(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 365 386 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
26.11.2003 Bulletin 2003/48

(51) Int Cl.[7]: **G10H 1/00**

(21) Application number: **02425318.9**

(22) Date of filing: **20.05.2002**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE TR
Designated Extension States:
AL LT LV MK RO SI

(71) Applicant: **Jinglebell Communication S.R.L.**
20123 Milano (IT)

(72) Inventor: **Nannicini, Alessandro**
20127 Milano (IT)

(74) Representative: **Riccardi, Elisa et al**
Porta, Checcacci & Associati S.p.A.
Viale Sabotino 19/2
20135 Milano (IT)

(54) Digital sound management

(57) In order to improve the playing out of digital sound data in an electronic computer system, in particular when the resources for storing and/or loading them are limited, as well as in order to provide more dynamic management of the digital sound data themselves, a method is described including: ad hoc fragmentation of the digital sound file representing the desired sound passage; loading the digital sound fragments into the computer and in particular downloading the fragments from the Internet; and playing out or executing the digital sound fragments in a manner controlled suitably by software, in particular by playing out immediately at least once, and preferably cyclically several times, the first fragment (down)loaded, without waiting for (down)loading of the second fragment and of the subsequent fragments to be completed, more preferably before starting to (down)load the second fragment and the subsequent fragments.

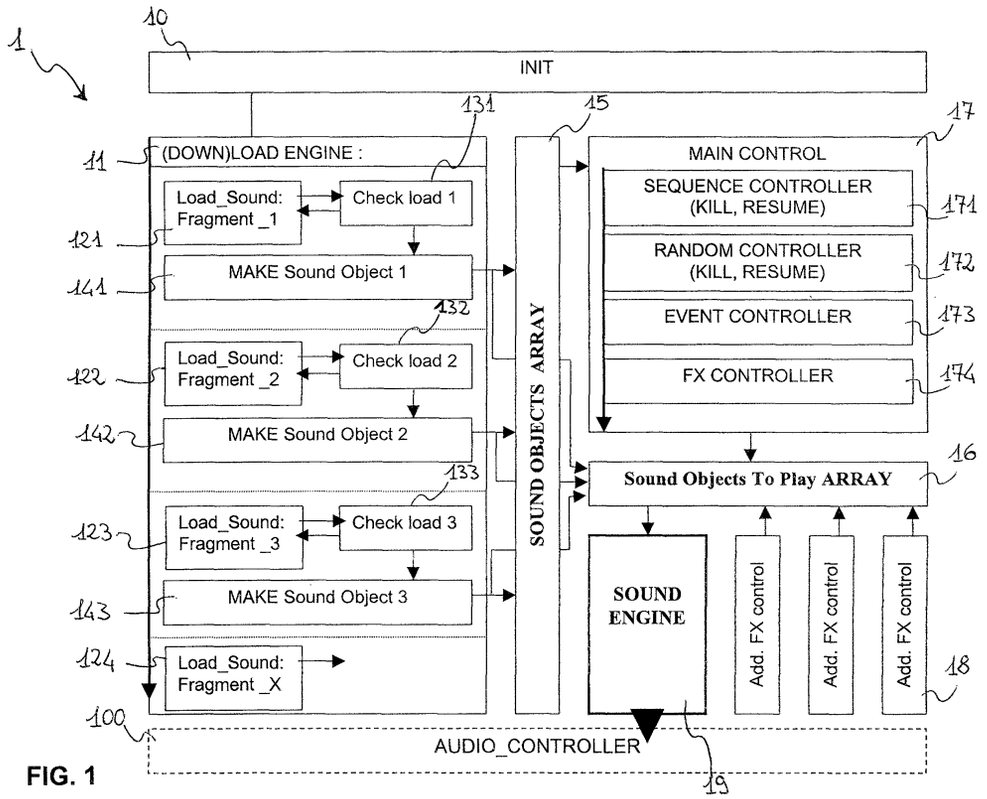

FIG. 1

EP 1 365 386 A1

Description

[0001] The present invention relates to the field of digital sound management in an electronic processor system (briefly, a computer).

[0002] It is well known that over the last decades, universal IT networks and in particular the Internet, have developed increasingly, offering more and more appealing sites, with the introduction of static images, moving graphics or animation, digital video footage and, also, this being significant for the purposes of the present invention, with the introduction of digital sound.

[0003] There are several known techniques for managing digital sound files intended to be transmitted from a server computer (the site manager) to a client computer (the user who links up with that site).

[0004] A first technique consists of sending, from the server to the client, a whole sound passage (music or speech) split up into packets, as is done for transmission of data of any kind whatsoever via the Internet. The packets, which are structured and sent in a sequential order by the server, but — because of the very nature of the Internet - not necessarily received in the same sequential order by the client, are reconstructed on the client computer before the passage is played out. More specifically, the client computer waits until receipt of all the packets has been completed, reconstructs the digital sound file by sequentially connecting the packets, and only then does it play out the sound file without any breaks in it.

[0005] One development of this technique consisted of the compression of the digital sound files by the server computer before they were fragmented into packets and sent.

[0006] The aforesaid technique, with or without compression, is well suitable for sending music or other digital sound files to a user (at the client computer) who deliberately decides to listen to it, and who is therefore willing to wait for the whole time required for all the packets making up the digital sound file to be received (downloading). It must be pointed out, however, that the such required time depends on many factors, including the speed of the physical link, the sending speed (related to the workload of the server), the speed of transmission (related to the receiving and forwarding times of the various different computers along the path that each packet follows within the network) and, to a certain extent, the speed of reception (conditioned by the workload of the client).

[0007] For this reason, the aforesaid technique is not suitable for sending a digital sound file regardless of the will of the user to listen to it, such as a background music associated with a web page. This is because there is a risk of the user navigating elsewhere before the process of reception of all the packets has been completed and therefore of him not enjoying the digital sound file associated with that web page being played out. It is obvious, however, that providing a soundtrack for web pages is desirable in order to make them more appealing.

[0008] Another technique recently developed for the same purpose provides for the use of special additions (plug-ins) to the various types of navigation software (browsers) available on the market, for providing soundtracks for web sites or pages. In these cases, they are genuine programs that are installed in the client computer and that are provided with one or more sound libraries. In order to play the digital sound, the web pages have to be written specifically for sending instructions to the plug-in, which instructions have to either identify the sounds to be played out among those available in the libraries already contained in the client computer, or require additional sound libraries to be downloaded and installed before use. In particular, in order to make the sound interactive with the graphical user interface (GUI), particularly with the .HTML page, for example in order to associate a specific sound event with the position of the pointer controlled by the tracking device (mouse, track-ball, etc.), the web page itself must be interspersed with special instructions. The initial creation of a web page, as well as the subsequent addition of a soundtrack, therefore make the web page far bulkier and, above all, entail lengthy programming times.

[0009] A further technique used at times consists of associating a small digital sound file, often tailored specifically for this purpose, with a web page; this file is played out repeatedly in loop fashion by the client computer to form a sound passage. However, precisely because of its repetitive nature, playing out a sound passage so conceived may sometimes have the opposite effect to that desired, which would be to improve the communication impact of the web page by making it more pleasant to visit and more appealing. To attenuate the repetitive effect, the digital sound file to be played out cyclically must also be fairly long, which may result in the initial waiting time between accessing the page and playing out the soundtrack still being considerable, especially for low-speed links or if the hardware and software resources of the client computer are limited.

[0010] Lastly, another technique being used currently is the so-called "Streaming Audio/Video Internet", which consists of continuous delivery of digital sound data from the server to the client while the web page is being viewed. This technology is extremely disadvantageous for the server since it requires a high-speed broadband link, in addition to very high processing power. The same requisites in terms of broadband and high speed link also have to be met by the client computer. The sounds incorporated into web pages according to this technology therefore typically feature very low quality and, notwithstanding, a considerable bulk of data delivered per second.

[0011] The technical problem underlying the present invention is that of improving the playing out of digital sound data in an electronic computer system, in particular when there are limited storing and/or loading resources, and also

that of making more dynamic management of the digital sound data available.

[0012] In the present invention and in the attached claims, the term "electronic computer system" or "computer" is used to encompass stand-alone or networked computers, such as personal computers (PC), laptops, workstations, et cetera, as well as dedicated processing units such as, by way of example, consoles for video games, interactive decoders and the like. In these latter cases, as also in the case of a computer intended to play out digital sound data off-line, the technical problem described above may arise both as a result of the lack of immediate availability of the whole file of digital sound data, due for example to insufficient bulk storage capacity, and as a result of the unavailability of other resources for playing out the digital sound data sequentially and uninterruptedly, due for example to overloading of the temporary storage facility (RAM) in which the digital sound data are loaded in order to be played out.

[0013] Furthermore, in the rest of the present invention and in the attached claims, all references to the Internet should be construed as made by way of example of a network of computers, including LANs, WANs, Intranets, etc.

[0014] The problem referred to above is solved, according to the invention, by means of a method for playing digital sound at an electronic computer system according to claim 1. The invention concerns, furthermore, a computer program including a code for implementing said method when it is carried out within an electronic computer system, according to claim 24, a corresponding computer program product, according to claim 25, and an electronic computer system including means for implementing the method, according to claim 30. The invention also concerns a method for causing the method of playing out to be executed, according to claim 31. The invention also concerns a parametric computer program according to claim 33, a corresponding program product according to claim 36 and an electronic computer system in which such a parametric computer program is loaded, according to claim 37. The invention also concerns a dynamic data structure in an electronic computer system according to claim 38.

[0015] Further advantageous features of the present invention are claimed in the secondary claims.

[0016] Indeed, the Applicant perceived that the technical problem described above may be solved in the following manner: ad hoc fragmentation of the digital sound file representative of the desired sound passage; loading of the digital sound fragments into the computer and in particular downloading of the fragments from the Internet; and the playing out or performance of the digital sound fragments in a manner that is suitably controlled by software, in particular by immediately playing out the first (down)loaded fragment, at least once and preferably several times cyclically, without waiting for (down)loading of the second and of the subsequent fragments to be completed, and more preferably before initiating (down)loading of the second and of the subsequent fragments.

[0017] The first digital sound fragment is, therefore, played out almost instantaneously, thus basically eliminating the waiting times.

[0018] In other words, a first digital sound fragment, lasting only for a short time and therefore of a small size, is loaded (in particular downloaded from the network, the Internet or other) and played out immediately, preferably in loop fashion, for the purpose of simulating continuity by means of its repetition, assuming naturally that the musical nature of said sound fragment enables it to be played out in loop fashion. In the meantime, other fragments are (down)loaded and played, overlapping or replacing said first digital sound fragment or other fragments that have been (down)loaded, as explained more precisely elsewhere.

[0019] In this way it is possible to build whole digital sound passages (arrangements) that are enriched progressively with more fragments, played out cyclically or not, (down)loaded from the network one at a time, thus eliminating the waiting times: this is because the first digital sound fragment is played out as soon as it reaches the client, and meanwhile downloading of new digital sound fragments continues in the background while playing out the material as it becomes available. The first digital sound fragment has therefore to be interpreted as an introductory element, which should be of a "light" nature, that is to say short, and effective, that is to say capable of creating a certain atmosphere even when recurring on its own.

[0020] To provide a quantitative example, the size of a fragment lasting 3 seconds will be only 40 Kbytes even if it is coded with a high resolution in order to preserve the listening quality, such as by way of example in MP3 (joint stereo) format at 64 or 80 Kbps. The average time required for downloading such a fragment corresponds therefore to the average time required for downloading an image with a size, for example, of 320x240 pixels with medium to high compression. In practice, this lapse of time is therefore imperceptible on the part of the user or in any case more than acceptable even in the least favourable links.

[0021] By controlling the synchronism between the digital sound fragments that have already been (down)loaded, it is possible to play out a number of them, preferably on more than one sound channel, to obtain the effect of an arrangement - that is to say breaking up the monotony of the cyclic repetition of a single fragment - in two different ways:

- by increasing the harmonic depth, that is to say superimposing several fragments (verticality) and/or
- by playing a new fragment on stopping the previous one(s) (linearity).

[0022] In a way, each digital sound fragment constitutes one sound "tessera" of a sound "mosaic", that constitutes

the final sound passage, so that in this way the passage acquires an almost infinite musical potential. The arrangement is progressively enriched by playing, also cyclically if required, several fragments downloaded one at a time from the network.

[0023] Indeed, according to the invention the multitasking capabilities of present-day Central Processing Units (hereinafter abbreviated as CPU) and of the associated Operating Systems (OS) are exploited in full in order to make up for the shortcomings in terms of speed of connection to the network, which is currently not evolving at the same pace as CPUs are at popular level, or in any case in order to make up for other factors that have a negative impact on the possibility of immediate access to substantial quantities of digital sound data. More specifically, a sound engine is provided that seizes and "masks" the activities of (down)loading of the sound fragments by playing out those that have already been (down)loaded, possibly in the recursive mode.

[0024] A further advantage of the present invention as compared with the known techniques referred to above consists of the fact that it does not need pre-installing in the client computer either sound libraries or other software, apart of course - as far as concerns the preferred use of the method according to the invention for broadcasting over the Internet - from a web browser, sound management hardware/software such as a sound card and loudspeakers and, possibly, a plug-in enabling the browser to manage the sound. One plug-in that is particularly useful in the framework of the present invention, as will be understood later, is Macromedia Flash Player™ by Macromedia, Inc. of San Francisco, California, U.S.A., which has become so widespread that it is a de facto standard for transmitting sound and animated graphics via the Internet. Thus, for the great majority of users, a program implementing the method according to the invention developed using the Macromedia Flash™ application, again by Macromedia, Inc., would not even require ad hoc installation of a plug-in for sound backing. Lastly, said plug-in has the considerable advantage of managing compiled files (files with a .swf extension), which protect both the sound contents and the software source.

[0025] Since the Macromedia Flash™ program also contains, in addition to programming tools, all the functions of the Macromedia Flash Player™ program or plug-in, in the rest of the present description and in the attached claims the expression Macromedia Flash (Player)™ will at times be used to refer indiscriminately to one or the other.

[0026] It is worthwhile, however, to point out here and now that digital sound management according to the present invention could be implemented using various different technologies, for example Javascript which, as it is well known, is an effective standard programming language for communicating on the Internet, which can be incorporated directly into the .html source of the web pages and that is run by the browser on the client computer, as well as Java, that is to say by having a .class executable file run by the Java Virtual Machine (JVM), the latter being part of the standard supply of browsers available on the market. Neither of these solutions requires any further plug-ins to be installed. In yet another technology, the invention could be implemented by means of a program developed in C++ or in C# or even by means of a small plug-in to be installed.

[0027] In the practice of the embodiment of the invention implemented by means of the Macromedia Flash (Player)™ plug-in mentioned above, the various different digital sound files forming the sound fragments that are part of the complex passage are (down)loaded once only to the (client) computer, and stored immediately in the "cache" memory of the browser (that is to say in the area or directory intended for containing temporary Internet files), enabling the digital sound fragments that have already been played out to be retrieved at a later time with no need to trigger (down) loading process once again. In this way, and also thanks to management of the digital sound files according to a special data structure that will be described below, it is possible to:

- Call up, while the complex sound passage is being performed, digital sound fragments that have already been played out earlier and were then stopped, and which are needed again;

- Retain a copy of the digital sound files in the client computer, thus further cutting the waiting time during a further access to the same web page or when listening again to the same complex sound passage;

- Possibly, by means of a further software module, retaining a copy of the digital sound files in the client computer, organising them suitably in other areas of the bulk storage so as to create, gradually, a sound library for use by other complex sound passages;

- Erase the digital sound files either after visiting the web page or periodically, in order to avoid them taking up space in the bulk storage facility of the client computer.

[0028] The various different fragments can be played out on a single channel, providing merging of the digital data of two (or more) fragments by specific software programs.

[0029] Preferably, however, the various different fragments are played out on several different sound channels, more preferably on several stereo sound channels.

[0030] Multi-channel playing out of several sound files is an opportunity provided by most current computers, and is

managed directly by the hardware and software installed in the client computer, in particular by the operating system and by the specific sound controller (sound board).

[0031] Multi-channel playing out of several digital sound files is currently exploited in the field of multimedia products such as video games, in order to enable the various effects to be managed separately depending on the events occurring in the game being played and, possibly, on the user's own preferences. In particular the following may be carried out:

- Independent volume adjustments for each sound channel;
- Stereo balancing effects (panpotting), that is to say different adjustment of the sound output from the left channel and from the right channel of each stereo sound channel;
- Independent activation and de-activation of each sound channel;
- Independent replacement of the digital sound fragment to be played on each sound channel;
- Advanced transition effects such as, for example, cross-fading between two sound channels, that is to say a fade-in effect on a first channel simultaneously to a fade-out effect on a second channel.

[0032] These operations may also be controlled dynamically in several different ways, providing all in all a very large capability of sound expression.

[0033] In the framework of the present invention, in addition to controlling how the various different digital sound fragments are played out, either individually (number of repetitions, volume and so on) and as a whole (sequence of the fragments), by means of suitable programming, including parametric programming, by the author, it is therefore also possible:

- To control playing out of the various different digital sound fragments in response to events triggered by other applications with which the sound is associated, for example the showing of footage;
- To control playing out of the various different digital sound fragments in such a way that the sound undergoes variations in response to actions triggered by the user by means of a Graphical User Interface (GUI), and in particular the GUI of the web page that is being visited (interactivity);
- To subordinate control of the way in which the various different digital sound fragments are played out to automatic events in time or events in time programmed by the user, for example at a given time of each day;
- To program playing of the digital sound fragments in such a way that there is an autonomous variation in the way in which the complex passage is played out, with random trajectories, by means of random simulation algorithms.

[0034] The various different digital sound fragments are suitably correlated with one another while being played out. According to one advantageous feature of the present invention, the co-ordination pattern is drafted with the help of a time grid. Each of the digital sound fragments is planned for playing out once or more in this time grid.

[0035] In a preferred embodiment, the time grid is split up according to a primary unit of time (LOOP), the length of which is expressed typically in milliseconds.

[0036] As stated above, in order to cut down the waiting times as much as possible, it is preferable for at least the first digital sound fragment to be intended for cyclic repetition. It will therefore be provided, typically, with a length equal to one primary unit of time, that is to say equal to 1 LOOP, and so as to be played for a given number of primary units of time LOOPs along the time grid.

[0037] Preferably, a secondary unit of time CYCLE is also established in the time grid, being a multiple of the primary unit LOOP and therefore expressed by an adimensional number. The secondary unit of time or CYCLE is particularly useful since it enables the information concerning the point in time when the playing of each digital sound fragment is scheduled to be limited. In other words, when performance of the complex passage features a certain degree of cyclicity, by using only the partition into primary units of time LOOPs, it would be necessary to have information at disposal concerning all the primary units of time LOOPs of the time grid during which a particular digital sound fragment has to be played out; on the other hand, by further providing for the secondary unit of time CYCLE, it will be sufficient to have information at disposal concerning all the primary units of time LOOPs of each secondary unit of time CYCLE during which a particular digital sound fragment has to be played out.

[0038] Similarly, it is also possible to group more than one secondary unit of time CYCLES into tertiary units of time

GROUPS and so on.

[0039] Using the time grid defined at least by the primary unit of time LOOP but preferably also by the units of time of a higher order CYCLEs, GROUPs, etc., also enables easy management of the synchronism between the digital sound fragments being played out and graphic parts, in particular animated graphics. It is thus possible to associate an image with a specific digital sound fragment and to control the displaying of the picture in the Graphical User Interface, in particular on the web page, so that the selected frames thereof are displayed while the digital sound fragment associated with it is being played out.

[0040] Of course, it is equally possible to associate more than one picture with each digital sound fragment, or also to associate more than one digital sound fragment with each picture.

[0041] In greater detail, in order to control synchronism of the graphics or animation with the playing out of the digital sound fragment associated with it, it is possible to associate a sequence of preselected frames, or key frames, of the animation with each digital sound fragment, each key frame being intended to be played out at the beginning of a particular primary unit of time LOOP. In other words, similarly to the definition of a playing sequence (START_POINTS) expressed in LOOPs and associated with the digital sound fragment in the manner described above, it is possible to associate a playing sequence of key frames (SPIN_POINTS), expressed in a similar manner, with each animation, as described in greater detail below.

[0042] It is possible to insert fragments of speech among the digital sound fragments. Speech is a manner of expression that is linear by nature, and therefore is not suitable for being played out recursively, however it can be partitioned into sentences, each of which can be stored in a digital sound fragment. The integration of digital sound fragments representative of speech with digital sound fragments representative of music and their management according to the present invention is advantageous in that the music fragments being played out cover the time required for (down)loading the digital sound fragments representative of the speech. It should also be noted that the time required for (down)loading the digital sound fragments representative of the speech is in any case short, since the human voice is often monophonic and its tone colour covers a more limited frequency range, and it can be compressed to a greater extent than digital sound files representative of music. It is also worthwhile stressing that multi-channel management of the digital sound fragments, of music and of speech, enables effects of lowering of the background music dynamics to be created while speech is played out, so that the voice will stand out, even automatically, with a "cinematographic" effect.

[0043] Management of the sequence according to which the digital sound fragments are played out can also be controlled by events determined by interaction of the user on the Graphical User Interface (GUI), in particular on the web page. As stated above, according to the prior art, interaction with a web page is intercepted by assigning events to the objects concerned, one by one, in the HTML source code. According to a particularly advantageous aspect of the present invention it is possible, on the other hand, to create a kind of universal statement valid for all the categories of objects of a web page, such as hypertext links, graphic maps, forms, et cetera, and their respective events, such as the pointer controlled by the tracking device (mouse, track-ball et cetera) passing within the boundaries of the object (OnMouseOver and OnMouseOut events), a key being pressed on the tracking device within the boundaries of the object (e.g. OnClick, OnRightClick events), keys being pressed on the keyboard, et cetera. This universal statement enables the HTML page to be activated automatically in order to dialogue with the sound engine in charge of starting to play out the digital sound fragments, making it possible to control the way in which the digital sound fragments are played out so that the sound environment can undergo changes depending on the action taken by the user within the graphical interface corresponding to the HTML page itself, for example on the actions mentioned above.

[0044] Such a universal statement may be included in each HTML page. Preferably, however, a separate file will be created, written for example in Javascript, containing this universal statement, and only a line of code, for example Javascript, capable of causing the contents of said separate file to be run, will be included in each HTML page.

[0045] Other features and advantages of the invention will now be described with reference to embodiments illustrated by way of not-limiting example in the attached drawings, wherein:

- Figure 1 is a block diagram representing an embodiment of the method of digital sound management according to the present invention,
- Figure 2 shows schematically the runtime environment (files, variables and processes) of digital sound management according to Figure 1 in an electronic computer system,
- Figure 3 illustrates schematically an example of embodiment of a computer program for implementing the method of digital sound management according to the present invention,
- Figure 4 shows various different sound objects and graphical objects at runtime at various moments in time along a time line, in accordance with Figures 2 and 3,
- Figure 5 illustrates schematically a preferred method for creating a fragment of digital sound data intended to be played out cyclically, as a pad for providing an atmosphere,
- Figure 6 is a flow chart of a sound engine of the digital sound management according to the present invention, and

- Figure 7 is a flow chart of a function for random management of the playing out of digital sound fragments according to the present invention.

[0046] Figure 1 is a block diagram representative of a method for playing out digital sound in an electronic computer system and of a software structure 1 of a form of embodiment of a computer program comprising code for implementing said method, according to the present invention.

[0047] More specifically, Figure 1 shows the functional blocks that are made active in the electronic computer system in which digital sound management according to the invention is implemented, for example in the client computer upon linking up via the Internet to a web page that implements digital sound management according to the invention.

[0048] With 10, an initialisation block INIT is indicated as a whole. The initialisation process represented schematically by the INIT block 10 supervises the definition of some global variables.

[0049] In particular, the following are defined:

- a primary unit of time LOOP, expressed for example in milliseconds;

- a secondary unit of time CYCLE that is a pre-established multiple of the primary unit of time LOOP, expressed as an adimensional number.

1600 milliseconds is an exemplary value of the length of a primary unit of time LOOP. In the practice of the present invention, such value must be sufficiently low to prevent the start-up of the digital sound fragments at this rate from putting an excessive load on the CPU of the computer and for a digital sound fragment lasting for a time equal to 1 LOOP to make sense from a musical point of view; such value must also be sufficiently low for the size in bytes of a digital sound fragment lasting for a time equal to 1 LOOP to be so low to enable almost immediate (down)loading thereof, and to allow great flexibility in the creation of the complex sound passage. In practice, a range of values comprised between 500 milliseconds and 5000 milliseconds is preferred and a range of values comprised between 1000 milliseconds and 2500 milliseconds is even more preferable.

[0050] Thus, the time axis is split up into several different secondary units of time or cycles CYCLEs lasting for CYCLE*LOOP, each cycle being split up into CYCLE primary units of time LOOPS (see Figure 4, wherein CYCLE=4 LOOPs). In this way a "time grid" is created.

[0051] A practical example will serve the purpose of explaining the usefulness of this time grid according to the invention. Let us assume that the primary unit of time LOOP lasts for 1600 seconds and that the secondary unit of time CYCLE lasts for 8 LOOPS. Let us also assume that a fragment X lasting 1600 seconds has to be played out 6 times consecutively, and then repeated another 6 times consecutively after a delay of 3200 seconds, and again with the same structure. If the time axis were not split up into primary units, it would be necessary to store the following information:

- start playing out at time 0, repeat 6 times;
- wait for 3200 seconds;
- start playing out, repeat 6 times;
- wait for 3200 seconds;...

[0052] Or, should it be wished to store the points in time of starting to play out: 0, 1600, 3200, 4800, 6400, 8000, 12800, 14400, 16000, 17600, 19200, 20800,...

[0053] If a time grid according to the invention is created splitting up the time axis into primary units of time LOOPs only, the information to be associated with fragment X will have to be simply: START_POINTS=(0,1,2,3,4,5,8,9, 10,11,12,13,....) or, more conveniently in a Boolean format representing the primary units of time LOOPs when playing out will have to take place ("1") or not ("0"): START_POINTS=(1,1,1,1,1,1,0,0,1,1,1,1,1,1,0,0,...).

[0054] Then, if the time grid is further split up into secondary units of time CYCLEs, each equal, for example, to eight primary units of time (CYCLE=8 LOOPs), it will be sufficient to store the information START_POINTS=(0,1,2,3,4,5) or, more conveniently, in Boolean binary format, START_POINTS=(1,1,1,1,1,1,0,0).

[0055] The information referred to the primary units of time LOOPs during which a particular digital sound fragment has to be played is managed, conveniently, by including the digital sound fragment in a special set before the start of the first secondary unit of time CYCLE in which it has to start being played out, and removing it from that set before the start of the first secondary unit of time CYCLE in which it does not have to start being played out, and including it once again in said set before the start of the first secondary unit of time CYCLE in which it has to start being played out again, as illustrated more precisely below.

[0056] As an alternative, the information concerning the secondary units of time CYCLES during which a particular digital sound fragment has to be played out can be stored in the same way in which the LOOPs are stored, for example

using a sequence of CYCLES=(0,1,4,6,8,9,12,14,...) to indicate that the sub-sequence of primary units of time LOOPs indicated by the START_POINTS sequences illustrated above must be used during the first, the second, the fifth, the seventh and the ninth secondary units of time CYCLES et cetera of the overall passage.

[0057] This latter method of expression turns out to be convenient when the grouping of units of time is repeated. In other words, just as it is possible to group together several primary units LOOPS into secondary units of time CYCLES, it is equally possible to group together several secondary units of time CYCLES into tertiary units of time GROUPS and so on.

[0058] Thus, by establishing e.g. a tertiary unit of time GROUP=8 CYCLES, the above sequence of CYCLES= 0,1,4,6,8,9,12,14,...) may be expressed more conveniently as CYCLES=(0,1,4,6) or, in Boolean format, as CYCLES= (1,1,0,0,1,0,1,0), to indicate that the digital sound fragment must be played out in the first, in the second, in the fifth and in the seventh secondary units of time CYCLES of each tertiary unit of time GROUP.

[0059] Of course, this may be repeated in a similar manner by establishing units of time of a higher order.

[0060] The sound engine needs only to take care of controlling the various different points in time at which the various different digital sound fragments must start to be played out, in particular those fragments that are present in the above set each time; that is to say, it needs only to track the time on the basis of the primary units of time LOOPS and, if any, of the secondary units of time CYCLES and of the units of time of a higher order GROUPs, ...

[0061] Thanks to the time grid according to the invention, a more or less pyramid-like structure is provided, allowing for playing out the digital sound fragments with a lower repetition; this structure also leads to particularly effective management of the start of playing out the digital sound fragments by means of the sound engine, which will only have to take care of controlling the various different points in time at which the digital sound fragments have to start (START_POINTS).

[0062] For the sake of simplicity, reference will be made below mainly to the primary units of time LOOPS and to the secondary units of time CYCLEs only. Any skilled in the art will immediately recognise the changes to be made in order to implement units of time of a higher order.

[0063] In reducing the invention to practice, each pre-recorded digital sound fragment is produced (composed or adapted) in relation to the units of time and the complex passage desired or, vice versa, the units of time are defined on the basis of a pre-existing complex passage. For ease of explanation, the digital sound fragments are indicated below as SOUND_FRAGMENT_X, in which X represents a numerical value.

[0064] Each digital sound fragment SOUND_FRAGMENT_X may last for one or more primary units of time LOOPS and it may be subject to be repeated cyclically up to a number of times equal to a secondary unit of time, that is to say up to CYCLE times within a secondary unit of time CYCLE, so as to create continuity (even until it occupies a whole CYCLE, if required, in which case it will be repeated constantly). By defining the secondary unit of time CYCLE, it is possible to program the start of playing out each digital sound fragment SOUND_FRAGMENT_X playing it out at the time of any primary unit of time LOOP of a secondary unit of time CYCLE, and not necessarily at the time of the first one. On the time grid defined by the units of time LOOP, CYCLE, ..., it is possible to position various sound elements represented by respective digital sound fragments SOUND_FRAGMENT_X, so as to create a more or less complex musical structure, also including speech (indicated in this description as "complex passage"), in which each digital sound fragment SOUND_FRAGMENT_X has properties of volume, start, length, repetitions, panpotting etc., as described more precisely below.

[0065] Furthermore, in initialisation block INIT 10 the functions or processes that can be called up during actuation of the method according to an embodiment of the invention are preferably defined, such as a process for creating a sound object MAKE(SOUND_OBJECT_X), a process for removing a sound object KILL(SOUND_OBJECT_X), a process for resuming a sound object RESUME(SOUND_OBJECT_X), a process for random mixing of sound objects SCRAMBLE(), a process of synchronism with moving graphic elements SPIN(), a sound fading process FADER(), one or more processes of management of special effects FX_CONTROL(), all described in greater detail below.

[0066] In Figure 1, 11 indicates a engine (DOWN)LOAD_ENGINE for (down)loading the digital sound fragments SOUND_FRAGMENT_X and for creating sound objects SOUND_OBJECT_X. If digital sound management according to the present invention is implemented in a computer network, in particular the Internet, the block (DOWN) LOAD_ENGINE 11 supervises downloading of the digital sound fragments SOUND_FRAGMENT_X from the network, starting the download from the physical location of the web page (host computer) and checking its progress assiduously by comparing the size in KBytes of the digital sound file and the quantity of data already downloaded, in a manner that is well known per se.

[0067] The download engine (DOWN)LOAD_ENGINE 11 therefore includes a set of loading sub-blocks 121, 122, 123, 124, ... labelled as LOAD_SOUND_FRAGMENT_1, LOAD_SOUND_FRAGMENT_2, LOAD_SOUND_FRAGMENT_3, LOAD_SOUND_FRAGMENT_X... in Figure 1. Each loading sub-block 121, 122, 123, 124, ... is associated with a loading check sub-block 131, 132, 133, ... labelled as CHECK_LOAD_1, CHECK_LOAD_2, CHECK_LOAD_3, ... in Figure 1. It will be understood that the loading sub-blocks 121,... and the loading check sub-blocks 131,... may also be lacking, in which case checking of the process of downloading data from the web is taken

care of by the browser itself, in a manner that is well known per se.

[0068] Upon completion of loading of each digital sound fragment SOUND_FRAGMENT_X, as carried out by loading sub-blocks 121,... and as checked by the loading check sub-blocks 131,..., a respective process of creation of a sound object is carried out. The processes for creating sound objects are represented schematically in Figure 1 by the sub-blocks for creating sound objects 141, 142, 143, ... MAKE_SOUND_OBJECT_1, MAKE_SOUND_OBJECT_2, MAKE_SOUND_OBJECT_3,...

[0069] It will of course be understood that, as an alternative, the processes for creating sound objects carried out by the creating sub-blocks MAKE_SOUND_OBJECT_X 141,... can be carried out in the background, together with the processes of downloading of the digital sound fragments SOUND_FRAGMENT_X carried out by sub-blocks 121,...

[0070] In particular, the processes of creation of sound objects MAKE_SOUND_OBJECT_X 141,... associate the file containing the digital sound fragment SOUND_FRAGMENT_X with an instance, attributing thereto a name and a set of properties until the sound object SOUND_OBJECT_X is created.

[0071] With reference to Figure 2, which illustrates schematically the runtime environment (file, variables and processes) in the computer, and in particular in the client computer, in which the digital sound management according to the invention is implemented, each sound object SOUND_OBJECT 2 includes, in a preferred embodiment:

- an identifier ID 20 of the sound object;

- a name of or pointer 21 to the digital sound fragment SOUND_FRAGMENT_X, in particular to a file SOUND_FRAGMENT_X.SWF 5 (5a, 5b, 5c, 5d, ...) containing it;

- playing sequence data 22, in particular the start points START_POINTS 221, representing the primary units of time LOOPS of a secondary unit of time CYCLE at which playing out of the digital sound fragment SOUND_FRAGMENT_X has to start, and the number of times LOOPS (data 222) that the fragment of digital sound data SOUND_FRAGMENT_X has to be repeated consecutively each time it is played out;

- playing out data, including data 23 referred to the sound properties relating to playing out, in particular the (initial) sound volume value VOLUME 231, the (initial) balancing value PANPOTTING 232 (stereo arrangement with left/right prevalence or centred, expressed for example in numbers from -100 to +100, where 0 indicates centring - it is worthwhile stressing that the panpotting value does not necessarily entails that the digital sound fragments SOUND_FRAGMENT_X are of a stereophonic nature, since a monophonic digital sound file can in any case be played out with prevalence of one of the right or left channels), and data 24 referred to the level or channel of execution LEVEL in a multilevel playing environment, as is possible with most of the sound boards currently on the market, for example the SoundBlaster™ board by Technology, Ltd., Singapore;

- animation data 25 referred to static or animated graphics associated with the digital sound fragment SOUND_FRAGMENT_X, in particular the name 251 of an animation ANIMATION_Y and the numbers or other identifiers SPIN_POINTS 252 of key frames of the animation ANIMATION_Y that must be synchronised with the start of each primary unit of time LOOP of a secondary unit of time CYCLE.

[0072] For example, if, among the various different frames of an animation, three key frames are labelled with numbers 1, 2 and 3, the sequence SPIN_POINTS=(1,1,2,0,1,1,2,3) on a time grid in which the secondary unit of time CYCLE is equal to 8 LOOPs will indicate that, in one secondary unit of time CYCLE, the key frame of the animation labelled as 1 will have to be displayed at the start of the first, of the second, of the fifth and of the sixth primary unit of time LOOP, that the key frame of the animation labelled as 2 will have to be displayed at the start of the third and of the seventh primary unit of time LOOP, and that the key frame of the animation labelled as 3 will have to be displayed at the start of the eighth primary unit of time LOOP, while at the start of the fourth primary unit of time LOOP there is no forcing of the animation, which will, for example, continue in a linear manner or will not be displayed at all.

[0073] In this respect it is worthwhile stressing that, for example in the framework of the program Macromedia Flash (Player)™, the animations are structured by means of a series of frames that are run sequentially at a given speed expressed in frames per second (fps). Some of the frames are created explicitly and these are defined as key frames. If two successive key frames are linked, the program itself takes care of creating the intermediate frames between the two linked successive key frames, by means of a sort of linear interpolation. If displaying of a particular key frame is forced so that it will be in sync with a specific primary unit of time LOOP in the manner expressed by the SPIN_POINTS data 252, the linear or sequential nature of the animation may be altered. If the key frames labelled as 1, 2 and 3 are also linked to one another, either directly or through other key frames, then the SPIN_POINTS sequence indicated above may provide a continuous display of the graphics moving between the second and the third, between the sixth and the seventh, and between the seventh and the eighth primary units of time LOOP of a secondary unit of time

CYCLE, and between the eighth primary unit of time LOOP of a secondary unit of time CYCLE and the first primary unit of time LOOP of a subsequent secondary unit of time CYCLE, while the image will appear unconstrained by the sync with the sound between the other primary units of time LOOPs.

[0074] Going back to Figure 1, as mentioned above the objects created by the sub-blocks for creating objects MAKE_SOUND_OBJECT_X 141,... are stored in a set SOUND_OBJECTS 15, for example an array, for storing sound objects SOUND_OBJECT_X 2 and/or in a current set SOUND_OBJECTS_TO_PLAY 16, for example an array, of sound objects SOUND_OBJECT_X 2 referred to digital sound fragments being played out. Although providing the two arrays 15 and 16 is not strictly necessary, the advisability of doing so will be more clearly understood later.

[0075] It will be understood that the term array is used to indicate a data structure capable of including the data of each sound object SOUND_OBJECT_X 2, but said structure need not necessarily be in the form of a table. Other suitable structures such as stacks, several aggregated tables, pointers et cetera may be used.

[0076] A block MAIN_CONTROL 17 has the function of a main controller of management of the digital sound fragments SOUND_FRAGMENT_X and of the sound objects SOUND_OBJECT_X 2 associated with them. In particular, said block supervises the execution of the various different processes described above (MAKE, KILL, RESUME, SPIN, SCRAMBLE, FX_EFFECTS) or, in other words, supervises use of the functions defined in the initialisation block INIT 10.

[0077] It will be understood in this respect that the fact that the various functions mentioned above are defined in the initialisation block INIT 10 is for convenience only. Of course these functions could be defined while the digital sound management method or program is being run, if and as they become necessary.

[0078] The block MAIN_CONTROL 17 supervises management of the digital sound fragments SOUND_FRAGMENT_X as a whole as far as concerns playing them out according to the pre-established time grid, to form the desired complex passage, as well as supervises management of other sound and graphic elements. In other words, one or more of the following sub-blocks could be contained in the block MAIN_CONTROL 17.

[0079] First of all, a sub-block SEQUENCE_CONTROLLER 171 has the function of controller of the sequence of digital sound fragments to be played out or de-activated, removing them from the array SOUND_OBJECTS_TO_PLAY 16 by means of the KILL(SOUND_OBJECT_X) process and inserting them back into the array SOUND_OBJECTS_TO_PLAY 16 by means of the RESUME(SOUND_OBJECT_X) process, in particular taking them from the array SOUND_OBJECTS 15.

[0080] In this respect it should be noted that the processes of MAKE_SOUND_OBJECT_X 141,.. could be lacking and be replaced by MAKE(SOUND_OBJECT_X) processes in the block MAIN_CONTROL 17. However, the embodiment described above, that is to say the processes of MAKE_SOUND_OBJECT_X 141,.. outside the main controller MAIN_CONTROL 17, is advantageous since the provision of the two separate arrays 15 and 16 and control over the contents of the array SOUND_OBJECTS_TO_PLAY 16 by means of the main controller MAIN_CONTROL 17 enable a sound object for each digital sound fragment SOUND_FRAGMENT_X (down)loaded to be created once only, by means of the MAKE function. In other words, the KILL(SOUND_OBJECT_X) process does not fully eliminate the sound object SOUND_OBJECT_X, but merely removes it from the array SOUND_OBJECTS_TO_PLAY 16, leaving it in the array SOUND_OBJECTS 15. The process of resuming a sound object, RESUME(SOUND_OBJECT_X), must therefore simply reinsert the sound object SOUND_OBJECT_X into the array SOUND_OBJECTS_TO_PLAY 16, taking it from the array SOUND_OBJECTS 15. Even more particularly, it will then be understood that the array SOUND_OBJECTS_TO_PLAY 16 need not include the whole data structure of a sound object 2 described above with reference to Figure 2; instead, it may include merely pointers to the objects of the array SOUND_OBJECTS 15, as illustrated in Figure 2 itself, in particular their identifiers ID 20.

[0081] On the other hand, it will be understood that, while some digital sound fragments SOUND_FRAGMENT_X are intended to be repeated during performance of the complex sound passage, in which case their management by means of the KILL and RESUME processes in the manner described above is advantageous, other digital sound fragments SOUND_FRAGMENT_X are intended to be played out only once. Sound objects SOUND_OBJECT_X corresponding to such digital sound fragments SOUND_FRAGMENT_X intended to be played out only once may therefore be inserted into the array SOUND_OBJECTS_TO_PLAY 16 only, and not into the array SOUND_OBJECTS 15, by the processes of creation of sound objects MAKE_SOUND_OBJECT_X 141,... (and/or by the main controller MAIN_CONTROL 17), provided, of course, said array SOUND_OBJECTS_TO_PLAY 16 contains in this case all the data, that is to say all the data of the data structure described with reference to the array SOUND_OBJECTS 15 (Figure 2), of the sound objects 2 contained in it, or at least all such data with reference to the sound objects 2 intended to be played out once only.

[0082] As a further alternative, it is possible to insert all the sound objects SOUND_OBJECT_X 2 only into the array SOUND_OBJECTS_TO_PLAY 16 at the time of their creation by means of the processes of MAKE (SOUND_OBJECT_X), and to insert into the array SOUND_OBJECTS 15 only those sound objects SOUND_OBJECT_X 2 that are removed by means of a process of KILL(SOUND_OBJECT_X), but for which subsequent resumption by means of a process of RESUME(SOUND_OBJECT_X) is envisaged.

[0083] Thus, the process of removal of a sound object KILL(SOUND_OBJECT_X) causes a digital sound fragment

SOUND_FRAGMENT_X to stop being played out by removing the sound object SOUND_OBJECT_X 2 corresponding to it - or the pointer to said sound object, in particular its identifier ID 20 - from the array SOUND_OBJECTS_TO_PLAY 16. Playing out may, furthermore, be stopped immediately, or if said digital sound fragment SOUND_FRAGMENT_X is currently being played out, it may be allowed to finish. Since, as described more precisely below, playing out of the various different digital sound fragments SOUND_FRAGMENT_X is started at the start of each primary unit of time LOOP, in the event that playing out is not stopped immediately it will be stopped at the end of the relevant digital sound data or, in any case, it will not be started again at the start of subsequent primary units of time LOOP or, at the latest, it will end at the start of the next secondary unit of time CYCLE if the playing out that has already started is programmed to be repeated cyclically for a certain number of main primary units LOOP, as expressed by the datum LOOPS 222 of the sound object SOUND_OBJECT_X 2 associated with the digital sound fragment SOUND_FRAGMENT_X.

[0084] The process of RESUME(SOUND_OBJECT_X) establishes that playing out of a digital sound fragment SOUND_FRAGMENT_X will be resumed by inserting the sound object SOUND_OBJECT_X 2 corresponding to it - or the pointer to that sound object, in particular its identifier ID 20 - into the array SOUND_OBJECTS_TO_PLAY 16, similarly to the MAKE(SOUND_OBJECT_X) process.

[0085] During the KILL(SOUND_OBJECT_X) process, in the event that an animation ANIMATION_Y is associated with the sound object SOUND_OBJECT_X 2 by means of corresponding values of the animation data 25, it is possible, furthermore, to establish whether the graphics or animation ANIMATION_Y must be hidden when the sound ceases to be played out, or whether the graphics must remain visible but its animation must be stopped or, also, whether the graphics must be unconstrained by the sync with the primary units of time LOOPS, that is to say left free to follow the sequential execution of its animation. In this latter case, at the time of resumption of the same sound object SOUND_OBJECT_X 2 by means of a RESUME(SOUND_OBJECT_X) process, it will be possible to resume, or not, synchronism between playing out of the digital sound fragment SOUND_FRAGMENT_X and execution of the animation ANIMATION_Y.

[0086] Furthermore, the sequence controller SEQUENCE_CONTROLLER 171 may also provide a process of creation of a sound object MAKE(SOUND_OBJECT), for example if the process of removal of a sound object KILL (SOUND_OBJECT_X) completely removes the sound object, or in order to vary some data among the data 22-25 associated with a digital sound fragment SOUND_FRAGMENT_X.

[0087] The main controller MAIN_CONTROL 17 may furthermore include a random controller of the sequence of digital sound fragments SOUND_FRAGMENT_X to be played out, represented by block RANDOM_CONTROLLER 172. This controller manages the sequence of digital sound fragments SOUND_FRAGMENT_X in the same way as the sequence controller SEQUENCE_CONTROLLER 171 (that is to say by means of the KILL and RESUME processes), but in a random manner. For example, said random controller RANDOM_CONTROLLER 172 may be activated, starting from a certain point in time, thus replacing the sequence controller SEQUENCE_CONTROLLER 171, or it may act in parallel thereto. Preferably, with said random controller RANDOM_CONTROLLER 172 an action range is associated, either pre-established or tunable by the main controller MAIN_CONTROLLER 17,, so that it will be able, for example, to remove and/or insert, by means of the RESUME and KILL processes, up to a given number of sound objects from/into the array SOUND_OBJECTS_TO_PLAY 16 each time it is run, for example each time a secondary unit of time CYCLE is started.

[0088] Similarly to the sequence controller SEQUENCE_CONTROLLER 171, the random controller RANDOM_CONTROLLER 172 can also provide for a new process of creation of a sound object MAKE (SOUND_OBJECT) in order to vary some data among the data 22-25 associated with a digital sound fragment SOUND_FRAGMENT_X, regardless of whether the previous sound object SOUND_OBJECT_X associated with said digital sound fragment SOUND_FRAGMENT_X is removed or not.

[0089] The main controller MAIN_CONTROL 17 may furthermore include an event controller, represented by the block EVENT_CONTROLLER 173. Said controller manages events that can condition playing out of the digital sound fragments SOUND_FRAGMENT_X and the displaying of the animations ANIMATION_Y, such as events generated by the user interacting with the Graphical User Interface (GUI), such as for example a shift from the left channel to the right channel in response to a tracking device of the client computer (a mouse, for example) being moved from the left towards the right, to a specific key on the keyboard being pressed, to pre-established points in time or to other internal events of the client computer such as other applications being launched, the current window being moved or minimised, et cetera.

[0090] The main controller MAIN_CONTROL 17 may furthermore include a special effects controller, represented by block FX_CONTROLLER 174. Said special effects controller FX_CONTROLLER 174 manages, for example, fade-in, fade-out and cross-fade effects, effects of transition from one channel to another of a multi-channel playing system, et cetera.

[0091] The special effects controller FX_CONTROLLER 174 can, furthermore, act on the parameters for playing out the sound objects present in the array SOUND_OBJECTS_TO_PLAY 16, in particular on the data 23 concerning volume and panpotting.

[0092] It is worthwhile stressing that, in the practice of the present invention, it is advisable for the (down)load engine (DOWN)LOAD_ENGINE 11 to take care of loading the digital sound fragments SOUND_FRAGMENT_X consistently with the choices made in the main controller MAIN_CONTROL 17. In other words, the order of (down)loading of the digital sound files SOUND_FRAGMENT_X, for example from the network, and the time line of the playing out or time grid shall be suitably designed in such a way as to take the actual time required for (down)loading the files in any condition, and in particular at any speed of connection to the network, into account.

[0093] Besides the special effects controller FX_CONTROLLER 174 of the main controller MAIN_CONTROL 17, there may also be additional special effects controllers, external to the main controller MAIN_CONTROL 17, represented by blocks ADDITIONAL_FX_CONTROL 19. These controllers manage the playing out of other digital sound fragments SOUND_FRAGMENT_X or of other digital sound data unconstrained by the main performance and/or by the synchronism with animations, as sound effects or the playing of speech (spoken comments), effects programmed ad hoc for specific requirements concerning a performance, independent from the sound engine SOUND_ENGINE 19.

[0094] Such additional special effects controllers ADDITIONAL_FX_CONTROL 19 condition the sound objects SOUND_OBJECTS 2 contained in the array SOUND_OBJECTS_TO_PLAY 16 directly, for example by changing the level data 24, the volume data 231 and the panpotting data 232 thereof.

[0095] Actual playing out of the digital sound fragments SOUND_FRAGMENT_X corresponding to the sound objects 2 contained in the array SOUND_OBJECTS_TO_PLAY 16 at each point in time is managed by a sound engine represented by a block SOUND_ENGINE 19. The sound engine SOUND_ENGINE 19 manages solely the sound objects contained in the array SOUND_OBJECTS_TO_PLAY 16 at each point in time. The sound engine SOUND_ENGINE 19 manages more specifically the synchronism for starting to play out the digital sound fragments SOUND_FRAGMENT_X, to play which it resorts for example to other hardware and software components for managing the sound that are present in the client computer, in particular interacting with a sound board. Such interaction is shown schematically in Figure 1 by a block AUDIO_CONTROLLER 100 external to the software structure 1 of the embodiment of the present invention. The sound engine SOUND_ENGINE 19 takes the properties expressed by the data 22-25 associated with each digital sound fragment SOUND_FRAGMENT_X in a corresponding sound object SOUND_OBJECT_X 2 into consideration; these properties include the level 24 at which the fragment has to be played out in an application providing objects on more than one level such as, for example, the Macromedia Flash (Player)™ application referred to above, the sound volume 231, the panpotting level 232, synchronism with the units of time LOOPs and CYCLES and synchronism with the animations.

[0096] In practice, the sound engine SOUND_ENGINE 19 tracks the time on the basis of the units of time LOOPs and CYCLES, and starts to play out the digital sound fragments SOUND_FRAGMENT_X at the start of each primary unit of time LOOP (during each secondary unit of time CYCLE) in which each fragment has to be played out, as expressed by their respective data 221. Furthermore, if an animation ANIMATION_Y is associated with the digital sound fragment SOUND_FRAGMENT_X by means of the data 251 in the corresponding sound object SOUND_OBJECT_X 2, the sound engine SOUND_ENGINE 19 takes care of forcing displaying the key frame of the animation ANIMATION_Y, the label of which is indicated by the data 252 at each start of a LOOP.

[0097] As already stated elsewhere, in the practice of the present invention the primary unit of time LOOP must be small enough to allow maximum freedom of composition of the complex passage. It must be stressed in this respect that the sound engine SOUND_ENGINE 19 takes care of managing only the start of playing out of a digital sound fragment SOUND_FRAGMENT_X, the actual playing out of which, and any repetitions, are controlled subsequently directly by the appropriate software and hardware.

[0098] If, for example, the value of the secondary unit of time is CYCLE=4 LOOPS, playing out of a digital sound fragment SOUND_FRAGMENT_X the sound object of which, SOUND_OBJECT_X, has the sequence (1,0,1,0) as its START_POINT, will be started, in each cycle in which said sound object SOUND_OBJECT_X is present in the array SOUND_OBJECTS_TO_PLAY 16, at the start of the first primary unit of time LOOP and at the start of the third. If superimpositions of the corresponding digital sound fragment SOUND_FRAGMENT_X are not desired, its length in time may therefore be comprised in a range from less than 1 LOOP to 2 LOOPS.

[0099] Therefore, by providing a sufficiently small primary unit of time LOOP, of the order of thousands of milliseconds (for example LOOP=1600 msec), it is possible to create digital sound fragments SOUND_FRAGMENT_X so short as lasting exactly 1 LOOP or even shorter, that can thus be (down)loaded and therefore become available in the (client) computer almost immediately, still without precluding the possibility of inserting digital sound fragments SOUND_FRAGMENT_X longer than 1 LOOP, lasting preferably but not necessarily for multiples of 1 LOOP.

[0100] The shorter fragments will be intended, preferably, for being (down)loaded and therefore played out first, so as to provide sound immediately. The longer fragments will, on the other hand, be preferably intended for being (down) loaded and therefore played out at a later stage in the performance of the complex passage, when one or several fragments are already being played out and their (down)loading in the background will therefore be "invisible" to the user.

[0101] The fragments intended to be played out cyclically (value of the LOOPS data 222 of the sound object

SOUND_OBJECT_X 2 greater than zero) should last preferably for 1 LOOP exactly. A method of preparation applicable to certain digital sound fragments intended to be played out cyclically, for example as pads for providing an atmosphere, will be described further below, with reference to Figure 5.

[0102] In principle, the sound engine SOUND_ENGINE 19 should constantly check the length of a primary unit of time LOOP so that it will not lose its synchronism with the rate of the primary units of time LOOPs themselves. Since, as already stated above, the length of the primary unit of time LOOP is chosen preferably to be fairly short, in the practice of the present invention it turns sometimes out to be impossible to track sufficiently the length of a primary unit of time LOOP. Indeed, if LOOP= 1600 msec is chosen, for example, the sound engine SOUND_ENGINE 19 ought to check the passing of time every millisecond, counting up to 1600 in order to determine the start of a new primary unit of time LOOP. It may, however, turn out to be impossible to do this in computers equipped with low-performance CPUs or anyway due to overloading of the computer's CPU. It is thus found advantageous to implement the sound engine SOUND_ENGINE 19 in such a way that, so long as the end of the current primary unit of time LOOP is far away, it carries out the check at a low rate, that is to say at a relatively low first frequency f1, while towards the end of the current primary unit of time LOOP, on the other hand, it intensifies the frequency of the checks to a relatively high second frequency f2. This can be done in the manner described below with reference to Figure 6.

[0103] One advantageous measure in the practice of the present invention, based on the consideration set forth above that the sound engine SOUND_ENGINE 19 is responsible only for the start of the playing out of the sound objects contained in the array SOUND_OBJECTS_TO_PLAY 16, but not for checking their repetition, is to exploit as much as possible, in the playing sequence data 22, the repetition in loop fashion represented by the parameter LOOPS 222 in contrast to the sequence of primary units of time for starting playing out, START_POINTS data 221. If, for example, a digital sound fragment SOUND_FRAGMENT_X lasting for 1 LOOP has to be repeated continuously throughout the secondary unit of time lasting for CYCLE=4 LOOPS, the playing sequence data 22 associated with the corresponding sound object SOUND_OBJECT_X 2 could contain only the START_POINTS=(1,1,1,1) sequence; that is to say that the data 222 are not strictly necessary. Such a sequence means that the sound engine SOUND_ENGINE 19 must start playing out of the digital sound fragment SOUND_FRAGMENT_X each time a primary unit of time LOOP starts, i.e. four (CYCLE) times in one secondary unit of time CYCLE. This may have a negative impact on the performance levels in the event of a large number of digital sound fragments SOUND_FRAGMENT_X to be started for one and the same primary unit of time LOOP. Continuity of the playing out throughout the secondary unit of time lasting for CYCLE=4 LOOPS can be achieved more advantageously by associating the following playing sequence data 22 with the digital sound fragment SOUND_FRAGMENT_X within the corresponding sound object SOUND_OBJECT_X 2: data 221, START_POINTS=(1,0,0,0); data 222, LOOPS=4, where the LOOPS parameter indicates the number of times the digital sound fragment SOUND_FRAGMENT_X has to be repeated each time it starts to be played out; these repetitions are managed by the software/hardware responsible for managing the sound, but not by the sound engine SOUND_ENGINE 19.

[0104] These playing sequence data 22 correspond to the playing sequence data 22: data 221, START_POINTS= (1,1,1,1); data 222, LOOPS=1, and also to the playing sequence data 22: data 221, START_POINTS=(1,0,1,0); data 222, LOOPS=2. If they are expressed in these latter ways, however, management of the sound object SOUND_OBJECT_X 2 becomes more of a burden for the sound engine SOUND_ENGINE 19; in a practical performance, the sound engine is probably taking care of starting the playing out not only of the digital sound fragment SOUND_FRAGMENT_X concerned, but of several fragments at the same time. If the number of digital sound fragments that have to be started for each primary unit of time LOOP is large, there is therefore a risk that delays may be perceived on starting to play them out, depending on how powerful the computer CPU is. In other words, it is advisable not to load the sound engine SOUND_ENGINE 19 needlessly, especially considering that other processes may be active in the computer.

[0105] As stated above, the sound engine SOUND_ENGINE 19 also takes care of synchronising displaying of any graphic images or animations ANIMATION_Y with the start of the primary units of time LOOP, on the basis of the animation data 25 of the sound objects SOUND_OBJECT_X 2 present in the array SOUND_OBJECTS_TO_PLAY 16; that is to say forcing, at each primary unit of time LOOP, the display of the key frame of the animation ANIMATION_Y labelled, as indicated by the SPIN_POINTS data 252, to be displayed at said primary unit of time LOOP.

[0106] If, for example, the value of the secondary unit of time is CYCLE=4 LOOPS, while a digital sound fragment SOUND_FRAGMENT_X that has, in the corresponding sound object SOUND_OBJECT_X 2, the animation ANIMATION_Y associated with it in the data 251 and the SPIN_POINTS=(1,2,1,3) sequence associated with it in the data 252, is being played out, the following key frames will be displayed in each secondary unit of time CYCLE in which said sound object SOUND_OBJECT_X 2 is present in the array SOUND_OBJECTS_TO_PLAY 16: the key frame of the animation ANIMATION_Y labelled as 1 at the start of the first primary unit of time LOOP, the key frame of the animation ANIMATION_Y labelled as 2 at the start of the second primary unit of time LOOP, again the key frame of the animation ANIMATION_Y labelled as 1 at the start of the third primary unit of time LOOP and the key frame of the animation ANIMATION_Y labelled as 3 at the start of the fourth primary unit of time LOOP.

[0107] Of course, more than one sound object SOUND_OBJECT_X 2, each having different values of the other data 22-25, can be associated with each digital sound fragment SOUND_PRAGMENT_X.

[0108] As already stated, Figure 2 is a schematic illustration of the environment in the (client) computer at a given point in time while the digital sound management program according to a preferred embodiment of the invention is running, that is to say the files, variables and processes as a whole and the relationships among them.

[0109] With 31,the set 31 of the files (down)loaded to the (client) computer is shown. This set includes first and foremost a file that is representative of the Graphical User Interface within which the program is implemented, in particular web page *Page.html* 6. Said set 31 also includes the files of the digital sound fragments *SoundFragment1.swf* 5a, *SoundFragment2.swf* 5b, *SoundFragment3.swf* 5c, *SoundFragment4.swf* 5d,..., and the files of the animations *AnimationA.swf* 253a, *AnimationB.swf* 253b.

[0110] It is worthwhile stressing that the extension *.swf,* which is typical of MacroMedia Flash (Player)™ files, is only used here by way of example, and should not be construed as a limitation to this invention, which can be applied to various different sound management programs and plug-ins, as mentioned above.

[0111] The set 31 also includes a further file indicated as *Main.swf* 7. As better illustrated below with reference to Figure 3, said file, also not necessarily a MacroMedia Flash (Player)™ file, includes, in a particularly preferred embodiment, the programming code of all the processes defined above, that is to say of the initialisation process INIT 10, of the (down)load engine (DOWN)LOAD_ENGINE 11, of the various different processes of the main controller MAIN_CONTROL 17, of the sound engine SOUND_ENGINE 19 and of the additional special effects controllers ADDITIONAL_FX_CONTROL 18, as well as of the necessary data structure, in particular, for creating the arrays SOUND_OBJECTS 15 and SOUND_OBJECTS_TO_PLAY 16. For greater clarity, the relationships between such processes and the code contained in the file *Main.swf* 7 is not shown in Figure 2.

[0112] The variables of the software environment of the (client) computer include first of all the array SOUND_OBJECTS 15, already described above, and the array SOUND_OBJECTS_TO_PLAY 16. This latter array is shown more specifically as a solid line at a first moment in time, and as a dotted line in two other different moments in time (blocks 16a and 16b). These three moments in time correspond to specific moments of an example that will be described subsequently with combined reference to Figures Figures 2 to 4.

[0113] The variables of the software environment of the client computer also include several global variables 41, defined in the initialisation block INIT 10, in particular the length of the primary unit of time LOOP and that of the secondary unit of time CYCLE, in the case illustrated. In the case of a time grid providing a single primary unit of time only, of course there will be only the LOOP variable, while in the case of a composite grid there will be further variables expressing the lengths of the tertiary unit of time GROUP, et cetera.

[0114] In Figure 2, the various different processes listed above are labelled with the same reference numbers used in Figure 1 and their relationships with the variables, that is to say the specific variables on which each process acts, is illustrated schematically by a thick arrow.

[0115] Figure 2 is now described in greater detail in combination with Figure 3, that illustrates schematically an example of embodiment of the method for managing digital sound files according to the present invention implemented using the MacroMedia Flash (Player)™ software. Examples will furthermore be provided of the programming of the various different functional blocks and processes, using the proprietary ActionScripting™ programming language of that software. Said language is, moreover, very similar to JavaScript language, which is so widespread in the Internet environment that it has now become a de facto standard.

[0116] Although, as stated, said software is extremely widespread, it is felt advisable to illustrate briefly some of its basic concepts.

[0117] MacroMedia Flash™ is an object-oriented programming tool particularly suitable for the management of audio-visual communications, that is to say of sound and animated graphics. It is based on a timeline TimeLine, and its design is therefore basically linear, although more than one independent time line can be launched in parallel. It provides advanced possibilities of improving performance by programming in the proprietary ActionScripting™ language.

[0118] The programming environment provides for creation of files with the extension .FLA, that are then compiled into files with the extension .SWF. Inside the compiled files, that is to say files that can be run directly by the MacroMedia Flash (Player)™ software, the source code and the digital sound data contained inside them, as will be stated immediately below, are protected (that is to say they are not visible without decompiling or reverse engineering): this is yet another advantageous aspect of this implementation of the present invention.

[0119] Each .FLA or .SWF file has two separate sections. A first section consists of a library of sounds (SoundLibrary), that is to say it can contain one or more sound passages (digital sound fragments in the case of the invention), typically non-compressed passages having the extension .WAV in the .FLA file, that are then exported in the form of MP3-type compressed passages to the .SWF file. A second section is constituted by the aforesaid time line (TimeLine). The time line is structured in a series of frames, of which some are key frames (KeyFrames), and can be labelled so that reference can be made to them. It is possible to insert a particular frame of an animation, sound data and/or code in ActionScripting™ language into each key frame.

[0120] When the execution of the TimeLine of the .swf file reaches a key frame, the image stored in it is displayed on the screen and/or the code contained in it, that can contain the launch of a sound contained in the SoundLibrary, is run.

[0121] The contents of .swf files can be shared with other .swf files, setting up a specific sharing property and attributing sharing names to these files. In this way, an .swf file can access images, sounds and code portions stored in another shared .swf file, and can cause another .swf file to be run.

[0122] The key frames can stop the timing of the time line from running (stop key frames). The key frames can start display of images, sounds being played or other .swf files being run, on a particular layer or level, to which reference is made by means of a number.

[0123] When a second element (sound, image or code) is started (played, displayed or made to run) using the same level number on which a first element had been started, the first element stops being played, displayed or run. However, until the same level number is used again for a second element, the first element continues to exist. In this way it is possible to have more than one element (animations, sounds or code) running at the same time on several different levels and, in particular, it is possible to create key frames that cause "stable" MovieClips to be run. This expression is used to indicate another .swf file, or another element of the same .swf file having its own time line, that can be controlled by means of a code or by means of the key frames of the main time line, that is to say of the time line of the .swf file that made it start to run.

[0124] As far as concerns management of the graphics, pairs of key frames of an animation can be interpolated by the program to display the graphics in the frames intervening between the key frames of the pair. For example, to move a small image such as a star from a position X,Y on the screen to a position X',Y', it is possible to insert a first key frame containing the image of the star in the position X,Y and a second key frame containing the image of the star in the position X',Y' and to connect the two key frames. At runtime, between the first key frame and the second key frame, the image of the star is displayed in a series of intermediate positions, thus achieving an almost continuous movement of the star with only two key frames. A similar sort of interpolation takes place as far as concerns changes in size or colour of an image.

[0125] Figure 3 illustrates schematically the structure of the various different files in an example of embodiment of digital sound management according to the present invention by means of Macromedia Flash (Player)™. Said Figure also shows the software environment existing in the computer (data available in the server and downloaded to the client) in the case of implementation on the Internet. Said Figure is based on an example of embodiment that is congruent with Figure 2 already discussed and with Figure 4, which shows the various different digital sound fragments SOUND_FRAGMENT_X and the various different key frames of animations ANIMATION_Y being run in various different primary units of time LOOPS along the time line; the time grid according to the invention is indicated schematically on the time line.

[0126] In Figure 3, with 5a, 5b, ..., 5n a certain number of files SOUND_FRAGMENT_X.SWF are shown, corresponding to the digital sound fragments SOUND_FRAGMENT_X to be played out during the performance of the complex passage associated with the web page indicated as *Page.html* 6.

[0127] Each of these files SOUND_FRAGMENT_X.SWF 5a, 5b,...,5n includes more specifically one sound fragment .WAV (FRAGMENT_X_SOUND_DATA), previously prepared in the manner described below, in its sound library SoundLibrary 51, while it contains nothing in its own time line TimeLine 52. A sharing name is assigned to each of the files SOUND_FRAGMENT_X.SWF, so that its elements and in particular its sounds FRAGMENT_X_SOUND_DATA can be accessed by other .swf files.

[0128] The sound fragments .WAV are prepared in such a way as to optimise the quality-data weight ratio of the digital sound. It is possible, for example, to use compressed formats such as the MP3 format at a certain number of Kbps (Kbit per second). Generally speaking, 80 Kbps is a good compromise, but more acute tones may require a higher Kbps rate, while lower tones may be reproduced well at a lower definition.

[0129] In any case, the files SOUND_FRAGMENT_X.SWF 5 structured as stated above are extremely small in size.

[0130] In addition to the digital sound files SOUND_FRAGMENT_X.SWF 5, the file *Main.swf* 7, already discussed with reference to Figure 2, is illustrated in Figure 3. The file *Main.swf* 7 is embedded as an object inside the BODY of page *Page.html* 6, for example with the following HTML code, that is supported by both the Internet Explorer™ browser by Microsoft Corporation of Redmond, WA, U.S.A., and by the Netscape Navigator™ browser by Netscape Communications Corporation, Mountain View, California, U.S.A.

```
<BODY >
.....
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=5,0,0,0" WIDTH=640
HEIGHT=480 id="UM">
 <PARAM NAME=movie VALUE="Main.swf"> <PARAM NAME=quality VALUE=high>
 <EMBED src="Main.swf" quality=high bgcolor=#B3B3B3 WIDTH=640 HEIGHT=480 TYPE="application/x-shockwave-
flash"
PLUGINSPAGE="http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFlash"
name="UM"></EMBED>
</OBJECT>
</BODY>
```

[0131] More specifically, the file *Main.swf* 7 illustrated here does not contain anything in its sound library SoundLibrary 71.

[0132] The time line TimeLine 72 of the file *Main.swf* 7 contains a plurality of key frames numbered as *KeyFrame_n* in Figure 3.

[0133] The first key frame of TimeLine 72 of the file *Main.swf* 7, KeyFrame_1, contains a portion of ActionScripting™ code implementing the functional blocks illustrated below.

- Initialisation block INIT 10, implemented for example by means of the following code that, like those following it, is felt to be sufficiently commented.

```
function hinit() {
loop = 1600; // loop base duration (milliseconds)
cycle = 4; // global cycle config
totcycles = 0;
// about sound
soundtot = 0;
_soundbuftime = 0;
sound_objects_to_play = new Array();
loops = new Array();
starts = new Array();
sound_objects = new Array();
remove = new Array();
// about anims
anims = new Array();
spins = new Array();



firstgo = new Array();
}
hinit();
```

- MAKE(SOUND_OBJECT) function, implementing an embodiment of functional blocks 141, 142, 143:

```
function make (zsound, zsequence, zloops, zanim, zspins, store) {
        sound_objects_to_play.push(zsound);
        starts.push(zsequence);
        loops.push(zloops);
        anims.push(zanim);
        zanim._visible = true;
        spins.push(zspins);
        howmany = sound_objects_to_play.length;
        if (store == 1) {
                temp = new Array(zsound, zsequence, zloops, zanim, zspins);
                sound_objects.push(temp) // stores all specifications with sound
        }
}
```

[0134] It will be understood that it is planned here to always create the sound object SOUND_OBJECT_X 2 in the

array SOUND_OBJECTS_TO_PLAY 16, and to create a complete copy of it in the array SOUND_OBJECTS 15 if the parameter *store* passed on to the function takes on a Boolean value of "1". As explained above with reference to Figure 1, this is one of the various possible equivalent implementation choices.

- KILL(SOUND_OBJECT_X) function, called up by the main controller MAIN_CONTROL 17, in particular by the sequence controller SEQUENCE_CONTROLLER 171 and by the random sequence controller RANDOM_CONTROLLER 172:

```
// SOUND KILL (whichSound, stopSound y/n, setfreeAnim y/n, hideAnim y/n)
function kill (zsound, andstop, andfree, andhide) {
        for (s=0; s< sound_objects_to_play.length; s++) {
                if (sound_objects_to_play[s] == zsound) {
                        // remove from arrays
                        sound_objects_to_play.splice(s,1);
                        starts.splice(s,1);
                        loops.splice(s,1);
                        if (andfree) {anims[s].gotoAndPlay("stasi")}
                        if (andhide) {anims[s]._visible = false}
                        anims.splice(s,1);
                        spins.splice(s,1);
                        firstgo.splice(s,1);
                }
        }
        if (andstop == 1) {zsound.stop()}
}
```

[0135] Consistently with the choice made in implementing the MAKE function, the KILL function and the subsequent RESUME function only manage the array SOUND_OBJECTS_TO_PLAY 16.

- RESUME(SOUND_OBJECT_X) function, also called up by the main controller MAIN_CONTROL 17, in particular by the sequence controller SEQUENCE_CONTROLLER 171 and by the random sequence controller RANDOM_CONTROLLER 172:

```
// SOUND RESUME, [num] or [name] (MAKE again a sound with its original specs)
function resume(zsound) {
        if (isNaN(zsound)) {
                for (r=0; r<sound_objects.length; r++) {
                        stock = sound_objects[r];
                        if (stock[0] == zsound) {
                                make(stock[0],stock[1],stock[2],stock[3],stock[4]);
                        }
                }
        } else {
                stock = sound_objects[zsound]
                make(stock[0],stock[1],stock[2],stock[3],stock[4]);
        }
}
```

- SCRAMBLE function, again called up by the RANDOM_CONTROLLER 172, and which in turn uses the functions KILL(SOUND_OBJECT_X) and RESUME(SOUND_OBJECT_X):

```
// SCRAMBLE tot sound_objects (kill some, resume others)
function scramble(tot) {
        hit = 0;
        out = 0;
        while (out < tot) {       // kill tot
                aim = int(random(sound_objects_to_play.length - 0.0001));
                kill (sound_objects_to_play[aim], 0, 1, 1);
                out++
        }
        var tries;
        tries = 0;
        while (hit < tot) {       // resume tot
                        var double;
                        double = false;  // also to prevent narrowloopin':
                        aim = int(random(sound_objects.length - 0.0001));
                        sneak = sound_objects[aim];
                        for(u=0; u< sound_objects_to_play.length; u++) {
                                if (sneak[0] == sound_objects_to_play[u]) {
                                        double = true;
                                        break;
                                }
                        }
                        // EXCEPTIONS (incompatibilities)
                        for(u=0; u< sound_objects_to_play.length; u++) {
                                //// list exceptions here
                                if ((sneak[0] == groove1) && (sound_objects_to_play[u] ==
piano1) || (sneak[0] == piano1) && (sound_objects_to_play[u] == groove1)) {
                                        double = true
                                }
                                if ((sneak[0] == pianotheme1) && (sound_objects_to_play[u]
== piano1) || (sneak[0] == piano1) && (sound_objects_to_play[u] == pianotheme1)) {
                                        double = true
                                }


                         }
                         if (double == false) {    // no DOUBLEs ? OK, launch!
                                 resume(sneak[0]);
                                 hit++;
                         }
         tries++         // anti-larsen (n tries)
         if (tries >= sound_objects.length - sound_objects_to_play.length) {hit++; tries = 0}
         }
 }
```

[0136] A flow chart of the SCRAMBLE function is illustrated in Figure 7. In a first block 701, removal of a pre-established number of sound objects SOUND_OBJECT_X 2, randomly chosen, from the array SOUND_OBJECTS_TO_PLAY 16, is caused.

[0137] In a second block 702, an attempt is made to restore a second number of sound objects SOUND_OBJECT_X 2, randomly chosen, from the array SOUND_OBJECTS 15 to the array SOUND_OBJECTS_TO_PLAY 16. Since it is possible, when randomly choosing the sound objects SOUND_OBJECT_X 2 , that they have already been restored from said array SOUND_OBJECTS_TO_PLAY 16, a check is made in a block 703 to determine whether this has occurred. In block 703 it is also possible to provide a step of incompatibility checking for some digital sound fragments that do not have to be removed. In the event of a negative result (exit NO from block 703) resumption takes place. In the event of a positive result (exit Yes from block 703), a check is made in a block 704 to see whether a counter of resumption attempts carried out at block 702 has been exceeded. In the event of a negative result (exit NO from block 704) block 702 is returned to for a further resumption attempt. In the event of a positive result (exit Yes from block 704), execution of the SCRAMBLE function ends in an output block 705 so as to avoid entering a critical loop.

- SPIN function, for managing animations, called up by the sound engine SOUND_ENGINE 19:

```
// animations 'spin' engine
function spin (slot) {
        for (a=0; a<anims.length; a++) {
                started = starts[a];      // 'first time' routine (don't spin until sound start)
                if (started[slot] == 1) {


                                        firstgo[a] = true;
                                }
                                tospin = spins[a];        // spin animation !
                                if (tospin[slot] == 0) {
                                        continue;
                                } else if (firstgo[a] == true) {
                                        anims[a].gotoAndPlay("spin"+tospin[slot]);
                                }
                        }
                }
```

[0138] The second key frame KeyFrame_2 shown in Figure 3 is in charge of (down)loading a first file SOUND_FRAGMENT_1.SWF containing a digital sound fragment FRAGMENT_1_SOUND_DATA and of checking the (down)loading, that is to say it represents the functional blocks LOAD_SOUND_FRAGMENT_1 121 and CHECK LOAD_1 131 of Figure 1. This is a stop key frame.

```
onClipEvent (load) {  //when the MovieClip is entered for the first time
        loadMovieNum ("soundfragment1.swf", 1); //name of file, level number
        done = false;
        _root.stop();    //the main timeline is stopped
}
onClipEvent (enterFrame) {  //each time the MovieClip is entered
        if (done == false) {
                progress =
Math.floor(_level20.getBytesLoaded()/_level20.getBytesTotal()*100);   //Checks the download
                _root.progress = progress;
                if (progress == 100) {
                        done = true;
                        _root.play();    //the main timeline is resumed
                }
        }
}
```

[0139] The third key frame KeyFrame_3 in Figure 3 implements the process of creation of a sound object MAKE (SOUND_OBJECT_1) associated with the digital sound fragment SOUND_FRAGMENT_1 that has just been downloaded, that is to say that it implements block 141 of Figure 1:

```
SOUND_OBJECT_1 = new Sound(_level1);
// "soundfragment1" is export name for the sound in soundfragment1.swf library
SOUND_OBJECT_1.attachSound("soundfragment1");

function give_ SOUND_OBJECT_X1() { // include in function for future reference
// define initial specs
SOUND_OBJECT_X1.setVolume(65);
SOUND_OBJECT_X1.setPan(0);
// the sound
zsound = SOUND_OBJECT_1;
// set start positions for this sound (array length=cycle)
zsequence = new Array(1,0,0,0); //START_POINTS
zloops = 4; // set loops for this sound
zanim = _root.AnimationA;      // set animation instance
//zanim=_level0.AnimationA; in case the sound engine is imported in another .swf
zspins = new Array(1,2,3,3);    // set spin points (markers)
```

```
// pass all to MAKE function (last value is STORE y/n, which means save for later use into
sound_objects array)
make(zsound, zsequence, zloops, zanim, zspins, 1);
}

give_ SOUND_OBJECT_X1();   // recall it now, of course
```

[0140] In the example of embodiment in Figures 2-4, the sound object 2 SOUND_OBJECT_1 associated with the first digital sound fragment SOUND_FRAGMENT_1 has the following values of the various different data (see the corresponding line in the array SOUND_OBJECTS 15 in Figure 2):

- data 20, ID=SO1, corresponding to the parameter *zsound* in the code given above;
- data 221, START_POINTS=(1,0,0,0), corresponding to the parameter *zsequence* in the code given above, and data 222, LOOPS=4, corresponding to the parameter *zloops* in the code given above; these data 22 as a whole indicate that the digital sound fragment SOUND_FRAGMENT_1 must be repeated cyclically throughout the secondary unit of time CYCLE;
- data 24, LEVEL=1, corresponding to the parameter passed to the first function call, *newSound()*, in the code given above;
- data 231 and 232, VOLUME=65 and PANPOTTING=0, corresponding to the settings for the properties *.setVolume ()* and *.setPan()* in the code given above;
- data 251, ANIMATION_NAME=AnimationA, pointer to or identifier of the file AnimationA.swf, corresponding to the parameter *zanim* in the code given above;
- data 252, SPIN_POINTS=(1,2,3,4), corresponding to the parameter *zpins* in the code given above.

[0141] The code given above envisages one last parameter of the function MAKE(SOUND_OBJECT_X). As already explained above, this parameter *store* is a Boolean parameter that, if it takes on a first Boolean value, the value "1", for example, indicates that the sound object SOUND_OBJECT_X 2 must be inserted into the array SOUND_OBJECTS 15 so as to be able to use it again by means of a process of RESTORE(SOUND_OBJECT_X) subsequently to a process of KILL(SOUND_OBJECT_X); if, on the other hand, it takes on a second Boolean value, the value "0", for example, it indicates that the sound object SOUND_OBJECT_X 2 does not have to be inserted into the array SOUND_OBJECTS 15, but only into the array SOUND_OBJECTS_TO_PLAY 16: thus, following a process of KILL (SOUND_OBJECT_X) it may not be restored by means of a process of RESTORE(SOUND_OBJECT_X) and will instead have to be (down)loaded once again by means of a respective process of LOAD(SOUND_OBJECT_X) or created again by means of a respective process of MAKE(SOUND_OBJECT_X).

[0142] The key frame KeyFrame_4 in Figure 3 immediately starts playing the digital sound fragment that has just been (down)loaded and inserted into the array SOUNDS_OBJECTS_TO_PLAY 16. In other words, said key frame is a "stable" key frame that implements the sound engine SOUND_ENGINE 19:

```
onClipEvent (load) {
        function hinit() {
                _root.totcycles = 0;     // reset tot cycles?
                count = 0;
                loop = _root.loop;       // time (milliseconds)
                freeze = -999999;        // set VERY low to start sound immediately (see after)
                edge = 1.2      // how much to get beyond prediction range: 1.0 = no
        }
        hinit();
}
onClipEvent (enterFrame) {
        timer = getTimer();
        mspf = timer - former;  // ms per (former) frame
        former = timer;
        ms = timer - freeze;    // CORE: ms from loop start
        if (ms + (mspf * edge) >= loop) {       // next frame prediction...
                // sound schedule extraction (look forward and reduce delays)
                picked = new Array();
                p_loops = new Array();
                for (pos=0; pos<_root.sound_objects_to_play.length; pos++) {
                        tostart = _root.starts[pos];
                        if (tostart[count] == 0) {
                                continue;
                        } else {
                                picked.push(_root.sound_objects_to_play[pos]);
                                p_loops.push(_root.loops[pos]);



                        }
                }
                while (ms<loop) {       // checks end of loop at high frequency
                        ms = getTimer()-freeze;
                }
                for (pos=0; pos<picked.length; pos++) {
                        picked[pos].start(0, p_loops[pos]);      // SOUNDS LAUNCH
                }
                // reset
                freeze = getTimer();
                former = freeze;
                _root.spin(count);      // spin animations
                count++;        // count cycles (ex. on 4: 0,1,2,3,0,1,2,3,...)
                if (count>_root.cycle-1) {
                        count = 0;
                }
                _root.totcycles++;      // total count
        }
}
```

[0143] A flow chart of this specific embodiment of the sound engine SOUND_OBJECT 19 is now described with reference to Figure 6.

[0144] A first block 601 is run each time the frame containing the code implementing the sound engine SOUND_ENGINE 19 is entered, that is to say at a first frequency f1 expressed in frames per second (since the sound engine is in a stable key frame). The first frequency f1 is relatively low, for example 15 times per second (15 fps).

[0145] In block 601, the time MS lapsed since the start of the current primary unit of time LOOP is calculated. In a second block 602, a comparison is made between the time that is estimated will have passed when exiting the current frame and the value equivalent to the primary unit of time LOOP (for example, 1600 msec), as declared in the process of INIT 11. The estimate is made by adding the time MS that has passed since the start of the current primary unit of time LOOP to the time MSPF that has passed while running the previous frame, up-dated in a block 607.

[0146] Preferably, in the above comparison, the time MSPF that has passed during the running of the previous frame is multiplied by a safety threshold value EDGE, set for example at a value of 1,2.

[0147] In practice, a check is carried out in block 602 to see whether, at the next comparison made at the relatively low first frequency f1, the value of the primary unit of time LOOP would be exceeded.

[0148] In the event of a negative result (exit NO from block 602), block 607 is run to up-date the variable MSPF to the time that has passed during the current frame and, in the subsequent frame (block 608), block 601 is run again.

[0149] In the event of a positive result (exit YES from block 602), a block 603 in which the time MS that has passed since the start of the current primary unit of time LOOP is calculated again, is run. In a block 604 a check is carried out to see whether said time MS is greater than or equal to the value equivalent to the primary unit of time LOOP. As long as the time MS is shorter than the LOOP value (exit NO from block 604), blocks 603 and 604 are repeated. This check is therefore carried out during the current running of a frame, at a relatively high frequency f2, equal to the time required to carry out the instructions of blocks 603, 604.

[0150] When the time MS is longer than or equal to the LOOP value (exit YES from block 604), the playing out of the digital sound fragments SOUND_FRAGMENT_X envisaged for the current primary unit of time LOOP, as well as displaying of any animation frames associated with them, are started in a block 605. In particular, reference is made in said block 605 to the array SOUND_OBJECTS_TO_PLAY 16.

[0151] After block 605 has been run, the value of the time MS that has passed since the start of the current primary unit of time LOOP is reset in block 606, since the next run of block 601 will be related to a new primary unit of time LOOP. Then block 607 is run again to up-date the time MSPF that has passed while running the frame, and block 608 for passing on to the next frame.

[0152] The above calculation of the duration of each primary unit of time LOOP on the basis of the actual time taken for each frame (MSPF) is a democratic concept that considers the different performance levels conditioned by the power of the computer's CPU: lower performance will probably mean a larger and/or more irregular value of MSPF, especially if other operations that load down the CPU are being carried out. By adding the MSPF value actually employed to run the previous frame, possibly multiplied by the safety threshold EDGE, each time the comparison is made, the check to find out whether the primary unit of time LOOP has been exceeded in the current frame is actually proportional to the instantaneous performance of the CPU.

[0153] In a manner corresponding to the situation described above, Figure 4 illustrates, for some secondary units of time CYCLES of the time line, the objects (digital sound fragments SOUND_FRAGMENT_X and key frames of animations ANIMATION_Y), the playing out and displaying of which is started at the start of the primary units of time (LOOP). For the sake of brevity, in Figure 4 the digital sound fragments SOUND_FRAGMENT_X are represented by their identifier number X and the key frames of an animation ANIMATION_Y are represented by the letter Y identifying the animation and by the label of the key frames.

[0154] Thus, within the first secondary unit of time CYCLE indicated explicitly as C4 in Figure 4, corresponding to running of the key frame KeyFrame_4 the digital sound fragment SOUND_FRAGMENT_1 and the key frame of the animation AnimationA labelled as 1 are started at the start of the first primary unit of time LOOP L1, the digital sound fragment SOUND_FRAGMENT_1 and the key frame of the animation AnimationA labelled as 2 are started at the start of the second primary unit of time LOOP L2, the digital sound fragment SOUND_FRAGMENT_1 and the key frame of the animation AnimationA labelled as 3 are started at the start of the third primary unit of time LOOP L3, and the digital sound fragment SOUND_FRAGMENT_1 and again the key frame of the animation AnimationA labelled as 3 are started at the start of the fourth and last primary unit of time LOOP L4. It is also worthwhile stressing that, although the digital sound fragment SOUND_FRAGMENT_1 is being played out during all four primary units of time LOOPs L1-L4, the sound engine SOUND_ENGINE 19 only starts to play it out at the start of the first primary unit of time LOOP L1; during the next three primary units of time LOOPS L2-L4, on the other hand, playing out is controlled directly by the process for playing out the digital sound fragment SOUND_FRAGMENT_1, exploiting the self-repetition property represented by its data 222, LOOPS=4. In this example, therefore, it is assumed that the digital sound fragment SOUND_FRAGMENT_1 will last for a time equal to one primary unit of time LOOP.

[0155] Key frames KeyFrame_5 and KeyFrame_6 are similar to key frames KeyFrame_2 and KeyFrame_3, but they refer to a second digital sound fragment SoundFragment2.swf that is associated with a second sound object SOUND_OBJECT_X2, therefore implementing the functional blocks 122, 132, 142 of Figure 1.

[0156] The second secondary unit of time CYCLE shown explicitly as C2 in Figure 4 corresponds to the situation existing when running of the key frame KeyFrame_6 has been completed. In addition to the objects being run described above with reference to the first secondary unit of time CYCLE C1, digital sound fragment SOUND_FRAGMENT_2 also starts to be played out in said second secondary unit of time CYCLE C2, at the start of the second and of the third primary units of time LOOPs L2, L3, in accordance with the contents of the data 22 of the object SOUND_OBJECT_2 shown in Figure 2: data 221 SPIN_POINTS=(0,1,1,0) and data 222 LOOPS=1.

[0157] Similarly, there will be other key frames that take care of downloading the other digital sound fragments (that is to say implementing the functional blocks 123, 124,..., 133,...,143,...).

[0158] In the example of application shown in Figure 3, to illustrate said sequence a key frame KeyFrame_i-1 for creating the sound object SOUND_OBJECT_X4, associated with the digital sound fragment SoundFragment4.swf, is shown.

[0159] At the point in time corresponding to the end of the run of the key frame KeyFrame_i-1, the array SOUND_OBJECTS 15 and the array SOUND_OBJECTS_TO_PLAY 16 will therefore have the contents illustrated in Figure 2. More specifically, the array SOUND_OBJECTS 15 contains four sound objects SOUND_OBJECT_1,

SOUND_OBJECT_2, SOUND_OBJECT_3 and SOUND_OBJECT_4, each having some values of its respective data 20-25, and the array SOUND_OBJECTS_TO_PLAY 16 contains pointers to all said sound objects SOUND_OBJECT_1, SOUND_OBJECT_2, SOUND_OBJECT_3 and SOUND_OBJECT_4. The pointers are indicated in Figure 2 by the names of the respective identifiers ID 20: SO1, SO2, SO3, SO4.

[0160] Correspondingly, in addition to the objects described above with reference to the second secondary unit of time CYCLE C2, in the third secondary unit of time CYCLE shown explicitly as C3 in Figure 4, playing out of the digital sound fragment SOUND_FRAGMENT_3 is also started at the start of the first and of the second primary units of time LOOPS L1 and L2, playing out of the digital sound fragment SOUND_FRAGMENT_4 is started at the start of the second, third and fourth primary units of time LOOPs L2, L3, L4, and displaying of the frames labelled with 1, 1, 2 and 3, respectively, of the animation ANIMATION_B is started or forced at the start of the four primary units of time LOOP L1-L4.

[0161] Similarly to the situation described for the first digital sound fragment SOUND_FRAGMENT_1, the sound engine SOUND_ENGINE 19 starts to play out the third and the fourth digital sound fragments SOUND_FRAGMENT_3 and SOUND_FRAGMENT_4 (assumed to last for a time equal to one primary unit of time LOOP) only once within a secondary unit of time CYCLE, exploiting the self-repeating property LOOPS 222 of their respective sound objects 2 SOUND_OBJECT_3 and SOUND_OBJECT_4; the second digital sound fragment SOUND_FRAGMENT_2 (whatever its length), on the other hand, is started to be played out twice by the sound engine SOUND_ENGINE 19.

[0162] The key frame KeyFrame_i in Figure 3 implements a process of removal KILL(SOUND_OBJECT_3) of the sound object SOUND_OBJECT_3. Accordingly, at the subsequent key frame KeyFrame_i+1, the array SOUND_OBJECTS_TO_PLAY, shown by a dotted line as 16a in Figure 2, only contains the pointers SO1, SO2 and SO4 to the objects SOUND_OBJECT_1, SOUND_OBJECT_2 and SOUND_OBJECT_4. The relevant code is simply a call to the function KILL(SOUND_OBJECT_X) defined in KeyFrame_1:

```
kill(soundobject1,0,1,1);
```

[0163] On the time line in Figure 4, in the fourth secondary unit of time CYCLE shown explicitly as C4, the various different objects being run are illustrated correspondingly. It should be noted that the animation ANIMATION_B associated with the digital sound fragment SOUND_FRAGMENT_3 in the sound object SOUND_OBJECT_3 is still displayed since in the example of Figures 2-4 it was assumed that said animation could survive the sound, by means of a true Boolean value of the parameter *andfree* passed on to the function KILL(SOUND_OBJECT_3). However, the frames of the animation ANIMATION_B follow their linear progress, that is to say the sound engine SOUND_ENGINE 19 does not force synchronism of certain frames with the primary units of time LOOPs. This is expressed in Figure 2 by indicating the frames displayed at the start of the primary units of time L1-L4 of secondary unit of time C4 as BX.

[0164] Subsequently, at the key frame KeyFrame_j in Figure 3, playing out of the digital sound fragment SOUND_FRAGMENT_3 is resumed by means of a resumption process RESUME(SOUND_OBJECT_3).

[0165] Accordingly, at the next key frame KeyFrame_j+1, the array SOUND_OBJECTS_TO_PLAY shown in dotted line as 16b contains once again the pointers SO1,SO2,SO3 and SO4 (ID 20) to all the sound objects SOUND_OBJECT_1, SOUND_OBJECT_2, SOUND_OBJECT_3 and SOUND_OBJECT_4. The relevant code is simply a call to the function RESUME(SOUND_OBJECT_X) defined in KeyFrame_1:

```
resume(soundobject1);
```

[0166] Correspondingly, the various different objects being run are illustrated on the time line in Figure 4, in the fifth secondary unit of time CYCLE shown explicitly as C5; these objects are identical to those of the third secondary unit of time CYCLE C3 described above.

[0167] It should be noted that the animation ANIMATION_B associated with the digital sound fragment SOUND_FRAGMENT_3 in the sound object SOUND_OBJECT_3 has resumed synchronism with the digital sound fragment SOUND_FRAGMENT_3 being played out, that is to say that the sequence of its frames displayed at the start of the various different primary units of time LOOPS is again in accordance with the data 252 SPIN_POINTS=(1,1,2,3) of the sound object 2 SOUND_OBJECT_3. It is assumed here that ANIMATION_B is still active, that it to say that it has not been removed in the meantime. If, on the other hand, it has been removed, the sound engine SOUND_ENGINE 19 ignores it.

[0168] Lastly, the key frame KeyFrame_k in Figure 3 represents a stable key frame that implements altogether the various different additional controllers of the main controller MAIN_CONTROL 17.

[0169] Among said controllers, by way of example the code that implements a function of fading-in and -out transition between volumes of groups of digital sound fragments SOUND_FRAGMENT_X being played out is illustrated as part of the special effects controller FX_CONTROLLER 174:

```
onClipEvent (load) {
        // "vols" array can be created here –or created/modified elsewhere when needed
        // these 2 arrays include sound lists to be faded/raised:
        tofade = new Array();
        toraise = new Array();
        slope = 7;      // fade i/o velocity
}

onClipEvent (enterFrame) {
        for (f = 0; f < tofade.length; f++) {
                tofade[f].setVolume(tofade[f].getVolume() - (tofade[f].getVolume()/slope))
        }
        for (f = 0; f < toraise.length; f++) {      // return to vols values
                if (toraise[f].getVolume() < vols[f]) {toraise[f].setVolume(toraise[f].getVolume()
+ (vols[f]/slope))}
}
```

[0170] In said code, the digital sound fragments to be faded out are inserted into the array-type variable *tofade*, while those to be faded in are inserted into the array-type variable *toraise*. It should be noted that the references for return of each digital sound fragment to its original volume were previously inserted into the array-type variable *vols.*

[0171] As a further example of such processes, the following code implements the random controller RANDOM_PLAYER 172, using the function SCRAMBLE defined in KeyFrame_1 periodically:

```
onClipEvent (load) {
        look = _root.totcycles;
        frequency = _root.cycle
}

onClipEvent (enterFrame) {
        wait = _root.totcycles - look;
        if (wait >= frequency) {          // SCRAMBLE
                //_root.scramble(int(random(4)))
                _root.scramble(6)
                look = _root.totcycles;
        }
}
```

[0172] As an alternative to creating the independent random playing out process described above, it is of course possible simply to call up the SCRAMBLE function in a key frame at the desired time.

[0173] In addition to the possibility of running animations ANIMATION_Y associated with the sound objects SOUND_OBJECT_X, it is possible, during the course of the time line TimeLine 72, to insert key frames that merely launch animations that are independent of the sound objects SOUND_OBJECT_X.

[0174] It should be noted, furthermore, that the key frames of the animation files can in turn contain other secondary sounds, that enrich the environment with additional effects such as, by way of example, pads for creating an atmosphere that are played out cyclically independently of the sound engine SOUND_ENGINE 19, or random melodies.

[0175] Furthermore, it is possible to insert an instruction into a key frame of the time line TimeLine 72 that refers to the once-only running of a given sound file, for example a compressed .MP3 file, containing a random note or phrasing. In this way, synchronism with the rest of the complex passage is ensured, with a very appealing effect, without the performance of the random note or phrasing being a burden for the sound engine SOUND_ENGINE 19.

[0176] It will be understood that the sequence described above for invoking the various different processes is given by way of example only and is in no way binding. In particular, the sound engine SOUND_ENGINE 19 and the various different controllers of the main controller MAIN_CONTROL 17 could be arranged on the time line TimeLine 72 immediately, even in the same initial key frame KeyFrame_1 or, on the contrary, they could be arranged on the time line TimeLine 72 only if and when it is necessary to make use of them. Similarly, it will be understood that the first key frame KeyFrame_1 need not contain those functions that are never called up in a specific embodiment.

[0177] A particularly preferred aspect of the present invention, described below, enables any HTML page whatsoever to be made interactive with the sound engine SOUND_ENGINE 19 in a manner that will depend on the user's behaviour at the graphical interface representing it, in particular in the page *Page.html* 6, it being unnecessary to insert proper controls into the BODY of the actual HTML page. This is particularly advantageous also for providing soundtracks for existing web sites, since it is not necessary to make any large-scale changes to the body of the HTML pages.

[0178] This is made possible by means of the JavaScript™ code indicated below, included in an HTML page, for

example at the bottom of its body section (<BODY>); this code is supported both by the Internet Explorer™ browser by Microsoft Corporation of Redmond, WA, U.S.A. and by the Netscape Navigator™ browser by Netscape Communications Corporation of Mountain View, California, U.S.A.

[0179] The code shown below allows stating once only which categories of elements in an HTML page are capable of interacting and what type of interaction this is. All the objects belonging to these categories are indexed and one or more interaction criteria are attributed to them.

```
// INIT
var order = "page";      // "user" or "page" (IE and N6 ONLY)
var targetobj = new Array ("A", "AREA");        // select what to index ("page" mode only)
var flagg = false;
var indexnum = -1;
var objlist = new Array();
if (navigator.appName == "Microsoft Internet Explorer") {
   document.onmouseover = fire;
   document.onmouseout = putout;
}
else if (navigator.appName == "Netscape") {
   document.captureEvents(Event.MOUSEOVER | Event.MOUSEOUT);
                if (navigator.appVersion.indexOf("4.")!=-1) {
                        // no "page" for N4.X
                        order = "user";
                document.onmouseover = N_fire;
                document.onmouseout = N_putout;
                } else if (navigator.appVersion.indexOf("5.")!=-1) {//appVersion 5 is Netscape6
                document.onmouseover = N6_fire;
                document.onmouseout = N_putout;
                }
}

function fire() {          // fire IE
                var obj = event.srcElement;
```

```
                while (obj.tagName != 'A' && obj.tagName != 'BODY') {          // A HREF
            obj = obj.parentElement;
            if (obj.tagName == 'A' || obj.tagName == 'BODY')
               break;
         }
           if (obj.tagName == 'A' && obj.href != '') {
                        indexer(obj);    // try adding to objlist
         } else {
                        var obj = event.srcElement;
                        if (obj.tagName == 'AREA' && obj.href != '') {   // MAP AREA
                                indexer(obj);    // try adding to objlist
           }
                  }
 }

 function putout() {       // putout IE
         indexnum = -1;          //label.innerText = indexnum;
         //document.title = "INDEXNUM = " + indexnum;
         soundengine.SetVariable("/switcher:trk", -1);
 }

 function N_fire(e) {      // fire NETSCAPE 4.X
         N4obj = e.target;        //(variant for N4)
       obj = e.target.toString();
                if ((obj.indexOf(":")!=-1) && (obj.indexOf("#")!=-1)) {
                        indexer(N4obj);          // try adding to objlist
         }
 }

 function N6_fire(e) {     // fire NETSCAPE 6
       obj = e.target;
       while (obj.nodeName != 'A' && obj.nodeName != 'BODY') {         // A HREF
          obj = obj.parentNode;
          if (obj.nodeName == 'A' || obj.nodeName == 'BODY')
             break;
       }
       if (obj.nodeName == 'A' && obj.href != '') {
                        indexer(obj);    // try adding to objlist
         } else {
                        obj = e.target;
                        if (obj.nodeName == 'AREA' && obj.href != '') {          // AREA
                                indexer(obj);    // try adding to objlist
         }

                  }
 }

 function N_putout() {    // putout Netscape (any)
         indexnum = -1;
         // soundengine is the id/name of swf object/embed
         soundengine.SetVariable("/switcher:trk", -1);
 }

 function indexer(entry) {
 if (order == "user") {
         // USER INTERACTION ORDER: let the user create the index by "touching" objects
         allow = true;
         for (i=0; i<objlist.length; i++) {
                 if (entry == objlist[i]) {
                        allow = false;
                        indexnum = i;
                 }
         }
         if (allow) {
                 newpos = objlist.length;
                 indexnum = newpos;    // meaning: append after last
                 objlist[newpos] = entry;
         }
         ////alert(indexnum);
 } else {
```

```
        // PAGE ORDER (IE & N6 ONLY): index by order of appearance in source code
        if (!flagg) {indexall(); flagg=true}
        for (i=0; i<objlist.length; i++) { // match and extract index from objlist
                if (entry == objlist[i]) indexnum = i;
        }
        //alert(indexnum);
}

        //document.title = "INDEXNUM = " + indexnum;
        // HERE CALL SWF FUNCTION (indexnum)
        mix(indexnum);
}

function indexall() {
if (document.all) {
        for (a=0; a<document.all.length; a++) {          // IE
                for (t=0; t<targetobj.length; t++) {
                        if (document.all[a].tagName == targetobj[t]) objlist[objlist.length] =
document.all[a];
                }
        }
} else {
        var Nclass = new Array();       // N6 (order for class, not for source)
        for (t=0; t<targetobj.length; t++) {
                Nclass[t] = document.getElementsByTagName(targetobj[t]);
                var temp = Nclass[t];
                for (n=0; n<Nclass[t].length; n++) {
                        objlist.push(temp[n]);
                }
        }
}
}

var zloaded;

function mix(trk) {
if (zloaded != "undefined") {     // zloaded is a flag coming from swf complete loading
        //if (document.all) {
                // soundengine is the id/name of swf object/embed
                soundengine.SetVariable("/switcher:trk", trk);
        //} else {
                // soundengine.SetVariable("/switcher:trk", trk); with “Netscape” suitable
syntax :)
        //}
}
}
```

[0180] Even more advantageously, this code may be stored in a separate file (JavaScript™ document with the extension .js, for example *interactionsoundengine.js*). It will thus be sufficient to insert a single instruction into each HTML page that is desired to be enabled automatically to intercept the user's actions and to transmit them to the sound engine SOUND_ENGINE 19:

<SCRIPT src="interactionsoundengine.js"></SCRIPT>

[0181] It will be understood that, as an alternative to the embodiment described, the files *Main.swf* 7 and *SoundFragment1.swf* 5a could be combined in a single file containing both the first digital sound fragment *Fragment_1_Sound_Data* in the sound library and the various different KeyFrame_X described above in the time line.

[0182] In the practice of the present invention, obviously, to produce a sound environment ex novo, designed specifically to be performed according to the present invention, is undoubtedly the best way to ensure maximum performance; notwithstanding, the management technique according to the present invention is, theoretically, applicable to any existing music or sound effect provided it is possible to adapt its arrangements by means of recursive and non-recursive sound fragments, possibly in a creative manner.

[0183] The music genres that are most suitable for being played out according to the digital sound management technique of the present invention are undoubtedly dance genres, such as techno, jungle, hip hop and funky, in which the philosophy of cyclic repetition is already a strong feature. Unexpectedly, however, excellent experimental results have been achieved even with piano scores of the minimalist avant-garde type, jazz sessions, with recurrent broken chords or with rock arrangements, and even with parts that are sung. Like all genres that are not based on a constant

rhythmic grid, classical music is the most difficult to adapt to the present invention, although it is not impossible.

[0184] During production, or while adapting pre-recorded parts (sound editing), it is found advisable - wherever possible and reasonable - to adopt the following provisions, that are based on the experience acquired in the experimental stage.

[0185] In the first place, it is better not to end the digital sound fragments intended to be played out recursively (value of the LOOPS variable greater than zero) with sixteenths or smaller units that insert resumption on the beat, since otherwise possible delays in restarting the playing out of the digital sound fragments, even if very minor, could be highlighted. It must be kept in mind, in this respect, that delays in starting to play out the digital sound fragments, that is to say a loss of synchronism with the primary unit of time LOOP, can be not negligible if the computer is subject to limits in terms of performance.

[0186] Furthermore, as already stated above, the digital sound fragments that are not intended to be recursively repeated do not necessarily have to be cut off so that they last for exactly one primary unit of time LOOP; instead, they may last less than 1 LOOP and even overflow into the next primary unit of time LOOP or beyond, considering that the sound engine SOUND_ENGINE 19 only controls their starting. This applies more generally speaking also to digital sound fragments intended for cyclic repetition since it is possible, by exploiting multi-channel performance, for a digital sound fragment SOUND_FRAGMENT_X associated with a sound object SOUND_OBJECT_X to start being played out for a second time while the same digital sound fragment SOUND_FRAGMENT_X, associated with the same or with another sound object, is still being played out for a first time.

[0187] Due to an effect of extremely precise synchronisation (quantisation) between two sound elements, even if the performance level is quite poor, they can be incorporated or merged into a single digital sound fragment, rather than being stored in two separate digital sound fragments intended to be played out on two separate channels, especially if the performance level of the computer is expected to be poor.

[0188] It is, furthermore, preferable for the first sounds to be (down)loaded and played out to be designed in such a way that they are as light as possible, so as to keep the waiting times down to the bare minimum. They can then be followed by more substantial sounds for which the (down)load time is by then covered and justified by sound already being played out.

[0189] PADs for creating the atmosphere can be reduced to a very small number of milliseconds in order to save bulk and can be played out cyclically independently rather than putting a burden on the sound engine SOUND_ENGINE 19: they can be launched, for example, by a dedicated key frame on the time line of any one of the .swf files present in a practical application.

[0190] In order to obtain a credible sustain effect from a PAD fragment, it can be built in the manner indicated Figure 5. In a first step, Figure 5(a), the digital sound fragment 500 is cut to a length exceeding one primary unit of time LOOP by a quantity equal to a fade effect, and is cut with a fade-in effect 501 at the initial edge and with a fade-out effect 502 at the final edge. In a second step, Figure 5(b), the digital sound fragment 500 is then cut into two parts 503 and 504 at a point 505 that is approximately half way through its length, in which the wave form 500 preferably has a volume of zero decibels. In a third step, Figure 5(c), the two parts 503 and 504 are then reversed, placing the second part 504 in front of the first part 503. In a fourth and last step, Figure 5(d), the first and the second parts 503 and 504 are overlapped for the length of the fade-out 502 of the second part 504 and of the fade-in 501 of the first part 503, thus giving rise to cross-fading of the two parts 503 and 504. When such a reconstructed digital sound fragment 506 is repeated over again, the joint between the two parts played out in succession is far less perceptible due to the fact that it corresponds to the point in time at which the original waveform 500 was cut. If, in particular, the cut is made at a point in time in which the original waveform has a value of 0 dB, the joint is even less perceptible.

[0191] It is well to consider, furthermore, that compression on the dynamics of the sound can be a winning strategy, above all if the sound is listened to through standard quality PC loudspeakers.

[0192] Lastly, it is advisable to make use of the unforeseeability of the random effect, managed by the random controller RANDOM_CONTROLLER 172 or in the other manners indicated above, since a random effect in a music performed under the control of software is far more appealing. In particular it is advisable to provide, at the discretion of the author, the emission of random notes or phrases or of complementary special effects for enriching the sound environment by means of the random controller RANDOM_CONTROLLER 172 or by starting to play them from any .swf file.

[0193] For the purpose of exploiting the innovative potential of interaction with the user via the graphical interface (GUI) and of interaction with other client computer events, it is advisable, at the time of composing/producing the overall sound, to consider the fact that it is possible, for example, to study several variations on a single theme, to be alternated or played out together, giving the user the choice, for example, of activating one or another in response, for example, to interaction on a suitable interface. In this way, it is possible to create unexpected deviations in real time while the music is being played out which are very gratifying; this can be achieved with a small number of additional digital sound fragments SOUND_FRAGMENT_X or even without any additional fragments but simply varying other properties associated with the various different digital sound fragments in the corresponding sound objects SOUND_OBJECT_X 2,

such as the volume 231, the start points in time START_POINTS 22 and the panpotting 232.

[0194] Those skilled in the art will understand that a computer program for digital sound management according to the invention may conveniently be designed as a programming kit or a parametric program including the functions or processes described above, or at least the process of creation of a sound object MAKE(SOUND_OBJECT) and the sound engine SOUND_ENGINE 19, and having means for receiving as inputs parameters such as the various data 21-25 of the sound objects 2, including the names of the files containing the digital sound fragments and, possibly, the global variables 41. Such a parametric program or programming kit will take care of generating in particular the file Main.swf 7 and, possibly, of incorporating it into a page Page.html 6.

Claims

1. Method for playing out digital sound at an electronic computer system, including the steps of:

    a) loading (121) a first fragment of digital sound data (5a),
    b) starting (19) to play out at least once said first fragment of digital sound data (5a),
    c) at least one step of loading (122, 123, 124) a respective second fragment of digital sound data (5b,5n), and
    d) starting (19) to play out at least once said second fragment or each of said second fragments of digital sound data (5b,5n),

    wherein the step b) of starting (19) to play out at least once said first fragment of digital sound data (5a) is carried out without waiting for the end of said at least one step c) of loading (122, 123, 124) a respective second fragment of digital sound data (5b,5n).

2. Method according to claim 1, **characterised in that** the step b) of starting (19) to play out at least once said first fragment of digital sound data (5a) is carried out before carrying out said at least one step c) of loading (122, 123, 124) a respective second fragment of digital sound data (5b,5n).

3. Method according to claim 1 o 2, **characterised in that** it includes a step e) of synchronising (19) each step b), d) of starting (19) to play out a respective fragment of digital sound data (5) on the basis of a primary unit of time (LOOP).

4. Method according to claim 3, **characterised in that** it includes the step f) of associating (141,142,143) with each fragment of digital sound data (5) playing sequence data (22) that is indicative, for each primary unit of time (LOOP), of whether or not the respective fragment of digital sound data (5) has to be played out during the primary unit of time (LOOP), and **in that** step e) of synchronising (19) includes, at each primary unit of time (LOOP), the step of carrying out those steps b), d) of starting (19) the playing out for which the playing sequence data (22) associated with the respective fragment of digital sound data (5) in said step f) of associating (141,142,143) indicate that it has to be played out during said primary unit of time (LOOP).

5. Method according to claim 4, **characterised in that** it includes the step g) of providing a secondary unit of time (CYCLE) that is a pre-established multiple of the primary unit of time (LOOP) and **in that** in step f), with each fragment of digital sound data (5) said playing sequence data (221) referred to each primary unit of time (LOOP) within a secondary unit of time (CYCLE) are associated.

6. Method according to claim 4 o 5, **characterised in that** the playing sequence data (22) further include repetition data (222) indicative of a number of consecutive repetitions of the digital sound fragment (5) each time it starts (19) to be played out.

7. Method according to any of the foregoing claims, **characterised in that** it includes a step h1) of associating (141,142,143) with each fragment of digital sound data (5) playing out data (23,24) indicative of the values of parameters for playing out their respective fragment of digital sound data (5).

8. Method according to any of the foregoing claims, **characterised in that** it includes a step h2) of associating with each fragment of digital sound data animation data (25) indicative of image data (253a, 253b) and a step i) of displaying said image data (253a, 253b) in association with the playing out of the respective fragment of digital sound data (5).

9. Method according to claim 8 when directly or indirectly dependent upon claim 4, **characterised in that** said animation data (25) include data (251) indicative of video animation (253a,253b) and, at least for each primary unit of time (LOOP) of a secondary unit of time (CYCLE), data (252) indicative of frames of the video animation (253a, 253b), and the step i) includes displaying said frames of the video animation (253a,253b) in synchronism with said primary units of time (LOOP) at least during those primary units of time (LOOP) in which said playing sequence data (22) are indicative of the playing out of the respective digital sound fragment (5).

10. Method according to claim 4 or any of claims 5 to 9 when directly or indirectly dependent upon claim 4, **characterised in that** it includes a step j) of maintaining (171,172) a current set (16) indicative of those fragments of digital sound data (5) which, according to said playing sequence data (22), have to start to be played out at least once during a primary unit of time (LOOP) within a current secondary unit of time (CYCLE) and **in that** in said step e) of synchronisation said steps b), d) of starting (19) to play out the fragments of digital sound data (5) in said current set (16) are carried out.

11. Method according to claim 10, **characterised in that** said step j) of maintaining a current set (16) includes removing (171, 172, 701) at least one fragment of digital sound data (5) from said current set (16).

12. Method according to claim 10 o 11, **characterised in that** it includes the step k) of maintaining (171, 172) a second set (15) indicative at least of the fragments of digital sound data (5) removed (171, 172, 701) from said current set (16) but intended to be played out in at least one future secondary unit of time (CYCLE).

13. Method according to claim 11 or claim 12 when dependent upon claim 11, **characterised in that** said step of removing (171, 172, 701) at least one fragment of digital sound data from said current set (16) includes a step of inserting said fragment of digital sound data (5) into said second set (15), carried out subject to the value of a variable associated with the fragment of digital sound data (5) and indicative of whether said fragment of digital sound data (5) has to be played out in a future secondary unit of time (CYCLE) or not.

14. Method according to claim 11 or 13 or to claim 12 when dependent upon claim 11, **characterised in that** said step of removing (171, 172, 701) at least one fragment of digital sound data (5) from said current set (16) includes continuing to display image data (253a, 253b) associated by means of animation data (25) with a fragment of digital sound data (5) removed from said current set (16) in said removal step (171, 172, 701).

15. Method according to any of claims 8, 9 or 14, **characterised in that** said image data (253a, 253b) are associated with respective second digital sound data other than said digital sound fragments (5), played out while said image data (253a, 253b)is being displayed.

16. Method according to any of the foregoing claims, **characterised in that** it includes the step 1) of modifying (11,171,172,173,174,18), within a current set (16) and/or a second set (15), playing sequence data (22), playing out data (23,24) and/or animation data (25) associated with at least one fragment of digital sound data (5) and/or of removing or inserting () at least one of said fragments of digital sound data (5) from or into a current set (16) and/or from or into a second set (15).

17. Method according to claim 16, **characterised in that** said step 1) of modifying includes up-dating said data (22,23,24,25) associated with said at least one fragment of digital sound data (5) and/or the fragments of digital sound data (5) in said current set (16) and/or in said second set (15) in a programmed manner (171), in a random manner (172) and/or in a manner (173) controlled by a Graphical User Interface (6).

18. Method according to claim 17, **characterised in that** said step 1) of up-dating the fragments of digital sound data (5) in said current set (16) in a random manner (172) includes a step of removing (701) from said current set (16) a first number (n) of fragments of digital sound data (5) randomly chosen and a step of adding (702) to said current set (16) from said second set (15) said first number (n) of fragments of digital sound data (5) randomly chosen.

19. Method according to claim 18, **characterised in that** said second set (15) contains all the fragments of digital sound data (5) loaded in said steps a) and c), said step of adding (702) includes a step of checking (703) whether the fragments of digital sound data (5) randomly chosen are already present in said current set (16) and/or if they cannot be removed from said current set (16); if the outcome of said checking step (703) is positive, said adding step (702) and said checking step (703) being repeated for a maximum number of times.

20. Method according to claim 4, **characterised in that** the synchronising step e) includes a step e1) of periodically checking (601) at a first frequency (f1), that is relatively low as compared with the length of said primary unit of time (LOOP), the time that has elapsed (MS) since the start of a current primary unit of time (LOOP) and estimating (602), for each periodic check, whether at the time of the subsequent check the time elapsed since the start of the current primary unit of time (LOOP) will be greater than the primary unit of time (LOOP); if the outcome of the estimation step (602) is negative, continuing said step e1) of periodic checking (601), if the outcome of said estimation step (602) is positive, carrying out a step e2) of periodic checking (604) at a second frequency (f2), that is relatively high as compared with the length of said primary unit of time (LOOP), the time elapsed (MS) since the start of the current primary unit of time (LOOP) until said elapsed time (MS) is equal to or greater than said primary unit of time (LOOP), and when said elapsed time (MS) is equal to or greater than said primary unit of time (LOOP), carrying out said steps a),c) of starting (19,605) to play a fragment of digital sound data (5).

21. Method according to claim 5, **characterised in that** it includes the step g1) of providing at least one tertiary unit of time (GROUP) being a pre-established multiple of the secondary unit of time (CYCLE), and **in that** in step f), second playing sequence data are associated with each fragment of digital sound data (5), said second playing sequence data being indicative, for each secondary unit of time (CYCLE) within one tertiary unit of time (GROUP), of whether or not the respective fragment of digital sound data (5) has to be played out during the secondary unit of time (CYCLE).

22. Method according to any of the foregoing claims, **characterised in that** said steps a), c) of loading include downloading of said fragments of digital sound data (5) from a remote computer.

23. Method according to any of the foregoing claims, **characterised in that** said steps b), d) of starting to play out a respective fragment of digital sound data (5) each include starting such playing out at a preselected level from among a plurality of playing out levels.

24. Computer program including a code for implementing the method according to any one of claims 1 to 23 when it is run in an electronic computer system.

25. Computer program product including a code for implementing the method according to any one of claims 1 to 23 when it is run in an electronic computer system.

26. Computer program product according to claim 25, including a plurality of Macromedia Flash™ or Macromedia Flash Player™ files, said plurality of files including a file (7) containing program code means on the time line (71) and a plurality of files (5a, 5b, 5n) containing a respective one of said fragments of digital sound data (5) in the sound library (51).

27. Computer program product according to claim 26, **characterised in that** said plurality of files further includes at least one image data file (253a, 253b) containing animation graphics.

28. Computer program product according to any one of claims 25 to 27, **characterised in that** it is embedded into or called up in an HTML page (6).

29. Computer program product according to claim 28, **characterised in that** it includes computer code means for causing interaction between the playing out of said fragments of digital sound data (5) and the Graphical User Interface corresponding to said HTML page (6).

30. Electronic computer system including means for implementing the method according to any one of claims 1 to 23.

31. Method for causing the execution of the playing out method according to any one of claims 1 to 23, including the steps of:

a) providing a plurality of fragments of digital sound data (5),
b) providing a performance time grid, split up into primary units of time (LOOP),
c) associating with each fragment of digital sound data (5), playing sequence data (22) indicative of the units of time (LOOP) of said time grid during which each of said fragments of digital sound data (5) has to be played out,
d) causing, within said computer system, the time to be tracked according to said primary units of time (LOOP),

and

e) causing, within said computer system, the playing out of said fragments of digital sound data (5) in accordance with said primary units of time (LOOP) and with said playing sequence data (22).

32. Method according to claim 31, in which said step a) of providing a plurality of fragments of digital sound data (5) includes providing a fragment of digital sound data (500), fading-in (501) said fragment of digital sound data (500), fading-out (502) said fragment of digital sound data (500), cutting said fragment of digital sound data (500) at a point (505), preferably where the waveform has a zero value, to obtain a first part (503) and a second part (504), placing the second part (504) in front of the first part (503), and overlapping the first part (503) and the second part (503) at an overlapping area (506), said overlapping area (506) essentially corresponding to said fading out (502) of said second part (504) and to said fading in (501) of said first part (503), a step of causing within said computer system said digital sound fragment (500) to be played out cyclically at least twice being further provided.

33. Parametric computer program including:

- code means (MAKE) for creating at least one set (15,16) of sound objects (2) each associating a fragment of digital sound data (5) with playing sequence data (22) indicative of primary units of time (LOOP) of a time grid during which said fragment of digital sound data (5) has to be played out, said fragments of digital sound data (5) and said playing sequence data (22) being supplied to said computer program in a parametric manner, and
- code means for causing tracking of the time according to said primary units of time (LOOP) and for starting the playing out of said fragments of digital sound data (5) in accordance with said primary units of time (LOOP) and with said playing sequence data (22).

34. Parametric computer program according to claim 33, further including code means for associating said sound objects (2) of said at least one set (15, 16) with the objects of a Graphical User Interface.

35. Parametric computer program according to claim 33 or 34, including code means for causing the steps of the method according to any one of the claims 1 to 23 to be carried out.

36. Parametric computer program product according to any one of the claims 33 to 35.

37. Electronic computer system in which a parametric computer program according to any one of the claims 33 to 35 is loaded.

38. Dynamic data structure in an electronic computer system including:

- a plurality of files containing a respective fragment of digital sound data (5),
- a current set (16) of sound objects (2) each representative of a fragment of digital sound data (5) chosen from among said plurality of files and of playing sequence data (22) indicative of primary units of time (LOOP) of a time grid during which said fragment of digital sound data (5) has to be played out by a sound engine (19) capable of tracking the time according to said primary units of time (LOOP) and of starting to play out said fragments of digital sound data (5) in accordance with said primary units of time (LOOP) and with said playing sequence data (22) in their respective sound object (2).

39. Dynamic data structure according to claim 38, further including data useful for executing the method according to any one of claims 1 to 23.

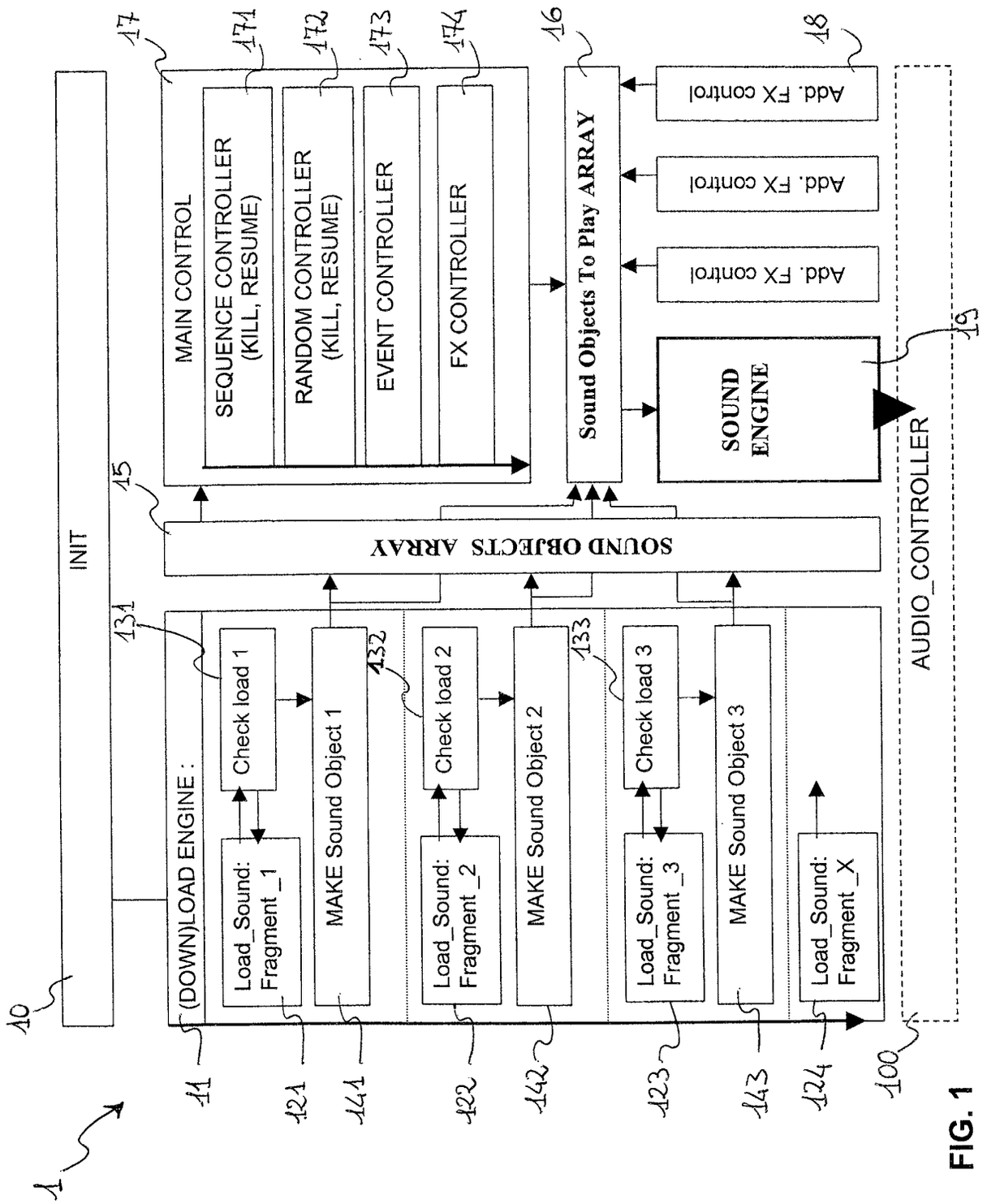
1
10
INIT
11
(DOWN)LOAD ENGINE :
121
Load_Sound: Fragment_1
131
Check load 1
141
MAKE Sound Object 1
132
122
Load_Sound: Fragment_2
Check load 2
142
MAKE Sound Object 2
133
123
Load_Sound: Fragment_3
Check load 3
143
MAKE Sound Object 3
124
Load_Sound: Fragment_X
100
AUDIO_CONTROLLER
15
SOUND OBJECTS ARRAY
17
MAIN CONTROL
171
SEQUENCE CONTROLLER (KILL, RESUME)
172
RANDOM CONTROLLER (KILL, RESUME)
173
EVENT CONTROLLER
174
FX CONTROLLER
16
Sound Objects To Play ARRAY
18
Add. FX control
Add. FX control
Add. FX control
19
SOUND ENGINE

FIG. 1

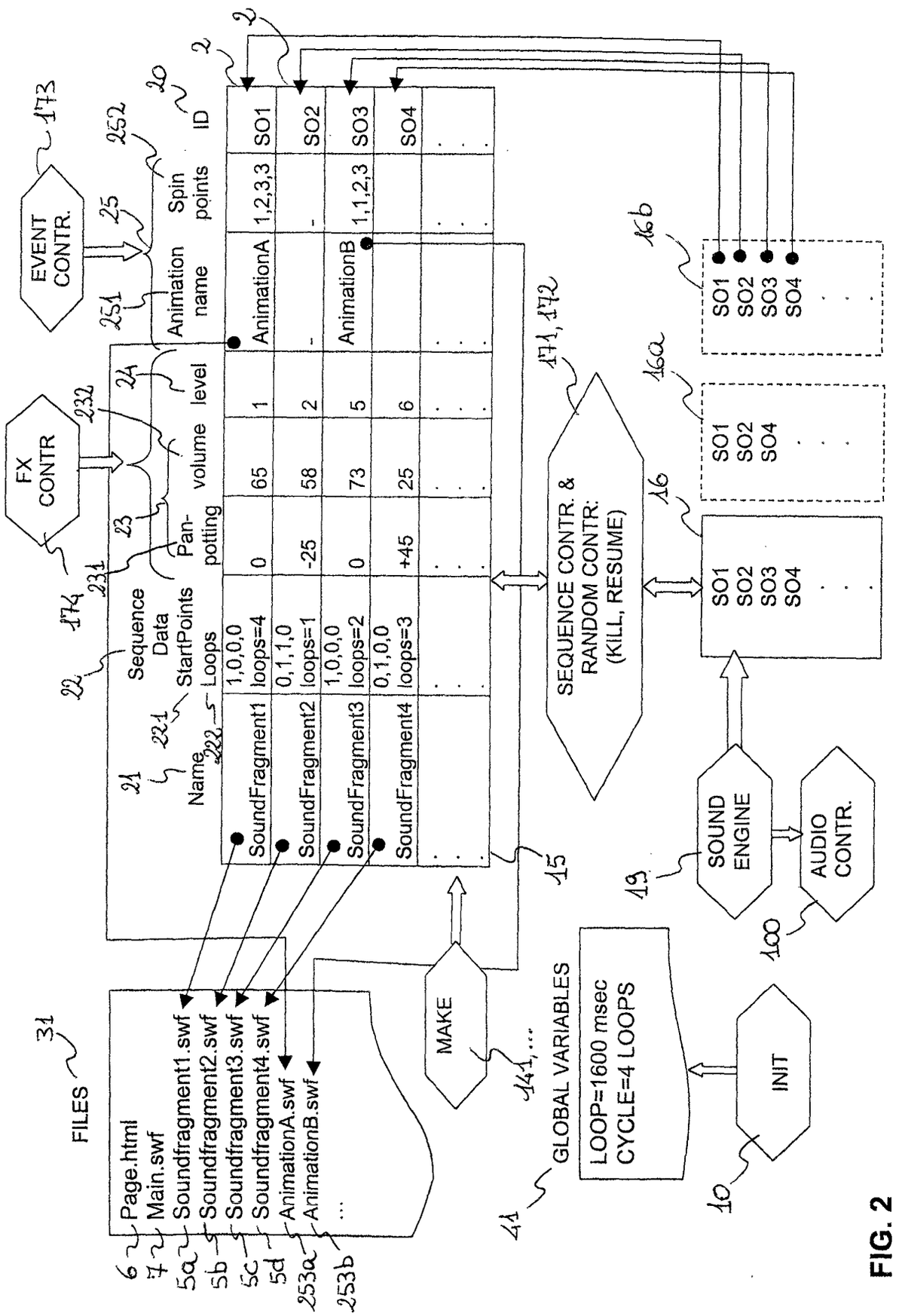
EVENT CONTR.
FX CONTR
Name
Sequence Data StartPoints Loops
Pan-potting
volume
level
Animation name
Spin points
ID
SoundFragment1
SoundFragment2
SoundFragment3
SoundFragment4
1,0,0,0 loops=4
0,1,1,0 loops=1
1,0,0,0 loops=2
0,1,0,0 loops=3
0
-25
0
+45
65
58
73
25
1
2
5
6
AnimationA
AnimationB
1,2,3,3
1,1,2,3
SO1
SO2
SO3
SO4
SEQUENCE CONTR. & RANDOM CONTR: (KILL, RESUME)
SOUND ENGINE
AUDIO CONTR.
FILES
Page.html
Main.swf
Soundfragment1.swf
Soundfragment2.swf
Soundfragment3.swf
Soundfragment4.swf
AnimationA.swf
AnimationB.swf
...
MAKE
GLOBAL VARIABLES
LOOP=1600 msec
CYCLE=4 LOOPS
INIT

FIG. 2

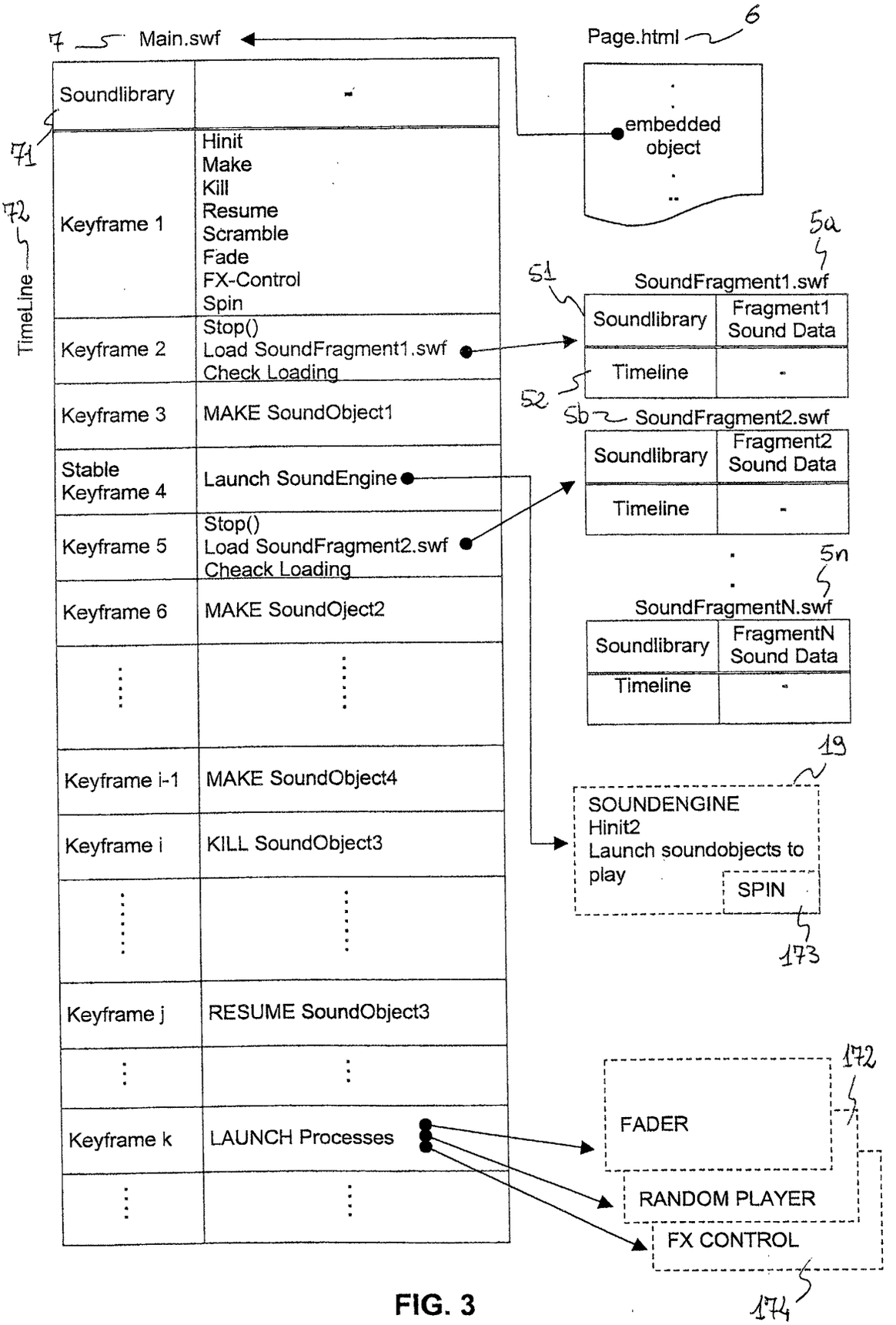
7
Main.swf
Page.html
6
Soundlibrary
-
71
Hinit
Make
Kill
Resume
Scramble
Fade
FX-Control
Spin
Keyframe 1
72
TimeLine
Keyframe 2
Stop()
Load SoundFragment1.swf
Check Loading
Keyframe 3
MAKE SoundObject1
Stable Keyframe 4
Launch SoundEngine
Keyframe 5
Stop()
Load SoundFragment2.swf
Cheack Loading
Keyframe 6
MAKE SoundOject2
Keyframe i-1
MAKE SoundObject4
Keyframe i
KILL SoundObject3
Keyframe j
RESUME SoundObject3
Keyframe k
LAUNCH Processes
embedded object
5a
51
SoundFragment1.swf
Soundlibrary
Fragment1 Sound Data
Timeline
-
52
5b
SoundFragment2.swf
Soundlibrary
Fragment2 Sound Data
Timeline
-
5n
SoundFragmentN.swf
Soundlibrary
FragmentN Sound Data
Timeline
-
19
SOUNDENGINE
Hinit2
Launch soundobjects to play
SPIN
173
172
FADER
RANDOM PLAYER
FX CONTROL
174

FIG. 3

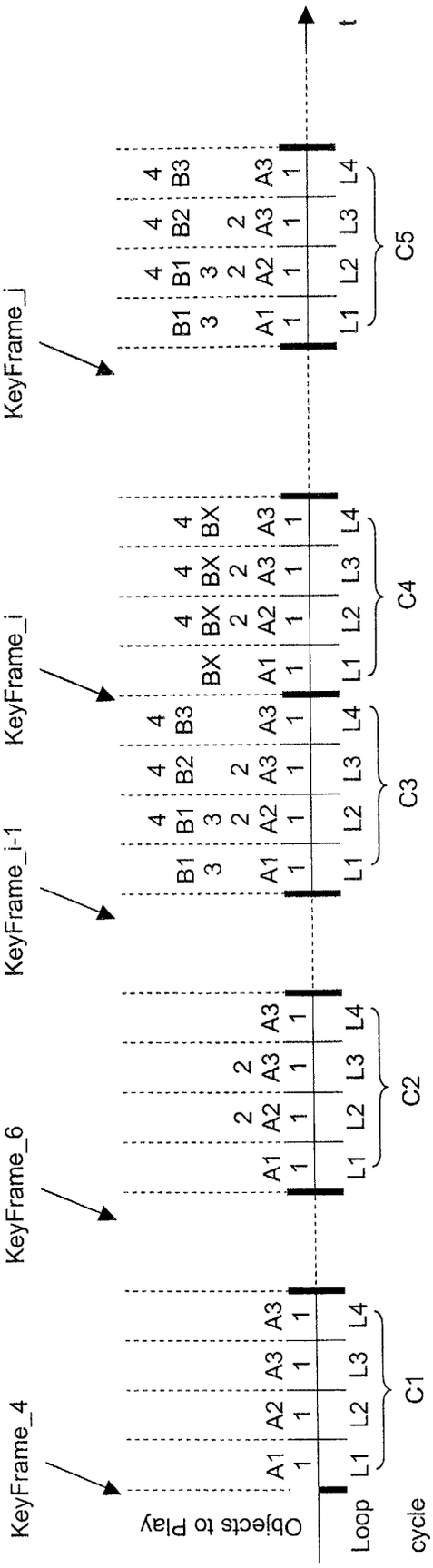
KeyFrame_4
KeyFrame_6
KeyFrame_i-1
KeyFrame_i
KeyFrame_j
Objects to Play
Loop cycle
t
C1
C2
C3
C4
C5
A1 A2 A3 B1 B2 B3 BX
L1 L2 L3 L4

FIG. 4

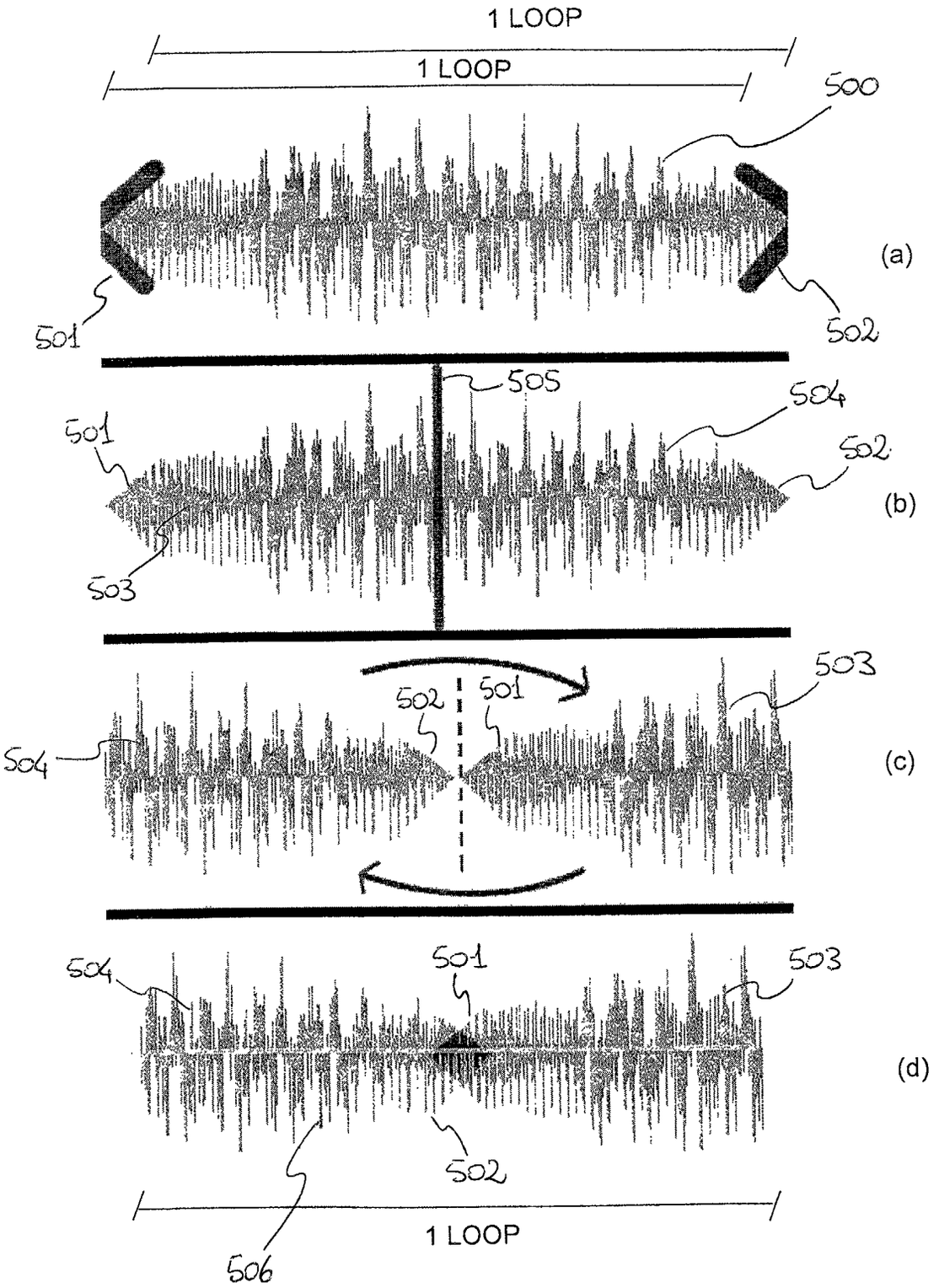
1 LOOP
1 LOOP
500
501
502
(a)
505
504
501
502
503
(b)
503
502
501
504
(c)
504
501
503
502
506
1 LOOP
(d)

FIG. 5

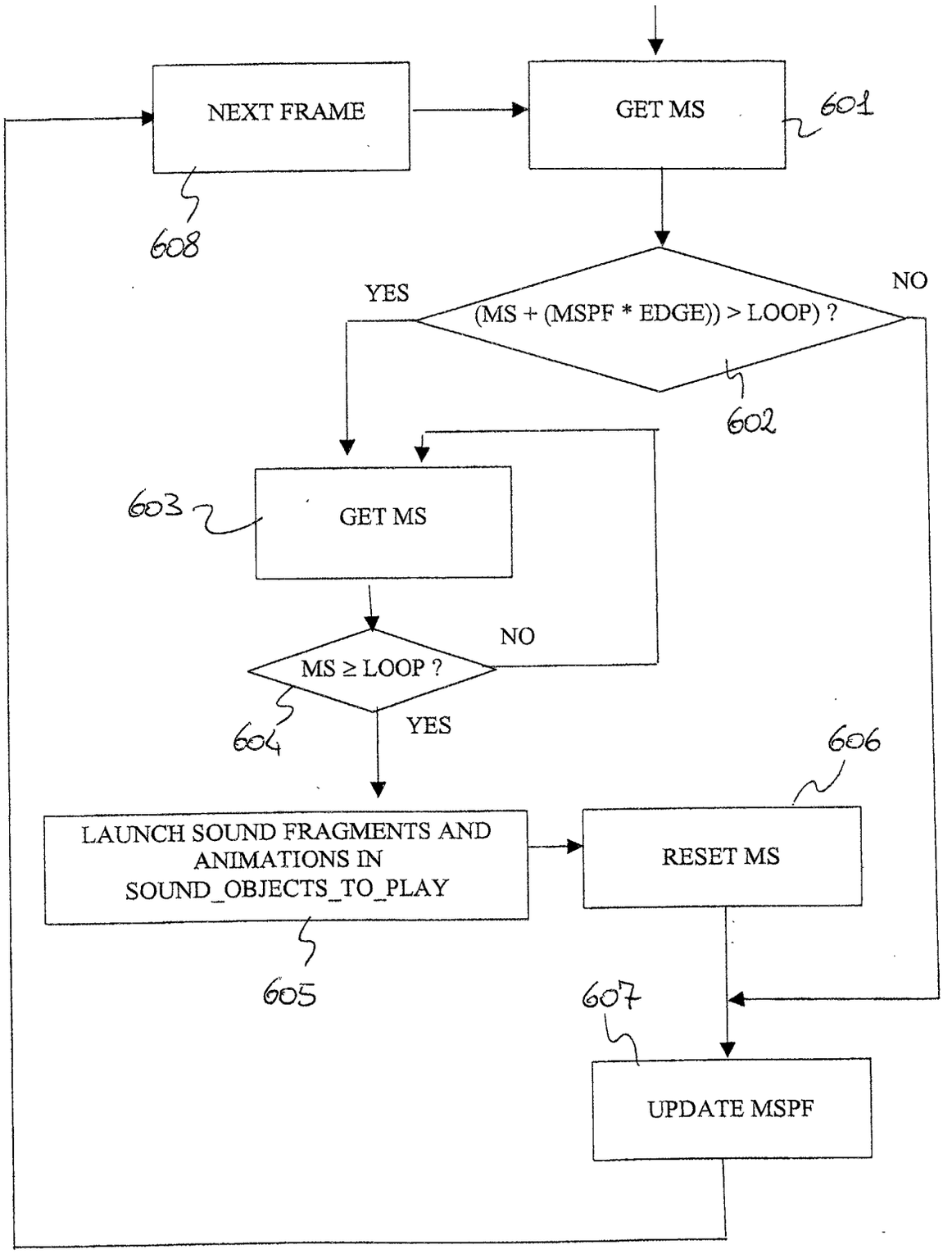
NEXT FRAME
GET MS
601
608
YES
NO
(MS + (MSPF * EDGE)) > LOOP) ?
602
603
GET MS
NO
MS ≥ LOOP ?
604
YES
606
LAUNCH SOUND FRAGMENTS AND ANIMATIONS IN SOUND_OBJECTS_TO_PLAY
RESET MS
605
607
UPDATE MSPF
19

FIG. 6

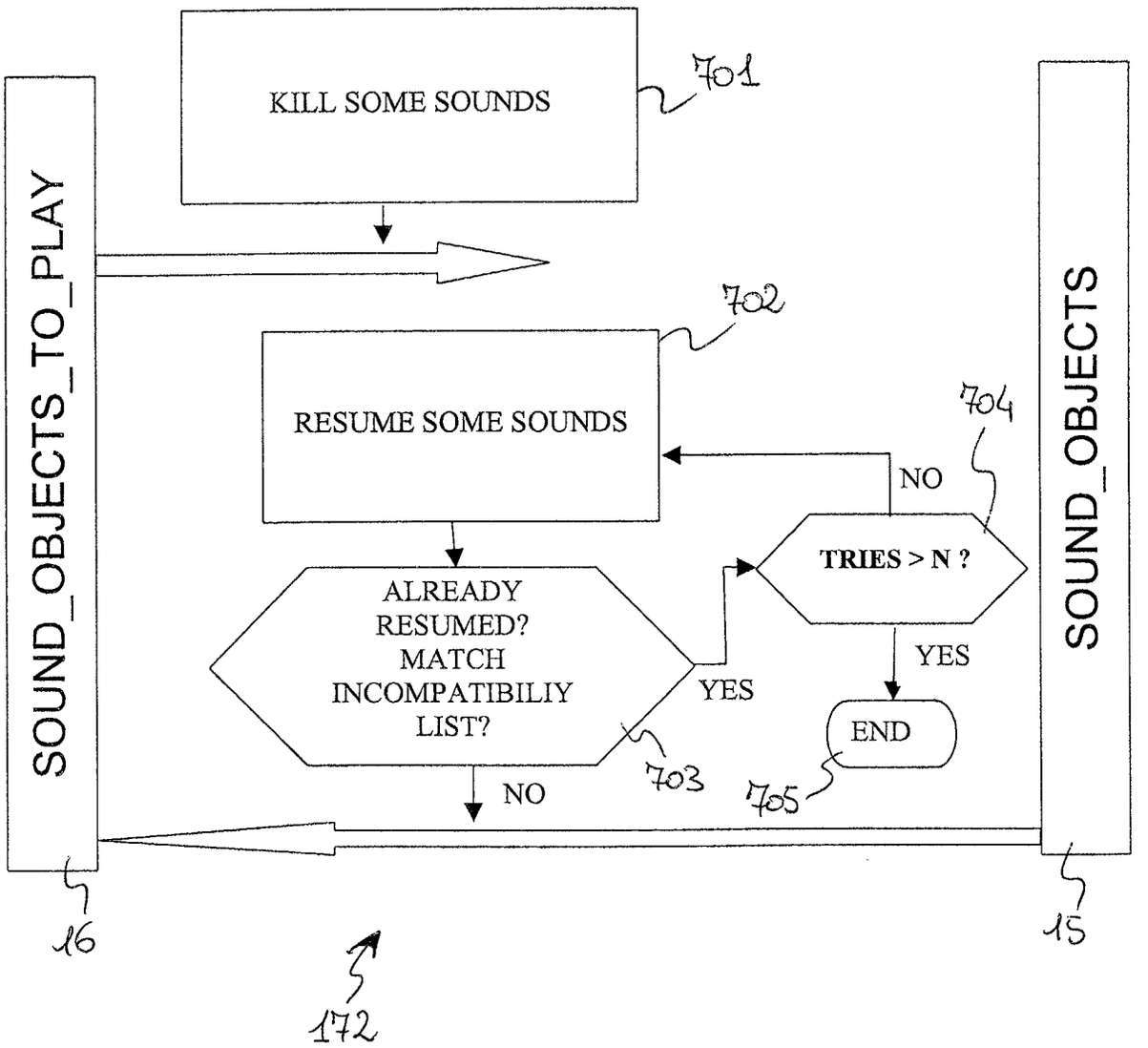
KILL SOME SOUNDS
701
SOUND_OBJECTS_TO_PLAY
SOUND_OBJECTS
702
RESUME SOME SOUNDS
704
NO
TRIES > N ?
ALREADY RESUMED? MATCH INCOMPATIBILIY LIST?
YES
YES
END
703
705
NO
16
15
172

FIG. 7

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 02 42 5318

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
A	US 5 886 275 A (KATO HIROKAZU ET AL) 23 March 1999 (1999-03-23) * column 1, line 66 - column 2, line 25 * * column 4, line 8 - column 5, line 9 * * column 7, line 11 - line 50; figure 2 *	1,2,8, 22,24, 25,30	G10H1/00
A	US 5 977 468 A (FUJII SHIGEKI) 2 November 1999 (1999-11-02) * column 5, line 58 - column 7, line 44; figures 1,2A,2B *	1,22,24, 25,30	
A	WO 01 16931 A (HAEMAELAEINEN MATTI ;NOKIA OYJ (FI); AALTONEN JANNE (FI); HOLM JUK) 8 March 2001 (2001-03-08) * page 4, line 25 - page 5, line 2 * * page 7, line 18 - page 8, line 2 * * page 15, line 14 - line 29; figures 1,8 *	1,22,24, 25,30	
A	EP 0 855 697 A (YAMAHA CORP) 29 July 1998 (1998-07-29) * page 5, line 17 - line 24 * * page 6, line 18 - line 23 * * page 7, line 14 - page 8, line 21; figures 1,7,8 *	1,22,24, 25,30	TECHNICAL FIELDS SEARCHED (Int.Cl.7) G10H

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	28 November 2002	Pulluard, R

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 02 42 5318

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

28-11-2002

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5886275 A	23-03-1999	JP 10293593 A	04-11-1998
US 5977468 A	02-11-1999	JP 11024661 A	29-01-1999
WO 0116931 A	08-03-2001	FI 991865 A	01-03-2001
		EP 1210709 A1	05-06-2002
		WO 0116931 A1	08-03-2001
EP 0855697 A	29-07-1998	DE 69710569 D1	28-03-2002
		DE 69710569 T2	31-10-2002
		EP 1126435 A2	22-08-2001
		EP 0855697 A1	29-07-1998
		JP 3271572 B2	02-04-2002
		JP 10240242 A	11-09-1998
		SG 74037 A1	18-07-2000
		US 2002027931 A1	07-03-2002
		US 2002027910 A1	07-03-2002
		US 2002085546 A1	04-07-2002
		JP 3180751 B2	25-06-2001
		JP 11231866 A	27-08-1999

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82