(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2006-92529
(P2006-92529A)
(43)公開日　平成18年4月6日(2006.4.6)

(51)Int.Cl.		F I		テーマコード（参考）
G06F 17/21	(2006.01)	G06F 17/21	501T	5B009

審査請求 未請求　請求項の数 20　ＯＬ　外国語出願　（全 49 頁）

(21)出願番号　特願2005-240380 (P2005-240380)
(22)出願日　平成17年8月22日 (2005.8.22)
(31)優先権主張番号　10/925,350
(32)優先日　平成16年8月23日 (2004.8.23)
(33)優先権主張国　米国 (US)

（特許庁注：以下のものは登録商標）
１．ＪＡＶＡ

(71)出願人　304038459
サン・マイクロシステムズ・インコーポレーテッド
ＳＵＮ　ＭＩＣＲＯＳＹＳＴＥＭＳ　ＩＮＣＯＲＰＯＲ
アメリカ合衆国　カリフォルニア州９５０５４　サンタ・クララ，ネットワーク・サークル，４１２０，エム／エス：エスシーエー　１２－２０３
(74)代理人　110000028
特許業務法人明成国際特許事務所
(72)発明者　アユブ・エス．・カーン
アメリカ合衆国　カリフォルニア州９５０５０　サンタ・クララ，ドン・アベニュー，１９６０，アパートメント　＃１

Ｆターム(参考)　5B009 QA06

(54)【発明の名称】ＸＭＬ入力文書を検証するＸＭＬスキーマを自動的に生成するシステムおよび方法

(57)【要約】
【課題】ＸＭＬ互換フォーマットで構築された初期文書を用いてスキーマを自動的に生成する技術、システムおよび装置を提供する。
【解決手段】本方法は、初期ＸＭＬ文書を与え、ＸＭＬ文書を分析して文書内のＸＭＬデータ構造を特定し、ＸＭＬ文書内のデータ構造のフォーマットに対応するデータフレームワークを生成することを伴う。初期ＸＭＬ文書のデータアイテムが分析され、初期ＸＭＬのデータアイテムに基づいてデータ制約を決定する。それから生成されたデータフレームワークおよび生ＸＭＬデータから決定されたデータ制約に基づいてスキーマが生成される。これら原理は、コンピュータシステム上で動作するソフトウェアとして、コンピュータモジュールとして、コンピュータプログラム製造物として、および一連の関連するデバイスおよび製造物として実現されえる。
【選択図】図３

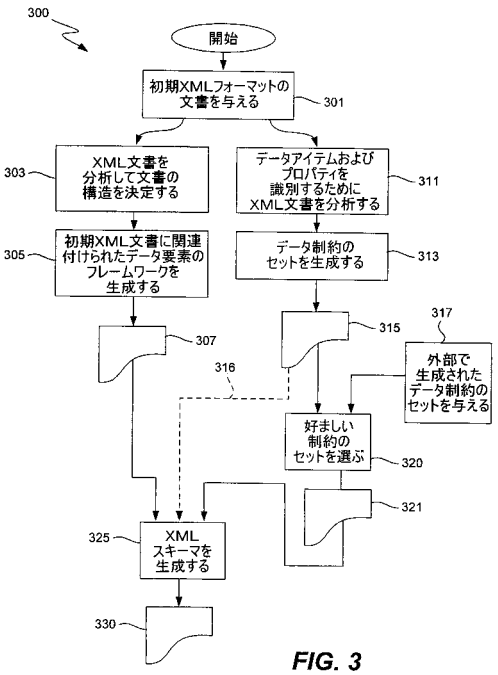

FIG. 3

【特許請求の範囲】
【請求項１】
ＸＭＬスキーマを生成する方法であって、
ＸＭＬデータ要素として構成されたデータアイテムを含む生ＸＭＬデータを含む初期ＸＭＬ文書を与えること、
前記ＸＭＬ文書を分析してＸＭＬデータ構造を特定すること、
前記ＸＭＬ文書内の前記データ構造の前記フォーマットに対応するデータフレームワークを生成すること、
前記初期ＸＭＬ文書から前記データアイテムを分析すること、
前記データアイテムに基づいてデータ制約を決定すること、
前記生成されたデータフレームワークおよび前記生ＸＭＬデータから決定された前記データ制約に基づいてＸＭＬスキーマを生成すること
を含む方法。
【請求項２】
請求項１に記載の方法であって、前記データアイテムに基づいて前記データ制約を決定することは、前記初期ＸＭＬ文書内の前記データアイテムの前記タイプに基づいてタイプ制約を決定することを含む方法。
【請求項３】
請求項１に記載の方法であって、前記データアイテムに基づいて前記データ制約を決定することは、前記初期ＸＭＬ文書内の前記データアイテムの前記属性に基づいて属性制約を決定することを含む方法。
【請求項４】
請求項１に記載の方法であって、前記方法は、ＸＭＬスキーマについてのネームスペースを変化させるよう実現されえる方法。
【請求項５】
請求項１に記載の方法であって、前記方法は、ＸＭＬスキーマについてのデータタイプを変化させるよう実現されえる方法。
【請求項６】
請求項１に記載の方法であって、前記方法は、ＸＭＬスキーマからのデータ要素を追加または除去するよう実現されえる方法。
【請求項７】
請求項１に記載の方法であって、データ制約を決定することは、外部で供給されるデータ制約を受け取ることを含み、
前記スキーマを生成することは、（ｉ）前記初期ＸＭＬ文書内の前記データから決定された前記データ制約、または（ｉｉ）前記外部から供給されるデータ制約を用いて前記ＸＭＬスキーマを生成することのうちの一つを用いることをさらに含む
方法。
【請求項８】
請求項１に記載の方法であって、データ制約を決定することは、外部で供給されるデータ制約を受け取ることを含み、
前記スキーマを生成することは、（ｉ）前記初期ＸＭＬ文書内の前記データから決定された前記データ制約、および（ｉｉ）前記外部から供給されるデータ制約を用いて前記ＸＭＬスキーマを生成することの少なくとも一つを用いることをさらに含む
方法。
【請求項９】
請求項１に記載の方法であって、データ制約を決定することは、外部で供給されるデータ制約を受け取ることを含み、
前記スキーマを生成することは、前記初期ＸＭＬ文書内の前記データから決定された前記データ制約の一部、および前記外部から供給されるデータ制約の一部をマージすることをさらに含む

方法。
【請求項１０】
ＸＭＬスキーマを生成するコンピュータプログラムコードを含むコンピュータで読み取り可能な媒体上で実現されるコンピュータプログラムであって、前記コンピュータプログラムは、
ＸＭＬデータ要素として構成されたデータアイテムを含む生ＸＭＬデータを含む初期ＸＭＬ文書を受け取るコンピュータプログラムコード命令、
前記ＸＭＬ文書を分析してＸＭＬデータ構造を特定するコンピュータプログラムコード命令、
前記ＸＭＬ文書内の前記データ構造の前記フォーマットに対応するデータフレームワークを生成するコンピュータプログラムコード命令、
前記初期ＸＭＬ文書から前記データアイテムを分析し、前記データアイテムに基づいてデータ制約を決定するコンピュータプログラムコード命令、および
前記生成されたデータフレームワークおよび前記生ＸＭＬデータから決定された前記データ制約に基づいてＸＭＬスキーマを生成するコンピュータプログラムコード命令
を含むコンピュータプログラム。
【請求項１１】
請求項１０に記載のコンピュータプログラムであって、前記データ制約を決定するコンピュータプログラムコード命令は、外部で供給されるデータ制約を受け取ることを含み、
前記スキーマを生成するコンピュータプログラムコード命令は、（ｉ）前記初期ＸＭＬ文書内の前記データから決定された前記データ制約、または（ｉｉ）前記外部から供給されるデータ制約を用いて前記ＸＭＬスキーマを生成することのうちの一つを用いることをさらに含む
コンピュータプログラム。
【請求項１２】
請求項１０に記載のコンピュータプログラムであって、前記データ制約を決定するコンピュータプログラムコード命令は、外部で供給されるデータ制約を受け取るコンピュータプログラムコード命令を含み、
前記スキーマを生成するコンピュータプログラムコード命令は、（ｉ）前記初期ＸＭＬ文書内の前記データから決定された前記データ制約、および（ｉｉ）前記外部から供給されるデータ制約を用いて前記ＸＭＬスキーマを生成することの少なくとも一つを用いることをさらに含む
コンピュータプログラム。
【請求項１３】
請求項１０に記載のコンピュータプログラムであって、前記データ制約を決定するコンピュータプログラムコード命令は、外部で供給されるデータ制約を受け取るコンピュータプログラムコード命令を含み、
前記スキーマを生成するコンピュータプログラムコード命令は、前記初期ＸＭＬ文書内の前記データから決定された前記データ制約の一部、および前記外部から供給されるデータ制約の一部をマージする命令をさらに含む
コンピュータプログラム。
【請求項１４】
コンピュータシステムであって、
少なくとも１つの中央処理ユニット（ＣＰＵ）、メモリ、およびユーザインタフェースが組み合わせられて、
ＸＭＬデータ要素として構成された生ＸＭＬデータを含む受け取られた初期ＸＭＬ文書を分析するＸＭＬ構造アナライザであって、前記分析は、前記初期ＸＭＬ文書の前記ＸＭＬデータ要素を特定すること、および前記初期ＸＭＬ文書の前記ＸＭＬデータ要素についてのデータ構造に関連付けられたデータフレームワークを生成することを含み、
前記初期ＸＭＬ文書からの前記生ＸＭＬデータを分析し、前記ＸＭＬ生データに基づい

てデータアイテムについてのＸＭＬデータ制約を決定するデフォールト制約ジェネレータ、

前記データ制約ジェネレータからの前記ＸＭＬデータ制約および外部から供給されるデータ制約のセットのうちの少なくとも１つを受け取り、最終データ制約ファイルを出力するデータ制約マージャ、および

前記ＸＭＬ構造アナライザからの前記データ構造および前記データ制約マージャからの最終データ制約ファイルを受け取り、前記データ構造および最終データ制約ファイルに関連付けられたＸＭＬスキーマを生成するＸＭＬスキーマジェネレータ

を含むコンピュータシステム。

【請求項１５】

請求項１４に記載のコンピュータシステムであって、前記データ制約マージャは、前記データ制約ジェネレータからの前記ＸＭＬデータ制約および前記外部から供給されたデータ制約のセットの両方を受け取り、データ制約の両方のセットを前記最終データ制約ファイルにマージするコンピュータシステム。

【請求項１６】

請求項１４に記載のコンピュータシステムであって、前記データ制約マージャは、前記データ制約ジェネレータからの前記ＸＭＬデータ制約および前記外部から供給されたデータ制約のセットの両方を受け取り、前記データ制約ジェネレータからの前記ＸＭＬデータ制約および前記外部から供給されたデータ制約のセットのうちの１つを選択し、前記最終データ制約ファイルとして出力するコンピュータシステム。

【請求項１７】

ＸＭＬスキーマを自動的に生成するコンピュータモジュールであって、

ＸＭＬデータ要素として構成された生ＸＭＬデータを含む受け取られた初期ＸＭＬ文書を分析するＸＭＬ構造アナライザであって、前記分析は、前記初期ＸＭＬ文書の前記ＸＭＬデータ要素を特定すること、および前記初期ＸＭＬ文書の前記ＸＭＬデータ要素についてのデータ構造に関連付けられたデータフレームワークを生成することを含み、

前記初期ＸＭＬ文書からの前記生ＸＭＬデータを分析し、前記ＸＭＬ生データに基づいてデータアイテムについてのＸＭＬデータ制約を決定するデフォールト制約ジェネレータ、

前記データ制約ジェネレータからの前記ＸＭＬデータ制約および外部から供給されるデータ制約のセットのうちの少なくとも１つを受け取り、最終データ制約ファイルを出力するデータ制約マージャ、および

前記ＸＭＬ構造アナライザからの前記データ構造および前記データ制約マージャからの最終データ制約ファイルを受け取り、前記データ構造および最終データ制約ファイルに関連付けられたＸＭＬスキーマを生成するＸＭＬスキーマジェネレータ

を含むコンピュータモジュール。

【請求項１８】

請求項１７に記載のコンピュータモジュールであって、前記データ制約マージャは、前記データ制約ジェネレータからの前記ＸＭＬデータ制約および前記外部から供給されたデータ制約のセットの両方を受け取り、データ制約の両方のセットを前記最終データ制約ファイルにマージするコンピュータモジュール。

【請求項１９】

請求項１７に記載のコンピュータモジュールであって、前記データ制約マージャは、前記データ制約ジェネレータからの前記ＸＭＬデータ制約および前記外部から供給されたデータ制約のセットの両方を受け取り、前記データ制約ジェネレータからの前記ＸＭＬデータ制約および前記外部から供給されたデータ制約のセットのうちの１つを選択し、前記最終データ制約ファイルとして出力するコンピュータモジュール。

【請求項２０】

ＸＭＬスキーマを自動的に生成するコンピュータモジュールであって、

ＸＭＬデータ要素として構成された生ＸＭＬデータを含む受け取られた初期ＸＭＬ文書

を分析するＸＭＬ構造アナライザであって、前記分析は、前記初期ＸＭＬ文書の前記ＸＭＬデータ要素を特定すること、および前記初期ＸＭＬ文書の前記ＸＭＬデータ要素についてのデータ構造に関連付けられたデータフレームワークを生成することを含み、

前記初期ＸＭＬ文書からの前記生ＸＭＬデータを分析し、前記ＸＭＬ生データに基づいてデータアイテムについてのＸＭＬデータ制約を決定するデフォールト制約ジェネレータ、

前記初期ＸＭＬ文書からの前記生ＸＭＬデータを分析し、前記ＸＭＬ生データに基づいてデータアイテムについてのＸＭＬデータ制約を決定するデフォールト制約ジェネレータ、および

前記ＸＭＬ構造アナライザからの前記データ構造および前記データ制約マージャからの最終データ制約ファイルを受け取り、前記データ構造および最終データ制約ファイルに関連付けられたＸＭＬスキーマを生成するＸＭＬスキーマジェネレータ

を含むコンピュータモジュール。

【発明の詳細な説明】

【背景技術】

【０００１】

ＸＭＬ（Extensible Markup Language）は、データ送信および処理ツールとしてますます広い応用例が見つかる自己記述型マークアップ言語である。ＸＭＬはデータを記述し定義するのに効率的であり、ステージますますデータ集約的アプリケーションに用いられる。このようにしてＸＭＬは、データを表示するよう設計されたＨＴＭＬとは異なる。

【０００２】

ＸＭＬデータは、任意のタイプのデータ転送媒体を用いて容易にコンピュータ間で転送されえる。このＸＭＬデータは、コンピュータプログラムおよび他の適切に構成されたアプリケーションを用いて処理されえる。一般にはＸＭＬファイルは、アプリケーションによって受け取られ、出力を生成するよう処理する。例えば、ある実現例において、ＸＭＬは、在庫情報を提供するように用いられえる。そのような情報は、ＸＭＬ準拠文書（ここではxml.docのＸＭＬ文書と呼ばれる）。ある実現例ではそのような情報は例えば以下のようでありえる。

【数１】

```
<Camera>
    <name>Canon-Sure-Shot-Z155</name>
    <f-stop>4.8-11.7</f-stop>
    <focal length>37-155mm zoom</ focal length >
    <cost>$318.00USD</cost>
</Camera>
```

【０００３】

ＸＭＬ文書のこの簡略化された例は、カメラに関する在庫情報の一例を与える。ＸＭＬ文書内では、データアイテムは、ＸＭＬ要素の部分としてフォーマットされ、ＸＭＬ文書は１つ以上のそのような要素を含む。ＸＭＬにおいて、データアイテムは、スタート／エンドタグで「包まれて」ＸＭＬ要素を形成する。例えばスタートタグ「<name>」およびエンドタグ「</name>」はデータ要素「Canon-Sure-Shot-Z155」を包み、ＸＭＬ要素「<name>Canon-Sure-Shot-Z155</name>」を形成する。

【０００４】

より複雑な要素は、例えば< camera > ... </camera>によって定義される要素を用いて

定義されえる。ＸＭＬデータを記述するのに用いられる方法およびフォーマットは、もちろん当業者にはよく知られており、よってここでは詳細には議論されない。

【０００５】

上述の例を用いて、ＸＭＬ情報は、ＸＭＬデータ構造に適合する。ここで用いられるように、ＸＭＬデータ構造とは「空の」データ要素の配置および構成のフォーマットを言う。このような構造は、与えられたＸＭＬ文書内での要素の互いの関係を定義する配置およびフォーマットによって定義される。再び、上述の例を用いて、ＸＭＬ文書の構造は、以下のようにフォーマットされえる。

【数２】

```
<Camera>
    <name> ...</name>
    <f-stop> ... </f-stop>
    <focal length> ... </ focal length >
    <cost> ... </cost>
</Camera>
```

【０００６】

前述の簡略化された例は、例示的文書についてのデータ構造を定義する例示的データのフレームワークを定義する。

【０００７】

多くの実現例において、ＸＭＬ文書はデータを、ＸＭＬデータを用いてさまざまな操作を実行するアプリケーションに提供するのに用いられる。一般にはそのようなアプリケーションは、特定のフォーマットを有するＸＭＬデータを与えられた順序で受け取るよう構成される。もしデータが不正な順序で、または不適切なフォーマットで与えられるなら、それはアプリケーションによって使用不可能になるかもしれない。不適切に構成されたＸＭＬデータはアプリケーションプログラムを動作不能にしたり、クラッシュさせたり、または他の不要な結果を引き起こしたりしえる。このような状況では、ＸＭＬ文書（および関連付けられたデータ）は、「不当」と考えられる。したがってアプリケーションは、ふつうは、受け取られたＸＭＬ文書を「検証」する小さなプログラムを備える。もしＸＭＬ文書が適切な順序の、かつ正しいフォーマットのＸＭＬデータを含むなら、それは妥当であるであるとされ、アプリケーションはそのデータに操作を行いえる。ＸＭＬ文書を検証するのに用いられる一つのアプローチは、ＸＭＬデータを検証するためにＸＭＬスキーマ（.xsdファイルとも呼ばれる）を用いることである。検証スキーマは、アプリケーションの一部として含まれえ、またはアドオン検証モジュールとして用いられえる。ＸＭＬスキーマは、ＸＭＬ文書の構造を記述するのに用いられる。当業者には知られるように、ＸＭＬスキーマは、文書に現れる要素および属性を定義するのに有用である。ＸＭＬスキーマは、要素が子要素であるか、および子要素の個数および順序を定義するのに用いられえる。ＸＭＬスキーマはまた、要素が空であるか、またはテキストを含みえるかを定義しえ、また要素および属性についてのデータタイプを定義しえ、それと共に要素および属性についてのデフォールトおよび固定された値を定義しえる。これら属性はＸＭＬ文書を定義および検証するのに非常に有用である。

【発明の開示】

【発明が解決しようとする課題】

【０００８】

しかし普通の使用においては、ＸＭＬ文書のデータおよび構造は、常に変化している。

加えて、データおよび構造のそれぞれの変化は典型的には関連付けられたＸＭＬスキーマにおける対応する変化を必然的に伴う。現在の技術においては、そのようなスキーマは、その変更を実現するために手で変更またはリライトされなければならない。ここで上に挙げられた例では、そのような変更は比較的簡単に達成されえる。しかしたいていの検証スキーマは、多くの要素があって非常に長く、適切なスキーマを注意深くかつ正確に生成するプロセスは非常に時間がかかりえる。ＸＭＬ文書を検証できる正確なスキーマを生成する従来のプロセスを用いることは、労力がかかり、細かく、時間を使うプロセスである。

【課題を解決するための手段】

【０００９】

大まかに言えば本発明は、手による記録の必要なしにスキーマを自動的に生成する技術、システムおよび装置に関する。特に本発明の実施形態は、ＸＭＬ互換フォーマットで構築された初期文書を用いてスキーマを自動的に生成しえる。

【００１０】

本発明は、システム、ソフトウェアモジュール、方法、またはコンピュータで読み取り可能な媒体および他の実現例を含む多くのやり方で実現されえる。本発明のいくつかの実現例が以下に説明される。

【００１１】

本発明のある実施形態は、ＸＭＬスキーマを生成する方法に関する。この方法は、ＸＭＬデータ要素として構成されたデータアイテムを含む生ＸＭＬデータを含む初期ＸＭＬ文書を与え、前記ＸＭＬ文書を分析してＸＭＬデータ構造を特定し、前記ＸＭＬ文書内の前記データ構造の前記フォーマットに対応するデータフレームワークを生成する操作を伴う。この方法は、前記初期ＸＭＬ文書から前記データアイテムを分析し、前記データアイテムに基づいてデータ制約を決定することを伴う。前記生成されたデータフレームワークおよび前記生ＸＭＬデータから決定された前記データ制約に基づいてＸＭＬスキーマが生成される。

【００１２】

他の実施形態において本開示は、ＸＭＬスキーマを生成するコンピュータプログラムコードを含むコンピュータで読み取り可能な媒体上で実現されるコンピュータプログラムを教示する。前記コンピュータプログラムは、ＸＭＬデータ要素として構成されたデータアイテムを含む生ＸＭＬデータを含む初期ＸＭＬ文書を受け取るコンピュータプログラムコード命令を含む。前記コンピュータプログラムは、前記ＸＭＬ文書を分析してＸＭＬデータ構造を特定するコンピュータプログラムコード命令を含む。前記コンピュータプログラムは、前記ＸＭＬ文書内の前記データ構造の前記フォーマットに対応するデータフレームワークを生成するコンピュータプログラムコード命令、および前記初期ＸＭＬ文書から前記データアイテムを分析し、前記データアイテムに基づいてデータ制約を決定するコンピュータプログラムコード命令を含む。前記コンピュータプログラムはまた、前記生成されたデータフレームワークおよび前記生ＸＭＬデータから決定された前記データ制約に基づいてＸＭＬスキーマを生成するコンピュータプログラムコード命令を含む。

【００１３】

本発明の原理の他の実施形態はコンピュータシステムを実現する。このコンピュータシステムは、少なくとも１つの中央処理ユニット（ＣＰＵ）、メモリ、およびユーザインタフェースが組み合わせられて、ＸＭＬデータ要素として構成された生ＸＭＬデータを含む受け取られた初期ＸＭＬ文書を分析するＸＭＬ構造アナライザであって、前記分析は、前記初期ＸＭＬ文書の前記ＸＭＬデータ要素を特定すること、および前記初期ＸＭＬ文書の前記ＸＭＬデータ要素についてのデータ構造に関連付けられたデータフレームワークを生成することを含む。加えて、このシステムは、前記初期ＸＭＬ文書からの前記生ＸＭＬデータを分析し、前記ＸＭＬ生データに基づいてデータアイテムについてのＸＭＬデータ制約を決定するデフォールト制約ジェネレータを含む。システムは、前記データ制約ジェネレータからの前記ＸＭＬデータ制約および外部から供給されるデータ制約のセットのうちの少なくとも１つを受け取り、最終データ制約ファイルを出力するデータ制約マージャを

含む、および前記ＸＭＬ構造アナライザからの前記データ構造および前記データ制約マージャからの最終データ制約ファイルを受け取り、前記データ構造および最終データ制約ファイルに関連付けられたＸＭＬスキーマを生成するＸＭＬスキーマジェネレータを含む。

【００１４】

他の実施形態において、コンピュータモジュールが開示される。このモジュールは、ＸＭＬデータ要素として構成された生ＸＭＬデータを含む受け取られた初期ＸＭＬ文書を分析するＸＭＬ構造アナライザであって、前記分析は、前記初期ＸＭＬ文書の前記ＸＭＬデータ要素を特定すること、および前記初期ＸＭＬ文書の前記ＸＭＬデータ要素についてのデータ構造に関連付けられたデータフレームワークを生成することを含む。このモジュールはさらに、前記初期ＸＭＬ文書からの前記生ＸＭＬデータを分析し、前記ＸＭＬ生データに基づいてデータアイテムについてのＸＭＬデータ制約を決定するデフォールト制約ジェネレータを含む。前記データ制約ジェネレータからの前記ＸＭＬデータ制約および外部から供給されるデータ制約のセットのうちの少なくとも１つを受け取り、最終データ制約ファイルを出力するデータ制約マージャが含まれる。また、前記ＸＭＬ構造アナライザからの前記データ構造および前記データ制約マージャからの最終データ制約ファイルを受け取り、前記データ構造および最終データ制約ファイルに関連付けられたＸＭＬスキーマを生成するＸＭＬスキーマジェネレータが含まれる。

【００１５】

他の実施形態において、他のモジュールが開示される。このモジュールは、ＸＭＬデータ要素として構成された生ＸＭＬデータを含む受け取られた初期ＸＭＬ文書を分析するＸＭＬ構造アナライザであって、前記分析は、前記初期ＸＭＬ文書の前記ＸＭＬデータ要素を特定すること、および前記初期ＸＭＬ文書の前記ＸＭＬデータ要素についてのデータ構造に関連付けられたデータフレームワークを生成することを含む。このモジュールはさらに、前記初期ＸＭＬ文書からの前記生ＸＭＬデータを分析し、前記ＸＭＬ生データに基づいてデータアイテムについてのＸＭＬデータ制約を決定するデフォールト制約ジェネレータを含む。前記初期ＸＭＬ文書からの前記生ＸＭＬデータを分析し、前記ＸＭＬ生データに基づいてデータアイテムについてのＸＭＬデータ制約を決定するデフォールト制約ジェネレータが含まれる。前記ＸＭＬ構造アナライザからの前記データ構造および前記データ制約マージャからの最終データ制約ファイルを受け取り、前記データ構造および最終データ制約ファイルに関連付けられたＸＭＬスキーマを生成するＸＭＬスキーマジェネレータが含まれる。

【００１６】

本発明の他の局面および優位性は、本発明の原理を例示的に示す添付の図面と併せて以下の詳細な説明から明らかになろう。

【００１７】

本発明は、添付の図面と併せれば以下の詳細な記載によって容易に理解されえ、ここで同様の参照番号は同様の構造要素を表す。

【発明を実施するための最良の形態】

【００１８】

本発明は、ＸＭＬ分析を検証するのに用いられえるＸＭＬスキーマを自動的に生成する技術、システム、およびモジュールに関する。本発明の実施形態は、以下に図１～１０Ｂを参照して記載される。しかし当業者なら、これら図を参照してここに与えられた詳細な記載は例示目的であって、本発明はこれらの限定的な実施形態を超えて広がることが容易に理解するだろう。

【００１９】

本発明のある局面によると、ＸＭＬスキーマを実現する典型的なアプリケーションが記載される。図１は、本発明の原理によって用いられるアプリケーションプログラム１００を示す簡略化されたブロック図である。プログラム１００は、プログラム１００に供給されるＸＭＬ文書１０１を操作する例示的プロセス１０２を含む。図示された実施形態において、バリデータ１０３（例えば検証プログラム）が用いられてプログラム１００に提供

されたＸＭＬ文書１０１を検証する。バリデータ１０３は、ＸＭＬ文書１０１のデータ要素のデータ構造およびフォーマットをＸＭＬスキーマ１０４（これもバリデータに提供される）と比較することによって、ＸＭＬ文書１０１のコンテンツの構造およびフォーマットがプロセス１０２で動作するのに「妥当である」（すなわち正しく構成され、構造化されている）かを決定する。一般にこれは、ＸＭＬ文書１０１のコンテンツの構造およびフォーマットがＸＭＬスキーマ１０４のそれと一致することを意味する。バリデータ１０３は、妥当なＸＭＬ文書１０５を出力する。示されたバリデータ１０３はプログラム１００の一部を形成するが、これは必ずしもそうでなくてもよいことを本発明の発明者は指摘しておく。バリデータ１０３はプロセス１０２から分離されえる。バリデータ１０３は単に検証されたＸＭＬファイル１０５をプロセス１０２に提供する。

【００２０】

図２は、本発明の原理による自動的にスキーマを生成するシステム２００のある実現例を示す。示される本発明の実施形態において、システムは、ソフトウェアモジュール２０２として実現されえる。当業者ならわかるように、システムは、多くの他の構成でも実現されえる。例えばシステムは、オペレーティングシステムの一部を形成しえ、アプリケーションプログラムの一部として実現されえ、加えて、システムはハードウェアとしても実現されえる。本発明のある実施形態によれば、システムは、ＸＭＬスキーマを生成するために採用されえるソフトウェアモジュール２１０として実現される。上で暗示されるように、ある実施形態において、モジュールはアプリケーションの一部として、または他のプログラムの一部として採用されえる。加えてここで示されるように、ある実施形態においてはモジュール２１０は、コンピュータシステムの一部を形成する。モジュール２１０はＸＭＬファイル２０１を受け取り、ファイル２０１を用いて、他のＸＭＬファイルを検証するのに用いられえる関連付けられたＸＭＬスキーマ２２０を生成するよう構成される。このような検証は例えば、さまざまなアプリケーションプログラムに入力されるよう意図された他のＸＭＬファイルを検証するのに用いられえる。

【００２１】

ＸＭＬデータ構造および関連付けられたＸＭＬデータアイテム（ここでは生ＸＭＬデータとも呼ばれる）を含むＸＭＬ文書２０１は、モジュール２１０に与えられる。モジュール２１０は、ＸＭＬ文書を受け取り、文書をＸＭＬ文書２０１のためのデータ構造を定義するデータフレームワークに変換できるＸＭＬ構造アナライザ２１２を含む。一般にデータフレームワークは、ＸＭＬ文書２０１を含む空要素の配置および構造を含む。このフレームワーク情報は、ＸＭＬスキーマジェネレータ２１８への出力２１２ｏである。加えて、モジュールは、デフォールト制約ジェネレータ２１４を含みえる。デフォールト制約ジェネレータ２１４はＸＭＬ文書を受け取り、ＸＭＬ文書２０１のデータアイテムを識別し、かつこれらデータアイテムのプロパティのセットを定義することができる。例えば、データアイテムについて識別および定義されえるプロパティのあるセットは、それぞれのデータアイテムについてのデータタイプ（例えばストリング、数値（例えば整数、小数など）、２進数、ブール、日付、時間、任意のＵＲＩ、倍精度、浮動、ノーテーション、Ｑｎａｍｅ、併せて多くの他の「タイプ」）である。これらプロパティは、それぞれのデータアイテムについての特定のプロパティを識別するのに用いられえ、それぞれのデータアイテムについての制約を生成しえる。この制約条件は、デフォールト制約ジェネレータ２１４からデフォールト制約ファイル２１４ｏ（例えばＸＭＬファイル）として出力される。デフォールト制約ファイル２１４ｏは、制約マージャ２１６によって受け取られえ、これは選択された制約情報をＸＭＬスキーマジェネレータ２１８に与える。スキーマジェネレータ２１８は、選択された制約情報をデータ構造と共に用いてＸＭＬ文書２０１を、ＸＭＬ文書２０１に関連付けられたＸＭＬスキーマに翻訳する。加えて、ユーザは追加の制約情報を外部で生成された制約ファイル２１９のかたちで供給しえ、これも制約マージャ２１６に入力される。制約マージャ２１６は、デフォールト制約ファイル２１４ｏまたは外部生成制約ファイル２１９をＸＭＬスキーマジェネレータ２１８への入力として選択することを選びうる。代替として、制約マージャ２１６は、ＸＭＬスキーマジェネレータ２１

８への入力を生成するために、デフォールト制約ファイル２１４ｏを外部生成された制約ファイル２１９にマージしえる。

【００２２】

代替として、モジュール２１０は、デフォールト制約ファイル２１４ｏがスキーマジェネレータ２１８によってマージャ２１６を用いることなく、外部生成された制約２１９なしで直接に受け取られるように（２１４ｏ’を参照）構成されえる。このような場合、外部生成された制約２１９は用いられず、マージャ２１６は必要ない。

【００２３】

図３は、本発明のある典型的な方法実施形態を記載するフロー図である。プロセスフロー３００は、初期ＸＭＬ文書を特定のすることによって始まりえる（ステップ３０１）。この文書は典型的には、所望のアプリケーションによって処理するのに適切な正しくフォーマットされた文書である。例えば、ＸＭＬ文書は、ＸＭＬスキーマを生成するようここで実現されたソフトウェアモジュールまたは他のシステムに提供されえる。初期ＸＭＬ文書は、ＸＭＬフォーマットに準拠するよう構成された適切にフォーマットされたＸＭＬ文書である。初期ＸＭＬ文書は典型的には、関連付けられたＸＭＬスキーマを生成するのに用いられるＸＭＬデータ構造および生ＸＭＬデータ（ＸＭＬデータアイテム）を含む。初期ＸＭＬ文書を提供することは一般に、初期ＸＭＬ文書をＸＭＬ構造アナライザ（例えば２１２）およびＸＭＬデフォールト制約ジェネレータ（例えば２１４）の両方に提供することを含む。初期ＸＭＬ文書は、文書のＸＭＬ構造を識別するために分析される（ステップ３０３）。これはＸＭＬ構造アナライザ（例えば２１２）を用いて達成されえる。ＸＭＬ要素が識別され、要素間の関係が認識される。初期ＸＭＬ文書のデータ要素に関連付けられたデータ構造フレームワークがそれから初期ＸＭＬ文書の分析から得られた情報を用いて生成される（ステップ３０５）。このフレームワークは典型的には初期ＸＭＬ文書のそれと類似の配置で構成された空ＸＭＬデータ要素のパターンを定義する。出力データ構造フレームワーク３０７は後のプロセスで用いられるようにＸＭＬ文書（XML.doc）として出力されえる。

【００２４】

初期ＸＭＬ文書はまた、その文書のＸＭＬデータ要素（生ＸＭＬデータ）を識別するためにも分析される（ステップ３１１）。これはステップ３０３、３０５と同時に、または異なる時になされえる。典型的にはこの文書は、ＸＭＬデフォールト制約ジェネレータ（例えば２１４）を用いて達成されえる。ＸＭＬデータアイテムが識別され、データアイテムに関するデータプロパティが確定される。例えば、データアイテムについてのデータタイプが識別され、データ属性も識別される。デフォールトデータ制約のセットがそれぞれのデータアイテムについてそれから生成される（ステップ３１３）。デフォールトデータ制約は初期ＸＭＬデータ中のデータアイテムのデータプロパティに基づいて生成される。ふつうは、デフォールトデータｓ要求は、タイプおよび属性情報と共に他のデータプロパティ制約を含む。デフォールトデータ制約３１５の出力セットはＸＭＬ文書としての出力でもありえる。例示的なデータ制約は、これに限定されないが、ターゲット情報を示す属性（すなわち制約が適用されなければならない要素または属性）を含む。例えば、ターゲットのタイプが識別されえる。ある例では、ターゲットはタイプ要素(type="elem")として、またはタイプ属性(type="attr")として識別されえる。ターゲットの名前が識別されえる（例えばname="orderDate"）。ルート（/）からのターゲットのパスを特定するパス名が識別されえる（例えばpath="/purchase0rder"）。ある限定的な例示的実現例においては、属性および要素は、以下の例示的フォーマットによって制約されえる。ある例では、属性情報は以下のように制約されえる。すなわち、<constraint type="attr" name="orderDate" path="/purchase0rder">である。他の例では要素情報は以下のように制約されえる。すなわち、<constraint type="elem" name="quantity" path="/purchase0rder/items/item/quantity">である。当業者には理解されえるように、多くの他の制約フォーマットが利用されえ、示されたものは単に例示的であり限定するものではない。他の例では、データタイプ要素は、デフォールトデータタイプ情報をタイプ属性内に保持しえ、ある例

では以下のようにフォーマットされえる。例えば<datatype type="xsd:string"> </datatype>である。

【００２５】

当業者には理解されえるように、多くの他の制約フォーマットが利用されえ、上で示されたものは、限定するものというよりも、単に例示的であることを意図されている。

【００２６】

ある実施形態において、出力データ構造フレームワーク３０７およびデフォールトデータ制約３１５の出力セットは、初期ＸＭＬ文書の導入を超えたさらなるユーザ入力なしにスキーマを自動的に生成するのに用いられる（ステップ３２５）。ある実施形態において、出力データ構造フレームワーク３０７およびデータ制約３１５の出力デフォールトセット（破線３１６）は、スキーマジェネレータ（例えば２１８）によって受け取られ、ＸＭＬファイルを検証できるＸＭＬスキーマ３３０を自動的に生成するよう処理される。

【００２７】

他の実施形態において、さらなる制約の外部生成されたセットが与えられえる（ステップ３１７）。外部生成された制約ファイルは、ユーザによって与えられうる（または所定の条件のセットによって機械によって生成される）。例えば郵便番号制約は、type=integerに限定されえるが、また５つのエントリ（すなわち「９０５０５」）を有するデータアイテムしか検証しないように限定され、または代替としてはＸＸＸＸＸ－ＸＸＸＸフォーマットの９つのエントリ（すなわち「９０５０５－１４０５」）を有するデータアイテムを検証するように限定されえる。外部生成された制約のセット３１７およびデフォールト制約３１５は共に、制約の選択されたセットを得るために処理されえる（ステップ３２０）。ある例では、制約３１５、３１７の２つの（またはそれより多い）セットが制約マージャ（例えば２１６）に入力されえ、このマージャが選択された出力制約３２１を与えるために制約群のうちのいずれかを選択しえ、選択された出力制約は、ＸＭＬファイルを検証できるＸＭＬスキーマ３３０を自動的に生成するために出力データ構造フレームワーク３０７と用いられる（ステップ３２５）。簡単な例では例えば、ステップ３２０で動作するマージャは、選択された出力制約３２１としてユーザによって外部で生成される（例えば３１７）制約のセット３１５、３１７を単に選択する。

【００２８】

上述のアプローチに加えて、他の実施形態は、外部で生成された制約のセット３１７およびデフォールト制約３１５を共に処理して、他の選択された制約のセット３２１を得る（ステップ３２０）。この場合、制約３１５、３１７は、制約マージャ（例えば２１６）に入力されえ、このマージャが制約３１５、３１７をマージして、両方のファイルからのデータを含むマージされたファイル（典型的にはＸＭＬフォーマット）の形で選択された出力制約３２１を与える。このマージされたファイルは、出力データ構造フレームワーク３０７と共に用いられて、ＸＭＬファイルを検証できるＸＭＬスキーマ３３０を自動的に生成する（ステップ３２５）。簡単なケースでは例えば、外部生成された制約情報は、デフォールト制約文書における任意のギャップを埋めるために用いられえる。加えて、デフォールト制約ファイルにおける制約情報が、外部で生成された制約ファイルに含まれる制約情報とコンフリクトする状況では、コンフリクトは所定のコンフリクト解決スキームに従って解決されえる。例えば優先順位によるスキームが実現されえる。または外部で生成された制約ファイル内に含まれる制約情報がデフォールト情報に優先して選択されえる。

【００２９】

以下の説明は、構造アナライザ（例えば２１２）および関連付けられた動作モードのある例示的実施形態を記載する。一般に構造アナライザは入力ＸＭＬ文書を読み、ＸＭＬデータ要素の構造を分析し、フレームワークの形の構造情報を含む、結果として生じるＸＭＬ文書を生成する。このフレームワークは一般に、ＸＭＬデータ要素、属性情報、およびデータ要素およびデータ属性のカーディナリティに関する選択されたメタデータを備えると考えられる。

【００３０】

図４は、図３の動作３０３、３０５を達成するよう動作する構造アナライザ（例えば２１２）のある実施形態のための例示的動作モードを示す簡略化フロー図４００である。

【００３１】

初期ＸＭＬファイルが読まれ（ステップ４０１）、「ルート要素」がその構造について生成される（ステップ４０３）。ある実施形態においてはシステムパーザがルート要素を生成するのに用いられえる。入力ＸＭＬファイルの「子」要素が識別される（ステップ４０５）。「兄弟」データ要素は他の関連付けられた兄弟と同じデータフォーマットを有するので（データの正確なコンテンツは変わりえるがデータフォーマットは同じである）、任意の兄弟についてのデータフォーマットを特定するために必要なデータプロパティを識別するために最初の兄弟だけが分析されなければならない。検証スキーマの目的は、入力ＸＭＬ文書が正しいフォーマットを有するかを確認することなので、最初の兄弟のそのような分析で充分である。それぞれの最初の兄弟は、その名前、属性、属性の個数（属性カウント）、値、および要素カウントを決定するよう処理されえる（ステップ４０７）。ふつうは冗長な子は分析される必要はないが、これはそれらが他の関連する兄弟と同じデータフォーマットを特定するからである。結果として生じる出力は、初期ＸＭＬファイルの構成フォーマットをキャプチャするデータ構造である（ステップ４０９）。

【００３２】

ある例示的な例では表１が入力初期ＸＭＬファイルを提供する。

【表１】

```
<?XML version="1.0" encoding="UTF-8"?>
  <purchaseOrder orderDate="1999-10-20">
    <items>
      <item partNum="242-NO" >
        <productName>Nosferatu - Special Edition (1929)</productName>
        <quantity>5</quantity>
        <USPrice>19.99</USPrice>
      </item>
      <item partNum="243-NO" >
        <productName>The Mummy (1959)</productName>
        <quantity>3</quantity>
        <USPrice>19.98</USPrice>
      </item>
    </items>
  </purchaseOrder>
```

【００３３】

表１の初期ＸＭＬファイルは、初期ＸＭＬファイルに関連付けられたデータ構造についての以下の結果として生じるフレームワーク（表２を参照）を生成するのに用いられえる。

【表２】

```
<purchaseOrder orderDate="1999-10-20" XSG_attrcount_orderDate="1"   XSG_count="1">
  <items XSG_count="1">
    <item partNum="242-NO" XSG_attrcount_partNum="2" XSG_count="2">
      <productName XSG_val="Nosferatu - Special Edition (1929)"
XSG_count="1"/>
      <quantity XSG_val="5" XSG_count="1"/>
      <USPrice XSG_val="19.99" XSG_count="1"/>
    </item>
  </items>
</purchaseOrder>
```

【００３４】

ルート要素は、例えばpurchaseOrderとして生成される。上のフレームワークの構造は、その全体で要素"purchaseOrder"を記述するために<purchaseOrder orderDate="1999-10-20" XSG_attrcount_orderDate="1" XSG_count="1">を含む。

【００３５】

例えば、orderDate="1999-10-20"は、orderDateと呼ばれる属性を記述する。またXSG_attrcount_orderDateは、ＸＭＬ文書中のタイプpurchaseOrderの全ての要素についての属性orderDateのカーディナリティ（個数）を記述する。XSG_count="1"は、ＸＭＬ文書中のタイプpurchaseOrderの要素のカーディナリティを記述する。アイテムpartNumberについての最初のデータ要素（例えば"242-NO"）は、保持され定義されている。これらの同じ定義およびフレームワークパラメータは、アイテムpartNumber（例えば"243-NO"）についての全ての他の兄弟データ要素に適用し、このアイテムはフレームワーク要素として既に定義されている。したがって"243-NO"に関するアイテムpartNumberは分析されなくてもよい。同じ種類の要素定義が、他のデータ要素、例えば、productName、quantity、およびUSPriceについても行われる。

【００３６】

図５は、例えば図４に示されるプロセスを実行するのに用いられえるＸＳＬＴスタイルシート、または図３の操作３０３、３０５を実行する構造アナライザ（例えば２１２）の動作のある実施形態を記載する。

【００３７】

図５のスタイルシート実施形態は、初期ＸＭＬ文書の構造フレームワークを生成する、あるアプローチを表す。当業者にはよくわかるように、多くの他のアプローチが用いられえる。図５の実施形態は、入力ＸＭＬ文書のルート（/）要素を特定し、その要素を「現在の要素」として指定することによって始まる。「現在の要素」中の「子」要素の全てが処理され、最初の兄弟が特定される。それから最初の兄弟が処理され、少なくともこの実施形態では以下の情報を含むＸＭＬ文書を生成する。

【００３８】

すなわち、

Name、ここでは最初の兄弟の要素名を含む。（purchaseOrder）；

Attributes（例えば、orderDate="1999-10-20"）；

Attribute count（すなわち、それぞれの属性についてのカーディナリティ情報）。例えばここで示されるものは、
XSG_attrcount_orderDate="1"
を含みえる。

また、value （すなわち、もしあるならテキストノード要素の値）。例えばアイテムproductNameについてXSG_val="Nosferatu - Special Edition (1929)"のような値である。

また、element count（それぞれの要素についてのカーディナリティ情報）が識別されえ、ここではXSG_count="1"である。このプロセスは、初期ＸＭＬファイルの全ての要素（第１子）が訪ねられるまで全ての第１の子について反復されえる。このようなプロセスは、本発明のある実施形態によって構造情報を生成するのに用いられる一つの例である。

【００３９】

前に示唆されたように本発明の構造は、デフォールト制約ジェネレータ（default constraint generator）を含む。図６は、図３の操作３１１、３１３を達成するよう動作するデフォールト制約ジェネレータ（例えば２１４）のある実施形態についての動作の例示的モードを示す簡略化されたフロー図６００を記載する。

【００４０】

初期ＸＭＬファイルが読まれ（ステップ６０１）、「ルート要素」がその構造について生成される（ステップ６０３）。ある実施形態において、ルート要素を生成するためにはシステムパーザが用いられえる。入力ＸＭＬファイルの「子」要素が特定される（ステップ６０５）。要素は、初期ＸＭＬファイルのそれぞれのデータアイテムに関連する制約情報を含むように生成される（ステップ６０７）。ある実施形態において、これら要素は<namespaces>要素と呼ばれる。それからそれぞれのデータアイテムは<namespace>データ要素と関連付けられる。加えて、それぞれの<namespace>要素は、それぞれのデータアイテムに関する制約情報を含む<constraint>要素を含むように構成される（ステップ６０９）。初期ＸＭＬファイルのこのデータアイテムは、処理されて関連付けられた制約情報を生成する（ステップ６１１）。それぞれのデータアイテムは分析されて、そのアイテムを名前によって特定する"name"を決定し、分析されてそのデータアイテムの属性を特定する"attributes"を決定し、さらに分析されてその属性または要素が属するＸＭＬ構造内の要素のパスについてのパス名を特定する構造のパス名を特定する"path"を決定する。また制約要素のために他の要素が作られえる（ステップ６１３）。この要素は生成された要素についての"type"情報を含みえる。ある実施形態において、このような要素は例えば"datatype"要素と呼ばれえる。このような場合、"type"要素は、生成され（ステップ６１５）、その値がデフォールト制約ジェネレータに入力された初期ＸＭＬファイルから生成されたデータタイプを表す単一の属性を備える。このプロセスは、初期ＸＭＬファイル内のそれぞれのデータアイテムについて反復されて（ステップ６１７）、初期ＸＭＬファイル内のデータアイテムの完全な特徴付け（characterization）を提供し、かつデフォールト制約ＸＭＬ文書を生成する。

【００４１】

一般に、上のプロセスは、デフォールト制約ＸＭＬファイルにおいて用いられる情報の３つのブロックを生成する。第１に"namespaces"要素が入力文書から生成されえる。この要素は、ＸＭＬスキーマの生成のあいだにネームスペース情報を変化させるために変更されえる（例えば２１８）。

【００４２】

例えば、"namespaces"要素は以下のように構成される。

【数３】

```
<namespaces>
  <namespace value="http://java.sun.com/XML/ns/jaxb"
name="XMLns:jaxb">
  </namespace>
</namespaces>
```

【００４３】

追加のネームスペースは、新しいネームスペースを"namespaces"要素の内部に挿入することによって容易に追加されえる。

【００４４】

またデータタイプは、本発明の原理に従って容易に変更されえる。例えば、以下の制約要素は初期では以下のように構成される。

【数４】

```
<constraint type="attr" name="orderDate" path="/purchaseOrder">
    <datatype type="xsd:date">
    </datatype>
  </constraint>
```

【００４５】

これは生成されたデータタイプを"1999-10-20"コンテンツに基づいてorderDateについてxsd:dateと記述する。

【００４６】

もしユーザがＸＭＬスキーマ生成のあいだにこの属性（datatype）についてのタイプをxsd:timeであるように変化させることを望むなら、それは例えば以下のように変更されえる。

【数５】

```
<constraint type="attr" name="orderDate" path="/purchaseOrder">
    <datatype type="xsd:time">
    </datatype>
</constraint>
```

【００４７】

この変更された制約は、変更されたタイプ情報に従って、生成するＸＭＬスキーマ文書について今度は用いられえる。

【００４８】

さらに、本発明の原理に従って追加の要素（コメントなど）が容易に追加されえる。ある例示的実現例においては、手でスキーマをリライトする必要なしに、自動的に新しい要素をスキーマに導入するために要素挿入ルーチンが用いられえる。

【数６】

```
<insert path="/" name="comment_def">
    <xsd_element type="xsd:string" name="comment">
    </xsd_element>
 </insert>
```

【００４９】

例えば、下の出力ＸＭＬファイルは、上の表１で示される初期ＸＭＬファイルを用いて生成された出力デフォールト制約の一例である。

【数７】

```
<?XML version="1.0" encoding="UTF-8"?>
<root>
        <namespaces>
              <namespace value="http://java.sun.com/XML/ns/jaxb"
name="XMLns:jaxb">
        </namespace>
      </namespaces>
<constraints>
    <constraint type="attr" name="orderDate" path="/purchaseOrder">
        <datatype type="xsd:date">
        </datatype>
    </constraint>
    <constraint type="elem" name="productName"
path="/purchaseOrder/items/item/productName">
                <datatype type="xsd:string">
                </datatype>
            </constraint>
            <constraint type="elem" name="quantity"
path="/purchaseOrder/items/item/quantity">
                <datatype type="xsd:integer">
                </datatype>
          </constraint>
          <constraint type="elem" name="USPrice"
path="/purchaseOrder/items/item/USPrice">
                    <datatype type="xsd:decimal">
                    </datatype>
            </constraint>
        </constraints>
        <inserts>
            <insert path="/" name="comment_def">
                  <xsd_element type="xsd:string" name="comment">
                  </xsd_element>
              </insert>
        </inserts>
</root>
```

【００５０】

図７は、例えば図４に示されるプロセスを実行するのに用いられえるＸＳＬＴスタイルシート、または図３の操作３１１、３１３を実行するデフォールト制約ジェネレータ（例えば２１４）の動作のある実施形態を記載する。

【００５１】

以下の説明は、マージャ（例えば２１６）および関連付けられた動作のモードを記載する。一般にマージャは、デフォールト制約情報（例えば２１４ｏ）の入力セットおよび外部から供給された制約（例えば２１９）の入力セットを読み、制約のマージされたセットの一つを生成する。代替として、マージャは制約のデフォールトセットまたは制約の外部から供給されたセットをスキーマジェネレータ（例えば２１８）への出力のために選択しえる。ある実現例において、外部から供給された制約は、副情報を提供するデフォールト制約情報と共に、主制約情報を提供する。

【００５２】

図８は、図３の操作３２５を達成するために動作するＸＭＬスキーマジェネレータ（例えば２１８）のある実施形態についての動作モードの一例を示す簡略化されたフロー図を記載する。

【００５３】

以下は、例えばスキーマジェネレータ（例えば２１８）を用いた本発明の原理によるスキーマ生成の簡単な簡略化された記載である。生成されたデータフレームワーク２１２ｏ（例えばＸＭＬファイルの形である）が構造アナライザ２１２（ステップ８０１）から受け取られる。同時に、または異なる時に、最終データ制約ファイル（例えばＸＭＬファイルの形である）がマージャ２１６から受け取られる（ステップ８０３）。２つのファイルが読まれ（ステップ８０５）、「ルート要素」がスキーマについて生成される（ｘｓｄファイル）（ステップ８０７）。要素および関連付けられた子要素のそれぞれが特定される（ステップ８０９）。それぞれの要素（および子要素）は、関連付けられた要素のセットを含む（ステップ８１１）ことによって関連するｘｓｄ要素を定義するＸＭＬ要素（または要素セット）を生成するために処理される。もし子要素がさらなる子（兄弟ではない）を含むなら、そのような要素は「複合型要素」であり、そのように処理される。例えばｘｓｄ要素は、子を有する要素をタイプ複合型要素（例えば"xsd:complexType"）として特定するよう生成される。完全に構築されたｘｓｄ（スキーマ）ファイルが生成される（ステップ８１３）。

【００５４】

このような処理（ステップ８１１）は典型的には、それぞれの要素（および子要素）をスキャンすることによってＸＭＬスキーマ要素を定義および生成することを伴う。

【００５５】

例えば複合型要素を生成する典型的な実現例において、付随する（accompanying）"complexType"セット（タイプ宣言およびcomplexType定義を含む）が生成される。例示的な例において、要素の複合型セットは宣言を含みえる。すなわち、

【数８】

```
<xsd:element name="purchaseOrder" type="_purchaseOrder"/>
and can include a complexType definition:
<xsd:complexType name="_purchaseOrder">
<xsd:sequence>
<xsd:element name="items" type="_items"/>
</xsd:sequence>
<xsd:attribute name="orderDate" type="xsd:date" use="required"/>
</xsd:complexType>
```

【００５６】

単純な要素において、対応する単純型ＸＭＬスキーマが生成されえる。以下の例において（子カウント＝０を有する）、宣言が定義される。例えば、<xsd:element name="produ

ctName" type="XYZ"/>でXYZはタイプでありえる（例えばストリング、整数、小数、ブール、データ、時間など）。ここで"XYZ"は、外部から提供される制約文書から、または生成されたデフォールト制約ＸＭＬ文書から選択される。プレファレンスが、外部から供給された制約ＸＭＬ文書タイプにもしそれらが供給されているなら与えられえる。加えて、もし要素が属性を含むなら、それが例えば以下のように構成されえる。

【数９】

```
<xs:element name="item">
  <xs:complexType>
    <xs:attribute name="partNum" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

【００５７】

上で挙げられた例は、本発明のある原理を示すために用いられる簡略化された例として意図されることに注意されたい。よってこれらの例は、本発明の範囲または実現例を限定するようには意図されていない。当業者にはわかるように、多くの関連する実現例の詳細が本発明の実施形態を達成するために実現されえる。

【００５８】

このようなスキーマ生成を達成する完全に動作可能な実施形態の一例は図９．１～９．５のスタイルシートによって示される。

【００５９】

ここで記載された全てのプロセス、方法、およびシステムは多くの異なる種類のツールで実現されえることに注意されたい。例としてこれらに限定されないがコマンドラインツールおよびサーブレットがある。また本発明の原理はＡＮＴアセンブリ言語プログラミング（例えばＡｎｔ３．０．１）および他のビルドツールに適用されえる。また発明者は本発明の原理がJAVA Studio IDE NBMモジュールに応用されることも想定している。ここで記載されたプロセスは、コンピュータで読み取り可能な媒体の実現例におけるインプリメンテーションにもよく適する。また本発明の原理は、ネットワークベースのコンピュータシステムにもよく適する。例えば、インターネット上でリモートコンピュータに与えられたあるコンピュータ上のＸＭＬ文書は、リモート使用のために、または送ってくるコンピュータへ戻してそこで使用するために、そのリモートコンピュータ上でＸＭＬスキーマに変換されえる。

【００６０】

図１０Ａおよび１０Ｂは、本発明によって用いられえるマルチプロセッサコンピュータシステム１０００の例を示す。図１０Ａは、ディスプレイ１００２、スクリーン１００４、筐体１００６、キーボード１０１０、およびマウス１０１２を含むコンピュータシステム１０００を示す。マウス１０１２は、グラフィカルユーザインタフェースとインタフェースするための１つ以上のボタンを有しえる。筐体１００６は、本発明を実現するコンピュータコードを含むソフトウェアプログラム、本発明と共に用いるデータなどを記憶および取り出しするために利用されえるＣＤ－ＲＯＭドライブ１００８、システムメモリおよびハードドライブ（図１０Ｂを参照）を収納しえる。ＣＤ－ＲＯＭドライブ１００８は、例示的なコンピュータで読み取り可能な記憶媒体として示されるが、フレキシブルディスク、ＤＶＤ、テープ、メモリスティック、フラッシュメモリ、システムメモリ、およびハードドライブを含む他のコンピュータで読み取り可能な記憶媒体も利用されえる。加えて、搬送波内で実現されるデータ信号（例えばインターネットを含むネットワーク内での）もコンピュータで読み取り可能な記憶媒体でありえる。ある実現例では、コンピュータシステム１０００のためのオペレーティングシステムがシステムメモリ、ハードドライブ、

ＣＤ－ＲＯＭ１００８または他のコンピュータで読み取り可能な記憶媒体内で提供され、本発明を実現するコンピュータコードを含むために機能する（例えばＭＭＵシステム）。オペレーティングシステムは、システムのプロセッサの全てを制御するよう構成されることに注意されたい。他のデバイス（例えばプリンタ、スキャナなど）もコンピュータシステム１０００内に存在しえることに注意されたい。

【００６１】

図１０Ｂは、本発明の実施形態のソフトウェアを実行するのに用いられるコンピュータシステム１０００のシステムブロック図を示す。コンピュータシステム１０００は、モニタ１００４、キーボード１０１０、およびマウス１０１２を含む。コンピュータシステム１０００は、複数の中央プロセッサ（ＣＰＵ群）１０２２（キャッシュメモリリソースを含む）、システムメモリ１０２４、固定記憶１０２６（例えばハードドライブ）、取り外し可能な記憶１０１４（例えばＣＤ－ＲＯＭドライブ）、ディスプレイアダプタ、サウンドカード、およびスピーカ１０３０、およびネットワークインタフェース１０４０のようなサブシステムをさらに含む。中央プロセッサ１０５１は例えば、本発明を実現するためにコンピュータプログラムコード（例えばオペレーティングシステム）を実行しえる。オペレーティングシステムは、その実行中にふつう（必ずしもそうではないが）システムメモリ１０２４中に常駐する。本発明と共に用いるのに適切な他のコンピュータシステムは、より多くの、またはより少ないサブシステムを含みえる。重要なことは、本発明の原理は、多くの個別のコンピュータ群を有するネットワークで結ばれたコンピュータシステム上で具体的に実現されえる。そのようなネットワークで結ばれたシステムは、ローカルエリアネットワーク（ＬＡＮ）またはワイドエリアネットワーク（ＷＡＮ）を含みえる。特に、発明者は、インターネットを用いて互いにネットワークで結ばれたコンピュータシステムを想定する。加えて、ＬＡＮの例は、総合ビルを持つ中規模の大きさの会社によって用いられるプライベートネットワークである。公でアクセス可能なＷＡＮは、インターネット、携帯電話ネットワーク、衛星システムおよび単純旧式電話サービス（ＰＯＴＳ）を含む。プライベートＷＡＮの例は、多国籍企業によってそれらの内部情報システムの要求のために用いられるものを含む。ネットワークはまた、プライベートおよび／またはパブリックＬＡＮおよび／またはＷＡＮの組み合わせでありえる。

【００６２】

コンピュータシステム１０００のシステムバス構成は、矢印１０２０によって表現される。しかしこれら矢印は、サブシステムを結合するよう機能する任意の相互接続スキームを例示的に表すものである。例えば、中央プロセッサをシステムメモリおよびディスプレイアダプタに接続するためにローカルバスが用いられえる。図１０Ｂに示されるコンピュータシステム１０００は、本発明と共に用いられるのに適切なコンピュータシステムの一例に過ぎない。サブシステムの異なる構成を有する他のコンピュータアーキテクチャも利用されえる。

【００６３】

本発明は、ハードウェアおよびソフトウェア要素の組み合わせを用いえる。ソフトウェアは、コンピュータによって読み取り可能なコード（またはコンピュータプログラムコード）としてコンピュータで読み取り可能な媒体上に実現されえる。コンピュータで読み取り可能な媒体は、コンピュータシステムによってその後、読み出されえるデータを記憶しえる任意のデータ記憶デバイスである。コンピュータで読み取り可能な媒体の例には、読み出し専用メモリ、ランダムアクセスメモリ、ＣＤ－ＲＯＭ、磁気テープ、および光データ記憶デバイスが含まれる。コンピュータで読み取り可能な媒体はまた、ネットワーク上で結合されたコンピュータシステム上で分散されえ、それによりコンピュータによって読み取り可能なコードが分散化された形で記憶および実行されえる。

【００６４】

本発明の利点は多い。異なる実施形態または実現例は、１つ以上の以下の効果を有しえる。本発明の一つの利点は、それが、これには限定されないがウェブベースのネットワークを含むネットワークで結ばれたコンピューティング環境で用いられえることである。Ｘ

ＭＬ文書は遠隔でアクセスされてスキーマを生成しえる。加えて、本発明のシステムの要素は、もし望まれるなら分散化された形で動作されえる。例えば、ＸＭＬ構造アナライザは、あるコンピュータ上に位置しえ、デフォールト制約ジェネレータは他のネットワークで結ばれたコンピュータ上に位置しえる。他の実現例においては、ＸＭＬ文書はあるネットワークで結ばれたコンピュータから、さらに他のネットワークで結ばれたコンピュータ上に位置する完全なモジュールへと提供されえる。

【００６５】

本発明の多くの特徴および利点は、記載された説明から明らかであり、よって添付の特許請求の範囲によって、本発明の全てのそのような特徴および利点が含まれることが意図される。さらに多くの変更および改変が当業者には容易になされえるので、図示され記載されたのと全く同じ構成および動作に本発明を限定することは望まれていない。したがって、全ての適切な改変および等価物は本発明の範囲に入るものである。

【図面の簡単な説明】

【００６６】

【図１】本発明の原理によって生成されたＸＭＬスキーマを実行するのに適切な例示的アプリケーションプログラムを示すブロック図である。

【図２】本発明の原理によってスキーマを自動的に生成するシステム実施形態のある実現例を示す簡略化されたブロック図である。

【図３】本発明のある方法実施形態を記載するフロー図である。

【図４】本発明のある実施形態によるＸＭＬファイルの構造を分析するある方法実施形態を記載するフロー図である。

【図５．１】本発明の原理によってＸＳＬＴスプレッドシートとして実現される構造アナライザを可能にする方法実施形態を示す図である。

【図５．２】本発明の原理によってＸＳＬＴスプレッドシートとして実現される構造アナライザを可能にする方法実施形態を示す図である。

【図６】本発明のある実施形態によって初期ＸＭＬ文書からデフォールト制約のセットを生成するある方法実施形態を記載するフロー図である。

【図７．１】本発明の原理によってＸＳＬＴスプレッドシートとして実現されるデフォールト制約生成を可能にする方法実施形態を示す図である。

【図７．２】本発明の原理によってＸＳＬＴスプレッドシートとして実現されるデフォールト制約生成を可能にする方法実施形態を示す図である。

【図７．３】本発明の原理によってＸＳＬＴスプレッドシートとして実現されるデフォールト制約生成を可能にする方法実施形態を示す図である。

【図８】本発明の原理によるＸＭＬスキーマ生成のある実施形態のための操作の例示的モードを示す簡略化されたフロー図である。

【図９．１】本発明の原理によってＸＳＬＴスプレッドシートとして実現されるＸＭＬスキーマ生成を可能にする方法実施形態を示す図である。

【図９．２】本発明の原理によってＸＳＬＴスプレッドシートとして実現されるＸＭＬスキーマ生成を可能にする方法実施形態を示す図である。

【図９．３】本発明の原理によってＸＳＬＴスプレッドシートとして実現されるＸＭＬスキーマ生成を可能にする方法実施形態を示す図である。

【図９．４】本発明の原理によってＸＳＬＴスプレッドシートとして実現されるＸＭＬスキーマ生成を可能にする方法実施形態を示す図である。

【図９．５】本発明の原理によってＸＳＬＴスプレッドシートとして実現されるＸＭＬスキーマ生成を可能にする方法実施形態を示す図である。

【図１０Ａ】本発明によって用いられえるコンピュータシステムの一例を示す図である。

【図１０Ｂ】本発明によって用いられえるコンピュータシステムの一例を示す図である。

【図１】

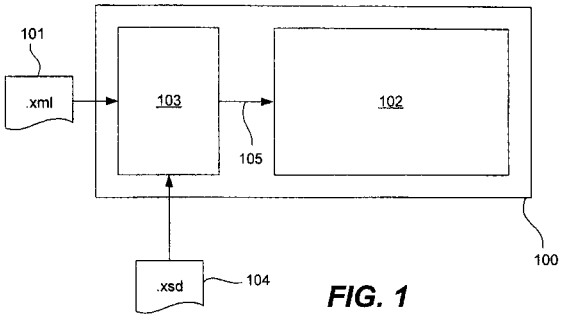

FIG. 1

【図２】

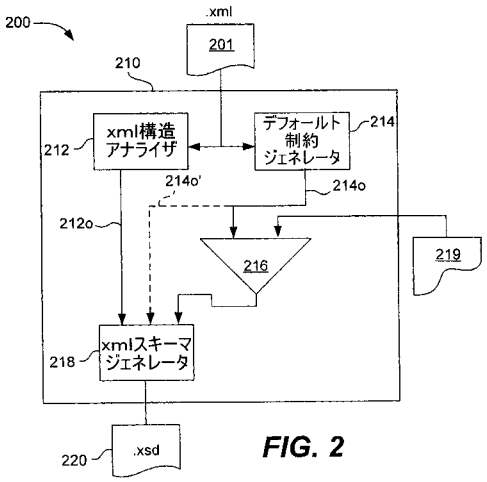

FIG. 2

【図３】

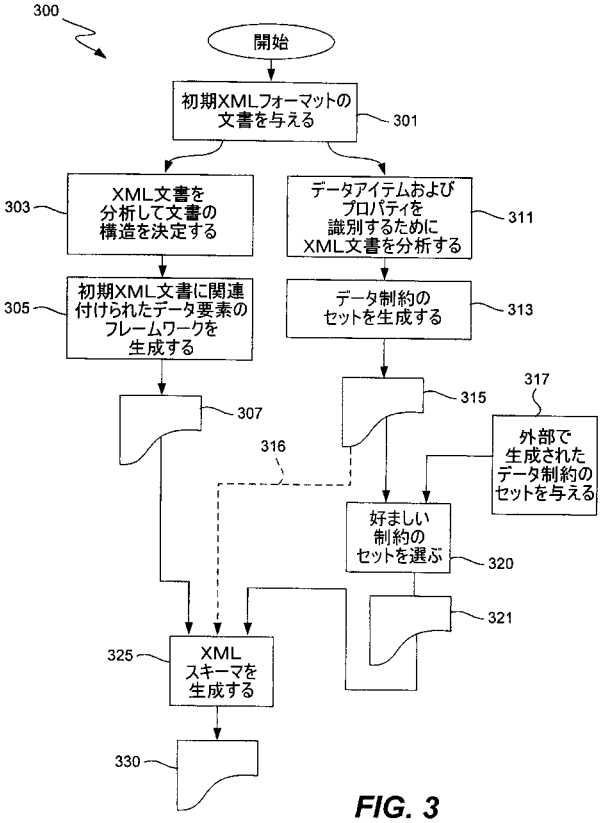

FIG. 3

【図４】

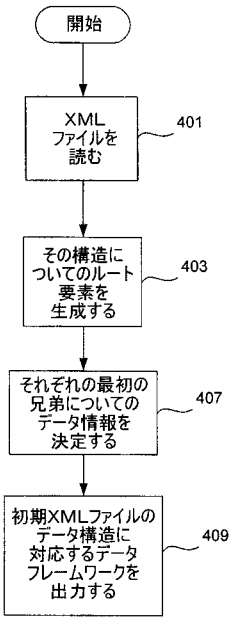

FIG. 4

【図５．１】

構造アナライザXMLスタイルシート

```
<xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transform version="1.0">
   <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/">
    <xsl:call-template name="process-all-category-children">
       <xsl:with-param name="children" select="*"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="process-all-category-children">
     <xsl:param name="children"/>

     <xsl:if test="count($children) > 0">
        <xsl:variable name="first-child"  select="local-name($children[1])"/>

        <xsl:call-template name="process-like-siblings">
           <xsl:with-param name="siblings" select="$children[local-name(.) = $first-child]"/>
        </xsl:call-template>

        <xsl:call-template name="process-all-category-children">
           <xsl:with-param name="children"  select="$children[local-name(.) != $first-child]"/>
        </xsl:call-template>
     </xsl:if>
  </xsl:template>

<!-- Pickup only the first sibling from the set of children for a node-->
  <xsl:template name="process-like-siblings">
     <xsl:param name="siblings"/>

     <xsl:variable name="category-name" select="$siblings[1]/@name"/>
     <xsl:variable name="el-name" select="local-name($siblings[1])"/>

     <xsl:element name="{$el-name}">

        <xsl:for-each select="$siblings[1]/@*">
           <xsl:variable name="attr-name" select="."/>
           <xsl:attribute name="{local-name()}"><xsl:value-of select="$attr-name"/></xsl:attribute>

                <xsl:attribute name="XSG_attrcount_{local-name()}"><xsl:value-of
     select="count($siblings/@*)"/></xsl:attribute>
           </xsl:for-each>

           <xsl:variable name="count"
     select="count($siblings[1]/../*[local-name() =
     local-name($siblings[1])])"/>
           <xsl:variable name="child_count" select="count($siblings/*)"/>

           <xsl:if test="$child_count = 0">
```

FIG. 5.1

【図５．２】

```
        <xsl:attribute name="XSG_val"><xsl:value-of select="$siblings[1]"/></xsl:attribute>
      </xsl:if>

      <xsl:attribute name="XSG_count"><xsl:value-of
select="$count"/></xsl:attribute>

      <xsl:call-template name="process-all-category-children">
        <xsl:with-param name="children" select="$siblings/*"/>
      </xsl:call-template>

    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

FIG. 5.2

【図６】

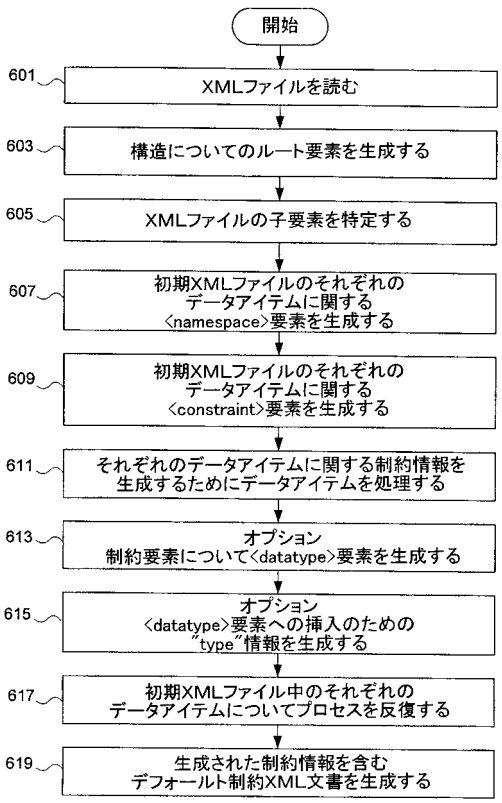

FIG. 6

【図７．１】

```
Example Default Constraint Generator XML Style Sheet
-------------------------------------------------------------------
> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>       xmlns:java="http://xml.apache.org/xalan/java"
>  version="1.0">
> <xsl:output method="xml" indent="yes"/>
>
> <!-- root node-->
>  <xsl:template match="/">
>  <root>
>    <namespaces>
>      <namespace name="xmlns:jaxb"
> value="http://java.sun.com/xml/ns/jaxb"/>
>    </namespaces>
>
>      <constraints>
>    <xsl:for-each select="*">
>
>      <xsl:variable name="el-name" select="local-name(.)"/>
>      <xsl:variable name="new-path" select="concat('/', $el-name)"/>
>
>      <xsl:variable name="elem-val" select="./@XSG_val"/>
>      <xsl:if test="string-length($elem-val) > 0">
>        <xsl:call-template name="gen-constraint">
>      <xsl:with-param name="name" select="local-name(.)"/>
>      <xsl:with-param name="path" select="$new-path"/>
>      <xsl:with-param name="type" select="'elem'"/>
>      <xsl:with-param name="val" select="$elem-val"/>
>        </xsl:call-template>
>      </xsl:if>
>
>       <xsl:for-each select="./@*">
>  <!-- do not process if node has XSG attributes-->
>        <xsl:if test="not(contains(local-name(.), 'XSG_'))">
>        <xsl:variable name="attr-val" select="."/>
>          <xsl:call-template name="gen-constraint">
>        <xsl:with-param name="name" select="local-name(.)"/>
>        <xsl:with-param name="path" select="$new-path"/>
>        <xsl:with-param name="type" select="'attr'"/>
>        <xsl:with-param name="val" select="$attr-val"/>
>          </xsl:call-template>
>        </xsl:if>
>       </xsl:for-each>
>       <xsl:call-template name="process-all-children">
>      <xsl:with-param name="children" select="./*"/>
>      <xsl:with-param name="path" select="$new-path"/>
>       </xsl:call-template>
>      </xsl:for-each>
>      </constraints>
```

FIG. 7.1

【図７．２】

```
>      <inserts>
>        <insert name="comment_def" path="/">
>      <xsd_element name="comment" type="xsd:string"/>

>        </insert>
>      </inserts>
>
>    </root>
>  </xsl:template>
>
> <!-- process all children -->
>  <xsl:template name="process-all-children">
>    <xsl:param name="children"/>
>    <xsl:param name="path"/>
> <!-- process if node has children-->
>    <xsl:if test="count($children) > 0">
> <!-- create a xsd:complexType tag-->
>
> <!-- call template to process other siblings-->
>      <xsl:for-each select="$children">
>        <xsl:variable name="el-name" select="local-name(.)"/>
>        <xsl:variable name="new-path" select="concat($path, '/',
> $el-name)"/>
>
>        <xsl:variable name="elem-val" select="./@XSG_val"/>
>        <xsl:if test="string-length($elem-val) > 0">
>          <xsl:call-template name="gen-constraint">
>        <xsl:with-param name="name" select="local-name(.)"/>
>        <xsl:with-param name="path" select="$new-path"/>
>        <xsl:with-param name="type" select="'elem'"/>
>        <xsl:with-param name="val" select="$elem-val"/>
>          </xsl:call-template>
>        </xsl:if>
>
>        <xsl:call-template name="process-all-children">
>      <xsl:with-param name="children" select="./*"/>
>      <xsl:with-param name="path" select="$new-path"/>
>        </xsl:call-template>
>
>
>      </xsl:for-each>
> <!-- end call template to process other siblings-->
>      </xsl:if>
> <!-- end process if node has children-->
>  </xsl:template>
```

FIG. 7.2

【図７．３】

```
>  <xsl:template name="gen-constraint">
>    <xsl:param name="name"/>
>    <xsl:param name="path"/>
>    <xsl:param name="type"/>
>    <xsl:param name="val"/>
>    <xsl:variable name="datatype"><xsl:value-of
> select="java:DataType.getType($name,$val)"/></xsl:variable>
>         <xsl:element name="constraint">
>              <xsl:attribute name="type"><xsl:value-of
> select="$type"/></xsl:attribute>
>              <xsl:attribute name="name"><xsl:value-of
> select="$name"/></xsl:attribute>
>              <xsl:attribute name="path"><xsl:value-of
> select="$path"/></xsl:attribute>
>              <datatype>
>              <xsl:attribute name="type"><xsl:value-of
> select="$datatype"/></xsl:attribute>
>              </datatype>
>         </xsl:element>
>  </xsl:template>
>
> </xsl:stylesheet>
```

FIG. 7.3

【図８】

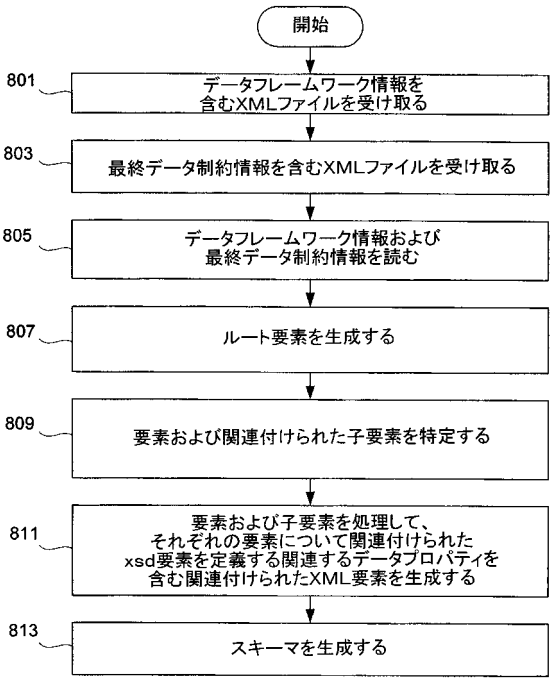

FIG. 8

【図９．１】

```
Process for Generating Schema
> ----------------------------------------------------------------------
> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>       xmlns:java="http://xml.apache.org/xalan/java"
>  version="1.0">
>  <xsl:output method="xml" indent="yes"/>
>  <xsl:param name="constraint-document" />
> <!-- root node-->
>  <xsl:template match="/">
>
>    <xsl:element name="xsd_schema">
>    <xsl:attribute
> name="xmlns_xsd">http://www.w3.org/2001/XMLSchema</xsl:attribute>
>
>    <xsl:variable name="namespaces"
> select="document(normalize-space($constraint-document))/root/namespaces/namespace"/>
>
>    <xsl:variable name="xmlns_jaxb" select="$namespaces[@name =
> 'xmlns_jaxb']/@value"/>
>      <xsl:if test="string-length($xmlns_jaxb) > 0">
>       <xsl:attribute name="xmlns_jaxb"><xsl:value-of
> select="$xmlns_jaxb"/></xsl:attribute>
>      </xsl:if>
>
>    <xsl:variable name="xmlns_xjc" select="$namespaces[@name =
> 'xmlns_xjc']/@value"/>
>      <xsl:if test="string-length($xmlns_xjc) > 0">
>       <xsl:attribute name="xmlns_xjc"><xsl:value-of
> select="$xmlns_xjc"/></xsl:attribute>
>      </xsl:if>
>
>    <xsl:variable name="jaxb_version" select="$namespaces[@name =
> 'jaxb_version']/@value"/>
>      <xsl:if test="string-length($jaxb_version) > 0">
>       <xsl:attribute name="jaxb_version"><xsl:value-of
> select="$jaxb_version"/></xsl:attribute>
>      </xsl:if>
>
>    <xsl:variable name="jaxb_extensionBindingPrefixes"
> select="$namespaces[@name = 'jaxb_extensionBindingPrefixes']/@value"/>
>      <xsl:if test="string-length($jaxb_extensionBindingPrefixes) > 0">
>       <xsl:attribute
> name="jaxb_extensionBindingPrefixes"><xsl:value-of
> select="$jaxb_extensionBindingPrefixes"/></xsl:attribute>
>      </xsl:if>
>
>    <xsl:copy-of
> select="document(normalize-space($constraint-document))/root/inserts/insert[@path
> = '/']/*"/>
```

FIG. 9.1

【図９．２】

```
>    <xsl:for-each select="*">
>     <xsl:element name="xsd_element">
>      <xsl:attribute name="name"><xsl:value-of
> select="local-name(.)"/></xsl:attribute>
>       <xsl:if test="count(./*) > 0">
>        <xsl:attribute name="type">XSG_<xsl:value-of
> select="local-name(.)"/></xsl:attribute>
>       </xsl:if>
>       <xsl:if test="count(./*) = 0">
>        <xsl:attribute name="type">xsd:string</xsl:attribute>
>       </xsl:if>
>     </xsl:element>
>    </xsl:for-each>
>
>    <xsl:for-each select="*">
>     <xsl:variable name="el-name" select="local-name(.)"/>
>     <xsl:variable name="new-path" select="concat('/', $el-name)"/>
>     <xsl:call-template name="process-all-children">
>    <xsl:with-param name="children" select="./*"/>
>    <xsl:with-param name="path" select="$new-path"/>
>     </xsl:call-template>
>    </xsl:for-each>
>
>          <xsl:variable name="attr-constraints"
> select="document(normalize-space($constraint-document))/root/constraints/
constraint[@type='attr']"/>
>          <xsl:variable name="attr-constraints-refs"
> select="$attr-constraints/datatype[@type='ref']"/>
>           <xsl:copy-of select="$attr-constraints-refs/*"/>
>
> </xsl:element>
>
>  </xsl:template>
>
> <!-- process all children -->
>  <xsl:template name="process-all-children">
>    <xsl:param name="children"/>
>    <xsl:param name="path"/>
> <!-- process if node has children-->
>    <xsl:if test="count($children) > 0">
> <!-- create a xsd:complexType tag-->
>     <xsl:element name="xsd_complexType">
>      <xsl:attribute name="name">XSG_<xsl:value-of
> select="local-name($children[1]/..)"/></xsl:attribute>
>       <xsl:element name="xsd_sequence">
> <!-- call template to process like siblings-->
>        <xsl:call-template name="process-like-siblings">
>      <xsl:with-param name="siblings" select="$children"/>
>      <xsl:with-param name="path" select="$path"/>
```

FIG. 9.2

【図９．３】

```
>       </xsl:call-template>
>       <xsl:copy-of
> select="document(normalize-space($constraint-document))/root/inserts/insert[@path
> = $path]/*"/>
>     </xsl:element>
> <!-- create attributes for each xsd:complexType tag-->
>     <xsl:for-each select="./@*">
>       <xsl:variable name="attr_name" select="local-name(.)"/>
>       <xsl:variable name="attr_val" select="."/>
>       <xsl:if test="not(contains($attr_name,'XSG_'))">
>        <xsl:element name="xsd_attribute">
>        <xsl:attribute name="name"><xsl:value-of
> select="$attr_name"/></xsl:attribute>
>        <xsl:variable name="attr_type" select="xsd_date"/>
>        <xsl:variable name="attr-constraints"
> select="document(normalize-space($constraint-document))/root/constraints/
constraint[@type='attr']"/>
>        <xsl:variable name="constraint-path"
> select="$attr-constraints[@path=string($path)]"/>
>        <xsl:variable name="constraint-path-val"
> select="$constraint-path/@path"/>
>        <xsl:variable name="constraint-attr-name"
> select="$constraint-path/@name"/>
>        <xsl:variable name="datatype"
> select="$constraint-path/datatype/@type"/>
>        <xsl:variable name="dataname"
> select="$constraint-path/datatype/@name"/>
>        <xsl:if test="($attr_name = $constraint-attr-name)">
>          <xsl:attribute name="type"><xsl:value-of
> select='$datatype'/></xsl:attribute>
>        </xsl:if>
>
>        <xsl:if test="$datatype = 'ref'">
>          <xsl:attribute name="type"><xsl:value-of
> select="$dataname'/></xsl:attribute>
>        </xsl:if>
>
>
>        <xsl:if test="not($attr_name = $constraint-attr-name)">
>         <xsl:attribute name="type"><xsl:value-of
> select="java:DataType.getType($attr_name,$attr_val)"/></xsl:attribute>
>        </xsl:if>
```

FIG. 9.3

【図９．４】

```
        <xsl:if test="string-length('$attr_val') > 0">
>         <xsl:attribute name="use">required</xsl:attribute>
>        </xsl:if>
>       </xsl:element>
>      </xsl:if>
>     </xsl:for-each>
>    </xsl:element>
> <!-- end create xsd:complexType tag-->
>
> <!-- call template to process other siblings-->
>     <xsl:for-each select="$children">
>       <xsl:variable name="el-name" select="local-name(.)"/>
>       <xsl:variable name="new-path" select="concat($path, '/',
> $el-name)"/>
>       <xsl:call-template name="process-all-children">
>     <xsl:with-param name="children" select="./*"/>
>     <xsl:with-param name="path" select="$new-path"/>
>       </xsl:call-template>
>     </xsl:for-each>
> <!-- end call template to process other siblings-->
>     </xsl:if>
> <!-- end process if node has children-->
>   </xsl:template>
>
> <!-- process like siblings-->
>   <xsl:template name="process-like-siblings">
>     <xsl:param name="siblings"/>
>     <xsl:param name="path"/>
>     <xsl:variable name="category-name" select="$siblings[1]/@name"/>
> <!-- create xsd:element tag for each siblings-->
>     <xsl:for-each select="$siblings">
>       <xsl:variable name="el-name" select="local-name(.)"/>
>       <xsl:variable name="new-path" select="concat($path, '/',
> $el-name)"/>
>        <xsl:element name="xsd_element">
>         <xsl:attribute name="name"><xsl:value-of
> select="local-name(.)"/></xsl:attribute>
>          <xsl:if test="count(./*) > 0">
>           <xsl:attribute name="type">XSG_<xsl:value-of
> select="local-name(.)"/></xsl:attribute>
>           <xsl:variable name="count" select="$siblings[1]/@XSG_count"/>
>           <xsl:if test="number($count) > 1">
>            <xsl:attribute name="minOccurs">1</xsl:attribute>
>            <xsl:attribute name="maxOccurs">unbounded</xsl:attribute>
>           </xsl:if>
>          </xsl:if>
>         <xsl:variable name="elem-constraints"
```

FIG. 9.4

【図９．５】

```
> select="document(normalize-space($constraint-document))/root/constraints/
constraint[@type='elem']"/>
>         <xsl:variable name="constraint-path"
> select="$elem-constraints[@path=string($new-path)]"/>
>         <xsl:variable name="constraint-path-val"
> select="$constraint-path/@path"/>
>         <xsl:variable name="constraint-attr-name"
> select="$constraint-path/@name"/>
>
>         <xsl:if test="count(./*) = 0">
>           <xsl:if test="$constraint-path-val = $new-path">
>            <xsl:variable name="datatype"
> select="$constraint-path/datatype/@type"/>
>
>             <xsl:if test="$datatype = 'local'">
>               <!--<xsl:attribute name="type"><xsl:value-of
> select='$datatype'/></xsl:attribute>-->
>               <xsl:copy-of select="$constraint-path/datatype/*"/>
>             </xsl:if>
>
>             <xsl:if test="starts-with($datatype, 'xsd:')">
>              <xsl:attribute name="type"><xsl:value-of
> select='$datatype'/></xsl:attribute>
>
>           <xsl:variable name="xsg_val" select="./@XSG_val"/>
>             <xsl:if test="not($constraint-path-val = $new-path)">
>            <xsl:attribute name="type"><xsl:value-of
> select="java:DataType.getType($el-name,$xsg_val)"/></xsl:attribute>
>              </xsl:if>
>             </xsl:if>
>            </xsl:if>
>         </xsl:if>
>       </xsl:element>
>     </xsl:for-each>
> <!-- end create xsd:element tag for each siblings-->
>    </xsl:template>
>
> </xsl:stylesheet>
```

FIG. 9.5

【図１０Ａ】

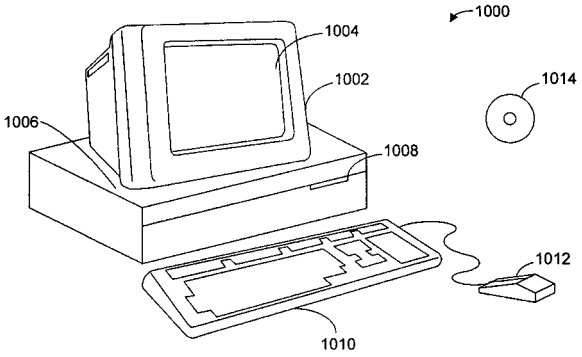

Fig. 10A

【図１０Ｂ】

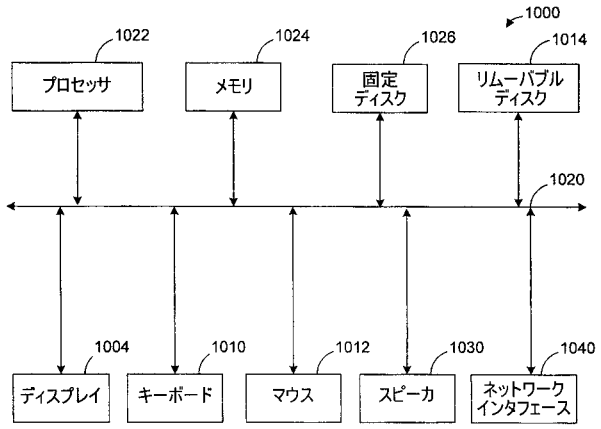
1000
1022
1024
1026
1014
プロセッサ
メモリ
固定
ディスク
リムーバブル
ディスク
1020
1004
1010
1012
1030
1040
ディスプレイ
キーボード
マウス
スピーカ
ネットワーク
インタフェース

Fig. 10B

【外国語明細書】
1. TITLE OF THE INVENTION
SYSTEM AND METHOD FOR AUTOMATICALLY GENERATING XML SCHEMA FOR VALIDATING XML INPUT DOCUMENTS

2. DETAILED DESCRIPTION OF THE INVENTION
BACKGROUND OF THE INVENTION
[0001] XML (EXtensible Markup Language) is a self-descriptive markup language that is finding ever wider application as a data transmission and processing tool. XML is efficient at describing and defining data and is therefore used ever increasingly in data intensive applications. In this way XML is different from HTML, which was designed for displaying data.

[0002] XML data can be readily transmitted between computers using any type of data transmission media. This XML data can be processed using computer programs and other suitably configured applications. Commonly, an XML file is received by an application and processed to generate an output. For example, in one implementation, XML can be used to provide inventory information. Such information can be provided in the form of an XML compliant document (referred to herein as and XML document of xml.doc). In one implementation, such information could, for example, be formatted as follows:

```
<Camera>
      <name>Canon-Sure-Shot-Z155</name>
      <f-stop>4.8-11.7</f-stop>
      <focal length>37-155mm zoom</ focal length >
      <cost>$318.00USD</cost>
</Camera>
```

[0003] This simplified example of an XML document provides an illustration of inventory information relating to a camera. In an XML document, the data items are formatted as parts of XML elements, with XML documents containing one or more such elements. In XML, a data item is "wrapped" between start/end tags to form an XML element. For example, a start tag "<name>" and an end tag "</name>" wrap the data element "Canon-Sure-Shot-Z155" to form an XML element "<name>Canon-Sure-Shot-Z155</name>".

[0004] A more complex element can be defined using, for example, an element defined by < camera > ... </camera>. Methods and formats used to describe XML data are, of course, well known to those having ordinary skill in the art and so will not be discussed in detail here.

[0005] Using the above-described example, the XML information conforms to an XML data structure. As used here, an XML data structure refers to the arrangement and organizational format of "empty" data elements. Such structure is defined by the arrangement and format that defines the relationship of elements to each other within a given XML document. Again, using the above-described example, the structure of the XML document can be formatted as follows:

```
<Camera>
```

```
            <name> ...</name>
            <f-stop> ... </f-stop>
            <focal length> ... </ focal length >
            <cost> ... </cost>
      </Camera>
```

[0006] The foregoing simplified example defines an example data framework th at defines a data structure for the example document.

[0007] In many implementations, XML documents are used to provide data to ap plications that perform various operations using the XML data. Commonly, such applications are configured to receive the XML data in a given order and having a specified format. If the data is provided in an incorrect order or having an improper format, it may be unusable by an application. Improperly configured X ML data can cause application programs to fail or crash or cause other undesirab le outcomes. Under such circumstances the XML document (and associated data) ar e considered "invalid". Consequently, applications are commonly equipped with s mall programs that "validate" received XML documents. If the XML documents cont ain XML data in the proper order and of the correct format it is said to be vali d and the application can operate on the data. One approach used to validate XM L documents is to use an XML schema (also referred to as .xsd files) to validate the XML data. The validation schema can be included as part of the application or used as an add-on validation module. XML schema are used to describe the st ructure of XML documents. As is known to those having ordinary skill in the art ,XML schema are useful for defining elements or attributes that can appear in a document. XML schema can be used to define whether elements are child elements and the number and order of child elements. XML schema can also define whether an element is empty or can include text and can also define data types for elem ents and attributes as well as defining default and fixed values for elements an d attributes. These attributes are quite useful for defining and validating XML documents.

[0008] However, in common usage, the data and structure of XML documents are constantly changing. Additionally, each change of data or structure typically necessitates a corresponding change in the associated XML schema. In the curr ent art, such schema must be changed or rewritten by hand to implement the chang es. In the short example provided herein above such changes may be relatively simple to effectuate. However, most validation schema are very long with many e lements and the process of carefully and accurately generating suitable schema c an be extremely time consuming. Using conventional processes generating accurat e schema capable of validating XML documents is a laborious, meticulous, and tim e consuming process.

SUMMARY OF THE INVENTION

[0009] Broadly speaking, the invention relates to techniques, systems and ap paratus for automatically generating schema without the need for recoding by han d. In particular, embodiments of the invention can automatically generate schem a using an initial documents constructed in an XML compatible format.

[0010] The invention can be implemented in numerous ways, including a system

, an software module, a method, or a computer readable medium as well as other i mplementations. Several embodiments of the invention are discussed below.

[0011] One embodiment of the invention is directed to a method for generatin g XML schema. Such method involves the operations of providing an initial XML d ocument that includes raw XML data comprising data items arranged in XML data el ements and analyzing the XML document to identify XML data structures and theref rom generating a data framework that corresponds to the format of the data struc tures in the XML document. The method involves analyzing the data items of the initial XML document to determine data constraints based on the data items. XML schema are then generated based on the data framework generated and the data co nstraints determined from the raw xml data.

[0012] In another embodiment, the disclosure teaches a computer program prod uct embodied in a computer readable media that includes code for generating XML schema. The computer program product includes code for receiving an initial XML document that that includes raw XML data comprising data items arranged in XML data structures. The product includes code for analyzing the XML document to id entify the XML data structures. The program code generates a data framework ass ociated with the format of the data structures in the XML document and includes code for analyzing data items from the initial XML document and determining XML data constraints based on the data items. The code also includes instructions f or generating XML schema based on the data framework generated and the XML data constraints determined from the data items.

[0013] In another embodiment the principles of the present invention enable a computer system. The computer system including at least one central processin g unit (CPU), memory, and user interface in combination configured to include an XML structure analyzer for analyzing a received initial XML document, wherein s aid analyzing includes identifying the XML data elements of the initial XML docu ment and generating a data framework associated with a data structure for the XM L data elements of the initial XML document. Additionally, the system includes a default constraint generator for analyzing the raw XML data from the initial X ML document and determining XML data constraints for data items based on the XML raw data. The system includes a data constraint merger for receiving at least one of the XML data constraints from the data constraint generator and a set of externally supplied data constraints and for outputting a final data constraint file that is input into a system XML schema generator that receives the data str uctures from the XML structure analyzer and final data constraint file from the data constraint merger and generates an XML schema associated with said data str uctures and final data constraint file.

[0014] In another embodiment, a computer module is disclosed. The module co mprising an XML structure analyzer for analyzing a received initial XML document that includes raw XML data comprising data items arranged in XML data elements, wherein said analyzing includes identifying the XML data elements of the initia l XML document and generating a data framework associated with a data structure for the XML data elements of the initial XML document. The module further inclu des a default constraint generator for analyzing the raw XML data from the initi al XML document and determining XML data constraints for data items based on the

XML raw data. A data constraint merger is included for receiving at least one of the XML data constraints from the data constraint generator and a set of externally supplied data constraints and for outputting a final data constraint file. Also, an XML schema generator is included for receiving the data structures from the XML structure analyzer and final data constraint file from the data constraint merger and generating an XML schema associated with said data structures and final data constraint file.

[0015] In another embodiment, another module is disclosed. This module includes an XML structure analyzer for analyzing a received initial XML document that includes raw XML data comprising data items arranged in XML data elements, wherein said analyzing includes identifying the XML data elements of the initial XML document and generating a data framework associated with a data structure for the XML data elements of the initial XML document. The module further includes a default constraint generator for analyzing the raw XML data from the initial XML document and determining XML data constraints for data items based on the XML raw data. An XML schema generator is included for receiving the data structures from the XML structure analyzer and default data constraint file from the default constraint generator and generating therefrom an XML schema associated with said data structures and final data constraint file.

[0016] Other aspects and advantages of the invention will become apparent from the following detailed description, taken in conjunction with the accompanying drawings, illustrating by way of example the principles of the invention.

[0017] The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements.

DETAILED DESCRIPTION OF THE INVENTION

[0018] The invention relates to techniques, systems, and modules for automatically generating XML schema capable of use for validating XML documents. Embodiments of the invention are discussed below with reference to Figs. 1-10B. However, those skilled in the art will readily appreciate that the detailed description given herein with respect to these figures is for explanatory purposes as the invention extends beyond these limited embodiments.

[0019] In accordance with one aspect of the invention a typical application for implementing an XML schema is described. Fig. 1 is a simplified block diagram illustrating an application program 100 for use in accordance with the principles of the invention. The program 100 includes an example process 102 for operating on XML documents 101 supplied to the program 100. In the depicted embodiment a validator 103 (e.g., a validation program) is used to validate XML documents 101 provided to the program 100. The validator 103 compares the data structure and the format of the data elements of the XML document 101 with an XML schema 104 (also provided to the validator) to determine if the structure and format of the content of the XML document 101 is "valid" (i.e., correctly configured and structured) to operate in the process 102. Generally, this means that the structure and format of the content of the XML document 101 matches that of the XML schema 104. The validator 103 outputs a valid XML document 105. The invent

or notes that although the depicted validator 103 forms part of the program 100, this need not be the case. A validator 103 can be separate from the process 102. The validator 103 simply provides the validated XML file 105 to the process 102.

[0020] Fig. 2 depicts one implementation of a system 200 for automatically generating schema in accordance with the principles of the invention. In the depicted embodiment of the invention, the system can be implemented as a software module 202. As persons of ordinary skill will be aware, the system can be implemented in many other configurations. For example, the system can form part of an operating system, be implemented as part of an applications program, additionally, the system can be implemented as hardware. In accordance with some embodiments of the invention, the system is implemented as a software module 210 that can be employed to generate XML schema. As alluded to above, in one embodiment, the module can be employed as part of an application or as part of some other program. Additionally, as depicted here, in some embodiments the module 210 forms part of a computer system. The module 210 is configured to receive an XML file 201 and use the file 201 to generate an associated XML schema 220 that can be used to validate other XML files. Such validation can, for example, be used to validate other XML files intended for input into various application programs.

[0021] An XML document 201 containing XML data structures and associated XML data items (also referred to herein as raw XML data) is provided to the module 210. The module 210 includes an XML structure analyzer 212 capable of receiving an XML document and transforming the document into a data framework that defines the data structure for the XML document 201. Generally, the data framework comprises the arrangement and structure of the empty elements comprising the XML document 201. This framework information is output 212o to an XML schema generator 218. Additionally, the module can include a default constraint generator 214. The default constraint generator 214 is capable of receiving an XML document, identifying the data items of the XML document 201 and defining a set of properties for those data items. For example, one set of properties that can be identified and defined for the data items is the data types (e.g., string, numeric (e.g., integer, decimal, etc.), binary, boolean, date, time, anyURI, double, float, NOTATION, Qname, as well as many other "types") for each of the data items. These properties can be used to identify specific properties for each data item and can generate constraints for each data item. This constraint information is output from the default constraint generator 214 as a default constraint file 214o (e.g., as an XML file). The default constraint file 214o can be received by a constraint merger 216 which provides selected constraint information to the XML schema generator 218. The schema generator 218 uses the selected constraint information together with the data structure to translate an XML document 201 into an XML schema associated with the XML document 201. Additionally, a user can supply additional constraint information in the form of an externally generated constraint file 219 that is also input into the constraint merger 216. The constraint merger 216 can choose to select the default constraint file 214o or the externally generated constraint file 219 for input into the XML schema generator 218. Alternatively, the constraint merger 216 can merge the default constraint file 214o with the externally generated constraint file 219 for to generate an input for the XML schema generator 218.

[0022] Alternatively, the module 210 can be configured so that the default constraint file 214o is received directly by the schema generator 218 (see the dashed line 214o') without using the merger 216 and without the externally generated constraints 219. In such case no externally generated constraints 219 are used and the merger 216 is not required.

[0023] Fig. 3 is a flow diagram that describes one typical method embodiment of the invention. The process flow 300 can begin by providing an initial XML document (Step 301). This document is typically a correctly formatted document that would be suitable for processing by the desired applications. For example, the XML document can be provided to a software module or other system embodied herein for generating an XML schema. The initial XML document is a properly formatted XML document configured in compliance with an XML format. The initial XML document typically includes XML data structures and raw XML data (XML data items) that will be used in generating an associated XML schema. Providing the initial XML document generally includes providing the initial XML document to both an XML structure analyzer (e.g., 212) and an XML default constraint generator (e.g., 214). The initial XML document is analyzed to identify the XML structure of the document (Step 303). This can be accomplished using an XML structure analyzer (e.g., 212). The XML elements are identified and the relationships between the elements are discerned. A data structure framework associated with data elements of the initial XML document is then generated using the information obtained from the analysis of the initial XML document (Step 305). This framework typically defines a pattern of empty XML data elements configured in an arrangement analogous to that of the initial XML document. The output data structure framework 307 can be output as an XML document (XML.doc) for later use in the process.

[0024] The initial XML document is also analyzed to identify the XML data elements (raw XML data) of the document (Step 311). This can be done at the same time as steps 303, 305 or at a different time. Typically, this analysis can be accomplished using an XML default constraint generator (e.g., 214). The XML data items are identified and data properties pertaining to the data items are ascertained. For example, the data types for the data items are identified and the data attributes are also identified. A set of default data constraints is then generated for each data item (Step 313). The default data constraints are generated based on the data properties of the data items in the initial XML document. Commonly, the default data constraints include type and attribute information as well as other data property constraints. The output set of default data constraints 315 can also be output as an XML document. Example data constraints include but are not limited to attributes that indicate target information (i.e., an element or attribute to which the constraint has to be applied). For example, the type of target can be identified. In one example, target can be identified as type element (type="elem") or as type attribute (type="attr"). The name of the target can be identified (e.g., name="orderDate"). A pathname specifying a path of the target from the root (/) can be identified (e.g., path="/purchaseOrder"). In one limited example implementation attributes and elements can be constrained in accordance with the following example formats. In one example, attribute information can be constrained as follows: <constraint type="attr"

name="orderDate" path="/purchaseOrder">. In another example, element information can be constrained as follows: <constraint type="elem" name="quantity" path="/purchaseOrder/items/item/quantity">. As can be appreciated by those of ordinary sill in the art many other constraint formats can be utilized and those shown are merely illustrative rather than limiting. In another example, a datatype element can hold default datatype information inside a type attribute, in one example, formatted as follows:

(e.g., <datatype type="xsd:string"> </datatype>).

[0025] As can be appreciated by those of ordinary skill in the art, many other constraint formats can be utilized and those shown above are merely intended to be illustrative rather than limiting.

[0026] In one embodiment, the output data structure framework 307 and the output set of default data constraints 315 are used to automatically generate a schema without any further user input beyond the introduction of the initial XML document (Step 325). In one embodiment, the output data structure framework 307 and the output default set of data constraints 315 (dashed line 316) are received by a schema generator (e.g., 218) and processed to automatically generate an XML schema 330 capable of validating XML files.

[0027] In another embodiment, an additional externally generated set of constraints can be provided (Step 317). An externally generated constraint file can be provided by a user (or generated by a machine in accordance with a predetermined set of conditions). For example, for a zip code constraint can be confined to a type=integer, but also constrained to only validate data items having five entries (i.e., "90505") or alternatively be constrained to validate data items having nine entries of the XXXXX-XXXX format (i.e., "90505-1405"). The externally generated set of constraints 317 and the default constraints 315 can be processed together to obtain a selected set of constraints (Step 320). In one instance, the two (or more) sets of constraints 315, 317 can be input into a constraint merger (e.g., 216) which can select either of the constraints to provide a selected output constraint 321 (typically in the form of an XML file) which is used with the output data structure framework 307 to automatically generate an XML schema 330 capable of validating XML files (Step 325). In a simple case, for example, a merger operating at Step 320 can simply select the set of constraints 315, 317 that is externally generated by a user (e.g., 317) as the selected output constraint 321.

[0028] In addition to the above approach, another embodiment can process the externally generated set of constraints 317 and the default constraints 315 together to obtain another selected set of constraints 321 (Step 320). In this case, the constraints 315, 317 can be input into a constraint merger (e.g., 216) which can merge the constraints 315, 317 to provide a selected output constraint 321 in the form of a merged file (typically in an XML format) that includes data from both files. This merged file is used with the output data structure framework 307 to automatically generate an XML schema 330 capable of validating XML files (Step 325). In a simple case, for example, the externally generated constraint information can be used to fill in any gaps in a default constraint docume

nt. Additionally, in conditions where constraint information in the default con straint file is in conflict with constraint information contained in the externa lly generated constraint file, conflicts can be resolved in accordance with a pr edetermined conflict resolution scheme. For example, a priority scheme can be implemented. Or the constraint information contained in an externally generated constraint file can be chosen over the default information.

[0029] The following discussion describes to one example embodiment of a stru ctural analyzer (e.g., 212) and an associated mode of operation. In general, a structural analyzer reads an input XML document, analyzes the structure of the X ML data elements, and generates a resultant XML document that contains structura l information in the form of a framework. This framework is generally conceived of as comprising XML data elements, attribute information, and selected metadat a concerning the cardinality of data elements and data attributes.

[0030] Fig. 4 describes a simplified flow diagram 400 illustrating an example mode of operation for one embodiment of a structural analyzer (e.g., 212) opera ting to accomplish operations 303, 305 of Fig. 3.

[0031] The initial XML file is read (Step 401) and a "root element" is gener ated for the structure (Step 403). In one embodiment a system parser can be us ed to generate the root element. The "child" elements of the input XML file are identified (Step 405). Since "sibling" data elements have the same data forma t as other associated siblings (the exact content of the data can vary, but the data format is the same) only the first sibling need be analyzed to identify the data properties needed to specify the data format for any of the siblings. And since the purpose of the validation schema is to confirm that an input XML docu ment has a correct format, such analysis of the first sibling is sufficient. Ea ch first sibling can be processed to determine its name, attributes, number of a ttributes (attribute count), value, and element count (Step 407). Commonly, the redundant children need not be analyzed since they specify the same data format as the other related siblings. The resulting output is a data structure that c aptures the organizational format of the initial XML file (Step 409).

[0032] In one illustrative example, Table 1 provides an input initial XML fi le.

Table 1

```
<?XML version="1.0" encoding="UTF-8"?>
  <purchaseOrder orderDate="1999-10-20">
    <items>
      <item partNum="242-NO" >
        <productName>Nosferatu - Special Edition (1929)</productName>
        <quantity>5</quantity>
        <USPrice>19.99</USPrice>
      </item>
      <item partNum="243-NO" >
        <productName>The Mummy (1959)</productName>
        <quantity>3</quantity>
        <USPrice>19.98</USPrice>
```

```
      </item>
   </items>
</purchaseOrder>
```

[0033] The initial XML file of Table 1 can be used to generate the following resultant framework (see, Table 2) for the data elements associated with the initial XML file.

Table 2

```
<purchaseOrder orderDate="1999-10-20" XSG_attrcount_orderDate="1"  XSG_count="1"
>
   <items XSG_count="1">
      <item partNum="242-NO" XSG_attrcount_partNum="2" XSG_count="2">
         <productName XSG_val="Nosferatu - Special Edition (1929)"          XSG
_count="1"/>
         <quantity XSG_val="5" XSG_count="1"/>
         <USPrice XSG_val="19.99" XSG_count="1"/>
     </item>
   </items>
</purchaseOrder>
```

[0034] The root element is generated as, for example, purchaseOrder. The structure of the above framework includes: <purchaseOrder orderDate="1999-10-20" XSG_attrcount_orderDate="1" XSG_count="1"> to describe the element: "purchaseOrder" in its entirety.

[0035] For example, orderDate="1999-10-20" describes the attribute called orderDate. Also, XSG_attrcount_orderDate - describes the cardinality (number) of attribute orderDate for all elements of type purchaseOrder in the XML document. XSG_count="1" describes the cardinality of elements of type purchaseOrder in the XML document. The first data element for item partNumber (i.e., "242-NO") has been kept and defined. These same definitions and framework parameters apply to all other sibling data elements for item partNumber (e.g., "243-NO") which has already been defined as a framework element. Therefore, the item partNumber relating to "243-NO" need not be analyzed. The same sort of element definition is conducted for the other data elements e.g., productName; quantity, and USPrice.

[0036] Fig. 5 describes one embodiment of, for example, an XSLT style sheet that can be used to execute the process illustrated in Fig. 4 or the operate the structural analyzer (e.g., 212) executing operations 303, 305 or Fig. 3.

[0037] The stylesheet embodiment of Fig. 5 represents one approach to generating the structural framework of an initial XML document. As is known to those having ordinary skill in the art, many other approaches can be used. The embodiment of Fig. 5 begin by identifiing a root (/) element of the input XML document and designates that element as the "current element". All of the "child" elements in the "current element" are processed and the first sibling is identified. The first sibling is then processed to generate an XML element containing, at least in this embodiment, the following information:

Name, here containing the element name of the first sibling (purchaseOrder);

Attributes (e.g.,orderDate="1999-10-20");

Attribute count (i.e., cardinality information for each attribute). Example, shown here can include:
XSG_attrcount_orderDate="1"

Also, value (i.e., the value of text node elements, if present). For example, a value like: XSG_val="Nosferatu - Special Edition (1929)") for item productName.

Also, element count (the cardinality information for each element) can be discerned, here XSG_count="1". This process can be repeated for all first children until all elements (first children) of the initial XML file are visited. Such a process is one example of a process used to generate structural information in accordance with one embodiment of the invention.

[0038] As previously alluded to, the structure of the invention includes a default constraint generator. Fig. 6 describes a simplified flow diagram 600 illustrating an example mode of operation for one embodiment of a default constraint generator (e.g., 214) operating to accomplish operations 311, 313 of Fig. 3.

[0039] The initial XML file is read (Step 601) and a "root element" is generated for the structure (Step 603). In one embodiment, a system parser can be used to generate the root element. The "child" elements of the input XML file are identified (Step 605). Elements are generated to contain constraint information related to each data item of the initial XML file (Step 607). In one embodiment, these elements are referred to as <namespaces> elements. Each data item is then associated with a <namespace> data element. Additionally, each <namespace> element is configured to include a <constraint> element (Step 609) that includes constraint information concerning each data item. The data items of the initial XML file are processed to generate associated constraint information (Step 611). Each data item is analyzed to determine a "name" which identifies the item by name, analyzed to determine "attributes" which identify attributes of the data item, and analyzed to determine a "path" that identifies a pathname of the structure that identifies the pathname for the path of the element in the XML structure where the attribute or element belongs. Also, another element can be created for the constraint element (Step 613). This element can contain "type" information for the generated element. In one embodiment, such an element can be referred to, for example, as a "datatype" element. In such case the "type" element comprises a single attribute that is generated (Step 615) and whose value represents a generated data type from the initial XML file input into the default constraint generator. This process can be repeated for each data item in the initial XML file (step 617) to provide a complete characterization of the data items in the initial XML file an generate a default constraint XML document.

[0040] In general, the above process generates three blocks of information for use in a default constraint XML file. First a "namespaces" element can be generated from the input document. This element can be modified to change namespace information during generation an XML Schema (e.g., 218).

[0041] For example, having a "namespaces" element configured as follows:

```
     <namespaces>
        <namespace value="http://java.sun.com/XML/ns/jaxb"
name="XMLns:jaxb">
     </namespace>
   </namespaces>
```

[0042] An additional namespace can be easily added by inserting the new name space inside the "namespaces" element.

[0043] Also, data types can be easily modified in accordance with the principles of the invention. For example, the following constraint element is initially configured as follows:

```
     <constraint type="attr" name="orderDate" path="/purchaseOrder">
          <datatype type="xsd:date">
          </datatype>
        </constraint>
```

[0044] This describes a generated datatype as xsd:date for orderDate based on the "1999-10-20" content.

[0045] If a user desires to change the type for this attribute (datatype) to be xsd:time during XML Schema generation, then it can be changed, for example, as follows:

```
     <constraint type="attr" name="orderDate" path="/purchaseOrder">
          <datatype type="xsd:time">
          </datatype>
     </constraint>
```

[0046] This modified constraint can now be used for generating XML Schema document in accordance with the modified type information.

[0047] Furthermore, additional elements (comments, etc.) can easily be added in accordance with the principles of the invention. In one example implementation, the element insertion routine can be used to automatically introduce new elements into a schema without having to rewrite the schema by hand.

```
     <insert path="/" name="comment_def">
          <xsd_element type="xsd:string" name="comment">
          </xsd_element>
      </insert>
```

[0048] For example, the output XML file below is an example of an output default constraint generated using the initial XML file shown above in Table 1.

```
<?XML version="1.0" encoding="UTF-8"?>
<root>
      <namespaces>
```

```
          <namespace value="http://java.sun.com/XML/ns/jaxb"
name="XMLns:jaxb">
      </namespace>
    </namespaces>
<constraints>
   <constraint type="attr" name="orderDate" path="/purchaseOrder">
      <datatype type="xsd:date">
      </datatype>
   </constraint>
   <constraint type="elem" name="productName"
path="/purchaseOrder/items/item/productName">
           <datatype type="xsd:string">
           </datatype>
        </constraint>
        <constraint type="elem" name="quantity"
path="/purchaseOrder/items/item/quantity">
          <datatype type="xsd:integer">
          </datatype>
       </constraint>
       <constraint type="elem" name="USPrice"
path="/purchaseOrder/items/item/USPrice">
             <datatype type="xsd:decimal">
             </datatype>
        </constraint>
      </constraints>
      <inserts>
          <insert path="/" name="comment_def">
               <xsd_element type="xsd:string" name="comment">
               </xsd_element>
          </insert>
     </inserts>
</root>
```

[0049] Fig. 7 describes one embodiment of, for example, an XSLT style sheet that can be used to execute the process illustrated in Fig. 4 or the operate the default constraint generator (e.g., 214) operating to accomplish operations 311, 313 of Fig. 3.

[0050] The following discussion describes a merger (e.g., 216) and an associated mode of operation. In general, a merger reads an input set of default constraint information (e.g., 214o) and an input set of externally supplied constraints (e.g., 219) and generates one of a merged set of constraints. Alternatively, the merger can select the default set of constraints or the externally supplied set of constraints for output to a schema generator (e.g., 218). In one implementation, the externally supplied constraints provide the primary constraint information with the default constraint information providing ancillary information.

[0051] Fig. 8 describes a simplified flow diagram illustrating an example mode of operation for one embodiment of an XML schema generator (e.g., 218) operat

ing to accomplish operation 325 of Fig. 3.

[0052] The following is a brief simplified description of schema generation in accordance with the principles of the invention, for example, using a schema generator (e.g., 218). The generated data framework 212o (e.g., in the form of an XML file) is received from the structural analyzer 212 (Step 801). At the same time, or at a different time, the final data constraint file (e.g., in the form of an XML file) can be received from the merger 216 (Step 803). The two files are read (Step 805) and a "root element" is generated for a schema (xsd file) (Step 807). Each of the elements and associated child elements are identified (Step 809). Each element (and child element) is processed to generate an XML element (or element set) that contains an associated set of elements (Step 811) to define an associate xsd element. If child elements contain further children (not siblings), such elements are "complex elements" and processed as such. For example, an xsd element is generated identifying the element having children as type complex elements (e.g., "xsd:complexType"). A fully constructed xsd (schema) file is then generated (Step 813).

[0053] Such processing (Step 811) typically involves scanning each element (and child element) to define and generate an XML schema element.

[0054] In a typical implementation, generating for example a complex element an accompanying "complexType" set (including a type declaration and complexType definition) is generated. In illustrative example the complex set of elements can include a declaration:

```
    <xsd:element name="purchaseOrder" type="_purchaseOrder"/>
```

and can include a complexType definition:

```
        <xsd:complexType name="_purchaseOrder">
            <xsd:sequence>
                <xsd:element name="items" type="_items"/>
            </xsd:sequence>
             <xsd:attribute name="orderDate" type="xsd:date" use="required"/>
  </xsd:complexType>
```

[0055] In a simple element, a corresponding simple XML schema can be generated. In the following example (having child count = 0) a declaration is defined. For example, <xsd:element name="productName" type="XYZ"/> where XYZ - could be of type (e.g., string; integer; decimal; boolean; date; time; etc.). Wherein "XYZ" is selected from an externally provided constraint document or from a generated default constraint XML document. A preference can be given to the externally provided constraint XML document types if they have been supplied. Additionally, if the element includes attributes, it can be configured, for example as follows:

```
    <xs:element name="item">
       <xs:complexType>
         <xs:attribute name="partNum" type="xs:string"/>
       </xs:complexType>
    </xs:element>
```

[0056] It should be noted, that the examples provided above are intended as simplified examples used to illustrate certain principles of the invention. As such they are not intended to confine the scope or implementations of the invention. As is known to those having ordinary skill in the art, many other related implementation details can be implemented to accomplish the embodiments of the invention.

[0057] One example of a fully operative embodiment for accomplishing such schema generation is depicted by the stylesheet of Fig. 9.

[0058] It should be noted that all the processes, methods, and systems described herein can be implemented in many different kinds of tools. Examples, include but are not limited to command-line tools and servlets. Also, the principles of the invention can be applied to ANT assembly language programming (e.g., Ant 3.0.1) and other build tools. Also, the inventor contemplates the application of the principles of the present invention to JAVA Studio IDE NBM modules. The processes described herein are well suited to implementation in computer readable medium implementations. Also, the principles of the present invention are well suited to network based computer systems. For example, an XML document on one computer provided over the internet to a remote computer can be converted to an XML schema on the remote computer for remote use or for return and use on the sending computer.

[0059] Figs. 10A and 10B illustrate an example of a multi-processor computer system 1000 that may be used in accordance with the invention. Fig. 10A shows a computer system 1000 that includes a display 1002, screen 1004, cabinet 1006, keyboard 1010, and mouse 1012. Mouse 1012 may have one or more buttons for interacting with a graphical user interface. Cabinet 1006 can house a CD-ROM drive 1008, system memory and a hard drive (see Fig. 10B) which may be utilized to store and retrieve software programs incorporating computer code that implements the invention, data for use with the invention, and the like. Although CD-ROM 1008 is shown as an exemplary computer readable storage medium, other computer readable storage media including floppy disk, DVD, tape, memory sticks, flash memory, system memory, and hard drive may be utilized. Additionally, a data signal embodied in a carrier wave (e.g., in a network including the Internet) may be the computer readable storage medium. In one implementation, an operating system for the computer system 1000 is provided in the system memory, the hard drive, the CD-ROM 1008 or other computer readable storage medium and serves to incorporate the computer code that implements the invention (e.g., MMU system). It is to be remembered that the operating system is configured so it controls all of the processors of the system. It should be noted that other devices (e.g., printers, scanners, etc.) may be present in the computer system 1000.

[0060] Fig. 10B shows a system block diagram of computer system 1000 used to execute the software of an embodiment of the invention. The computer system 1000 includes monitor 1004, keyboard 1010, and mouse 1012. Computer system 1000 further includes subsystems, such as a plurality of central processors (CPU's) 1022 (including cache memory resources), system memory 1024, fixed storage 1026 (e.g., hard drive), removable storage 1014 (e.g., CD-ROM drive), display adapter,

sound card and speakers 1030, and network interface 1040. The central processors 1051, for example, can execute computer program code (e.g., an operating system) to implement the invention. An operating system is normally (but not necessarily) resident in the system memory 1024 during its execution. Other computer systems suitable for use with the invention may include additional or fewer subsystems. Importantly, the principles of the invention can specifically be implemented on networked computer systems having many individual computers. Such networked systems can include local area networks (LAN's) or a wide area network (WAN's). Particularly, the inventors contemplate computer systems networked together using the Internet. Additionally, an example of a LAN is a private network used by a mid-sized company with a building complex. Publicly accessible WAN's include the Internet, cellular telephone network, satellite systems and plain-old-telephone systems (POTS). Examples of private WAN's include those used by multi-national corporations for their internal information system needs. The network may also be a combination of private and/or public LANs and/or WANs.

[0061] The system bus architecture of computer system 1000 is represented by arrows 1020. However, these arrows are illustrative of any interconnection scheme serving to link the subsystems. For example, a local bus could be utilized to connect the central processor to the system memory and display adapter. Computer system 1000 shown in Fig. 10B is but an example of a computer system suitable for use with the invention. Other computer architectures having different configurations of subsystems may also be utilized.

[0062] The invention can use a combination of hardware and software components. The software can be embodied as computer readable code (or computer program code) on a computer readable medium. The computer readable medium is any data storage device that can store data which can thereafter be read by a computer system. Examples of the computer readable medium include read-only memory, random-access memory, CD-ROMs, magnetic tape, and optical data storage devices. The computer readable medium can also be distributed over a network coupled computer systems so that the computer readable code is stored and executed in a distributed fashion.

[0063] The advantages of the invention are numerous. Different embodiments or implementations may have one or more of the following advantages. One advantage of the invention is that it can be used in a networked computing environment, to include, but not limited to a web-based network. XML documents can be remotely accessed to generate schema. Additionally, the components of the inventive system can be operated in a distributed fashion if desired. For example, an XML structure analyzer can be located on one computer and a default constraint generator can be located on another networked computer. In another implementation, an XML documents can be provided from one networked computer to a complete module located on yet another networked computer.

[0064] The many features and advantages of the present invention are apparent from the written description, and thus, it is intended by the appended claims to cover all such features and advantages of the invention. Further, since numerous modifications and changes will readily occur to those skilled in the art, it is not desired to limit the invention to the exact construction and operation

as illustrated and described. Hence, all suitable modifications and equivalents may be resorted to as falling within the scope of the invention.

3. BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a simplified block diagram illustrating an example application program suitable for executing an XML schema generated in accordance with the principles of the invention.

Fig. 2 is a simplified block diagram depicting one implementation of a system embodiment for automatically generating schema in accordance with the principles of the invention.

Fig. 3 is a flow diagram that describes one method embodiment of the invention.

Fig. 4 is a flow diagram that describes one method embodiment for analyzing the structure of an XML file in accordance with one embodiment of the present invention.

Fig. 5.1-5.2 depicts a method embodiment enabling a structure analyzer implemented as an XSLT spreadsheet in accordance with the principles of the invention.

Fig. 6 is a flow diagram that describes one method embodiment for generating a set of default constraints from an initial XML document in accordance with one embodiment of the present invention.

Fig. 7.1-7.3 depicts a method embodiment enabling default constraint generation implemented as an XSLT spreadsheet in accordance with the principles of the invention.

Fig. 8 describes a simplified flow diagram illustrating an example mode of operation for one embodiment of XML schema generation in accordance with the principles of the invention.

Fig. 9.1-9.5 depicts a method embodiment enabling XML schema generation implemented as an XSLT spreadsheet in accordance with the principles of the invention.

Figs. 10A and 10B illustrate an example of a computer system that may be used in accordance with the invention.

Claim 1. A method generating XML schema comprising:

providing an initial XML document that includes raw XML data comprising data items arranged in XML data elements;

analyzing the XML document to identify XML data structures;

generating a data framework that corresponds to the format of the data structures in the XML document;

analyzing the data items from the initial XML document;

determining data constraints based on the data items;

generating an XML schema based on the data framework generated and the data constraints determined from the raw XML data.

Claim 2. The method of Claim 1 wherein determining the data constraints based on the data items includes determining type constraints based on the types of the data items in the initial XML document.

Claim 3. The method of Claim 1 wherein determining the data constraints based on the data items includes determining attribute constraints based on the attributes of the data items in the initial XML document.

Claim 4. The method of Claim 1 wherein the method can be implemented to chang e namespaces for an XML schema.

Claim 5. The method of Claim 1 wherein the method can be implemented to chang e data types for an XML schema.

Claim 6. The method of Claim 1 wherein the method can be implemented to add o r remove data elements from an XML schema.

Claim 7. The method of Claim 1 wherein determining data constraints includes receiving externally supplied data constraints and
wherein generating the schema further includes using one of: (i) the data cons traints determined from the data in the initial XML document or (ii) using the e xternally supplied data constraints to generate the XML schema.

Claim 8. The method of Claim 1 wherein determining data constraints includes receiving externally supplied data constraints and
wherein generating the schema further includes using at least one of: (i) the data constraints determined from the data in the initial XML document and (ii) u sing the externally supplied data constraints to generate the XML schema.

Claim 9. The method of Claim 1 wherein determining data constraints includes receiving externally supplied data constraints and
wherein generating the schema further includes merging portions of the data co nstraints determined from the data in the initial XML document and portions of t he externally supplied data constraints.

Claim 10. A computer program product embodied on a computer readable media i ncluding computer program code for generating XML schema, the computer program p roduct including:
computer program code instructions for receiving an initial XML document that that includes raw XML data comprising data items arranged in XML data structures ;
computer program code instructions for analyzing the XML document to identify the XML data structures;
computer program code instructions for generating a data framework associated with the format of the data structures in the XML document;
computer program code instructions for analyzing the data items from the initi al XML document and determining XML data constraints based on the data items; and
computer program code instructions for generating an XML schema based on the d ata framework generated and the XML data constraints determined from the data it ems.

Claim 11. The method of Claim 10 wherein the computer program code instructio ns for determining data constraints includes receiving externally supplied data constraints and
wherein the computer program code instructions for generating the schema furth er includes using one of: (i) the data constraints determined from the data item

s in the initial XML document or (ii) using the externally supplied data constra ints to generate the XML schema.

Claim 12. The method of Claim 10 wherein the computer program code instructio ns for determining data constraints includes computer program code instructions for receiving externally supplied data constraints and

further computer program code instructions for using at least one of: (i) the data constraints determined from the data in the initial XML document and (ii) u sing the externally supplied data constraints to generate the XML schema.

Claim 13. The method of Claim 10 wherein the computer program code instructio ns for determining data constraints includes computer program code instructions for receiving externally supplied data constraints and

wherein the computer program code instructions for generating the schema furth er includes instructions for merging portions of the data constraints determined from the data in the initial XML document and portions of the externally suppli ed data constraints.

Claim 14. A computer system comprising:

at least one central processing unit (CPU), memory, and user interface in comb ination configured to include:

XML structure analyzer for analyzing a received initial XML document that incl udes raw XML data comprising data items arranged in XML data elements, wherein s aid analyzing includes identifying the XML data elements of the initial XML docu ment and generating a data framework associated with a data structure for the XM L data elements of the initial XML document;

default constraint generator for analyzing the raw XML data from the initial X ML document and determining XML data constraints for data items based on the XML raw data;

data constraint merger for receiving at least one of the XML data constraints from the data constraint generator and a set of externally supplied data constra ints and for outputting a final data constraint file; and

XML schema generator for receiving the data structures from the XML structure analyzer and final data constraint file from the data constraint merger and gene rating an XML schema associated with said data structures and final data constra int file.

Claim 15. The computer system of Claim 14 wherein the data constraint merger receives both the XML data constraints from the data constraint generator and th e set of externally supplied data constraints and merges both sets of data const raints into the final data constraint file.

Claim 16. The computer system of Claim 14 wherein the data constraint merger receives both the XML data constraints from the data constraint generator and th e set of externally supplied data constraints and selects one of the XML data co nstraints from the data constraint generator and the set of externally supplied data constraints for output as the final data constraint file.

Claim 17. A computer module for automatically generating XML schema, the mod ule comprising:

XML structure analyzer for analyzing a received initial XML document that includes raw XML data comprising data items arranged in XML data elements, wherein said analyzing includes identifying the XML data elements of the initial XML document and generating a data framework associated with a data structure for the XML data elements of the initial XML document;

default constraint generator for analyzing the raw XML data from the initial XML document and determining XML data constraints for data items based on the XML raw data;

data constraint merger for receiving at least one of the XML data constraints from the data constraint generator and a set of externally supplied data constraints and for outputting a final data constraint file; and

XML schema generator for receiving the data structures from the XML structure analyzer and final data constraint file from the data constraint merger and generating an XML schema associated with said data structures and final data constraint file.

Claim 18. The computer module of Claim 17 wherein the data constraint merger receives both the XML data constraints from the data constraint generator and the set of externally supplied data constraints and merges both sets of data constraints into the final data constraint file.

Claim 19. The computer module of Claim 17 wherein the data constraint merger receives both the XML data constraints from the data constraint generator and the set of externally supplied data constraints and selects one of the XML data constraints from the data constraint generator and the set of externally supplied data constraints for output as the final data constraint file.

Claim 20. A computer module for automatically generating XML schema, the module comprising:

XML structure analyzer for analyzing a received initial XML document that includes raw XML data comprising data items arranged in XML data elements, wherein said analyzing includes identifying the XML data elements of the initial XML document and generating a data framework associated with a data structure for the XML data elements of the initial XML document;

default constraint generator for analyzing the raw XML data from the initial XML document and determining XML data constraints for data items based on the XML raw data; and

XML schema generator for receiving the data structures from the XML structure analyzer and default data constraint file from the default constraint generator and generating therefrom an XML schema associated with said data structures and final data constraint file.

1. ABSTRACT

Techniques, systems and apparatus for automatically generating schema using an initial documents constructed in an XML compatible format are disclosed. A method involves providing an initial XML document that and analyzing the XML document to identify the XML data structures in the document and generating a data framework that corresponds to the data structure of the XML document. The data items of the initial XML document are analyzed to determine data constraints based on the data items of the initial XML. Schema are then generated based on the data framework generated and the data constraints determined from the raw xml dat

a. These principles can be implemented as software operating on a computer system, as a computer module, as a computer program product and as a series of related devices and products.

2. REPRESENTATIVE DRAWING

Fig. 3

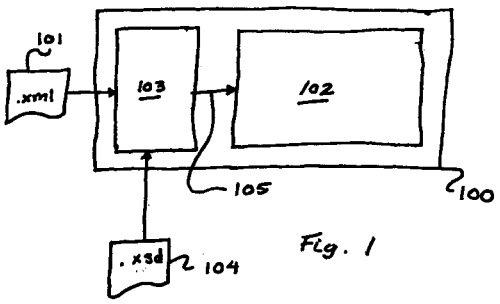

Fig. 1

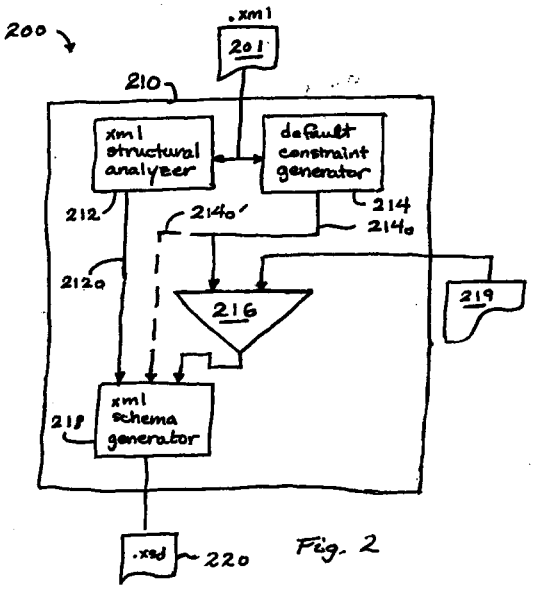

Fig. 2

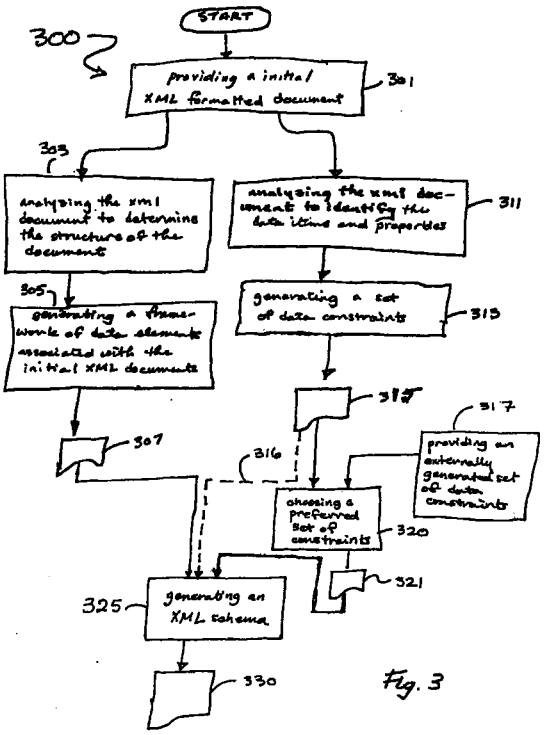

Fig. 3

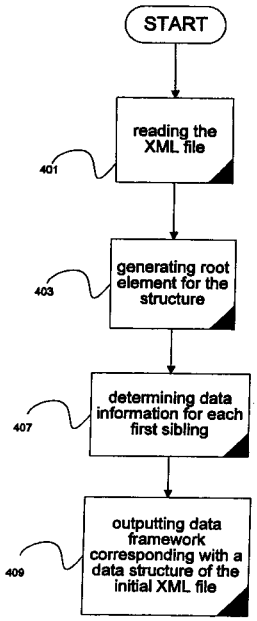

Fig. 4

```
Structure Analyzer XML Stylesheet
-----------------------------------------------------------
<xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transform version="1.0">
   <xsl:output method="xml" indent="yes"/>

   <xsl:template match="/">
     <xsl:call-template name="process-all-category-children">
        <xsl:with-param name="children" select="*"/>
     </xsl:call-template>
   </xsl:template>

   <xsl:template name="process-all-category-children">
     <xsl:param name="children"/>

     <xsl:if test="count($children) > 0">
       <xsl:variable name="first-child"  select="local-name($children[1])"/>

       <xsl:call-template name="process-like-siblings">
          <xsl:with-param name="siblings" select="$children[local-name(.) = $first-child]"/>
       </xsl:call-template>

       <xsl:call-template name="process-all-category-children">
          <xsl:with-param name="children"  select="$children[local-name(.) != $first-child]"/>
       </xsl:call-template>
     </xsl:if>
   </xsl:template>

<!-- Pickup only the first sibling from the set of children for a node-->
   <xsl:template name="process-like-siblings">
     <xsl:param name="siblings"/>

     <xsl:variable name="category-name" select="$siblings[1]/@name"/>
     <xsl:variable name="el-name" select="local-name($siblings[1])"/>

     <xsl:element name="{$el-name}">

       <xsl:for-each select="$siblings[1]/@*">
          <xsl:variable name="attr-name" select="."/>
          <xsl:attribute name="{local-name()}"><xsl:value-of select="$attr-
name"/></xsl:attribute>
          <xsl:attribute name="XSG_attrcount_{local-name()}"><xsl:value-of
select="count($siblings/@*)"/></xsl:attribute>
       </xsl:for-each>

       <xsl:variable name="count"
select="count($siblings[1]/../*[local-name() =
local-name($siblings[1])])"/>
       <xsl:variable name="child_count" select="count($siblings/*)"/>

       <xsl:if test="$child_count = 0">
```

Fig. 5.1

```
          <xsl:attribute name="XSG_val"><xsl:value-of
select="$siblings[1]"/></xsl:attribute>
       </xsl:if>

       <xsl:attribute name="XSG_count"><xsl:value-of
select="$count"/></xsl:attribute>

       <xsl:call-template name="process-all-category-children">
          <xsl:with-param name="children" select="$siblings/*"/>
       </xsl:call-template>

     </xsl:element>
   </xsl:template>

</xsl:stylesheet>
```

Fig. 5.2

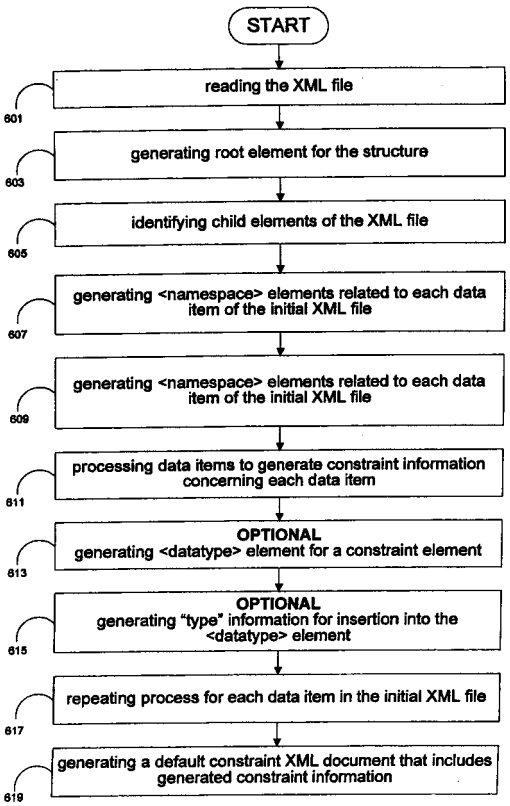

Fig. 6

```
Example Default Constraint Generator XML Style Sheet
-----------------------------------------------------
> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>       xmlns:java="http://xml.apache.org/xalan/java"
>  version="1.0">
>  <xsl:output method="xml" indent="yes"/>
>
> <!-- root node-->
>  <xsl:template match="/">
>  <root>
>    <namespaces>
>      <namespace name="xmlns:jaxb"
> value="http://java.sun.com/xml/ns/jaxb"/>
>    </namespaces>
>
>      <constraints>
>     <xsl:for-each select="*">
>
>      <xsl:variable name="el-name" select="local-name(.)"/>
>      <xsl:variable name="new-path" select="concat('/', $el-name)"/>
>
>      <xsl:variable name="elem-val" select="./@XSG_val"/>
>      <xsl:if test="string-length($elem-val) > 0">
>        <xsl:call-template name="gen-constraint">
>       <xsl:with-param name="name" select="local-name(.)"/>
>       <xsl:with-param name="path" select="$new-path"/>
>       <xsl:with-param name="type" select="'elem'"/>
>       <xsl:with-param name="val" select="$elem-val"/>
>        </xsl:call-template>
>       </xsl:if>
>
>       <xsl:for-each select="./@*">
> <!-- do not process if node has XSG attributes-->
>        <xsl:if test="not(contains(local-name(.), 'XSG_'))">
>        <xsl:variable name="attr-val" select="."/>
>         <xsl:call-template name="gen-constraint">
>        <xsl:with-param name="name" select="local-name(.)"/>
>        <xsl:with-param name="path" select="$new-path"/>
>        <xsl:with-param name="type" select="'attr'"/>
>        <xsl:with-param name="val" select="$attr-val"/>
>         </xsl:call-template>
>        </xsl:if>
>       </xsl:for-each>
>       <xsl:call-template name="process-all-children">
>      <xsl:with-param name="children" select="./*"/>
>      <xsl:with-param name="path" select="$new-path"/>
>       </xsl:call-template>
>      </xsl:for-each>
>      </constraints>
```

Fig. 7.1

```
>      <inserts>
>        <insert name="comment_def" path="/">
>       <xsd_element name="comment" type="xsd:string"/>

>        </insert>
>      </inserts>
>
>    </root>
>  </xsl:template>
>
> <!-- process all children -->
>  <xsl:template name="process-all-children">
>    <xsl:param name="children"/>
>    <xsl:param name="path"/>
> <!-- process if node has children-->
>    <xsl:if test="count($children) > 0">
> <!-- create a xsd:complexType tag-->
>
> <!-- call template to process other siblings-->
>      <xsl:for-each select="$children">
>       <xsl:variable name="el-name" select="local-name(.)"/>
>       <xsl:variable name="new-path" select="concat($path, '/',
> $el-name)"/>
>
>       <xsl:variable name="elem-val" select="./@XSG_val"/>
>       <xsl:if test="string-length($elem-val) > 0">
>         <xsl:call-template name="gen-constraint">
>       <xsl:with-param name="name" select="local-name(.)"/>
>       <xsl:with-param name="path" select="$new-path"/>
>       <xsl:with-param name="type" select="'elem'"/>
>       <xsl:with-param name="val" select="$elem-val"/>
>        </xsl:call-template>
>       </xsl:if>
>
>       <xsl:call-template name="process-all-children">
>      <xsl:with-param name="children" select="./*"/>
>      <xsl:with-param name="path" select="$new-path"/>
>       </xsl:call-template>
>
>
>      </xsl:for-each>
> <!-- end call template to process other siblings-->
>    </xsl:if>
> <!-- end process if node has children-->
>  </xsl:template>
```

Fig. 7.2

```
>  <xsl:template name="gen-constraint">
>    <xsl:param name="name"/>
>    <xsl:param name="path"/>
>    <xsl:param name="type"/>
>    <xsl:param name="val"/>
>    <xsl:variable name="datatype"><xsl:value-of
> select="java:DataType.getType($name,$val)"/></xsl:variable>
>        <xsl:element name="constraint">
>              <xsl:attribute name="type"><xsl:value-of
> select="$type"/></xsl:attribute>
>              <xsl:attribute name="name"><xsl:value-of
> select="$name"/></xsl:attribute>
>              <xsl:attribute name="path"><xsl:value-of
> select="$path"/></xsl:attribute>
>              <datatype>
>              <xsl:attribute name="type"><xsl:value-of
> select="$datatype"/></xsl:attribute>
>              </datatype>
>        </xsl:element>
>  </xsl:template>
>
> </xsl:stylesheet>
```

Fig. 7.3

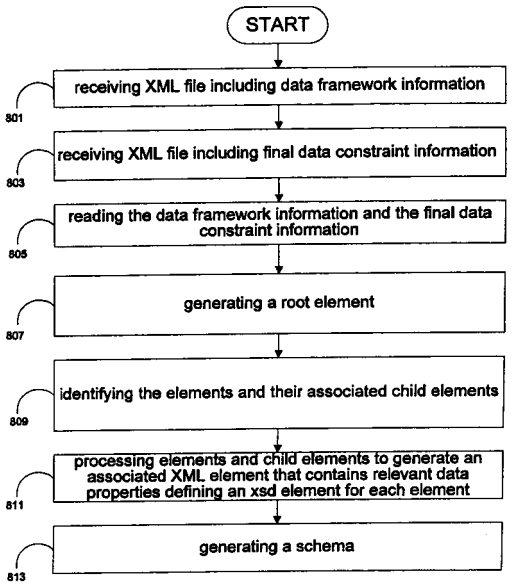

Fig. 8

```
Process for Generating Schema
> ----------------------------------------------------------
> <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>       xmlns:java="http://xml.apache.org/xalan/java"
>  version="1.0">
> <xsl:output method="xml" indent="yes"/>
> <xsl:param name="constraint-document" />
> <!-- root node-->
>  <xsl:template match="/">
>
>     <xsl:element name="xsd_schema">
>     <xsl:attribute
> name="xmlns_xsd">http://www.w3.org/2001/XMLSchema</xsl:attribute>
>
>     <xsl:variable name="namespaces"
> select="document(normalize-space($constraint-document))/root/namespaces/namespace"/>
>
>     <xsl:variable name="xmlns_jaxb" select="$namespaces[@name =
> 'xmlns_jaxb']/@value"/>
>     <xsl:if test="string-length($xmlns_jaxb) > 0">
>      <xsl:attribute name="xmlns_jaxb"><xsl:value-of
> select="$xmlns_jaxb"/></xsl:attribute>
>     </xsl:if>
>
>     <xsl:variable name="xmlns_xjc" select="$namespaces[@name =
> 'xmlns_xjc']/@value"/>
>     <xsl:if test="string-length($xmlns_xjc) > 0">
>      <xsl:attribute name="xmlns_xjc"><xsl:value-of
> select="$xmlns_xjc"/></xsl:attribute>
>     </xsl:if>
>
>     <xsl:variable name="jaxb_version" select="$namespaces[@name =
> 'jaxb_version']/@value"/>
>     <xsl:if test="string-length($jaxb_version) > 0">
>      <xsl:attribute name="jaxb_version"><xsl:value-of
> select="$jaxb_version"/></xsl:attribute>
>     </xsl:if>
>
>     <xsl:variable name="jaxb_extensionBindingPrefixes"
> select="$namespaces[@name = 'jaxb_extensionBindingPrefixes']/@value"/>
>     <xsl:if test="string-length($jaxb_extensionBindingPrefixes) > 0">
>      <xsl:attribute
> name="jaxb_extensionBindingPrefixes"><xsl:value-of
> select="$jaxb_extensionBindingPrefixes"/></xsl:attribute>
>     </xsl:if>
>
>     <xsl:copy-of
> select="document(normalize-space($constraint-document))/root/inserts/insert[@path
> = '/']/*"/>
```

Fig. 9.1

```
>     <xsl:for-each select="*">
>      <xsl:element name="xsd_element">
>       <xsl:attribute name="name"><xsl:value-of
> select="local-name(.)"/></xsl:attribute>
>       <xsl:if test="count(./*) > 0">
>        <xsl:attribute name="type">XSG_<xsl:value-of
> select="local-name(.)"/></xsl:attribute>
>       </xsl:if>
>       <xsl:if test="count(./*) = 0">
>        <xsl:attribute name="type">xsd:string</xsl:attribute>
>       </xsl:if>
>      </xsl:element>
>     </xsl:for-each>
>
>     <xsl:for-each select="*">
>      <xsl:variable name="el-name" select="local-name(.)"/>
>      <xsl:variable name="new-path" select="concat('/', $el-name)"/>
>      <xsl:call-template name="process-all-children">
>     <xsl:with-param name="children" select="./*"/>
>     <xsl:with-param name="path" select="$new-path"/>
>      </xsl:call-template>
>     </xsl:for-each>
>
>          <xsl:variable name="attr-constraints"
> select="document(normalize-space($constraint-
document))/root/constraints/constraint[@type='attr']"/>
>          <xsl:variable name="attr-constraints-refs"
> select="$attr-constraints/datatype[@type='ref']"/>
>          <xsl:copy-of select="$attr-constraints-refs/*"/>
>
> </xsl:element>
>
>  </xsl:template>
>
> <!-- process all children -->
>  <xsl:template name="process-all-children">
>     <xsl:param name="children"/>
>     <xsl:param name="path"/>
> <!-- process if node has children-->
>     <xsl:if test="count($children) > 0">
> <!-- create a xsd:complexType tag-->
>     <xsl:element name="xsd_complexType">
>      <xsl:attribute name="name">XSG_<xsl:value-of
> select="local-name($children[1]/..)"/></xsl:attribute>
>      <xsl:element name="xsd_sequence">
> <!-- call template to process like siblings-->
>       <xsl:call-template name="process-like-siblings">
>      <xsl:with-param name="siblings" select="$children"/>
>      <xsl:with-param name="path" select="$path"/>
```

Fig. 9.2

```
>       </xsl:call-template>
>       <xsl:copy-of
> select="document(normalize-space($constraint-document))/root/inserts/insert[@path
> = $path]/*"/>
>      </xsl:element>
> <!-- create attributes for each xsd:complexType tag-->
>     <xsl:for-each select="./@*">
>      <xsl:variable name="attr_name" select="local-name(.)"/>
>      <xsl:variable name="attr_val" select="."/>
>      <xsl:if test="not(contains($attr_name,'XSG_'))">
>       <xsl:element name="xsd_attribute">
>       <xsl:attribute name="name"><xsl:value-of
> select="$attr_name"/></xsl:attribute>
>       <xsl:variable name="attr_type" select="xsd_date"/>
>       <xsl:variable name="attr-constraints"
> select="document(normalize-space($constraint-
document))/root/constraints/constraint[@type='attr']"/>
>       <xsl:variable name="constraint-path"
> select="$attr-constraints[@path=string($path)]"/>
>       <xsl:variable name="constraint-path-val"
> select="$constraint-path/@path"/>
>       <xsl:variable name="constraint-attr-name"
> select="$constraint-path/@name"/>
>       <xsl:variable name="datatype"
> select="$constraint-path/datatype/@type"/>
>       <xsl:variable name="dataname"
> select="$constraint-path/datatype/@name"/>
>       <xsl:if test="($attr_name = $constraint-attr-name)">
>        <xsl:attribute name="type"><xsl:value-of
> select="$datatype"/></xsl:attribute>
>       </xsl:if>
>
>       <xsl:if test="$datatype = 'ref'">
>        <xsl:attribute name="type"><xsl:value-of
> select="$dataname"/></xsl:attribute>
>       </xsl:if>
>
>
>       <xsl:if test="not($attr_name = $constraint-attr-name)">
>        <xsl:attribute name="type"><xsl:value-of
> select="java:DataType.getType($attr_name,$attr_val)"/></xsl:attribute>
>       </xsl:if>
>
```

Fig. 9.3

```
      <xsl:if test="string-length('$attr_val') > 0">
>       <xsl:attribute name="use">required</xsl:attribute>
>       </xsl:if>
>       </xsl:element>
>      </xsl:if>
>     </xsl:for-each>
>    </xsl:element>
> <!-- end create xsd:complexType tag-->
>
> <!-- call template to process other siblings-->
>      <xsl:for-each select="$children">
>       <xsl:variable name="el-name" select="local-name(.)"/>
>       <xsl:variable name="new-path" select="concat($path, '/',
> $el-name)"/>
>       <xsl:call-template name="process-all-children">
>      <xsl:with-param name="children" select="./*"/>
>      <xsl:with-param name="path" select="$new-path"/>
>       </xsl:call-template>
>      </xsl:for-each>
> <!-- end call template to process other siblings-->
>     </xsl:if>
> <!-- end process if node has children-->
>  </xsl:template>
>
> <!-- process like siblings-->
>  <xsl:template name="process-like-siblings">
>     <xsl:param name="siblings"/>
>     <xsl:param name="path"/>
>     <xsl:variable name="category-name" select="$siblings[1]/@name"/>
> <!-- create xsd:element tag for each siblings-->
>     <xsl:for-each select="$siblings">
>      <xsl:variable name="el-name" select="local-name(.)"/>
>      <xsl:variable name="new-path" select="concat($path, '/',
> $el-name)"/>
>      <xsl:element name="xsd_element">
>       <xsl:attribute name="name"><xsl:value-of
> select="local-name(.)"/></xsl:attribute>
>       <xsl:if test="count(./*) > 0">
>        <xsl:attribute name="type">XSG_<xsl:value-of
> select="local-name(.)"/></xsl:attribute>
>        <xsl:variable name="count" select="$siblings[1]/@XSG_count"/>
>        <xsl:if test="number($count) > 1">
>         <xsl:attribute name="minOccurs">1</xsl:attribute>
>         <xsl:attribute name="maxOccurs">unbounded</xsl:attribute>
>        </xsl:if>
>       </xsl:if>
>       <xsl:variable name="elem-constraints"
```

Fig. 9.4

```
> select="document(normalize-space($constraint-
document))/root/constraints/constraint[@type='elem']"/>
>       <xsl:variable name="constraint-path"
> select="$elem-constraints[@path=string($new-path)]"/>
>       <xsl:variable name="constraint-path-val"
> select="$constraint-path/@path"/>
>       <xsl:variable name="constraint-attr-name"
> select="$constraint-path/@name"/>
>
>       <xsl:if test="count(./*) = 0">
>         <xsl:if test="$constraint-path-val = $new-path">
>           <xsl:variable name="datatype"
> select="$constraint-path/datatype/@type"/>
>
>           <xsl:if test="$datatype = 'local'">
>             <!--<xsl:attribute name="type"><xsl:value-of
> select='$datatype'/></xsl:attribute>-->
>             <xsl:copy-of select="$constraint-path/datatype/*"/>
>           </xsl:if>
>
>           <xsl:if test="starts-with($datatype, 'xsd:')">
>             <xsl:attribute name="type"><xsl:value-of
> select='$datatype'/></xsl:attribute>
>
>         <xsl:variable name="xsg_val" select="./@XSG_val"/>
>           <xsl:if test="not($constraint-path-val = $new-path)">
>           <xsl:attribute name="type"><xsl:value-of
> select="java:DataType.getType($el-name,$xsg_val)"/></xsl:attribute>
>           </xsl:if>
>         </xsl:if>
>       </xsl:if>
>     </xsl:element>
>   </xsl:for-each>
> <!-- end create xsd:element tag for each siblings-->
>   </xsl:template>
>
> </xsl:stylesheet>
```

Fig. 9.5

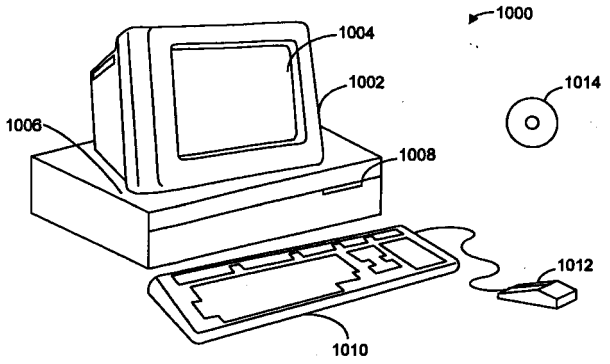

Fig. 10A

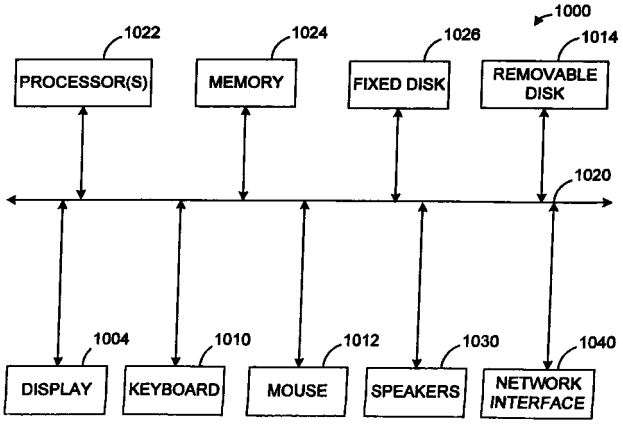

Fig. 10B