(54) Title: OBJECT-ORIENTED AUDIO RECORD/PLAYBACK SYSTEM

(57) Abstract

A technique for providing routing of various multimedia events throughout the course of a multimedia presentation using a computer with a storage and a display. A processor with an attached display, storage and multimedia device builds a component object in the storage of the processor for managing the multimedia device including at least one port for exchanging multimedia information. The processor includes a connection object for connecting the at least one port to the multimedia device to facilitate the exchange of multimedia information and the processor routes information between the multimedia device and the component object port. A list of component objects are stored in the storage and current status for each of the components in the list is also stored. Then, when a multimedia player is invoked, a test is performed on each of the components in the list, and their associated multimedia devices, to determine what aspects of the multimedia presentation can be run.
FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>Austria</td>
<td>GB</td>
<td>United Kingdom</td>
<td>MR</td>
<td>Mauritania</td>
</tr>
<tr>
<td>AU</td>
<td>Australia</td>
<td>GE</td>
<td>Georgia</td>
<td>MW</td>
<td>Malawi</td>
</tr>
<tr>
<td>BB</td>
<td>Barbados</td>
<td>GN</td>
<td>Guinea</td>
<td>NE</td>
<td>Niger</td>
</tr>
<tr>
<td>BE</td>
<td>Belgium</td>
<td>GR</td>
<td>Greece</td>
<td>NL</td>
<td>Netherlands</td>
</tr>
<tr>
<td>BF</td>
<td>Burkina Faso</td>
<td>HU</td>
<td>Hungary</td>
<td>NO</td>
<td>Norway</td>
</tr>
<tr>
<td>BG</td>
<td>Bulgaria</td>
<td>IE</td>
<td>Ireland</td>
<td>NZ</td>
<td>New Zealand</td>
</tr>
<tr>
<td>BJ</td>
<td>Benin</td>
<td>IT</td>
<td>Italy</td>
<td>PL</td>
<td>Poland</td>
</tr>
<tr>
<td>BR</td>
<td>Brazil</td>
<td>JP</td>
<td>Japan</td>
<td>PT</td>
<td>Portugal</td>
</tr>
<tr>
<td>BY</td>
<td>Belarus</td>
<td>KE</td>
<td>Kenya</td>
<td>RO</td>
<td>Romania</td>
</tr>
<tr>
<td>CA</td>
<td>Canada</td>
<td>KG</td>
<td>Kyrgyzstan</td>
<td>RU</td>
<td>Russian Federation</td>
</tr>
<tr>
<td>CF</td>
<td>Central African Republic</td>
<td>KP</td>
<td>Democratic People’s Republic of Korea</td>
<td>SD</td>
<td>Sudan</td>
</tr>
<tr>
<td>CG</td>
<td>Congo</td>
<td>KR</td>
<td>Republic of Korea</td>
<td>SE</td>
<td>Sweden</td>
</tr>
<tr>
<td>CH</td>
<td>Switzerland</td>
<td>KZ</td>
<td>Kazakhstan</td>
<td>SI</td>
<td>Slovenia</td>
</tr>
<tr>
<td>CI</td>
<td>Côte d’Ivoire</td>
<td>LI</td>
<td>Liechtenstein</td>
<td>SK</td>
<td>Slovakia</td>
</tr>
<tr>
<td>CM</td>
<td>Cameroon</td>
<td>LK</td>
<td>Sri Lanka</td>
<td>SN</td>
<td>Senegal</td>
</tr>
<tr>
<td>CN</td>
<td>China</td>
<td>LU</td>
<td>Luxembourg</td>
<td>TD</td>
<td>Chad</td>
</tr>
<tr>
<td>CS</td>
<td>Czechoslovakia</td>
<td>LV</td>
<td>Latvia</td>
<td>TG</td>
<td>Togo</td>
</tr>
<tr>
<td>CZ</td>
<td>Czech Republic</td>
<td>MC</td>
<td>Monaco</td>
<td>TJ</td>
<td>Tajikistan</td>
</tr>
<tr>
<td>DE</td>
<td>Germany</td>
<td>MD</td>
<td>Republic of Moldova</td>
<td>TT</td>
<td>Trinidad and Tobago</td>
</tr>
<tr>
<td>DK</td>
<td>Denmark</td>
<td>MG</td>
<td>Madagascar</td>
<td>UA</td>
<td>Ukraine</td>
</tr>
<tr>
<td>ES</td>
<td>Spain</td>
<td>ML</td>
<td>Mali</td>
<td>US</td>
<td>United States of America</td>
</tr>
<tr>
<td>FI</td>
<td>Finland</td>
<td>MN</td>
<td>Mongolia</td>
<td>UZ</td>
<td>Uzbekistan</td>
</tr>
<tr>
<td>FR</td>
<td>France</td>
<td></td>
<td></td>
<td>VN</td>
<td>Viet Nam</td>
</tr>
<tr>
<td>GA</td>
<td>Gabon</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
OBJECT-ORIENTED AUDIO RECORD/PLAYBACK SYSTEM

Copyright Notification

Portions of this patent application contain materials that are subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

Field of the Invention

This invention generally relates to improvements in computer systems and more particularly to a system for mixing digital representations of analog signals using an object-oriented operating system.

Background of the Invention

Multimedia is perhaps the fastest growing application for computer systems. Increasingly, users are employing computers to present graphic, sound and imaging information to end users. Users are increasingly demanding ergonomic interfaces for managing multimedia presentations. In the past, a time matrix and programming language were used to implement a multimedia presentation. However, simulating a flexible mixing board to enable the presentation of music or sound with the display of information as a multimedia presentation unfolded was not possible.

Examples of current multimedia systems that do not have the capability of the subject invention are Apple's Quicktime and Microsoft's Video for Windows as described in the March issue of NEWMEDIA, "It's Showtime", pp. 36-42 (1993). The importance of obtaining a solution to the routing problem encountered in the prior art is discussed in the March issue of IEEE Spectrum, "Interactive Multimedia", pp. 22-31 (1993); and "The Technology Framework", IEEE Spectrum, pp. 32-39 (1993). The articles point out the importance of an aesthetic interface for controlling multimedia productions.

Summary of the Invention

Accordingly, it is a primary objective of the present invention to provide a system and method for routing multimedia data throughout the course of a multimedia presentation using a computer with a storage and a display. A processor with an attached display, storage and multimedia device builds a component object in the storage of the processor for managing the multimedia device including at least one port for exchanging multimedia information. The
processor includes a connection object for connecting the at least one port to the multimedia device to facilitate the exchange of multimedia information and the processor routes information between the multimedia device and the component object port. A list of component objects are stored in the storage and current status for each of the components in the list is also stored. Then, when a multimedia player is invoked, a test is performed on each of the components in the list, and their associated multimedia devices, to determine what aspects of the multimedia presentation can be run.

Brief Description of the Drawings

Figure 1 is a block diagram of a personal computer system in accordance with a preferred embodiment;

Detailed Description Of The Invention

The invention is preferably practiced in the context of an operating system resident on a personal computer such as the IBM ® PS/2 ® or Apple ® Macintosh ® computer. A representative hardware environment is depicted in Figure 1, which illustrates a typical hardware configuration of a workstation in accordance with the subject invention having a central processing unit 10, such as a conventional microprocessor, and a number of other units interconnected via a system bus 12.

The workstation shown in Figure 1 includes a Random Access Memory (RAM) 14, Read Only Memory (ROM) 16, an I/O adapter 18 for connecting peripheral devices such as disk units 20 to the bus, a user interface adapter 22 for connecting a keyboard 24, a mouse 26, a speaker 28, a microphone 32, and/or other user interface devices such as a touch screen device (not shown) to the bus, a communication adapter 34 for connecting the workstation to a data processing network and a display adapter 36 for connecting the bus to a display device 38. The workstation has resident thereon an operating system such as the Apple System/7 ® operating system.

In a preferred embodiment, the invention is implemented in the C++ programming language using object oriented programming techniques. As will be understood by those skilled in the art, Object-Oriented Programming (OOP) objects are software entities comprising data structures and operations on the data. Together, these elements enable objects to model virtually any real-world entity in terms of its characteristics, represented by its data elements, and its behavior, represented by its data manipulation functions. In this way, objects can model concrete things like people and computers, and they can model abstract concepts like numbers or geometrical concepts. The benefits of object technology arise out of three basic principles: encapsulation, polymorphism and inheritance.

Objects hide, or encapsulate, the internal structure of their data and the algorithms by which their functions work. Instead of exposing these
implementation details, objects present interfaces that represent their abstractions cleanly with no extraneous information. Polymorphism takes encapsulation a step further. The idea is many shapes, one interface. A software component can make a request of another component without knowing exactly what that component is.

The component that receives the request interprets it and determines, according to its variables and data, how to execute the request. The third principle is inheritance, which allows developers to reuse pre-existing design and code. This capability allows developers to avoid creating software from scratch. Rather, through inheritance, developers derive subclasses that inherit behaviors, which the developer then customizes to meet their particular needs.

A prior art approach is to layer objects and class libraries in a procedural environment. Many application frameworks on the market take this design approach. In this design, there are one or more object layers on top of a monolithic operating system. While this approach utilizes all the principles of encapsulation, polymorphism, and inheritance in the object layer, and is a substantial improvement over procedural programming techniques, there are limitations to this approach. These difficulties arise from the fact that while it is easy for a developer to reuse their own objects, it is difficult to use objects from other systems and the developer still needs to reach into the lower, non-object layers with procedural Operating System (OS) calls.

Another aspect of object oriented programming is a framework approach to application development. One of the most rational definitions of frameworks came from Ralph E. Johnson of the University of Illinois and Vincent F. Russo of Purdue. In their 1991 paper, Reusing Object-Oriented Designs, University of Illinois tech report UIUCDCS91-1696 they offer the following definition: "An abstract class is a design of a set of objects that collaborate to carry out a set of responsibilities. Thus, a framework is a set of object classes that collaborate to execute defined sets of computing responsibilities." From a programming standpoint, frameworks are essentially groups of interconnected object classes that provide a pre-fabricated structure of a working application. For example, a user interface framework might provide the support and "default" behavior of drawing windows, scrollbars, menus, etc. Since frameworks are based on object technology, this behavior can be inherited and overridden to allow developers to extend the framework and create customized solutions in a particular area of expertise. This is a major advantage over traditional programming since the programmer is not changing the original code, but rather extending the software. In addition, developers are not blindly working through layers of code because the framework provides architectural guidance and modeling but at the same time frees them to then supply the specific actions unique to the problem domain.
From a business perspective, frameworks can be viewed as a way to encapsulate or embody expertise in a particular knowledge area. Corporate development organizations, Independent Software Vendors (ISV)s and systems integrators have acquired expertise in particular areas, such as manufacturing, accounting, or currency transactions. This expertise is embodied in their code. Frameworks allow organizations to capture and package the common characteristics of that expertise by embodying it in the organization’s code. First, this allows developers to create or extend an application that utilizes the expertise, thus the problem gets solved once and the business rules and design are enforced and used consistently. Also, frameworks and the embodied expertise behind the frameworks, have a strategic asset implication for those organizations who have acquired expertise in vertical markets such as manufacturing, accounting, or bio-technology, and provide a distribution mechanism for packaging, reselling, and deploying their expertise, and furthering the progress and dissemination of technology.

Historically, frameworks have only recently emerged as a mainstream concept on personal computing platforms. This migration has been assisted by the availability of object-oriented languages, such as C++. Traditionally, C++ was found mostly on UNIX systems and researcher’s workstations, rather than on computers in commercial settings. It is languages such as C++ and other object-oriented languages, such as Smalltalk and others, that enabled a number of university and research projects to produce the precursors to today’s commercial frameworks and class libraries. Some examples of these are InterViews from Stanford University, the Andrew toolkit from Carnegie-Mellon University and University of Zurich’s ET++ framework.

Types of frameworks range from application frameworks that assist in developing the user interface, to lower level frameworks that provide basic system software services such as communications, printing, file systems support, graphics, etc. Commercial examples of application frameworks are MacApp (Apple), Bedrock (Symantec), OWL (Borland), NeXTStep App Kit (NeXT), and Smalltalk-80 MVC (ParcPlace).

Programming with frameworks requires a new way of thinking for developers accustomed to other kinds of systems. In fact, it is not like “programming” at all in the traditional sense. In old-style operating systems such as DOS or UNIX, the developer’s own program provides all of the structure. The operating system provides services through system calls—the developer’s program makes the calls when it needs the service and control returns when the service has been provided. The program structure is based on the flow-of-control, which is embodied in the code the developer writes.
When frameworks are used, this is reversed. The developer is no longer responsible for the flow-of-control. The developer must forego the tendency to understand programming tasks in term of flow of execution. Rather, the thinking must be in terms of the responsibilities of the objects, which must rely on the framework to determine when the tasks should execute. Routines written by the developer are activated by code the developer did not write and that the developer never even sees. This flip-flop in control flow can be a significant psychological barrier for developers experienced only in procedural programming. Once this is understood, however, framework programming requires much less work than other types of programming.

In the same way that an application framework provides the developer with prefab functionality, system frameworks, such as those included in a preferred embodiment, leverage the same concept by providing system level services, which developers, such as system programmers, use to subclass/override to create customized solutions. For example, consider a multimedia framework which could provide the foundation for supporting new and diverse devices such as audio, video, MIDI, animation, etc. The developer that needed to support a new kind of device would have to write a device driver. To do this with a framework, the developer only needs to supply the characteristics and behaviors that are specific to that new device.

The developer in this case supplies an implementation for certain member functions that will be called by the multimedia framework. An immediate benefit to the developer is that the generic code needed for each category of device is already provided by the multimedia framework. This means less code for the device driver developer to write, test, and debug. Another example of using system frameworks would be to have separate I/O frameworks for SCSI devices, NuBus cards, and graphics devices. Because there is inherited functionality, each framework provides support for common functionality found in its device category. Other developers could then depend on these consistent interfaces for implementing other kinds of devices.

A preferred embodiment takes the concept of frameworks and applies it throughout the entire system. For the commercial or corporate developer, systems integrator, or OEM, this means all the advantages that have been illustrated for a framework such as MacApp can be leveraged not only at the application level for such things as text and user interfaces, but also at the system level, for services such as graphics, multimedia, file systems, I/O, testing, etc.

Application creation in the architecture of a preferred embodiment will essentially be like writing domain-specific pieces that adhere to the framework protocol. In this manner, the whole concept of programming changes. Instead of
writing line after line of code that calls multiple API hierarchies, software will be
developed by deriving classes from the preexisting frameworks within this
environment, and then adding new behavior and/or overriding inherited behavior
as desired.

Thus, the developer's application becomes the collection of code that is
written and shared with all the other framework applications. This is a powerful
concept because developers will be able to build on each other's work. This also
provides the developer the flexibility to customize as much or as little as needed.
Some frameworks will be used just as they are. In some cases, the amount of
customization will be minimal, so the piece the developer plugs in will be small. In
other cases, the developer may make very extensive modifications and create
something completely new.

In a preferred embodiment, as shown in Figure 1, a multimedia data routing
system manages the movement of multimedia information through the computer
system, while multiple media components resident in the RAM 14, and under the
control of the CPU 10, or externally attached via the bus 12 or communication
adapter 34, are responsible for presenting multimedia information. No central
player is necessary to coordinate or manage the overall processing of the system.
This architecture provides flexibility and provides for increased extensibility as new
media types are added. The system makes use of a variety of multimedia objects,
some of which that can be used as connecting objects. The connecting objects
include gain, filters, amplifiers, mixers, players and other multimedia components
that are individually implemented as objects in an object oriented operating system.
Objects, as discussed above include a code or method component and a data
component. The system includes a mouse for facilitating iconic operations such as
drag/drop, double-clicking, drop-launching, cursor positioning and other typical
operations.

In video and audio production studios, media such as sound, MIDI, and
video make use of physical patch cords to route signals between sources, effects
processors, and sinks. Signal processing algorithms are also often represented as
networks of sources, sinks, and processors. Both of these models can be represented
as directed graphs of objects that are connected. A preferred embodiment allows
this model - connecting objects together - to be realized on a computer system.
Figure 2 illustrates a prior art, simple, home studio setup utilizing a tape deck,
mixer, reverberation unit, pair of microphones, and pair of speakers. Since the
microphones are connected to the tape deck, sound input is routed from the
microphone to the tape deck, where it can be recorded. When the tape deck plays
back, its signal is routed to the mixer because of the connection from the tape deck
to the mixer. Similarly, the reverberation unit and the speakers are connected an
amplifier connected to the mixer.

A preferred embodiment utilizes object-oriented technology to represent a
connection model. Multimedia objects can be connected to each other, creating
directed data flow graphs. In addition, a standard set of multimedia objects is
defined to the system. These objects can be connected together to facilitate
multimedia data flow from object to object. The connection operations can be
facilitated by connecting multimedia objects via a geometric figure such as a line,
line segment or other appropriate geometry. The figures discussed below show
examples of various multimedia objects, including connecting objects and the
geometric figures that are used to represent the internal data structures and logic
joining the multimedia objects.

Albert Architecture Patent

15 Application Programming Interface
The “basic components” are
– a fixed set of Geometric Primitives (Point, Rectangle, Line, Curve, Polygon,
Polyline, Area in 2D; Line, Polyline, Curve and Surface in 3D. This set of geometry
is not intend to be user extensible. This limits the complexity of the lower level
graphic devices, and provides a “contract” between the user–level API and the low
level device for consistent data.
– Discretized data sets. This include 2D raster images (with a number of possible
components) and triangulated 3D datasets.
– High level modelling tools, that can express hierarchical groups of graphic objects.
– Transforms. These objects represent the operations available with a traditional
3X3 (in 2D) or 4X4 (in 3D) matrices to rotate, scale, translate, etc. objects.
– Bundles
These encapsulate the appearance of the geometry. Standard attributes include (2D
& 3D) frame and/or fill color (fill for area enclosing geometry only), pen thickness,
dash patterns, etc. In 3D, bundles also define shading attributes, a la renderman.
Custom attributes (often specific to a particular device) may be specified via a
keyword/value pair.

All numeric values are expressed in double precision floating point.

API uses model of:

    port->Draw( geometry, bundle, transform )

The latter two default to an “empty” bundle (using the port’s default) and an
identity transform. The port (more formally, a TGrafPort) is an application–level
view that encapsulates the state. The TGrafPort re-routes the Draw() call to a
number of possible devices (monitors, off screen frame buffers, PostScriptPrinter on a network, a window, etc.).

Graphical "state" (current transform, bundle, clipping region, etc) is managed by Albert at the port level. However, at the device level Albert is "stateless", i.e., the complete state for a particular rendering operation is presented to the device when that rendering occurs.

The Grafport translates the "port->Draw()" call described above into a device level "Render()" call as follows:

TGrailPort::Draw( geometry, bundle, transform )
{
    GetGrafportDevice()->Render( geometry, grafstate )
}

Note the device may turn around and invoke other devices (for example, a device for the entire desktop may first decide which screen the geometry falls upon, and then invoke the render call for that particular screen.)

Art–etecture diagram
In the diagram above, a modelling layer (described below, either user-defined or provided by Albert) generates calls to a GrafPort 210 using the API described above. This GrafPort interface accepts only a specific, fixed set of primitives forming a “contract” 250 between the user level API and the device level API. The graphport captures state (including transform, appearance (“bundle”), clipping, etc.) into a polymorphic cache 220 that is used across multiple types of devices\(^1\). For each render call, the geometry and all relevant accumulated state 230 is presented at once to the device via a polymorphic device interface 240. This device may take the form of a page description language 260 (such as PostScript\(^\text{TM}\)), a vector plotting device 270, a device with custom electronic hardware for rendering geometric primitives 280, or a traditional framebuffer 290 (either on or off screen).

**Modelling Layer**

Above the GrafPort and geometry layers there is an optional modelling layer. The modelling layer may be user-supplied, or the user may use the implementation
provided by Albert, call the "MGraphic" layer. An MGraphic object encapsulates both geometry and appearance (a bundle). To render an MGraphic a Draw method is used. This takes a the GrafPort the MGraphic is drawn into as an argument. The MGraphic Draw method turns this into a GrafPort calls like this:

```cpp
MGraphic::Draw( grafport )
{
    grafport->Draw( fGeometry, fBundle );
}
```

[More specifics can be lifted from the MGraphic & MGraphic3D ERS, as well as some figures to go with it. This is the portion of the architecture that covers the same ground as the Ingalls paper]

The goal behind separating the MGraphic layer from the GrafPort/Geometry layer is to avoid a rigid structure suited to only one type of database. If the structure provided by the MGraphic objects does not meet the client's needs, the architecture still permits a different data structure to be used, as long as it can be expressed in terms of primitive geometries, bundles, and transforms.

```cpp
```
CLAIMS

Having thus described our invention, what we claim as new, and desire to secure by Letters Patent is:

1. A system for multimedia presentations, comprising:
   (a) a processor;
   (b) a storage under the control of and attached to the processor;
   (c) a display under the control of and attached to the processor;
   (d) a plurality of multimedia devices under the control of and attached to the processor;
   (e) a component object in the storage of the processor for managing the multimedia device including at least one port for exchanging multimedia information;
   (f) means for storing a list of component objects in the storage;
   (g) means for marking the list of component objects to indicate resort;
   (h) means for testing each of the component objects on the list and activating the multimedia device associated with the component object if the multimedia device associated with the component object can process information;
   (i) means for connecting the at least one port to the multimedia device to facilitate the exchange of multimedia information; and
   (j) means under the control of the processor for routing information between the multimedia device and the component object port.

2. The system as recited in claim 1, including means for deactivating components.

3. The system as recited in claim 1, including means for ordering a plurality of component objects in the storage of the processor so that component objects that produce a multimedia output are processed before component objects requiring the multimedia output.

4. The system as recited in claim 3, including means for processing a second component object that is dependent upon the output of a first component object after the first component object is processed.

5. The system as recited in claim 4, including means for managing component objects so that each component object performs its processing operation within a fixed period of time (frame).
6. The system as recited in claim 4, including means for the second component
object dependent upon the output of the first component object to wait until
the first component object has completed processing to commence processing
of the second component object.

7. The system as recited in claim 3, including means for recursively invoking
the plurality of component objects in their designated order.

8. A method for facilitating multimedia presentations on a processor with an
attached storage, multimedia device and display under the control of the
processor, comprising the steps of:
(a) creating a component object in the storage of the processor including at least
one port for exchanging multimedia information;
(b) managing the multimedia device with the component object;
(c) storing a list of component objects in the storage;
(d) marking the list of component objects to indicate resort;
(e) testing each of the component objects on the list and activating the
multimedia device associated with the component object if the multimedia
device associated with the component object can process information;
(f) connecting the at least one port to the multimedia device to facilitate the
exchange of multimedia information; and
(g) routing information between the multimedia device and the component
object port.

9. The method as recited in claim 8, including the step of connecting a plurality
of multimedia devices and their associated component objects and
exchanging multimedia information.

10. The method as recited in claim 8, including the step of ordering a plurality of
component objects in the storage of the processor so that component objects
that produce a multimedia output are processed before component objects
requiring the multimedia output.

11. The method as recited in claim 10, including the step of processing a second
component object that is dependent upon the output of a first component
object after the first component object is processed.
12. The method as recited in claim 11, including the step of managing component objects so that each component object performs its processing operation within a fixed period of time (frame).

13. The method as recited in claim 11, including the step of having the second component object dependent upon the output of the first component object wait until the first component object has completed processing to commence processing of the second component object.

14. The method as recited in claim 10, including the step of recursively invoking the plurality of component objects in their designated order.
FIGURE 2

FIGURE 3

FIGURE 4

FIGURE 5
FIGURE 9

FIGURE 10

<table>
<thead>
<tr>
<th>Audio Type</th>
<th>Sample Rate (samples/sec)</th>
<th>Sample Width (bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>k44KHz16BitLinear</td>
<td>44100</td>
<td>16</td>
<td>Compact Disc Quality. Two's complement integer samples.</td>
</tr>
<tr>
<td>k22KHz8bitOffsetBinary</td>
<td>22050</td>
<td>8</td>
<td>Medium quality. Offset binary (128 is silence) samples.</td>
</tr>
<tr>
<td>k8KHzMuLaw</td>
<td>8000</td>
<td>8</td>
<td>North American telephone standard. Coded samples.</td>
</tr>
<tr>
<td>k8KHzALaw</td>
<td>8000</td>
<td>8</td>
<td>European telephone standard. Coded samples.</td>
</tr>
</tbody>
</table>

FIGURE 11
FIGURE 12

FIGURE 13

FIGURE 14
FIGURE 15

FIGURE 16

FIGURE 17

Audio output port :Fire() → Audio Input Port :IsAvailable() = TRUE if output port has fired, FALSE otherwise.
FIGURE 18
1900
Activate audio processor

1910
\[ i = 0 \]
\[ N = \text{number of audio processor output ports} \]

1920
\text{Does } i = N? \]

1930
\text{return}

1940
Fire output port \( i \)

1950
Mark output port \( i \) as having fired

1960
Get audio processor that is connected to output port \( i \)

1970
\[ i = i + 1 \]

1980
\text{Can audio processor run?}

1990
Recursively call
\text{Activate audio processor}

\text{FIGURE 19}
Deactivate audio processor

\[ i = 0 \]
\[ N = \text{number of audio processor's output ports} \]

\[ \text{Does } i = N? \]
\[ \text{Mark output port } i \text{ as having NOT fired} \]

\[ i = i + 1 \]

\[ \text{Can audio processor run?} \]
\[ \text{Deactivate audio processor} \]

\[ \text{Get audio processor that is connected to output port } i \]

\[ \text{Does } i = N? \]
\[ \text{Mark input port } i \text{ as having NOT fired} \]

\[ i = i + 1 \]

\[ \text{Can audio processor run?} \]
\[ \text{Deactivate audio processor} \]

FIGURE 20

\[ i = 0 \]
\[ N = \text{number of audio processor's input ports} \]

\[ \text{Does } i = N? \]
\[ \text{return} \]

\[ \text{Get audio processor that is connected to input port } i \]

\[ i = i + 1 \]

\[ \text{Can audio processor run?} \]

\[ \text{Deactivate audio processor} \]
10/25

Start 2100

2104

Is Run List marked "valid"?

N

2106

Perform a Topological Sort on the network of audio processors. For each audio processor visited, if it has an output that is marked as having fired, add it in order to the run list.

Y

2100

2110

Mark run list as "valid"

i = 0

N = number of audio processors in the run list

2120

2122

Does i = N?

Y

N

2124

Run audio processor i

2126

i = i + 1

End 2130

FIGURE 21

SUBSTITUTE SHEET (RULE 26)
12/25

Start 2500

2510 Is this port connected? N Throw exception 2520

Y 2530

Is connected port in same address space

Y 2550

Copy pointer to graphic object into memory location

N 2540

Copy entire graphic object into shared memory.

2560

Send "graphic object ready" notification to input port.

2570

Block thread until input port sends "done with graphic" notification.

End 2580

FIGURE 25
Start 2600

Is this port connected? 2610

Y → 2630
Is "graphic ready"

N → Throw exception 2620

Y → Block thread until "graphic object ready" notification is received. 2640

Return pointer to graphic object. 2650

End 2660

FIGURE 26
Start 2700

2710

Is this port connected?

2720

Throw exception

2730

Send "done with graphic" notification to output port...

2740

Is connected port in same address space?

2750

Delete copy of graphic object, it's no longer referred to.

End 2760

FIGURE 27
<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Status</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 34**

![Diagram](image-url)

**FIGURE 35**
Start

i = 0
N = number of connected input ports

Does i = N?

Insert copy of packet into port i's buffer

Buffer Empty?

Send "buffer not empty" notification to all input ports.

i = i + 1

return

FIGURE 36
Start

Buffer Empty?

Y
Block thread by waiting for "buffer not empty" or "cancel" notification.

N
Copy packet from buffer to caller

Was wait cancelled?

Y
Throw exception

N
return

FIGURE 37

FIGURE 38

FIGURE 39
FIGURE 40

FIGURE 41

FIGURE 42

FIGURE 43
21/25

FIGURE 44

FIGURE 45

FIGURE 46

FIGURE 47

FIGURE 48
FIGURE 49

Audio MultiConverter

FIGURE 50

Sound

FIGURE 51

5100

5110  5120  5130  5140  5150

FIGURE 52

Physical Speaker
Physical Microphone

FIGURE 53

Graphic Player

FIGURE 54

Graphic Viewer

FIGURE 55

Video Digitizer

FIGURE 56
A. CLASSIFICATION OF SUBJECT MATTER
   IPC 6 G06F9/44 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED
   Minimum documentation searched (classification system followed by classification symbols)
   IPC 6 G06F

   Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

   Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
</table>
| Y        | COMPUTER JOURNAL
          | vol. 36, no. 1, October 1992, LONDON GB pages 4 - 18
          | F. HORN ET AL. 'On programming and
          | supporting multimedia object
          | synchronization' see the whole document
          | Y                                                                                 | 1, 2, 8, 9            |
| Y        | US, A, 5 057 996 (DIGITAL EQUIPMENT CORPORATION) 15 October 1991
          | see the whole document
          | Y                                                                                 | 1, 2, 8, 9            |

Further documents are listed in the continuation of box C.

Patent family members are listed in annex.

1. Date of the actual completion of the international search
   4 May 1994

2. Date of mailing of the international search report
   24.05.94

Name and mailing address of the ISA
   European Patent Office, P.B. 5818 Patentlaan 2
   NL - 2280 HV Rijswijk
   Tel: (+31-70) 340-2040, Tw: 31 651 epos nl,
   Fax: (+31-70) 340-3216

Authorized officer
   Katerbau, R
<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>COMPUTER COMMUNICATIONS&lt;br&gt;vol. 15, no. 10, December 1992, GUILDFORD GB&lt;br&gt;pages 611 - 618&lt;br&gt;G. BLAKOWSKI ET AL. 'Tool support for the synchronization and presentation of distributed multimedia'&lt;br&gt;see the whole document</td>
<td>1,8</td>
</tr>
<tr>
<td>A</td>
<td>US A 5 129 083 (DIGITAL EQUIPMENT CORPORATION) 7 July 1992&lt;br&gt;see the whole document</td>
<td>1,8</td>
</tr>
<tr>
<td>Patent document cited in search report</td>
<td>Publication date</td>
<td>Patent family member(s)</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-----------------</td>
<td>-------------------------</td>
</tr>
</tbody>
</table>

Form PCT/ISA/210 (patent family annex) (July 1992)