパーサ１０９のＡＰＩを使用して処理を行う。また、レガシーアプリケーション１１０と型対応アプリケーション１１１とは、両方ともネットワーク上から受信したＸＭＬ文書を処理するサービスとして機能するものである。
【００３９】
なお、記憶装置１０４に保存されているものとして説明した各ソフトウェアによって実

現される処理については後述する。
【００４０】
ネットワークインターフェース１５０は、コンピュータ１００をＬＡＮやインターネット等に接続するためのものであり、コンピュータ１００はこのネットワークインターフェース１５０を介して、外部機器とのデータ通信を行うことができる。
【００４１】
１１２は、上述の各部を繋ぐバスである。
【００４２】
図２は、上記コンピュータ１００を適用したネットワークの構成例を示す図である。
【００４３】
図２に示す如く、コンピュータ１００をサーバとしてネットワーク２０１に接続する。ネットワーク２０１は、ＬＡＮやインターネット等により構成されている。２０２，２０３はそれぞれクライアント端末で、ネットワーク２０１に接続されている。
【００４４】
ここで、クライアント端末２０２はバイナリＸＭＬ文書を生成し、生成したバイナリＸＭＬ文書をコンピュータ１００に送信するものとする。一方、クライアント端末２０３はテキストＸＭＬ文書を生成し、生成したテキストＸＭＬ文書をコンピュータ１００に送信するものとする。
【００４５】
次に、テキストＸＭＬパーサ１０５のＡＰＩについて、図３を用いて説明する。図３は、テキストＸＭＬパーサ１０５のＡＰＩの一例を示す図である。
【００４６】
「ＳｅｔＤｏｃｕｍｅｎｔ」は、解析対象のＸＭＬ文書を開く為の関数である。
【００４７】
「Ｒｅａｄ」は、解析対象のＸＭＬ文書の先頭からノード１つ分読み進める為の関数である。ここで、ノードとはＸＭＬ文書を構成する単位であり、開始タグ（ＳｔａｒｔＥｌｅｍｅｎｔ）、終了タグ（ＥｎｄＥｌｅｍｅｎｔ）、要素の内容（Ｃｏｎｔｅｎｔ）などがある。
【００４８】
「ＧｅｔＮｏｄｅＴｙｐｅ」は、現在参照しているノードの型（ノード型）を返す為の関数であり、「ＳｔａｒｔＥｌｅｍｅｎｔ」や「ＥｎｄＥｌｅｍｅｎｔ」といった値を返す。
【００４９】
「ＧｅｔＮａｍｅ」は、現在参照しているノードの名前を返す為の関数である。つまり、現在参照しているノードが開始タグの場合には、開始タグのタグ名を返す。
【００５０】
「ＧｅｔＶａｌｕｅ」は、現在参照しているノードの値を返す為の関数である。つまり、現在参照しているノードが「Ｃｏｎｔｅｎｔ」の場合には、その要素の内容を返す。テキストＸＭＬ文書はすべてテキスト形式で記述されているため、「ＧｅｔＶａｌｕｅ」の戻り値もｓｔｒｉｎｇ型である。
【００５１】
「Ｃｌｏｓｅ」は、解析処理を終了し、確保していたメモリ資源などを解放する為の関数である。
【００５２】
次に、バイナリＸＭＬパーサ１０６のＡＰＩについて、図４を用いて説明する。図４は、バイナリＸＭＬパーサ１０６のＡＰＩの一例を示す図である。
【００５３】
「ＳｅｔＤｏｃｕｍｅｎｔ」、「Ｒｅａｄ」、「ＧｅｔＮｏｄｅＴｙｐｅ」、「ＧｅｔＮａｍｅ」、「Ｃｌｏｓｅ」の各関数については図３に示したものと同じであり、その説明も上述の通りである。即ち、これらの関数は、テキストＸＭＬパーサ１０５の同名のＡ

ＰＩと同じ役目を果たす。しかし、ノードの値を取得する為の関数については、テキストＸＭＬパーサ１０５とバイナリＸＭＬパーサ１０６とでは、大きく異なる。

【００５４】

「ＧｅｔＶａｌｕｅＴｙｐｅ」は、現在参照しているノードの値の型を返す為の関数である。例えば、現在参照しているノードの値が、バイナリＸＭＬ文書内に整数値として記述されている場合は「ｉｎｔ」を返し、浮動小数として記述されている場合は「ｄｏｕｂｌｅ」を返す。

【００５５】

「ＧｅｔＳｔｒｉｎｇＶａｌｕｅ」は、現在参照している文字列型のノードの値を取得する為の関数である。

【００５６】

「ＧｅｔＩｎｔＶａｌｕｅ」は、現在参照している整数型のノードの値を取得するための関数である。

【００５７】

「ＧｅｔＤｏｕｂｌｅＶａｌｕｅ」は、現在参照している浮動小数型のノードの値を取得する為の関数である。

【００５８】

つまり、バイナリＸＭＬパーサ１０６のＡＰＩは、現在参照しているノードの値を、ＸＭＬ文書内に記されている型で返す。

【００５９】

次に、共通ＸＭＬパーサ１０９のＡＰＩについて、図５を用いて説明する。図５は、共通ＸＭＬパーサ１０９のＡＰＩの一例を示す図である。

【００６０】

「ＳｅｔＤｏｃｕｍｅｎｔ」、「Ｒｅａｄ」、「ＧｅｔＮｏｄｅＴｙｐｅ」、「ＧｅｔＮａｍｅ」、「Ｃｌｏｓｅ」の各関数については図３に示したものと同じであり、その説明も上述の通りである。即ち、これらの関数は、テキストＸＭＬパーサ１０５、バイナリＸＭＬパーサ１０６の同名のＡＰＩと同じ役目を果たす。

【００６１】

「ＧｅｔＶａｌｕｅＡｓＳｔｒｉｎｇ」は、現在参照しているノードの値を、文字列として取得する関数である。

【００６２】

「ＧｅｔＶａｌｕｅＡｓＩｎｔ」は、現在参照しているノードの値を、整数として取得する関数である。

【００６３】

「ＧｅｔＶａｌｕｅＡｓＤｏｕｂｌｅ」は、現在参照しているノードの値を、浮動小数型として取得する関数である。

【００６４】

次に、図６に例示する構成を有するＸＭＬ文書を処理対象とする場合における、コンピュータ１００の動作について説明する。図６は、コンピュータ１００による処理対象としてのＸＭＬ文書の構成例を示す図である。図６に示す構成を有するＸＭＬ文書は、人の名前（ｎａｍｅ）と身長（ｈｅｉｇｈｔ）とを格納する個人情報データである。

【００６５】

「＜」と「＞」で囲まれたものは開始タグを表す。図６では、６０２，６０３，６０６が開始タグに相当する。

【００６６】

「＜／＞」は終了タグを表す。図６では、６０５，６０８，６０９が終了タグに相当する。

【００６７】

要素の内容部分６０４，６０７はＳやＦといった記号から始まり、その後に実際の値が記されている。内容部分６０４における先頭の「Ｓ」は、それに後続する値がＵＴＦ－８

で記されている文字列であることを示している。「Ｆ」は、それに後続する値がＩＥＥＥ７５４形式の４バイト浮動小数形式で記されていることを示している。

【００６８】

ＩＥＥＥ７５４形式は、アプリケーションが扱う浮動小数型と同じ形式である。ＸＭＬ文書における先頭部分、即ち６０１で示す部分は、マジックナンバーと呼ばれており、ＸＭＬ文書の先頭付近の数バイトを見ることで、このＸＭＬ文書のフォーマットが認識できるようになっている。本実施形態では、ＸＭＬ文書がバイナリＸＭＬ文書であることを示すためには、マジックナンバー６０１として、「０ｘ０１、０ｘ０２、０ｘ０３」を用いる。

【００６９】

次に、図６に示すＸＭＬ文書のデータを記憶装置１０４からＲＡＭ１０３にロードした後に、コンピュータ１００が行う処理について、図８を用いて説明する。図８は、ＣＰＵ１０１が、型対応アプリケーション１１１のプログラムを実行することでなされる処理のフローチャートである。係る処理では、図６に示すＸＭＬ文書、即ち、個人情報データから、名前と身長をそれぞれ文字列、整数として取得する。

【００７０】

先ず、ステップＳ８０２では、関数「ＳｅｔＤｏｃｕｍｅｎｔ」を実行し、図６に示すＸＭＬ文書を開く。なお、ステップＳ８０２における処理を行うと、図９に示したフローチャートに従った処理が開始される。図９のフローチャートについては後述する。

【００７１】

次にステップＳ８０３では、最初の開始タグが「ｐｅｒｓｏｎ」であることを確認する為に、関数「Ｒｅａｄ」を実行し、係る実行により進めた現在の参照位置に対して関数「ＧｅｔＮｏｄｅＴｙｐｅ」、関数「ＧｅｔＮａｍｅ」を実行する。そして、関数「ＧｅｔＮｏｄｅＴｙｐｅ」の返り値が開始タグ、且つ関数「ＧｅｔＮａｍｅ」による返り値が「ｐｅｒｓｏｎ」となるまで、関数「Ｒｅａｄ」を実行する。

【００７２】

次にステップＳ８０４では、関数「Ｒｅａｄ」を実行し、係る実行により進めた現在の参照位置に対して関数「ＧｅｔＮｏｄｅＴｙｐｅ」、関数「ＧｅｔＮａｍｅ」を実行する。そして、関数「ＧｅｔＮｏｄｅＴｙｐｅ」の返り値がｎａｍｅタグ、且つ関数「ＧｅｔＮａｍｅ」による返り値が「ｎａｍｅ」となるまで、関数「Ｒｅａｄ」を実行する。

【００７３】

次にステップＳ８０５では、関数「ＧｅｔＶａｌｕｅＡｓＳｔｒｉｎｇ」を実行し、ｎａｍｅタグ（要素）の内容、即ち、「Ａｌｉｃｅ」を文字列として取得する。ステップＳ８０５における処理の詳細については図１０を用いて後述する。

【００７４】

次にステップＳ８０６では、関数「Ｒｅａｄ」を実行し、係る実行により進めた現在の参照位置に対して関数「ＧｅｔＮｏｄｅＴｙｐｅ」、関数「ＧｅｔＮａｍｅ」を実行する。そして、関数「ＧｅｔＮｏｄｅＴｙｐｅ」の返り値がｈｅｉｇｈｔタグ、且つ関数「ＧｅｔＮａｍｅ」による返り値が「ｈｅｉｇｈｔ」となるまで、関数「Ｒｅａｄ」を実行する。

【００７５】

次にステップＳ８０７では、関数「ＧｅｔＶａｌｕｅＡｓＤｏｕｂｌｅ」を実行し、ｈｅｉｇｈｔタグ（要素）の内容、即ち、「１６０．５」を浮動小数形式の値として取得する。ステップＳ８０７における処理の詳細については図１０を用いて後述する。

【００７６】

次にステップＳ８０８では、関数「Ｃｌｏｓｅ」を実行し、ＲＡＭ１０３に対するメモリ資源等を解放する。

【００７７】

次に、上記ステップＳ８０２における処理が実行されると共に開始される処理について、同処理のフローチャートを示す図９を用いて以下説明する。図９のフローチャートに従

った処理は、ＣＰＵ１０１が共通ＸＭＬパーサ１０９のプログラムを実行することでなされる処理である。

【００７８】

ステップＳ９０２では、フォーマット判別部１０８を実行し、フォーマット判別部１０８に、上記ステップＳ８０２で開いたＸＭＬ文書におけるマジックナンバー（図６における６０１）を取得させる。そして、フォーマット判別部１０８が取得したマジックナンバーを、共通ＸＭＬパーサ１０９が取得する。そしてこの取得したマジックナンバーを用いて、ＸＭＬ文書のフォーマットを判別する。即ち、ＸＭＬ文書がテキストＸＭＬ文書であるのか、バイナリＸＭＬ文書であるのかを判別する。

【００７９】

ここで、係る判別では、マジックナンバーが「＜？」という文字列で始まっている場合、テキストＸＭＬ文書であると判別し、「０ｘ０１、０ｘ０２、０ｘ０３」という文字列で始まっている場合、バイナリＸＭＬ文書であると判別する。図６に示したＸＭＬ文書の場合、バイナリＸＭＬ文書と判別されることになる。

【００８０】

しかし、ＸＭＬ文書のフォーマットを判別する方法はこれに限定するものではなく、様々な方法が考えられる。例えば、ＨＴＴＰヘッダのＣｏｎｔｅｎｔ－Ｔｙｐｅフィールドの情報を参照することでフォーマットを判別したり、ＸＭＬ文書の拡張子を参照することでＸＭＬ文書のフォーマットを判別しても良い。

【００８１】

そして、ステップＳ９０２における判別処理の結果、テキストＸＭＬ文書であると判別された場合には、処理をステップＳ９０３を介してステップＳ９０４に処理を進める。一方、バイナリＸＭＬ文書であると判別された場合には、処理をステップＳ９０３を介してステップＳ９０５に進める。

【００８２】

ステップＳ９０４では、共通ＸＭＬパーサ１０９は、テキストＸＭＬパーサ１０５の関数「ＳｅｔＤｏｃｕｍｅｎｔ」を呼び出し、ＸＭＬ文書をテキストＸＭＬパーサ１０５に渡す。これにより、テキストＸＭＬパーサ１０５にこのＸＭＬ文書の解析を行わせる。

【００８３】

一方、ステップＳ９０５では、共通ＸＭＬパーサ１０９は、バイナリＸＭＬパーサ１０６の関数「ＳｅｔＤｏｃｕｍｅｎｔ」を呼び出し、ＸＭＬ文書をバイナリＸＭＬパーサ１０６に渡す。これにより、バイナリＸＭＬパーサ１０６にこのＸＭＬ文書の解析を行わせる。

【００８４】

テキストＸＭＬパーサ１０５、バイナリＸＭＬパーサ１０６は何れも、ＸＭＬ文書中（構造化文書中）に記述されている要素に対する解析処理を行う。即ち、ＸＭＬ文書のフォーマットに応じた解析処理を実現する。

【００８５】

共通ＸＭＬパーサ１０９の関数「Ｒｅａｄ」、「ＧｅｔＮｏｄｅＴｙｐｅ」、「ＧｅｔＮａｍｅ」、「Ｃｌｏｓｅ」は、テキストＸＭＬパーサ１０５、バイナリＸＭＬパーサ１０６の同名の関数をそのまま呼び出し、戻り値もそのまま渡すだけのラッパである。

【００８６】

次に、上記ステップＳ８０５，Ｓ８０７における処理の詳細について、図１０を用いて説明する。図１０は、ステップＳ８０５，Ｓ８０７における処理の詳細を示すフローチャートである。

【００８７】

先ず、ステップＳ１００２では、上記ステップＳ９０２における判別処理の結果として、テキストＸＭＬパーサ１０５、バイナリＸＭＬパーサ１０６の何れに解析処理を行わせているのかを判別する。係る判別の結果、現在テキストＸＭＬパーサ１０５に解析処理を行わせている場合には、処理をステップＳ１００８に進める。一方、現在バイナリＸＭＬ

パーサ１０６に解析処理を行わせている場合には、処理をステップＳ１００３に進める。図６に示したＸＭＬ文書の場合、バイナリＸＭＬパーサ１０６を用いてこのＸＭＬ文書に対する解析処理を行わせていることになるので、処理はステップＳ１００３に進むことになる。

【００８８】

ステップＳ１００３以降の処理を、ステップＳ８０５において行う場合と、ステップＳ８０７において行う場合とに分けて説明する。

【００８９】

先ず、ステップＳ８０５においてステップＳ１００３以降の処理を行う場合について説明する。

【００９０】

ステップＳ１００３では、関数「ＧｅｔＶａｌｕｅＴｙｐｅ」を実行することで、バイナリＸＭＬパーサ１０６が解析した結果を取得する。ステップＳ８０５では、関数「ＧｅｔＶａｌｕｅＡｓＳｔｒｉｎｇ」が実行されるので、バイナリＸＭＬパーサ１０６はｎａｍｅタグの型を取得することになり、図６に示したＸＭＬ文書の場合、ｓｔｒｉｎｇ型を取得する。従って、ステップＳ１００３では、このｓｔｒｉｎｇ型を「型情報」として取得する。

【００９１】

次に、ステップＳ１００４では、関数「ＧｅｔＳｔｒｉｎｇＶａｌｕｅ」を実行することで、バイナリＸＭＬパーサ１０６が解析した結果を取得する。ステップＳ８０５では、関数「ＧｅｔＶａｌｕｅＡｓＳｔｒｉｎｇ」が実行されるので、バイナリＸＭＬパーサ１０６はｎａｍｅタグの内容を取得することになり、図６に示したＸＭＬ文書の場合、文字列”Ａｌｉｃｅ”を取得する。従って、ステップＳ１００４では、この文字列”Ａｌｉｃｅ”を取得する。

【００９２】

次にステップＳ１００５では、ステップＳ８０５で実行した関数が要求する（受け付けた）データの型（依頼された型）と、ステップＳ１００３で取得した型とが一致しているか否かを判断する。係る判断の結果、一致する場合には、処理をステップＳ１００７に進める。図６に示したＸＭＬ文書の場合、ステップＳ８０５で実行した関数が要求するデータの型はｓｔｒｉｎｇ型であるし、ステップＳ１００３で取得した型もまたｓｔｒｉｎｇ型であるので、一致すると判断される。この場合、ステップＳ１００７では、上記ステップＳ１００４で取得したデータ（文字列）を、要求元（型対応アプリケーション１１１）に出力する。

【００９３】

一方、ステップＳ１００５における判断の結果、一致していない場合には、処理をステップＳ１００６に進める。ステップＳ１００６では、ステップＳ１００４で取得したデータの型を、ステップＳ８０５で実行した関数が要求するデータの型に変換する。そして、その後、ステップＳ１００７では、ステップＳ１００６で型を変換したデータを、上記要求元に対して出力する。

【００９４】

次に、ステップＳ８０７においてステップＳ１００３以降の処理を行う場合について説明する。

【００９５】

ステップＳ１００３では、関数「ＧｅｔＶａｌｕｅＴｙｐｅ」を実行することで、バイナリＸＭＬパーサ１０６が解析した結果を取得する。ステップＳ８０７では、関数「ＧｅｔＶａｌｕｅＡｓＤｏｕｂｌｅ」が実行されるので、バイナリＸＭＬパーサ１０６はｈｅｉｇｈｔタグの型を取得することになり、図６に示したＸＭＬ文書の場合、ｄｏｕｂｌｅ型を取得する。従って、ステップＳ１００３では、このｄｏｕｂｌｅ型を「型情報」として取得する。

【００９６】

次に、ステップＳ１００４では、関数「ＧｅｔＳｔｒｉｎｇＶａｌｕｅ」を実行することで、バイナリＸＭＬパーサ１０６が解析した結果を取得する。ステップＳ８０７では、関数「ＧｅｔＶａｌｕｅＡｓＤｏｕｂｌｅ」が実行されるので、バイナリＸＭＬパーサ１０６はｈｅｉｇｈｔタグの内容を取得することになり、図６に示したＸＭＬ文書の場合、実数値”１６０．５”を取得する。従って、ステップＳ１００４では、この実数値”１６０．５”を取得する。

【００９７】

次にステップＳ１００５では、ステップＳ８０７で実行した関数が要求するデータの型（依頼された型）と、ステップＳ１００３で取得した型とが一致しているか否かを判断する。係る判断の結果、一致する場合には、処理をステップＳ１００７に進める。図６に示したＸＭＬ文書の場合、ステップＳ８０７で実行した関数が要求するデータの型はｄｏｕｂｌｅ型であるし、ステップＳ１００３で取得した型もまたｄｏｕｂｌｅ型であるので、一致すると判断される。この場合、ステップＳ１００７では、上記ステップＳ１００４で取得したデータ（実数値）を、要求元（型対応アプリケーション１１１）に出力する。

【００９８】

一方、ステップＳ１００５における判断の結果、一致していない場合には、処理をステップＳ１００６に進める。ステップＳ１００６では、ステップＳ１００４で取得したデータの型を、ステップＳ８０７で実行した関数が要求するデータの型に変換する。そして、その後、ステップＳ１００７では、ステップＳ１００６で型を変換したデータを、上記要求元に対して出力する。

【００９９】

次に、図６に示したＸＭＬ文書の代わりに、図７に示した構成を有するＸＭＬ文書を処理対象とした場合における、コンピュータ１００の動作について説明する。図７は、コンピュータ１００による処理対象としてのＸＭＬ文書の構成例を示す図である。図７に示す構成を有するＸＭＬ文書は、図６に示したＸＭＬ文書と同様の内容が記述されている個人情報データであるが、図６に示したＸＭＬ文書がバイナリＸＭＬ文書であるのに対し、図７に示すＸＭＬ文書は、テキストＸＭＬ文書である。

【０１００】

タグ７０１は、このＸＭＬ文書がテキスト形式のものであることを示すものである。

【０１０１】

タグ７０２，７０３，７０５，７０６，７０８，７０９はそれぞれ、図６のタグ６０２，６０３，６０５，６０６，６０８，６０９に対応するもので、テキスト形式固有の表現となっている。

【０１０２】

７０４，７０５はそれぞれ、人の名前を示す文字列、身長を示す実数値、であり、内容が異なるのみで、実質的には図６の６０４，６０７と同じである。

【０１０３】

図７に示したＸＭＬ文書を処理対象とする場合に図８～図１０に示したフローチャートに従った処理を行う場合、図８～図１０ついて説明した上記処理と異なる点は以下の通りである。

【０１０４】

ステップＳ９０２では、フォーマット判別部１０８を実行し、フォーマット判別部１０８に、上記ステップＳ８０２で開いたＸＭＬ文書におけるマジックナンバー（図７における７０１）を取得させる。そして、フォーマット判別部１０８が取得したマジックナンバーを、共通ＸＭＬパーサ１０９が取得する。そしてこの取得したマジックナンバーを用いて、ＸＭＬ文書のフォーマットを判別する。即ち、ＸＭＬ文書がテキストＸＭＬ文書であるのか、バイナリＸＭＬ文書であるのかを判別する。図７に示したＸＭＬ文書の場合、テキストＸＭＬ文書と判別されることになる。従って、処理はステップＳ９０３を介してステップＳ９０４に進み、ステップＳ９０４では、共通ＸＭＬパーサ１０９は、テキストＸＭＬパーサ１０５の関数「ＳｅｔＤｏｃｕｍｅｎｔ」を呼び出し、ＸＭＬ文書をテキスト

ＸＭＬパーサ１０５に渡す。これにより、テキストＸＭＬパーサ１０５にこのＸＭＬ文書の解析を行わせる。

【０１０５】

先ず、ステップＳ１００２では、上記ステップＳ９０２における判別処理の結果として、テキストＸＭＬパーサ１０５、バイナリＸＭＬパーサ１０６の何れに解析処理を行わせているのかを判別する。図７に示したＸＭＬ文書の場合、テキストＸＭＬパーサ１０５を用いてこのＸＭＬ文書に対する解析処理を行わせていることになるので、処理はステップＳ１００８に進むことになる。

【０１０６】

ステップＳ１００８以降の処理を、ステップＳ８０５において行う場合と、ステップＳ８０７において行う場合とに分けて説明する。

【０１０７】

先ず、ステップＳ８０５においてステップＳ１００８以降の処理を行う場合について説明する。

【０１０８】

ステップＳ１００８では、関数「ＧｅｔＶａｌｕｅ」を実行することで、テキストＸＭＬパーサ１０５が解析した結果を取得する。ステップＳ８０５では、関数「ＧｅｔＶａｌｕｅＡｓＳｔｒｉｎｇ」が実行されるので、テキストＸＭＬパーサ１０５はｎａｍｅタグの内容を取得することになり、図７に示したＸＭＬ文書の場合、文字列”Ｂｏｂ”を取得する。従って、ステップＳ１００８では、この文字列”Ａｌｉｃｅ”を取得する。

【０１０９】

次にステップＳ１００９では、ステップＳ８０５で実行した関数が要求するデータの型（依頼された型）が、ｓｔｒｉｎｇ型（文字列型）若しくは指定無しであるか否かを判断する。係る判断の結果、ｓｔｒｉｎｇ型若しくは指定無しである場合には、処理をステップＳ１００７に進める。図７に示したＸＭＬ文書の場合、ステップＳ８０５で実行した関数が要求するデータの型はｓｔｒｉｎｇ型であるので、処理をステップＳ１００７に進める。ステップＳ１００７では、上記ステップＳ１００８で取得したデータ（文字列）を、要求元（型対応アプリケーション１１１）に出力する。

【０１１０】

一方、ステップＳ１００９における判断の結果、ｓｔｒｉｎｇ型若しくは指定無しではない場合には、処理をステップＳ１０１０に進める。ステップＳ１０１０では、上記ステップＳ１００６と同様の処理を行う。そして、その後、ステップＳ１００７では、ステップＳ１０１０で型を変換したデータを、上記要求元に対して出力する。

【０１１１】

次に、ステップＳ８０７においてステップＳ１００８以降の処理を行う場合について説明する。

【０１１２】

ステップＳ１００８では、関数「ＧｅｔＶａｌｕｅ」を実行することで、テキストＸＭＬパーサ１０５が解析した結果を取得する。ステップＳ８０７では、関数「ＧｅｔＶａｌｕｅＡｓＤｏｕｂｌｅ」が実行されるので、テキストＸＭＬパーサ１０５はｈｅｉｇｈｔタグの内容を取得することになり、図７に示したＸＭＬ文書の場合、文字列”１７５．３”を取得する。従って、ステップＳ１００８では、この文字列”１７５．３”を取得する。

【０１１３】

次にステップＳ１００９では、ステップＳ８０７で実行した関数が要求するデータの型（依頼された型）が、ｓｔｒｉｎｇ型（文字列型）若しくは指定無しであるか否かを判断する。係る判断の結果、ｓｔｒｉｎｇ型若しくは指定無しである場合には、処理をステップＳ１００７に進める。一方、ステップＳ１００９における判断の結果、ｓｔｒｉｎｇ型、指定無しの何れでもない場合には、処理をステップＳ１０１０に進める。

【０１１４】

図７に示したＸＭＬ文書の場合、ステップＳ８０７で実行した関数が要求するデータの型はｄｏｕｂｌｅ型（浮動小数点型）であり、ｓｔｒｉｎｇ型、指定無しの何れでもない。従ってこの場合、処理をステップＳ１０１０に進める。

【０１１５】

ステップＳ１０１０では、ステップＳ１００８で取得したデータの型を、ステップＳ８０７で実行した関数が要求するデータの型に変換する。その結果、ＩＥＥＥ７５４形式の１７５．３という浮動小数値を取得することができる。

【０１１６】

そして、その後、ステップＳ１００７では、ステップＳ１０１０で型を変換したデータを、上記要求元に対して出力する。

【０１１７】

次に、レガシーアプリケーション１１０の動作について説明する。レガシーアプリケーション１１０は元々、バイナリＸＭＬ文書を対象としていなかったものであるため、テキストＸＭＬパーサ１０５のＡＰＩを使用して作られている。このレガシーアプリケーション１１０が個人情報データを扱う場合にコンピュータ１００が行う処理は、図１１に示したフローチャートに従ったものとなる。

【０１１８】

図１１は、レガシーアプリケーション１１０が個人情報データを扱う場合にコンピュータ１００が行う処理のフローチャートである。

【０１１９】

ステップＳ１１０２～ステップＳ１１０４、ステップＳ１１０６、ステップＳ１１０８はそれぞれ、図８に示したステップＳ８０２～ステップＳ８０４、ステップＳ８０６、ステップＳ８０８と同じである。以下では、ステップＳ１１０５，Ｓ１１０７における処理について説明する。

【０１２０】

ステップＳ１１０５，Ｓ１１０７において、ノードの値はすべて関数「ＧｅｔＶａｌｕｅ」を用いて取得する。ステップＳ１１０５，Ｓ１１０７における処理の詳細は、図１０に示したフローチャートに従ったものとなる。

【０１２１】

この場合、テキストＸＭＬパーサ１０５を用いることになるので、ステップＳ１００２からステップＳ１００８に処理を進めることになる。

【０１２２】

ステップＳ１００８では、関数「ＧｅｔＶａｌｕｅ」を実行することで、テキストＸＭＬパーサ１０５が解析した結果を取得するので、ステップＳ１１０５では、図６に示したＸＭＬ文書の場合、文字列”Ａｌｉｃｅ”を取得する。従って、ステップＳ１００８では、この文字列”Ａｌｉｃｅ”を取得する。

【０１２３】

次に、ステップＳ１１０５で実行した関数が要求するデータの型（依頼された型）はｓｔｒｉｎｇ型であるので、処理をステップＳ１００９を介してステップＳ１００７に進める。ステップＳ１００７では、上記ステップＳ１００８で取得したデータ（文字列）を、要求元（型対応アプリケーション１１１）に出力する。

【０１２４】

また、ステップＳ１００８では、関数「ＧｅｔＶａｌｕｅ」を実行することで、テキストＸＭＬパーサ１０５が解析した結果を取得するので、ステップＳ１１０７では、図６に示したＸＭＬ文書の場合、文字列”１６０．５”を取得する。従って、ステップＳ１００８では、この文字列”１６０．５”を取得する。

【０１２５】

次に、ステップＳ１１０７で実行した関数が要求するデータの型（依頼された型）はｄｏｕｂｌｅ型（浮動小数点型）であり、ｓｔｒｉｎｇ型ではないので、処理をステップＳ１００９を介してステップＳ１０１０に進める。

【０１２６】
ステップＳ１０１０では、ステップＳ１００８で取得したデータの型を、ステップＳ１１０７で実行した関数が要求するデータの型に変換する。その結果、ＩＥＥＥ７５４形式の１６０．５という浮動小数値を取得することができる。
【０１２７】
そしてその後、ステップＳ１００７では、この浮動小数値１６０．５を上記要求元に対して出力する。
【０１２８】
このようにして、レガシーアプリケーション１１０はバイナリＸＭＬ文書から値を取得することができる。
【０１２９】
レガシーアプリケーション１１０に対してテキストＸＭＬ文書が渡された場合、共通ＸＭＬパーサ１０９は特別な処理を行わず、テキストＸＭＬパーサ１０５の単なるラッパとして振舞うことになるため、正常に値を取得することができる。
【０１３０】
以上説明したように、本実施形態によれば、共通ＸＭＬパーサ１０９は、２種類のアプリケーションと２種類の形式のＸＭＬ文書の組み合わせ、つまり計４つの場合のすべてにおいて正しく値を取得する機能を提供することができる。
【０１３１】
さらに、型対応アプリケーション１１１がバイナリＸＭＬ文書を扱う場合は、途中で値の型の変換が行われないため、無駄のない高速な処理が可能となる。これにより、ＸＭＬ文書を使用するアプリケーションにおいて、バイナリＸＭＬ文書を用いた高速な処理をサポートし、かつテキストＸＭＬ文書も扱うことを可能にする。
【０１３２】
また、テキストＸＭＬ文書用に作成されたアプリケーションでバイナリＸＭＬ文書を扱うことが可能になる。
【０１３３】
［第２の実施形態］
図１２は、本実施形態に係る構造化文書処理装置に適用可能なコンピュータ１２００のハードウェア構成を示すブロック図である。図１２において、図１に示したものと同じものについては同じ番号を付けており、その説明は省略する。即ち、図１２に示した構成は、図１に示したバイナリＸＭＬパーサ１０６の代わりに、Ｆａｓｔ　Ｉｎｆｏｓｅｔパーサ１２０６が記憶装置１０４内に保存されている構成となっている。
【０１３４】
Ｆａｓｔ　Ｉｎｆｏｓｅｔパーサ１２０６は、バイナリＸＭＬフォーマットの一つであるＦａｓｔ　Ｉｎｆｏｓｅｔ形式のＸＭＬ文書を解析するパーサである。
【０１３５】
本実施形態において、型対応アプリケーション１１１による処理対象となるＸＭＬ文書の一例を、図１３，１４に示す。図１４は、テキストＸＭＬ文書の構成例を示す図であり、図１３は、図１４に示したＸＭＬ文書をＦａｓｔ　Ｉｎｆｏｓｅｔ形式で表現した場合の構成例を示す図である。
【０１３６】
図１３において、１３０１で示す「Ｅ０００」はマジックナンバーであり、係るＸＭＬ文書がＦａｓｔ　Ｉｎｆｏｓｅｔ形式であることを示している。
【０１３７】
次の１３０２で示す「０００１」はＦａｓｔ　Ｉｎｆｏｓｅｔのバージョンであり、この例では１である。
【０１３８】
次の１３０３で示す「００」はオプションとなるデータの存在を示すものであり、「００」は存在しないことを意味する。

【０１３９】
次の１３０４で示す「３Ｃ００」は１ビットごとに意味を持つため多くの意味を持つが、主に次のノードが要素であることを意味している。他に、属性の有無や名前空間名の存在、要素名のバイト数などの情報も含まれているが、ここでの説明の本質とは関連が薄いため詳細な説明は省略する。
【０１４０】
次の１３０５で示す「６１」はＵＴＦ－８でエンコードされた要素名「ａ」である。
【０１４１】
次の１３０６で示す「９Ｃ１Ａ」という２バイトも同様に多くの意味を持つが、主に次のノードが要素の内容であり、またその値が浮動小数型であることを意味している。その他にもバイト数などの情報も含まれている。
【０１４２】
次の１３０７で示す「Ｃ２ＥＤ４０００」はＩＥＥＥ７５４形式でエンコードされた－１１８．６２５という浮動小数の値である。
【０１４３】
最後の１３０８で示す「ＦＦ」は、はじめのＦが要素の終端、次のＦが文書の終端を表している。すなわち、図１３に示したＸＭＬ文書は、図１４のテキストＸＭＬ文書とほぼ同じ意味である。また文書の意味だけでなく、ノードの出現する順序も同じである。
【０１４４】
ここで、型対応アプリケーション１１１が、図１３に示したａ要素の値を取得する場合は、図１５に示したフローチャートに従った処理を、共通ＸＭＬパーサ１０９が行う。
【０１４５】
図１５は、型対応アプリケーション１１１が、図１３に示したａ要素の値を取得する場合に、コンピュータ１２００が行う処理のフローチャートである。
【０１４６】
先ず、ステップＳ１５０２では、関数「ＳｅｔＤｏｃｕｍｅｎｔ」を実行し、図１３に示すＸＭＬ文書を開く。なお、ステップＳ１５０２における処理を行うと、図９に示したフローチャートに従った処理が第１の実施形態と同様に開始される。なお、図９のフローチャートに従った処理において、フォーマットの判別処理は、Ｆａｓｔ　Ｉｎｆｏｓｅｔ形式であるか否かを判断するのであるが、これは、マジックナンバーとして「Ｅ０００」が記述されているのかを判断することで行う。マジックナンバーとして「Ｅ０００」が記述されていれば、Ｆａｓｔ　Ｉｎｆｏｓｅｔパーサ１２０６を使用する。記述されていなければ、テキストＸＭＬパーサ１０５を使用する。
【０１４７】
次にステップＳ１５０３では、関数「Ｒｅａｄ」を実行し、係る実行により進めた現在の参照位置に対して関数「ＧｅｔＮｏｄｅＴｙｐｅ」、関数「ＧｅｔＮａｍｅ」を実行する。そして、関数「ＧｅｔＮｏｄｅＴｙｐｅ」の返り値が開始タグ、且つ関数「ＧｅｔＮａｍｅ」による返り値が「ａ」となるまで、関数「Ｒｅａｄ」を実行する。Ｆａｓｔ　Ｉｎｆｏｓｅｔ形式もテキストＸＭＬ形式と同様、まず要素の開始を表すバイト列と要素の名前を示すバイト列が現れるため、最初のノードは開始タグａとなる。
【０１４８】
次にステップＳ１５０４では、関数「ＧｅｔＶａｌｕｅＡｓＤｏｕｂｌｅ」を実行し、ａタグ（要素）の内容、即ち、「－１１８．６２５」を実数値として取得する。ステップＳ１５０４における処理の詳細については、図１０に示したフローチャートに従ったものとなる。
【０１４９】
即ち、Ｆａｓｔ　Ｉｎｆｏｓｅｔパーサ１２０６を使用しているため、先ず、ステップＳ１００３では、Ｆａｓｔ　Ｉｎｆｏｓｅｔパーサ１２０６からデータの型情報を受け取る。Ｆａｓｔ　Ｉｎｆｏｓｅｔパーサ１２０６は図１３の１３０６で示す「９Ｃ１Ａ」の部分により、このデータの値が浮動小数であることを判断するので、ｄｏｕｂｌｅ型を型

情報として返す。そしてステップＳ１００４では、その型で値、つまり「－１１８．６２５」を取得する。

【０１５０】

次にステップＳ１００５では、ステップＳ１５０４で実行した関数が要求するデータの型（依頼された型）と、ステップＳ１００３で取得した型とが一致しているか否かを判断する。係る判断の結果、一致する場合には、処理をステップＳ１００７に進める。図１３に示したＸＭＬ文書の場合、ステップＳ１５０４で実行した関数が要求するデータの型はｄｏｕｂｌｅ型であるし、ステップＳ１００３で取得した型もまたｄｏｕｂｌｅ型であるので、一致すると判断される。この場合、ステップＳ１００７では、上記ステップＳ１００４で取得したデータ（実数値）を、要求元（型対応アプリケーション１１１）に出力する。

【０１５１】

これにより、無駄な変換を行うことなく、アプリケーションにデータを渡すことができる。

【０１５２】

なお、図１４に示したテキストＸＭＬ文書を処理対象とする場合であっても、第１の実施形態と同様に処理すれば良い。

【０１５３】

以上のようにして、従来のテキストＸＭＬ形式のＸＭＬ文書とＦａｓｔ　Ｉｎｆｏｓｅｔ形式の文書の両方に対応可能で、かつ無駄なデータ型変換を行わずに処理が可能な構造化文書処理装置が実現できる。

【０１５４】

ここで、上記コンピュータ１００、１２００としては、携帯電話や複写機など、ＸＭＬ文書が使用可能な通信機器を用いることができる。

【０１５５】

［その他の実施形態］

また、本発明の目的は、以下のようにすることによって達成されることはいうまでもない。即ち、前述した実施形態の機能を実現するソフトウェアのプログラムコードを記録した記録媒体（または記憶媒体）を、システムあるいは装置に供給する。係る記憶媒体は言うまでもなく、コンピュータ読み取り可能な記憶媒体である。そして、そのシステムあるいは装置のコンピュータ（またはＣＰＵやＭＰＵ）が記録媒体に格納されたプログラムコードを読み出し実行する。この場合、記録媒体から読み出されたプログラムコード自体が前述した実施形態の機能を実現することになり、そのプログラムコードを記録した記録媒体は本発明を構成することになる。

【０１５６】

また、コンピュータが読み出したプログラムコードを実行することにより、そのプログラムコードの指示に基づき、コンピュータ上で稼働しているオペレーティングシステム（ＯＳ）などが実際の処理の一部または全部を行う。その処理によって前述した実施形態の機能が実現される場合も含まれることは言うまでもない。

【０１５７】

さらに、記録媒体から読み出されたプログラムコードが、コンピュータに挿入された機能拡張カードやコンピュータに接続された機能拡張ユニットに備わるメモリに書込まれたとする。その後、そのプログラムコードの指示に基づき、その機能拡張カードや機能拡張ユニットに備わるＣＰＵなどが実際の処理の一部または全部を行い、その処理によって前述した実施形態の機能が実現される場合も含まれることは言うまでもない。

【０１５８】

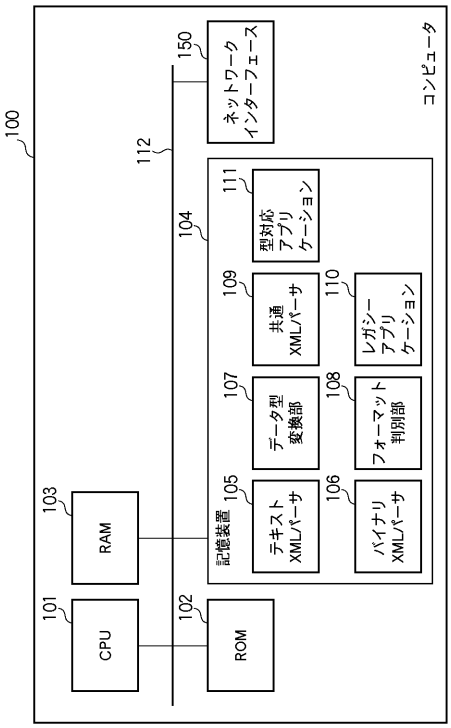
本発明を上記記録媒体に適用する場合、その記録媒体には、先に説明したフローチャートに対応するプログラムコードが格納されることになる。

【図面の簡単な説明】

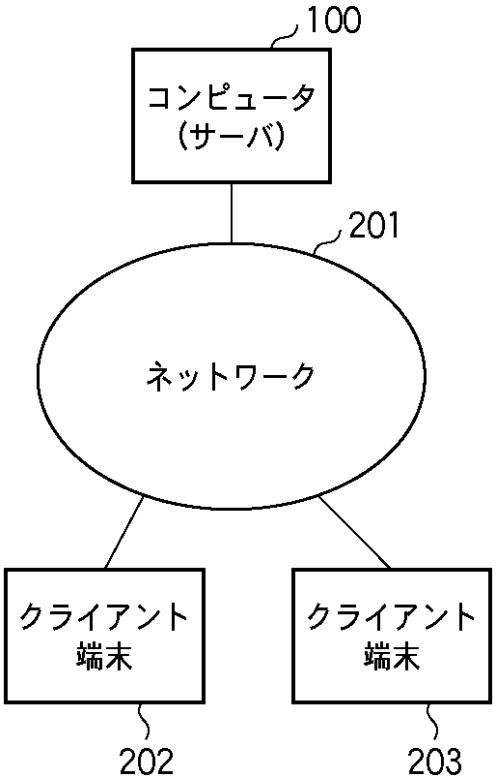
【０１５９】

【図１】本発明の第１の実施形態に係る構造化文書処理装置に適用可能なコンピュータのハードウェア構成例を示すブロック図である。
【図２】コンピュータ１００を適用したネットワークの構成例を示す図である。
【図３】テキストＸＭＬパーサ１０５のＡＰＩの一例を示す図である。
【図４】バイナリＸＭＬパーサ１０６のＡＰＩの一例を示す図である。
【図５】共通ＸＭＬパーサ１０９のＡＰＩの一例を示す図である。
【図６】コンピュータ１００による処理対象としてのＸＭＬ文書の構成例を示す図である。
【図７】コンピュータ１００による処理対象としてのＸＭＬ文書の構成例を示す図である。
【図８】ＣＰＵ１０１が、型対応アプリケーション１１１のプログラムを実行することでなされる処理のフローチャートである。
【図９】ステップＳ８０２における処理が実行されると共に開始される処理のフローチャートである。
【図１０】ステップＳ８０５，Ｓ８０７における処理の詳細を示すフローチャートである。
【図１１】レガシーアプリケーション１１０が個人情報データを扱う場合にコンピュータ１００が行う処理のフローチャートである。
【図１２】本発明の第２の実施形態に係る構造化文書処理装置に適用可能なコンピュータ１２００のハードウェア構成を示すブロック図である。
【図１３】図１４に示したＸＭＬ文書をＦａｓｔ　Ｉｎｆｏｓｅｔ形式で表現した場合の構成例を示す図である。
【図１４】テキストＸＭＬ文書の構成例を示す図である。
【図１５】型対応アプリケーション１１１が、図１３に示したａ要素の値を取得する場合に、コンピュータ１２００が行う処理のフローチャートである。

【図１】



【図２】



【図３】

関数名	戻り値の型	関数の意味
SetDocument	bool	解析するXML文書を渡す
Read	bool	次のノードに進む
GetNodeType	nodetype	ノードの型を取得する
GetName	string	ノードの名前を取得する
GetValue	string	ノードの値を取得する
Close	void	解析を終え、リソースを解放する

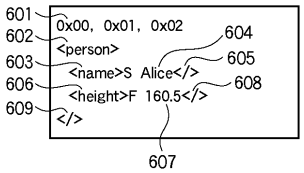
【図４】

関数名	戻り値の型	関数の意味
SetDocument	bool	解析するXML文書を渡す
Read	bool	次のノードに進む
GetNodeType	nodetype	ノードの型を取得する
GetName	string	ノードの名前を取得する
GetValueType	valuetype	ノードの値の型を取得する
GetStringValue	string	文字列型の値を取得する
GetIntValue	int	整数型の値を取得する
GetDoubleValue	double	浮動小数型の値を取得する
Close	void	解析を終え、リソースを解放する

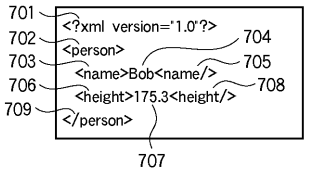
【図５】

関数名	戻り値の型	関数の意味
SetDocument	bool	解析するXML文書を渡す
Read	bool	次のノードに進む
GetNodeType	nodetype	ノードの型を取得する
GetName	string	ノードの名前を取得する
GetValueType	valuetype	ノードの値の型を取得する
GetValueAsString	string	ノードの値を文字列として取得する
GetValueAsInt	int	ノードの値を整数として取得する
GetValueAsDouble	double	ノードの値を浮動小数として取得する
Close	void	解析を終え、リソースを解放する

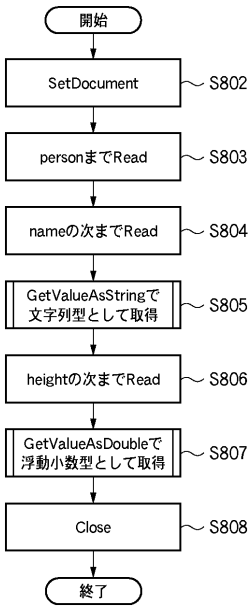
【図６】



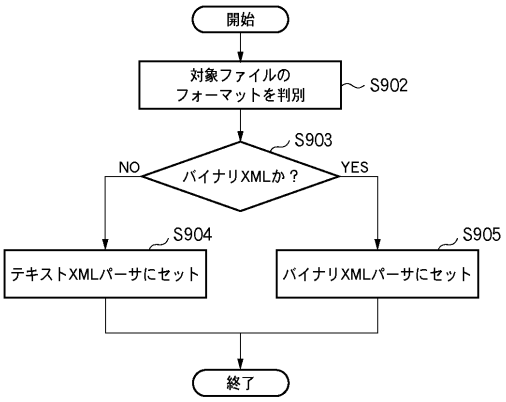
【図７】



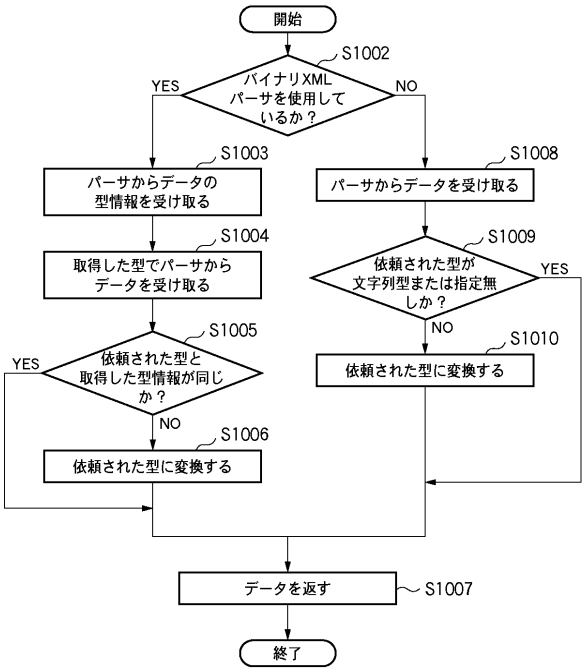
【図８】



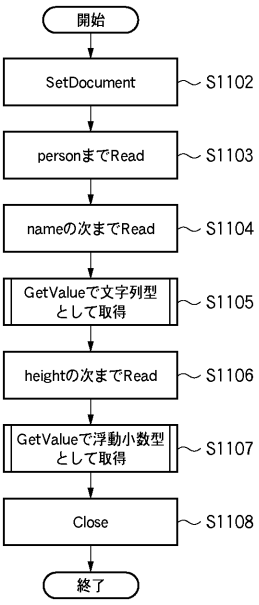
【図９】



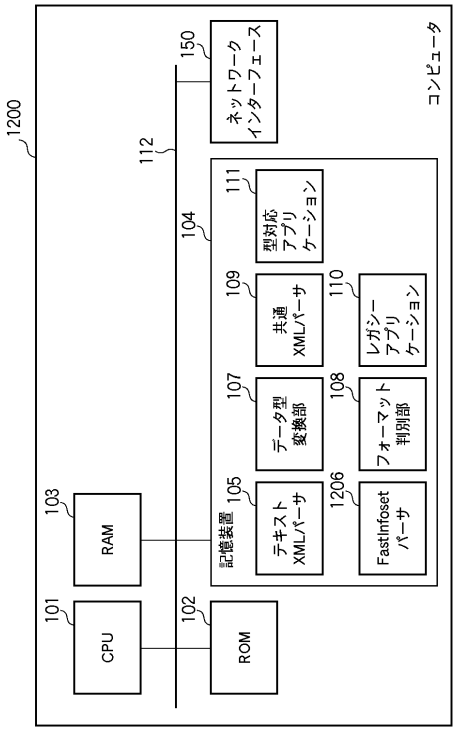
【図１０】



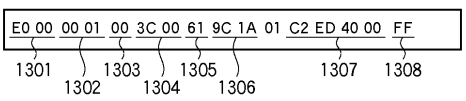
【図１１】



【図１２】



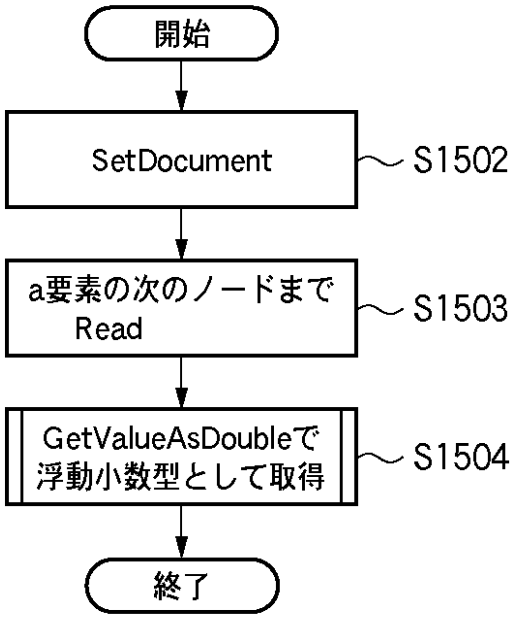
【図１３】



【図１４】

```
<?xml version="1.0"?>
<a>−118.625</a>
```

【図１５】



フロントページの続き

(72)発明者　清水　渉
東京都大田区下丸子３丁目３０番２号　キヤノン株式会社内

Ｆターム(参考) 5B009 NA05 QA06
5B075 ND03 UU06
5B082 GA02
5B109 NA05 QA06