(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3544666号
(P3544666)
(45)発行日　平成16年7月21日(2004.7.21)　(24)登録日　平成16年4月16日(2004.4.16)

(51)Int.Cl.[7]　F I
G09G 5/14　G09G 5/14 C
G06F 3/14　G06F 3/14 350A

請求項の数 12　(全 28 頁)

(21)出願番号　特願平7-512578
(86)(22)出願日　平成6年1月6日(1994.1.6)
(65)公表番号　特表平9-504884
(43)公表日　平成9年5月13日(1997.5.13)
(86)国際出願番号　PCT/US1994/000145
(87)国際公開番号　WO1995/012194
(87)国際公開日　平成7年5月4日(1995.5.4)
審査請求日　平成13年1月9日(2001.1.9)
(31)優先権主張番号　08/142,894
(32)優先日　平成5年10月25日(1993.10.25)
(33)優先権主張国　米国(US)

(73)特許権者
オブジェクト　テクノロジー　ライセンシング　コーポレイション
アメリカ合衆国　９５０１４―２０８３　カリフォルニア州　クパチーノ　メールステーション　３８―オーティーエル　インフィニト　ループ　１
(74)代理人
弁理士　谷　義一
(74)代理人
弁理士　阿部　和夫
(72)発明者　ローゼンシュタイン，ラリー，エス．
アメリカ合衆国　９５０５１　カリフォルニア州　サンタ　クララ　ミュア　アヴェニュ　１８２

最終頁に続く

(54)【発明の名称】オブジェクト指向ディスプレイ・システム

(57)【特許請求の範囲】

【請求項１】

ディスプレイ・デバイスを制御してディスプレイを生成し、該ディスプレイはデスクトップ・バックグラウンド上に表示される複数のウィンドウ・エリアを有し、前記複数のウィンドウ・エリアの各々は、複数のアプリケーション・プログラムの１つによって生成されるスクリーン情報を表示するコンピュータ・システムにおいて、

メモリと、

複数のストレージ・エリアを有するスクリーン・バッファ・ストレージ装置であって、前記複数のストレージ・エリアの各々はサイズを有し、前記複数のウィンドウ・エリアの１つにスクリーン情報をストアし、ストアされたスクリーン情報が表示を生成するために前記ディスプレイ・デバイスによって使用されるスクリーン・バッファ・ストレージ装置と、

前記複数のアプリケーション・プログラムによって制御されるプロセッサであって、前記複数のアプリケーション・プログラムの各々が、前記スクリーン・バッファ・ストレージ装置のステージ・エリアに直接スクリーン情報をストアする手段を有するプロセッサと、

前記ディスプレイ・デバイスを制御するために、前記プロセッサと一緒に動作するオペレーティング・システムと、

前記ウィンドウ・エリア内の変更に応じて、前記ウィンドウ・エリアの変更を複数のアプリケーション・プログラムに伝達して、前記複数のアプリケーション・プログラムが前記ウィンドウ・エリアの変更に応答する手段と、

前記メモリ内にあり、および前記複数のウィンドウの各々に関連付けられたウィンドウ・オブジェクトであって、該ウィンドウ・オブジェクトの各々は、ウィンドウ・データと、ウィンドウ関数とを含み、アプリケーション・プログラムによって作成されているウィンドウ・オブジェクトにおいて、前記ウィンドウ・オブジェクトが、前記複数のアプリケーション・プログラムの１つからの要求を受け取るメカニズムと、受け取った要求に応じて、前記要求をした前記複数のアプリケーション・プログラムの１つへストレージ・エリア・ロケーションとサイズを引渡す機能とを有するウィンドウ・オブジェクトとを備え、
前記ウィンドウ・データが、前記ウィンドウに関連付けられたストレージ・エリアのサイズのコピーを有し、前記ウィンドウ・オブジェクトが前記複数のアプリケーション・プログラムの１つからの要求に応じて、前記ストレージ・エリアのサイズのコピーの有効性を判定する機能をさらに有し、
前記伝達手段が、各ストレージ・エリアのサイズと一緒に、第１スタンプ、すなわち、前記ストレージ・エリアのサイズが再計算された時間を示す第１タイムスタンプをストアする機能を含み、前記ウィンドウ・オブジェクトが、前記ストレージ・エリアのサイズ・コピーと一緒に、第２スタンプ、すなわち、前記ストレージ・エリアのサイズ・コピーがウィンドウ・オブジェクトにストアされた時間を示す第２タイムスタンプをストアする機能を有することを特徴とするコンピュータ・システム。

【請求項２】
選択された１つのストレージ・エリアのサイズが１つのウィンドウに対応しており、前記有効性を判定する機能は、選択されたストレージ・エリアのサイズのための前記アプリケーション・プログラムからの要求に応じて、当該選択されたストレージ・エリアのサイズと一緒にストアされた第１タイムスタンプを前記伝達する手段から取り出す機能と、当該取り出された第１タイムスタンプに応じて、前記取り出された第１タイムスタンプを前記ウィンドウ・オブジェクトにストアされた第２タイムスタンプと比較する機能とを含むことを特徴とする請求項１に記載のコンピュータ・システム。

【請求項３】
前記タイムスタンプ比較機能が、前記ウィンドウ・オブジェクトにストアされている前記ストレージ・エリアのサイズが有効でないことを示すとき、前記伝達する手段にストアされている前記ストレージ・エリアのサイズを取り出す手段を前記有効性を判定する機能がさらに備えることを特徴とする請求項２に記載のコンピュータ・システム。

【請求項４】
ディスプレイ・デバイスを制御して、ディスプレイを生成し、該ディスプレイはデスクトップ・バックグラウンド上に表示された複数のウィンドウ・エリアを有し、前記複数のウィンドウ・エリアの各々が、複数のアプリケーション・プログラムの１つによって生成されているスクリーン情報を表示するコンピュータ・システムにおいて、
メモリと、
複数のサイズ変更可能ストレージ・エリアを有するスクリーン・バッファ・ストレージ装置であって、前記複数のサイズ変更ストレージ・エリアの各々はサイズを有し、それらの各々が前記複数のウィンドウ・エリアの１つのためのスクリーン情報をストアするスクリーン・バッファ・ストレージ装置と、
前記複数のアプリケーション・プログラムによって制御されて前記複数のアプリケーション・プログラム内の選択された１つからスクリーン情報を受け取り、前記選択した１つのアプリケーション・プログラム対応する、前記複数の変更可能ストレージ・エリアの１つにある、当該受け取ったスクリーン情報をストアするプロセッサと、
前記ディスプレイ・デバイスを制御する前記プロセッサと一緒に動作するオペレーティング・システムと、
前記ストレージ・エリアのサイズをストアするための共有データエリアを有し、１つのストレージ・エリアのストレージ・エリアのサイズの変化に応じ、少なくとも１つの他のストレージ・エリアのストレージ・エリアのサイズを変更する手段と、
前記複数のアプリケーション・プログラムの各々によって前記メモリ内に作成されたウィ

ンドウ・オブジェクトであって、作成に際し、ウィンドウ・オブジェクトが前記変更する手段と一緒に動作して、複数のストレージ・エリアの１つを割り当てることによりウィンドウ・エリアを作成するウィンドウ・オブジェクトと
を備え、
前記変更する手段が前記オペレーティング・システムにより作成され、
前記複数のウィンドウ・エリアはオーバーラップしており、前面から後面への配列で可視的に表示され、前記変更する手段は、リンクリストで複数ウィンドウの各々を保持するメカニズムであって、該リンクリストは各ウィンドウに関する識別番号を収め、当該各ウィンドウは、複数のウィンドウ中の位置を示すウィンドウの種類が割り当てられているメカニズムを備え、
前記変更する手段が、前記複数のウィンドウ・エリアの１つの位置およびサイズの変化に応じてウィンドウ・エリアを計算する機能を有し、該ウィンドウ・エリアは位置とサイズの変化によって変更された前記複数のウィンドウ・エリアの１つの部分を示し、前記変更する手段は、変更のウィンドウ・エリアに応じてストレージ・エリアを再計算する機能を有し、該ストレージ・エリアは前記リンクリストによって判定された１つのウィンドウ・エリアの後ろに表示されるウィンドウ・エリアに対応し、
前記再計算する機能は、変更のエリアと、ウィンドウ・エリアに対応する前記ストアしたエリア・サイズを組み合わせることによって前記ストレージ・エリアのサイズを再計算し、前記ウィンドウ・エリアは前記リンクリストによって判定された１つのウィンドウ・エリアの後ろに表示され、
前記複数のウィンドウ・オブジェクトの各々は、キャッシュ・メモリと、前記ウィンドウ・エリアを操作する複数のメソッドを有し、
各ウィンドウ・オブジェクトと前記変化する手段との間で、情報が、データ・ストリーム・オブジェクトによって転送され、
各ウィンドウ・オブジェクトは、前記複数のアプリケーション・プログラムの１つから要求を受け取るメカニズムと、受け取った要求に応じて、ストレージ・エリア・ロケーションとストレージ・エリアのサイズを、前記要求の複数のアプリケーション・プログラムの１つに引き渡す機能とを有し、
前記関連付けられたウィンドウの前記ストレージ・エリアのサイズのコピーが、前記ウィンドウ・オブジェクト・キャッシュ・メモリにストアされ、前記ウィンドウ・オブジェクトは、前記複数のアプリケーションの１つからの要求に応じ、前記ストレージ・エリアのサイズ・コピーの有効性を判定する機能を有する
ことを特徴とするコンピュータ・システム。
【請求項５】
前記有効性を判定する機能は、前記ウィンドウに対応した、選択のストレージ・エリアのサイズのための前記アプリケーション・プログラムからの要求に応じて、前記選択のストレージ・エリアのサイズと一緒にストアされた第１タイムスタンプを前記変更する手段から取り出す機能と、前記取り出された第１タイムスタンプに応じて、前記取り出された第１タイムスタンプを前記キャッシュ・メモリにストアされた第２タイムスタンプと比較する機能とを含むことを特徴とする請求項４に記載のコンピュータ・システム。
【請求項６】
前記ウィンドウ・マネージャ・オブジェクトによる最後の再計算の前に、前記キャッシュ・メモリにストアされている前記ストレージ・エリアのサイズがストアされていたことを示すとき、前記有効性を判定する機能が、前記変更する手段にストアされている前記ストレージ・エリアのサイズを取り出す手段をさらに有していることを特徴とする請求項５に記載のコンピュータ・システム。
【請求項７】
ディスプレイ・デバイスを制御してディスプレイを生成し、該ディスプレイは、デスクトップ・バックグラウンド上で表示され、オーバーラップする複数のウィンドウを有し、前記複数のウィンドウの各々が、複数のアプリケーション・プログラムの１つによって生成

されるスクリーン情報を表示する可視エリアを有するコンピュータ・システムにおいて、
メモリと、
前記複数のアプリケーション・プログラムによって制御されて、前記コンピュータ・システムのオペレーションを制御および調整するプロセッサと、
前記ディスプレイ・デバイスを制御するために、プロセッサと一緒に動作するオペレーティング・システムと、
共有データを有する手段であって、ウィンドウ可視エリアをストアし、前記複数のウィンドウの１つの可視エリアの変化に応じて他の前記複数のウィンドウの前記可視エリアを変更する手段と、
前記メモリ内にあり、前記複数のアプリケーション・プログラムの各々によって作成されたウィンドウ・オブジェクトであって、各ウィンドウ・オブジェクトが、前記複数のウィンドウの１つに関連付けられ、各ウィンドウ・オブジェクトが、前記関連付けられたウィンドウの前記可視エリアのコピーをストアするキャッシュ・メモリを含むウィンドウ・オブジェクトと
を備え、
各ウィンドウ・オブジェクトが、前記複数のアプリケーション・プログラムの１つからの要求に応じ、ウィンドウ可視エリアコピーの有効性を判定し、
前記ストアする手段が、各ウィンドウ可視エリアと一緒に、前記ウィンドウ可視エリアが最後に再計算された時刻を示す第１タイムスタンプをストアする機能を有し、前記ウィンドウ可視エリアコピーが前記キャッシュ・メモリにストアされた時刻を示す第２タイムスタンプを前記キャッシュメモリ内にストアする機能を前記ウィンドウ・オブジェクトが有することを特徴とするコンピュータ・システム。
【請求項８】
前記有効性を判定する機能は、選択のウィンドウ可視エリアのためのアプリケーション・プログラムからの要求に応じて、第１タイムスタンプを前記ストアする手段から取り出す機能と、当該取り出された第１タイムスタンプに応じて、前記取り出された第１タイムスタンプを前記キャッシュ・メモリにストアされた第２タイムスタンプと比較する機能とを含むことを特徴とする請求項７に記載のコンピュータ・システム。
【請求項９】
前記ウィンドウ・マネージャ・オブジェクトによる最後の計算の前に、前記キャッシュ・メモリにストアされた前記可視エリアがストアされたことを前記タイムスタンプ比較装置が示すとき、前記ストアする手段から前記ウィンドウ可視エリアを取り出す手段を前記有効性を判定する機能がさらに有することを特徴とする請求項８に記載のコンピュータ・システム。
【請求項１０】
デスクトップ・バックグラウンド上に表示された複数のウィンドウ・エリアの１つにスクリーン情報を表示するアプリケーション・プログラムの方法であって、前記複数のウィンドウ・エリアの各々が、変更可能な可視エリアを有する前記方法において、
A.共有データエリアを有し、１つのウィンドウの可視エリアの変更に応じて、少なくとも１つの他のウィンドウの可視エリアを変更するウィンドウ・マネージャを作成するステップと、
B.前記アプリケーション・プログラムに前記ウィンドウ・マネージャからの、複数のウィンドウ・エリアの１つの可視エリアを要求させるステップと、
B1.前記複数のウィンドウの各々に関連付けられたウィンドウ・オブジェクトを作成するステップであって、前記各ウィンドウ・オブジェクトはウィンドウ・データとウィンドウ関数を含んでおり、前記各ウィンドウ・オブジェクトはアプリケーション・プログラムによって作成されるステップと、
B2.前記ウィンドウ・オブジェクトに前記可視エリアを要求させるステップと、
B2B.可視エリアのキャッシュ・コピーを前記ウィンドウ・オブジェクト内で保持するステップと、

B2C.前記可視エリアのコピーを前記ウィンドウ・マネージャ・オブジェクトに要求する前に前記ウィンドウ・オブジェクト内のキャッシュ・コピーをチェックするステップと、
B2D.前記可視エリアが再計算された時の時刻を示す第１タイムスタンプを前記各ウィンドウ可視エリアと一緒にストアするステップと、
B2E.前記キャッシュ・コピーが前記ウィンドウ・オブジェクトにストアされた時刻を示す第２タイムスタンプをウィンドウ・オブジェクトにストアするステップと、
C.前記アプリケーション・プログラムに、前記ウィンドウ・マネージャから取得した可視エリアにだけスクリーン情報を表示させるステップとを含むことを特徴とする方法。
【請求項１１】
前記ステップB2は、
B2F.前記ウィンドウ可視エリアについての要求を前記アプリケーション・プログラムから受け取ったとき、選択した前記ウィンドウ可視エリアと一緒にストアされた前記第１タイムスタンプを前記ウィンドウ・マネージャから取り出すステップと、
B2G.当該取り出された第１タイムスタンプを前記ウィンドウ・オブジェクトにストアされた第２タイムスタンプと比較するステップとを含むことを特徴とする請求項10に記載の方法。
【請求項１２】
前記ステップB2は、
B2H.ウィンドウ・オブジェクトにストアされたキャッシュ・コピーが有効でないことを、前記ステップB2Gのタイムスタンプ比較が示しているとき、前記ウィンドウ・マネージャにストアされたウィンドウ可視エリアを取り出すステップをさらに含むことを特徴とする請求項11に記載の方法。
【発明の詳細な説明】
著作権表記
本特許出願の一部には、著作権保護の対象となる内容が含まれている。著作権所有者は、米国特許商標庁に記録されている特許文書または特許開示内容を何人もが原文通りに複製することを妨げるものではない。
発明の分野
本発明はコンピュータ・システムにおける改良に関し、具体的には、グラフィック・ユーザ・インタフェースにおいてウィンドウ・ディスプレイ・エリアを管理するためのオペレーティング・システム・ソフトウェアに関する。
発明の背景
現代のコンピューティング・システムの最も重要な側面の１つは、人間のユーザとマシンとの間のインタフェースである。初期の最もポピュラーのタイプのインタフェースはテキストをベースとしていた。つまり、ユーザはテキスト文字をキーボードからタイプすることでマシンと通信し、マシンはテキスト文字をディスプレイ・スクリーン上に表示することでユーザと通信していた。最近では、グラフィック・ユーザ・インタフェース（graphic user interface）がポピュラーになっているが、そこではマシンはテキストとピクチャを含むグラフィックスをディスプレイ・スクリーン上に表示することによってユーザと通信し、ユーザはテキスト・コマンドをタイプするだけではなく、表示されたピクチャをマウスなどのポインティング・デバイスで操作することによってマシンと通信している。
最新コンピュータ・システムの多くは、ウィンドウ環境と呼ばれるグラフィック・ユーザ・インタフェースを使用して動作している。代表的なウィンドウ環境では、ディスプレイ・スクリーン上で縦長に表示されるグラフィカル・ディスプレイが電子的「デスクトップ」の表面と同じように配置され、コンピュータ上で実行される各アプリケーション・プログラムは、１つ以上の電子的「ペーパ・シート」で表され、これらシートは「ウィンドウ」と呼ばれるスクリーンの長方形領域に表示される。
各ウィンドウ領域には、関連のアプリケーション・プログラムによって生成された情報が表示されるのが一般的であり、複数のウィンドウ領域が同時にデスクトップ上に存在することもあり、その場合は、各ウィンドウ領域は異なるアプリケーション・プログラムによ

って生成された情報を表している。アプリケーション・プログラムは、イメージ、グラフィックスまたはテキストをウィンドウ領域内でドロー（描画）または「ペイント」することによって、情報を各ウィンドウを通してユーザに提示している。ユーザの方は、ウィンドウ領域内のオブジェクトを、ポインティング・デバイスで制御されるカーソルで「ポイント」（指すこと）して、オブジェクトを操作または移動することにより、さらに、情報をキーボードからタイプインすることによりアプリケーションと通信している。ウィンドウ領域はディスプレイ・スクリーン上で各所に動かしたり、サイズや外観を変更したできるので、ユーザはデスクトップを都合のよいように配置することが可能になっている。

ウィンドウ領域の各々は、サイジング・ボックス（sizing box）、ボタン、スクロール・バーなどの、いくつかの標準グラフィカル・オブジェクトを含んでいるのが代表的である。これらは、ユーザがカーソルでポイントして選択し、操作できるというユーザ・インタフェース・デバイスの特徴を表している。これらのデバイスが選択され、操作されると、ウィンドウ・システムを通してその基となるアプリケーション・プログラムに対して、ユーザの操作による制御が行われたことが、通知される。

一般的に、上述したウィンドウ環境はコンピュータのオペレーティング・システムの一部になっている。オペレーティング・システムはユーティリティ・プログラム群も含んでいるのが代表的である。ユーティリティ・プログラムは、コンピュータ・システムが基本的操作を実行することを可能にするものである。基本的操作としては、情報をディスク・メモリにストアし、そこから情報を取り出すこと、ファイルの作成、命名、改名といったファイル操作を実行すること、場合によっては、誤動作の検出や回復のための診断操作を実行すること、などの操作がある。

コンピュータ・システムの最後の部分は「アプリケーション・プログラム」であり、これはオペレーティング・システムとやりとりすることによって、さらにハイレベルの機能を実現し、特定のタスクを実行し、ユーザとの直接的インタフェースを提供する。アプリケーション・プログラムは、一連のタスク・コマンドをオペレーティング・システムに渡し、これを受けてオペレーティング・システムが要求タスクを実行することにより、オペレーティング・システムの機能を利用するのが代表的である。例えば、オペレーティング・システムが特定の情報をコンピュータのディスク・メモリにストアすること、あるいはビデオ・ディスプレイから情報を表示することをアプリケーション・プログラムが要求することができる。

図１は、アプリケーション・プログラムとオペレーティング・システムの双方を利用する、代表的な従来のコンピュータ・システムを示す概略図である。コンピュータ・システムは点線の枠100で示され、アプケーション・プログラムは枠102で示され、オペレーティング・システムは枠106で示されている。アプリケーション・プログラム102とオペレーティング・システムとの前述したやりとりは、矢印104で図示されている。このデュアル・プログラム・システムは、メインフレームからパーソナル・コンピュータに至るまでの、多種類のコンピュータ・システムで使用されている。

スクリーン・ディスプレイの扱い方はコンピュータによって異なり、この点に関して、図１は従来のパーソナル・コンピュータ・システムを示している。スクリーン・ディスプレイを提供するためには、アプリケーション・プログラムは表示すべき情報をスクリーン・バッファ110にストアしておくのが一般的である（このストア操作は矢印108で図示されている）。システム内の種々のハードウェアとソフトウェアの制御の下で、スクリーン・バッファ110の内容はバッファから読み出され、矢印114で図示するように、ディスプレイ・アダプタ112に渡される。ディスプレイ・アダプタ112はハードウェアとソフトウェア（ファームウェアの形になっていることもある）を内蔵しており、スクリーン・バッファ110に入っている情報を、ケーブル116でディスプレイ・アダプタ112に接続されたディスプレイ・モニタ118をドライブするために使用できる形体に変換する。

図１に示す従来の構成は、どの時点においても単一のアプリケーション・プログラム102を動かすシステムで好適に作動する。この単純なシステムが正しく作動するのは、上記単一のアプリケーション・プログラム102がスクリーン・バッファ領域110全体のいずれかの

領域に情報を書き込んでも、表示の問題が起こらないためである。しかし、図１に示す構成を、２つ以上のアプリケーション・プログラム102を同時に動作状態にできるようなシステム（例えば、「マルチタスキング」コンピュータ・システム）で使用すると、表示の問題が発生するおそれがある。具体的には、各アプリケーション・プログラムがスクリーン・バッファ110全体に対してアクセスできるとき、アプリケーション同士がなんらかの方法で直接に連絡し合っていないと、あるアプリケーションは別のアプリケーションが使用しているスクリーン・バッファ部分をオーバライト（重書き）するおそれがあるので、一方のアプリケーションによって生成された表示が他方のアプリケーションによって生成された表示でオーバライトされることになる。

そのために、複数のアプリケーション・プログラムのオペレーションを調整するためのメカニズムが開発され、各アプリケーション・プログラムをスクリーン・バッファの一部だけに限定し、それにより、他のディスプレイから切り離すようにしている。この調整は、複数のウィンドウがスクリーン・ディスプレイ上で「オーバラップ」することを許容しているシステムでは複雑化する。ウィンドウが「オーバラップ」して現れるようにスクリーン・ディスプレイが構成されているときは、スクリーン上で別のウィンドウの「手前」に現れるウィンドウはその下のウィンドウの一部に被さって隠すことになる。従って、最も手前のウィンドウを除き、その下のウィンドウは、一部だけがスクリーン上に描画されるので、どの時点でも「可視である」のはその部分だけになる。さらに、ウィンドウはユーザが動かしたり、サイズを変えたりできるので、各ウィンドウで可視な部分は、他のウィンドウを動かしたり、サイズを変えたりすると、変化することになる。そのために、各アプリケーション・プログラムに割り当てられたスクリーン・バッファ部分も、他のアプリケーションからのウィンドウを動かしたり、サイズを変えたりすると変化することになる。

ウィンドウを動かしたり、サイズを変えたりすると起こる、迅速なスクリーン変化に対処するために必要なスクリーン・バッファの変更を効率的に管理するために、図１に示す従来のコンピュータ構成は図２に示すように改良されている。この新しい構成では、コンピュータ・システム200はコンピュータ・システムで同時に実行されている、１つ以上のアプリケーション・プログラムによって制御されている。なお、図には、プログラム202と216が示されており、コンピュータシステムにおいて同時に移動する。プログラムの各々は、矢印206と220で図示するように、オペレーティング・システム204と接続されている。しかし、ディスプレイ・スクリーン上で情報を表示するために、アプリケーション・プログラム202と216は、オペレーティング・システム204内に置かれている中央ウィンドウ管理プログラム218に表示情報を送っている。ウィンドウ管理プログラム218の方は、矢印208で図示するようにスクリーン・バッファ210と直接的に接続されている。スクリーン・バッファ210の内容は、矢印212で示すように、ケーブル222でディスプレイ・モニタ224に接続されたディスプレイ・アダプタ214に渡される。

このようなシステムでは、ウィンドウ・マネージャ218は、ウィンドウ表示の全てのメンテナンス、すなわち、アプリケーション・プログラムの動作時にユーザが見るウィンドウ表示についての全てをメンテナンス（管理）することを受け持つのが一般である。ウィンドウ・マネージャ218は、すべてのアプリケーション・プログラムと連絡をとっているので、ウィンドウ表示がオーバラップしないようにアプリケーション間の調整を行うことができる。従って、ウィンドウの位置とサイズを記録しておくと共に、ドロー（描画）し、ウィンドウの移動時に再ドローしなければならないウィンドウ・エリアを記録しておくのはウィンドウ・マネージャの仕事である。

ウィンドウ・マネージャ218はアプリケーション202と216の各々から表示要求を受け取る。しかし、ウィンドウ・マネージャ218だけがスクリーン・バッファ210と接続するので、ウィンドウ・マネージャはスクリーン・バッファ210のそれぞれのエリアを各アプリケーション用に割り振ることができ、別のアプリケーションによって生成された表示を誤って各アプリケーションがオーバライトすることがないようにする。図２に示す構成を利用するウィンドウ環境は、数種類の異なるものが市販されている。そのようものとして、X/Wi

ndowオペレーティング環境、マイクロソフト社によって開発されたWINDOWSグラフィカル・ユーザ・インタフェース、およびインターナショナル・ビジネス・マシンズ（IBM）社によって開発されたOS/2プレゼンテーション・マネージャがある。
これらのウィンドウ環境の各々は独自の内部ソフトウェア・アーキテクチャをもっているが、これらのアークテクチャはいずれも、コンピュータ・ネットワーク・ソフトウエアを記述するために使用される多層（マルチレイヤ）モデルに類似した多層モデルを使用することにより分類することができる。代表的な多層モデルは次のような層（レイヤ）を含んでいる。
ユーザ・インタフェース
ウィンドウ・マネージャ
資源管理と通信
コンポーネント・ドライバ・ソフトウェア
コンピュータ・ハードウェア
ただし、「ウィンドウ環境」という用語は上記の層のすべてを一体化したものを指している。
最下位の、つまり、コンピュータ・ハードウェア・レベルには、ディスプレイ・モニタ、キーボード、ポインティング・デバイス（マウスやトラックボールなど）を含む基本的コンピュータおよび関連入出力デバイス、ならびにその他の標準コンポーネント（プリンタやディスク・ドライブなど）が含まれている。次のレベルの「コンポーネント・ドライバ・ソフトウェア」は、デバイス依存ソフトウェアからなり、これは、種々のハードウェア・コンポーネントの動作に必要なコマンドおよび信号を生成する。資源管理と通信層はコンポーネント・ドライバと接続し、ソフトウェア・ルーチン、すなわち、資源を割り振り、アプリケーション相互間で通信し、上位層によって生成された通信内容を下位層に多重通信するソフトウェア・ルーチンを含んでいる。ウィンドウ・マネージャは基本的ウィンドウ操作に対するユーザ・インタフェースを取り扱う。このような基本的ウィンドウ操作としては、ウィンドウの移動およびサイズ変更、ウィンドウの起動と終了、ウィンドウの再ドローと再ペイントなどがある。最後のユーザ・インタフェース層は、ハイレベル機能、すなわち、アプリケーション・プログラムが完全なユーザ・インタフェースを開発するために使用する種々のコントロール（ボタン、スライダ、ボックス、その他のコントロール）を実現するハイレベル機能を提供する。
図２に示す構成はディスプレイ・スクリーン干渉問題を解決しているが、この構成には、アプリケーション・プログラムの全てによって生成されたスクリーン・ディスプレイ要求をウィンドウ・マネージャ218に処理させなければならないという欠点がある。これらの要求は直列的にしか処理できないので、要求はウィンドウ・マネージャへ渡すための待ち行列（queu）に置かれてから、各要求は処理され、端末24上の表示を生成する。多数のウィンドウがスクリーン上に同時に存在するような表示では、ウィンドウ・マネージャ218は表示情報の「ボトルネック」になりやすいので、アプリケーション・プログラム202と216による迅速な表示の変更の妨げになる。ウィンドウがユーザによって移動または再位置付けされたときに起こる、スクリーン再ドローの遅延は、ウィンドウが外見上断片的に作られていると、煩わしくなり、システムのオペレーションからかけ離れることになるので、顕在化することがよくある。
以上に鑑みて、本発明の目的は、アプリケーション・プログラムと接続し、各アプリケーション・プログラムによって生成されたスクリーン表示を高速にかつ効率的に再ドローできるようにするウィンドウ・マネージャを提供することにある。
本発明の別の目的は、アプリケーション・プログラムの全ての表示の生成を調整して、アプリケーション・プログラムが相互に干渉し合ったり、スクリーンの表示上で相互にオーバライトするのを防止するウィンドウ・マネージャを提供することにある。
さらに、本発明の別の目的は、アプリケーション・プログラムが実際のインプリメンテーション詳細にとらわれることなく、単純なコマンド構造によってアプリケーション・プログラムとの対話することができるウィンドウ・マネージャを提供することである。

さらに、本発明の別の目的は、スクリーン表示プロセスに対する詳細な制御を必要とするアプリケーション・プログラム開発者が、フルセットの表示制御コマンドであって、各アプリケーション・プログラムが使用しなくてもよい表示制御コマンドによって、上記制御を達成できるようにするウィンドウ・マネージャを提供することにある。

発明の概要

上述した問題を解消し、上述した目的を達成するために、本発明の図示の実施例によれば、オブジェクト指向のウィンドウ・マネージャは、表示されているウィンドウが変更されるたびに、各アプリケーション・ウィンドウの可視エリア（visible area）を計算し、ストアすることにより、別々のアプリケーション・プログラム間の調整を行う機能を備えている。各アプリケーションはスクリーン・バッファ・メモリと直接にやりとりすることにより、ウィンドウ・マネージャによって計算された可視エリアを使用してそのディスプレイ・エリアに対応するスクリーン部分を再ドローする。

各アプリケーション・プログラムは、柔軟性に飛んだ表示機能を備えたウィンドウ・オブジェクトを作成することによりオブジェクト指向ウィンドウ・マネージャと通信するが、表示機能はアプリケーション・プログラムからは見えないようになっている。ウィンドウ・オブジェクトは、ウィンドウ・マネージャと直接的に接続するためのコマンドと、ウィンドウ・マネージャによって計算された関連の可視エリアを一時的にストアしておくデータ・エリアとを含んでいる。

可視エリアの計算時間を削減する手法のいくつかが使用される。第一は、上述したように、可視エリアのコピーを各ウィンドウ・オブジェクトにストアまたは「キャッシュ」する方法である。このコピーは、アプリケーション・プログラムがウィンドウ・エリアを再ドローする必要があって、可視エリアが変更されていなかったとき使用することができる。

さらに、ウィンドウ・マネージャは、可視エリアの計算を高速化する２つのルーチンのうちの一方を用いて、各アプリケーション・ウィンドウの可視エリアを計算する。そのルーチンの一方は単一のウィンドウだけが変更されたことを想定して、ウィンドウの新しい可視エリアを元の可視エリアと比較して変更エリアを求める。この変更エリアは変更ウィンドウの背後にあるすべてのウィンドウの可視エリアを再計算するために使用される。他方の再計算ルーチンは可視エリアのすべてを再計算するもので、ウィンドウが例えば除去されたとき使用できる。

【図面の簡単な説明】

本発明の上記およびその他の利点をよりよく理解するために、以下、添付図面を参照して説明する。添付図面において、

図１は従来のコンピュータ・システムを示す概要ブロック図であり、アプリケーション・プログラム、オペレーティング・システム、スクリーン・バッファおよびディスプレイ・モニタ間の関係を示している。

図２は図１に示す従来のシステムの改良型を示す概略ブロック図であり、同時に実行されている複数のアプリケーション・プログラムがスクリーン表示を生成することを可能にしている。

図３はコンピュータ・システム、例えば、本発明によるオブジェクト指向ウィンドウ・マネージャが稼働しているパーソナル・コンピュータ・システムを示す概略ブロック図である。

図４は改良型コンピュータ・システムを示す概略ブロック図であり、複数のアプリケーション・プログラムとウィンドウ・マネージャとがスクリーン・バッファでやりとりして、グラフィック情報をディスプレイ・モニタから表示することを示している。

図５は情報経路を示す概略ブロック図であり、アプリケーション・プログラムが本発明によるオブジェクト指向ウィンドウ・マネージャとどのようにやりとりするかを示している。

図６は、ウィンドウ指向のディスプレイをサポートするグラフィカル・ユーザ・インタフェースの代表的な外観を示す概略図であり、ウィンドウのコンポーネントと部分を示している。

図7Aおよび図7Bは、アプリケーション・ウィンドウのサイズが変更されたとき再ドローする必要のあるディスプレイ・スクリーン部分を示す図である。
図８は、アプリケーション・プログラムがオブジェクト指向ウィンドウ・マネージャとやりとりして、情報をディスプレイ・スクリーン上で表示するときのメソッドのフローチャートを示す図である。
図９は、新しいウィンドウを作成するときアプリケーション・プログラムによって使用されるメソッドのフローチャートを示す図である。
図10は、既存のウィンドウをディスプレイから削除するとき使用されるメソッドのフローチャートを示す図である。
図11は、アプリケーション・プログラムが可視エリア情報をオブジェクト指向ウィンドウ・マネージャに要求するときのメソッドのフローチャートを示す図である。
図12は、ウィンドウ・オブジェクトがウィンドウ・マネージャとやりとりすることにより新しいウィンドウを作成するときのメソッドのフローチャートを示す図である。
図13は、ウィンドウ・オブジェクトがウィンドウ・マネージャとやりとりすることによりウィンドウを削除するときのメソッドのフローチャートを示す図である。
図14は可視エリア・マネージャとウィンドウ・マネージャが、選択したウィンドウを他のウィンドウの手前に置くようにウィンドウ・ディスプレイを並べ換えるときのメソッドのフローチャートを示す図である。
図15は、新しいウィンドウがウィンドウ・マネージャに置かれた可視エリア・マネージャによってアクチベートされるときのメソッドのフローチャートを示す図である。
図16は、新しいウィンドウの配列またはサイズが変更されたとき可視エリア・マネージャが新しい可視エリアを計算するときのメソッドのフローチャートを示す図である。
図17は、１つのウィンドウだけが変更されている、ウィンドウの新しい可視エリアを計算するために可視エリア・マネージャによって使用されるメソッドのフローチャートを示す図である。
発明の好適実施例の詳細な説明
本発明は、好ましくは、IBM社PS/2またはアップル社マッキントッシュ・コンピュータなどのパーソナル・コンピュータに内蔵されているオペレーティング・システムを背景に実施される。図３は代表的なハードウェア環境を示しもので、本発明によるコンピュータ300の代表的なハードウェア構成を示している。このコンピュータ300は中央処理ユニット302（これは従来のマイクロプロセッサにすることが可能である）によって制御され、すべてがシステム・バス308を介して相互接続されている複数の他のユニットは特定のタスクを遂行するためのものである。特定のコンピュータは図３に示すユニットの一部だけを装備している場合もあれば、図示していない別のコンポーネントを装備している場合もあるが、大部分のコンピュータは少なくとも図示のユニットを装備している。
具体的には、図３に示すコンピュータ300は、情報を一時的にストアしておくためのランダム・アクセス・メモリ（RAM）306、コンピュータの構成と基本的操作コマンドを永続的にストアしておくためのリードオンリ・メモリ（ROM）304、およびディスク・ユニット313やプリンタ314などの周辺デバイスをそれぞれケーブル315,312を介してバス308に接続するための入出力（I/O）アダプタ310を装備している。また、ユーザ・インタフェース・アダプタ316は、キーボード320などの入力デバイスおよびマウス、スピーカ、マイクロホンなどの他の公知インタフェース・デバイスをバス308に接続するためのものである。ビジュアル出力は、バス308をビデオ・モニタなどのディスプレイ・デバイス322に接続するディスプレイ・アダプタ318から得られる。ワークステーションは、そこに常駐しているアップル・システム/7オペレーティング・システムなどのオペレーティング・システム・ソフトウェアによって制御され、調整されている。
好適実施例において、本発明はオブジェクト指向プログラミング技法を用いてＣ＋＋プログラミング言語で実現されている。Ｃ＋＋はコンパイラ型言語（compiled language）である。つまり、プログラムは人間が理解できるスクリプト（human－readable script）で書かれ、このスクリプトはコンパイラと呼ばれる別のプログラムに渡される。これを受け

て、コンパイラはマシン可読の数値コードを生成する。このコードはコンピュータにロードすることも、コンピュータに直接に実行させることも可能である。下述するように、Ｃ＋＋言語はある種の特性をもっている。この特性によると、ソフトウェア開発者は他人が書いたプログラムを簡単に利用することができると同時に、プログラムの再利用の管理を強化することによりプログラムの破壊や正しくない使用を防止することができる。Ｃ＋＋言語は周知であり、この言語を詳しく説明した論文や文献が多数発表されている。さらに、Ｃ＋＋コンパイラは、Borland International,Inc.やマイクロソフト社などの、数社のベンダから市販されている。従って、以下では、混乱を避けるために、CC＋言語の詳細とＣ＋＋コンパイラの動作については、これ以上詳しく説明することは省略する。

この分野の精通者ならば理解されるように、オブジェクト指向プログラミング（Object－Oriented Programming－OOP）技法では、「オブジェクト」を定義し、作成し、使用し、消去することからなっている。これらのオブジェクトは、データ・エレメントと、そのデータ・エレメントを操作するルーチン、つまり、関数とからなるソフトウェア・エンティティ（software entity）である。データおよび関連の関数は、ソフトウェアによってエンティティとして扱われ、あたかも１つのアイテム（項目）であるかのように、作成し、使用し、削除することができる。データと関数が一緒になって、オブジェクトは、その特性（これはデータ・エレメントで表すことができる）とその振舞い（behavior）（これはそのデータ操作関数で表すことができる）からとらえて、ほとんどどのような実世界のエンティティでもモデル化することができる。このようにすると、オブジェクトは人やコンピュータのような具体物をモデル化することも、数や幾何学的デザインのような抽象的概念をモデル化することもできる。

オブジェクトは「クラス」を作成することにより定義される。クラスとは、それ自体はオブジェクトでないが、実際のオブジェクトをどのように構築するかをコンパイラに指示するテンプレートの働きをするものである。例えば、クラスはデータ変数の数とタイプ、およびデータを操作する関数を実行するのに必要なステップを指定することができる。オブジェクトは、実際には、コンストラクタ（constructor）と呼ばれる特殊な関数によってプログラムの中に作成される。つまり、コンストラクタは対応するクラス定義と、オブジェクト作成期間に渡される引数などの追加情報とを使用して、オブジェクトを構築する。同様に、オブジェクトは、デストラクタ（destructor）と呼ばれる特殊な関数によって破壊（消去）される。オブジェクトはそのデータを使用し、その関数を呼び出すことによって使用することができる。オブジェクト指向プログラミング技法の主な利点は、３つの基本原理、つまり、カプセル化（encapsulation）、多態（polymorphism）および継承（inheritance）に起因している。具体的に説明すると、オブジェクトは、内部データ構造と内部関数のすべてまたは一部を外部から見えないように、つまり、カプセル化するように設計することができる。これを具体的に説明すると、プログラム設計期間に、プログラム開発者は、データ変数のすべてまたは一部および関連関数のすべてまたは一部が「私有（private）」、つまり、そのオブジェクト自身だけが使用するものと扱われるようなオブジェクトを定義することが可能である。その他のデータまたは関数は「公開（public）」、つまり、他のプログラムが使用できるものと宣言することができる。他のプログラムによる私有変数へのアクセスは、そのオブジェクトの私有データをアクセスするオブジェクト用に公開関数を定義することによって制御することができる。公開関数は、私有データと「外部」世界とを結ぶ、制御された統一的インタフェースとなるものである。私有変数を直接にアクセスするプログラム・コードを書くようなことをすると、コンパイラはプログラムのコンパイル時にエラーを引き起こし、そのエラー個所でコンパイル・プロセスは中止されるので、プログラムは実行されないことになる。

多態は、全体のフォーマットが同じであるが、異なるデータを取り扱うオブジェクトと関数が異なる働き方をして、統一的な結果を得られるように概念である。例えば、加算関数は、変数Ａプラス変数Ｂ（Ａ＋Ｂ）と定義することができ、ＡとＢが数であるか、文字であるか、またはドルとセントであるかに関係なく、この同じフォーマットを使用することができる。しかし、加算を実行する実際のプログラム・コードは、ＡおよびＢを構成する

変数のタイプによって大幅に異なることがある。多態によると、変数の各タイプ（数、文字、およびドル）別に３種類の関数定数を書くことができる。関数が定義されたあと、プログラムはいつでも、その共通フォーマット（Ａ＋Ｂ）によってその加算関数を参照することができ、コンパイル時には、Ｃ＋＋コンパイラは、３つの関数のどれが実際に使用されているかを、変数のタイプを調べることによって判断する。そのあと、コンパイラは正しい関数コードを代入する。多態によると、類似の結果を出力する類似の関数をプログラムのソース・コードの中で「グループ化」して、より論理的で明確化されたプログラムの流れを得ることができる。

オブジェクト指向プログラミングの基礎となっている第３の原理である継承によると、プログラム開発者は既存のプログラムを容易に再利用できるので、ソフトウェアを初めから作る必要がなくなる。継承の原理によると、ソフトウェア開発者はクラス（およびそのクラスからあとで作成されるオブジェクト）を、関連するものとして宣言することができる。具体的には、クラスは他の基底クラス（base class）のサブクラスと指定することができる。サブクラスはその基底クラスの公開関数のすべてを「継承する」ので、その関数がサブクラスに置かれているのと同じように、その関数をアクセスすることができる。別の方法として、サブクラスは継承した関数の一部または全部をオーバライドすることも、同じフォーマットの新しい関数を定義するだけで、継承した関数の一部または全部を変更することもできる（オーバライドまたは変更を行っても、基底クラスの関数はめったに変更されず、サブクラスでの関数の使い方が変更されるだけである）。別のクラスの機能の一部を持つ（選択的変更による）新しいクラスを作成すると、ソフトウェア開発者は既存のコードを特定の要求に合わせて容易にカストマイズすることができる。

オブジェクト指向プログラミングは他のプログラミング概念に比べて大幅に改善されているが、特に、変更の対象となる既存ソフトウェア・プログラムがない場合プログラム開発には相当の時間量と作業量が必要である。その結果、従来の手法では、オブジェクト群とその他の雑多なルーチンを作成するための、相互に接続された事前定義クラス群をプログラム開発者のために用意しておく必要があったが、これらのルーチンはいずれも、普通に見られるタスクを特定の環境で実行することを目的としていた。このような事前定義クラスとライブラリは普通「フレームワーク」と呼ばれており、稼働アプリケーション用の事前に作られた構造を提供している。

例えば、ユーザ・インタフェース用のフレームワークは、ウィンドウ、スクロールバー、メニューなどを作成する、事前定義のグラフィック・インタフェース・オブジェクト群を提供すると共に、これらのグラフィック・インタフェース・オブジェクトをサポートする機能と「デフォルト」の行動を提供する。フレームワークはオブジェクト指向技法に基づいているので、事前定義クラスを基底クラスとして使用することができ、さらに、組込みのデフォルト行動を開発者が定義したサブクラスによって継承して、変更またはオーバライドすることにより開発者はフレームワークを拡張し、カストマイズした問題解決手法を特定の専門知識分野で作成することができる。このオブジェクト指向手法が従来のプログラミングに比べて大きな利点となっているのは、プログラマはオリジナル・プオグラムを変更するのではなく、むしろ、オリジナル・プログラムの機能を拡張しようとするからである。さらに、開発者がコード層を始めから最後まで盲目的に作業しないのは、フレームワークにアーキテクチャに関するガイダンスとモデリングが用意されており、それと同時に、開発者は問題分野に対する固有の特定アクションを行わなくてもよいからである。

どのレベルのシステムに関心があるか、どのような問題を解決しようとしているかに応じて、フレームワークには多種類のものが用意されている。フレームワークの種類は、ユーザ・インタフェースの開発を支援するハイレベルのアプリケーション・フレームワークから、通信、印刷、ファイル・システム・サポート、グラフィックスなどの基本的システム・ソフトウェア・サービスを提供するローレベルのフレームワークまでの範囲にわたっている。市販されているアプリケーション・フレームワークの例を挙げると、MacApp（アップル社）、Bedrock（Sysmantec社）、OWL（Borland社）、NeXT Step App Kit（NeXT社）、およびSmalltalk－80 MVC（ParcPlace社）がある。

フレームワーク手法はカプセル化、多態および継承のすべての原理をオブジェクト層に採用しており、他のプログラミング技法を大幅に改善しているが、いくつかの困難は問題が発生している。アプリケーション・フレームワークはモノリシック・オペレーティング・システムに上乗せされた１つ以上のオブジェクト「層」（レイヤ）から構成されているのが一般であり、オブジェクト層に柔軟性があるとしても、もとの扱いずらい手続き型コールによるオペレーティング・システムと直接にやりとりする必要がよく起こっている。

アプリケーション・フレームワークがアプリケーション・プログラム用のプレハブ機能を開発者に提供しているのと同じように、好適実施例に取り入れられているようなシステム・フレームワークも、システム・レベルのサービス用のプレハブ機能を提供することができる。開発者は、これらの機能を変更またはオーバライドすることにより、カストマイズされた問題解決手法を作成することができるので、従来のアプリケーション・フレームワーク・プログラムで必要であった扱いずらい手続き型コールの使用を避けることができる。以下では、アプリケーション・プログラムによって生成された情報を表示するウィンドウを作成、削除および操作するときの土台を提供することができるディスプレイ・フレームワークを例にして説明する。これらの機能を必要とするアプリケーション・ソフトウェア開発者は、通常は、これらの機能を得るために特定のルーチンを書く必要があった。フレームワークを使用してこれを行う場合は、開発者は完成したディスプレイの特性と動作を与えるだけでよく、これらのタスクを実行する実際のルーチンはフレームワークから提供される。

好適実施例ではフレームワークの概念を採用し、アプリケーションとオペレーティング・システムを含むシステム全体にこの概念を適用している。商業的なまたは企業の開発者、システム・インテグレータ、またはOEMにとっては、このことは、MacAppなどのフレームワークについて説明してきた利点のすべては、テキストやユーザ・インタフェースなどの事物についてはアプケーション・レベルで、印刷、グラフィックス、マルチメディア、ファイル・システム、入出力、テストなどのサービスについてはシステム・レベルで生かすことができる。

図４は、本発明のオブジェクト指向ウィンドウ・マネージャを利用するコンピュータ・システムを示す概要図である。コンピュータ・システムは全体が点線枠400で示されており、複数のアプリケーション・プログラム（図にはアプリケーション・プログラム402と418が示されている）とオペレーティング・システム404は、コンピュータのオペレーションを制御し、調整するために用意されたものである。図４を簡略化するために、アプリケーション・プログラム402、418とオペレーティング・システム404とのやりとりは、スクリーン・ディスプレイを取り扱うやりとりに限定されている。図に示すように、アプリケーション・プログラム402と418は両方とも、オペレーティング・システム404のウィンドウ・マネージャ420と接続している。他方、ウィンドウ・マネージャ420は、矢印408で図示するように情報をスクリーン・バッファへ送っている。

しかるに、本発明によれば、図４に示すように、アプリケーション・プログラム402、418も、矢印410と428で図示するように情報をスクリーン・バッファ412へ直接に送っている。以下で詳しく説明するように、アプリケーション・プログラム402、418は表示情報を直接にウィンドウ420へ渡し、ウィンドウ表示が変更されたときは、ストアされた情報をウィンドウ・マネージャ420から取り出している。具体的に説明すると、ウィンドウが変更されると、ウィンドウ・マネージャ420は各ウィンドウの可視エリアを再計算してストアしておく。このストアされた可視エリアはそれぞれのアプリケーション・プログラムによって取り出され、クリッピング領域として使用されて、アプリケーション・プログラムはその中に表示情報をドローイングする。ウィンドウの再ペイントまたはドローイングは、スクリーン・再ペイント速度を向上するためにアプリケーション・プログラムによって同時に実行される。

アプリケーションの表示がディスプレイ・スクリーン上で分離されたままになっているのは、ウィンドウ・マネージャ420が、ウィンドウの可視エリアのいずれもがオーバラップしないようにこれらのエリアを再計算するからである。従って、アプリケーション・プロ

グラム402やアプリケーション・プログラム418のように、各アプリケーション・プログラムがウィンドウ・マネージャ420から送られてきた可視エリアにだけドローイングすれば、スクリーン・バッファによって作られた表示内容がオーバラップしないことになる。表示情報がドローイングされて、スクリーン・バッファ412に入れられると、この情報は矢印414で示すように、ディスプレイ・アダプタ416に渡される。このディスプレイ・アダプタはケーブルまたはバス424を介してディスプレイ・モニタ426に接続されている。

アプリケーション・プログラムとウィンドウ・マネージャとのやりとりは、図５の系統図で詳しく示されている。前述したように、ウィンドウ・マネージャ（図５にボックス510で示されている）は、オブジェクト指向のプログラムである。従って、アプリケーション・プログラム508は「オブジェクト」を作成し操作することによって、ウィンドウ・マネージャと接続する。具体的には、各アプリケーション・プログラムはウィンドウ・オブジェクト（例えば、ウィンドウ・オブジェクト500）を作成して、ウィンドウ・マネージャ510と通信する。そのあと、アプリケーション・プログラム508は矢印502で図示するようにウィンドウ・オブジェクト500と通信する。ウィンドウ・マネージャ自体は、オペレーティング・システムの始動時に作成されるオブジェクトであり、ウィンドウ・オブジェクトが作成されると、ウィンドウ・マネージャ510は関連のウィンドウをディスプレイ・スクリーン上に作成する。

以下で詳しく説明するように、各ウィンドウ・オブジェクト500は小型のデータ・ストアまたは「キャッシュ」エリア（図示せず）を含んでおり、これは関連ウィンドウの可視エリアをストアするために使用される。アプリケーション・プログラムがその関連ウィンドウの１つで情報を再ドローイングする必要が起こると、ウィンドウ・オブジェクトは、まずキャッシュのステータス（状況）をチェックする。キャッシュ・エリアにストアされた情報が変更されていなければ、この情報はウィンドウを再ドローイングするために使用される。キャッシュ・エリアを使用すると、可視エリアをウィンドウ・マネージャ510から取り出す必要がなくなるので、再ドローイング・オペレーションを完了するまでの時間が短縮化される。

多数のウィンドウ・オブジェクトは、多数のウィンドウをディスプレイ・スクリーン上に同時に表示するように同時に作成することができるので、各ウィンドウ・オブジェクトはデータ・ストリーム504を通してウィンドウ・マネージャ510と通信する。データ・ストリーム504は、「ストリーム・オブジェクト」、すなわち、情報をあるオブジェクトから別のオブジェクトへ転送するために必要なソフトウェア・コマンドを収めている「ストリーム・オブジェクト」を作成することにより作られる。例えば、ウィンドウ・オブジェクト500が情報をウィンドウ・マネージャ・オブジェクト510へ転送する必要が起こると、ウィンドウ・オブジェクト500はデータを「ストリーム化」してウィンドウ・マネージャ・オブジェクト510に入れるストリーム・オブジェクトを作成する。同様に、ウィンドウ・マネージャ・オブジェクト510が情報をウィンドウ・オブジェクト500へ送り返す必要が起こると、ウィンドウ・マネージャ・オブジェクト510は、データを「ストリーム化」してウィンドウ・オブジェクト500に入れるストリーム・オブジェクトを作成する。このようなストリーム・オブジェクトはそれ自体周知であるので、詳しく説明することは省略する。データをウィンドウ・オブジェクト500からウィンドウ・マネージャ・オブジェクト510へ伝達するストリーム・オブジェクトと、情報をウィンドウ・マネージャ・オブジェクト510からウィンドウ・オブジェクト500へ伝達するストリーム・オブジェクトは矢印504で代表して示されている。

図５に示すように、ウィンドウ・マネージャ510は３つの主要部分から構成されている。つまり、可視エリア・マネージャ506、共有データ・エリア512およびウィンドウ・リスト514である。可視エリア・マネージャ506は、ウィンドウ・マネージャ510が作成されたときウィンドウ・マネージャ510によって始動される独立タスクである。以下で詳しく説明するように、可視エリア・マネージャは、データ・ディスプレイ・スクリーン上で可視化できるウィンドウの部分の管理を担当する。この目的のために、このマネージャは、ウィンドウまたはそのウィンドウの「手前」に置かれている別のウィンドウが変更されると、

ウィンドウの可視エリアを再計算する。また、ウィンドウの向きが変わったときやサイズが変更されて、「デスクトップ」にあり、予め隠れていたエリアが見えるようになったときに起こるデスクトップの損傷の修復といった、他のいくつかのハウスキーピング・タスクも実行する。

共有データ・エリア512は共有メモリと関連ストレージ内のエリアと、取出しルーチンとからなり、これらは一緒になってウィンドウに関係する情報をストアする。共有データ・エリアはウィンドウ・マネージャによって共有メモリ内に作成され、共有データ・エリアの一部はウィンドウの各々に割り当てられ、可視エリアのバーションを示す「タイムスタンプ」を含む、種々のウィンドウ・パラメータを収めている。

前述したような、可視エリアへアクセスするためのメッセージ・ストリームを使用することを少なくするために、各ウィンドウ・オブジェクト500はローカル「キャッシュ」メモリも持っており、ここには、関連ウィンドウの可視エリアのコピーがストアされている。タイムスタンプもローカル・キャッシュ・メモリにストアされ、このタイムスタンプは、可視エリア、すなわち、ウィンドウ・サーバから取り出された可視エリアの最後のバージョンを示している。アプリケーション・プログラムは再ドローイング・オペレーションを開始するとき、可視エリアをウィンドウ・オブジェクトから要求する。これを受けて、ウィンドウ・オブジェクトはタイムスタンプを共有メモリ・エリアから取り出し、取り出したタイムスタンプをローカル・キャッシュ・メモリにストアされたタイムスタンプと比較する。２つのタイムスタンプの比較の結果、可視エリアが変更されていないことが分かると、ローカル・キャッシュ・メモリにストアされた可視エリアのコピーが再ドローイング・オペレーションのために使用される。逆に、タイムスタンプの比較の結果、ウィンドウ・マネージャが可視エリアを更新していたことが分かったときは、新しい可視エリアを取り出して、ローカル・キャッシュ・エリアにストアしておく必要がある。タイムスタンプを単独で取り出すようにすると、可視エリア全体を取り出す場合よりも高速化するので、ローカル・キャッシュに置かれたコピーが使用できるときは、総再ドローイング時間が削減されることになる。

ウィンドウ・マネージャ510はウィンドウ・リスト514も保全し、これは図示のようにリンク・リストとして実現され、システムに現在存在する各ウィンドウの識別番号を収めている。本発明の好適実施例によれば、各ウィンドウには、ウィンドウの「種類（kind）」が割り当てられている。ウィンドウの種類はその全体が、スクリーン上に置かれているウィンドウの相対的位置に従っている種類階層から選択される。図示している種類階層は次のようになっている（ウィンドウ種類は図示のように、通常最前面のウィンドウ位置に現れるウィンドウ種類から始まっている）。

最前面位置　スクリーン・セーバ
メニューバー
メニュー
ウィンドイド（フローティング・パレットおよび他の同種タイプのウィンドウ用）
ドキュメント
最後方位置　デスクトップ

ウィンドウ・マネージャはスクリーン上に表示されているウィンドウを自動的に保全して、同種類のウィンドウがすべて種類「層」に一緒に置かれるようにする。この配置は、種類層間の区画を示す「プレースホルダ」をウィンドウ・リスト内に挿入することにより行われる。ウィンドウ・マネージャは、これらのプレースホルダの１つに到達するまでウィンドウ・リスト全体にわたって繰り返し、特定の種類層の終わりにいつ到達したかを判断して、新しい種類層の先頭から始めることができる。

前述したように、好適実施例によれば、オペレーティング・システムは複数のタスクを同時に実行させる機能を備えているので、２つ以上のタスクが同時に動作していると、相互にやりとりする可能性がある。この相互のやりとりは、２つ以上のタスクが、共有データ・エリアやウィンドウ・リストなどのような共有資源を同時にアクセスしようとする起こることがある。従って、このようなやりとりを管理し、望ましくない干渉を防止するため

に並行性コントロール（concurrency control）が必要になる。図示の並行性コントロール技法はセマフォ（semaphore）と呼ばれるもので、一実施例で使用されている。セマフォとは、ある資源への並行アクセスの試みを「逐次化」する周知のデバイスである。具体的には、あるタスクがセマフォによって制御されている資源をアクセスするには、その前に、そのタスクはセマフォを「獲得」しなければならない。タスクが資源の使用を終えると、そのセマフォを解放して、他のタスクが獲得できるようにする。一般に、各セマフォは要求待ち行列（キュー）が関連づけられているので、受け付けられないでセマフォを獲得しようとする要求は（セマフォが別のタスクによってすでに獲得されているので）待ち行列上で保留にされる。

現存システムでは、セマフォは複数の異なる共有資源を保護するために使用されている。具体的には、グローバル・ドローイング・セマフォはアプリケーション・プログラムがウィンドウ・マネージャと相互に作用し合うのを防止するために使用されている。可視エリアをアクセスする前に、各アプリケーション・プログラムはドローイング・セマフォを獲得しなければならない。現存システムで使用されているドローイング・セマフォには、２つのモードがある。共有モードと排他モードである。本発明によれば、アプリケーション・プログラムは他のアプリケーション・プログラムと同時にスクリーン・バッファに書き込む場合があるので、ドローイング・セマフォを「共有」モードで獲得しなければならない。アプリケーション・プログラムはウィンドウ・マネージャによってスクリーン・バッファ内で別々に保たれているので、この同時アクセスは問題とならない。しかし、ウィンドウ・マネージャが新しいウィンドウの作成といったオペレーションを行うと、すべてのウィンドウが影響を受けるので、ウィンドウ・マネージャはドローイング・セマフォを「排他」モードで獲得しなければならない。ウィンドウ・マネージャがドローイング・セマフォを排他的に獲得したときは、アプリケーションはそのセマフォを獲得できないので、スクリーン・バッファに書く込むことができない。この排他的オペレーションは、変更の通知があるまでは、アプリケーションがスクリーン・バッファの変更部分にオーバライト（重書き）するのを禁止する。

同じように、グローバル・セマフォはウィンドウ・マネージャ510で共有データ・エリア512を保護するために使用されるが、これは、この共有エリアも共有資源であるためである。異なるアプリケーション・プログラムがウィンドウ・サーバを使用して同時にアクセスすることからウィンドウ・リスト514を保護するために類似のローカル・セマフォが使用される。種々のセマフォの獲得と解放の具体的説明は、プログラムによって使用される実際のルーチンの説明個所で詳しく説明する予定である。

図６は、代表的な「ウィンドウ環境」プログラムによって生成されるスクリーン・ディスプレイを示す図である。ウィンドウ600はボーダ（境界）で囲まれた矩形エリアであり、これは従来と同じように動かし、サイズを変えることができる。ウィンドウ600は、タイトル・バー606とメニューバー604を含んでいるのが普通で、その各々はそれ自体が別のウィンドウを有している場合がある。メニューバー604は周知の方法で操作される複数のプルダウン・メニュー（図示せず）へのアクセスを可能にし、ユーザが種々のファイル、編集および他のコマンドへアクセスすることを可能にする。

タイトルバー606、メニューバー604およびボーダを除いたあとの、ウィンドウ内に残っているエリアは「クライアント」エリアと呼ばれ、ドローイング・プログラムなどのアプリケーション・プログラムがドローまたはペイントすることができるメイン・エリアを構成している。クライアント・エリアは「子」ウィンドウと呼ばれ、メイン・ウィンドウと関連づけられている別のウィンドウを取り囲んでいる場合がある。この場合、メイン・ウィンドウは子ウィンドウとの関係で「親」ウィンドウと呼ばれている。各子ウィンドウには、それが親ウィンドウとなっている１つまたは複数のウィンドウを関連づけることも可能である（以下同様）。

多くのアプリケーション・プログラムはクライアント・エリアを、それぞれが独立に制御される複数の子ウィンドウに細分割する。これらはドキュメント・ウィンドウ622、「ツールバー」または「パレット」ウィンドウ616、場合によっては、ステータス・ライン・

ウィンドウ（図示せず）を含んでいるのが代表例である。ドキュメント・ウィンドウ622は水平および垂直スクロールバー618,614を備えることが可能であり、これらのバーを使用すると、ドキュメント・ウィンドウ内のオブジェクトをスクリーン上で動かすことができる。ドキュメント・ウィンドウ622は子ウィンドウ602,610,620に細分割することが可能であり、これらは相互にオーバラップすることもできる。どの時点でも、通常は、子ウィンドウ602,610,620の１つだけがアクティブ状態にあり、１つのウィンドウだけが入力「フォーカス」（input focus）をもっている。入力フォーカスを持つウィンドウだけが、マウスやキーボードなどの入力デバイスからの入力アクションやコマンドに応答する。キーボード入力に応答するウィンドウは、再位置付けやサイズ変更コマンド以外の入力に応答するので非位置付けウィンドウ（non－positional window）とも呼ばれている。

ツールバー／パレット・ウィンドウ616は、アイコン608,602のように、使用頻度の高い、ある種のプログラムやサブルーチンを開始するときに使用すると便利な複数のアイコン・イメージを含んでいるのが普通である。例えば、アイコン608を選択すると、ボックスをスクリーン上にドローするドローイング・ルーチンを開始することができ、アイコン612を選択すると、スペリング・チェック・ルーチンを開始することができる。この種のツールバーとパレットの操作は一般に周知であるので、詳細説明は省略する。

表示の制御はマウスや他の入力デバイスによって選択されるのが一般的である。マウスは、ウィンドウ・プログラムによってスクリーン上にドローされるカーソルを制御する。カーソルが選択すべきグラフィック・イメージ上に置かれているとき、マウスでボタンをアクチベートすると、アプリケーション・プログラムがこれに応答する。

上述した制御は、一般的には、アプリケーション・プログラムにより移動したり、サイズ変更したりできないけれども、親ウィンドウと子ウィンドウは全体がアプリケーション・プログラムの制御下に置かれているのが普通である。アプリケーション・プログラムが複数のウィンドウを持っているとき、あるいは同時に実行される複数のアプリケーション・プログラムがウィンドウを表示しているときは、あるウィンドウのサイズまたは位置が変更されると、変更されたウィンドウの「下」にあるウィンドウの表示されているまたは可視エリアが変更されることになる。図7Aと図7Bは、あるアプリケーションに関連するあるウィンドウを操作すると、同じアプリケーションおよび他の独立アプリケーションに関連する他のウィンドウの可視エリアがどのように変更されるかを示している。

具体的には、図7Aは、バックグラウンドまたはデスクトップに置かれた３つのウィンドウを示している。これらのウィンドウはオーバラップしている。つまり、ウィンドウ700はバックグラウンドにあり、ウィンドウ702はウィンドウ700の手前にあり、ウィンドウ704はウィンドウ702の手前にある。図7Aに示すように、ウィンドウ704はウィンドウ702,700の一部を隠している。ウィンドウ700,702,704の各々は独立に移動し、サイズ変更できるので、オーバラップしたウィンドウ内のエリアを見えるようにしたり、隠したりできるので、これらのウィンドウの外観を変更することができる。しかし、ウィンドウの外観はオーバラップしているので、選択したウィンドウを変更しても、その影響を受けるのは選択したウィンドウの背後にあるウィンドウだけである。例えば、ウィンドウ704を変更すると、ウィンドウ702と700が影響を受けるが、ウィンドウ700を変更したときは、ウィンドウ702または704はオーバラップしてウィンドウ700の一部を隠しているので、これらのウィンドウはその影響を受けることがない。

図7Bは、図7Aにおいて前面ウィンドウ704のサイズを変更したときの結果を示している。具体的には、図7Bは３つのウィンドウ712,714,706を示しており、これらはそれぞれ図7Aのウィンドウ704,702,700に対応している。しかし、図7Bでは、ウィンドウ712はサイズが変更されており、具体的には、サイズがオリジナル・サイズのウィンドウ704から縮小されている。ウィンドウ712のサイズが縮小されると、以前は全体がウィンドウ712によって隠されていたウィンドウ710のエリア（図には陰影エリアで示されている）が見えるようになる。同様に、ウィンドウ706の陰影部分708も見えるようになる。通常のウィンドウ環境オペレーションによれば、ウィンドウの可視部分だけがペイントされる。従って、エリア708と706はそれらが可視エリアとなったとき再ドローまたは再ペイントしなければなら

ない。この再ドローイングは、前述したようにウィンドウ・マネージャとアプリケーション・プログラムとの間の調整により行われる。
具体的に説明すると、ウィンドウ・マネージャは変更された各ウィンドウと、変更されたウィンドウの背景にあるすべてのウィンドウの新しい可視エリアを計算する。そのあと、ウィンドウ・マネージャは変更されたウィンドウに関連する各アプリケーションに「更新チックル（update tickle）」を送って、その可視エリアの一部を再ドローする必要があることをアプリケーションに通知する。これを受けて、各アプリケーションは、変更された可視エリアに関連するタイムスタンプをウィンドウ・マネージャ内のウィンドウ・サーバから取り出すように、変更されたウィンドウに関連するウィンドウ・オブジェクトに働きかけることになる。
ウィンドウ・オブジェクトは取り出したタイムスタンプを、そのローカル・キャッシュ・メモリにストアされたタイムスタンプと比較する。関連のウィンドウ可視エリアはウィンドウ・マネージャによって更新されているので、タイムスタンプの比較は、ウィンドウ・オブジェクトにキャッシュされている可視エリアが有効でなくなったことを示している。従って、ウィンドウ・オブジェクトは新しい可視エリアを共有データ・メモリから取り出したあと、上述したセマフォ・メカニズムを用いてスクリーン・バッファに直接に書き込むことにより可視エリアを更新する。
新しい可視エリアを再ペイントするプロセスは、図８に示すフローチャートに詳しく示されている。具体的には、再ペイント・ルーチンはステップ800から開始して、ステップ802へ進み、そこで再ペイントすべきウィンドウに関連するウィンドウ・オブジェクトはウィンドウ・マネージャから更新チックルを受け取る。更新チックルは、ウィンドウ・マネージャが図7Aと図7Bに示すようにウィンドウのサイズを変更すると生成される。これを受けて、ウィンドウ・オブジェクトは機能ブロック804でドローイング・セマフォを獲得し、ステップ804に示すように、新しい可視領域に関連するタイプスタンプを共有データ・エリアから取り出す。ステップ806で、取り出されたタイムスタンプはウィンドウ・オブジェクトに既にキャッシュされているタイムスタンプと比較され、キャッシュ内の可視エリアが有効かどうかが判断される。キャッシュ内の可視エリアが有効であるとステップ808で判断されたときは、ルーチンはステップ812へ進む。逆に、キャッシュ内の可視エリアが有効でなければ、ルーチンはステップ810へ進み、そこでウィンドウ・サーバからの新しい可視エリアがウィンドウ・オブジェクトのキャッシュにロードされる。
前述したように、スクリーン・バッファにドローイングする前に、各アプリケーション・プログラムは、ステップ804で示すようにグローバル・ドローイング・セマフォを共有モードで獲得しなければならない。更新領域は、機能ブロック812でスクリーン・メモリに直接に書き込むことによりアプリケーションによって再ペイントされる。再ペイントが終了すると、ルーチンはステップ816へ進み、そこでドローイング・セマフォは解放され、ルーチンはステップ818で終了する。
また、前述したように、ウィンドウ・オブジェクトはウィンドウ・マネージャとやりとりすることにより、新しいウィンドウの作成といった、種々のウィンドウ管理関数を得ることができる。新しいウィンドウを作成するためにウィンドウ・オブジェクトによって使用されるルーチン例は、図９のフローチャートに詳しく示されている。ルーチンはステップ900から開始し、ステップ902へ進み、そこでローカル・セマフォがウィンドウ・オブジェクトによって獲得される。ローカル・セマフォが必要であるのは、ウィンドウ・オブジェクトが複数のアプリケーションによってアクセスされる可能性があるからであり、従って、このセマフォは、ウィンドウ・オブジェクトと種々のアプリケーションとの間の望ましくないやりとりを防止するために使用される。
ローカル・セマフォがステップ902で獲得されると、ルーチンはステップ904へ進み、そこで望ましいウィンドウを定義しているパラメータがウィンドウ・マネージャへ渡されるデータ・ストリーム（図５に矢印504で示されている）上でストリーム化される。さらに、新しいウィンドウを「登録」する要求がウィンドウ・マネージャ内の可視エリア・マネージャに対して行われる。この要求を受けると、可視エリア・マネージャは新しいウィンド

ウを作成する。新しいウィンドウを作成するときのウィンドウ・マネージャのオペレーションは以下で詳しく説明するが、簡単に説明すると、ウィンドウ・マネージャは新しいウィンドウをウィンドウ・リストに追加し、新しいウィンドウを作成するが、その過程で新ウィンドウの識別番号（ID）を生成する。

ウィンドウ・マネージャによって生成されたIDは、ステップ908に示すように、戻りデータ・ストリーム上でストリーム化されてウィンドウ・オブジェクトに返却され、ステップ910で、ウィンドウ・オブジェクトはそのキャッシュ・データ内のウィンドウIDを、データ・ストリームから読み取ったIDにセットする。次に、ルーチンはステップ912へ進み、そこでローカル・セマフォが解放される。ルーチンはステップ914で終了する。

ウィンドウ・オブジェクトは、選択したウィンドウを削除するようにウィンドウ・マネージャに要求することも可能である。このオペレーションにおけるステップは図10に詳しく示されている。図示のフローチャートに示すように、ウィンドウ削除ルーチンはステップ1000から開始し、ステップ1002へ進み、そこでローカル・セマフォが獲得される。次に、ルーチンはステップ1004へ進み、そこで削除しようとするウィンドウのウィンドウIDがローカル・メモリから取り出され、ウィンドウ・マネージャへ渡されるデータ・ストリーム上にストリーム化される。次に、ウィンドウを削除する要求が可視エリア・マネージャに対して行われる（ステップ1006）。そのあと、ローカル・セマフォはステップ1008で解放され、ルーチンはステップ1010で終了する。

前述したように、スクリーン・バッファでドローイングする前に、各ウィンドウ・オブジェクトはそのキャッシュ内の可視エリアが有効であるかどうかをチェックして確かめるが、これは、ウィンドウ・オブジェクト・キャッシュ・メモリにストアされたタイムスタンプとウィンドウ・マネージャが管理する共有データから取り出されたタイムスタンプを調べ、比較することにより行われる。キャッシュ内の可視エリアが有効でなければ、新しい可視エリアがウィンドウ・マネージャ共有データ・エリアから要求される。新しい可視エリアを要求するとき行われるステップは図11に詳しく示されている。具体的には、可視エリア要求ルーチンはステップ1100から開始し、ステップ1102へ進み、そこでローカル・セマフォが獲得される。そのあと、ウィンドウ・オブジェクトは、ステップ1104に示すように、キャッシュ内の可視エリアに関連するタイムスタンプが有効かどうかをチェックする。タイムスタンプに関する判定はステップ1106で行われ、タイムスタンプが最新であり、キャッシュ可視エリアが有効であることを示していれば、ルーチンはステップ1106へ進み、そこでキャッシュされた可視エリアが使用される。次に、ルーチンはステップ1122へ進み、そこでローカル・セマフォは解放され、ルーチンはステップ1124で終了する。

逆に、比較ステップ1106がキャッシュ内のタイムスタンプが最新でないことを示しているときは、新しい可視エリアを取得する要求を可視エリア・マネージャに対して行わなければならない。要求ルーチンはステップ1100から開始し、そこで新しい可視エリアを収容するウィンドウのウィンドウIDがウィンドウ・マネージャへ渡されるデータ・ストリーム上にストリーム化される。そのあと、ルーチンはステップ1122へ進み、そこで新しい可視エリアを取り出す要求が可視エリア・マネージャに対して行われる。ステップ1114で、ウィンドウ・オブジェクトは戻りデータ・ストリームをチェックして、新しい可視エリアが用意されているかどうか確かめる（これは前に置かれたフラグで示される）。新しい可視エリアが用意されていなければ、ウィンドウ・オブジェクトにストアされたキャッシュ内の可視エリアはステップ1120で空にされるか、あるいはクリアされる。そのあと、ルーチンはステップ1122でローカル・セマフォを解放し、ステップ1124で終了する。

逆に、ステップ1116に示すように新しい可視エリアが用意されていれば、ルーチンはステップ1118へ進み、そこで新しい可視エリアが戻りストリームから読み取られ、ウィンドウ・オブジェクトのローカル・キャッシュにストアされる。次に、ルーチンはステップ1122へ進み、そこでローカル・セマフォは解放され、ルーチンはステップ1124で終了する。

図12は、ウィンドウ・オブジェクトから要求があったとき新しいウィンドウを作成するためにウィンドウ・マネージャによって使用されるルーチン例のフローチャートである。ルーチンはステップ1200から開始し、ステップ1202へ進み、そこでウィンドウ・オブジェク

トによって生成され、ウィンドウ・マネージャへ渡すようにストリーム化されたウィンドウ・パラメータが到来データ・ストリームから読み取られる。次に、ステップ1204で、ウィンドウ・マネージャはグローバル・ドローイング・セマフォを排他モードで獲得する。前述したように、ドローイング・セマフォを排他モードで獲得すると、すべてのアプリケーションは、ウィンドウ・マネージャが終了するまでドローすることが禁止される。この並行性制御は、新しいウィンドウが作成中のときアプリケーションがスクリーン・バッファを変更することを防止するものである。このインタロックが必要であるのは、新しいウィンドウが残りのウィンドウの可視エリアを変更するおそれがあり、変更すると、可視エリアの再計算がトリガされることになるからである。

ドローイング・セマフォが獲得されると、ルーチンはステップ1206へ進み、そこで新しいウィンドウが作成されるので、新しいウィンドウIDが生成される。このウィンドウIDは、ステップ1208でウィンドウ・リストに追加される。また、このIDはステップ1210でストリーム化されてウィンドウ・オブジェクトに戻される。新しいウィンドウの作成は、まず表示不能ウィンドウを作成し、次にウィンドウを表示可能にすることにより行われる。ウィンドウの可視性を表示不能から表示可能に変更するステップは、以下で詳しく説明するようにウィンドウ可視エリアの再計算をトリガする。最後に、ステップ1212で、ドローイング・セマフォは解放され、ルーチンはステップ1214で終了する。

図13のフローチャートは、ウィンドウ・オブジェクトからの要求に応答してウィンドウを削除するときウィンドウ・マネージャによって使用されるルーチン例を示している。ルーチンはステップ1300から開始し、ステップ1302へ進み、そこでグローバル・ドローイング・セマフォが排他モードで獲得される。次に、ルーチンはステップ1304へ進み、そこでウィンドウ・リスト・セマフォが獲得される。次に、ステップ1306で、ウィンドウは表示不能にされ、そのあとルーチンはステップ1308へ進み、そこでウィンドウは、そのウィンドウに関連するウィンドウIDを削除することによりウィンドウ・リストから取り除かれる。そのあと、ウィンドウ・リスト・セマフォはステップ1310で解放され、ドローイング・セマフォはステップ1312で解放され、ルーチンはステップ1314へ進んで終了する。

図14は、特定のウィンドウをウィンドウ種類層の先頭に置くためにウィンドウ・マネージャによって使用されるルーチンのフローチャートを示している。ルーチンはステップ1400から開始し、ステップ1402へ進み、そこでウィンドウ・リストが調べられ、先頭に移動しようとするウィンドウと同じ種類の最初のウィンドウが得られる。次に、ステップ1404で、選択されたウィンドウはウィンドウ・リスト内の種類層の先頭に移される。この変更はウィンドウ・リストを調べ、種類プレースホルダ（kind place holder）に到達するまでウィンドウを先頭に向かって移動することにより行われる。ウィンドウを先頭に移動すると、移動したウィンドウの背後にあるウィンドウの可視エリアが影響されることがあるので、ルーチンはステップ1406へ進み、そこで選択したウィンドウの背後にあるウィンドウの可視エリアが再計算される（その方法は下述する）。

次に、ステップ1408で、ウィンドウに関連する更新エリアがウィンドウ・オブジェクトから得られる。この更新エリアはデスクトップ・エリアと比較されデスクトップが「壊されて」いたかどうか、つまり、デスクトップのエリアが隠れていないので、選択したデスクトップ・カラーに再ペイントする必要があるかどうかが判断される。ルーチンはステップ1410では、デスクトプが壊されていたかどうかを、選択したウィンドウの更新エリアをデスクトップ・エリアと数学的に比較することによって判断する。壊されていれば、ルーチンはステップ1412へ進み、そこで損傷が修復される（これは、一般に、デスクトップ損傷エリアをあらかじめ決めたカラーを使用して再ペイントすることにより行われる）。

どちらの場合も、ルーチンはステップ1414へ進み、そこで新しく移動したウィンドウが新しいアプリケーションに関連するものであるか、あるいは最前面のウィンドウになったかどうかの判断が行われる。そうであれば、「フォーカス」スイッチがステップ1416で生成される。このフォーカス・スイッチはメッセージをオペレーティング・システムに送ることにより開始されるのが一般であり、フォーカス・スイッチは選択されたウィンドウのビジュアル変更と関連づけられていることがよくある（例えば、ウィンドウのキャプション

やメニューバーは異なるカラーになることがある）。そのあと、ルーチンはステップ1418へ進んで終了する。

図15は、ウィンドウの可視性を変更するために可視エリア・マネージャによって使用されるルーチン例のフローチャートである。このルーチンはステップ1500から開始し、ステップ1502へ進み、そこで可視性を変更するためにウィンドウ・オブジェクトから受け取った要求が実際に可視性を変更するものかどうかの判断が行われる。例えば、すでに表示不能になっているウィンドウを表示不能にする要求があったときは、変更を行う必要はない。同様に、ウィンドウがすでに表示可能であって、それを表示可能にする要求を受けたときも、変更を行う必要がない。変更の必要がなければ、ルーチンは直接にステップ1516へ移って終了する。

ステップ1502で、可視性の変更が実際に要求されていると判断されると、ルーチンはステップ1504へ進み、そこで「損傷」エリア変数がウィンドウの可視エリアにセットされる。この損傷エリア変数は、以下で説明するようにデスクトップの損傷を修復するときに使用される。次に、ルーチンはステップ1506へ進み、そこでウィンドウ・リストにストアされた可視性変数が新しい可視性を示すように変更される。次に、ステップ1508で、変更されたウィンドウの背後にあるウィンドウの可視エリアは以下で説明するように増分的に再計算される。

次に、ウィンドウ可視ルーチンはステップ1510へ進み、そこでウィンドウが可視にされるものであるかどうかを判断するチェックが行われる。そうでなければ、ステップ1504でセットされた損傷変数は、ウィンドウが表示不能にされるものであり、可視エリア全体が見えなくなるので、正しい「損傷」を表している。従って、ルーチンは1516へ進み、そこで損傷エリアを消去するための要求がウィンドウ・マネージャに対して行われる。

逆に、ステップ1510で、ウィンドウが可視にされるものであると判断されたときは、損傷エリア変数は、損傷エリアの一部が新しく可視になったウィンドウで隠されることを表すように調整する必要がある。この場合には、ルーチンはステップ1512へ進み、そこで新しい可視エリアはそれを再計算することにより得られる。次に、ルーチンはステップ1514へ進み、そこで「損傷」エリアは新しい可視エリアに基づいてリセットされる。具体的には、新しく可視になったウィンドウがデスクトップに損傷を引き起こしていれば、損傷エリア変数は可視エリアと排他的OR（XOR）がとられる。そうでなければ、損傷エリア変数は減算によって新可視エリアだけ減少される。次に、ルーチンはステップ1516へ移り、そこでウィンドウ・マネージャにデスクトップ損傷を消去するように要求する。そのあと、ルーチンはステップ1518へ進んで終了する。

図16と図17は、ウィンドウが変更されたときウィンドウ可視エリアを再計算するために可視エリア・マネージャによって使用される２つのルーチンを示している。図16に示すルーチンを使用すると、変更されたウィンドウの背後にあるすべてのウィンドウの可視エリアを、どのような条件でも再計算することができるが、このルーチンは図17に示すルーチンよりも低速である。図17に示すルーチンは、１つのウィンドウだけがサイズ変更または移動されたとき可視エリアを再計算するために使用される。図17に示すルーチンは高速であるため、計算回数が少なくて済む。図16に示すルーチンは２つの変数AreaAboveとCurrentWindowを使用している。CurrentWindow変数は、可視エリア・マネージャがそこで稼働している「現在のウィンドウ」を表し、AreaAbove変数は、現在のウィンドウよりも「上」かあるいは「前面」近くにあるウィンドウのトータルのエリアを表している。可視エリア・マネージャは可視エリアが実際に変更されたかどうかに関係なく、図16に示すルーチンを使用して、変更されたウィンドウの背後にあるすべてのウィンドウの可視エリアを再計算するが、これは、あるウィンドウの変更が別のウィンドウの可視エリアに影響にしなかったときそれを検出するメカニズムが図16に示すルーチンでは使用されないためである。

具体的には、ルーチンはステップ1600から開始し、ステップ1602へ進み、そこでAreaAbove変数がクリアされる。次に、ステップ1604で、CurrentWindow変数はディスプレイ・スクリーン上に現れる最初のウィンドウまたは前面ウィンドウにセットされる。次に、ルーチンはステップ1606へ進み、そこで新しいCurrentWindow（これは現在最前面ウィンドウに

なっている）が変更されたウィンドウであるかどうかを判断するためのチェックが行われる。CurrentWindowが変更されたウィンドウであれば、ルーチンはステップ1608へ進み、そこでAreaAbove変数を現ウィンドウ・エクステント（ウィンドウのフル・エリア）から減算することによりCurrentWindowの可視エリアが計算される。現ウィンドウが最前面ウィンドウである場合には、AreaAbove変数はクリアされているので（ステップ1602で）、ステップ1608で計算された可視エリアは最前面ウィンドウ・エクステント・エリアであるにすぎない。次に、ルーチンはステップ1610へ進み、そこでAreaAbove変数は、変更された現ウィンドウの影響を反映するように補正される。この補正は、AreaAbove変数および現ウィンドウのエクステントを合体（ユニオン）にAreaAbove変数をセットすることにより行われる。次に、ルーチンはステップ1614へ進み、そこでウィンドウ・リストがチェックされ、現ウィンドウの背後に別のウィンドウがあるかどうかが判断される。もしあれば、CurrentWindow変数はステップ1612で次のウィンドウにセットされ、ルーチンはステップ1606へ戻り新しい現ウィンドウの可視エリアを再計算する。逆に、ステップ1614で残っているウィンドウがなければ、ルーチンはステップ1616で終了する。

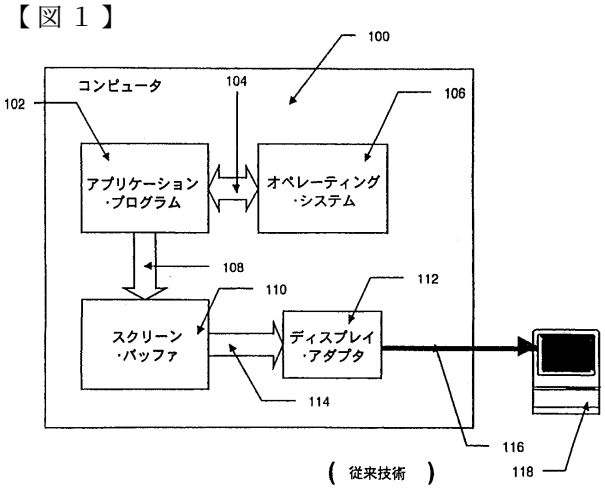
図17示す増分ルーチンは、ある種の場合においてパフォーマンスを向上するようにウィンドウの可視エリアを増分的に計算する方法を示している。この増分方法は、図16に示すようなルーチンを使用して、変更されたウィンドウの可視エリアを最初に計算することから開始される。次に、増分ルーチンは変更ウィンドウの旧可視エリアと新可視エリアの排他的ORを計算する。この「変更エリア」はウィンドウの変更によって影響を受けたエリアを表している。最後に、図17に示す増分ルーチンは影響を受けたまたは変更されたエリアを変更ウィンドウの背後にあるウィンドウの各々に適用する。影響を受けたエリアだけが後続のウィンドウに適用されるので、パフォーマンス向上は２通りの方法で達成することができる。第一に、ウィンドウ・エクステント・エリアが影響を受けたまたは変更されたエリアと交差していなければ、そのウィンドウ可視エリアについてなにも行う必要がない。このことを判断するチェックを行うと、影響を受けなかったウィンドウに関係する計算を行わないで済むことになる。第二に、影響を受けたまたは変更されたエリアが空になっているかどうかを判断するチェックを行うことができる。空になっていれば、ルーチンがリスト上の最後のウィンドウまで到達していなくても、ルーチンを途中で終了させることができる。このルーチンは図17に詳しく示されているが、ステップ1700から開始する。次に、ルーチンはステップ1702へ進み、そこで変更されたウィンドウの旧またはオリジナル可視エリアがセーブされる。次に、ステップ1704で、変更ウィンドウについて新しい可視エリアが計算される。新しい可視エリアの計算は、図16に示すようなルーチンを使用して行うことができる。具体的には、図16に示すルーチンは、現ウィンドウが変更ウィンドウであることをステップ1606が示すまで、ステップ1606－1614からなるループを繰り返すことにより使用できる。この時点で、ステップ1608で計算された可視エリアがステップ1704で使用される。次に、ステップ1706で、“ ChangedArea"と名づけた変数は、ステップ1702でストアされた旧可視エリアとステップ1704で計算された新可視エリアとの排他的ORに等しくなるように計算される。このChangedArea変数は、変更ウィンドウの下にあるウィンドウのすべてを変更するときに使用される。この目的のために、ルーチンはステップ1708へ進み、そこでChangedArea変数が空であるかどうかを判断するためのチェックが行われる。空であれば、ルーチンはステップ1716で終了する。

ステップ1708で、ChangedArea変数が空でないと判断されたときは、ルーチンはステップ1710へ進み、そこでウィンドウ・リストがチェックされ、別のウィンドウがリストに残っているかどうかが判断される。別のウィンドウがあれば、次のウィンドウの可視エリアが、ステップ1712に示すように、ChangedArea変数と次のウィンドウのエクステントとの排他的ORをとることにより計算される。次に、ChangedArea変数は、ステップ1714に示すように、次のウィンドウのエクステントをChangedArea変数から減算することにより更新される。そのあと、ルーチンはリスト内の次のウィンドウを処理するために、まず新しいChangedArea変数が空であるかどうかをステップ1708で判断する。空であれは、ルーチンはステップ1716で終了する。空でなければ、次のウィンドウが処理される。このオペレーシ

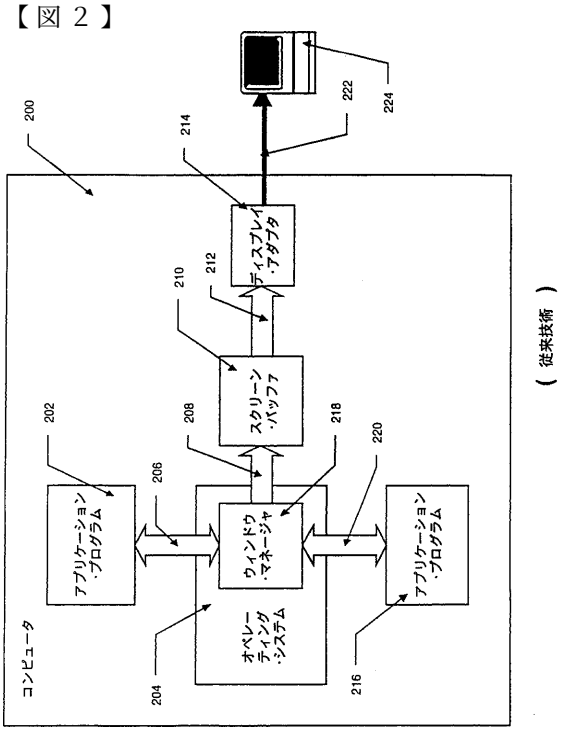
ョンは、ChangedArea変数が空になるか、あるいはウィンドウ・リストに残っているウィンドウがなくなるまで上記のように続けられる。

以上、特定のシステム環境における好適実施例を参照して本発明を説明してきたが、この分野に精通した当業者が理解されるように、本発明は、変更を加えることにより、他の異なるハードウェアおよびソフトウェア環境で実施することが可能であり、このことは請求の範囲に明確化されている本発明の精神と範囲に属することはもちろんである。

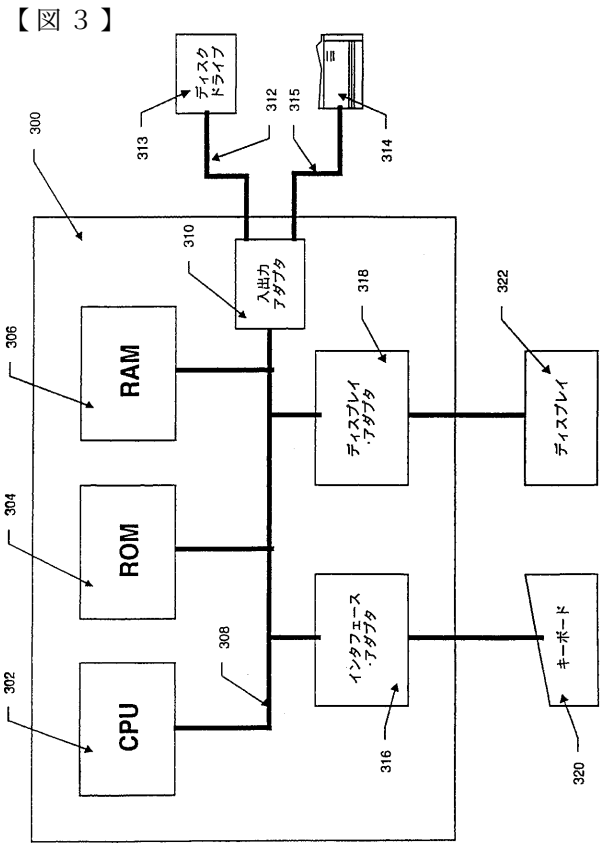
【図１】



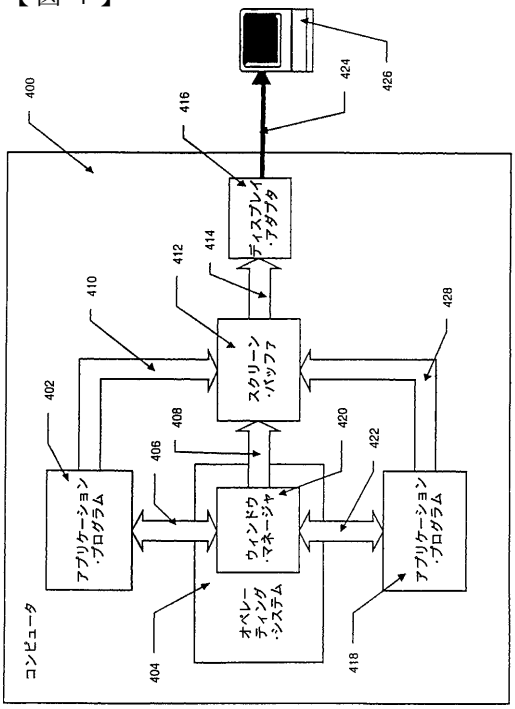
【図２】



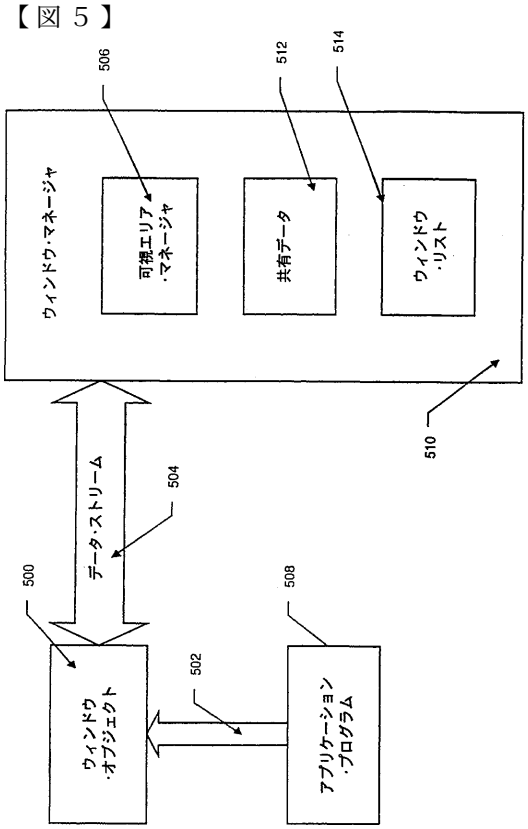
【図３】


ディスクドライブ
313
312
315
314
300
310
入出力アダプタ
318
322
306
RAM
ディスプレイアダプタ
ディスプレイ
304
ROM
308
インタフェースアダプタ
キーボード
302
CPU
316
320

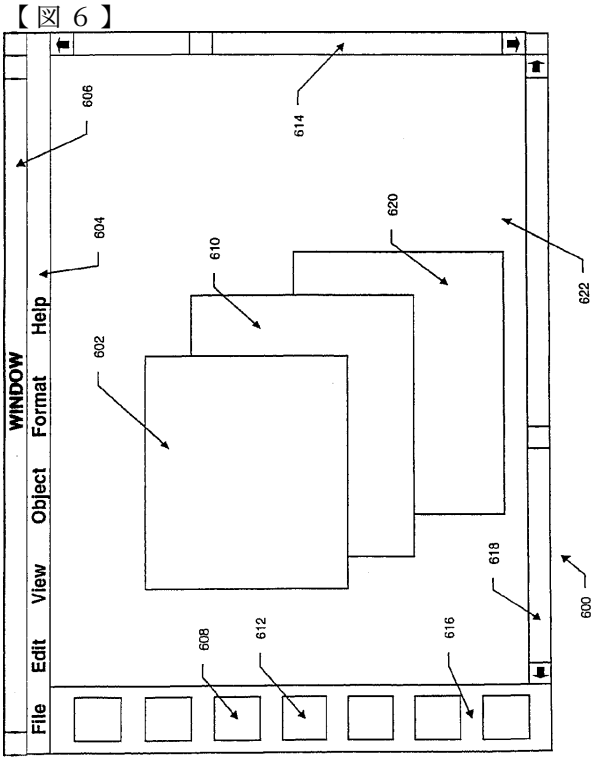
【図４】


400
416
424
426
ディスプレイアダプタ
412
414
410
428
スクリーンバッファ
402
408
420
406
422
アプリケーションプログラム
ウィンドウマネージャ
アプリケーションプログラム
オペレーティングシステム
404
418
コンピュータ

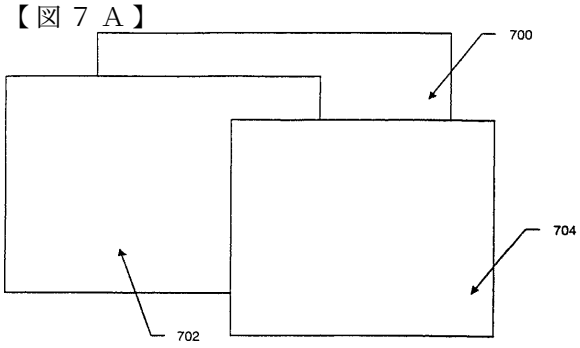
【図５】


ウィンドウ・マネージャ
506
可視エリアマネージャ
512
共有データ
514
ウィンドウリスト
510
データ・ストリーム
504
500
508
ウィンドウオブジェクト
502
アプリケーションプログラム

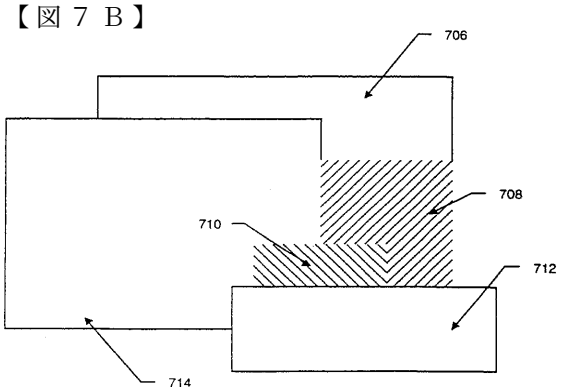
【図６】


614
606
620
604
610
622
602
WINDOW
File Edit View Object Format Help
618
600
608
612
616

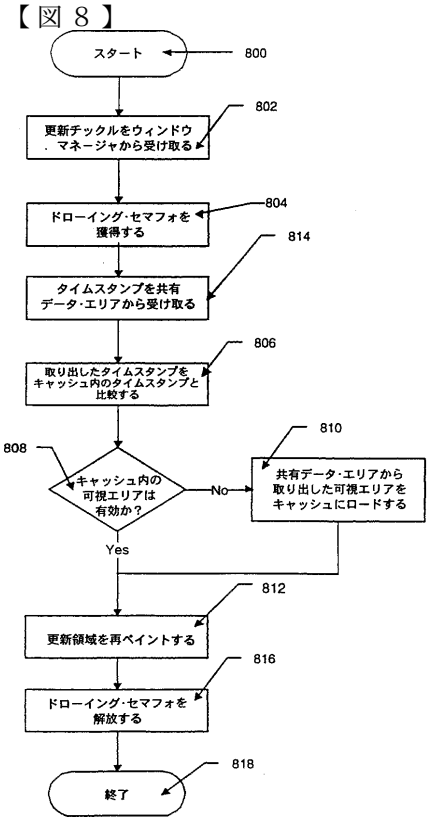
【図７Ａ】



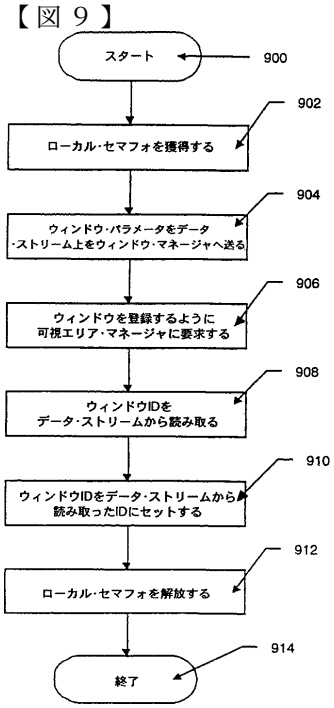
【図７Ｂ】



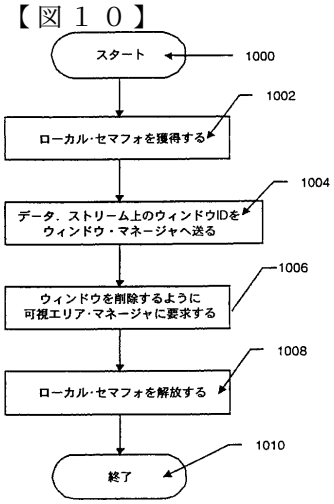
【図８】



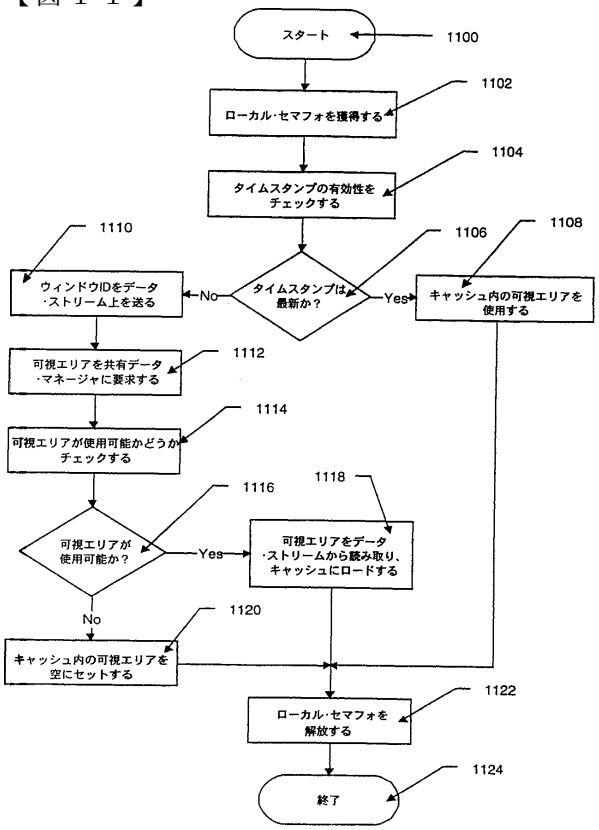
【図９】



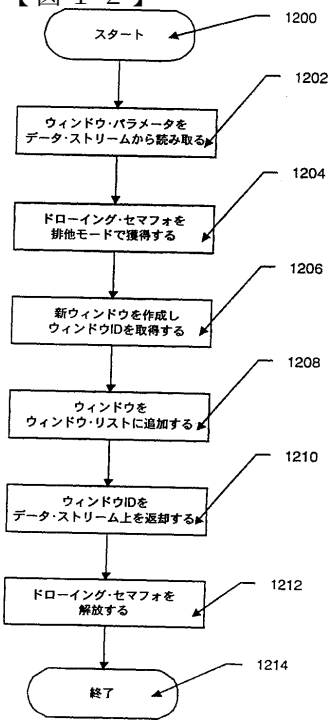
【図１０】



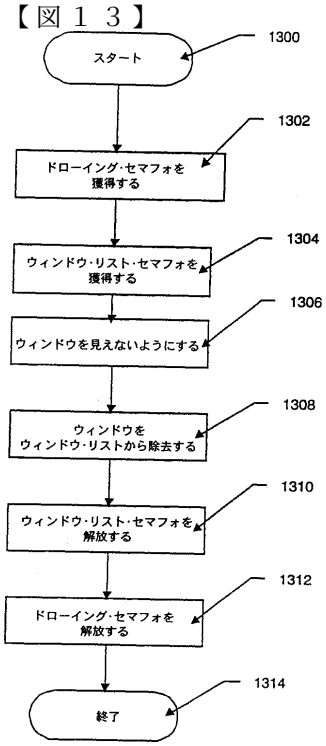
【図１１】



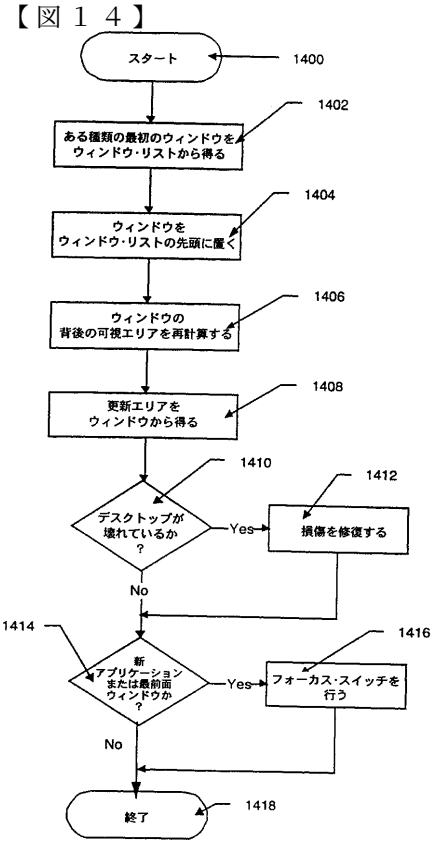
【図１２】



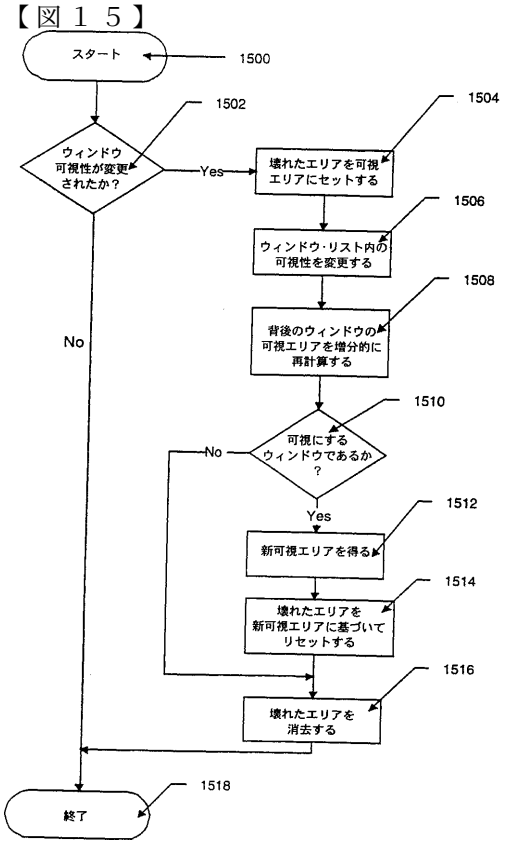
【図１３】



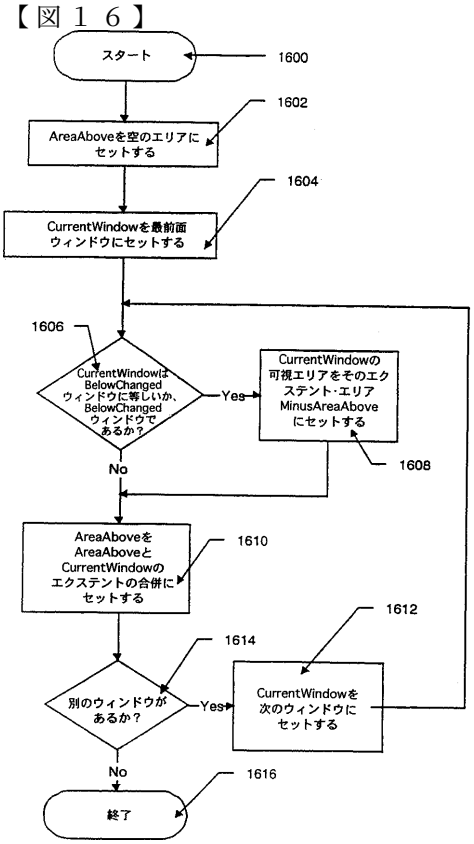
【図１４】



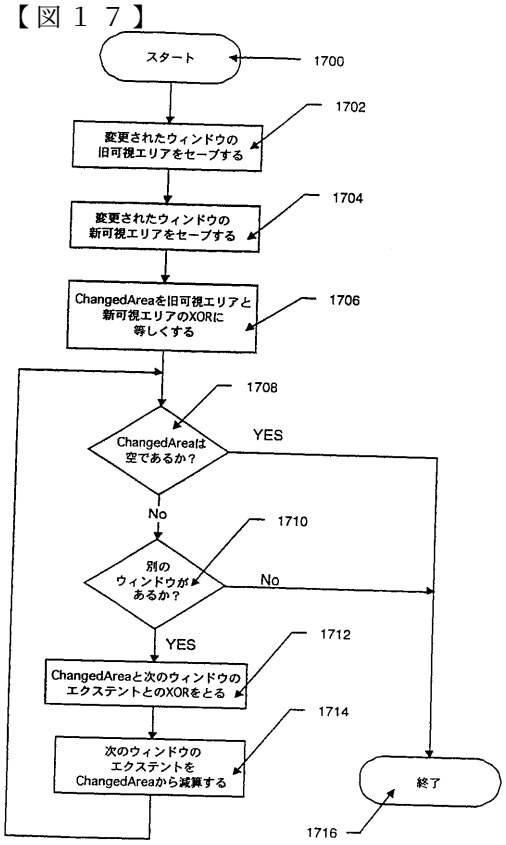
【図１５】


スタート
1500
ウィンドウ可視性が変更されたか？
1502
Yes
壊れたエリアを可視エリアにセットする
1504
ウィンドウ・リスト内の可視性を変更する
1506
背後のウィンドウの可視エリアを増分的に再計算する
1508
No
可視にするウィンドウであるか？
1510
No
Yes
新可視エリアを得る
1512
壊れたエリアを新可視エリアに基づいてリセットする
1514
壊れたエリアを消去する
1516
終了
1518

【図１６】


スタート
1600
AreaAboveを空のエリアにセットする
1602
CurrentWindowを最前面ウィンドウにセットする
1604
1606
CurrentWindowはBelowChangedウィンドウに等しいか、BelowChangedウィンドウであるか？
Yes
CurrentWindowの可視エリアをそのエクステント・エリアMinusAreaAboveにセットする
1608
No
AreaAboveをAreaAboveとCurrentWindowのエクステントの合併にセットする
1610
1612
1614
別のウィンドウがあるか？
Yes
CurrentWindowを次のウィンドウにセットする
No
終了
1616

【図１７】


スタート
1700
変更されたウィンドウの旧可視エリアをセーブする
1702
変更されたウィンドウの新可視エリアをセーブする
1704
ChangedAreaを旧可視エリアと新可視エリアのXORに等しくする
1706
1708
ChangedAreaは空であるか？
YES
No
1710
別のウィンドウがあるか？
No
YES
1712
ChangedAreaと次のウィンドウのエクステントとのXORをとる
1714
次のウィンドウのエクステントをChangedAreaから減算する
終了
1716

フロントページの続き

審査官　後藤　亮治

(56)参考文献　特表昭５９－５０１８４２（ＪＰ，Ａ）
特開昭６２－０９１９８７（ＪＰ，Ａ）
特開昭６１－２９６３８４（ＪＰ，Ａ）
欧州特許出願公開第２１２０１６（ＥＰ，Ａ１）
特開昭６２－２８８９８４（ＪＰ，Ａ）
特開平０４－３６１３２２（ＪＰ，Ａ）
米国特許第５１３３０７５（ＵＳ，Ａ）

(58)調査した分野(Int.Cl.[7]，ＤＢ名)
G09G 5/14
G06F 3/14