

(19) 日本国特許庁(JP)

(12) 特 許 公 報(B2)

(11) 特許番号
特許第4694355号
(P4694355)

(45) 発行日 平成23年6月8日(2011.6.8)

(24) 登録日 平成23年3月4日(2011.3.4)

(51) Int.Cl.

F I

G O 6 T 13/00 (2011.01)

G O 6 T 13/00 B

G O 6 T 13/80 (2011.01)

請求項の数 13 (全 38 頁)

(21) 出願番号	特願2005-334113 (P2005-334113)	(73) 特許権者	500046438
(22) 出願日	平成17年11月18日 (2005.11.18)		マイクロソフト コーポレーション
(65) 公開番号	特開2006-146924 (P2006-146924A)		アメリカ合衆国 ワシントン州 9805
(43) 公開日	平成18年6月8日 (2006.6.8)		2-6399 レッドモンド ワン マイ
審査請求日	平成20年11月18日 (2008.11.18)		クロソフト ウェイ
(31) 優先権主張番号	10/992,462	(74) 代理人	100140109
(32) 優先日	平成16年11月18日 (2004.11.18)		弁理士 小野 新次郎
(33) 優先権主張国	米国 (US)	(74) 代理人	100089705
			弁理士 社本 一夫
		(74) 代理人	100075270
			弁理士 小林 泰
		(74) 代理人	100080137
			弁理士 千葉 昭男
		(74) 代理人	100096013
			弁理士 富田 博行

最終頁に続く

(54) 【発明の名称】 コンピュータディスプレイ出力におけるアニメーションおよびメディアの調整

(57) 【特許請求の範囲】

【請求項1】

コンピューティングシステムのユーザに対して表示するグラフィカル情報又は他のオーディオビジュアル情報を処理するコンピューティングシステムにおいて、時間をかけて個別に各アニメーションの変化を定義するのではなく、異なるアニメーションプロパティの各種タイミングを協調させるシングルタイムラインを用いて、アニメ化される1つ又は複数のオブジェクトの異なるアニメーションプロパティの各種タイミングを協調することによって、より直接的な手法で互いが同期化されたアニメーションの作成を可能にする方法であって、

前記方法は、

高レベルコンポジションとアニメーションエンジンが、アプリケーションプログラム又は他の高レベルプログラムによって提供されるグラフィックシーンを表す階層的エレメントツリーを生成するステップであって、前記ツリーの各ノードは、前記グラフィックシーンにおいてアニメ化されるオブジェクトのための1つ又は複数のアニメーションコンポーネントを表すステップと、

前記高レベルコンポジションとアニメーションエンジンが、前記グラフィックシーンにおいてアニメ化される前記1つ又は複数のオブジェクトの前記異なるアニメーションプロパティのための前記協調されたタイミングと同期化を表す階層的クロックツリーを生成するステップであって、前記階層的クロックツリーの各ノードは、オブジェクトの特定のアニメーションプロパティのタイミングを表し、前記階層的クロックツリーの各レベルは互

いに同期状態であるステップと、

前記高レベルコンポジションとアニメーションエンジンが、エレメントツリーのリンクされたアニメーションコンポーネントのタイミングを表す前記クロックツリーの前記ノードの少なくとも一部を有する前記グラフィックシーンにおいてアニメ化されるオブジェクトのアニメーションコンポーネントを表す前記エレメントツリーの前記少なくとも一部をリンクすることによって前記エレメントとクロックツリーを統合し、次に、異なるアニメーションコンポーネントの異なるプロパティに対する前記タイミングが、前記クロックツリーの前記異なるレベルとノードで提供される前記タイミングによって制御され同期化され、さらに、前記クロックツリーにおける最上位レベルで提供される前記タイミングによって最終的に同期化されるように各種アニメーションコンポーネントに影響を与える他の独立タイムラインの協調を提供するステップと、

10

前記高レベルコンポジションとアニメーションエンジンが、イベントリストジェネレータの形式で状態マシンを用い、一旦、前記クロックツリーが前記エレメントツリーと統合されると、前記クロックツリーからイベントリストを生成するステップであって、前記イベントリストは、独立した各アニメーション又は各アニメーションセットに相当するイベントリストが存在するように前記リンクされたアニメーションコンポジションと前記2つのツリーのタイミングクロックによって最初にスケジューリングされたイベントをグループ化するステップと、

前記高レベルコンポジションとアニメーションエンジンが、インターバルジェネレータを用いて、インターバルリストにコンパイルされる対応インターバルで前記イベントリストの各アニメーション又は各アニメーションセットを準備するステップと、

20

低レベルコンポジションとアニメーションエンジンが、前記インターバルリストを受信し、前記インターバルリストを低レベル計算エンジンに入力し、当該低レベル計算エンジンが、グラフィックサブシステムが前記アニメ化されるシーンをレンダリングし表示するのに用いる出力を生成するステップと

を含むことを特徴とする方法。

【請求項2】

前記エレメントツリーの前記リンクされたアニメーションコンポーネントのタイミングを表す前記クロックツリーの前記ノードの少なくとも一部を有するオブジェクトのアニメーションコンポーネントを表す前記エレメントツリーの前記ノードの少なくとも一部をリンクすることによって前記エレメントとクロックツリーを統合する前記ステップは、拡張アプリケーションマークアップ言語を用いて実行されることを特徴とする請求項1に記載の方法。

30

【請求項3】

前記クロックツリーの前記ノードの少なくとも一部は、エレメントに関連付けられたスタイルからのタイミングデータを表すことを特徴とする請求項1に記載の方法。

【請求項4】

一旦、前記クロックツリーが前記エレメントツリーと統合されると、前記クロックツリーからイベントリストを生成する前記ステップは、全体のストーリーボードタイムライン情報を維持するクロックオブジェクトのインスタンスを作成するステップをさらに含むことを特徴とする請求項1に記載の方法。

40

【請求項5】

全体のストーリーボードタイムラインを維持する前記クロックオブジェクトのインスタンスを作成する前記ステップは、トリガーに応答して前記ストーリーボードタイムラインを開始することによって実行されることを特徴とする請求項4に記載の方法。

【請求項6】

前記トリガーは、プロパティ・トリガーに相当し、前記ストーリーボードタイムラインを開始することは、前記プロパティ・トリガーの状態を評価することをさらに含むことを特徴とする請求項5に記載の方法。

【請求項7】

50

前記トリガーは、イベント・トリガーに相当し、前記ストーリーボードタイムラインを開始することは、前記イベント・トリガーによって起こされるイベントを検知することをさらに含むことを特徴とする請求項 5 に記載の方法。

【請求項 8】

前記トリガーは、エンター／エグジット・トリガーに相当し、前記ストーリーボードタイムラインを開始することは、エンターイベントのプロパティ・トリガーの状態を評価することをさらに含むことを特徴とする請求項 5 に記載の方法。

【請求項 9】

前記階層的エレメントツリーのコンプレックス・プロパティ・パスにおけるターゲットエレメントのサブプロパティを変更するステップをさらに含むことを特徴とする請求項 1

10

【請求項 10】

前記サブプロパティを変更するステップは、前記ターゲットエレメントのプロパティ値を確立し、前記サブプロパティ値を変更するステップをさらに含むことを特徴とする請求項 9 に記載の方法。

【請求項 11】

前記ターゲットエレメントの前記プロパティ値を確立するステップは、競合解消順序付けに基づいて前記プロパティ値を検索するステップをさらに含むことを特徴とする請求項 10 に記載の方法。

【請求項 12】

20

前記サブプロパティ値を変更するステップは、前記サブプロパティ値を変更する基礎値を決定するステップをさらに含むことを特徴とする請求項 9 に記載の方法。

【請求項 13】

コンピューティングシステムのユーザに対して表示するグラフィカル情報又は他のオーディオビジュアル情報を処理するコンピューティングシステムにおいて、時間をかけて個別に各アニメーションの変化を定義するのではなく、異なるアニメーションプロパティの各種タイミングを協調させるシングルタイムラインを用いて、アニメ化される 1 つ又は複数のオブジェクトの異なるアニメーションプロパティの各種タイミングを協調することによって、より直接的な手法で互いが同期化されたアニメーションの作成を可能にする方法を前記コンピューティングシステムに実行させるコンピュータ実行可能命令を記録するコンピュータ読み取り可能な記録媒体であって、

30

前記方法は、

高レベルコンポジションとアニメーションエンジンが、アプリケーションプログラム又は他の高レベルプログラムによって提供されるグラフィックシーンを表す階層的エレメントツリーを生成するステップであって、前記ツリーの各ノードは、前記グラフィックシーンにおいてアニメ化されるオブジェクトのための 1 つ又は複数のアニメーションコンポーネントを表すステップと、

前記高レベルコンポジションとアニメーションエンジンが、前記グラフィックシーンにおいてアニメ化される前記 1 つ又は複数のオブジェクトの前記異なるアニメーションプロパティのための前記協調されたタイミングと同期化を表す階層的クロックツリーを生成するステップであって、前記階層的クロックツリーの各ノードは、オブジェクトの特定のアニメーションプロパティのタイミングを表し、前記階層的クロックツリーの各レベルは互いに同期状態であるステップと、

40

前記高レベルコンポジションとアニメーションエンジンが、エレメントツリーのリンクされたアニメーションコンポーネントのタイミングを表す前記クロックツリーの前記ノードの少なくとも一部を有する前記グラフィックシーンにおいてアニメ化されるオブジェクトのアニメーションコンポーネントを表す前記エレメントツリーの前記少なくとも一部をリンクすることによって前記エレメントとクロックツリーを統合し、次に、異なるアニメーションコンポーネントの異なるプロパティに対する前記タイミングが、前記クロックツリーの前記異なるレベルとノードで提供される前記タイミングによって制御され同期化さ

50

れ、さらに、前記クロックツリーにおける最上位レベルで提供される前記タイミングによって最終的に同期化されるように各種アニメーションコンポーネントに影響を与える他の独立タイムラインの協調を提供するステップと、

前記高レベルコンポジションとアニメーションエンジンが、イベントリストジェネレータの形式で状態マシンを用い、一旦、前記クロックツリーが前記エレメントツリーと統合されると、前記クロックツリーからイベントリストを生成するステップであって、前記イベントリストは、独立した各アニメーション又は各アニメーションセットに相当するイベントリストが存在するように前記リンクされたアニメーションコンポジションと前記2つのツリーのタイミングクロックによって最初にスケジューリングされたイベントをグループ化するステップと、

10

前記高レベルコンポジションとアニメーションエンジンが、インターバルジェネレータを用いて、インターバルリストにコンパイルされる対応インターバルで前記イベントリストの各アニメーション又は各アニメーションセットを準備するステップと、

低レベルコンポジションとアニメーションエンジンが、前記インターバルリストを受信し、前記インターバルリストを低レベル計算エンジンに入力し、当該低レベル計算エンジンが、グラフィックサブシステムが前記アニメ化されるシーンをレンダリングし表示するのに用いる出力を生成するステップと

を含むことを特徴とする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】

20

【0001】

本発明は、一般にコンピュータシステムに関し、より詳細には、コンピュータシステム上に表示するためにグラフィカルな情報および他のオーディオビジュアル情報を処理することに関する。

【背景技術】

【0002】

現代のコンピューティングシステムでは、マークアップ言語技術の発展の結果、プログラム作成者が貴重な情報をユーザに提供し、ユーザと表示された情報との対話に回答して要求などを受け取るための簡単な方法が生まれた。一般に、マークアップが処理されて、エレメントツリーが構築され、次いでエレメントツリーが処理されて、対応する出力データがディスプレイ上にレンダリングされる。

30

【0003】

とりわけ、一般にプログラムの作成を単純化するように構成されている1つのマークアップ言語、すなわちXAML（拡張可能アプリケーションマークアップ言語：eXtensible Application Markup Language）について記載している（本発明の譲受人に譲渡された）複数の特許出願の中に（特許文献1）および（特許文献2）がある。一般にプログラム作成者は、XAMLの命令をアプリケーションプログラミングインターフェース（API）に対する呼び出しと共に使用して、オブジェクトをシーングラフデータ構造内に配置する。例えばユーザインターフェースを構築するために、XAMLベースのプログラムは通常、子ビジュアルオブジェクトを他のビジュアルオブジェクトに追加して、階層型のシーングラフを構築し、出力用のジオメトリデータ、イメージデータ、および他のデータなどの命令リストを作成し、ビジュアルオブジェクト上の変形プロパティ、クリッピングプロパティ(clipping property)、および不透明度プロパティを指定することができる。

40

【0004】

前述の特許出願に記載されているXAMLの機能の1つは、プログラムがアニメーションパラメータおよび／またはメディア関連データを指定できるようにし、対応するアニメーションおよび／またはメディアを含む対応するディスプレイデータを効率よく出力する機能である。例えばプログラム作成者は、表示されるオブジェクトの何らかのプロパティが時間の経過と共にどれぐらい変化すべきかなどのパラメータの点から所望のアニメーシ

50

ョン効果について記述することができ、例えば長方形の不透明度プロパティは、一定の時間にわたって不透明から透明へと徐々に薄まり、そして戻るように設定することができる。典型的な状況では、このシステムは、効果を達成するために時間の経過と共に変更されたイメージを定期的に描き直すことをプログラムコード自体に要求するのではなく、プログラムコードが最初にアニメーションパラメータを指定するだけですむようにアニメーションを管理する。

【0005】

この目的のために、一実装形態では、オペレーティングシステムが、時間の経過に伴うアニメーションの展開を定義するタイムラインオブジェクト(timeline object)の使用を可能にし、必要に応じて、そのアニメーション用のタイムラインを表すためにクロックオブジェクトがインスタンス化される。動画化されたイメージをその更新された特性によって描き直すためにアプリケーションプログラムを定期的に（例えば、フレームごとに1回）コールバックする代わりに、より低いレベルのグラフィックスコンポーネントがクロックを使用して、動画化されて表示されているオブジェクト用の1つまたは複数の更新された、時間と共に変化するプロパティパラメータを確定し、その更新されたプロパティデータに基づいてイメージを描き直す。一般に、メディアも同様に処理される。

【0006】

【特許文献1】米国特許出願第10／693, 630号明細書

【特許文献2】米国特許出願第10／693, 822号明細書

【特許文献3】米国特許出願第10／401, 717号明細書

【特許文献4】米国特許出願第10／693, 438号明細書

【発明の開示】

【発明が解決しようとする課題】

【0007】

こうした簡単なプログラミングは、このようにアニメーションおよびメディアをユーザーインターフェースの一部として実装するための改良された簡易な方法をプログラム作成者に提供するが、これまではタイミング情報をそれぞれのアニメーションまたはメディアエレメントに個々に適用しなければならなかった。結果として、これまでプログラム作成者は、別々のタイムライン上で発生する可能性のある様々なアクション（例えば、アニメーションおよびメディアの開始時刻および停止時刻、ならびに／または受け取られるイベントがどのように処理されるか）を手動で同期させる必要があった。これは、プログラム作成者が、1つの動画化されたプロパティと別の動画化されたプロパティの双方を同期させたいにもかかわらず、一方が他方に（少なくとも一時的に）先行するなど、整合しない状態につながる場合がある。これはまた、複雑さが増すことにもつながり、例えばイベントを再開する際に、そのイベントに関連する個々のアニメーションをそれぞれ見つけて再開する必要がある。

【課題を解決するための手段】

【0008】

簡潔に言えば、本発明は、ストーリーボードと呼ばれる名前付きのタイムラインオブジェクト内でアニメーションのセットをグループ化することによって、それらのセットを作成者がスケジュールできるようにする方法およびシステムを提供する。一実装形態では、ストーリーボードAPIセットは、様々なオブジェクトおよびプロパティに影響を与えるタイムラインを単一のタイムラインツリーに統合して、プログラム作成者がタイミングイベントの複雑なセットを簡単な方法で整理および制御できるようにする方法を提供する。ストーリーボードは、アニメーション／ビデオ／オーディオの特定のインスタンス上で、または特定のスタイルのアニメーションに対して使用することができる。

【0009】

一実装形態では、ストーリーボードは、タイムラインツリーをエレメントツリー内のエレメント用に指定できるようにするオブジェクトを含む。ツリー内のそれぞれのタイムラインは、エレメントのプロパティ値を設定し、エレメントおよびそのサブツリー用のアニ

10

20

30

40

50

メーションを制御する。ストーリーボードは基本的に、開始すること、停止または一時停止すること、あるいは検索するために使用することが可能な特定のタイプのタイムラインオブジェクトに対応し、これによって、動画化されたエレメントの外観、および／またはそのストーリーボードに関連付けられているメディアの動作を制御する。

【0010】

ストーリーボードは、トップレベルのタイムラインのコレクションを含むこともでき、これは、1つのエレメントに関連付けることができ、かつ独立して開始することができる。例えば作成者は、2つの異なるアニメーションを同一のストーリーボードコレクション内の1つのエレメントに関連付けることができ、それらのアニメーションは、独立してトリガできるが、1つのユニットとして含まれている。したがってストーリーボードは、タイミ

10

【0011】

ストーリーボードはまた、状態の変化、イベント、または他のアクションによってアクティブ化できるトリガのコレクションを含むこともでき、そのトリガがアクティブ化されたときに実行される拡張可能なアクションのセット（例えば、タイムラインの開始または停止）を含むこともできる。ストーリーボードは、ユーザがストーリーボード内のタイムラインをコードから対話式に開始および停止できるようにする、基礎をなすオブジェクトモデルの一部である。

20

【0012】

一般にストーリーボード内のタイムラインは、様々なサブツリー、エレメント、および他のオブジェクトに適用できるアニメーションテンプレートを表している。ストーリーボードは、ツリー内のフレームワークエレメント上に直接設定するか、またはスタイル（ボタンの色、サイズ、および他のプロパティなど、エレメントに適用できる1つまたは複数のデータ値を含むリソース）を介してフレームワークエレメントに適用することができる。オリジナルの情報を複製する、チェンジャブル(changeable)などのクローニングと、それに続く複製された情報上での動作とをサポートするオブジェクトにアニメーションを適用するためにストーリーボードを使用することもでき、これによってオリジナルの情報を復元することができる。

30

【0013】

したがってストーリーボードは、様々なオブジェクトおよびプロパティに影響を与える、その他の点では独立したタイムラインを単一のタイムラインツリーに統合して、プログラム作成者がタイミ

40

【0014】

ストーリーボードは、動画化するオブジェクトおよびプロパティと、動画化する元となる基本値とを作成者が指定できるようにするプロパティに関連付けられている。ストーリーボードは、作成者が、時間の経過に伴うそれぞれのアニメーションの変更を個々に定義する代わりに、アニメーションのセットをグループとして整理および制御することを可能にし、これによって、並列アニメーション(parallel animation)およびアニメーションのシーケンス（後述する）など、複雑な方法で相互に同期するアニメーションを簡単に作成することを可能にする。

【0015】

ストーリーボードは、プロパティトリガおよびイベントトリガを使用して、それらのオペレーション（例えば開始、一時停止、検索、および停止のアクション）を制御すること

50

を可能にする。一般にプロパティトリガは、真や偽などの何らかの状態／値に対応し、その一方でイベントが始動する。例えばプロパティトリガは、マウスがエレメント上に存在していた場合は真の値を、エレメント上に存在していなかった場合は偽の値を有することができ、イベントは、マウスがエレメント上でクリックされた場合に始動することができる。それぞれの真または偽の状態および／またはイベントは、（１つまたは複数の）ストーリーボード内で別々に開始、停止、一時停止、および／または検索を行うために使用することができる。このようにして、アニメーションは、ユーザと表示されたエレメントツリーの表示との対話を介して制御することができる。イベントトリガに加えて、プロパティ入力／終了トリガ(property enter/exit trigger)も、タイムラインをトリガする際に使用することができる。

10

【００１６】

ストーリーボードおよびトリガは、スタイル付けすることができ、スタイルは作成者が、１つまたは複数のプロパティを表すデータをグループとしてまとめ、次いでこれを、エレメントの対応するプロパティに１つのセットとして適用することを可能にする。スタイル内のこれらのプロパティは、動画化するか、および／またはトリガに対応することができる。

【００１７】

他の利点および有利性は、以降の詳細な説明を図面と併せて理解すれば、明らかになるであろう。

【発明を実施するための最良の形態】

20

【００１８】

（典型的な動作環境）

図１は、本発明を実装できる適切なコンピューティングシステム環境１００の一例を示している。このコンピューティングシステム環境１００は適切なコンピューティング環境の一例にすぎず、本発明の使用または機能の範囲に対して何らかの限定を提示することを意図するものではない。またコンピューティングシステム環境１００が、この典型的な動作環境１００内に示されたコンポーネントの任意の１つまたは組合せに関して何らかの依存性または必要性を有すると解釈すべきでもない。

【００１９】

本発明は、他の多くの汎用または専用のコンピューティングシステム環境または構成と共に使用することができる。本発明と共に使用するのに適する可能性のあるよく知られたコンピューティングシステム、環境、および／または構成の例は、パーソナルコンピュータ、サーバコンピュータ、ハンドヘルドデバイスまたはラップトップデバイス、タブレットデバイス、マルチプロセッサシステム、マイクロプロセッサベースのシステム、セットトップボックス、プログラム可能な家庭用電化製品、ネットワークＰＣ、ミニコンピュータ、メインフレームコンピュータ、上記のシステムまたはデバイスのいずれかを含む分散コンピューティング環境などを含むが、これらには限定されない。

30

【００２０】

本発明については、コンピュータによって実行される、プログラムモジュールなどのコンピュータ実行可能命令という一般的なコンテキストにおいて説明することができる。一般にプログラムモジュールは、特定のタスクを実行したり特定の抽象データ型を実装したりするルーチン、プログラム、オブジェクト、コンポーネント、データ構造などを含む。本発明は、通信ネットワークを介してリンクされるリモート処理デバイスによってタスクが実行される分散コンピューティング環境において実施することもできる。分散コンピューティング環境では、プログラムモジュールは、メモリストレージデバイスを含むローカルコンピュータストレージメディアおよびリモートコンピュータストレージメディアの双方に配置することができる。

40

【００２１】

図１を参照すると、本発明を実装するための典型的なシステムは、汎用コンピューティングデバイスをコンピュータ１１０の形態で含む。コンピュータ１１０のコンポーネント

50

は、処理装置120と、システムメモリ130と、システムメモリを含む様々なシステムコンポーネントを処理装置120に結合するシステムバス121とを含むことができるが、これらには限定されない。システムバス121は、メモリバスまたはメモリコントローラと、ペリフェラルバスと、様々なバスアーキテクチャのいずれかを使用するローカルバスとを含む複数のタイプのバス構造のいずれにすることもできる。例えばこのようなアーキテクチャは、ISA (Industry Standard Architecture) バス、MCA (Micro Channel Architecture) バス、EISA (Enhanced ISA) バス、VESA (Video Electronics Standards Association) ローカルバス、AGP (Accelerated Graphics Port) バス、およびメザニンバスとしても知られているPCI (Peripheral Component Interconnect) バスを含むが、これらには限定されない。

10

【0022】

コンピュータ110は通常、様々なコンピュータ可読メディアを含む。コンピュータ可読メディアは、コンピュータ110によってアクセスできる利用可能な任意のメディアとすることができ、揮発性メディアおよび不揮発性メディア、ならびに取り外し可能メディアおよび固定式メディアの双方を含む。例えばコンピュータ可読メディアは、コンピュータストレージメディアおよび通信メディアを含むことができるが、これらには限定されない。コンピュータストレージメディアは、コンピュータ可読命令、データ構造、プログラムモジュール、他のデータなどの情報を記憶するための任意の方法または技術において実装される揮発性メディアおよび不揮発性メディア、ならびに取り外し可能メディアおよび固定式メディアを含む。コンピュータストレージメディアは、RAM、ROM、EEPROM、フラッシュメモリまたは他のメモリ技術、CD-ROM、デジタル多用途ディスク (DVD) または他の光ディスクストレージ、磁気カセット、磁気テープ、磁気ディスクストレージまたは他の磁気ストレージデバイス、あるいは希望の情報を保存するために使用可能で、コンピュータ110によってアクセス可能な他の任意のメディアを含むが、これらには限定されない。通信メディアは通常、搬送波や他の伝送メカニズムなどの変調されたデータ信号内のコンピュータ可読命令、データ構造、プログラムモジュール、または他のデータを具体化し、任意の情報伝達メディアを含む。「変調されたデータ信号」という用語は、その特性のうちの1つまたは複数が、情報を信号に符号化するような方法で設定または変更された信号を意味する。例えば通信メディアは、有線ネットワークや直接有線接続などの有線メディアと、音波メディア、RFメディア、赤外線メディア、他の無線メディアなどの無線メディアとを含むが、これらには限定されない。また上記のいずれの組合せも、コンピュータ可読メディアの範囲内に含まれるものである。

20

30

【0023】

システムメモリ130はコンピュータストレージメディアを読み取り専用メモリ (ROM) 131およびランダムアクセスメモリ (RAM) 132などの揮発性メモリおよび／または不揮発性メモリの形態で含む。基本入出力システム133 (BIOS) は、起動中などにコンピュータ110内の要素間における情報伝達を補助する基本ルーチンを含み、通常はROM131内に格納されている。RAM132は通常、処理装置120にすぐにアクセスできるか、および／または処理装置120によってその時点で操作されているデータモジュールおよび／またはプログラムモジュールを含む。図1は、例としてオペレーティングシステム134、アプリケーションプログラム135、他のプログラムモジュール136、およびプログラムデータ137を示しているが、これらには限定されない。

40

【0024】

またコンピュータ110は、他の取り外し可能／固定式、揮発性／不揮発性コンピュータストレージメディアを含むこともできる。図1は、例示のみを目的として、固定式の不揮発性の磁気メディアとの間で読み取りや書き込みを行うハードディスクドライブ141と、着脱式不揮発性の磁気ディスク152との間で読み取りや書き込みを行う磁気ディスクドライブ151と、CD-ROMや他の光メディアなどの着脱式不揮発性の光ディスク

50

156との間で読み取りや書き込みを行う光ディスクドライブ155とを示している。典型的な動作環境において使用できる他の取り外し可能／固定式、揮発性／不揮発性コンピュータストレージメディアとしては、磁気テープカセット、フラッシュメモリカード、デジタル多用途ディスク、デジタルビデオテープ、ソリッドステートRAM、ソリッドステートROMなどがあるが、これらには限定されない。ハードディスクドライブ141は通常、インターフェース140などの固定式のメモリインターフェースを介してシステムバス121に接続されており、磁気ディスクドライブ151および光ディスクドライブ155は通常、インターフェース150などの着脱式メモリインターフェースによってシステムバス121に接続されている。

【0025】

図1に示した上述のドライブおよびそれらに関連するコンピュータストレージメディアは、コンピュータ110用のコンピュータ可読命令、データ構造、プログラムモジュール、および他のデータの記憶を提供する。例えば図1において、ハードディスクドライブ141は、オペレーティングシステム144、アプリケーションプログラム145、他のプログラムモジュール146、およびプログラムデータ147を記憶するものとして図示されている。これらのコンポーネントは、オペレーティングシステム134、アプリケーションプログラム136、他のプログラムモジュール137、およびプログラムデータ138と同一とするか、または異なってもよいという点に留意されたい。ここでは、オペレーティングシステム144、アプリケーションプログラム145、他のプログラムモジュール146、およびプログラムデータ147が最低限異なるコピーであることを示すために、異なる番号を割り当てている。ユーザは、タブレット（電子デジタイザ）164、マイクロフォン163、キーボード162、および通常はマウス、トラックボール、またはタッチパッドと呼ばれるポインティングデバイス161などの入力デバイスを介してコンピュータ110にコマンドおよび情報を入力することができる。他の入力デバイス（図示せず）は、ジョイスティック、ゲームパッド、衛星パラボラアンテナ、スキャナなどを含むことができる。これらの入力デバイスおよび他の入力デバイスは、システムバスに結合されているユーザ入力インターフェース160を介して処理装置120に接続される場合が多いが、パラレルポート、ゲームポート、ユニバーサルシリアルバス（USB）などの他のインターフェースおよびバス構造によって接続することもできる。またモニタ191や他のタイプのディスプレイデバイスも、ビデオインターフェース190などのインターフェースを介してシステムバス121に接続される。モニタ191は、タッチスクリーンインターフェース192などのインターフェースを介してコンピュータシステム110に手書きなどのデジタル化された入力を入力できるタッチスクリーンパネル193などと一体化することもできる。モニタおよび／またはタッチスクリーンパネルは、タブレットタイプのパーソナルコンピュータにおけるように、コンピューティングデバイス110が内蔵されている筐体に物理的に結合することができ、タッチスクリーンパネル193は基本的にタブレット164として機能するという点に留意されたい。さらに、コンピューティングデバイス110などのコンピュータは、スピーカ195およびプリンタ196などの他の周辺出力デバイスを含むこともでき、これは周辺出力インターフェース194などを介して接続することができる。

【0026】

コンピュータ110は、リモートコンピュータ180などの1つまたは複数のリモートコンピュータへの論理接続を使用して、ネットワーク化された環境内で動作することができる。リモートコンピュータ180は、パーソナルコンピュータ、サーバ、ルータ、ネットワークPC、ピアデバイス、または他の一般的なネットワークノードとすることができ、図1にはメモリストレージデバイス181しか示されていないが、通常はコンピュータ110に関連する上述の要素の多くまたはすべてを含む。図1に示されている論理接続は、ローカルエリアネットワーク（LAN）171およびワイドエリアネットワーク（WAN）173を含むが、他のネットワークを含むこともできる。こうしたネットワーキング環境は、オフィス、企業規模のコンピュータネットワーク、イントラネット、およびイン

10

20

30

40

50

ターネットにおいてよく見受けられる。

【0027】

L A Nネットワーク環境において使用する場合、コンピュータ110は、ネットワークインターフェースまたはアダプタ170を介してL A N171に接続される。W A Nネットワーク環境において使用する場合、コンピュータ110は通常、モデム172、またはインターネットなどのW A N173上で通信を確立するための他の手段を含む。モデム172は内蔵型または外付け型とすることができ、ユーザ入力インターフェース160または他の適切なメカニズムを介してシステムバス121に接続することができる。ネットワーク化された環境では、コンピュータ110に関連して示されているプログラムモジュール、またはその一部をリモートメモリストレージデバイス内に格納することができる。図1は、例としてリモートアプリケーションプログラム185をメモリデバイス181上に常駐するものとして示しているが、この形態には限定されない。示されているネットワーク接続は代表的なものであり、コンピュータ間に通信リンクを確立する他の手段も使用できることが理解できるであろう。

10

【0028】

(階層化アーキテクチャの例)

本発明の一態様は、一般にスムーズで複雑なアニメーションおよび／またはメディアをコンピュータシステム上に提供することを対象としている。この目的のために、図2に一般的に示されているように、メディア統合レイヤアーキテクチャ200を提供する。アプリケーションプログラム、制御、または他の類似のより高いレベルのプログラムコード（例えば、オペレーティングシステムコンポーネントのユーザインターフェース）202は、アプリケーションプログラミングインターフェース（A P I）204のセットなどを介してメディア統合レイヤアーキテクチャ200にアクセスして、グラフィカルな情報へのアクセス（書き込みまたは読み取り）を行う。本明細書に記載する例の多くは、A P Iとインターフェースで接続されているアプリケーションプログラムに言及するが、他のより高いレベルのプログラムコードおよびコンポーネント（例えば、オペレーティングシステムのユーザインターフェース）は、本明細書に記載されているより低いレベルのコンポーネントとインターフェースで接続することもできることを理解できるという点に留意されたい。そのようなものとして、アプリケーションプログラムやユーザインターフェースなど、その呼称を問わず、このようなより高いレベルのプログラムコードに対するいかなる言及も同義とみなすべきである。

20

30

【0029】

一実装形態では、メディア統合レイヤアーキテクチャ200は、高いレベルの合成およびアニメーションエンジン206、タイミングおよびアニメーションコンポーネント208、ならびに低いレベルの合成およびアニメーションエンジン210を含む。本明細書で使用する「高いレベル」および「低いレベル」という用語は、他のコンピューティングシナリオで使用される場合と同義であり、一般にソフトウェアコンポーネントが、より高位のコンポーネントと比較して低位にあればあるほど、そのコンポーネントはハードウェアに近くなる。したがって、例えば高いレベルの合成およびアニメーションエンジン206から送信されるグラフィックス情報は、低いレベルの合成およびアニメーションエンジン210で受信することができ、この情報を使用して、ハードウェアを含むグラフィックスサブシステムにグラフィックスデータを送信する。

40

【0030】

一般に、高いレベルの合成およびアニメーションエンジン（本明細書では高いレベルのコンポジタ(compositor)およびアニメータ(animater)、あるいは高いレベルのエンジンまたはコンポーネントとも呼ばれる）206は、アプリケーションプログラム202によって提供されるグラフィックスシーンを表すディスプレイエレメントツリーを構築し、その一方でタイミングおよびアニメーションコンポーネントは、宣言型の（または他の）アニメーションおよびタイミングの制御を提供する。低いレベルの合成およびアニメーションエンジン（本明細書では低いレベルのコンポジタおよびアニメータあるいは低いレベルの

50

エンジンまたはコンポーネントとも呼ばれる) 210は、複数のアプリケーションのシーン用のレンダリングを合成し、レンダリングコンポーネントによって、画面に対するグラフィックスの実際のレンダリングを実施する。時間のかかるまたはアプリケーションに特有なレンダリングをより高いレベルで行うこと、およびビットマップなどへの参照をより低位のレイヤに渡すことも依然として可能であるという点に留意されたい。

【0031】

高いレベルの合成およびアニメーションエンジン206は、エレメントツリー構造を構築して、その構造を行き来し、レンダリング命令および単純なアニメーションインターバルを作成して、低いレベルの合成およびアニメーションエンジン210に渡す。高いレベルのコンポジタによって作成されるレンダリング命令は、タイミングおよびアニメーションの情報を含むことができる。低いレベルの合成およびアニメーションエンジン210は、レンダリング命令およびアニメーションインターバルを受け取り、次いでグラフィックスサブシステム(例えば、グラフィックスソフトウェアおよびハードウェア)212に提供されるシーンの動画化、レンダリング、および合成を管理する。ローカルに表示された出力の代替として、または追加として、高いレベルの合成およびアニメーションエンジン206(またはこれに類似したもの)は、固定されたイメージデータをプリンタ222などへ送信するためのより低いレベルの印刷コード220にレンダリングおよびアニメーション命令を適切なフォーマットで提供することができ、および/またはリモートマシン228への送信用のより低いレベルのターミナルトランスポートサーバ226にレンダリング命令および単純なアニメーションインターバルを適切なフォーマットで提供することができる。より豊かな情報をネットワーク全体に渡すこともできるという点に留意されたい。例えば何のネットワークトラフィックも伴わずにリモートマシンにマウスロールオーバー効果(mouse rollover effect)をローカルに処理させることが望ましい場合もある。

【0032】

この実装形態では、メディア統合レイヤアーキテクチャ200は、このようにグラフィックス処理を複数のレベルに分割し、これらのレベルのそれぞれが、何らかのインテリジェントなグラフィックス処理を実行し、これらが合わさることは、アプリケーションのユーザインターフェースなど202が、スムーズなアニメーションでグラフィックスを出力し、そのグラフィックスを他のアプリケーションのグラフィックスと合成し、ビデオフレームと連動することを可能にする。アニメーションおよび/または合成は、オーディオ出力と同期することもできる。例えば低いレベルのコンポーネントでオーディオとフレーム率を同期させることによって、オーディオのタイミングは、ビデオまたはグラフィックスのタイミングと基本的に一致することができ、リフレッシュ速度についていくためのタスクとしてスケジュールされた複雑な事前処理の能力に依存することはない。

【0033】

図3は、XMLベースのコードなどのマークアップコード302をパーサ/トランスレータ304によって解読できる一実装形態を表している。一般にパーサ/トランスレータ304は、エレメントツリー/プロパティシステム314にエレメントを追加し、これらのエレメントは、自分自身のレイアウトを行うビジュアルオブジェクトである。さらにマークアップコードのいくつかまたはすべては、オンデマンドで解釈するのではなく、コンパイルすることができ、これによって効率が高まるという点に留意されたい。

【0034】

一般にエレメントは、プロパティシステム、トリガリングおよびレイアウト/プレゼンテーションシステムに参画するエレメントレイヤ内のオブジェクトである。パーサ304はタグを見つけて、それらのタグがエレメントまたはリソースオブジェクトを定義するのに役立つかどうかを判断する。VisualBrushの特別なケースでは、(特許文献3)に記載されているように、例えば同じタグを、それらのタグが現れるコンテキストに応じて、例えば複合プロパティシntax内に見れるかどうかに応じて、エレメントとして解釈したり、あるいはリソースオブジェクトとして解釈したりすることもできる。

【0035】

マークアップ内でインラインで存在することに加えて、リソースインスタンスは、別の場所に（例えばローカルに配置できるか、またはリモートネットワーク上に配置して適切にダウンロードできるマークアップ内またはファイル内に）配置し、名前によって（例えばテキスト名、参照、または他の適切な識別子によって）参照することができる。このようにして、シーンの設計者は、複合プロパティシンタックスによって記述されたエレメントを含む、エレメントツリー内の1つのエレメントを1つのシーンを通じて再利用することができる。

【0036】

パーサ304は、必要に応じてタイプコンバータ308にアクセスすることによって、また指定されたパラメータをオブジェクトプロパティにマッチさせることによって、複合プロパティシンタックス内のマークアップを処理し、これによってシーンの設計者にとっての複雑さに対処する。したがってパーサ304は、オブジェクトをセットアップするだけでなく、オブジェクト上のプロパティも設定する。同じレンダリングモデルがエレメントレベルとAPIレベルの間で共有されるため、オブジェクトの多くは基本的に同じである。これは、解析／翻訳をきわめて効率的にし、また様々なタイプのプログラミング言語（例えば、C#のような言語）に、マークアップからそれ自身のシンタックスへ、およびその逆へ容易に変換できる機能を与える。図3に示されているように、（コンパイルされたマークアップを含むことができる）このような別のプログラミング言語310は、エレメントをエレメントツリー208に追加するか、またはビジュアルAPIレイヤ316と直接インターフェースで接続できるという点に留意されたい。

【0037】

図3にも示されているように、同じマークアップ302を使用して、エレメントレベルおよびリソースレベルでプログラムを作成することができる。一般にエレメントレベルは、十分なプログラム可能性と、継承（例えばスタイルシートのような機能）を提供するプロパティシステムの使用と、トリガ（例えば、これによってエレメントがユーザの入力したイベントまたはアクションに応答してその外観や位置などを変更するコードを添付した可能性がある）とをシーンの設計者に与える。しかし本発明は、リソースレベルのメカニズムも提供し、これによってシーンの設計者は、基本的にエレメントツリーおよびプログラムを直接ビジュアルAPIレイヤヘショートカットすることができる。エレメントレベルの機能を必要としない多くのタイプの静的な形状やイメージなどにとって、これは、適切なオブジェクトを出力するためのより効率的で軽快な方法を提供する。

【0038】

アニメーションおよびメディアの出力を制御する目的で、図4および5を参照して後述するように、クロックを含むタイミングツリーも保持される。一般に高いレベルのコンポジットおよびアニメータエンジン206は、（コンパイルと呼ばれる場合もある）複雑な処理を実行し、処理量を著しく簡素化し、正しい出力をレンダリングするためにより低いレベルで処理する必要のあるデータ量を大幅に削減する。しかし、より高いレベルによって実行される処理の量およびタイプは、より低いレベルの負荷、構成、および性能にかなりの程度依存する場合があるという点に留意されたい。例えば高性能のグラフィックスハードウェアがある場合、より高いレベルで行う処理の量はより少なくすることができ、逆もまた同様である。高いレベルのレイヤおよび低いレベルのレイヤは、これらの要因に適応することができる。

【0039】

一般にアニメーションは、高いレベルのコンポジットおよびアニメーションエンジン206と低いレベルのコンポジットおよびアニメーションエンジン210の双方によって達成される。一実装形態では、高いレベルのエンジン206は、シーンを行き来し、その後の補間用のインターバルによってアニメーションパラメータを更新し、これらの簡素化されたデータ構造を命令にパッケージして、より低いレベルのエンジン210へ渡す。これは、同期的なおよび／または非同期的な方法で行うことができる。インターバルデータは、タイミングエンドポイント（開始および終了のタイミングのデータ）、ならびにレンダリン

グ命令用のパラメータ化された値を含むものとみなすことができる。高いレベルのエンジン204は、要求された補間のいくつかまたはすべてを実行できるという点に留意されたい。例えば補間または他のモーション機能が複雑すぎて、より低いレベルのエンジン210が処理できない場合、またはより低いレベルが、その上に課された処理要求についていけない場合、より高いレベルのエンジンは、その計算のいくつかまたはすべてを実行することができ、簡素化されたデータ、命令、テッセレーション(tessellation)などをより低いレベルに提供して、所望の結果を達成する。

【0040】

典型的なケースでは、より低いレベルが、アニメーションのそれぞれのフレームごとに実際に補間を実行する際、低いレベルのエンジン210は、パラメータインターバルを補間して瞬時値を得て、命令をレンダリングコマンドへと復号し、このレンダリングコマンドは、グラフィックスデバイスによって実行される。グラフィックスデバイスは、最終シーン内に現れる可能性のあるすべてのビデオフレームを追加して、最終シーンを合成する。デジタル権利管理によって保護されているコンテンツなど、他のデータを追加することもできる。

【0041】

したがって高いレベルのエンジン206は、シーンデータ構造を行き来し、一定の時間にわたるそれぞれの動画化されたパラメータを記述するインターバルを計算し、これらのインターバルおよび簡素化されパラメータ化された描画命令を低いレベルのエンジン210に渡す。パラメータデータは、開始時刻、終了時刻、インターポレータ(interpolator)、および補間データを含む。例えば、イメージが動いて見えるようにそのイメージを消去して描き直すのではなく、高いレベルのコンポジタおよびアニメーションエンジン206は、そのイメージが時間の経過と共にどのように変化すべきかについて、例えば開始座標、終了座標、イメージが座標間で移動すべきである時間の量(インターバル)、および線形などのモーション機能について、低いレベルのコンポジタおよびアニメーションエンジン210に指示することができる(例えば静止オブジェクトは、その色プロパティを変更することによって動画化できるため、モーションはアニメーションにとって必要とされないという点に留意されたい)。低いレベルのコンポジタおよびアニメーションエンジン210は、補間を行って、フレーム間の新たな位置を確定し、これらをグラフィックスデバイスの理解できる描画命令に変換し、そのコマンドをグラフィックスデバイスに渡すことになる。高いレベルのエンジン206が渡すたびに、低いレベルのエンジン210が複数のフレームにわたってスムーズなアニメーションを実行するのに十分なデータが提供されることが好ましい。

【0042】

低いレベルの(例えばファーストチック(first-tick))エンジン210は、高いレベルのエンジン206から分離されたタスクである。低いレベルのエンジン210は、シーンについて記述する簡素化されパラメータ化された描画命令およびパラメータインターバルを高いレベルのエンジン206から受け取る。低いレベルのエンジンは、高いレベルのエンジン206によって新たなデータ構造が提供されるまで、これらのデータ構造を保持して行き来する。低いレベルのエンジンは、複数の高いレベルのエンジン206にサービスを提供することができ、それぞれごとに別々のデータ構造を保持する。低いレベルのエンジン210と高いレベルのエンジン206の間におけるこの一对多の関係は、システムが同時に複数のシーンをスムーズに動画化することを可能にする。

【0043】

低いレベルのエンジン210は、高いレベルのエンジンの提供されたインターバルに基づいて基本的に瞬時のアニメーションパラメータを補間し、描画命令を更新し、あらゆるフレーム用のシーンをレンダリングする。低いレベルのエンジン210のタスクは、システム上において高い優先順位で実行され、フレームをグラフィックスハードウェアの画面のリフレッシュ速度などで提示できる状態にしておく。したがって低いレベルのエンジン210によって実行される補間は通常、線形、区分線形、三次スプライン、および類似の

速度の関数などの高速な単関数に限定される。

【0044】

アニメーションおよびメディアに関しては、図4および5に全般的に示されているように、アプリケーションプログラム202などのプログラムは、高いレベルのコンポーネント206に対して、クロックまたはクロックプロパティと呼ばれる、タイミング情報を伴うアニメーションプロパティ値を指定する。後述のように、基本的にすべての独立したアニメーションまたはメディア（例えば、ビデオおよびオーディオなどの線形メディア）、ならびに指定されたアニメーションを調整するストーリーボードは、そのために保持されるクロックを高いレベルのコンポーネントにおいて有することになる。一般に作成者は、クロックへとインスタンス化されるタイムラインデータを必要に応じて指定して、それらの同期を維持する。

10

【0045】

クロックプロパティは、所与のクロックとタイミング構造の残りとの間における最初の同期関係を定義するタイミングプロパティを含む。図5に示されているように、高いレベルのコンポーネント206は、アニメーションのプロパティのその時点での値を確定するために、高いレベルのアニメーション関数520_H（例えばマネージドのコードで書かれており、これは、情報をメタデータの形態で提供するコードであり、共通言語ランタイムを可能にし、この上で動作して、そのオペレーションを制御する）を呼び出すことができる。高速のフレーム率の計算中に、低いレベルのコンポーネント210は、アニメーションのその時点でのプロパティ値を確定するために、エンジン414によって計算された増加を有する類似の（または同じ）アニメーション関数520_Lを呼び出す。代替実装形態では、例えばアニメーション関数をより低いレベルのコンポーネントに組み込むことができるという点に留意されたい。

20

【0046】

一般にアニメーションおよび線形メディアは、クロックのセットに関連付けられ、これらのクロックは、同期化プリミティブ(synchronization primitive)およびルールによって相互に関連付けられている。クロックは階層的に構成することができ、例えばアプリケーションプログラムは親クロックを有し、アプリケーションプログラムの動画化されたオブジェクトは子であり、これも同様に他の子を有することができる。1つのクロックのプロパティが定義または修正されると、そのクロックのすべての子が影響を受ける。例えば親クロックを一時停止すると、その子クロックもすべて一時停止し、親クロックの速度を倍増させると、そのすべての子クロックの速度も倍増する。

30

【0047】

これらのクロックプロパティは、ランタイムにアプリケーションによって開始された対話式の制御イベントを含むソースイベントによって修正することができる。したがってクロックは対話式であり、それぞれのクロックは、アプリケーションによって任意の時点で、例えばユーザの入力に応答して、個々に開始、一時停止、再開、および停止することができる。その上、新たなクロックをタイミング構造に追加することができ、また既存のクロックを除去することもできる。

【0048】

前述の（特許文献2）に記載されているように、高いレベルのタイミングコンポーネントは、保存されているイベント（開始や一時停止など）のリストおよび関連付けられている同期化プリミティブに基づいて、それぞれのクロックごとにインターバルリストを作成することができる。アクティブ化インターバルは、現実の時間における様々な時点で所与のクロックによって表される時間について記述する単純で重ならないセグメントである。

40

【0049】

図4に示されているように、クロックプロパティの少なくとも一部は、クロックのタイミングツリー402内で階層的に関連付けることができる。このような3つのクロックとそのプロパティ、すなわちクロックプロパティ404₁~404₃が図4に示されているが、与えられた状況によっては、より多くのクロックおよび代わりのタイミングツリーが存

50

在してもよいことを理解することができる。例えば表示される動画化されたオブジェクト（またはストーリーボードによって調整されたオブジェクトのセット）に対応できるそれぞれのクロックごとに、高いレベルのコンポーネント206は、イベントリストジェネレータ408と呼ばれる状態マシン(state machine)を含み、これは、クロックプロパティからイベントリスト（例えば406₁）を作成する。一般にイベントリストジェネレータ408は、指定されたクロックプロパティによって最初にスケジュールされたイベントを、アニメーションに関して受信される一時停止要求および再開要求など、何らかの明示的な対話型のイベントと共に1つのイベントリストにまとめる。クロックは、それぞれのアニメーション（または後述のようなストーリーボードを介したアニメーションのグループ）ごとに保持され、したがって、それぞれの独立したアニメーション／アニメーションセ

10

【0050】

高いレベルのコンポーネントは、インターバルジェネレータ410を含み、これは、それぞれのアニメーションまたはメディア用のイベントリスト（例えば406₃）を使用して、対応するインターバルリスト（例えば412₃）を計算し、これを低いレベルのコンポーネント210に渡す。次いで、低いレベルのコンポーネント210は、低いレベルの計算エンジン414を含み、これは、そのオブジェクト用のインターバルデータおよび現在の時刻に基づく増加値を、動画化されたオブジェクトの変わりゆくプロパティに関する現在の値を確定する低いレベルのアニメーション関数サブシステム520_Lに提供することなどによって、インターバルデータに関して現在の時刻に基づいて出力を制御する。例えば任意の所与のフレームに関して、低いレベルの計算エンジン414は、動画化されたオブジェクトの位置をインターバルデータおよび現在の時刻（これは、システム時間または相対的時間とすることができる）に基づいて補間する。図4および5に示されているように、高いレベルのタイミングコンポーネントは、タイミングツリー402およびイベントリスト406₁～406₃を含み、その一方で低いレベルのタイミングコンポーネント210は、インターバルリスト412₁～412₃を含むが、これらのデータ構造は、基本的にストレージ内のいずれの場所に保持することもでき、通常は高速のランダムアクセスメモリ内に保持できるという点に留意されたい。

20

【0051】

要約すると、高いレベルのタイミングコンポーネント206は、同期化ルールおよびプリミティブによって（例えば階層的に）関連付けられているクロックのツリー402を参照する。本発明は、高いレベルのタイミングコンポーネント206内のクロックを使用して、低いレベルのタイミングエンジンによって消費されるアクティブ化インターバルのリスト412をコンパイルする。低いレベルのタイミングエンジンは、インターバルリスト412を参照し、これは、（例えば動画化されたオブジェクトごとに、または線形メディアごとに）独立したクロックを表す。図5に示されているように、システムクロック524など、双方にとって整合性のある時間を提供するクロックがあり、これによって複数のレベルが同期を維持する。

30

【0052】

アプリケーション202（または何らかの他のソース）から受け取った対話型の変更に応答して、高いレベルのタイミングコンポーネントは、その変更に直接関与しているクロックの状態、さらには間接的に関与しているすべてのクロックの状態を、同期化プリミティブによって指定されるように更新する必要がある。単一のクロックに対する変更が、タイミング構造内の他のクロックにも影響を与える場合がありうるという点に留意されたい。例えばスライドショーの発表者は、ディスプレイ全体を一時停止する場合があり、これによって、その時点で表示されているすべてのアニメーションおよび線形メディアは、一時停止する必要がある。後述するように、本発明の一態様によれば、ストーリーボードは、アニメーションおよび／またはメディアをグループにまとめて、そのような望ましい方法で調整することを可能にする。

40

50

【0053】

一般にオブジェクトに関して対話が発生するときは常に、イベントリストが必要に応じて再作成され、インターバルリストが再計算されて、より低いレベルのタイミングコンポーネントへ送信される。より高いレベルのコンポーネントは、これらのオペレーションを（例えば毎秒6回から10回程度の頻度で）実行し、これによって、（例えば毎秒60回動作する）より低いレベルのコンポーネントは、それぞれの動画化されたオブジェクトごとにその時点のインターバルリストを処理するだけでよい。

【0054】

このインターバル情報ならびにオブジェクト用の開始値および終了値に加えて、さらなるパラメータをより高いレベルのコンポーネントへ送信することができ、その結果としてインターバルリストが変更される。例えばアプリケーションは、イベントに対応する、時間における期間への検索を行うことができる。アプリケーションは、その検索がいつ、どの程度の分量で発生するかを指定する。巻き戻し中または早送り中のメディアのように、アニメーション上の検索によって、アプリケーションは、時間における特定のポイントまで前または後ろにジャンプすることができる。リバースイベント(reverse event)は、アプリケーションが指定できる別のタイプのスケジュールされたまたは対話型のイベントである。リバースイベントがリスト内にある場合、増加は、例えば100パーセントから0パーセントへ自動的に逆転されることになる。速度の変更もサポートされ、例えば特定のクロックを下回るものはすべて、実際の時間よりも速くまたは遅く実行されるように設定することができる。特定のクロックに関して、再開が認められるかどうか、および／またはその時点で実行されていない場合、再び開始するかどうか、例えば1秒で開始し、10秒で再び開始し、100秒で再び開始するなど、ただしその時点で実行されている場合は、任意選択で再開するかしないかを示すことなど、他の属性を指定することもできる。

【0055】

（アニメーションおよびメディアを調整するためのストーリーボード）

本発明は、とりわけストーリーボードを対象としており、これは、タイムラインツリーをエレメント（例えば、ウィンドウ等のFramework Element、パネル、ボタン等の制御など）用に指定できるようにするオブジェクトを含む。タイムラインは、エレメントのプロパティ値を設定し、エレメントおよびそのサブツリー用のアニメーションを制御する。ストーリーボードは基本的に、開始すること、停止または一時停止すること、あるいは検索するために使用することが可能な特定のタイプのタイムラインオブジェクトに対応し、これによって、動画化されたエレメントの外観、および／またはそのストーリーボードに関連付けられているメディアの動作を制御する。

【0056】

理解できることであろうが、本発明の一態様では、ストーリーボードは、トップレベルのタイムラインのコレクションを含み、これは、1つのエレメントに関連付けることができ、かつ独立して開始することができる。例えば後述するように、このモデルは作成者が、mouseOverアニメーションおよびmouseClickアニメーションを、同一のストーリーボードコレクション内にある1つのエレメント上に有することを可能にする。このmouseOverアニメーションタイムラインとmouseClickアニメーションタイムラインは、独立してトリガできるが、同じユニット内に含まれている。ストーリーボードは、タイミングツリーと、ストーリーボード内のアニメーションが、そのストーリーボードが添付されているエレメントに関連してそれらのターゲットを分解できるようにするエレメントツリーとの接続ポイントも提供する。ストーリーボードはまた、状態の変化、イベント、または他のアクションによってアクティブ化できるトリガのコレクションを含むこともでき、そのトリガがアクティブ化されたときに実行される拡張可能なアクションのセット（例えば、タイムラインの開始または停止）を含むこともできる。ストーリーボードは、作成者がストーリーボード内のタイムラインをコードから対話式に開始および停止できるようにする、基礎をなすオブジェクトモデルの一部である。

【0057】

一般にストーリーボード内のタイムラインは、様々なサブツリー、エレメント、および他のオブジェクト（例えば、テキストの文字、単語、行、パラグラフなど）に適用できるアニメーションテンプレートを表している。ストーリーボードは、FrameworkElement上に直接設定するか、またはStyle（後述のように、ボタンの色、サイズ、および他のプロパティなど、エレメントに適用できる1つまたは複数のデータ値を含むリソース）を介してFrameworkElementに適用することができる。ストーリーボードを使用して、チェンジャブル（例えばブラシ(brush)、変形、ジオメトリなど。（特許文献4）に記載されているように、チェンジャブルを使用して、FrameworkElementsのプロパティを設定することができる）などのクローニングをサポートするオブジェクトにアニメーションを適用することもできる。例えばブラシは、オリジナルの情報を復元できるように、オリジナルのバックグラウンド情報を複製して、複製された情報上で動作することによって、ボタンまたはパネルのバックグラウンドを設定するチェンジャブルである。したがってストーリーボードを使用して、FrameworkElementのプロパティを設定するチェンジャブルを動画化することができる。言い換えれば、ストーリーボードに関連する2種類のオブジェクト、すなわちストーリーボードと共に参画するエレメント（FrameworkElementなど）、およびストーリーボードと共に参画しないエレメント（例えば、前述のようなチェンジャブル）がある。ストーリーボードのアルゴリズム／データ構造は、参加するものは指定されたように作業するが、クローニングをサポートする限り、参加しないものすら指定されたように作業するようなものである。

【0058】

本発明の一態様によれば、ストーリーボード（例えば、APIを介してアクセスされるオブジェクト）は、プログラム作成者が、タイムライン内でアニメーションを（エレメントプロパティへ）グループ化することによって、調整されたアニメーションのセットをスケジュールすることを可能にする。すなわちストーリーボードは、タイミングツリーをエレメントツリーと統合し、同じタイムラインを介して調整された方法で様々なアニメーションの様々なプロパティ用のタイミングを操作する機能を提供する。したがってストーリーボードは、様々なオブジェクトおよびプロパティに影響を与える、その他の点では独立したタイムラインを単一のタイムラインツリーに統合して、プログラム作成者がタイミングイベントの複雑なセットを簡単な方法で整理および制御できるようにする方法を提供する。例えば個々のアニメーションを見つけて再開しなければならない代わりに、アニメーションをストーリーボードの一部としてタイムラインの下でグループ化することは、ストーリーボードを再開するだけですべてのアニメーションを同時に再開することを可能にする。さらに、タイミングツリーとエレメントツリーを分離することは、ユーザが、アニメーションのセットを順番に実行することや、複雑なタイミング構造をループすることなど、時間における操作を行うためにタイミングツリーを使用することを可能にする。

【0059】

ストーリーボードは、アニメーション／ビデオ／オーディオの特定のインスタンス上で、または特定のスタイルのアニメーションに対して使用することができる。

【0060】

対比の目的で、ストーリーボードなしで書かれた以下のコード例について考察すると、これは、対話式の応答を追加するためのコードビハインド(code-behind)を必要としている（この例では、マウスをクリックした後に2つの別々のボタンを希望どおりに同時に動画化すること）。このコードの作成者は、それぞれのボタンのアニメーション用に必ず同じ開始時刻を設定する必要がある。これは、これらの時刻をそれぞれのボタンのタイムライン上で別々に宣言する必要があるためである。このコードは、2つのボタンを同時に動画化することを目的としているが、第1のボタンのアニメーションおよび第2のボタンのアニメーションは、2つの別々の（Begin）コールによって開始されるため、実際には第1のボタンのアニメーションと第2のボタンのアニメーションの間に何らかのラグが生じるという点に留意されたい。

【0061】

【表1】

```

<Canvas Background="#ffffffff" >
  <Button Opacity="0" Click="StartAnimations">
    <Button.Opacity>
      <Timeline def:ID="_timeline1" BeginTime="5">
        <DoubleAnimation To="1" Duration="1" />
      </Timeline>
    </Button.Opacity>
    Hello
  </Button>
  <Button Canvas.Left="100" Canvas.Top="0" >
    <Canvas.Top>
      <Timeline def:ID="_timeline2" BeginTime="5">
        <LengthAnimation To="300" Duration="5" />
      </Timeline>
    </Canvas.Top>
    Hello 2
  </Button>
</Canvas>

public void StartAnimations(object sender, RoutedEventArgs args)
{
  _timeline1.InteractiveController.Begin();
  _timeline2.InteractiveController.Begin();
}

```

10

20

【0062】

本発明の様々な態様によれば、一実装形態では、アニメーション用にストーリーボードを使用するために、プログラム作成者は、Timeline、関連付けメカニズム(association mechanism) (例えば、SetterTimelineまたは添付されているStoryboardプロパティ)、およびAnimationオブジェクトをFrameworkElementのStoryboardsプロパティに追加する。ストーリーボードは、特定の要素に直接適用するか、または(後述するように)スタイル内で定義することができる。この実装形態では、ストーリーボードは、少なくとも1つのSetterTimelineを有し、これを使用して、1つまたは複数のタイムラインを指定されたオブジェクトの指定されたプロパティにリンクする。上述したタイムラインの基本的なタイミングプロパティに加えて、SetterTimelineは、3つの独特なプロパティを有し、これによって作成者は、動画化するオブジェクトおよびプロパティ、ならびに動画化する元となる基本値、すなわちオブジェクトの動画化をターゲットとするTargetID、動画化するプロパティをターゲットとするPropertyPath(本明細書では「Property」または「Path」と同義である)、およびターゲットとするプロパティ用の基本値が存在する場合にその基本値を指定する任意選択のプロパティであるValueを指定することができる。作成者は、SetterTimelineによってオブジェクトおよびプロパティをターゲットとした後に、タイムラインおよびアニメーションをその子として追加して、ターゲットとするプロパティにそれらを適用するという点に留意されたい。

30

40

【0063】

以下のコード例は、前のコード例との比較から、ストーリーボードがどのようにコーディングを簡素化し、機能性を高めるかを示している(この例は、ストーリーボードクラス上の添付プロパティを介して動作する後述のような代替実装形態ではなく、SetterTimelineの実装形態例を使用しているという点に留意されたい)。

【0064】

【表 2】

```

<Canvas Background="#ffffff" >
  <Canvas.Storyboards>
    <Timeline def:ID="myStoryboard" BeginTime="5">
      <SetterTimeline TargetID="myButton">
        <DoubleAnimation To="1" Duration="1" />
      </SetterTimeline>
      <SetterTimeline TargetID="myButton2"
        PropertyPath="(Canvas.Top)" Value="0">
        <LengthAnimation To="300" Duration="5" />
      </SetterTimeline>
    </Timeline>
  </Canvas.Storyboards>
  <Canvas.Triggers>
    <EventTrigger TargetID="myButton" Event="Click">
      <TimelineBeginAction TargetID="myStoryboard" />
    </EventTrigger>
  </Canvas.Triggers>
  <Button ID="myButton" Opacity="0">Hello</Button>
  <Button ID="myButton2" Canvas.Left="100">Hello 2</Button>
</Canvas>

```

10

【0065】

別法として、このXAMLコード例の第2のセットは、ストーリーボードを使用して、1つのクリックイベントによって2つのボタン上で同時にアニメーションを開始する。この目的のために、キャンバス(canvas)の単一のストーリーボードタイムライン内には、1つのエレメントプロパティ(myButtonエレメントの不透明度プロパティ)をそのストーリーボードタイムラインに結び付ける1つのSetterTimelineと、別のエレメントプロパティ(myButton2エレメントの位置プロパティ)をその同じストーリーボードタイムラインに結び付ける別のSetterTimelineとがある。単一のイベント(myButtonをクリックすること)は、ストーリーボードをトリガし、調整されたアニメーションを開始し、時間の経過と共にmyButtonの不透明度およびmyButton2の位置を変える。

20

【0066】

対照的なコード例から分かるように、ストーリーボードを使用することは、コードの複雑さを緩和し、アニメーションの開始時刻間において純粋な調整を行う。この例では、双方のアニメーションは、純粋に同じ時刻に開始される。これは、Beginに対して別々のコールがないためである。さらに、双方向性を実現する上でXAMLを補助するコードは不要であり、したがって作成者は、XAMLだけでコードを書くことができる。その上、ストーリーボードでは、作成者は、調整された開始時刻を1つの場所に、すなわちストーリーボードのタイムライン上に設定するだけでよい。

30

【0067】

したがってストーリーボードは、作成者が、時間の経過に伴うそれぞれのアニメーションの変更を個々に定義する代わりに、アニメーションのセットをグループとして整理および制御することを可能にし、これによって、相互に同期するアニメーション、およびアニメーションのシーケンス(後述する)を簡単に作成することができる。以下の例について考察すると、これは、異なる幅および色を有する3本の長方形の針、すなわち黒色の時針、緑色の分針、および赤色の秒針を備えたクロックに相当し、角度を設定する回転の変形を使用することによって、その時点でのシステム時間を自動的に表示する。この例では、単一のストーリーボードタイムラインを使用して、3本の針のアニメーションを調整する。

40

【0068】

【表 3】

```

<Canvas xmlns="http://schemas.microsoft.com/xaml/2005"
Height="100%" Width="100%" Background="White"
ID="Canvas1">

<!-- We initialize the clock hands to stand at zero degrees at midnight.
The hands appear at the current global time when displayed because they
have theoretically) continued rotating ever since midnight. -->

<Canvas.Storyboards>
<Timeline Begin = "@12:00:00 AM"> //Set the zero-time position for the
clock hands.
<SetterTimeline TargetID="hourhand" Property="RenderTransform"
SubPropertyPath="RotateTranform.Angle">
<DoubleAnimation From="0" To="360" Duration="12:0:0" Repeat="Indefinite"/>
</SetterTimeline>
<SetterTimeline TargetID="minutehand" Property="RenderTransform"
SubPropertyPath="RotateTranform.Angle">
<DoubleAnimation From="0" To="360" Duration="1:0:0" Repeat="Indefinite"/>
</SetterTimeline>
<SetterTimeline TargetID="secondhand" Property="RenderTransform"
SubPropertyPath="RotateTranform.Angle">
<DoubleAnimation From="0" To="360" Duration="0:1:0" Repeat="Indefinite"/>
</SetterTimeline>
</Timeline>
</Canvas.Storyboards>

<Rectangle ID="secondhand" Fill="Red" Top="0" Left="50" Height="50"
Width="2">
<Rectangle.RenderTransform>
<RotateTransform Center="0 50"/>
</Rectangle.RenderTransform>
</Rectangle>

<Rectangle ID="minutehand" Fill="Green" Top="0" Left="50" Height="50"
Width="3">
<Rectangle.RenderTransform>
<RotateTransform Center="0 50"/>
</Rectangle.RenderTransform>
</Rectangle>

<Rectangle ID="hourhand" Fill="Black" Top="0" Left="50" Height="50"
Width="4">
<Rectangle.RenderTransform>
<RotateTransform Center="0 50"/>
</Rectangle.RenderTransform>
</Rectangle>

</Canvas>

```

10

20

30

【0069】

レンダリング変形プロパティ(render transform property)および回転変形角度サブプロパティ(rotate transform angle sub-property)を介して、上記のコードを実行すると、タイムマネージャーは、(これらの針が0度の角度にあることが分かっていたAD0年の00:00:00以降の)その時点の時刻と、レンダリングの際に使用された更新された角度とに基づいて、それぞれの針の回転量を自動的に計算する。それぞれの新たなアニメーションの計算は、例えばフレームごとに1回、または他の何らかの割合で、これらの針の角度位置を更新する。3本の針は、すべていっせいに始動し、例えば(上記のコード例には示されていない)何らかのイベントがクロックを停止した場合、3本ともいっせいに停止する。

40

【0070】

容易に理解できることだが、例えば複数のSetterTimelinesは、単一のストーリーボードタイムラインの下でグループ化することができ、それぞれのSette

50

`rTimeline`は、様々なターゲットエレメントおよびそれらのエレメントのプロパティを参照するため、エレメントプロパティのアニメーションの複雑度は著しくなる場合がある。ストーリーボードは、組み合わせることができる。例えば1つのストーリーボードは、1つの変形を1つのエレメントに関連付けることができ、別のストーリーボードは、その変形用の値を設定することができる（ただし、その値が設定される前にその変形が存続しているように、コード内の順序を正しくする必要がある）。いずれにしても、ストーリーボードを使用することは、作成者が、所望のエレメント、プロパティ、およびアニメーションの動作をレイアウトすることを容易にする。

【0071】

図6は、そうしたコードをどのように構成できるかを一般的に示したものであり、ストーリーボードのセット（例えば`Storyboard1~Storyboardn`）を有するキャンバスがあり、そのセットのそれぞれのストーリーボード（例えば`Storyboard1`）は、場合によっては複数の`SetterTimelines`、例えば`SetterTimelines1~SetterTimelinesm`（または後述のような、プロパティを有する`ParallelTimeline`など、同等のエレメントツリー関連付けメカニズム）を有するタイムラインを含む。このキャンバスはまた、以降にも記載するイベントトリガ情報、ならびに動画化されたプロパティを有するターゲットとしてストーリーボードに関連付けられた（スタイル識別子およびファイル識別子などの）エレメントおよびリソースに関する情報を含むこともできる。

【0072】

容易に理解できることだが、本発明の様々な態様を実施するための代替方法が実行可能である。例えば`SetterTimeline`を使用する代わりに、代替の関連付けメカニズムは、`Storyboard`クラス上の添付プロパティを介して、同様の関連付け情報を指定する。これによって、メディアをストーリーボードに組み込むことが容易になるという点に留意されたい。この代替方法用のXAMLの例を以下に示す。

【0073】

【表4】

```
<Canvas.Storyboards>
  <MediaTimeline Source = "my.wmv" Storyboards.TargetID = "myvideo">
    <ParallelTimeline Storyboard.TargetID="myButton"
Storyboard.Property="Width" Storyboard.Value=...>
      <LengthAnimation .../> //no SB.TargetID on nested targets
    </ParallelTimeline>
    <LengthAnimation Storyboard.Property="..." Storyboard.
  </Canvas.Storyboards>

<Video ID= ... // etc.
```

【0074】

上記の例では、何らかのメディアファイル（`my.wmv`）をストーリーボードに関連付けるための`MediaTimeline`、およびストーリーボードプロパティが設定される`ParallelTimeline`がある（説明を簡単にするために、いくつかの省略記号が示されている）という点に留意されたい。「`ParallelTimeline`」は、前に理解したように、基本的にストーリーボードタイムラインであって、他のタイプのタイムラインと区別するために、変更された名前を有し、（上記の）`MediaTimeline`および（後述する）`SequentialTimeline`を含むという点に留意されたい。しかし`ParallelTimeline`はまた、ストーリーボード用のプロパティ、例えば（`TargetID`、`PropertyPath`、および`Value of a SetterTimeline`の代わりに）`Storyboard.Property`、`Storyboard.Value`、および`Storyboard.Target`を使用することによって、`SetterTimeline`の必要性を解消するが

、これによって、添付プロパティが長くなり、見つかりにくくなる傾向がある。一般にストーリーボードプロパティは、それぞれのストーリーボードに関して、何らかのターゲットプロパティ（例えば `TargetID`）があり、他のストーリーボードプロパティがその `TargetID` と同じタイムライン上にあり、ネスト化された `Targets` がない限り、いかなるタイムラインにも添付することができる。 `TargetID` がまったく指定されていない場合、そのターゲットは、ストーリーボードをホストするエレメント、例えば前の例では `Canvas` であるとみなされるという点に留意されたい。メディアに関しては、 `MediaTimelines` は、 `Storyboard.Property` または `Storyboard.Value` を指定しない。

【0075】

次のテーブルは、 `Storyboard` クラスならびにそのオブジェクトプロパティおよびメソッドの一例について記述している。

【0076】

【表5】

```
Public static class Storyboard
{
    // Dependency Properties
    public static readonly DependencyProperty TargetIDProperty;
    public static readonly DependencyProperty ValueProperty;
    public static readonly DependencyProperty PropertyPathProperty;
    public static readonly DependencyProperty SubPropertyPathProperty;

    // Static Methods
    public static void SetTargetID(DependencyObject d, String targetID);
    public static string GetTargetID(DependencyObject d);
    public static void SetValue(DependencyObject d, String targetID);
    public static string GetValue(DependencyObject d);
    public static void SetPropertyPath(DependencyObject d, String targetID);
    public static string GetPropertyPath(DependencyObject d);
    public static void SetSubPropertyPath(DependencyObject d, String targetID);
    public static string GetSubPropertyPath(DependencyObject d);
}
```

【0077】

エレメントツリーをストーリーボードのタイムラインツリーに関連付けるための他の代替方法が、以下の XAML の例に示されている。

【0078】

【表6】

```
<Canvas.Storyboard>
  <SetterTimeline>
    <SetterTimeline.Value>
      <MediaTimeline>
    </SetterTimeline.Value>
  </SetterTimeline>
  <SetterTimeline Storyboard.Target="myButton" Property="Width" Value=...>
    <LengthAnimation .../>
  </SetterTimeline>
</Canvas.Storyboards>

<Video ID= ... // etc.
```

【0079】

この上記の代替方法では、 `MediaTimeline` は、 `SetterTimeline` の子ではないが、タイムライン上の値ではなく、タイムラインツリーの一部であるためには、 `SetterTimeline` の子となるべきであるという点に留意されたい。したがってこれは、一般にあまり望ましくないシンタックスである。

【0080】

10

20

30

40

50

【表 7】

```

<Canvas.Storyboard>
  <MediaSetterTimeline>
    <MediaTimeline Source = "my.wmv" TargetID = "myvideo">
  </MediaSetterTimeline>
  <SetterTimeline TargetID="myButton" Property="Width" Value=...>
    <LengthAnimation ..../>
  </SetterTimeline>
</Canvas.Storyboards>

<Video ID= ... // etc.

```

10

【0081】

上記の代替方法では、MediaSetterTimelineという別のクラスが追加されているという点に留意されたい。

【0082】

【表 8】

```

<Canvas.Storyboard>
  <SetterTimeline TargetID="myvideo" Property="Source" >
    <MediaTimeline Source = "my.wmv" >
  </SetterTimeline>
  <SetterTimeline TargetID="myButton" Property="Width" Value=...>
    <LengthAnimation ..../>
  </SetterTimeline>
</Canvas.Storyboards>

<Video ID= ... // etc.

```

20

【0083】

この上記の代替方法は、アニメーションパターンに従っているが、値および子が同じプロパティの値になりたがるため、またそれによって、MediaTimelineを幅上に置くことが認められるため、混乱が生じる傾向がある。

【0084】

多数の代替方法が可能ではあるが、説明上の目的から、一般に本明細書における他の例については、SetterTimelineの使用を介して説明する。

30

【0085】

一実装形態では、ストーリーボードは、FrameworkElement上のタイムラインコレクションプロパティとすることができ、このタイムラインコレクションは、最初にエレメントに適用されたときに、そのエレメント上でクロックへとインスタンス化される。Storyboardsプロパティ内の値は、例えばそれぞれのタイムライン階層の葉(leaves)が(一実装形態では、プロパティの基本値を設定するために使用される)SetterTimelineであるIList<Timeline>などのタイムラインインスタンスか、またはこの実装形態ではSetterTimelineの下にアニメーションを有する(基本値を修正するために使用される)アニメーションインスタンスである。SetterTimelineはTimelineであり、したがって子を有することができるが、一実装形態では、SetterTimelineの唯一の正当な子はAnimationである。

40

【0086】

Storyboardsプロパティは依存プロパティ(dependency property)ではないという点に留意されたい。ストーリーボードプロパティ内のタイムラインインスタンスのそれぞれは、潜在的に階層を表している。これらのタイムライン階層は、TimelineClock階層へとインスタンス化され、ホストのFrameworkElement内に保存される。次いでこれらの個々のクロックインスタンスは、(ストーリーボードからのタイムラインの1つであることが必要とされる)Timelineを入力として受け

50

取る `FindStoryboardClock` API を使用して得ることができ、この API を使用して、関連付けられているクロックを見つけ出す。

【0087】

最新の実装形態では、ストーリーボードは、ホストのフレームワークエレメント上で `SupportInitialize`、`EndInit` がコールされたときに処理される。いったんストーリーボードが処理されると、`FrameworkElement.Storyboards` は読み取り専用にすることができ、変更が処理されないという事実を反映する上で役に立つ。

【0088】

前述のように、`SetterTimeline` は、ホストの `FrameworkElement` にとって、または指定された場合はターゲットの `FrameworkElement` にとって暗黙のうちにターゲットとすることができるプロパティ、および任意選択で値を指定するためのタイムラインである。指定された場合、`TargetID` は、ホストの `FrameworkElement` の子孫 (descendency) における `FrameworkElementID` への参照である。`SetterTimeline` 上の `Path` (本明細書では `PropertyPath` と同義である) プロパティは、どのプロパティがストーリーボードを介して動画化されているか、また必要とされているかを記述する。

【0089】

`SetterTimeline` の `Value` プロパティも、任意選択である。指定されている場合、これは、`Timeline` がアクティブなときにプロパティがとるべき基本値を示す。この基本値は、まったくアニメーションがない場合か、またはアニメーションが暗黙の値 (implied value) を必要とする場合に使用される。この基本値は、(エレメント上に直接設定される) ローカルな値から、継承から、スタイリングなどから、同等に由来することができる。例えば以下の例は、10～20 の値からのプロパティを動画化する。

【0090】

【表9】

```
...
<SetterTimeline PropertyPath="..." Value="10">
<LengthAnimation To="20"/>
</SetterTimeline>
...
```

【0091】

上記の `LengthAnimation` が `From` と `To` の双方の値を指定するならば、基本値 (`SetterTimeline.Value`) は無視されるであろうという点に留意されたい。

【0092】

一実装形態では、ターゲットは、`Storyboards` プロパティのホストの `FrameworkElement` と同じツリー内で、`FrameworkElement` への参照である必要がある。`PropertyPath` の第1のコンポーネントは、依存プロパティに裏付けられたプロパティ (dependency property-backed property) である必要がある。例えば `Window.WindowAutoLocation` は、依存プロパティではないため、ストーリーボードには対応 (storyboard-able) しない。

【0093】

プロパティは、インデックス付きのコレクション内のアイテムとすることができる。`SetterTimeline` 上の `PropertyPath` (または `Path`) 属性は、オブジェクト内に「簡単に書き留める」ことができ、プロパティを添付した特別なケースは、丸かっこ内に挿入される。例えば「ブラシ」が `Brush` である場合、次のプロパティ

10

20

30

40

50

ィパスは、線形階調ブラシ(linear gradient brush)のストップの色上のアルファにアクセスする。

【0094】

brush.Stops[1].Color.A

【0095】

その一方で、次のパスは、エレメント上のドッキングしている添付プロパティ(docking attached property)にアクセスする。

【0096】

button.(DockPanel.Dock)

【0097】

上記のプロパティパスは、添付されているプロパティのシンタックス内にあるが、実際には添付されているプロパティではないという点に留意されたい。すなわち BackgroundPropertyはButton自体の上に定義され、共通言語ランタイムプロパティアクセッサはボタン上にあるが、これは、静的なSet/GetMethodアクセッサ(static set/get method accessor)を有する従来の添付されているプロパティとは対照的である。コンパイル時には、TargetIDは特定のタイプに分解することができず、したがって「Background」プロパティがどのクラスに分解されるかが分からないため、これが必要となる。PropertyPathは、次のように定義される。

【0098】

【表10】

```
public sealed class PropertyPath
{
    public PropertyPath(string path, params DependencyProperty[]
    pathParameters);
    public PropertyPath(DependencyProperty dependencyProperty);
    public string Path { get; set; }
    public DependencyProperty[] PathParameters { get; set; }
}
```

【0099】

複合プロパティパス、すなわち指定された複数のコンポーネントを有するパスについて、以降で説明する。

【0100】

本発明の一態様によれば、タイミングツリーをエレメントツリーと統合することに加えて、ストーリーボードは、プロパティトリガおよびイベントトリガを使用して、それらのオペレーション（例えば開始、一時停止、検索、および停止のアクション）を制御することを可能にする。一般にプロパティトリガは、真や偽などの何らかの状態／値に対応し、その一方でイベントが始動する（したがって、偽の値を有さない）。例えばプロパティトリガは、マウスがエレメント上に存在していた場合は真の値を、エレメント上に存在していなかった場合は偽の値を有することができる。それぞれの真または偽の状態は、（1つまたは複数の）ストーリーボード内で別々に開始、停止、一時停止、および／または検索を行うために使用することができる。エレメントでのマウスのクリックなどのイベントは、始動し、したがって可変の状態を有さないが、同様にストーリーボードのオペレーションを制御することができる。このようにして、アニメーションは、ユーザと表示されたエレメントツリーの表示との対話を介して制御することができる。

【0101】

Triggersは、FrameworkElement上のコレクションプロパティである。トリガは、SetActionおよびTimelineActionを含むアクションを指定することができる。SetActionは、プロパティの値を指定する。TimelineActionは、Storyboardsプロパティ内のタイムラインを

10

20

30

40

50

操作する。

【0102】

例えば次の例は、ボタンにマウスオーバー上の特別なバックグラウンドを与える。

【0103】

【表11】

```
<Window.Triggers>
  <PropertyTrigger TargetID="button" Property="IsMouseOver"
    Value="true">
    <SetAction Path="(Button.Background)" Value="red"/>
  </PropertyTrigger>
</Window.Triggers>

<Button ID="button">OK</Button>

</Window>
<Window>
```

10

【0104】

TriggerActionおよびSetActionは、次のように定義される。

【0105】

【表12】

```
abstract class TriggerAction
{
}

class SetAction : TriggerAction
{
    String TargetID {get; set; }
    DependencyProperty Property {get; set; }
    PropertyPath Path{ get; set; }
    Object Value {get; set; }
}
```

20

30

【0106】

SetActionは、SetterTimelineのパターンに従うが、Timelineではないという点に留意されたい。

【0107】

TimelineActionは抽象型クラスであり、動詞メソッド(verb method)のそれぞれ用のサブクラスをTimelineClock上に有する。

【0108】

40

【表 1 3】

```

abstract class TimelineAction : TriggerAction
{
    Timeline TargetID {get; set; }
}

class BeginInAction : TimelineAction
{
    int Offset {get; set; }
}

...

```

10

【0109】

TimelineActionのTargetIDは文字列であり、この文字列はその時点の範囲内で分解され、タイムラインへと分解される。

【0110】

前述のように、プロパティトリガに加えてイベントトリガがあり、イベントトリガは、ホストエレメント上でそのイベントが始動したときにアクションを起こさせるイベントをそのソースとして受け取る。プロパティトリガとは異なり、イベントトリガは範囲が設定されておらず、オペレーションを実行し、そのオペレーションが自身のライフタイムを決定するという点に留意されたい。例えば1つのイベントは1つのタイムラインを開始することができ、そのタイムラインは、その存続期間が終了するまで、または別のイベントトリガなどによって停止されるまで、継続することができる。

20

【0111】

次の例は、トリガについて一般的に説明している。

【0112】

【表 1 4】

```

class Trigger
{
    TriggerActionCollection TriggerActions {get;}
}

class PropertyTrigger : Trigger
{
    String TargetID {get; set;}
    DependencyProperty Property {get; set;}
    PropertyPath Path {get; set;}
    Object Value { get; set; }
}

class EventTrigger : Trigger
{
    String SourceID { get; set; }
    RoutedEventID Event { get; set; }
}

```

30

40

【0113】

次のコード例は、関連付けられているタイムラインと共にイベントトリガを使用して、「オン」ボタンおよび「オフ」ボタンによって操作される、タイムラインによって制御さ

50

れるコンテンツを示している。

【0114】

【表15】

```
<Window>

  <Window.Storyboards>
    <SequentialTimeline ID="myTimeline" >
      <SetterTimeline TargetID="foo" .../>
      <SetterTimeline TargetID="bar" .../>
      ...
    </SequentialTimeline>
  </Window.Storyboards>

  <Window.Triggers>
    <EventTrigger SourceID="b1" Event="Click">
      <BeginInAction TargetID="myTimeline" Offset="0"/>
    </EventTrigger>
    <EventTrigger SourceID="b2" Event="Click">
      <EndInAction TargetID="myTimeline" Offset="0"/>
    </EventTrigger>
  </Window.Triggers>

  <DockPanel>
```

10

20

【0115】

【表16】

```
<DockPanel DockPanel.Dock="Top">
  ... (content) ...
</DockPanel>

  <Button ID="b1" DockPanel.Dock="Left">Start</Button>
  <Button ID="b2" DockPanel.Dock="Right">Stop</Button>
</DockPanel>

</Window>
```

30

【0116】

上記のコードは、独立したタイムラインではなく、シーケンシャルタイムラインを使用しているという点に留意されたい。シーケンシャルタイムラインは、その子タイムラインを順番にトリガさせる。例えば「bar」（図示せず）に対応するいずれかのストーリーボードは、「foo」の完了によってトリガされ、いずれのプログラムによって指定された時間データによってもトリガされない。

40

【0117】

イベントトリガに加えて、プロパティ入力／終了トリガも、タイムラインをトリガする際に使用することができ、これについては、次の例に一般的に示されている。

【0118】

【表 17】

```

<PropertyTrigger Property="MouseOver" Value="True">
  <PropertyTrigger.Enter>
    <BeginInAction Offset="0" TargetId="myTimeline"/>
  </PropertyTrigger.Enter>
  <PropertyTrigger.Exit>
    <EndInAction Offset="0" TargetId="myTimeline"/>
  </PropertyTrigger.Exit>
</PropertyTrigger>

```

10

【0119】

本発明の別の態様によれば、ストーリーボードおよびトリガは、スタイル付けすることができる（ただし、*Storyboards*そのものは依存プロパティではないため、完全に依存プロパティと同様にはならない）。すなわち前述のように、*Style*は作成者が、1つまたは複数のプロパティを表すデータをグループとしてまとめることを可能にし、次いでこれは、エレメントの対応するプロパティに1つのセットとして適用することができる。*Style*内のこれらのプロパティは、動画化するか、および／またはトリガに

20

【0120】

【表 18】

```

<DockPanel>
  <DockPanel.Resources>
    <Style>
      <Button/>
      <Style.Storyboards>
        <SequentialTimeline RepeatDuration="Indefinite">
          <SetterTimeline
            Path="(Button.Background)"
            Value="Red"
            Duration="1"/>
          <SetterTimeline
            Path="(Button.Background)"
            Value="Blue"
            Duration="1"/>
        </SequentialTimeline>
      </Style.Storyboards>
    </Style>
  </DockPanel.Resources>
  ...
</DockPanel>

```

30

40

【0121】

理解できることだが、トリガ、タイムライン、ストーリーボード、マークアップ内でローカルに設定された値、およびその他を含む様々なメカニズムからエレメントプロパティの値を制御する機能によって、プロパティ値間の競合解消(conflict resolution)に対処して、何をプロパティ値とすべきかを決定する必要がある。例えば*Storyboards*プロパティの一部として設定された値は、エレメント上にローカルに設定された値に優先する。したがって、特定の*FrameworkElement*上のプロパティ値の優先順位があり、これは、最も優先順位が高いものとして*Triggers*プロパティから開始する。複数のトリガが同じプロパティを設定しようと試みる場合、「最後に書かれたも

50

のが優先する」(last writer-wins rule)ルールが適用され、例えばマークアップのケースでは、マークアップ内で最後に発生したトリガがその値を適用されるという点に留意されたい。

【0122】

Storyboardsプロパティは、もしも複数のストーリーボードセッターが同じプロパティを設定しようと試みた場合、やはり「最後に書かれたものが優先する」解決策を用いて、次に高い優先順位を有する。最後に書かれたものは、タイムラインツリーのデプスファーストウォーク(depth-first walk)を使用して解釈される。

【0123】

エレメント上にローカルに設定された値(例えば、FrameworkElement.SetValue)が次に来る。一般に、1つのスタイル内で同じ優先順位付けを有するStylesが次に続き、すなわちStyleが(同じレベルで競合が発生する場合は、やはり「最後に書かれたものが優先する」ルールを適用して)トリガされ、次いでStyle Storyboards、そしてStyleのターゲットタイプ値と続く。

【0124】

複合プロパティパスの説明に移ると、セッターにとっては、ターゲット上のプロパティだけでなく、状況によってはターゲットのサブプロパティの1つを修正することも可能である。一般に複合プロパティは、まずエレメントにとってのプロパティ値が何であるかを確立し、次いでそのプロパティをアニメーション用に簡単に書き留めることによって処理される。例えば次のコードは、不透明度を動画化することによって、繰り返し徐々にフェードインおよびフェードアウトする赤色の長方形を例示している。

【0125】

【表19】

```
<Window>

  < Window.Storyboards>

    <SetterTimeline TargetID="rectangle" Begin="0"
      RepeatDuration="Indefinite"
      PropertyPath="{Rectangle.Fill}.(SolidColorBrush.Opacity)" />

    <DoubleAnimation From="0" To="1"
      Begin="0" Duration="5" AutoReverse="true"/>

  </SetterTimeline>

</Window.Storyboards>

<Rectangle ID="rectangle" RectangleWidth="100" RectangleHeight="100"
  Fill="Red" />

</Window>
```

【0126】

一実装形態では、プロパティの第1のコンポーネントは、依存プロパティであることが必要とされる(例えば、上記の例で満たされているBackground)。またこの依存プロパティのタイプは、Changeableであることが必要とされる。これは、コピーを複製する必要があるためであり、これによって、複製されたコピーを修正することができる(別法としては、ディープクローニングソリューション、またはテンプレートとして保持することが含まれる)。これは、スタイルおよびStoryboards/Triggersが、やはり基本的に依存プロパティベースであるために機能するという点に留意されたい。Button.Backgroundに関連するいかなるものも変化しても、すべてのバックグラウンド値が再計算される。

【0127】

したがって上記の例では、まずソリッドカラーブラシ(solid color brush)のコピーが作成され、長方形のインスタンス上に保存される。内部では、ローカルな値は、トリガが非アクティブ化した場合のために保存する必要があるため、実際には置換されない。次いでトリガが、あたかもスクリプトを再生するかのように、そのコピー上へと実行される。

【0128】

したがってプロパティ無効化アルゴリズムは、2つのパスで発生する。すなわち、まさにあたかも複合プロパティパスがないかのようにプロパティの無効化をトリガする第1のパスと、複合プロパティパスを有する（すなわち、その中のドットを有する）トリガが基本の依存プロパティのみを無効化する第2のパスとで発生する。

【0129】

10

動画化を行うために、プロパティ値計算アルゴリズムは、標準的な検索アルゴリズムを使用して基本プロパティ値を見つけ出し、その一方で、その上に複合プロパティパスを有するすべての値を無視する。次いでこのアルゴリズムは、複合プロパティパスを有する適用可能なトリガおよびセッターを見つけ出し、それらを上記の基本値に適用する。これによって、Changeableのコピーを作成することが必要となる場合があるが、これは、可能な場合は常に値を共有することによって最適化することができ、コピーは、必要な場合に作成されるだけである。

【0130】

複合プロパティパスを使用すると、プロパティ再構築の全体にわたって一貫したクロックを維持することが必要となる。例えば次のマークアップコードは、線形階調ブラシのバックグラウンドをストーリーボードから得るボタンを示しており、第1のストップは、1つのプロパティトリガ（マウスがボタン上に存在する場合）によって修正され、第2のストップは、第2のトリガ（ボタンが押された場合）によって修正される。

20

【0131】

【表 2 0】

```

<Window>
  < Window.Resources>
    <Style>
      <Button/>

      < Style.Storyboards>
        <SetterTimeline Start="0" Duration="60"
          Path="(Button.Background)">
          <SetterTimeline.Value>
            <LinearGradientBrush>
              <GradientStop Color="#FF000000"/>
              <GradientStop Color="#00FF0000"/>
            </LinearGradientBrush>
          </SetterTimeline.Value>
        </SetterTimeline>
      </ Style.Storyboards>

      <Style.VisualTriggers>
        <PropertyTrigger Property="IsMouseOver" Value="true">
          <SetAction
            Path="(Button.Background).Stops[0]"
            Value="#80000000"/>
        </PropertyTrigger>
        <PropertyTrigger Property="IsPressed" Value="true">
          <SetAction Path="(Button.Background).Stops[1]"
            Value="#00800000"/>
        </PropertyTrigger>
      </Style.VisualTriggers>

    </Style>
  </ Window.Resources>

  <Button>OK</Button>

</ Window>

```

【0 1 3 2】

この例では、最初のうちは、ボタンのバックグラウンドは、赤からグリーンへの線形階調である。マウスがボタン上へ移動すると、ブラシは、ピンクからグリーンへの線形階調へ変化する。マウスボタンを押すと、色は、マウスボタンが戻るときまで、ピンクからライトグリーンへ線形に変化し、マウスボタンが戻った時点で、バックグラウンドは、ピンクからグリーンへの線形階調ブラシへ戻る。マウスがボタンから離れると、バックグラウンドは、赤からグリーンへの線形階調へ戻る。Set Actionオブジェクト内に複合プロパティパスを有することが可能であるという点に留意されたい。

【0 1 3 3】

このシナリオを適合させることへの第1のステップは、スタイルが適用される際に（すなわちOnStyleInvalidatedの間に）スタイル付けされる親エレメント上にクロックを保存することである。これによって、プロパティが再構築される際に、同じクロックを再利用することができる。第2のステップは、プロパティパスのエラーを無視することである。すなわち最初のわずかな時間が過ぎて、タイムラインが非アクティブになると、ボタンはもはや線形階調ブラシを持たなくなる。この時点では、ストップを修正するトリガはもはや意味をなさなくなり、（データバインディングとの整合性から）暗黙のうちに無視される。

【0 1 3 4】

Storyboardsプロパティから、それぞれのTimelineツリーがTimelineClockツリーへとインスタンス化され、そのStoryboardsをホ

10

20

30

40

50

ストする同じ `FrameworkElement` 内に保存される。それぞれのセッタークロックおよびアニメーションクロックごとに、ターゲットとされているエレメントも、そのクロックへの参照を与えられる。それぞれのセッタークロックごとに、ターゲットとされているエレメントは、そのクロック上のアクティブ／非アクティブなイベントをリスン(listen)し、それらのイベントのそれぞれは、プロパティの無効化をトリガする。それぞれのアニメーションクロックごとに、そのクロックは、そのプロパティ用のターゲットエレメントのアニメーションクロックリストへと組み込まれる。

【0135】

図7は、ストーリーボードがエレメント上に設定され処理される際のステップおよびデータ構造の更新の一例を提供する。一般に図7は、次の簡単な `SetterTimeline` の例のためのオブジェクトおよび参照を示している。

【0136】

【表21】

```
<DockPanel>
  <DockPanel.Storyboards>
    <SetterTimeline TargetID="_rectangle"
      Path="(Rectangle.RectangleWidth)"
      Value="10"
      Start="0" End="10"/>
  </DockPanel.Storyboards>

  <Rectangle ID="_rectangle" RectangleHeight="10"/>
</DockPanel>
```

【0137】

はじめに1つまたは複数の `Timeline` インスタンスが、例えば `DockPanel.Storyboards.Add(new SetterTimeline(. . .))` をコールすることによって、`DockPanel` の `Storyboards` プロパティに追加される。

【0138】

`DockPanel` の `ISupportInitialize.EndInit` がコールされると、(例えば、図7の `DockPanel702` の `SetterTimeline700` に対応する) ストーリーボードが処理され、これによって、`Storyboards` 内のタイムラインのそれぞれが、`TimelineClock.CreateFromTimeline` を介して(例えば、図7で704という番号を付けられている) クロックへとインスタンス化され、`FrameworkElement` (例えば、図7の長方形706) 上に非公式(privately)に保存される。

【0139】

それぞれのセッターのターゲットは、影響を受ける依存プロパティに関連付けられているクロックへの参照を与えられる。このターゲットはまた、(`Begun` and `Ended`) などのクロック境界イベント(clock boundary event)上にリスナ(listener)を設定する。セッターの下すべてのアニメーションが、(`FrameworkElement.GetAnimationClocks` を介して) ターゲットのアニメーションリストに追加される。

【0140】

スタイルが、`FrameworkElement` (それぞれの `FrameworkElement`) 上に設定されると、クロックは、`Style.Storyboards` コレクション内の `Timeline` 階層からインスタンス化される。スタイル付けされた親をターゲットとするセッターのために、`FrameworkElement` は、(関連する依存プロパティに関連付けられている) クロックへの参照を保持し、スタイル付けされた親は、そのクロック用の `Begun/Ended` イベント上にリスナを設定する。 `Sty`

le. Visual Tree内のエレメントをターゲットとするセッターのために、Framework Elementがそのエレメントをインスタンス化すると、同様の参照／リスナが設定される。

【0141】

タイムラインが処理される際には、単一プロパティパス（ただ1つのコンポーネント）を有するセッタータイムラインが、複合パスを有するセッタータイムラインとは別個に維持されるという点に留意されたい。これによって、前述のように、プロパティ値に関する2つのパスの検索が可能となる。

【0142】

（結論）

上述の詳細な説明から分かるように、他の利点の中でもとりわけ、作成者が、アニメーション中に発生する様々なオペレーション（例えば、開始および停止の時期や、イベントに対する応答）を様々なタイムライン上において手動で同期させる必要性を解消するメカニズムがストーリーボードの形態で提供される。さらに、スタイリングシステム(styling system)を介してストーリーボードを利用できることは、それぞれのアニメーションまたはメディアエレメントに個々に適用しなければならないタイミング情報に対する必要性を解消する。

【0143】

本発明は、様々な修正および代替構成を受け入れることができるが、その特定の例示的な実施形態について図示し、詳細に前述した。しかし開示された特定の形態に本発明を限定する意図はなく、むしろ本発明の趣旨および範囲内に収まるすべての修正、代替構成、および均等物を含むことを意図していることが理解できるはずである。

【図面の簡単な説明】

【0144】

【図1】本発明を組み込むことができる典型的なコンピュータシステムを示すブロック図である。

【図2】本発明の一態様によるメディア統合レイヤーアーキテクチャを示すブロック図である。

【図3】本発明の一態様による、ビジュアルAPIレイヤーとのインターフェースをとるためにマークアップ言語コードを解読するためのコンポーネントを示す図である。

【図4】本発明の一態様による、クロックプロパティデータを、進展データを測定する際に使用するためのインターバルに変換するためのタイミングコンポーネントを有する2つのレベルのアーキテクチャを全般的に示すブロック図である。

【図5】本発明の一態様による、クロックプロパティデータを、進展データを測定する際に使用するためのインターバルに変換するためのタイミングコンポーネントを有する2つのレベルのアーキテクチャを全般的に示すブロック図である。

【図6】本発明の一態様による、タイムラインをエレメントツリーのエレメントに関連付けるためにコード内でストーリーボードをどのように構成できるかを示す図である。

【図7】本発明の一態様による、ストーリーボードのタイムラインを表すためにインスタンス化されたクロックを含むデータ構造間の関係を示す図である。

【符号の説明】

【0145】

- 200 メディア統合レイヤ
- 202 アプリケーション／制御／UIコンポーネント
- 204 メディアAPI
- 206 高いレベルの合成およびアニメーションエンジン
- 208 タイミングおよびアニメーション
- 210 低いレベルの合成およびアニメーションエンジン
- 212 グラフィックスサブシステム（ソフトウェアおよびハードウェア）
- 220 印刷レベル

10

20

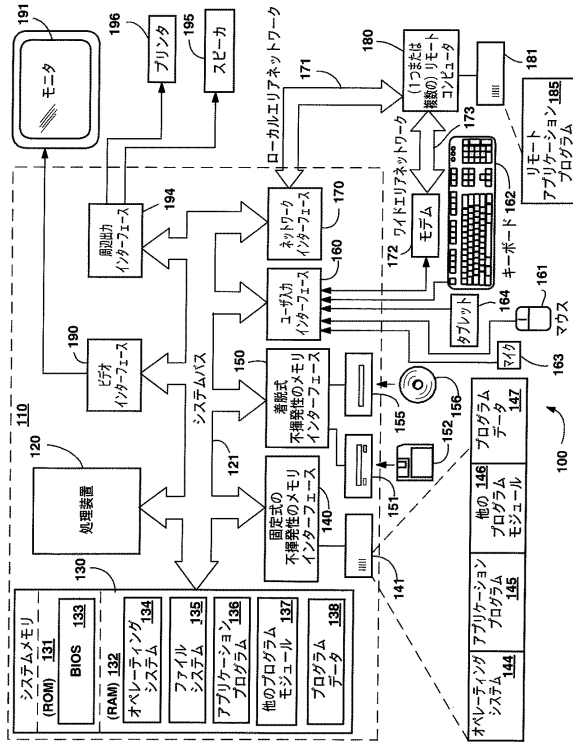
30

40

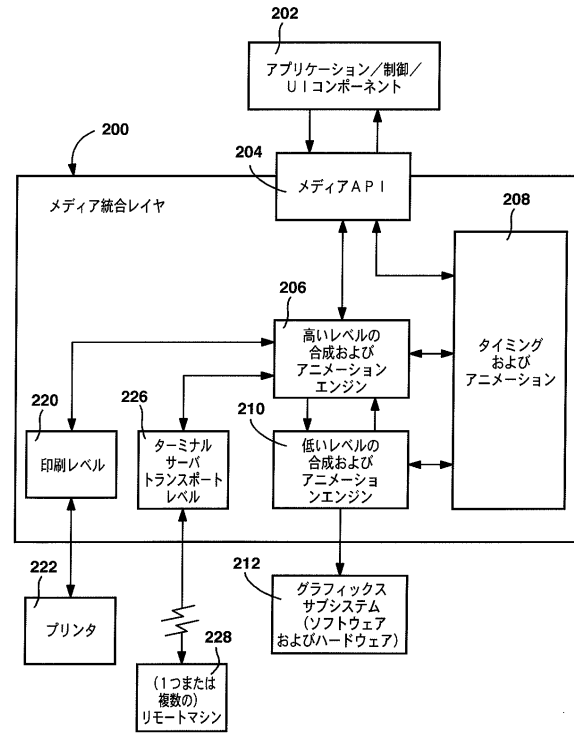
50

2 2 2	プリンタ	
2 2 6	ターミナルサーバトランスポートレベル	
2 2 8	1つまたは複数のリモートマシン	
3 0 2	マークアップ	
3 0 4	パーサ／トランスレータ	
3 0 8	タイプコンバータ	
3 1 0	他のプログラミング言語	
3 1 2	リソースレベルデータ	
3 1 4	エレメントツリー／プロパティシステム	
3 1 6	ビジュアル A P I	10
4 0 2	タイミングツリー	
4 0 4 ₁	クロック ₁ のプロパティ	
4 0 4 ₂	クロック ₂ のプロパティ	
4 0 4 ₃	クロック ₃ のプロパティ	
4 0 6 ₁	イベントリスト ₁	
4 0 6 ₂	イベントリスト ₂	
4 0 6 ₃	イベントリスト ₃	
4 0 8	イベントリストジェネレータ (状態マシン)	
4 1 0	インターバルジェネレータ	
4 1 2 ₁	インターバルリスト ₁	20
4 1 2 ₂	インターバルリスト ₂	
4 1 2 ₃	インターバルリスト ₃	
4 1 4	低いレベルの計算エンジン	
5 2 0 _H	アニメーション関数サブシステム	
5 2 0 _L	アニメーション関数サブシステム	
5 2 4	クロック	

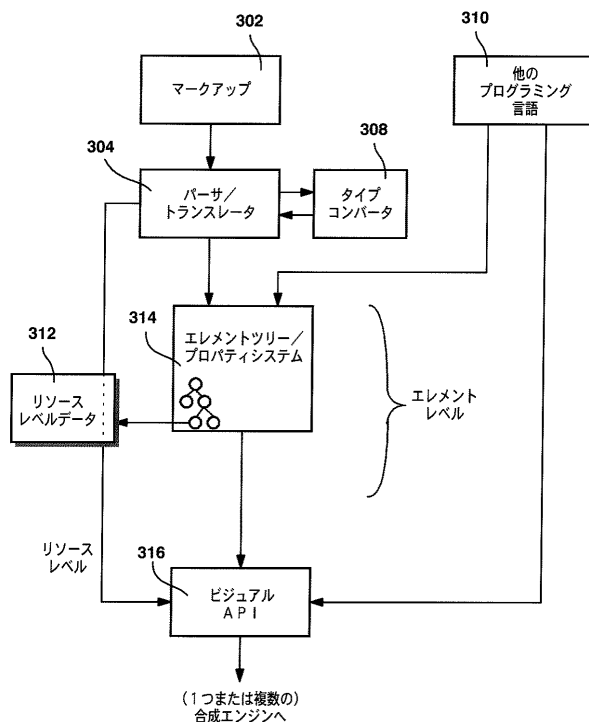
【図 1】



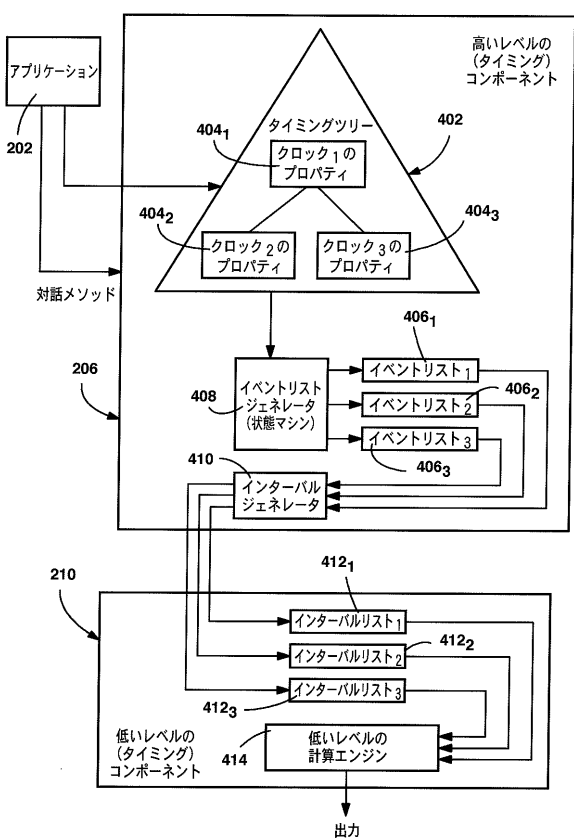
【図 2】



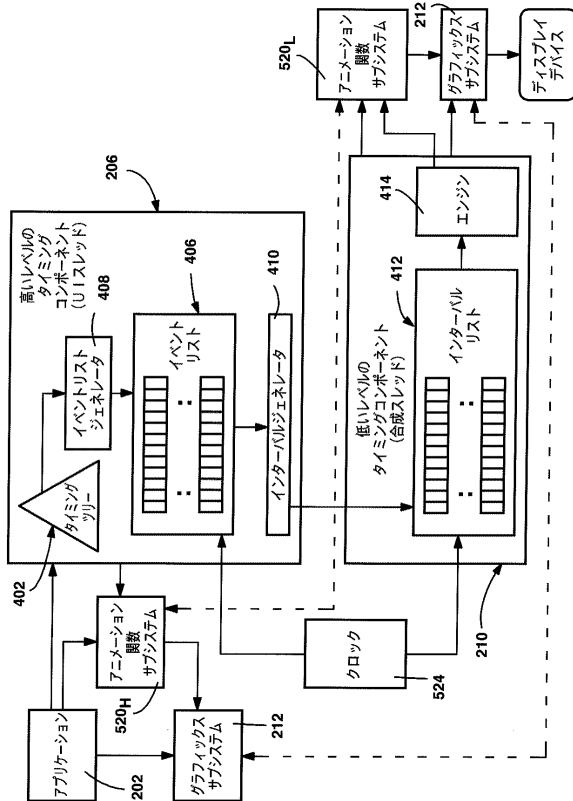
【図 3】



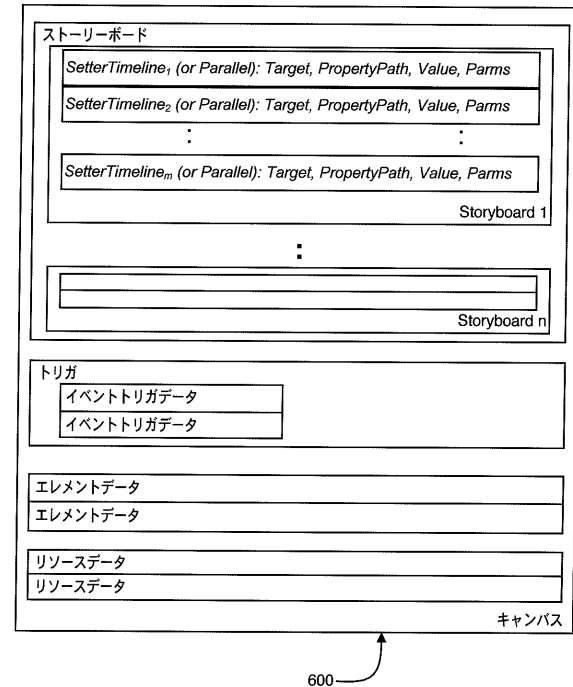
【図 4】



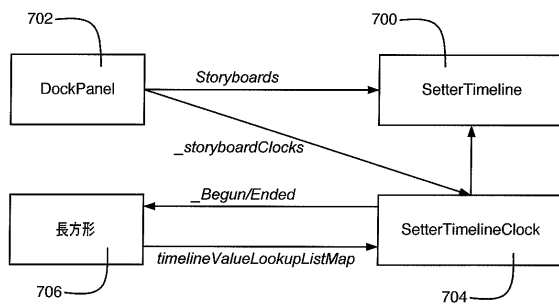
【図 5】



【図 6】



【図 7】



フロントページの続き

- (74)代理人 100091063
弁理士 田中 英夫
- (74)代理人 100153028
弁理士 上田 忠
- (74)代理人 100120112
弁理士 中西 基晴
- (74)代理人 100113974
弁理士 田中 拓人
- (72)発明者 エリザベス ケー. ネルソン
アメリカ合衆国 98052 ワシントン州 レッドモンド ワン マイクロソフト ウェイ マ
イクロソフト コーポレーション内
- (72)発明者 グレッグ ディー. シェクター
アメリカ合衆国 98052 ワシントン州 レッドモンド ワン マイクロソフト ウェイ マ
イクロソフト コーポレーション内
- (72)発明者 レオナルド イー. ブランコ
アメリカ合衆国 98052 ワシントン州 レッドモンド ワン マイクロソフト ウェイ マ
イクロソフト コーポレーション内
- (72)発明者 マシュー ダブリュー. コーキンズ
アメリカ合衆国 98052 ワシントン州 レッドモンド ワン マイクロソフト ウェイ マ
イクロソフト コーポレーション内
- (72)発明者 マイケル ジェイ. ヒルベルグ
アメリカ合衆国 98052 ワシントン州 レッドモンド ワン マイクロソフト ウェイ マ
イクロソフト コーポレーション内
- (72)発明者 ナミタ グブタ
アメリカ合衆国 98052 ワシントン州 レッドモンド ワン マイクロソフト ウェイ マ
イクロソフト コーポレーション内
- (72)発明者 スリラム サブラマニアン
アメリカ合衆国 98052 ワシントン州 レッドモンド ワン マイクロソフト ウェイ マ
イクロソフト コーポレーション内
- (72)発明者 カート ジャコブ
アメリカ合衆国 98052 ワシントン州 レッドモンド ワン マイクロソフト ウェイ マ
イクロソフト コーポレーション内
- (72)発明者 ケネス エル. ヤング
アメリカ合衆国 98052 ワシントン州 レッドモンド ワン マイクロソフト ウェイ マ
イクロソフト コーポレーション内
- (72)発明者 パトリック ミューレン
アメリカ合衆国 98052 ワシントン州 レッドモンド ワン マイクロソフト ウェイ マ
イクロソフト コーポレーション内

審査官 岡本 俊威

(56)参考文献 特開2002-44594 (JP, A)

(58)調査した分野(Int.Cl., DB名)

G06T 13/00

G06T 13/80