(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3770616号
(P3770616)
(45)発行日　平成18年4月26日(2006.4.26)　(24)登録日　平成18年2月17日(2006.2.17)

(51)Int.Cl.			F I		
G06F	9/54	(2006.01)	G06F	9/46	480A
G06F	9/46	(2006.01)	G06F	9/46	420A
H04N	7/10	(2006.01)	H04N	7/10	

請求項の数 20　(全 33 頁)

(21)出願番号	特願平7-509147
(86)(22)出願日	平成6年1月6日(1994.1.6)
(65)公表番号	特表平9-503081
(43)公表日	平成9年3月25日(1997.3.25)
(86)国際出願番号	PCT/US1994/000131
(87)国際公開番号	WO1995/008149
(87)国際公開日	平成7年3月23日(1995.3.23)
審査請求日	平成13年1月9日(2001.1.9)
(31)優先権主張番号	08/120,879
(32)優先日	平成5年9月13日(1993.9.13)
(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　チンデル，ジェイムス，マイケル
アメリカ合衆国　９４０２０　カリフォルニア州　ラ　ホンダ　ピー．オー．ボックス　２４３

最終頁に続く

(54)【発明の名称】オブジェクト指向ビデオ・システム

(57)【特許請求の範囲】

【請求項１】

アドレス空間内に位置するソースとアドレス空間内に位置するデスティネーションとの間でグラフィック・イメージ・データをルーティングするための方法であって、ソースおよびデスティネーションは共用メモリを有し、前記方法は、

（ａ）ソース・アドレス空間がデスティネーション・アドレス空間と異なる場合に、グラフィック・イメージ・データを共用メモリにコピーするステップと、

（ｂ）ソース・アドレス空間がデスティネーション・アドレス空間と同じ場合に、グラフィック・イメージ・データに対するポインタを共用メモリにコピーするステップと、

（ｃ）グラフィック・イメージ・データが処理のために使用可能であることを示す通知を、ソースからデスティネーションに送出するステップと

を備えたことを特徴とする方法。

【請求項２】

請求項１に記載の方法において、

（ｄ）共用メモリの情報を使用して、グラフィック・イメージ・データを処理するステップと、

（ｅ）処理を完了したという通知をデスティネーションからソースへ送出するステップと、

（ｆ）通知を受け取った時に、共用メモリ内のグラフィック・イメージ・データのコピーを削除するステップと

をさらに備えたことを特徴とする方法。
【請求項３】
請求項２に記載の方法において、ステップ（ｄ）は、
（ｄ１）情報がグラフィック・イメージ・データのコピーである場合に、情報に対するポインタを共用メモリに渡すステップと、
（ｄ２）情報がポインタである場合に、情報を共用メモリに渡すステップと
を備えたことを特徴とする方法。
【請求項４】
請求項１に記載の方法において、ソースは、グラフィック・イメージ・データを生成するタスクおよび出力ポート・オブジェクトを備えており、ステップ（ａ）は、
（ａ１）グラフィック・イメージ・データに対するポインタをタスクから出力ポート・オブジェクトへ渡すステップと、
（ａ２）タスクをブロックするステップと
を備えたことを特徴とする方法。
【請求項５】
請求項４に記載の方法において、ステップ（ａ１）は、
（ａ１ａ）ステップ（ａ１）で渡されるポインタを使用して、グラフィック・イメージ・データを共用メモリにコピーするステップ
をさらに備えたことを特徴とする方法。
【請求項６】
請求項４に記載の方法において、デスティネーションはタスクおよび入力ポート・オブジェクトを備えており、ステップ（ａ）は、
（ａ３）グラフィック・イメージ・データのコピーに対するポインタを出力ポート・オブジェクトから入力ポート・オブジェクトへ渡すステップと、
（ａ４）共用メモリのグラフィック・イメージ・データがレディでない場合に、タスクをブロックするステップと、
（ａ５）ステップ（ａ３）で渡されたポインタを入力ポート・オブジェクトからタスクへ渡し、共用メモリのグラフィック・イメージ・データがレディである場合に、タスクをブロック解除するステップと
をさらに備えたことを特徴とする方法。
【請求項７】
請求項１に記載の方法において、ソースは、グラフィック・イメージ・データを生成するタスクおよび出力ポート・オブジェクトを備えており、ステップ（ｂ）は、
（ｂ１）グラフィック・イメージ・データに対するポインタをタスクから出力ポート・オブジェクトへ渡すステップと、
（ｂ２）タスクをブロックするステップと
を備えたことを特徴とする方法。
【請求項８】
請求項７に記載の方法において、ステップ（ｂ１）は、
（ｂ１ａ）ステップ（ｂ１）で渡されるポインタを共用メモリにコピーするステップ
をさらに備えたことを特徴とする方法。
【請求項９】
請求項７に記載の方法において、デスティネーションはタスクおよび入力ポート・オブジェクトを備えており、ステップ（ｂ）は、
（ｂ３）ステップ（ｂ１）で渡されたポインタを出力ポート・オブジェクトから入力ポート・オブジェクトへ渡すステップと、
（ｂ４）グラフィック・イメージ・データがレディでない場合に、タスクをブロックするステップと、
（ｂ５）ステップ（ｂ３）で渡されたポインタを入力ポート・オブジェクトからタスクへ渡し、グラフィック・イメージ・データがレディである場合に、タスクをブロック解除す

るステップと
をさらに備えたことを特徴とする方法。
【請求項１０】
請求項１に記載の方法において、
（ｇ）ビデオ・データのストリームを受け取るステップと、
（ｈ）ビデオ・データを複数のグラフィック・イメージに分割し、各イメージがビデオ情報の１つのフレームを表すステップと、
（ｉ）ステップ（ａ）から（ｃ）に従って、各グラフィック・イメージを処理するステップと
をさらに備えたことを特徴とする方法。
【請求項１１】
アドレス空間内に位置するソースとアドレス空間内に位置するデスティネーションとの間でグラフィック・イメージ・データをルーティングするための装置であって、ソースおよびデスティネーションは共用メモリを有し、前記装置は、
ソース・アドレス空間がデスティネーション・アドレス空間と異なる場合に、グラフィック・イメージ・データを共用メモリにコピーする複製装置と、
ソース・アドレス空間がデスティネーション・アドレス空間と同じ場合に、グラフィック・イメージ・データに対する第１ポインタを共用メモリにコピーするロケータと、
グラフィック・イメージ・データが処理のために使用可能であることを示す通知を、ソースからデスティネーションに送出する通知メカニズムと
を備えたことを特徴とする装置。
【請求項１２】
請求項１１に記載の装置において、
共用メモリの情報を使用して、グラフィック・イメージ・データを処理するグラフィック・プロセッサと、
処理を完了したという完了通知をデスティネーションからソースへ送出する通知メカニズムと、
完了通知に応答して、共用メモリの中のグラフィック・イメージ・データのコピーを削除する除去メカニズムと
をさらに備えたことを特徴とする装置。
【請求項１３】
請求項１２に記載の装置において、グラフィック・プロセッサは、
情報がグラフィック・イメージ・データのコピーである場合に、情報に対する第２のポインタを共用メモリに渡すメカニズムと、
情報が第１ポインタである場合に、情報を共用メモリに渡すメカニズムと
を備えたことを特徴とする装置。
【請求項１４】
請求項１１に記載の装置において、ソースは、グラフィック・イメージ・データを生成するタスクおよび出力ポート・オブジェクトを備えており、複製装置は、
グラフィック・イメージ・データに対する第１ポインタをタスクから出力ポート・オブジェクトへ渡すメカニズムと、
タスクをブロックする同期装置と
を備えたことを特徴とする装置。
【請求項１５】
請求項１４に記載の装置において、第１ポインタをタスクから出力ポート・オブジェクトへ渡すメカニズムは、第１ポインタを使用して、グラフィック・イメージ・データを共用メモリにコピーする複写装置をさらに備えたことを特徴とする装置。
【請求項１６】
請求項１４に記載の装置において、デスティネーションがタスクおよび入力ポート・オブジェクトを備えており、複製装置は、

グラフィック・イメージ・データのコピーに対する第２ポインタを出力ポート・オブジェクトから入力ポート・オブジェクトへ渡すメカニズムと、
共用メモリのグラフィック・イメージ・データがレディでない場合に、タスクをブロックする第１同期装置と、
第２ポインタを入力ポート・オブジェクトからタスクへ渡し、共用メモリのグラフィック・イメージ・データがレディである場合に、タスクをブロック解除する第２同期装置と
をさらに備えたことを特徴とする装置。
【請求項１７】
請求項１１に記載の装置において、ソースは、グラフィック・イメージ・データを生成するタスクおよび出力ポート・オブジェクトを備えており、ロケータは、
グラフィック・イメージ・データに対する第１ポインタをタスクから出力ポート・オブジェクトへ渡すメカニズムと、
タスクをブロックする同期装置と
を備えたことを特徴とする装置。
【請求項１８】
請求項１７に記載の装置において、グラフィック・イメージ・データに対する第１ポインタをタスクから出力ポート・オブジェクトへ渡すメカニズムは、第１ポインタを共用メモリにコピーするストレージ・メカニズムをさらに備えたことを特徴とする装置。
【請求項１９】
請求項１８に記載の装置において、デスティネーションはタスクおよび入力ポート・オブジェクトを備えており、ロケータは、
第１ポインタを出力ポート・オブジェクトから入力ポート・オブジェクトへ渡すメカニズムと、
グラフィック・イメージ・データがレディでない場合に、タスクをブロックする同期装置と、
第１ポインタを入力ポート・オブジェクトからタスクへ渡し、グラフィック・イメージ・データがレディである場合に、タスクをブロック解除するメカニズムと
をさらに備えたことを特徴とする装置。
【請求項２０】
請求項１１に記載の装置において、
ビデオ・データのストリームを複数のグラフィック・イメージに分割し、各イメージがビデオ情報の１つのフレームを表すビデオ・プロセッサと、
各グラフィック・イメージを複製装置およびロケータで利用するメカニズムとをさらに備えたことを特徴とする装置。
【発明の詳細な説明】
著作権所有表記
本明細書の一部には、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、米国特許商標庁の特許ファイルまたは記録に記載されている特許ドキュメントまたは特許開示事項を第三者が複製することを妨げるものではないが、その他の場合には、著作権に係る一切の権利を留保する。
発明の分野
本発明は、一般的には、コンピュータ・システムにおける改良に関し、より具体的には、ビデオ情報をマルチメディア・コンポーネント（構成要素）間でルーチング（経路指定）するシステムに関する。
発明の背景
マルチメディアは、コンピュータ・システムのアプリケーションの中でおそらく最も急成長しているアプリケーションであろう。グラフィック（図形）、サウンド（音声）およびイメージング（画像）情報をエンドユーザに提供するためにユーザによるコンピュータの使用がますます増加している。マルチメディア・プレゼンテーション(multimedia presentation)を管理するためのエルゴノミック（人間工学的）インタフェースに対するユーザ

の要求が高まっている。従来は、マルチメディア・プレゼンテーションを実現するためにタイム・マトリックス(time matrix)とプログラミング言語が使用されていた。しかし、ミュージックやサウンドのプレゼンテーションが情報の表示と一緒にマルチメディア・プレゼンテーションとして展開されるように、柔軟性のあるミキシング・ボード(mixing board)をシミュレーションすることは可能ではなかった。

本発明の機能を備えていない現存マルチメディア・システムの例として、アップル社のQuicktimeとマイクロソフト社のVideo for Windowsがある。これらはNEWMEDIAの３月号の"It's Showtime"、pp.36-42(1993)に記載されている。従来技術に当面しているルーチング問題の解決手法を得ることの重要性は、IEEE Spectrumの３月号、“Interactive Multimedia”（「インタラクティブ・マルチメディア」）、pp.22-31(1993)および“Technology Framework”（「テクノロジ・フレームワーク」）、IEEE Spectrum、pp.32-39(1993)に論じられている。これらの論文はマルチメディア・プロダクションを制御するためのエステティック・インタフェース(aesthetic interface)の重要性を指摘している。

発明の概要

以上に鑑みて、本発明の主目的は、マルチメディア・プレゼンテーションが進行している過程で、ストレージ（記憶装置）とディスプレイ（表示装置）を備えたコンピュータを使用してオーディオ・マルチメディア・データを結合し、ルーチング（routing－経路指定）し、フィルタリングするシステムと方法を提供することである。システム内のプロセッサは接続されたストレージと接続されたディスプレイをもち、これらはプロセッサの制御下に置かれている。複数のマルチメディア・オブジェクトはストレージ内に作成され、ディスプレイから表示される。結合オブジェクトもストレージ内に作成され、ディスプレイから表示されるが、これはマルチメディア・オブジェクトをオーディオ・オブジェクトに結合する際に使用される。

【図面の簡単な説明】

図１は、好適実施例によるパーソナル・コンピュータ・システムを示すブロック図である。

図２は、テープ・デッキ、ミキサ、残響ユニット、ペアのマイクロホン、およびペアのスピーカを利用している、従来の単純なホームスタジオ・セットアップ（配置）を示す図である。

図３は、好適実施例によるメディア・コンポーネントを示す図である。

図４は、好適実施例によるオーディオ・プレーヤ・コンポーネントを示す図である。

図５は、好適実施例によるファンイン・ポートとファンアウト・ポートを示す図である。

図６は、好適実施例によるゼロ個または１個以上のオーディオ入力ポートとゼロ個または１個以上のオーディオ出力ポートをもつオーディオ・コンポーネントを示す図である。

図７は、オーディオ・プレーヤの使用によって音声注釈(voice annotaions)をレコーディング（録音）しプレイ（再生）するように働く、好適実施例による音声注釈アプリケーションを示す図である。

図８は、好適実施例による音声メール／電話応答アプリケーションを示す図である。

図９は、好適実施例においてマスタ・クロックと外部から同期がとられるマルチメディア・プレーヤを示す図である。

図10は、好適実施例において１つにミキシングされた、スピーカなどの出力デバイスから出力される３つのサウンド（ミュージック用、サウンド効果用、ナレータ音声用）を示す図である。

図11は、好適実施例に従ってサポートされるオーディオ・タイプの一部を示す図である。

図12は、好適実施例に従ってあるオーディオ・タイプから別のオーディオ・タイプへ変換する変換プロセスを示す図である。

図13は、好適実施例によるリモート・プロシージャ・コール(remote procedure call)を示す図である。

図14は、好適実施例に従ってデータをその入力バッファーから読み取り、そのデータを処理し、処理結果をその出力バッファに書き出す、関連のRun()メンバ関数をもつオーディ

オ・プロセッサ・アーキテクチャを示す図である。
図15は、好適実施例によるオーディオ・プロセッサを示す図である。
図16は、好適実施例に従って図15に示すネットワークで処理がどのように行われるかを示す図である。
図17は、好適実施例によるオーディオ・ポートを示す図である。
図18は、好適実施例によるプレーヤなどのオーディオ・プロセッサを示す図である。
図19は、好適実施例に従ってオーディオ・プロセッサをアクチベートするときに関連するリカーシブ・ロジックを示すフローチャートである。
図20は、好適実施例に従ってオーディオ・プロセッサを非アクチベートするときに関連する詳細ロジックを示すフローチャートである。
図21は、好適実施例に従ってオーディオ・プロセッサを実行するときに関連するロジックを示すフローチャートである。
図22は、好適実施例に従ってビデオ・ディジタイザ・コンポーネントをビュワー・コンポーネントにパッチングしてコンピュータ・スクリーン上に表示する例を示す図である。
図23は、好適実施例に従って２つのビデオ・オブジェクトからのイメージを効果プロセッサでミキシングして、その結果をコンピュータ・スクリーンから表示する例を示す図である。
図24は、好適実施例に従ってグラフィック・ポートがどのように使用されるかを示す図である。
図25は、好適実施例による出力ポートのWriteメンバ関数に関連するロジックを示すフローチャートである。
図26は、好適実施例による入力ポートのRead処理を示す図である。
図27は、好適実施例による入力ポートのnextメンバ処理を示す図である。
図28は、好適実施例に従ってMIDIプレーヤ2800とMIDIインタフェース2810の２コンポーネントがどのように使用されて、コンピュータに結合されたミュージック・シンセサイザをプレイするかを示す図である。
図29は、好適実施例に従ってMIDIデータが外部のミュージック・シンセサイザからどのようにレコーディング（録音）されプレイバック（再生）されるかを示す図である。
図30は、好適実施例に従ってMIDIデータがどのようにプレイバック（再生）されるかを示す図である。
図31は、好適実施例によるMIDIポートとオーディオ・ポートの両方を含んでいるメディア・コンポーネントを示す図である。
図32は、好適実施例に従って共通(common)メッセージ、リアルタイム(realtime)メッセージおよび排他(exclusive)メッセージに区分されたシステム・メッセージの詳細を示す図である。
図33は、好適実施例によるMIDIメッセージのフォーマットの一部を示す図である。
図34は、好適実施例によるMIDIメッセージのタイプと構造、ステータス（状態）バイトおよびタイムスタンプをカプセル化しているMIDIパケット・オブジェクトを示す図である。
図35は、好適実施例によるファンニング(fanning)オペレーションの例を示す図である。
図36は、好適実施例によるMIDI出力ポートのWrite()メンバ関数を示す図である。
図37は、好適実施例によるMIDI入力ポートのRead()メンバ関数を示す図である。
図38は、好適実施例による単一入力ポートと２出力ポートをもつメディア・コンポーネントを示す図である。
図39は、好適実施例に従って時間基準(time-based)メディア・シーケンスをプレイ（再生）できるメディア・コンポーネントの例を示す図である。
図40は、好適実施例に従ってオーディオ・コンポーネントがオーディオ・データをプレイ（再生）し、レコーディング（録音）するためのオーディオ・プレーヤを示す図である。
図41は好適実施例によるスピーカ・コンポーネントの例を示す図である。
図42は好適実施例によるマイクロホン・コンポーネントを示す図である。
図43は好適実施例によるミキサ・コンポーネントの例を示す図である。

図44は好適実施例によるスプリッタ・コンポーネントの例を示す図である。
図45は好適実施例によるゲイン・コンポーネントの例を示す図である。
図46は好適実施例によるエコー・コンポーネントを示す図である。
図47は好適実施例によるファズ(fuzz)コンポーネントを示す図である。
図48は好適実施例によるオーディオ・タイプ・コンバータの例を示す図である。
図49は好適実施例によるオーディオ・マルチコンバータを示す図である。
図50は好適実施例によるサウンド・コンポーネントを示す図である。
図51は好適実施例に従ってサウンド・コンポーネントに組み込まれたコンポーネントを示す図である。
図52は好適実施例による物理スピーカ・コンポーネントを示す図である。
図53は、好適実施例による物理マイクロホン・コンポーネントを示す図である。
図54は、好適実施例によるグラフィック・プレーヤ・コンポーネントを示す図である。
図55は、好適実施例によるグラフィック・ビュワー・コンポーネントを示す図である。
図56は、好適実施例によるビデオ・ディジタイザ・コンポーネントを示す図である。
図57は好適実施例によるMIDIプレーヤ・コンポーネントを示す図である。
図58は好適実施例によるMIDIインタフェース・コンポーネントを示す図である。
図59は好適実施例によるMIDIフィルタ・コンポーネントを示す図である。
図60は、好適実施例によるMIDIマッパ(mapper)コンポーネントを示す図である。
図61は、好適実施例によるMIDIプログラム・マッパ(program mapper)コンポーネントを示す図である。
図62は、好適実施例によるMIDI音符マッパ(note mapper)コンポーネントを示す図である。
図63は、好適実施例によるMIDIチャネル・マッパ(channel mapper)コンポーネントを示す図である。

発明の詳細な説明

本発明は、好ましくは、IBM（登録商標）社PS/2（登録商標）またはアップル（登録商標）社マッキントッシュ（登録商標）コンピュータなどのパーソナル・コンピュータ上に置かれているオペレーティング・システムの環境（コンテキスト）で実施される。代表的なハードウェア環境は図１に示されたいるが、同図は従来のマイクロプロセッサのような中央演算処理ユニット10、およびシステム・バス12を介して相互に接続された複数の他のユニットを装備した、本発明によるワークステーションの代表的ハードウェア構成を示している。図１に示すワークステーションは、ランダム・アクセス・メモリ(RAM)14、リードオンリ・メモリ(ROM)16、ディスク・ユニット20などの周辺デバイスをバスに接続するための入出力アダプタ18、キーボード24、マウス26、スピーカ28、マイクロホン32および／またはタッチ・スクリーン・デバイス（図示せず）などのユーザ・インタフェース・デバイスをバスに接続するためのユーザ・インタフェース・アダプタ22、ワークステーションをデータ処理ネットワーク23に接続するための通信アダプタ34、およびバスをディスプレイ・デバイス38に接続するための表示アダプタ36を装備している。ワークステーションには、アップル社System/7（登録商標）オペレーティング・システムなどのオペレーティング・システムが置かれている。

好適実施例では、本発明は、オブジェクト指向プログラミング技法を使用してC++プログラミング言語で実現されている。この分野に精通しているものならば容易に理解されるように、オブジェクト指向プログラミング(Object-Oriented Programming-OOP)のオブジェクトは、データ構造とデータに対するオペレーション（操作、演算など）からなるソフトウェア・エンティティ(software entity)である。これらのエレメントが協力し合うことにより、オブジェクトは、そのデータ・エレメントで表されたその特性、およびそのデータ操作関数(data manupulation functions)で表されたその作用(behavior)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仮想的(virtually)にモデル化することができる。この方法によると、オブジェクトは人やコンピュータなどの具体物をモデル化することができ、また、数や幾何学的概念などの抽象概念をモデル化することができる。

オブジェクト・テクノロジの利点は次の３つの基本的原理から得られるものである。それはカプセル(encapsulation)、多態(polymorphism)および継承(inheritance)である。
オブジェクトは、そのデータの内部構造と、その関数が実行されるときのアルゴリズムとを隠して外から見えないようにする。つまりカプセル化する。これらのインプリメンテーション細部を見せる代わりに、オブジェクトは、外部情報(extraneous information)のないクリーンな形でその抽象化(abstractions)を表しているインタフェースを提示する。多態はカプセル化をさらに一歩進めたものである。この考え方は、多数の形態をもつ、１つのインタフェースである。あるソフトウェア・コンポーネントは、別のコンポーネントがなんであるかを正確に知らなくても、その別のコンポーネントを要求することができる。要求を受け取ったコンポーネントはその要求を解釈し、その要求をどのように実行すべきかを、その変数とデータに従って判断する。三番目の原理である継承によれば、開発者は既存の設計やコードを再利用することができる。この機能を利用すると、開発者はソフトウェアを初めから作ることが回避される。むしろ、継承を通して、開発者は作用を継承するサブクラスを派生し、その作用を開発者独自の要求に合ったものにカストマイズすることができる。
従来のアプローチでは、オブジェクトとクラス・ライブラリを手続き型環境で階層化する方法がとられていた。市販されている多くのアプリケーション・フレームワークはこの設計手法を採用している。この設計では、モノリシック・オペレーティング・システム(monolithic operating system)の上に１つまたは２つ以上のオブジェクト層が置かれている。このアプローチでは、カプセル化、多態および継承のすべての原理がオブジェクト層に採用され、手続き型プログラミング技法を大幅に改善しているが、このアプローチにはいくつかの制約がある。これらの問題は、開発者が自身のオブジェクトを再利用することは容易であるが、他のシステムからのオブジェクトを使用することが困難であるため、手続き型オペレーティング・システム(OS)のコールを使用して下位の非オブジェクト層まで到達する必要があることに起因している。
オブジェクト指向プログラミングが持つもう１つの側面は、アプリケーションを開発するのにフレームワーク・アプローチを採用していることである。フレームワークの最も合理的な定義の１つとして、イリノイ大学のRalph E.JohnsonとPurdueのVincent F.Russoによるものがある。両氏の1991年の論文“Reusing Object-Oriented Designs”（「オブジェクト指向設計の再利用」、University of Illinois techreport UIUCDCS91-1696）の中で、次のような定義を提案している。「抽象クラスとは、一組の責任分担を協力し合って実行するオブジェクトの集合を設計したものである。従って、フレームワークとは、組で定義された計算責任分担を協力し合って実行するオブジェクト・クラスが集まったものである。」プログラミング側から見たときは、フレームワークとは、基本的には、実働アプリケーション(working application)の事前に作られた構造を提供する、相互に接続されたオブジェクト・クラスの集まりである。例えば、ユーザ・インタフェース・フレームワークは、ウィンドウ(drawing window)、スクロールバー、メニューなどの描画をサポートし、これらの「デフォルト（省略時）」の作用を提供することができる。フレームワークはオブジェクト・テクノロジを基礎においているので、この作用を継承してオーバライド(override)することにより、開発者はフレームワークを拡張し、カストマイズした解決手法を特定の専門分野で作ることができる。これが従来のプログラミングに比べて大きな利点であるのは、プログラマはオリジナル・コードを変更することなく、むしろソフトウェアを拡張できるからである。さらに、フレームワークは、アーキテクチャに関するガイダンスとモデリングを提供すると同時に、問題領域に特有の個々のアクションを特定することから開発者を解放するので、開発者はいくつかのコード層を通って盲目的に作業する必要がない。
ビジネス側から見たときは、フレームワークは、特定の知識分野において専門知識をカプセル化または具現化する方法と見ることができる。企業の開発集団、独立ソフトウェア・ベンダ(Independent Software Vendors-ISV)およびシステム・インテグレータ(systems integrators)は、前述した例のように、製造、会計、通貨取引きといった特定分野におけ

る専門知識をもっている。この専門知識はコードで具現化されている。フレームワークを使用すると、企業は、その専門知識を企業のコードで具現化することにより、専門知識の共通特性を取り込んで、パッケージ化することができる。まず、そのようにすると、開発者は専門知識を利用するアプリケーションを作成または拡張できるので、問題がいったん解決されれば、ビジネス・ルールや設計は統一的に実施され、使用されることになる。また、フレームワークと、フレームワークの背景にある具現化された専門知識は、製造、会計、バイオテクノロジといった垂直的マーケットにおいて専門知識を得た企業にとっては、戦略的資産をもつことを意味し、企業の専門知識をパッケージ化し、再販し、流布し、テクノロジの進歩と普及化を促進するための分配メカニズムをもつことになる。

歴史的には、フレームワークがパーソナル・コンピューティング・プラットフォーム上の主流概念として出現したのは、つい最近のことである。この移行を助長したのが、C++といったオブジェクト指向言語の出現である。従来において、C++は、コマーシャル・ベースのコンピュータよりむしろ、大部分がUNIXシステムと研究者のワークステーション上に搭載されていた。いくつかの大学や研究所のプロジェクトが今日の商用フレームワークおよびクラス・ライブラリの先駆けとなり得たのは、C++などの言語やSmalltalkなどの他のオブジェクト指向言語のお陰である。その例のいくつかを挙げると、スタンフォード大学のInterViews、カーネギ・メロン大学のAndrewツールキット、チューリッヒ大学のET++フレームワークがある。

フレームワークは、どのレベルのシステムに関心があり、どのような問題を解決しようとしているかに応じて、多種類のものがある。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低レベルのフレームワークまでにわたっている。商用化されているアプリケーション・フレームワークの例をいくつか挙げると、MacApp(Apple)、Bedrock(Symantec)、OWL(Borland)、NeXtStep App Kit(NeXT)、Smalltak-80 MVC(ParcPlace)がある。

フレームワークでプログラミングを行うには、他の種類のシステムに慣れている開発者は、新しい考え方が必要がある。確かに、このプログラミングは、従来考えられていたプログラミングとはまったく異なっている。DOSやUNIXなどの旧スタイルのオペレーティング・システムでは、開発者自身のプログラムが構造全体になっている。オペレーティング・システムはシステム・コール(system call)を通してサービスを提供している。つまり、開発者のプログラムはサービスの必要時にコールを行い、サービスが提供されたとき制御が返却される。プログラム構造は抑制の流れに基礎を置き、これは開発者が書いたコードに具現化されている。

フレームワークが使用されるときは、これとは反対である。開発者は制御の流れに責任をもつ必要がなくなる。開発者は、プログラミング・タスクを実行の流れからとらえて理解しようとする癖を止めなければならない。むしろ、オブジェクトの責任分担からとらえて思考する必要があるので、タスクをいつ実行させるかを判断するには、フレームワークに頼らざるを得なくなる。開発者が書いたルーチンは、開発者が書かなかった、従って開発者には見えないコードによってアクチベート(activate－活性化）される。この制御の流れのフリップフロップは、手続き型プログラミングだけに経験のある開発者にとっては、大きな心理的障壁となっている。しかし、このことをいったん理解してしまえば、フレームワーク・プログラミングによると、他のタイプのプログラミングによる場合よりも作業量が大幅に減少することになる。

アプリケーション・フレームワークが開発者にプレハブの機能を提供するのと同じように、本発明の好適実施例に含まれているようなシステム・フレームワークはシステム・レベルのサービスを提供することによって同じ考え方をてこにしている。つまり、システム・プログラマといった開発者は、システム・レベルのサービスをサブクラス化またはオーバライドして、カストマイズした解決手法を作成することができる。例えば、オーディオ、ビデオ、MIDI、アニメーションなどの新規なデバイスや異種デバイスをサポートする基礎を提供できるマルチメディア・フレームワークについて考えてみることにする。新しい種

類のデバイスをサポートする必要がある場合、開発者はデバイス・ドライバを書く必要が起こることになる。フレームワークでこれを行う場合は、開発者に要求されることは、その新デバイスに特有の特性と作用を指定するだけである。
この場合の開発者は、マルチメディア・フレームワークから呼び出されるある種のメンバ関数(member functions）のインプリメンテーションを用意することになる。開発者にとって即時に利点となることは、デバイスのカテゴリ別に必要になる汎用コード(generic code)がマルチメディア・フレームワークにすでに用意されていることである。このことは、デバイス・ドライバ開発者が書き、テストし、デバッグするコードが少なくなることを意味する。システム・フレームワークを使用するもう１つの例は、入出力フレームワークがSCSIデバイス、NuBusカード、およびグラフィックス・デバイス別になっていることである。機能は継承されるので、各フレームワークには、そのデバイス・カテゴリに見られる共通機能に対するサポートが用意されている。この場合、他の開発者は、これらの統一インタフェース(consistent interfaces)を通して、あらゆる種類のデバイスにアクセスすることが可能になる。
好適実施例ではフレームワークという考え方が取り入れられ、この考え方をシステム全体に適用している。業者または企業の開発者、システム・インデグレータ、またはOEMにとっては、このことは、MacAppのようなフレームワークの場合に説明されているすべての利点が、テキストやユーザ・インタフェースといったものではアプリケーション・レベルで生かせるだけでなく、グラフィックやマルチメディア、フィアル・システム、入出力、テストなどのサービスではシステム・レベルでも生かせることを意味する。
好適実施例のアーキテクチャでのアプリケーション作成は、基本的には、フレームワーク・プロトコルに準拠するドメイン（問題領域）別の断片を書くのと似ている。このように、プログラミングという概念全体が変わっている。複数のAPI階層をコール（呼び出すこと）するコードを一行ずつ書くのではなく、ソフトウェアはこの環境内の既存のフレームワークからクラスを導出し、そのあとで、必要に応じて新しい作用(behavior)を追加したりおよび／または継承した作用をオーバライド(override)したりすることによって開発される。
以上のように、開発者のアプリケーションは、書かれたあと、他のすべてのフレームワーク・アプリケーションによって共有されるコードの集合となる。これが強力な概念となっているのは、開発者は各人の作業に積み上げるように構築していくことができるからである。また、開発者は、必要に応じて多かれ少なかれカストマイズする柔軟性も得られる。フレームワークには、そのままで使用されるものがある。ある場合には、カストマイズ量は最小限になるので、開発者がプラグインする断片は小さくなる。他の場合には、開発者は大幅に修正を行って、まったく新しいものを作成することも可能である。
好適実施例では、図１に示すように、マルチメディア・データ・ルーチング・システムがコンピュータ・システムを通るマルチメディア情報の移動を管理し、他方では、RAM 14に常駐しているCPU 10の制御下に置かれている、あるいはバス12または通信アダプタ34を通して外部接続されている複数のメディア・コンポーネントがマルチメディア情報を提示すること（プレゼンテーション）を受け持っている。システムの処理全体を調整または管理するための中央プレーヤ(central player)は不要になっている。このアーキテクチャは柔軟性があり、新しいメディア・タイプが追加されたときの拡張性を向上している。システムは様々なメディア・オブジェクトを利用しており、その一部は結合オブジェクト(connecting object)として使用できるようになっている。結合オブジェクトとしては、ゲイン、フィルタ、増幅器、ミキサ、プレーヤ、その他のマルチメディア・コンポーネントがあり、これらはオブジェクト指向オペレーティング・システムにおけるオブジェクトとして個別的に実現されている。オブジェクトは、上述したように、コードまたはメソッド・コンポーネントとデータ・コンポーネントとを含んでいる。システムには、ドラッグ／ドロップ、ダブルクリック、ドロップ・ラーンチ(drop-launching)、カーソル位置付けなどやその他の代表的なアイコン操作を容易にするマウスが含まれている。
ビデオおよびオーディオ・プロダクション・スタジオでは、サウンド、MIDI、ビデオなど

のメディアは物理的パッチ・コード(patch cord)を利用してソース（送信側）、効果プロセッサ(effects processor)、およびシンク（受信側）の間で信号を転送している。信号処理アルゴリズムも、ソース、シンク、およびプロセッサのネットワークとして表されることがよくある。これらのモデルはどちらも、結合されているオブジェクトの有向グラフ(directed graphs)として表現することができる。好適実施例によれば、このモデル（複数のオブジェクトを１つに結合する）をコンピュータ・システム上に実現することが可能である。図２はテープ・デッキ、ミキサ、残響ユニット(reverberation unit)、ペアのマイクロホン、およびペアのスピーカを利用している従来の単純なホームスタジオ・セットアップ（配置）を示している。マイクロホンはテープ・デッキに接続されているので、サウンド入力はマイクロホンからテープ・デッキへ転送され、そこでサウンドをレコーディング（録音）することができる。テープ・デッキをプレイバック（再生）するときは、テープ・デッキがミキサに接続されているのでその信号はミキサへ送られる。同様に、残響ユニットとスピーカはミキサに接続された増幅器に接続されている。

好適実施例では、オブジェクト指向テクノロジを利用して結合モデル(connection model)を表現している。マルチメディア・オブジェクトを相互に結合することにより、有向データ・フロー・グラフ(directed data flow graphs)を作成することができる。さらに、マルチメディア・オブジェクトの標準セットがシステムに対し定義されている。これらのオブジェクトを共に結合することにより、オブジェクトからオブジェクトへのマルチメディア・データ・フローが容易化される。これらの結合オペレーションは、線(line)や線分(line segment)、その他の適当な幾何学的形状などの幾何学図形を通してマルチメディア・オブジェクトを結合することにより容易化されている。以下に説明する図形は種々のマルチメディア・オブジェクトの例を示しているが、この中には、結合オブジェクトと、マルチメディア・オブジェクトを結合する内部データ構造とロジックを表すために使用される幾何学図形とが含まれている。

ルーチングのためのクラス

時間基準(time-based)メディア・コンポーネント（以下、メディア・コンポーネントと呼ぶ）基底クラス(base class)は、ルーチングのとき使用される中心の抽象クラス(abstraction)である。メディア・コンポーネントはゼロ個または１個以上の入力ポートおよびゼロ個または１個以上の出力ポートをもっている。図３に示す例では、メディア・コンポーネントは１つの入力ポート300と２つの出力ポート310,320をもっている。ポート300,310および320は塗りつぶした三角形で表されている。

メディア・コンポーネントのサブクラスはそれぞれのポートを結合することにより１つに結合される。この処理はレコーディング・スタジオにおいて、パッチ・コードを使用してオーディオ・コンポーネントとビデオ・コンポーネントを１つに結合するのと似ている。図４において、メディア・コンポーネントのサブクラスであるオーディオ・プレーヤ・コンポーネント・オブジェクトは別のメディア・コンポーネント・サブクラスであるスピーカ・コンポーネント・オブジェクトに結合されている。オーディオ・プレーヤは１つの出力ポートをもち、スピーカは１つの入力ポートをもっている。メディア・コンポーネントはメンバ関数コール(member function call)を使用して制御される。例えば、図４に示すオーディオ・プレーヤはオーディオ・データをプレイ（再生）するためのメンバ関数をもっている。オーディオ・プレーヤのメンバ関数Play()がコールされると、オーディオ・データはオーディオ・プレーヤからスピーカへ送られ、そこでオーディオ（音声）がコンピュータのスピーカから聞こえるようにされる。スピーカ・コンポーネントがPlay()関数を持っていないのは、そこに送られてきたどのオーディオ・データでもプレイするからである。メディア・コンポーネントは全体をソフトウェアで実現することができる。しかし、あるメディア・コンポーネントが物理的ハードウェア部分を表すことも可能である。例えば、スピーカ・オブジェクトはコンピュータのプレイバック（再生）・ハードウェアを表すために使用できる。このようにすると、ビデオ・テープ・レコーダ、ミキサ、効果プロセッサなどの外部メディア・デバイスをメディア・コンポーネントとして表して、１つに結合することができる。

メディア・コンポーネントの結合
メディア・コンポーネントはそれぞれのポートを結合することによって１つに結合される。複数のクライアント・オブジェクトとコンポーネントがデータを同じポートに同時に書いてデータ保全性(data integrity)を破壊するのを防止するために、クライアントはポートを直接にアクセスすることが禁止されている。その代わりに、クライアントはマルチスレッド・セーフ(multithread-safe)のサロゲート・オブジェクト(surrogate object)について結合オペレーションを実行する。以下の説明の文脈において、「サロゲート(surrogate)-代用」という用語は基礎となるクラスの特殊表現であり、複数のクライアントがそのクラスのインスタンスを安全に共有できるようにすることを意味する。メディア・コンポーネントの場合には、サロゲート・ポートによると、実際のポートの制限された間接的操作が可能である。すべてのメディア・コンポーネントは、その入力ポートと出力ポートの各々ごとにサロゲートを作成するメンバ関数をもっている。これらのサロゲート・ポートは非常に軽量なオブジェクトであり、アドレス境界をトラバース(traverse-通り抜けること）することに適しているので、異なるアドレス空間に置かれているメディア・コンポーネントの結合が容易化されている。
各ポートとポート・サロゲートはそれぞれに関連づけられたデータ・タイプ(data type-データ型)をもっている。データ・タイプの例としては、MIDIデータ、44kHz 16ビット・オーディオ・データ、22kHz 8ビット・オーディオ・データ、およびビデオのグラフィック・データがある。２つのポートを結合することが要求されると、タイプ・ネゴシエーション・プロトコル(type negotiation protocol)は、これらのポートにコンパチブル（互換性をもつ）データ・タイプをサポートする能力があるかどうかを確かる。こっれらのポートが共通のタイプをもっていなければ、例外が引き起こされる。
非互換性ポート・タイプをもつオブジェクト間にはコンバータ・オブジェクトを挿入することが可能である。コンバータは、あるタイプのデータを受け入れて異なるタイプのデータを出力するコンポーネントである。ファンイン・ポートとファンアウト・ポートの例を示したのが図５である。特定のサブクラスは、ファンインおよび／またはファンアウトを許可しないことを選択することができる。例えば、オーディオ・ポートは両方を許可しないので、１つのポートを２つ以上の他のポートに結合しようとすると例外が引き起こされる。ファンインとファンアウトのプロパティ（特性）は特定のメディア・サブクラスによって処理される。例えば、MIDIでは、ファンアウトは同じデータを複数の受信側へ送信し、ファンインは複数の送信側からのデータをマージする。
２つのメディア・コンポーネントＡとＢは、次の方法によって１つに結合される。
1)　Ａのメンバ関数をコールして出力ポートのサロゲートPaを要求する。
2)　Ｂのメンバ関数をコールしてＢのメンバ関数に入力ポートのサロゲートPbを要求するように求める。
3)　Paのメンバ関数ConnectToをコールし、Pbを引数として渡す。
ポートは、Paのメンバ関数DisconnectFromをコールし、Pbを引数として渡すことにより切り離すことができる。上記のプロシージャは、ポート・タイプがオーディオであるか、グラフィックであるか、MIDIであるか、他のマルチメディア・データ・タイプであるかに関係なく、いずれかのメディア・コンポーネントを結合または切り離すとき呼び出される。
２つのポートが結合されるときコンパイル時検査(compile-time checking)は行われない。例えば、ソフトウェア開発者がオーディオ・ポートをMIDIポートに結合することを試みるコードを書き、コンパイルし、リンクするとき、それを妨げるものは何もない。その代わりに、実行時(run time)に例外が引き起こされる。この作用(behavior)はメディア・ポートの多態的(polymorphic)結合を可能にするために、ルーチン・システムに組み込まれるように明示的に設計されている。例えば、パッチング・モジュール(patching module)は、異種のポート・タイプに特別な扱いをしなくても、特定の結合時にペアのメディア・ポートを１つに結合することを可能にする。オーディオ、ビデオおよびMIDIポートはすべて、パッチャ(patcher)によって全く同じ扱いを受けることになる。オブジェクトの結合は、ディスプレイ上のアイコンのような、マルチメディア・オブジェクトを表す記号(ind

icia)間で線分で引くことにより視覚的に表現することができる。

オーディオ・データのルーチング

オーディオはコンピュータ上でディジタル化し、ストアし、処理し、プレイバック（再生）することができる。また、コンピュータはオーディオをシンセサイズしてプレイバックするためにも使用できる。アナログ－ディジタル・コンバータ(Analog to Digital Converter-ADC)はアナログ・オーディオ電気信号を、ディジタル・オーディオ・サンプル(digital audio sample)と呼ばれる数字の列に変換するために使用される。オーディオは、電話品質のオーディオでは毎秒8000サンプルのレートで、コンパクトディスク(CD)品質のオーディオでは毎秒44,100サンプルのレートで、ディジタル・オーディオ・テープ(Digital Audio Tape-DAT)では毎秒48,000サンプルのレートでサンプリングされるのが代表的である。数字の形態に変換されたオーディオはストアして、コンピュータによって処理することができる。ディジタル・オーディオ・サンプルはディジタル－アナログ・コンバータ(Digital to Analog Converter-DAC)を使用してアナログ・オーディオ電気信号に逆変換される。好適実施例では、オーディオ・ポートと名づけたポート・オブジェクトのサブクラスが定義されている。これらは、ディジタル・オーディオ・データをメディア・コンポーネント間で転送（ルーチング）するために使用される。オーディオ出力ポートをオーディオ入力ポートに結合すると、オーディオ・データを転送することができる。あるメディア・コンポーネントが少なくとも１つのオーディオ・ポートをもっていれば、それはオーディオ・コンポーネントと呼ばれる。すべてのオーディオ・コンポーントはメディア・コンポーネント基底クラスのサブクラスである。オーディオ・コンポーネントは図６に示すように、ゼロ個または１個以上のオーディオ入力ポートとゼロ個または１個以上のオーディオ出力ポートをもっている。オーディオ・コンポーネントはそのポートを結合することにより、共に結合することができる。これは、ハードウェアを利用してステレオ・コンポーネントを１つに結合するためにパッチ・コード(patch cord)を使用するのと似ている。

好適実施例によれば、オーディオ・コンポーネントの結合が容易化されているので、様々な興味深いアプリケーションを作ることができる。図７，図８，図９および図10はオーディオ・コンポーネントを使用して作ることができるアプリケーション例のいくつかを示したものである。図７はオーディオ・プレーヤを使用すると、音声注釈(voice annotation)をレコーディング（録音）して、プレイ（再生）するための音声注釈アプリケーションをどのように書くことができるかを示している。電話ハンドセット（送受話器）は入出力のために使用されている。図８は音声メール／電話応答アプリケーションをどのように作ることができるかを示している。一方のオーディオ・プレーヤはあいさつをプレイして電話回線上に送出し、その間に他方のオーディオ・プレーヤは到来メッセージをレコーディングしている。図９はミュージック・アプリケーションを示している。特殊効果のエコーが楽器のサウンドに加えられ、スピーカからプレイされている。図10はミュージック用とサウンド効果用とナレータ音声(voice over)用の３つのサウンドがどのように１つにミックスされて、スピーカなどの出力デバイスから出力されるかを示している。

すべてのメディア・コンポーネントと同じように、オーディオ・コンポーネントはそれぞれのポートを結合することにより共に結合される。このオペレーションは、オーディオ・コンポーネント・ポートを選択し、線などの幾何学図形をそのコンポーネント・ポートから別のマルチメディア・コンポーネント・ポートまで延長し、ディスプレイ上に図形表現されたリンケージを収めておくデータ構造を作ると、簡単に行うことができる。すべてのオーディオ・コンポーネントはその入力ポートと出力ポートのサロゲートを作るメンバ関数をもっている。クライアントは、まず入力および／または出力ポート・サロゲートを各コンポーネントに要求し、次に、サロゲート・オブジェクトから提供されるメンバ関数を使用して実際の入力ポートを実際の出力ポートに結合することによって結合オペレーションを実行する。各オーディオ・ポートとポート・サロゲートはそれぞれに関連づけられたオーディオ・タイプをもっている。図11は好適実施例によってサポートされるオーディオ・タイプのいくつかを示している。

ディスプレイ上にリンキング線分で表されたコネクションは単一のオーディオ・チャネル

を表している。ステレオは左チャネル用と右チャネル用の２つのコネクションを使用して処理される。２つのポートを結合することが要求されると、タイプ・ネゴシエーション・プロトコルは、これらのポートに互換性データ・タイプをサポートする能力があるかどうかを確かめる。ポートが共通のタイプをもっていないと、例外が引き起こされる。このようなことが起こったとき、図12に示すようにオーディオ・タイプ・コンバータを２つのポート間に挿入しておくと、一方のタイプから他方のタイプへ変換することができる。オーディオ・コンポーネントのネットワークではループ（サイクル）が許されない。ループになるような結合を行おうとすると、例外が引き起こされる。オーディオ・コンポーネントを結合するプロセスは、マルチメディア・オブジェクトを表す記号(indicia)のオーディオ入力と出力ポートを表すアイコンに似た図形の間を、線分などの幾何学図形で結ぶことによりディスプレイ上に図形的に表現される。

オーディオ・コンポーネントの実現

クライアント－サーバ・ベースのアーキテクチャはオーディオ・コンポーネントを実現するために使用される。すべてのオーディオ・コンポーネントに対して、オーディオ・プロセッサ・オブジェクトがあり、これはサウンド・サーバと名づけたタスクに置かれている。オーディオ・プロセッサ・オブジェクトは実際の信号処理を行うことを受け持っている。オーディオ・コンポーネント・サブクラス側(subclasser)はオーディオ・コンポーネントとオーディオ・プロセッサ・サブクラスの両方を書く必要がある。クライアント／サーバ・アプローチはパフォーマンスを向上するために使用される。オーディオをパッチ・コード・スタイルでルーチングすることは、データ移動と信号処理をすべて単一のタスクで行うようにすれば、最も効率的に実現される。そうすれば、不必要なプロセス間通信(Interprocess Communication-IPC)とコンテクスト・スイッチ(context switch)を使用しないで済むことになる。クライアントが多数の異なるタスクに置かれているとすると、サウンド・サーバには処理を単一タスクで集中化することが要求されることになる。

これとは別に実現方法も開発されているが、この実現方法によれば、信号処理はオーディオ・コンポーネントごとに別々のタスクで扱われている。オーディオ処理にはサーバは不要である。これは多数の利点をもつエレガントな方法であるが、残念ながら欠点が１つある。それは、本発明の好適実施例で提案しているアプローチに比べて速度が１桁以上遅いことである。モノプロセッサでは、この比率は、処理能力が向上したマシンが作られたときでも、ほぼ一定のままになっている。

クライアント／サーバ・モデル

オーディオ・コンポーネントには、それぞれに対応するオーディオ・プロセッサと通信するための機能が必要である。リモート・プロシージャ・コール(Remote Procedure Call-RPC)インタフェースはこの目的のために使用されている。一般的に、オーディオ・コンポーネント・サブクラスのメンバ関数はオーディオ・プロセッサ上の「実」メンバ関数を遠隔的(remotely)に呼び出している。図13は好適実施例によるリモート・プロシージャ・コールを示すブロック図である。

結合方法

オーディオ・プロセッサは、オーディオ・コンポーネントとは異なり、オーディオ・ポートを所有している。オーディオ・コンポーネントはオーディオ・ポート・サロゲートを所有している。２つのポート・サロゲートがクライアント・アドレス空間で共に結合されると、対応するオーディオ・ポートがサウンド・サーバ・アドレス空間で共に結合される。サブクラス側はどのイネーブリング・アクションもとる必要がない。その代わりに、フレームワークがこれを行う。

オーディオの処理

オーディオ・ポート間のコネクションはオーディオ・データのバッファを使用して実現されている。各コネクション、つまり、「パッチ・コード」はそれと関連づけられたバッファをもっている。オーディオ・プロセッサはそのオーディオ・ポートの各々に対して、そのバッファのアドレスとサイズを要求することができる。図14はデータをその入力バッファから読み取り、そのデータを処理し、その結果をその出力バッファに書き出す関連のRu

n()メンバ関数をもつオーディオ・プロセッサ・アーキテクチャを示すブロック図である。バッファのサイズは可変であるが、好ましいサイズは５ミリ秒に相当するサンプル数を収容するようになっている。このサイズは、最大待ち時間を10msにしてサウンドを開始し、停止できるパフォーマンス目標を達成するために必要である。

フレーム・ベース処理

サウンド・サーバはフレーム・ベース処理(frame-based processing)と呼ばれる手法を使用してサウンド・サンプルを処理する。基本的プロセスは以下に説明するとおりである。

1) プロデューサーがコンシューマの前に来るようにオーディオ・プロセッサを順序付ける。

2) 各オーディオ・プロセッサごとに、そのRun()メンバ関数をコールする。このステップは入出力デバイスがデータを要求するつど繰り返される。

この方法によると、オーディオが非常に効率よく処理されるが、これは実行順序が決まれば、Run()をコールすることによってフレームを安価に作ることができるからである。Run()は単一フレームで完全に処理できる固定サイズのバッファを利用するので、そのインプリメンテーションを非常に効率よく行うことができる。オーディオ・プロセッサのネットワークを示している図15を例にして説明する。各オーディオ・プロセッサ、つまり、ノードは英字を付けて示されている。

サウンドを作るために、サウンド・サーバは以下のようにステップを実行する。

1) プロデューサーがコンシューマの前に来るようにオーディオ・プロセッサを順序付ける。そして、

2) スピーカなどのオーディオ入出力デバイスがデータを要求するたびにオーディオ・プロセッサを実行する。

図16は、図15に示すネットワークでこの処理がどのように行われるかを示している。最初のブロックでは、オーディオ・プロセッサが順序付けられ、そのあとプロセッサはその特定順序で実行される。これと並行して、プロセッサの各々ごとに情報が得られる。どのプロセッサも、その時間が来るまでは実行されない。言い換えれば、そのプロセッサで該当の情報が得られるまでは、そのプロセッサは待ち状態に置かれている。

オーディオ・プロセッサの順序付け

オーディオ・プロセッサはシミュレート・データ・フロー(simulated data flow)と呼ばれる手法を用いて順序付けがされる。順序付けは、多くの場合、フレームワークによって自動的に判断されるので、大部分のサブクラス側は順序付けについて気にする必要はない。シミュレート・データ・フローは、あたかもデータがあるプロセッサから次のプロセッサへ渡されるかのように、トポロジ的順序でネットワークを通過するように繰り返されていく。これが行われるのは、どのオーディオ・プロセッサが実行できるか、どのオーディオ・プロセッサが実行できないかを確かめるためである。実行されるオーディオ・プロセッサは実行リスト(run list)上に置かれ、実行フェーズの期間何度も利用される。

オーディオ・プロセッサ・ノードは次のとき実行可能になっている。

#1 その入力ポートにデータがあり、かつ

#2 その出力ポートにデータを置いておく場所がある。

各オーディオ・プロセッサはCanRun()メンバ関数をもっており、ノードが実行可能であればTRUEが返却される。CanRun()のデフォルト（省略時）では上記のルールを使用するので、サブクラス側がCanRun()をオーバライド(override)する必要があるのは、オーディオ・プロセッサがこのルールをオーバライドする必要がある場合だけである。あるオーディオ・プロセッサが実行可能であると、そのプロセッサはすべての出力ポートからデータが流れ出るようにシミュレートする。Fire()は、実際にはターンアラウンドして、各出力ポートに対して別のFire()メンバ関数をコールするだけである。出力ポートをファイア(fire)することは、データがポートから流れ出ることをシミュレートし、シミュレートされたデータは他方の側の入力ポート上に取り出すことができる。入力ポートはIsAvailable()メンバ関数をもっており、そのポートにシミュレートされたデータがあれば、TRUEが返却される。図17は好適実施例によるオーディオ・ポートを示している。

ディレイ(delay)への入力はサイレントになるので、ディレイはCanRun()に優先していなければならない。従って、ディレイは、ディレイが使用尽くされるまで出力を作り出していなければならない。入力ポートにデータが残っていなくても、オーディオ・プロセッサは実行されていなければならない。CanRun()は、入力が得られないが、出力する必要のあるデータがまだディレイに残っているときTRUEを返却するように修正しなければならない。プレーヤが停止されると、プレーヤはデータフロー基準に合致していても実行できないので、プレーヤもCanRun()をオーバライドしなければならない。プレーヤは、データフロー基準に合致しているかどうかに関係なく、プレーヤが停止されたときCanRun()が常にFALSEを返却するようにルールを変更しなければならない、オーディオ・プロセッサはそのオーディオ出力ポートに問い合わせて、データがポートのバッファに書かれたときからオーディオが実際にユーザに聞こえるようになるまでのディレイを確かめることができる。これにより、オーディオ・プロセッサは任意の時間基準と同期をとることが可能になる。

図18, 図19および図20はオーディオ・プロセッサに関連するロジックを示すフローチャートである。プレーヤなどのオーディオ・プロセッサは実行が要求されていると判断すると、図18に示すようにRequestOrderを呼び出す。

図18は、好適実施例に従って順序付け要求(order request)をコールするときの詳細ロジックを示すフローチャートである。処理は端子1800から開始され、即時に機能ブロック1810へ移って実行リストに無効(invalid)のマークを付ける。この実行リストは、マルチメディア・オブジェクトをドロップ・ラーンチ(drop launching)するか、マルチメディア・オブジェクトに対してダブルクリックするか、あるいはオペレーションの初期化を意味する他のアイコン操作を使用すると、無効のマークが付けられる。次に、判定ブロック1820でテストが行われ、オーディオ・プロセッサが実行可能かどうかが判断される。実行可能であれば、次に、判定ブロック1803で、オーディオ・プロセッサは非アクチベート(inactivate)され、コントロールが端子1850へ渡される。逆に、オーディオ・プロセッサが実行不能であれば、機能ブロック1840で、オーディオ・プロセッサはアクチベート(activate)され、コントロールが端子1850へ渡される。

図19は好適実施例に従ってオーディオ・プロセッサをアクチベートするときのリカーシブ・ロジック(recursive logic)を示すフローチャートである。処理は端子1900から開始され、そこでコントロールが機能ブロック1840から渡されるか、リカーシブ・コール(recursive call)が端子1990から出される。どちらの場合も、コントロールは即時に機能ブロック1910へ渡され、そこでインデックスｉがゼロに等しくされ、カウンタＮがオーディオ・プロセッサ出力ポートの数と等しい値にセットされる。そのあと、判定ブロック1920でテストが行われ、カウンタｉ＝Ｎかどうかが判断される。そうであれば、コントロールが端子1930で返却される。そうでなければ、出力ポートｉが機能ブロック1940でファイアされ、ファイアされたとのマークが機能ブロック1950で出力ポートに付けられ、出力ポートに結合されたオーディオ・プロセッサが機能ブロック1960で取得され、カウンタｉが機能ブロック1970でインクリメントされ、テストが判定ブロック1980で行われてオーディオ・プロセッサが実行可能かどうかが判断される。オーディオ・プロセッサが実行可能ならば、図19に示すロジックへのリカーシブ・コールが端子1900から出される。そうでなければ、コントロールが判定ブロック1920へ渡されて処理が続行される。

図20はオーディオ・プロセッサを非アクチベートするときの詳細ロジックを示すフローチャートである。処理は端子2000から開始され、そこでコントロールが機能ブロック1830から渡されるか、リカーシブ・コールが端子2018または2080から出される。どちらの場合も、コントロールは即時に機能ブロック2004へ渡され、そこでインデックスｉがゼロに等しくされ、カウンタＮがオーディオ・プロセッサ出力ポートの数に等しい値にセットされる。次に、テストが判定ブロック1920で行われ、カウンタｉ＝Ｎかどうかが判断される。そうであれば、コントロールが機能ブロック2020へ渡され、インデックスとカウンタが再初期化される。そのあと、判定ブロック2024でテストが行われ、ｉ＝Ｎかどうかが判断される。そうでなければ、入力ポートｉはファイアしなかったとのマークが機能ブロック2040で付けられ、入力ポートに結合されたオーディオ・プロセッサが機能ブロック2050で取得

され、カウンタが機能ブロック2060でインクリメントされ、テストが判定ブロック2070で行われてオーディオ・プロセッサが実行可能であるかどうかが判定される。そうであれば、コントロールが端子2024から返却される。オーディオ・プロセッサが実行不能ならば、図20に示すロジックへのリカーシブ・コールが端子2080から出力される。オーディオ・プロセッサが実行可能ならば、コントロールが判定ブロック2024へ渡されインデックスが再びテストされる。

インデックスｉが判定ブロック2006でＮに等しくなければ、出力ポートｉはファイアしていなかったとのマークが機能ブロック2008で付けられ、出力ポートに結合されたオーディオ・プロセッサが機能ブロック2010で取得され、カウンタｉが機能ブロック2012でインクリメントされ、テストが判定ブロック2014で行われてオーディオ・プロセッサが実行可能かどうかが判断される。オーディオ・プロセッサが実行不能ならば、図20に示すロジックへのリカーシブ・コールが端子2018から出される。そうでなければ、コントロールが判定ブロック2006へ渡されてインデックスが再テストされ、処理が続行される。

図21は好適実施例に従ってオーディオ・プロセッサを実行するときのロジックを示すフローチャートである。処理は端子2100から開始され、即時に判定ブロック2104へ移って実行リストが有効かどうかが判断される。実行リストが有効でなければ、トポロジカル・ソート(topological sort)が機能ブロック2106で行われ、オーディオ・プロセッサがファイア順にソートされる。次に、機能ブロック2110で、ソートされたリストに有効のマークが付けられ、コントロールが機能ブロック2120へ渡される。実行リストが判定ブロック2104で有効でなかったときは、コントロールは機能ブロック2120へ渡されて、インデックスｉがリセットされカウンタＮが初期化される。次に、テストが判定ブロック2122で行われ、カウンタｉがカウントＮまで達したかどうかが判断される。そうであれば、処理は完了し、コントロールは端子2130へ渡される。そうでなければ、オーディオ・プロセッサｉは機能ブロック2124に示すように実行され、インデックスｉは機能ブロック2126に示すようにインクリメントされる。

ビデオとグラフィカル・データのルーチング

ビデオ情報はコンピュータ上で、ディジタル化し、ストアし、処理し、プレイバック（再生）することができる。ビデオ・ディジタイザは、アナログ・ビデオ電気信号をフレームと呼ばれるディジタル・イメージの列に変換するために使用される。１秒当たりにディジタル化されるフレームの数はフレーム・レート(frame rate)と呼ばれる。１秒当たり15～30フレームが代表的なフレーム・レートである。ディジタル化されたビデオは格納して、コンピュータによって処理することができる。ビデオはディジタル・イメージをコンピュータ・スクリーンから元のフレーム・レートで順番に表示することによってプレイバックすることができる。

グラフィック・オブジェクトは、コンピュータ・スクリーンから表示できるオブジェクトを表現するために使用される基底クラスである。グラフィック・オブジェクトのサブクラスとしては、多角形、曲線、ディジタル・イメージなどがあるが、これらに限定されない。ディジタル化ビデオの各フレームはディジタル・イメージであるので、グラフィック・オブジェクトによって表現することができる。

グラフィック入力および出力ポートはグラフィック・オブジェクトを一方のメディア・コンポーネントから他方のメディア・コンポーネントへ転送するために使用される。ディジタル・ビデオがこのように転送できるのは、各ビデオ・フレームがグラフィック・オブジェクトであるからである。グラフィック・ポートはどのグラフィック・オブジェクトでも転送できるので、アニメーション・データも、グラフィック・ポートを使用して転送することが可能である。グラフィック・ポートを収めているメディア・コンポーネントを使用すると、ビデオ・オブジェクトのネットワークを構築してビデオ・オブジェクト間でビデオをストリーム化することが可能である。このようにすると、種々の興味のあるアプリケーションを作ることができる。図22は、好適実施例に従ってビデオ・ディジタイザ・コンポーネントをビュワー・コンポーネントにパッチングして、コンピュータのスクリーンから表示する例を示している。

図23は、好適実施例に従って２つのビデオ・オブジェクトからのイメージを効果プロセッサでミキシングして、その結果をコンピュータ・スクリーンから表示する、もっと複雑な例を示したものである。グラフィック・ポートはポート・サロゲートによって１つに結合される。すべてのビデオ・コンポーネントはその入力ポートと出力ポートのサロゲートを作るメンバ関数をもっている。クライアントは入力および／出力ポート・サロゲートを各コンポーネントに要求することによって結合オペレーションを実行する。そのあと、サロゲート・オブジェクトから提供されるメンバ関数を使用して、実際の入力ポートを実際の出力ポートに結合する。各グラフィック・ポートとポート・サロゲートはそれぞれに関連づけられたグラフィック・タイプをもっている。２つのポートが結合するように要求されると、タイプ・ネゴシエーション・プロトコルは、これらのポートに互換性データ・タイプをサポートする能力があるかどうかを確かめる。ポートが共通のタイプをもっていなければ、例外が引き起こされる。

ビデオ・コンポーネントを結合するプロセスは、ビデオ入力と出力ポートを表す記号(indicia)間を線分などの幾何学図形で結ぶことにより、ディスプレイ上で図形表現することができる。

データのルーチング

グラフィック・ポートはグラフィック・オブジェクトを指すポインタを読み書きすることによってグラフィック・データを操作する。書くときのポート・インプリメンテーションは同期式になっている。グラフィック出力ポートのWrite()メンバ関数は、送られてきたグラフィック・オブジェクトの処理を、受信側が完了するまでブロックされる。グラフィック・ポートはパフォーマンス上の理由からコピー・セマンティックス(copy semantics)を採用していない。ＲＧＢビットマップ・イメージをコピーしないようにしたのは、そのようなオペレーションを行うために含まれるプロセッサとストレージ量のためである。その代わりに、グラフィック・オブジェクトを指すポインタがグラフィック出力ポートからグラフィック入力ポートへ送られる。同期式インタフェースは、読み書きされるグラフィック・オブジェクトを格納するために共有メモリが使用されるときや、あるいはグラフィック・オブジェクトがタスクのアドレス空間にまたがってコピーされるとき、複数のアドレス空間にまたがって機能する。ブロックされたグラフィック・ポートは、コネクションが切り離されるイベントでアンブロックされる。

図24はグラフィック・ポートが好適実施例に従ってどのように使用されるかを示している。1)　ソース・メディア・コンポーネント中のタスクはグラフィック・オブジェクトをメディア・コンポーネントの出力ポートに書き出す。2)　グラフィック・オブジェクトを指すポインタはデスティネーション・メディア・コンポーネントの結合された入力ポートへ転送される。3)　デスティネーション・メディア・コンポーネント中のタスクはその入力ポートのRead()メンバ関数をコールしてブロックしたあとで、アンブロックしてグラフィック・オブジェクト・ポインタを読み取っている。4)　このタスクはグラフィック・オブジェクトの処理を終えると、デスティネーション・メディア・コンポーネントの入力ポートのNext()メンバ関数をコールする。5)　ソース・メディア・コンポーネントのタスクはアンブロックして、Writeコールからリターンする。デスティネーションはすでにグラフィック・オブジェクトの処理を終えているので、そのグラフィック・オブジェクトはタスクによって安全に処分される。図25は出力ポートのWriteメンバ関数に関連するロジックを示すフローチャートである。グラフィック・オブジェクトを指すポインタはこのメンバ関数に渡される。処理は端子2500から開始され、即時に判定ブロック2510へ移ってポートが結合されているかどうかが判断される。ポートが結合されていなければ、端子2520に示すように例外が引き起こされる。ポートが結合されていれば、テストが判定ブロック2530で行われその結合ポートが同じアドレス空間内にあるかどうかが判断される。ポートが同じアドレス空間内になければ、機能ブロック2540で、グラフィック・オブジェクト全体が共有メモリにコピーされる。ポートが同じアドレス空間内にあれば、グラフィック・オブジェクトを指すポインタがメモリにコピーされる。どちらの場合も、次のステップでは、機能ブロック2560に示すように通知が入力ポートへ送られ、機能ブロック2570に示すよう

にポートの処理が完了したとの通知が入力ポートからあるまでタスクがブロックされ、端子2580で終了する。
図26は好適実施例による入力ポートのRead（読取り）処理を示している。処理は端子2600から開始され、即時に判定ブロック2610へ移ってポートが結合されているかどうかが判断される。ポートが結合されていなければ、フレームワークは端子2620で例外を引き起こす。ポートが結合されていれば、別のテストが判定ブロック2630で行われ、グラフィックがレディ（準備状態）にあるかどうかが判断される。そうでなければ、機能ブロック2640に示すようにオブジェクトがレディになるまでブロック・タスク(block task)が処理され、コントロールが機能ブロック2650へ渡される。グラフィックがレディにあれば、グラフィック・オブジェクトを指すポインタが機能ブロック2650で返され、コントロールが端子2660から返される。
図27は好適実施例による入力ポートのNextメンバ処理を示している。処理は端子2700から開始され、即時に判定ブロック2710へ移ってポートが結合されているかどうかが判断される。ポートが結合されていなければ、端子2720に示すように例外が引き起こされる。ポートが結合されていれば、機能ブロック2730に示すように該当の通知が送られ、別のテストが判定ブロック2740で行われて、結合ポートが同じアドレス空間内にあるかどうかが判断される。ポートが同じアドレス空間内にあれば、処理は端子2760で完了する。ポートが同じアドレス空間内になければ、グラフィック・オブジェクトのコピーが機能ブロック2750に示すように削除され、処理は端子2760で終了する。
MIDIデータのルーチング
楽器ディジタル・インタフェース(Musical Instrument Digital Interface-MIDI)は電子楽器、コンピュータ、シーケンサ、照明コントローラ、ミキサ、およびテープ・レコーダ間で情報をやりとりするためのインタフェースを定義している。これについては、MIDI 1.0 Detailed Specification(1990)というタイトルのMIDI Manufacturers Association出版物に説明されている。MIDIはレコーディング・スタジオとライブ・パフォーマンスの両方で幅広く使用されており、スタジオのレコーディングと自動化コントロール、オーディオ・ビデオ・プロダクション、およびコンポジションの分野で大きな影響を与えている。MIDIは単独でも、他のメディアと併用されるときも、コンピュータをマルチメディア・アプリケーションに応用する上で統合的な役割を果たしている。ディジタル・オーディオと比べると、MIDIファイルが占めるスペースははるかに少なく、情報は操作と表示に都合がよいように記号化されている。例えば、代表的な３分のMIDIファイルでは、ディスク上に30～60キロバイトが必要になるのに対し、CD品質のステレオ・オーディオ・ファイルでは毎秒約200キロバイト、つまり、３分間では36メガバイトが必要になる。MIDIデータは記譜法(musical notaion)としても、グラフィカル・ピアノロール(graphical piano roll)としても、編集や異なる楽器への再割当てに適したメッセージ・リストとしても表示することが可能である。一般的なMIDIでは、楽器割当てが標準化されているので、マルチメディア・タイトル・プロデューサーに大きな刺激を与えている。
MIDI入力ポートと出力ポートは、タイムスタンプ付きMIDIパケットを一方のメディア・コンポーネントから他方のメディア・コンポーネントへ転送するために使用される。MIDIポートは、MIDIパケットをアドレス空間にまたがって伝達するときのメールボックスの働きをする。多くの興味のあるMIDIアプリケーションはMIDIポートを収めているメディア・コンポーネントを結合することによって作成することができる。図28は、MIDIプレーヤ2800とMIDIインタフェース2810の２つのコンポーネントをどのように使用すると、コンピュータに結合されたミュージック・シンセサイザをプレイ（演奏）できるかを示している。MIDIインタフェースはミュージック・シンセサイザなどの外部デバイスを接続するために使用される。MIDIパケットはMIDIプレーヤからMIDIインタフェースへ送られる。MIDIインタフェース2810はMIDIパケットをMIDIデータに変換し、これはミュージック・シンセサイザへ送られてプレイバックされる。
図29はMIDIデータがどのようにレコーディングされ、外部ミュージック・シンセサイザからプレイバックされるかを示している。MIDIインタフェース2910はMIDI出力ポートをもち

、ミュージック・シンセサイザから送られてきたデータに基づいてMIDIパケットを出力する。MIDIプレーヤ2900はMIDI入力ポートをもっている。入力ポートはこれらのパケットを読み取って、コンピュータ上に格納する。図30はMIDIデータがどのようにプレイバック(3000)され、フィルタ(3010)にかけられて、外部ミュージック・シンセサイザ(3020)へ送られるかを示している。フィルタ3010はその入力3000についてオペレーションを行い、その結果をその出力へ受け渡すことができる。例えば、MIDIエコー、ディレイを作るために余分の音符(note)を追加したり、バンド幅負荷を軽減するためにピッチ・バンド(pitch band)を取り除いたりするように特定のフィルタを書くことができる。

図31はMIDIポートとオーディオ・ポートの両方を収めているメディア・コンポーネントを示している。ソフトウェア・ベースのミュージック・シンセサイザはその入力ポートからMIDIパケットを読み取り、入力ポートから読み取った音符を表しているディジタル・オーディオを出力する。MIDIポートはポート・サロゲートによって共に結合されている。すべてのMIDIコンポーネントはその入力ポートと出力ポートのサロゲートを作るメンバ関数をもっている。クライアントはまず入力および／または出力ポート・サロゲートを各コンポーネントに要求し、次にサロゲート・オブジェクトから提供されるメンバ関数を使用して実際の入力ポートを実際の出力ポートに結合することによって結合操作を実行する。各MIDIポートとポート・サロゲートはそれぞれに関連づけられたMIDIタイプをもっている。２つのポートを結合することが要求されると、タイプ・ネゴシエーション・プロトコルは、これらのポートに互換性データ・タイプをサポートする能力があるかを確かめる。ポートが共通のタイプをもっていないと、例外が引き起こされる。

MIDIコンポーネントを結合するプロセスは、MIDI入力ポートと出力ポートを表す記号(indicia)間を線分などの幾何学図形で結ぶことにより図形表現することができる。

MIDIパケット

MIDIをサポートするデバイスは、MIDIメッセージを送受することにより相互に通信する。MIDI標準は２種類のメッセージを定義している。チャネル・メッセージとシステム・メッセージである。チャネル・メッセージはさらに音声(voice)メッセージとモード・メッセージに分かれている。システム・メッセージは、図32に示すようにさらに共通(common)メッセージ、リアルタイム(realtime)メッセージおよび排他(exclusive)メッセージに分かれている。チャネル音声メッセージはMIDIデバイスが聞き取ることができるチャネル番号(0-15)を収容している。チャネル・モード・メッセージは基本チャネル上を送出されて、チャネル音声メッセージに対する楽器の応答を決定する。システム共通メッセージはすべての受信器へ送られ、システム・リアルタイム・メッセージはクロック・ベースの楽器へ同期化情報を搬送する。システム排他メッセージを使用すると、メーカは標準で規定されたサポートを越えるMIDIサポートを提供することができる。すべてのメッセージはステータス（状況）バイトから始まっている。ただし、同じタイプのメッセージが連続するときは、ステータス・バイト（ランニング状況）を除くことができる（除くかどうかは任意である）。システム排他メッセージを除くすべてのメッセージ・タイプは、０個、１個または２個のデータ・バイトをもっている。システム排他メッセージは任意数のデータ・バイトからなり、最後はEOXバイトで終わっている。図33は好適実施例によるメッセージのフォーマット（形式）を示している。

MIDIパケットは標準をカプセル化する

MIDIパケット・オブジェクトはすべてのMIDIメッセージ・タイプと構造をカプセル化している。さらに、すべてのMIDIパケット・オブジェクトは図３４に示すようにステータス・バイトとタイムスタンプをもっている。MIDIパケットのサブクラスは便利なコンストラクタとアクセス・メンバ関数を使用してメッセージ・タイプを定義することによってMIDIプロトコルを反映している。

MIDIポート

MIDIポートはメディア・コンポーネント間でMIDIパケットをやりとりするために使用される。MIDI出力ポートはMIDIパケットを書き出すことができ、MIDI入力ポートはMIDIパケットを読み取ることができる。出力ポートはサロゲート・ポートによって入力ポートに結合

することができる。サロゲート・ポートはMIDIパケットを読むことも、書くこともできない。その代わりに、これらは受動的オブジェクト(passive object)であるので、他のアドレス空間に渡すことによってコネクション（結合）をサポートすることができる。メンバ関数は一度に１つまたは多数のメッセージを出力ポートに書くために用意されたものである。同様に、入力ポートから読み取るとき、次のメッセージを要求することも、バッファに入っているすべてのメッセージのカウントを要求することもできる。Readはバッファが空でなくなる(non-empty)までブロックされ、ブロックされたReadコールは別のタスクによってキャンセルすることができる。出力ポートに書かれたパケットのコピーは入力ポートから読み取られる。

パケットは到着順に読み取られるのでタイムスタンプ別にはソートされない。２つのストリームを時間順にマージする必要があるときは、ソーティング・マージ(sorting merge)オブジェクトが必要になる。入力ポートがなぜソートをしないのかを理解するには、まず時間が前後の両方向に流れることを思い起こせばよい。そこで、イベントが６、８および10秒のシーケンスで起こる、ある特定のプレイバック・シナリオについて検討することにする。時間が５から始まり、11まで進み、ターンアラウンドして再び５に戻るとすると、イベントの順序は６、８、10、10、８、６になるはずであり、６、６、８、８、10、10にはならない。パケットをバッファリングするのにソーティングが十分でないのは、まさに時間が両方向に流れるためである。

コネクションのファンインとファンアウトがサポートされている。ファンニング(fanning)とは、MIDI出力ポートを２つ以上のMIDI入力ポートに結合できる能力のことであるので、MIDI入力ポートはそこに結合された２つ以上のMIDI出力ポートをもつことができる。図35は好適実施例によるファンニング・オペレーションの例を示している。

MIDIポートの実現

好適実施例では、結合ポートのリストを利用してMIDIポートのコネクションを記録し、共有バッファを利用してパケットの受渡しを行っている。各ポートはそこに結合された他のすべてのポートのリストを維持している。このリストはConnect（結合）コールとDisconnect（切離し）コールによって更新され、ポートが壊されたときに使用されて、ポートが消滅したことを、そこに結合されているすべてのポートに通知する。この処理によると、実際に入力ポートが壊されたときは、出力ポートはその結合リストにその入力ポートを持つことができないので、そこに書くことができなくなる。そのために、出力ポートは入力ポートのリストを維持しており、それを使用してWriteコールを実現し、リスト内の各ポートに書くようにしている。また、入力ポートはそこに結合されている出力ポートのリストを維持しており、入力ポートが消滅したとき、出力ポートに通知しなければならない。

共有バッファはプロデューサーコンシューマ・プロトコル(producer-consumer protocol)をサポートしている。バッファが空になったとき、タスクはバッファが空でなくなるまでブロックすることができる。そのあとでそのバッファに置く(deposit)別のタスクは、バッファに書いたことをブロックされたタスクに通知することができる。別の方法として、さらに別のタスクがブロックをキャンセルして、別の種類の通知を行うことができる。

MIDI出力ポートのWrite()メンバ関数は図36に示されている。処理は端子3600から開始され、即時に機能ブロック3610へ移ってカウンタと、ループの制限値とを初期化する。次に、判定ブロック3620でテストが行われ、カウンタが制限値まで達したかどうかが判断される。カウンタが制限値まで達していれば、処理は端子3670で完了している。カウンタが達していなければ、機能ブロック3630でパケットのコピーがポートのバッファに挿入され、そして、テストが判定ブロック3640で行われ、バッファにメッセージがあるかどうかが判断される。バッファが空でなければ、該当のメッセージが機能ブロック3650に示すように送信される。そうでなければ、メッセージは送信されない。どちらの場合も、カウンタは機能ブロック3660でインクリメントされ、コントロールが端子3670から返される。

MIDI入力ポートのRead()メンバ関数は図37に示されている。処理は判定ブロック3700から開始され、バッファが空であるかどうかが判断される。そうであれば、タスクはバッファに情報が入るまで、あるいは機能ブロック3710でキャンセルが行われるまでブロックされ

る。テストが判定ブロック3730で行われ、待ちがキャンセルされたかどうかが判断される。そうであれば、例外が端子3750で引き起こされる。そうでなければ、あるいはバッファが判定ブロック3700で空になっていなければ、パケットが機能ブロック3720でバッファからコピーされて呼出し側へ送られ、端子3740からリターンして処理を完了する。

抽象マルチメディア・コンポーネント

メディア・コンポーネント

時間基準メディア・コンポーネントはメディア・コンポーネント基底クラスと呼ばれ、ルーチングを行うとき使用される中心の抽象クラスである。メディア・コンポーネントはゼロ個または１個以上の入力ポートとゼロ個または１個以上の出力ポートをもっている。図38に示す例では、メディア・コンポーネントは１つの入力ポートと２つの出力ポートをもっている。

メディア・シーケンス

メディア・シーケンス(mediasequence)はオーディオ・シーケンスを含めて、メディア内容を表している抽象基底クラスである。メディア・シーケンスのサブクラスはオーディオ、ビデオ、およびMIDIのクリップを表すために使用される。メディア・シーケンスの特徴は持続時間(duration)とタイプのリストをもっていることである。持続時間は浮動小数点値で表され、データがどれだけの長さであるかを示している。データもタイプを有しており。どのタイプのサウンドがデータによって表されているか（例えば、ビデオ、オーディオなど）を示している。サブクラスは複数のタイプをサポートすることが可能である。例えば、オーディオ・サブクラスはリニア形式でも、圧縮形式でもデータを与えることができる。これが可能であるために、メディア・シーケンスはタイプのリストをもっている。

プレーヤ

図39は時間基準メディア・プレーヤ（プレーヤ）基底クラスの例を示しており、これは時間基準メディア・シーケンス（メディア・シーケンスと呼ぶことにする）をプレイできるメディア・コンポーネントである。メディア・シーケンスはオーディオ、ビデオ、アニメーション、またはMIDIデータのクリップを表すために使用できる抽象基底クラスである。時間基準メディア・シーケンスのサブクラスはオーディオ、ビデオ、アニメーション、およびMIDIシーケンスを実現するために使用される。プレーヤはテープ・レコーダに似ており、メディア・シーケンスはカセット・テープに似ている。プレーヤはメディア・シーケンスをプレイするPlay()メンバ関数、シーケンスの中にレコーディングするRecord()メンバ関数、およびプレイバックまたはレコーディングを停止するStop()メンバ関数をもっている。プレーヤはシーケンス内の位置までシークするSeek()メンバ関数ももっており、ほかにも、プレーヤが他のプレーヤと、あるいはソフトウェア・クロックと同期をとることを可能にするメンバ関数も持っている。プレーヤをデータから分離すると、プレーヤを再使用することも可能である。あるカセットをプレイしたあと、プレーヤに別のカセットをプレイまたはレコーディングするように指示することができる。

標準オーディオ・コンポーネント

オーディオ・プレーヤ

図40はオーディオ・プレーヤを示しており、これはオーディオ・コンポーネントがオーディオ・データをプレイし、レコーディングするための関連の抽象基底クラスをもっている。オーディオ・データは、オーディオ・シーケンスと呼ばれ、メディア・シーケンスのサブクラスであるオブジェクトに格納される。オーディオ・プレーヤはテープ・レコーダに似ており、オーディオ・シーケンスはカセットに似ている。オーディオ・プレーヤはプレーヤのサブクラスである。すべてのプレーヤ（サウンド、ビデオ、またはMIDI）と同じように、オーディオ・プレーヤはサウンドを聞こえるようにするPlay()メンバ関数と、シーケンスをレコーティングできるようにする（ただし、書込みが許可されている場合）Record()メンバ関数と、レコーディングまたはプレイを停止するStop()とをもっている。また、サウンド内の個所をランダムにアクセスするSeek()メンバ関数をもっている。さらに、プレーヤが別のメディア・プレーヤまたはソフトウェア・クロックと同期を取ることを可能にするメンバ関数を持っている。

スピーカ
図41はスピーカ・コンポーネントの例を示している。スピーカ・コンポーネントはオーディオ出力デバイスの抽象基底クラスである。システムには、複数のサウンド出力デバイスが結合可能である。スピーカ・コンポーネントをサブクラス化すると、出力デバイスの種々の特徴を表すことができる。例えば、スピーカ・コンポーネントのサブクラスは、電話ハンドセット（送受話器）または電話回線を利用してプレイバックを可能にするものが存在している。スピーカ・コンポーネントは論理出力デバイスである。従って、スピーカ・コンポーネントの多数のインスタンスが存在することができるので、すべてが１つにミックスされて、コンピュータの物理スピーカからプレイアウトされることになる。ステレオ出力デバイスはスピーカ・コンポーネントの２つのサブクラス（左チャネル用と右チャネル用）で表されることになる。
マイクロホン
図42は好適実施例によるマイクロホン・コンポーネントを示している。マイクロホン・コンポーネントは、マイクロホンやライン入力などのサウンド入力ソースを表している抽象基底クラスである。システムには、複数のサウンド入力デバイスが結合可能である。マイクロホン・コンポーネントをサブクラス化するとこれらのデバイスを表すことができる。例えば、マイクロホン・コンポーネントのサブクラスを使用すると、電話ハンドセットや電話回線からのレコーディングが容易化される。スピーカ・コンポーネントと同様に、マイクロホン・コンポーネントは論理デバイスである。従って、マイクロホン・コンポーネントの多数のインスタンスが存在できる。すべてのインスタンスは同じ物理マイクロホンから入力された同一のオーディオを出力する。マイクロホン・コンポーネントは１つの出力をもっている。マイクロホン・コンポーネントはデータのソースであるので、Start()をコールして明示的に始動しなければならない。Stop()をコールすると停止できる。ステレオ入力デバイスはマイクロホン・コンポーネントの２つのサブクラス（左チャネル用と右チャネル用）によって表される。
ミキサ
図43はミキサ・コンポーネントの例を示している。ミキサ・コンポーネントは２つまたはそれ以上のオーディオ入力を総和して１つのオーディオ出力に送る抽象基底クラスである。入力の数はコンストラクタ(constructor)への入力によって決まる。
スプリッタ
図44はスプリッタ・コンポーネントの例を示している。スプリッタ・コンポーネントは１つの入力を受け取り、それを２つまたはそれ以上の出力に分割する抽象基底クラスである。出力の数は、コンストラクタになにが渡されたかによって決まる。
ゲイン
図45はゲイン・コンポーネントの例を示している。ゲイン・コンポーネントは信号の振幅（サイズ）を増減できる抽象基底クラスである。これはボリューム・コントロールのために使用され、SetGain()関数とGetGain()関数をもっている。数学的には、ゲイン・コンポーネントは各入力サウンド・サンプルにゲイン値を掛けて、その結果をその出力へ送る。ゲイン値が1.0のときは、信号はまったく変更されない。ゲインが2.0のときは、信号は音量の２倍になり、ゲイン値が.5のときは、信号は音量の半分になる。大きな信号レベルはクリップされる。
エコー
図46は好適実施例によるエコー・コンポーネントを示している。エコー・コンポーネントはエコーをその入力に付加し、その結果をその出力から発生する抽象基底クラスである。エコー・コンポーネントはユーザがセットできる３つのパラメータをもっている。それはディレイ長さ、フィードバック、およびミックス比である。ディレイ長さが大きいと、入力は大きな深い峡谷の中にいるような音を出し、ディレイが小さいと、ジェット機のヒュー音(whoosing)に似たフランジング効果(flanging effect)が得られる。フィードバックは聞こえるエコーの数を決めるものである。ミックス比はエコーされた信号がどれだけミックスされて戻されるかを決めるものである。

ファズ
図47は好適実施例によるファズ(fuzz)コンポーネントを示している。ファズ・コンポーネントは歪みをサウンドに加える抽象基底クラスである。この効果はギターや楽器のサウンドで最も有用である。
オーディオ・タイプ・コンバータ
図48はオーディオ・タイプ・コンバータの例を示している。オーディオ・タイプ・コンバータ・コンポーネントはあるオーディオ・タイプから別のオーディオ・タイプへ変換する抽象基底クラスである。
オーディオ・マルチコンバータ
図49は好適実施例によるオーディオ・マルチコンバータを示している。オーディオ・マルチコンバータ・コンポーネントは特定の選択に基づいて複数のオーディオ・タイプから複数の他のオーディオ・タイプへ変換するものである。
サウンド
図50は好適実施例によるサウンド・コンポーネントを示している。サウンド・コンポーネントはサウンドをレコーディングし、プレイするとき利用すると便利なオブジェクトである。図51はサウンド・コンポーネントに組み込まれたコンポーネントを示している。サウンド・コンポーネント5100はマイクロホン・コンポーネント5110、ゲイン・コンポーネント5120（入力レベルを制御するためのもの）、オーディオ・プレーヤ・コンポーネント5130、別のゲイン・コンポーネント5140（出力レベルを制御するもの）、およびスピーカ・コンポーネント5150を収めている。サウンド・ファイル名が与えられていれば、サウンド・コンポーネントは正しいオーディオ・シーケンスと必要なオーディオ・コンポーネントを自動的に作成して、そのファイルをプレイし、レコーディングする。
物理スピーカ
図52は好適実施例による物理スピーカ・コンポーネントを示している。物理スピーカ・コンポーネントはコンピュータの出力増幅器とスピーカのサロゲートである。これはコンピュータ全体のボリュームを制御するために使用される。コンピュータの他のサウンド出力デバイスがあれば、物理スピーカの対応するサブクラスが存在することになる。物理スピーカ・コンポーネントは、総ボリューム・レベルをセットし、得るためのSetVolume()とGetVolume()メンバ関数をもっている。
物理マイクロホン
図53は好適実施例による物理マイクロホン・コンポーネントを示している。物理マイクロホン・コンポーネントはマイクロホンまたはライン入力の実際のサードウェアを表している。コンピュータに入力される総入力レベルをセットすることができる。
標準ビデオ・コンピュータ
グラフィック・プレーヤ
図54は好適実施例によるグラフィック・プレーヤ・コンポーネントを示している。グラフィック・プレーヤ・コンポーネントは、時間により変化するグラフィック・オブジェクトである時間グラフィック・オブジェクトをプレイし、レコーディングすることができるコンポーネントの抽象基底クラスである。グラフィック・シーケンスは時間グラフィックのサブクラスである。グラフィック・シーケンスはグラフィック・オブジェクトが、順序付けられたシーケンスであり、そこでは各グラフィック・オブジェクトは持続時間を持っている。グラフィック・プレーヤはビデオ・テープ・レコーダに似ており、グラフィック・シーケンスはビデオ・カセットに似ている。グラフィック・プレーヤはプレーヤのサブクラスである。すべてのプレーヤ（サウンド、ビデオまたはMIDI）と同様に、グラフィック・プレーヤはグラフィック・オブジェクトが見えるようにするPlay()メンバ関数と、シーケンスをレコーディングできるようにするRecord()メンバ関数（ただし、書込みが許される場合）と、レコーディングまたはプレイを停止するStop()とをもっている。シーケンス内の個所をランダムにアクセスするためのSeek()メンバ関数ももっている。プレーヤが別のメディア・プレーヤまたはソフトウェア・クロックと同期をとることを可能にするメンバ関数をもっている。

グラフィック・ビュワー
図55は好適実施例によるグラフィック・ビュワー(graphic viewer)コンポーネントを示している。グラフィック・ビュワー・コンポーネントはグラフィック・オブジェクトをコンピュータ・ディスプレイから表示するために使用される。プレイされているグラフィック・オブジェクトを見るためには、グラフィック・プレーヤをグラフィック・ビュワーに結合しなければならない。コンシューマ・エレクトロニック・ビデオにたとえると、グラフィック・プレーヤはビデオ・テープ・レコーダに似ており、グラフィック・ビュワーはビデオ・ディスプレイ・モニタに似ている。
ビデオ・ディジタイザ
図56は好適実施例によるビデオ・ディジタイザ・コンポーネントを示している。ビデオ・ディジタイザ・コンポーネントはコンピュータに接続されたハードウェア・ビデオ・ディジタイザによってディジタル化されたアナログ・ビデオをグラフィック・オブジェクトに変換するものである。ビデオ・ディジタイザをグラフィック・プレーヤに結合すると、コンピュータに入ってくるアナログ・ビデオをレコーディングすることができる。コンシューマ・エレクトロニック・ビデオにたとえると、ビデオ・ディジタイザはビデオ・カメラに似ており、グラフィック・プレーヤはビデオ・テープ・レコードに似ている。
標準MIDIコンポーネント
MIDIプレーヤ
図57は好適実施例によるMIDIプレーヤ・コンポーネントを示している。MIDIプレーヤ・コンポーネントはMIDIシーケンスをプレイし、レコーディングすることができるコンポーネントの抽象基底クラスである。MIDIシーケンスはMIDIトラックが集まったものである。MIDIトラックはMIDIパケットが順序付けられたシーケンスである。MIDIプレーヤはプレーヤのサブクラスである。すべてのプレーヤ（サウンド、ビデオ、またはMIDI）と同様に、MIDIプレーヤはMIDIパケットを外部ミュージック・シンセサイザへ送信できるようにするPlay()メンバ関数と、外部キーボードからのMIDIメッセージをレコーディングできるようにする（ただし、書込みが許される場合）Record()メンバ関数と、レコーディングまたはプレイを停止するStop()とをもっている。シーケンス内の個所をランダムにアクセスするSeek()メンバ関数をもっている。また、プレーヤが別のメディア・プレーヤまたはソフトウェア・クロックと同期をとることを可能にするメンバ関数をもっている。
MIDIインタフェース
図58は好適実施例によるMIDIインタフェース・コンポーネントを示している。MIDIインタフェース・コンポーネントはMIDIパケットを外部ミュージック・シンセサイザとの間で送受信する。MIDIメッセージをプレイまたはレコーディングするためには、MIDIプレーヤがMIDIインタフェースに結合されていなければならない。
MIDIフィルタ
図59は好適実施例によるMIDIフィルタ・コンポーネントを示している。MIDIフィルタ・コンポーネントは１つのMIDI入力ポートと１つのMIDI出力ポートをもつオブジェクトの抽象基底クラスである。サブクラスは入力データを出力データに変換する変換アルゴリズムを提供する。
MIDIマッパ
図60は好適実施例によるMIDIマッパ・コンポーネントを示している。MIDIマッパ・コンポーネントはMIDIフィルタのサブクラスであり、ディクショナリ（辞書）を使用して入力MIDIパケットを出力MIDIパケットにマッピング（対応づけ）する。ディクショナリの各エントリはペアのMIDIパケット、つまり、入力パケット（キーと呼ばれる）と出力パケット（値と呼ばれる）になっている。MIDIマッパに送り込まれた入力MIDIパケットはディクショナリへのルックアップ・キーとして使用される。ルックアップの結果は出力MIDIパケットであり、これは出力ポートから書き出される。
MIDIプログラム・マッパ
図61は好適実施例によるMIDIプログラム・マッパ(program mapper)コンポーネントを示している。MIDIプログラム・マッパ・コンポーネントはMIDIマッパのサブクラスである。こ

れはMIDIプログラム変更メッセージを他のMIDIプログラム変更メッセージに変換する。ある楽器を他の楽器に対応づけるために使用することができる。

MIDI音符マッパ

図62は好適実施例によるMIDI音符マッパ(note mapper)コンポーネントを示している。MIDI音符マッパ・コンポーネントはMIDIマッパのサブクラスである。これはMIDI音符のオンとオフのメッセージを変換し、音符を置き換え（トランスポーズ）できるようにする。

MIDIチャネル・マッパ

図63は好適実施例によるMIDIチャネル・マッパ(channel mapper)コンポーネントを示している。MIDIチャネル・マッパ・コンポーネントはMIDIマッパのサブクラスである。これはMIDIチャネル音声とモード・メッセージを変換し、汎用目的のチャネル変更に使用できる。

【図１】

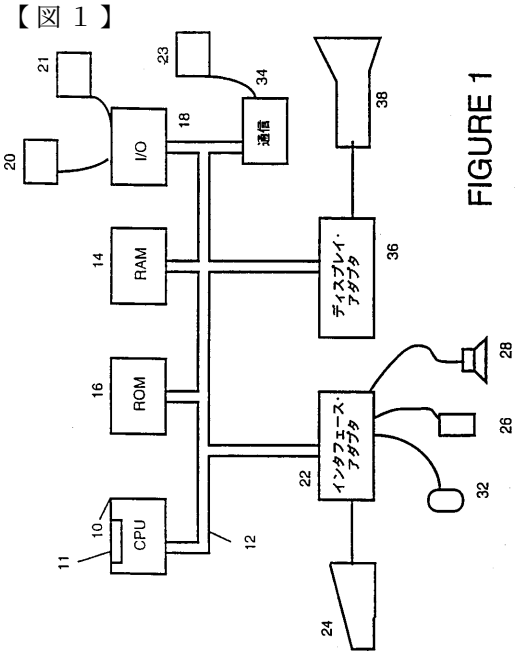


FIGURE 1

【図２】

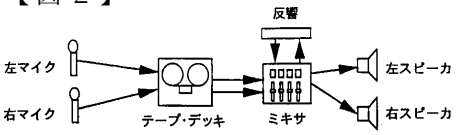


FIGURE 2

【図３】

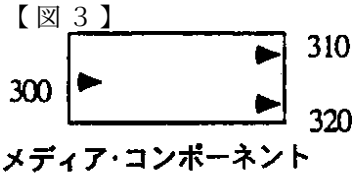


FIGURE 3

【図４】

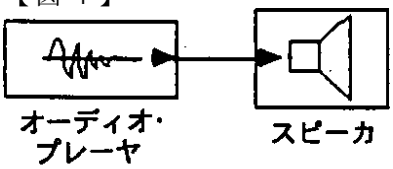


FIGURE 4

【図５】

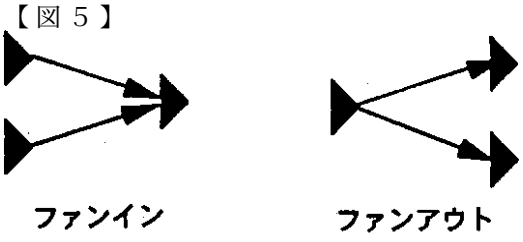


FIGURE 5

【図６】

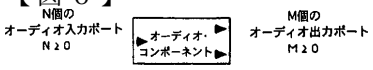


FIGURE 6

【図７】

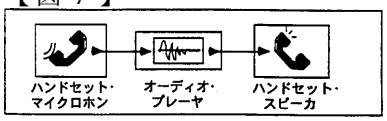


FIGURE 7

【図８】

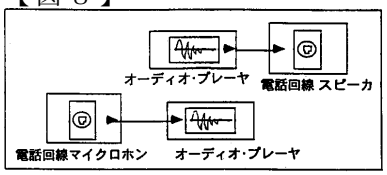


FIGURE 8

【図９】

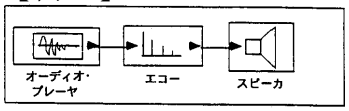


FIGURE 9

【図１０】

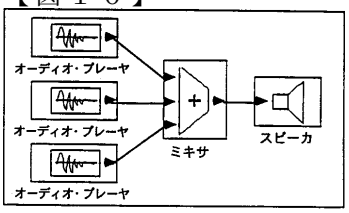


FIGURE 10

【図１１】

オーディオ・タイプ	サンプル・レート（サンプル/秒）	サンプル幅（ビット）	説明
k44KHz16BitLinear	44100	16	コンパクトディスク品質。2の補数整数サンプル
k22KHz8bitOffsetBinary	22050	8	媒体品質。オフセット・バイナリ（128はサイレンス）サンプル
k8KHzMuLaw	8000	8	北米電話標準。コード化サンプル
k8KHzALaw	8000	8	ヨーロッパ電話標準。コード化サンプル

FIGURE 11

【図１２】

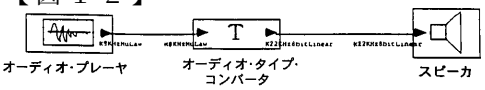


FIGURE 12

【図１３】

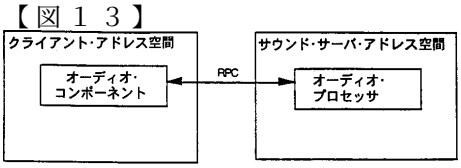


FIGURE 13

【図１４】

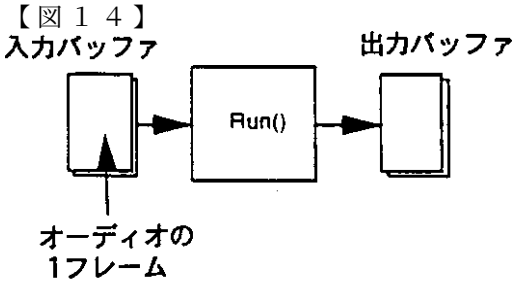


FIGURE 14

【図１５】

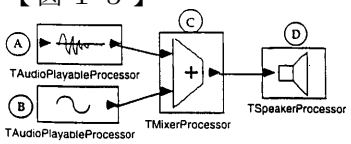


FIGURE 15

【図１６】

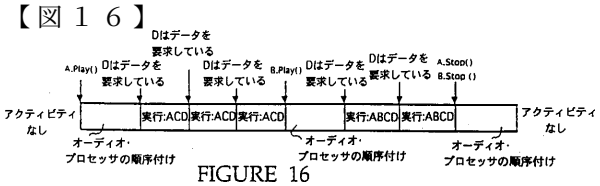


FIGURE 16

【図１７】



FIGURE 17

【図１８】

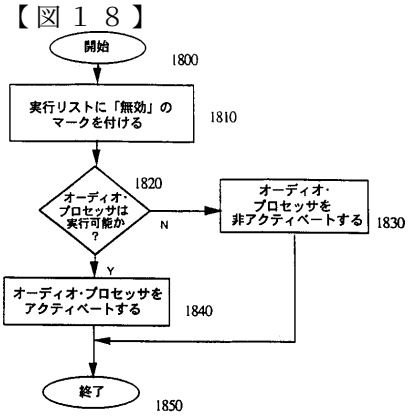


FIGURE 18

【図１９】

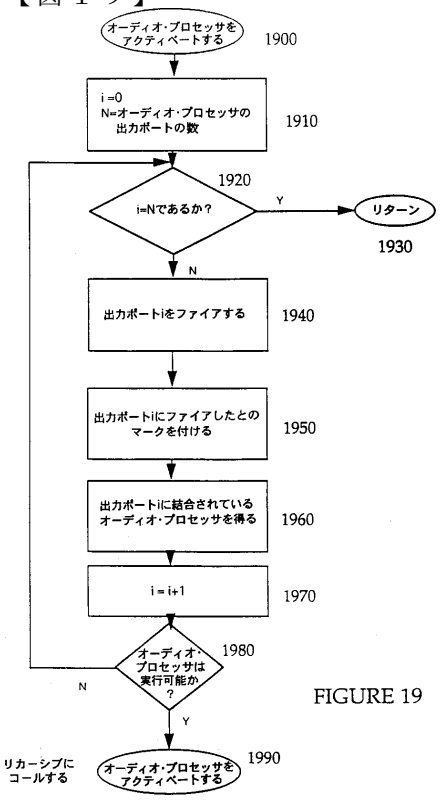


FIGURE 19

【図２０】

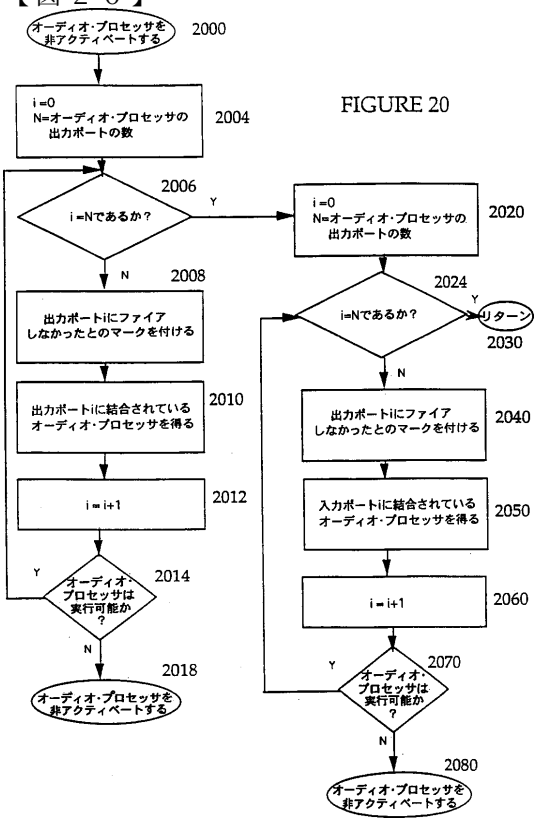


FIGURE 20

【図２１】

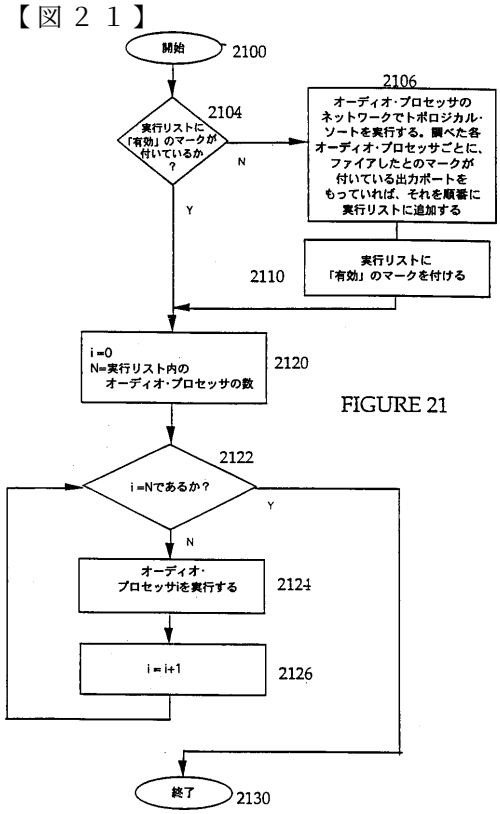


FIGURE 21

【図２２】

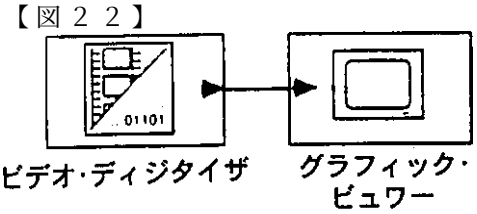


FIGURE 22

【図２３】

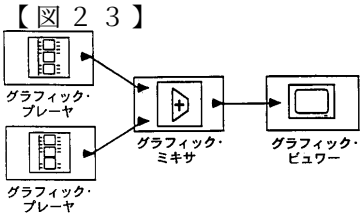


FIGURE 23

【図２４】

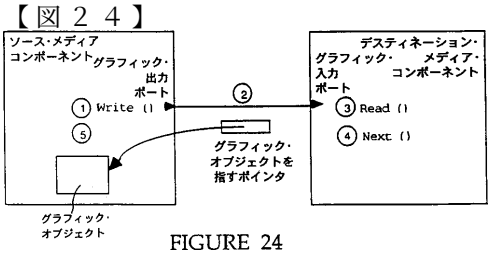


FIGURE 24

【図２５】

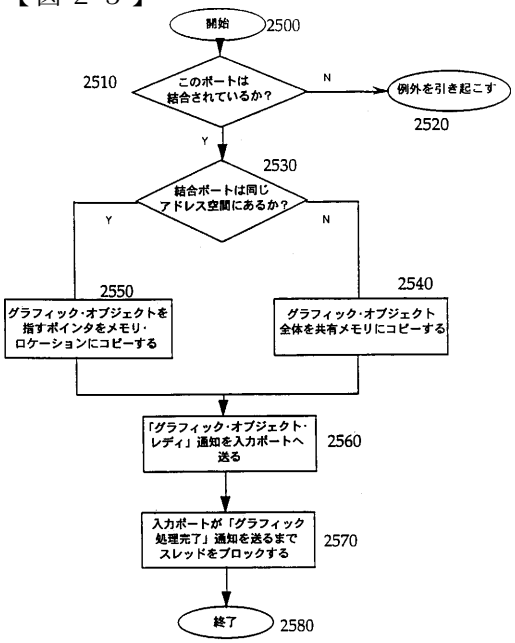


FIGURE 25

【図２６】

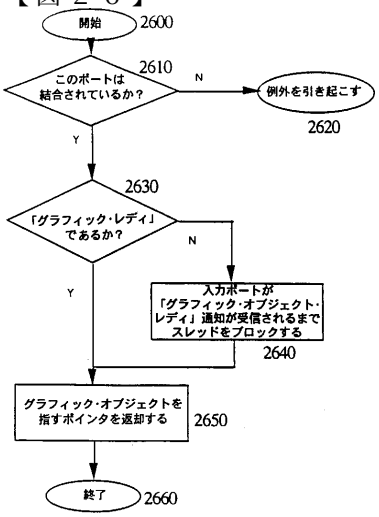


FIGURE 26

【図２７】

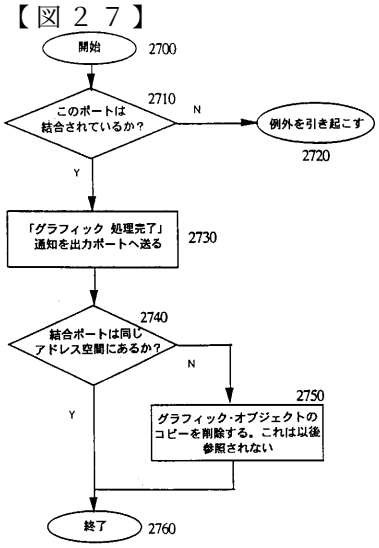


FIGURE 27

【図２８】

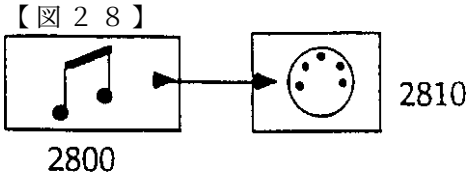


FIGURE 28

【図２９】

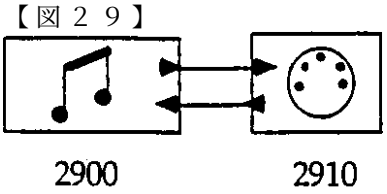


FIGURE 29

【図３０】

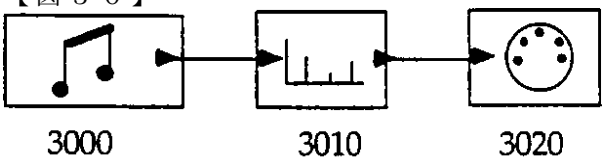


FIGURE 30

【図３１】

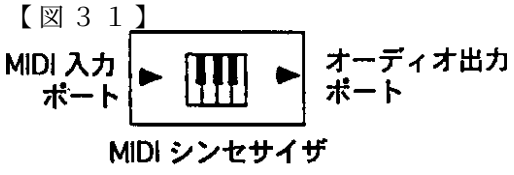


FIGURE 31

【図３２】

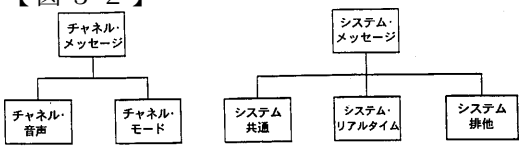


FIGURE 32

【図３３】

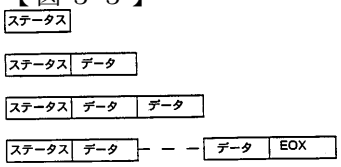


FIGURE 33

【図３４】

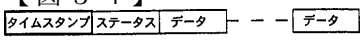


FIGURE 34

【図３５】

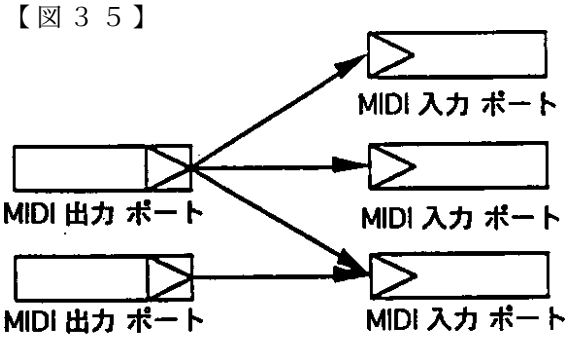


FIGURE 35

【図３６】

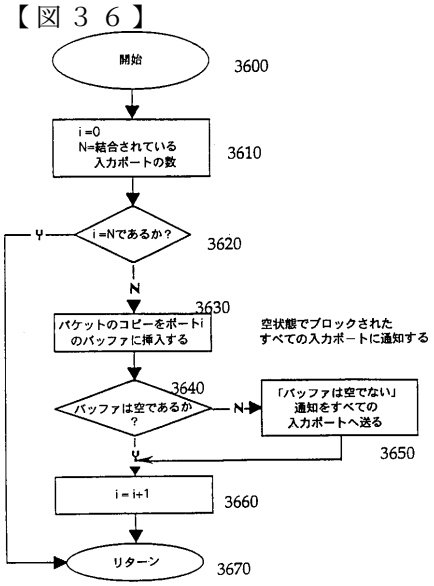


FIGURE 36

【図３７】

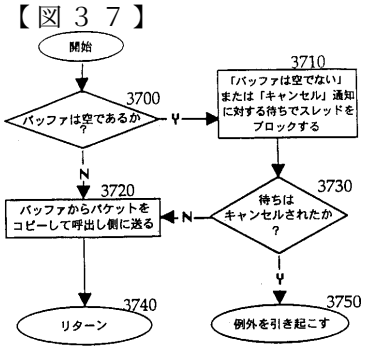


FIGURE 37

【図３８】

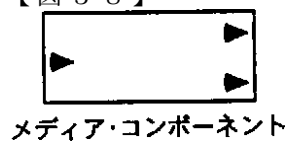


FIGURE 38

【図３９】

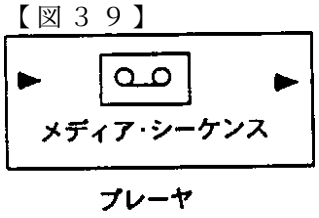


FIGURE 39

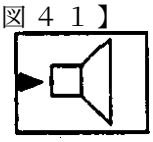
【図４０】



オーディオ･プレーヤ

FIGURE 40

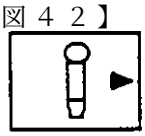
【図４１】



スピーカ

FIGURE 41

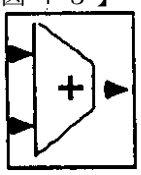
【図４２】



マイクロホン

FIGURE 42

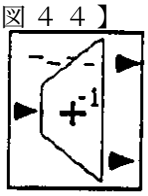
【図４３】



ミキサ

FIGURE 43

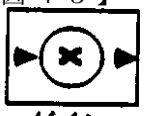
【図４４】



スプリッタ

FIGURE 44

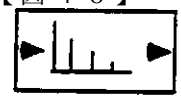
【図４５】



ゲイン

FIGURE 45

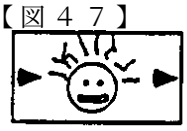
【図４６】



エコー

FIGURE 46

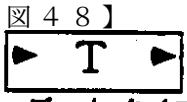
【図４７】



ファズ

FIGURE 47

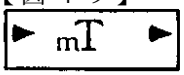
【図４８】



オーディオ･タイプ･
コンバータ

FIGURE 48

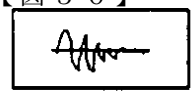
【図４９】



オーディオ･
マルチコンバータ

FIGURE 49

【図５０】



サウンド

FIGURE 50

【図５１】

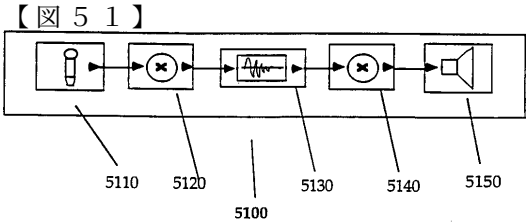
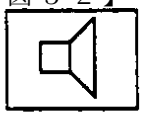


FIGURE 51

【図５２】



物理スピーカ

FIGURE 52

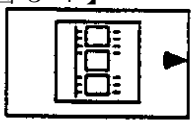
【図５３】



物理マイクロホン

FIGURE 53

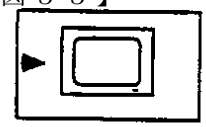
【図５４】



グラフィック・プレーヤ

FIGURE 54

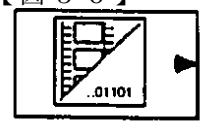
【図５５】



グラフィック・ビュワー

FIGURE 55

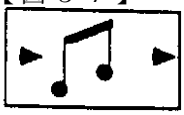
【図５６】



ビデオ・ディジタイザ

FIGURE 56

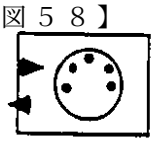
【図５７】



MIDI プレーヤ

FIGURE 57

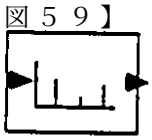
【図５８】



MIDI インタフェース

FIGURE 58

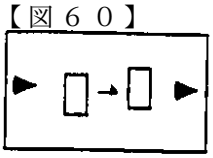
【図５９】



MIDI フィルタ

FIGURE 59

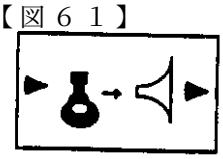
【図６０】



MIDI マッパ

FIGURE 60

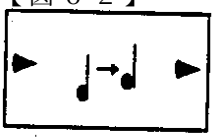
【図６１】



MIDI プログラム・マッパ

FIGURE 61

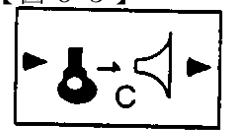
【図６２】



MIDI 音符マッパ

FIGURE 62

【図６３】



MIDI チャネル・マッパ

FIGURE 63

フロントページの続き

(72)発明者　ミルネ，スティーヴン，エイチ.
アメリカ合衆国　９４３０１　カリフォルニア州　パロ　アルト　ウエブスター　ストリート　１３７

審査官　殿川　雅也

(56)参考文献　特開平０３－２０５９８５（ＪＰ，Ａ）
特開平０５－２２４９５６（ＪＰ，Ａ）
特許第３２２８３４０（ＪＰ，Ｂ2）
田中聡他，分散環境に適したＣＨＩＬＬプロセス間通信機能，電子情報通信学会技術研究報告，社団法人電子情報通信学会，１９９０年　7月２７日，第９０巻，第１５９号，第7－１２頁，ＳＳＥ９０－５３

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 9/46 - 9/54
H04N 7/