(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2005-63243
(P2005-63243A)
(43)公開日　平成17年3月10日(2005.3.10)

(51)Int.Cl.[7]	F I		テーマコード（参考）
G06F 3/12	G06F 3/12	Z	5B009
G06F 17/21	G06F 17/21	566A	5B021
	G06F 17/21	570L	

審査請求 未請求　請求項の数 21　ＯＬ　（全 13 頁）

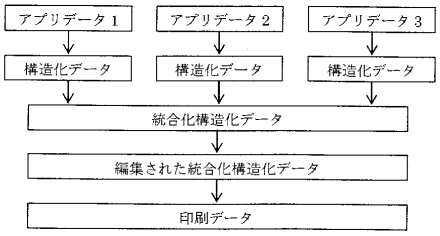
(21)出願番号　特願2003-293942(P2003-293942)
(22)出願日　平成15年8月15日(2003.8.15)

(71)出願人　000006747
株式会社リコー
東京都大田区中馬込１丁目３番６号
(74)代理人　100084250
弁理士　丸山　隆夫
(72)発明者　小出　雅巳
東京都大田区中馬込１丁目３番６号
株式会社リコー内
(72)発明者　鈴木　明
東京都大田区中馬込１丁目３番６号
株式会社リコー内

Ｆターム(参考)　5B009 NA06　RC01　SA13
5B021 AA01　CC06　CC07

(54)【発明の名称】情報処理装置、プログラム及びコンピュータ読み取り可能な記録媒体

(57)【要約】
【課題】　各種アプリケーションデータを統一的に扱うことができる統合されたデータを生成しこの統合されたデータから印刷データを生成する。
【解決手段】　各アプリケーションのデータ（アプリデータ１，２，３）は、「データ構造化手段」によりそれぞれデータが構造化されて構造化データとなる。次に、各アプリケーションの構造化データは「データ統合構造化手段」により一つに統合されて統合化構造化データとなる。この統合化構造化データは、「データ編集手段」により編集され、この編集された統合化構造化データから「印刷データ生成手段」により印刷データが生成される。
【選択図】　図２



【特許請求の範囲】
【請求項１】
アプリケーションが生成したドキュメントデータに応じて印刷装置が解釈可能な印刷データを生成する情報処理装置において、
複数のアプリケーションが生成した複数のドキュメントデータを所定のフォーマットで構造化して複数の構造化データを得るデータ構造化手段と、
各構造化データを統合して構造化し直す統合構造化手段と、
統合したデータから印刷データを生成する印刷データ生成手段とを有することを特徴とする情報処理装置。
【請求項２】
前記所定のフォーマットは、ＸＭＬ（eXtensible Markup Language）などのマークアップ言語で記述することを特徴とする請求項１に記載の情報処理装置。
【請求項３】
前記統合したデータのレイアウトを編集するデータ編集手段を有し、データ編集手段による編集に伴い、レイアウトおよび構造を変更することを特徴とする請求項１に記載の情報処理装置。
【請求項４】
前記統合したデータの構造を編集する構造編集手段を有し、構造編集手段による編集に伴い、構造および統合したデータのレイアウトを変更することを特徴とする請求項１に記載の情報処理装置。
【請求項５】
前記印刷データ生成手段は、前記統合したデータの構造の印刷データを生成することを特徴とする請求項１に記載の情報処理装置。
【請求項６】
前記統合したデータの構造の一部分を選択する構造選択手段を有し、前記印刷データ生成手段は、構造選択手段により選択された部分の構造の印刷データを生成することを特徴とする請求項１に記載の情報処理装置。
【請求項７】
前記印刷データ生成手段は、前記構造選択手段によって選択された部分の構造に相当するレイアウトの印刷データを生成することを特徴とする請求項６に記載の情報処理装置。
【請求項８】
前記統合したデータのレイアウトの一部を選択するレイアウト選択手段を有し、前記印刷データ生成手段は、レイアウト選択手段により選択された部分のデータの構造の印刷データを生成することを特徴とする請求項１に記載の情報処理装置。
【請求項９】
前記統合化したデータをカプセル化文書として出力するカプセル化文書出力手段を有することを特徴とする請求項１に記載の情報処理装置。
【請求項１０】
カプセル化文書を読み込むカプセル化文書読み込み手段を有し、前記統合構造化手段は、読み込まれたカプセル化文書と前記構造化データを統合することを特徴とする請求項１に記載の情報処理処置。
【請求項１１】
複数のアプリケーションが生成した複数のドキュメントデータを所定のフォーマットで構造化して複数の構造化データを得る構造化処理と、
各構造化データを統合して構造化し直す統合処理と、
統合したデータから印刷装置が解釈可能な印刷データを生成する生成処理とをコンピュータに実行させるプログラム。
【請求項１２】
前記所定のフォーマットは、ＸＭＬ（eXtensible Markup Language）などのマークアップ言語で記述することを特徴とする請求項１１に記載のプログラム。

【請求項１３】
前記統合したデータのレイアウトを編集し、編集に伴いレイアウトおよび構造を変更する編集処理をコンピュータに実行させる請求項１１に記載のプログラム。
【請求項１４】
前記統合したデータの構造を編集し、編集に伴い構造および統合したデータのレイアウトを変更する構造編集処理をコンピュータに実行させる請求項１１に記載のプログラム。
【請求項１５】
前記統合したデータの構造の印刷データを生成する生成処理をコンピュータに実行させる請求項１１に記載のプログラム。
【請求項１６】
前記統合したデータの構造の一部分を選択する選択処理と、選択された部分の構造の印刷データを生成する生成処理とをコンピュータに実行させる請求項１１に記載のプログラム。
【請求項１７】
前記選択処理によって選択された部分の構造に相当するレイアウトの印刷データを生成する生成処理をコンピュータに実行させる請求項１６に記載のプログラム。
【請求項１８】
前記統合したデータのレイアウトの一部を選択するレイアウト選択処理と、選択された部分のデータの構造の印刷データを生成する生成処理をコンピュータに実行させる請求項１１に記載のプログラム。
【請求項１９】
前記統合したデータをカプセル化文書として出力するカプセル化処理をコンピュータに実行させる請求項１１に記載のプログラム。
【請求項２０】
カプセル化文書を読み込む読み込み処理と、読み込まれたカプセル化文書と前記構造化データを統合する統合処理をコンピュータに実行させる請求項１１に記載のプログラム。
【請求項２１】
請求項１１～２０のいずれか１項記載のプログラムを記録したコンピュータ読み取り可能な記録媒体。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、アプリケーションが生成したドキュメントデータに応じて印刷装置が解釈可能な印刷データを生成する情報処理装置、この装置で用いられるプログラム及びコンピュータ読み取り可能な記録媒体に関するものである。
【背景技術】
【０００２】
プリンタで印刷を行う場合、現状では印刷はアプリケーションごとに行われている。しかしながら会議の資料などを用意する場合などでは、複数のアプリケーションによって作成されたドキュメントを用意しなければならないことが多い。例えば１枚目はワープロソフトで作成した文書、２枚目は表計算ソフトで作成した文書、３枚目はＨＴＭＬ文書、４枚目からは再びワープロソフトで作成文書と印刷したい場合には、現状ではそれぞれプリントアウトして並べ替えなければならず、複数人に配布しなければならない場合は非常に手間のかかる作業となる。
また、ばらばらにアプリケーションごとに印刷するとなると、ページ単位でのレイアウトの変更は印刷物を並べ替えることによって可能であるが、ページ内で異なるアプリケーション間でレイアウトを変更するのは不可能である。
【０００３】
各種アプリケーションのデータを統合する手段としては、Adobe Systems Incorporatedが開発、販売しているAcrobat というソフトウエアがある。Acrobat は、アプリケーショ

ンの生成する印刷コマンドを解析し、ＰＤＦ（Portable Document Format）という文書フォーマットに変換するものである。様々なアプリケーションをＰＤＦファイルにし、複数のＰＤＦファイルを統合して一つのＰＤＦファイルすることが可能である。しかしながら、アプリケーションごとに順番を決めて統合することは可能であるが、ページ単位で順番を入れ替えたり、ページ内でレイアウトを変更したりすることはできない。

【０００４】

各種アプリケーションの印刷データを統合しレイアウトを編集する従来技術としては、例えば、ベクトル図、ラスタ図、写真、文書や表計算書など、種々のアプリケーションプログラムに基づいて作成された種々の形式のアプリケーションデータについて、アプリケーションプログラムの能力に左右されることなくレイアウト編集、印刷することができる技術が提案されている。この例では、全アプリケーションのデータを統合的に扱うことができる中間ファイルを生成するほか、複数のアプリケーションを１枚の任意の位置にレイアウトする編集機能を有している。

【０００５】

しかしながらこの例では、アプリケーションデータを統合し、そのレイアウトを編集するということができているものの、ここで言う中間ファイルは印刷データをさしていると思われ、統合したデータを構造的に扱うことができない。現在では、ドキュメントの大部分は、ＸＭＬ（eXtensible Markup Language）を用いて構造化されてきており、構造化されていない文書は、コンピュータが内容を認識して何らか（検索、データ取得、データ交換など）の操作を自動で行うのは困難であるため、文書の再利用性、利便性、検索性などの面で劣るものとなってしまう。このため、これからの文書は構造化されている文書が主流になると見られ、構造化されていない文書は減少していくものと思われる。従って、様々なアプリケーションのデータを構造化して統合し、かつ、レイアウトも自由に編集できるアプリケーション、装置の開発が望まれている（例えば、特許文献１参照）。

【０００６】

また、プリンタドライバにおいて、印刷データをラスタライズし、かつ既存のイメージファイルに追加する手段を設けて、複数のアプリケーションで作成されるデータを一つのイメージデータファイルとしてまとめる従来技術がある。しかしながら、まとめられたデータが構造化されたものではなく、また、データをまとめるにあたっても、あるアプリケーションデータの後ろに、違うアプリケーションデータを加える方法であるので、ページ単位での入れ替えすらできないという問題がある（例えば、特許文献２参照）。

また、後述する本発明の実施例におけるカプセル化文書については、文書のデータ構造、記憶媒体及び情報処理装置等の一連のカプセル化文書に関する特許文献群がある（例えば、特許文献３参照）。

【特許文献１】特開２００２－２２２１８４号公報
【特許文献２】特開２００１－３１８７７４号公報
【特許文献３】特開２００３－１５９４１号公報

【発明の開示】

【発明が解決しようとする課題】

【０００７】

従って本発明は、各種アプリケーションデータを構造化して統合することにより、異なるアプリケーションデータを統一的に扱い、統合したデータの印刷を行うことを課題とする。

【課題を解決するための手段】

【０００８】

上記の課題を解決するために本発明による情報処理装置は、アプリケーションが生成したドキュメントデータに応じて印刷装置が解釈可能な印刷データを生成する情報処理装置において、複数のアプリケーションが生成した複数のドキュメントデータを所定のフォーマットで構造化して複数の構造化データを得るデータ構造化手段と、各構造化データを統合して構造化し直す統合構造化手段と、統合したデータから印刷データを生成する印刷デ

ータ生成手段とを有することを特徴とするものである。
【０００９】
また、本発明によるプログラムは、複数のアプリケーションが生成した複数のドキュメントデータを所定のフォーマットで構造化して複数の構造化データを得る構造化処理と、各構造化データを統合して構造化し直す統合処理と、統合したデータから印刷装置が解釈可能な印刷データを生成する生成処理とをコンピュータに実行させるようにしたものである。
また、本発明によるコンピュータ読み取り可能な記録媒体は、前記プログラムを記録したものである。
【００１０】
また、前記所定のフォーマットは、ＸＭＬ（eXtensible Markup Language）などのマークアップ言語で記述するようにしてよい。
また、前記統合したデータのレイアウトを編集し、編集に伴いレイアウトおよび構造を変更するようにしてよい。
また、前記統合したデータの構造を編集し、編集に伴い構造および統合したデータのレイアウトを変更するようにしてよい。
【００１１】
また、前記統合したデータの構造の印刷データを生成するようにしてよい。
また、前記統合したデータの構造の一部分を選択する選択処理と、選択された部分の構造の印刷データを生成するようにしてよい。
また、前記選択処理によって選択された部分の構造に相当するレイアウトの印刷データを生成するようにしてよい。
【００１２】
また、前記統合したデータのレイアウトの一部を選択するレイアウト選択処理と、選択された部分のデータの構造の印刷データを生成するようにしてよい。
また、前記統合したデータをカプセル化文書として出力するようにしてよい。
さらに、カプセル化文書を読み込み、読み込まれたカプセル化文書と前記構造化データを統合するようにしてよい。
【発明の効果】
【００１３】
各種アプリケーションデータを同一フォーマットで構造化し、統合することで、各種アプリケーションデータを統一的に扱うことができ、統合したデータの印刷ができる。
【発明を実施するための最良の形態】
【００１４】
各種アプリケーションデータを統一的に扱うことができるようにするために、各種アプリケーションデータを同一フォーマットで構造化し、統合するようにしたものである。
【実施例】
【００１５】
図１は本発明の実施例による情報処理装置のハードウエア構成を示すブロック図である。
情報処理装置としてのパーソナルコンピュータ１００は、情報処理を行うＣＰＵ１と、情報処理中にデータを格納するＲＯＭ２、ＲＡＭ３等の一次記憶装置と、処理結果等を保存するＨＤＤ４等の二次記憶装置と、情報を外部に保管、配布、入手するためのＣＤ－ＲＯＭ５等のリムーバブルメディアと、処理経過や結果等を操作者に表示する表示装置６と、操作者がパーソナルコンピュータに命令や情報等を入力するキーボード７、マウス８と、インターフェイス９，１０と、これらの各部間のデータのやり取りを調停するバスコントローラ１１とで構成されている。
インターフェイス９は外部のプリンタ１２に接続され、インターフェイス１０は、他のコンピュータやプリンタと通信により情報を伝達するためのネットワーク１３に接続されている。

【００１６】
一般にパーソナルコンピュータ１００は、ユーザが電源をＯＮすると、ＣＰＵ１はＲＯＭ２内のローダーというプログラムを起動し、ＨＤＤ４よりオペレーティングシステムというコンピュータのハードウエアとソフトウエアを管理するプログラムをＲＡＭ３に読み込み、オペレーティングシステムが起動する。
このオペレーションシステムは、ユーザの操作に応じてプログラムの起動、情報の読み込み、保存などを行う。オペレーティングシステムとして代表的なものでは、Ｗｉｎｄｏｗｓ（Ｒ）、ＵＮＩＸ、Ｍａｃ　ＯＳ等がある。これらのオペレーティングシステム上で走るプログラムは通常アプリケーションと呼ばれている。印刷は、インターフェイス９を介して接続されたプリンタ１２もしくはネットワーク１３を介してつながっているプリンタ（図示せず）を用いて行われる。
【００１７】
印刷までの流れを、Ｗｉｎｄｏｗｓ（Ｒ）を例として簡単に示す。ワープロなどのアプリケーションソフトが印刷要求を出すと、プリンタドライバは扱うデータの形式とプリンタドイライバ側で扱える処理に関する情報をアプリケーションに送る。ここでは、印刷可能なラスタデータに変換し処理できるものかどうかのチェックが行われる。
印刷可能と判断されると、ラスタイメージを表現するために必要なＧＤＩ（Graphics Device Interface ：出力装置に線・円・多角形・テキストなどを描画するためのＷｉｎｄｏｗｓ（Ｒ）の標準機能）コマンドをコード化し、ジャーナルファイル（一般には、C:\Windows\Temp\ｘｘｘｘ.jnl）に収める。
【００１８】
このジャーナルファイルの生成が完了した時、印刷用ファイル（xxx.prn ）を作成する準備が完了する。アプリケーションはここで開放される（アプリケーションを終了してもプリントが可能になる）。
そして生成されたジャーナルファイルを読み出し、印刷用ラスタデータを生成してプリンタに送られると印刷が開始される。
【００１９】
図２に本実施例によりアプリケーションデータが印刷データになるまでの流れを示す。
各アプリケーションのデータ（アプリデータ１，２，３）は、「データ構造化手段」によりそれぞれデータが構造化されて構造化データとなる。次に、各アプリケーションの構造化データは「データ統合構造化手段」により一つに統合されて統合化構造化データとなる。この統合化構造化データは、「データ編集手段」により編集され、この編集された統合化構造化データから「印刷データ生成手段」により印刷データが生成される。
【００２０】
上記の各アプリケーションデータを構造化する部分について説明する。
図３にアプリケーションデータを構造化する処理の流れを示す。
この例では、アプリケーションデータをＳＶＧ（Scalable Vector Graphics）フォーマットにページごとに変換し、それを最後にカプセル化文書にまとめている。以下に詳しく説明する。
図３において、先にも述べたように、例えばＷｉｎｄｏｗｓ（Ｒ）では、アプリケーションにおいて印刷要求が出され、印刷可能であると判断されると、ラスタイメージを表現するために必要なＧＤＩコマンドをコード化してジャーナルファイルを作成し、印刷用データを作成している。
【００２１】
本実施例では、ここで出てきたＧＤＩコマンドを解釈し、アプリケーションデータを構造化文書（構造化データ）に変換する。また、ＵＮＩＸでは、多くの場合、印刷データはPostScript形式で扱っており、PostScript形式を解釈することにより、アプリケーションデータを構造化文書に変換する。アプリケーションデータを構造化するにあたって採用するフォーマットはＸＭＬ（eXtensible Markup Language）が挙げられる。また、ＸＭＬで記述するグラフィックスフォーマットであるＳＶＧ（Scalable Vector Graphics）を使用

すると、統合したデータをVector Graphics で表現することが可能であり、ＸＭＬで記述されているので構造化にも適している。ここでは、ＳＶＧフォーマットに変換する例を示す。

【００２２】

ＳＶＧは文字通り、スケーラブルなベクタ形式のグラフィックスを定義するための技術であり、ＳＶＧはＸＭＬやＨＴＭＬなどのインターネット関連の仕様を数多く策定しているＷ３Ｃによって作成されたオープンでスタンダードな規格である。ＳＶＧを用いることによって、アプリケーションデータを構造化できるだけでなく、ベクタ形式のグラフィックスで文書を表示、印刷することが可能になる。

【００２３】

ＳＶＧフォーマットの例を図４に示す。

図４を見るとわかるように、ＳＶＧはＸＭＬで記述されている。ＳＶＧタグで囲まれている部分がＳＶＧの書式に相当する部分である。例えば図４中にはＴＥＸＴタグがあるが、このＴＥＸＴタグで囲まれた部分に様々な属性をつけることによって、表示位置や、フォント、文字色などを変えることができる。ＩＭＡＧＥタグも同様に、画像の保存先と表示位置、高さや幅をきめることができる。

【００２４】

これをベクタ形式で表示した例を図５に示す。このように、図や文字列が混在した表示をすることができるため、例えば、ワープロソフトなどに代表されるほとんどのアプリケーションは、ＳＶＧフォーマットに変換することが可能である。

また、円や、楕円、長方形、直線、曲線など様々な図形をベクタ形式で描けるため、例えば、表計算ソフトのようなテーブルの表示も容易であり、ほとんどのアプリケーションデータをＳＶＧフォーマットに変換することが可能である。

【００２５】

ＳＶＧフォーマットへの変換は、１ページごとに行う。つまり、アプリケーションデータの１ページごとにＳＶＧファイルが１ファイルできることになる。しかしながらこのままではページ数の多いアプリケーションデータの場合に取り扱いが煩雑になるため、それを避けるためにカプセル化を行うとよい。カプセル化についてはここでは詳しくは述べないが、ページを管理するファイルを一つ用意して、ＳＶＧビューワプログラムとともに１つのファイルにまとめるのがよい。

【００２６】

ページを管理するファイルの例を図６に示す。図６に示したようにｔｅｘｔタグでは、そのページ数を表し、そのｔｅｘｔタグをａタグのxlink 属性で該当するファイルへのリンクを設定している。このファイルもＳＶＧフォーマットで示したが、ＳＶＧフォーマットである必要は必ずしもない。

【００２７】

このページ管理ファイルを含めた全頁をカプセル化した場合の内部構造を図７に示しておく。図中のProgramsはフォルダであり、ＳＶＧファイルを閲覧するためのSVG Viewerプログラムとを編集するためのSVG Editorプログラムが格納されている。Resourcesフォルダでは、ユーザインターフェイスにおいて、閲覧する際に使用されるボタンなどのリソースが格納されている。またCONTENTSフォルダには、図６に示したコンテンツを管理するページ管理ファイルと、実際のコンテンツである各ページのＳＶＧファイルが格納されており、Media フォルダには、コンテンツ内で使用される画像などのファイルが格納されている。SVG Viewerプログラムは、起動命令が来ると、ページ管理ファイルを読み込み、必要なファイルを取り込んで起動する。

以上のようにアプリケーションデータをＳＶＧフォーマットに変換することで、ＸＭＬの利点を享受し、文書を構造化した上で表示をVector Graphics で行うことができる。

【００２８】

次に、ＳＶＧフォーマットという形で構造化した図２の構造化データを統合する方法を説明する。

各種アプリケーションデータがＳＶＧフォーマットという共通のフォーマットに変換されてカプセル化されているため、単にそれぞれのアプリケーションごとに統合するのは非常に簡単である。各アプリケーションデータは図７に示すような構造のカプセル化文書になっており、SVG Viewerプログラムおよび SVG Editorプログラムは、ページ管理ファイルを読み込んで、必要なリソースを判断する。

【００２９】

従って、カプセル化文書１とカプセル化文書２を統合する場合に変更を要する部分は、ページ管理ファイルと、保持するページ群である。例えば、３ページあるカプセル化文書１と２ページあるカプセル化文書２を統合する場合、ページ管理ファイルは、図８に示すように、５ページ分の記述になる。統合化されたカプセル化文書の構造は図９のようになる。ページ１からページ３およびMedia フォルダのcaplio.jpgがカプセル化文書１からのコンテンツであり、ページ４からページ５およびMedia フォルダのotherfile がカプセル化文書２からのコンテンツである。

【００３０】

次にこの統合されたカプセル化文書のページ順の編集について説明する。

ページ順の編集も非常に簡単である。例えば、統合されたカプセル化文書の構造（図９）に含まれているSVG Editorプログラムは、図１０に示したようなユーザインターフェイスにより、各ページのサムネールを順に表示する機能を持つとする。p000x.svg は該当するＳＶＧファイルのファイル名を示している。ユーザがマウスで例えばページ２を選択し、ページ５の次に移動する（図１１）と、エディタは、図８に示したページ管理ファイルを図１２のように書き換える。

【００３１】

つまり、図８中では、Page2 に相当するフィルはp0002.svg であったが、これがp0003.svg になる。Page3 、Page4 はそれぞれp0004.svg 、p0005.svg になり、Page5 に相当するファイルはp0002.svg になる。もしくは、保持しているファイルのファイル名をそれぞれ相当するファイル名に書き直してもよい。このように比較的容易にページ順の編集をすることができる。すなわち、Acrobat にみられるような単純なアプリケーションの統合ではなく、アプリケーションにかかわらず、ページ順を柔軟に変更することが可能である。

以上のように、アプリケーションデータの印刷コマンドを解析し、ＳＶＧというＸＭＬで記述されるフォーマットに変換し、カプセル化文書することにより、アプリケーションによらない柔軟なかたちで統合、編集が可能である。

【００３２】

次に印刷データ生成手段による印刷データの生成についてであるが、印刷に関しては、通常のアプリケーションからの印刷と同様である。

印刷要求を出すと、印刷可能かどうか判断し、ラスタイメージ＃を表現するために必要なＧＤＩ（Graphics Device Interface ：出力装置に線・円・多角形・テキストなどを描画するためのＷｉｎｄｏｗｓ（Ｒ）の標準機能）コマンドをコード化しジャーナルファイル（一般には、C:\Windows\Temp\ｘｘｘｘ.jnl）に収める。このジャーナルファイルの生成が完了した時、印刷用ファイル（xxx.prn ）を作成する準備が完了する。生成されたジャーナルファイルを読み出し、印刷用ラスタデータを生成し、プリンタに送られると印刷が開始される。

以上が本発明における主な動作である。

【００３３】

次に、その他の動作に関して述べる。

まず、構造の印刷データ生成について述べる。これまで説明してきたように、アプリケーションデータが構造化され、統合されるので、その構造を表示させるのは容易である。図１３に前述の統合化した文書の構造図の例について示す。図９に示したのは、カプセル化文書時の構造であり、プログラムなども含んでいる。図１３はカプセル化文書の内部構造的なものではなく、文書としての構造である。図１３を見るとわかるように、この図に示したドキュメントは、ページが１から５まであり、ページ１にはテキストタブが２つに

イメージタブが一つある。他のページは表示してない（構造を展開していない）が、展開すれば、ページ１のような構造が現れる。

【００３４】

図１３のページ１の部分をさらに展開した図が図１４である。このように、ドキュメントを構造化形式で表示することが可能であり、先に述べたのと同様に、この構造自体の印刷データを生成することもできる。また、編集機能としては、例えば、図１３中のページ１を指定してページ３の下にドラッグアンドドロップすることにより、文書の構造自体を変更できるようにもできる（図１５）。この場合、先に述べたように、ページ管理ファイルを書き換えることにより実際にページレイアウトの変更が可能である。つまり、ユーザは、図１３などに示したような構造表示から、文書自体の構造を直感的に変更することが可能である。また、ページだけでなく、ページ内のコンテンツに関しても編集可能である。また、構造図中の例えばページ１を指定して、ページ１の構造およびその構造に相当するレイアウト（表示）の印刷データを作成することも容易である。

以上の例では、構造図を利用した編集に関して述べたが、レイアウトの一部を指定して、その構造をしてすることも同様に可能である。

【００３５】

エディタアプリケーションの例を図１６に示す。図１６のように、作業ウインドウ中に構造パネルとレイアウトパネルがあり、構造パネルには文書の構造が示され、レイアウトパネルには文書のレイアウトが表示される。構造パネルとレイアウトパネルは連動しており、構造パネル側で編集すればそれがレイアウトパネルにも反映され、レイアウトパネルで編集すれば、それが構造パネルに反映される。図１６はページ３がレイアウトパネルに表示されており、構造パネル中にはページ３がハイライトされている。

以上では、印刷コマンドを解析→構造化→統合という流れであったが、例えばカプセル化文書のように既に構造化されていれば、このカプセル化文書を読み込むことにより、印刷コマンド→構造化の流れは省略することができ、カプセル化文書どうしの統合により、統合データを生成することもできる。

【００３６】

本実施例によれば、次の効果を得ることができる。

１）各種アプリケーションデータを同一フォーマットで構造化し、統合することで、各種アプリケーションデータを統一的に扱うことができ、統合したデータの印刷ができる。

２）ＸＭＬに代表されるようなマークアップ言語で記述することでアプリケーションデータを構造化することにより、アプリケーションデータを汎用的でプログラムなどが扱いやすいものにすることができる。

【００３７】

３）異なるアプリケーションデータから生成された共通フォーマットを編集することで、アプリケーションによらず、統一的にデータを編集し、データ編集に伴う変更を自動的に文書構造に反映させることにより、同一フォーマットで構造化し、統合されるので、結果としてアプリケーションによらない印刷レイアウトの編集が容易にでき、統合されたデータを編集すると自動的にその構造も編集されることにより、ユーザは文書構造を意識することなく、文書の構造を変更できる。

【００３８】

４）ユーザが文書構造という観点で文書の編集を行えるようにし、文書の構造の編集に連動して、レイアウトを自動的に変更することにより、３）とは逆に、構造自体を編集できるようになり、ユーザは文書構造の把握および明示的な構造変更を行うことができ、構造の編集の結果をレイアウトに自動的に反映させることで、構造変化に伴うレイアウトの変更がユーザに対して容易に示すことができる。

【００３９】

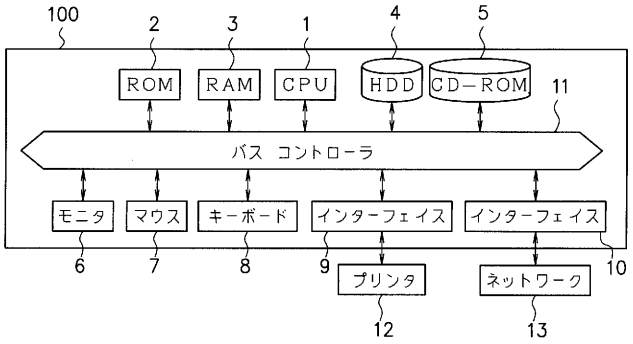
５）文書構造の印刷データを生成することにより、文書のレイアウトの印刷データだけでなく、文書構造の印刷データも印刷することができる。

６）文書構造上で選択された部分の構造の印刷データを生成することにより、文書構造

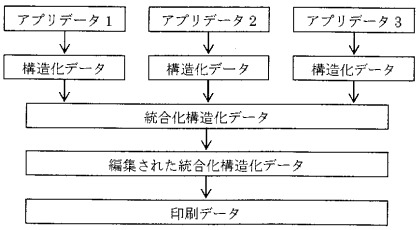
全体だけでなく、文書構造の一部の印刷データを生成することができる。
７）文書構造上で選択された部分のレイアウトの印刷データを生成することにより、選択された文書構造に対応するレイアウトの印刷データを生成でき、ユーザは必要とする部分のレイアウトを文書構造の観点から指定し、印刷することができる。
【００４０】
８）レイアウト上で選択された部分の構造の印刷データを生成することにより、選択されたレイアウトに相当する文書構造の印刷データを生成でき、ユーザは必要とする部分の構造をレイアウトの観点から指定し、印刷することができる。
９）統合されたデータをカプセル化文書として出力することにより、様々なアプリケーションデータからなる単一電子文書を生成することができ、その上専用のアプリケーションを不要にすることができる。
１０）カプセル化文書ともデータを統合できるようにすることにより、カプセル化文書との親和性を高めることができる。
【００４１】
尚、前述の動作に基づく処理を、情報処理装置のＣＰＵが実行するためのプログラムは本発明によるプログラムを構成し、このプログラムを記憶する記憶媒体は、本発明によるコンピュータ読み取り可能な記憶媒体を構成する。この記憶媒体としては、半導体記憶装置や光学的及び／又は磁気的な記憶装置等を用いることができる。このようなプログラム及び記憶媒体を、上述した実施の形態とは異なる構成のシステム等で用い、そこのＣＰＵで上記プロラムを実行させることにより、本発明と実質的に同じ効果を得ることができる。
【図面の簡単な説明】
【００４２】
【図１】本発明の実施例による情報処理装置の構成を示すブロック図である。
【図２】アプリケーションデータが印刷データになるまでの流れを示すフローチャートである。
【図３】アプリケーションデータを構造化する流れを示すフローチャートである。
【図４】ＳＶＧフォーマットの例を示す構成図である。
【図５】ベクタ形式による表示例を示す構成図である。
【図６】ページ管理ファイルの例を示す構成図である。
【図７】ページ管理ファイルを含む全ページをカプセル化文書にした場合の内部構造を示す構成図である。
【図８】ページ管理ファイルの記述例を示す構成図である。
【図９】統合されたカプセル化文書の構造を示す構成図である。
【図１０】ユーザインターフェイスの表示例を示す構成図である。
【図１１】ファイルのページ移動を行った場合のユーザインターフェイスの表示例を示す構成図である。
【図１２】ファイルのページ移動を行った場合のユーザインターフェイスの記述例を示す構成図である。
【図１３】カプセル化文書の構造を示す構成図である。
【図１４】カプセル化文書の構造を展開した場合を示す構成図である。
【図１５】カプセル化文書の構造を変更した場合を示す構成図である。
【図１６】エディタアプリケーションの表示例を示す構成図である。
【符号の説明】
【００４３】
１００　パーソナルコンピュータ
１　ＣＰＵ
２　ＲＯＭ
３　ＲＡＭ
４　ＨＤＤ
５　ＣＤ－ＲＯＭ

６　　表示装置
７　　キーボード
８　　マウス８
９、１０　　インターフェイス
１１　　バスコントローラ
１２　　プリンタ
１３　　ネットワーク

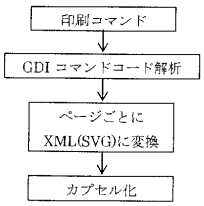
【図１】



【図２】



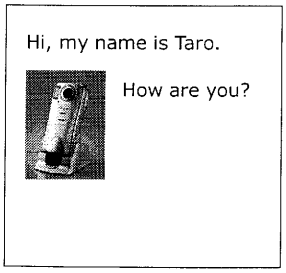
【図３】



【図４】

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="20cm" height="40cm" viewBox="0 0 2000 4000"
xmlns="http://www.w3.org/2000/svg" version="1.1">
    <text x="50" y="50" font-family="Verdana" font-size="55" fill="black" >
        Hi, my name is Taro.
    </text>
    <text x="200" y="150" font-family="Verdana" font-size="55" fill="black" >
        How are you?
    </text>
    <image width="322" height="447" xlink:href="Media/caplio.jpg"
transform="matrix(0.36 0 0 0.36 53.8574 66.3584)"/>
</svg>
```

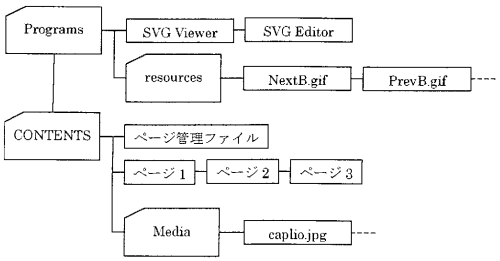
【図５】



【図６】

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="20cm" height="40cm" viewBox="0 0 2000 4000"
xmlns="http://www.w3.org/2000/svg" version="1.1">
    <a xlink:href="p0001.svg">
        <text x="50" y="150" font-family="Verdana" font-size="55" fill="black" >
            Page1
        </text>
    </a>
    <a xlink:href="p0002.svg">
        <text x="50" y="250" font-family="Verdana" font-size="55" fill="black" >
            Page2
        </text>
    </a>
    <a xlink:href="p0003.svg">
        <text x="50" y="350" font-family="Verdana" font-size="55" fill="black" >
            Page3
        </text>
    </a>
</svg>
```

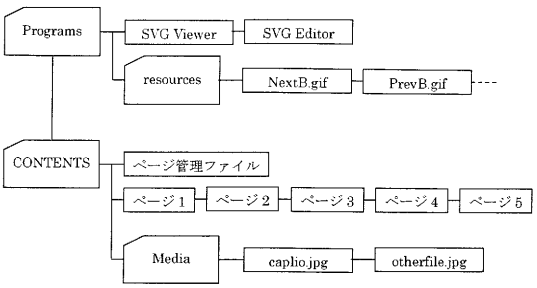
【図７】



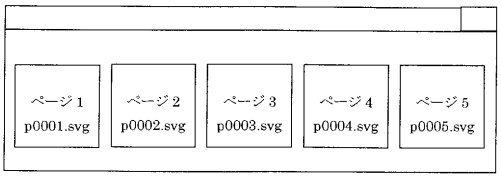
【図８】

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="20cm" height="40cm" viewBox="0 0 2000 4000"
xmlns="http://www.w3.org/2000/svg" version="1.1">
    <a xlink:href="p0001.svg">
        <text x="50" y="150" font-family="Verdana" font-size="55" fill="black" >
            Page1
        </text>
    </a>
    <a xlink:href="p0002.svg">
        <text x="50" y="250" font-family="Verdana" font-size="55" fill="black" >
            Page2
        </text>
    </a>
    <a xlink:href="p0003.svg">
        <text x="50" y="350" font-family="Verdana" font-size="55" fill="black" >
            Page3
        </text>
    </a>
    <a xlink:href="p0004.svg">
        <text x="50" y="450" font-family="Verdana" font-size="55" fill="black" >
            Page4
        </text>
    </a>
    <a xlink:href="p0005.svg">
        <text x="50" y="550" font-family="Verdana" font-size="55" fill="black" >
            Page5
        </text>
    </a>
```

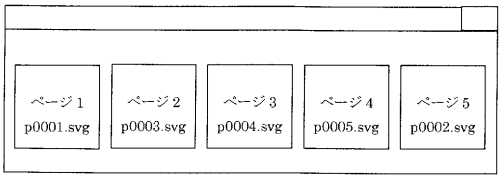
【図９】



【図１０】



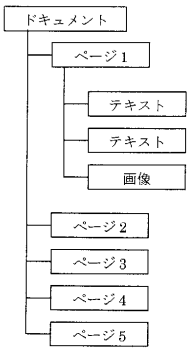
【図１１】



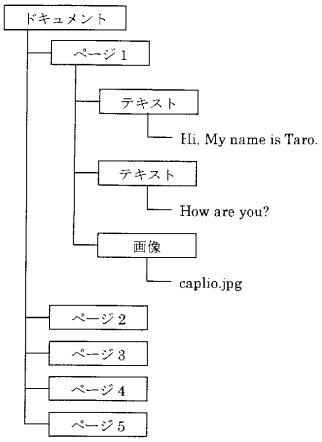
【図１２】

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="20cm" height="40cm" viewBox="0 0 2000 4000"
xmlns="http://www.w3.org/2000/svg" version="1.1">
    <a xlink:href="p0001.svg">
        <text x="50" y="150" font-family="Verdana" font-size="55" fill="black" >
            Page1
        </text>
    </a>
    <a xlink:href="p0003.svg">
        <text x="50" y="250" font-family="Verdana" font-size="55" fill="black" >
            Page2
        </text>
    </a>
    <a xlink:href="p0004.svg">
        <text x="50" y="350" font-family="Verdana" font-size="55" fill="black" >
            Page3
        </text>
    </a>
    <a xlink:href="p0005.svg">
        <text x="50" y="450" font-family="Verdana" font-size="55" fill="black" >
            Page4
        </text>
    </a>
    <a xlink:href="p0002.svg">
        <text x="50" y="550" font-family="Verdana" font-size="55" fill="black" >
            Page5
        </text>
    </a>
```

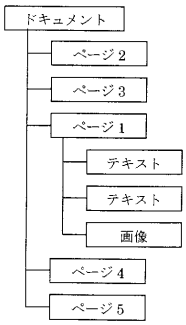
【図１３】



【図１４】



【図１５】



【図１６】