GAIA: Graphical Audio Interface Application

David Topper
McIntire Dept of Music, University of Virginia
topper@virginia.edu

Abstract

GAIA (Graphical Audio Interface Application) is an open source interface for controlling the RTcmix (Garton, Topper 1997) synthesis and effects processing engine. Until recently, most RTcmix research has been limited to using text-based scorefiles. The primary motivation behind GAIA is to build upon this paradigm by providing a graphical front end. An emphasis has been placed on creating an environment that is easy to learn, robust and open source to allow for third party contribution. GAIA breaks new ground in that it supports both graphical and text based programming in the same application. Objects (or nodes within the program's graph-like control structure) can themselves be small scripts, written in Perl. These scripts can operate on data within the application as well as trigger RTcmix events in real time. GAIA unites two powerful open source projects, RTcmix and Perl, and provides a powerful high level GUI for working with both. Through these mechanisms, GAIA creates a flexible and powerful environment for controlling any number of synthesis and effects processing parameters, bringing various techniques to a new level of realization.

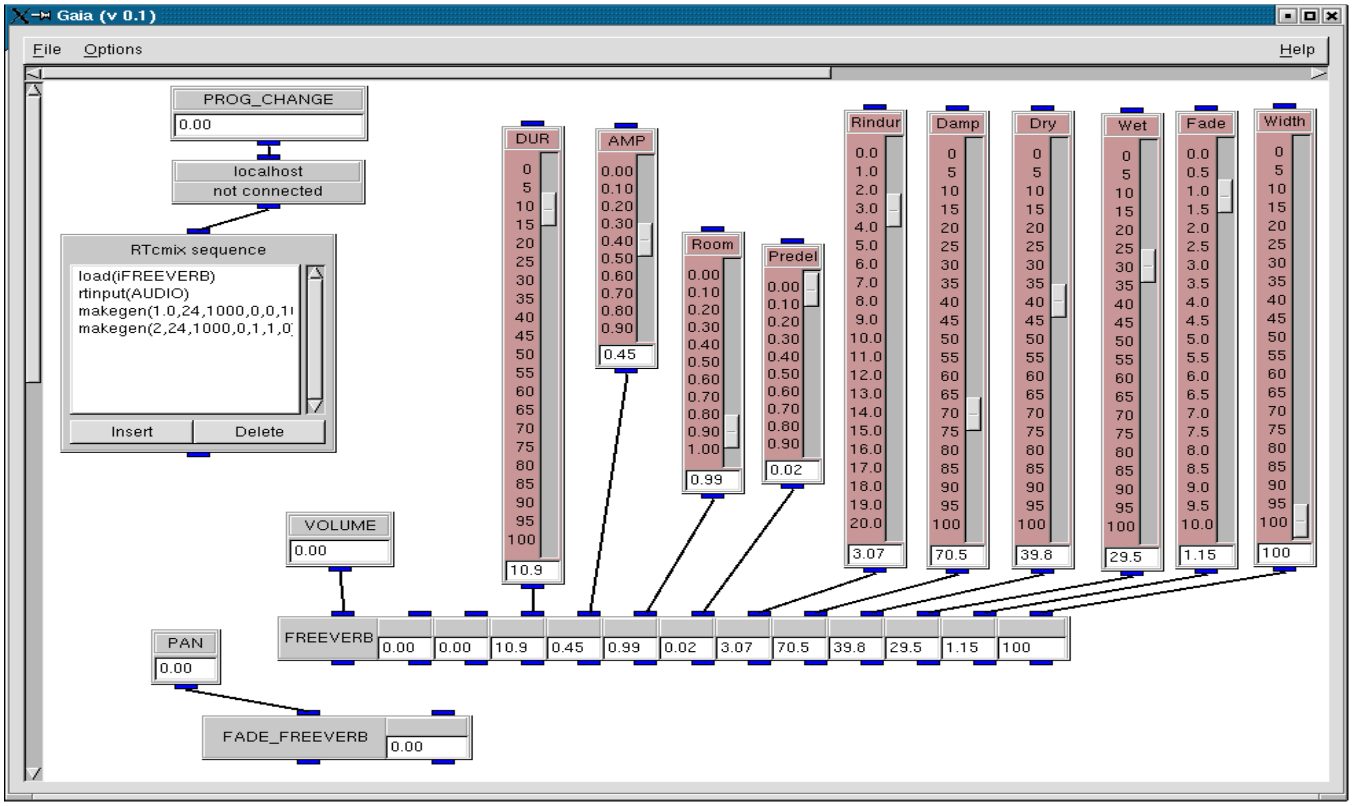
1 Introduction

GAIA is a graphical network-style application for controlling various musical processes. Specific structures (called objects) are created on a blank screen (called a canvas) and connected together like objects in a multi-node graph. Signals are sent from parents to children. The basic concept is similar to that employed by other graphical network based control applications (eg., Max/MSP, Pd, Open Music, Alsa Modular Synth, GLAME, etc...). GAIA differs from other such applications on several levels. It is completely open source, and distributed under the GNU General Public License (GPL) (gnu.org 2004). Synthesis methods are not "sub patches" within the program, but are instead RTcmix "instruments." GAIA's communication with RTcmix is done via both TCP connection and directly via the new "imbed" module. So the coding of various synthesis and effects processing algorithms is kept in an environment best suited for speed and low latency. This is becoming a trend in other applications as well (eg., Percolate (columbia.edu 2004)). Perhaps the most notable difference is GAIA's incorporation of Perl directly within the graphical structure. User's can combine any level of graphical or scripted control to suit the compositional requirements at hand.

In order to help support the Open Source paradigm, GAIA offers an API for third party developers to facilitate the creation of new objects and features within the program. Recent work has focused around video processing, which is now part of the main GAIA distribution, not an add-on module or separate package. Video data is broadcast from the "video tracking object" in the same fashion as any other object. GAIA forms part of the development on the Portable and Semi-Portable Audio Workstation (Topper 2001).

2 Design and Implementation

GAIA is written using a combination of C and C++ and built with the GTK toolkit (gtk.org 2004), running under Linux. Somewhat derivative of the Motiff (opengroup.org 2004) API, GTK objects and functions are connected via callback structure. Specifically, the program makes extensive use of the GTK/GNOME canvas. This toolkit allows for many different types of graphical objects to be rendered, then moved around the program window without extensive programming overhead. After the basic design is implemented, the canvas takes care of moving and redrawing GTK objects and other structures. As a result, someone designing a new GAIA object only needs a very basic knowledge of GTK.



```
load(iFREEVERB)
rtinput(AUDIO)
makegen(1.0,24,1000,0,0,1
makegen(2,24,1000,0,1,1,0
```

Figure 1.

This versatility has facilitated several useful features common throughout the environment. The entire program window, or canvas, can be predefined to virtually any size then scrolled up/down or left/right. This allows for the creation of a graphical network larger than the actual screen size. Similarly, the canvas and all the objects contained therein can be dynamically resized via zooming in or out. The GTK/GNOME canvas takes care of resizing the individual items therein. Specific objects are also available to control any of these functions. For example, if a user wanted to trigger a screen change via MIDI foot switch message, they would simply connect the MIDI event object to the screen change object.

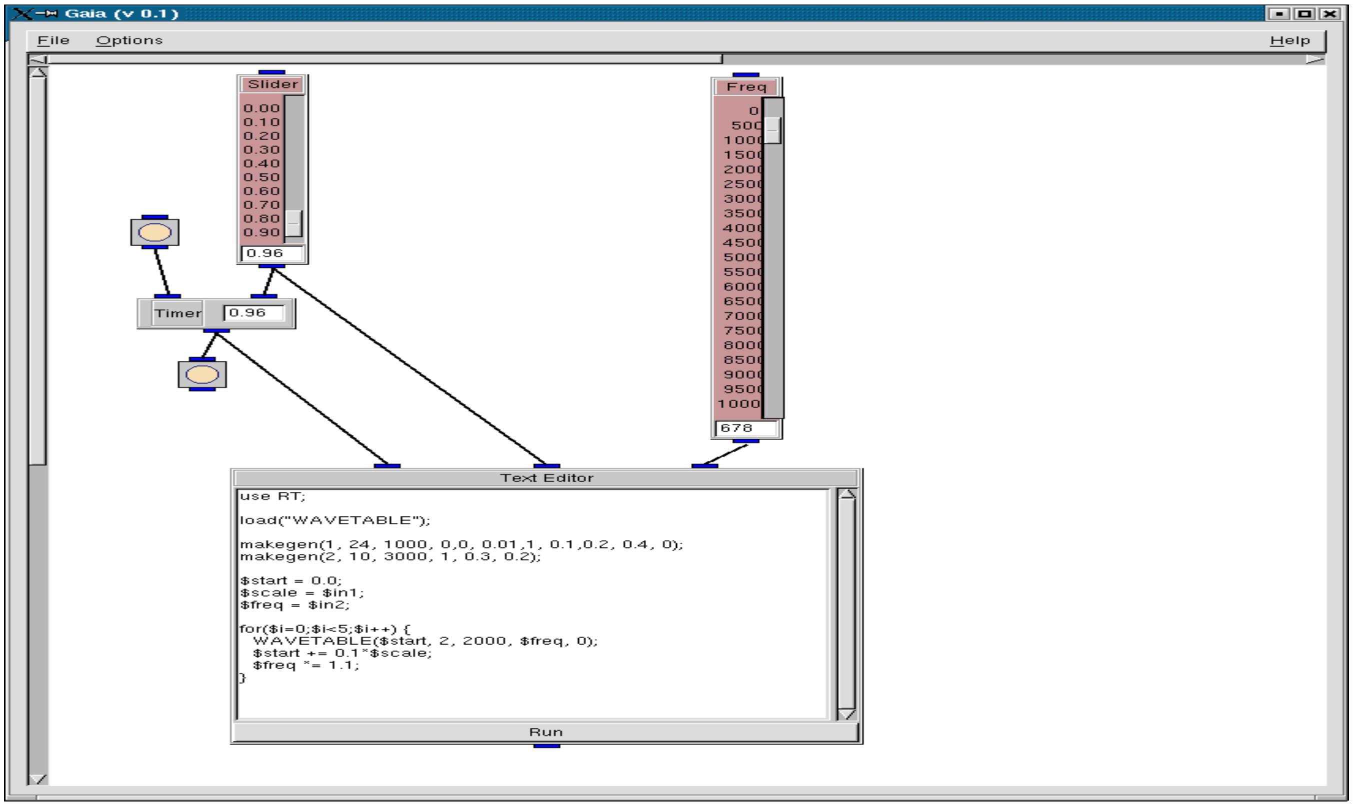
Special purpose objects can also be displayed outside the canvas window. In the case of the Perl text editor, the object itself shows up as a small node with “show” and “hide” buttons which respectively call up an external text editor for working on the specific script. Also, in the case of the video tracking object, the video image is displayed in a separate window external to the main canvas. These features facilitate cleaner screen organization.

Objects are highlighted following mouse focus, including connection lines (also known as patch cords in other applications). So when the mouse is positioned over a particular item, its outline appears in bright red. This lets the user know precisely what he/she is about to select. Multiple canvases can be created, all of which are linked in the same application or patch.

As with most graphical network based GUI applications, signals are sent throughout tree like structures. A single object can broadcast a signal to multiple child objects. Similarly, an object can broadcast different types of data through its output connections. Signal execution can be ordered left-to-right, right-to-left or any custom ordered variation. Double clicking any object will by default initiate a signal path. All objects take input signals from their first, uppermost left-hand side input connection box.

2.1 Basic GAIA Implementation

GAIA objects fall into three general categories: objects specific to GAIA, those used to directly control an individual RTcmix process, and Perl scripts. All objects have configurable options specific to their operation (eg., minimum



```
use RT;

load("WAVETABLE");

makegen(1, 24, 1000, 0,0, 0.01,1, 0.1,0.2, 0.4, 0);
makegen(2, 10, 3000, 1, 0.3, 0.2);

$start = 0.0;
$scale = $in1;
$freq = $in2;

for($i=0;$i<5;$i++) {
  WAVETABLE($start, 2, 2000, $freq, 0);
  $start += 0.1*$scale;
  $freq *= 1.1;
}
```

Figure 2.

and maximum value, string or floating point data, orientation, size, etc...).

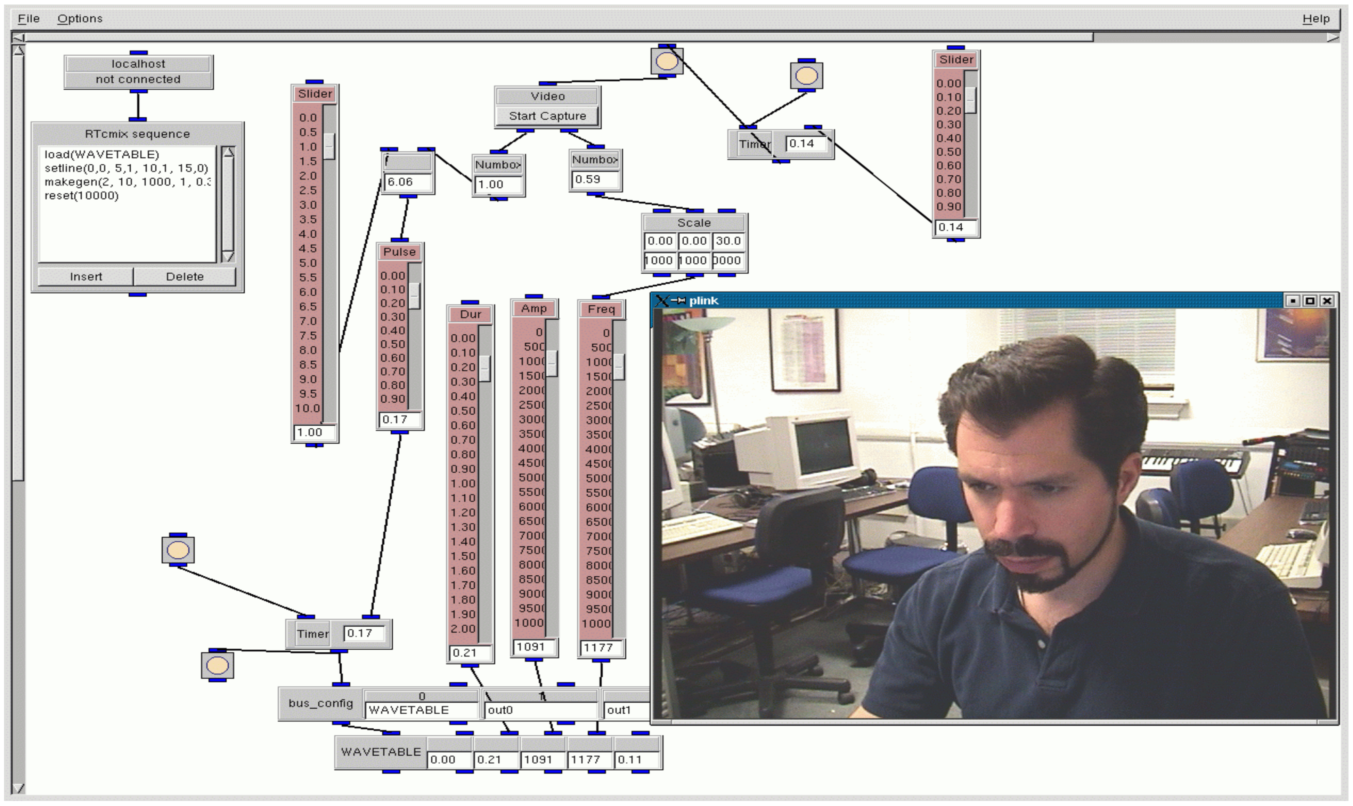
GAIA specific objects include the standard range of sliders, data boxes, timers, 2D graph plots, and logical operators. *Figure 1* illustrates a subset of those objects controlling a single running RTcmix process. In this example, MIDI events are used to both start and control the musical process. This is done by connecting the MIDI object to an RTcmix TCP object, which in turn triggers a short RTcmix script (non Perl). So when a PROGRAM_CHANGE signal is received, it triggers a sequence of events. Similarly, VOLUME and PAN messages are used to process the incoming audio stream using the RTcmix "FREEVERB" reverberation algorithm, each parameter of which is controlled via slider objects.

Singular RTcmix objects (as in *figure 1*) can communicate either to a running RTcmix engine via a TCP socket connection, or directly via the RTcmix "imbed" module. The latter allows RTcmix to be incorporated into the GAIA (or any other) program itself via statically linked library. In the case of TCP sockets, multiple engines can be controlled on virtually any number of hosts. This allows GAIA to be used as a network controller for a cluster of RTcmix engines running on different machines.

2.2 The Power of Perl

Although RTcmix has been using Perl as a front end parser for many years, combining that ability with a graphical front end represents a new paradigm. In the case of GAIA, Perl is not parsing a text file. Instead, it parses and evaluates a buffer in real time, maintaining a persistent state during the entire run of the application. Perl provides built in functionality for this. Specifically, GAIA uses the eval_sv() function which is part of perlembed (perl.org 2004).

Figure 2 illustrates a basic implementation of the Perl text buffer object. In this example, a Perl script takes input from from two slider objects and is executed by a GAIA timer object, which takes two inputs (a trigger to start pulsing and a pulse interval). The script contains a loop which plays five notes in ascending pitch. This particular script has three inputs, one to execute, and two for input variables. The first input variable controls the "pulse" rate at which the script is executed. It is identical to the value sent to the timer object, but in this case is also used to control scaling of note duration as the tempo changes. The second variable is frequency, used



```
load(WAVETABLE)
setline(0,0, 5,1, 10,1, 15,0)
makegen(2, 10, 1000, 1, 0.3
reset(10000)
```

Figure 3.

to control the base starting frequency of the script loop. Both variables are controlled via slider objects. In the case of tempo, the same slider is used to control the GAIA timer object as well as send the timer interval value to the Perl script.

Perl objects can also be created in a separate text editing window. As illustrated by *Figure 2*, special reserved variables are $inX and $outX where X corresponds to the respective input or output connection box number. The complete Perl programming language is supported and many separate Perl objects can be used simultaneously. In this sense, GAIA could serve as an IDE for Perl development, facilitating the creation of distinct subroutines as visibly separate objects. Data is shared globally by default, but can also be made local in scope as an option. Each Perl text box in GAIA that is not defined to be global in scope is actually a Perl subroutine. This fact is hidden from the user, with GAIA maintaining a list of all created Perl buffers and their respective subroutine names. This allows data to be cast as local in scope, as well as providing a speed increase by having Perl buffers evaluated in advance of signal execution. Buffers are evaluated either by sending a signal to their second input data box, or by clicking on the "eval" button in the perl text buffer editor. Any valid perl expression can be executed.

GAIA also enlists the power of Perl for logical expression evaluators. These objects, called Perl signal condition objects, are similar to Perl text buffer objects. The main difference is that they take input data and are evaluated for boolean true/false condition. If true, the signal and data are passed along to the object's outputs.

2.3 Real Time Video Processing

Since GAIA places control of virtually any synthesis / effects processing parameter into a robust and vastly reconfigurable environment, control mechanisms can move beyond MIDI and other standard protocols. By reading pixels from a live video feed it is possible to implement any number of computer vision algorithms to control audio synthesis and effects processing.

Figure 3 shows a simple RTcmix WAVETABLE instrument being controlled by short and long-term image pixel difference change. In this example, a process is started by double clicking a start- node object, called a blinker object. Blinker objects simply allow signals to pass through, registering a brief color change (or blink) when the signal is received/transmitted. They may also be used to initiate a signal path. In this case, the signal starts a timer object which sets the control rate for video processing. With each signal sent to the video tracking object, a single frame is read and analyzed. Currently up to thirty frames per second can be processed, but this is only a hardware limitation imposed by commercial video equipment. Theoretically, higher end video frame grabbers can also be used to achieve an even higher data acquisition rate. The video tracker object takes a trigger as input, and sends out visual analysis data of the incoming video stream. In this case, once the process is started, starting frame difference is used to set the tempo of a pulsed sinusoid. Previous frame difference is used to control pitch. So as an object enters the field of view, a tone begins pulsing. As the object moves about in that field, the pitch increases / decreases in conjunction with that movement.

Currently, only very basic image processing algorithms are supported: frame difference (both from immediately previous and start frame) and color intensity. Previous frame difference can be used to approximate current movement in a field of view, it can be thought of as measuring motion. Start frame difference can be used to measure the change in state of a scene (eg., a dancer's entrance), it also sometimes referred to as occlusion. Color intensity can be used to monitor the entrance or exit of a specific color coded entity.

Mapping of the start frame difference data is done via the GAIA scale object, which is particularly interesting due to it's dynamic ability. The scale object can take five inputs: input data, max and min, and output max and min. Each of these can be varied in real time. So, for example, if the user wanted to change the output register in this example, s/he would simply change the output maximum and/or minimum.

GAIA uses the Video4Linux (V4L) and the Linux IEEE 1394 (aka. Firewire) drivers for grabbing video frames. The former offers a wide range of support for PCI and PCMCIA video input devices, many of which perform a good deal of computational work on board, thus increasing the effective frame processing rate. Whereas the latter provides support for more recent hardware and is more general purpose.

3 Internal Programming API

GAIA is written primarily in C but uses some C++ where applicable, particularly when communicating with the RTcmix “imbed” module. Various structures and functions exist to provide an easy to use API for creating new objects.

3.1 Structures

The top level structure is called a "canvas object." It represents the top level of the visual component you see on the screen. The object contains pointers to specific objects as well as the graphical widgets necessary for rendering and movement; this includes the graphical component of an object's connection (eg., input and output) boxes. Canvas objects point to a single or linked list of "signal_objects."

The second level structure is called a "signal_object." This is the primary unit of object design. All objects are signal objects, or groups of signal objects. This structure contains various pointers to specific objects and their internal data. All computational aspects of an object are done on this level as well. So, for example, in the case of the Perl object the text editing window is part of the canvas object (mentioned previously) but the specifics of executing the running Perl interpreter (to parse and evaluate the specific text) are done in the signal object, specifically by the signal_function.

The signal object also maintains pointers to an object's internal (vs. graphic) connection box structure. Each "box" is a basically a pointer to a series of "connection" structures which are in essence special purpose objects themselves. As a result, it is the signal object that receives event triggering routed through the input and output boxes for a given object network.

3.2 Functions

Creating a new object is relatively easy. There are four basic functions required: properties_window(), creation_function(), connection_function(), and signal_function().

The properties_window() function allows the user to configure the particular object. An API is provided to help the programmer create a pop-up dialog box for the given object. This is used to set initial parameters such as maximum and minimum value, data type, size, filename, and any other specific options necessary to the object in question.

Once the user enters the information from the above dialog, it is then passed to the creation_function() via an in_params[] array. The creation function basically defines the main aspects of the object, including any special purpose graphical objects or GTK structures, as well as input/output boxes. GTK supports a box packing mechanism (gnome.org 2004) for graphical objects. GAIA simplifies that paradigm somewhat by requiring the programmer to provide only a top level pointer to the highest level packed box. This pointer is used by the canvas_object structure to draw the objects and move or resize them on the screen canvas as needed.

As in the properties_window() function, an API is provided here to facilitate connection box creation. The programmer simply needs to define the total number of boxes, their type (INPUT, OUTPUT, DATA, TEXT, etc...), the edge on which they will be drawn (NORTH, SOUTH, EAST, WEST) and the location along the respective edge (in percent).

When object's are connected together, the respective connection_function() is executed. This function allows objects to make specific configurations depending on what they're connected to. For example, if a parent object outputs floating point data the child object will receive this information at connection time and then be aware of this data type as input. This system allows for the creation of "smart connections" which can be used to facilitate special purpose connections between objects. Scale objects, for example, make use of the connection_function() in order to set minimum and maximum values for their input and output.

Event triggering is accomplished through each object's signal_function(). Signal functions are member functions of the signal_object structure. The programmer simply needs to determine in this function what is done with input and output data, or if some other action needs to be taken (eg., writing to a TCP socket). Once an object is triggered, it sends out signals to every object connected to all of its output boxes (defined earlier). The process is recursive and continues until there are no more connections.

Once the four main functions are defined, the object's respective C file can be included in the Makefile and then compiled into the executable. There also is some minor additional bookkeeping necessary to maintain object types and specific attributes.

4. Future Directions

GAIA, while stable, is a relatively new application. A main area of work in the short term will focus around the creation of new objects. GAIA also currently stores configuration data in a proprietary format. The next logical stage is to replace this storage mechanism with XML (xml.org 2004) which is rapidly becoming a new standard.

RTcmix has recently been ported to OS X. GAIA support for OS X, however, is still in a very preliminary phase. Support for X windows based libraries like the GTK canvas, however, still involves some additional work.

As a corollary to GAIA's new video processing object, a video output object is in the works using the Linux SDL library and Gstreamer. The SDL library has already been used to port many commercial games and graphics applications to Linux. Gstreamer aims at being the new default Linux multimedia API, but is still in the early development phase. GAIA can be downloaded via ftp at: http://presto.music.virginia.edu/pub/gaia

References

Garton B. G., and D. Topper. 1997. RTcmix -- Using CMIX in Real Time, *Proceedings of the International Computer Music Conference, 1997.*

Topper, D. T. 2001. PAWN and SPAWN (Portable and Semi Portable Audio Workstation). *Proceedings of the International Computer Music Conference, 2001.*

http://developer.gnome.org/doc/API/gtk (n.d.) Retrieved, 1/16/04

http://www.gnu.org (n.d.) Retrieved 1/10/04

http://www.cycling74.com (n.d.) Retrieved 1/20/04

http://http://www-crca.ucsd.edu/~msp/software.html (n.d.) Retrieved 1/22/04

http://www.ircam.fr/equipes/repmus/OpenMusic/ (n.d) Retrieved 1/22/04

http://www.agnula.org/packages/alsa_modular_synth/ (n.d.) Retrieved 1/28/04

http://glame.sourceforge.net/index.var (n.d.) Retrieved 1/28/04

http://www.perl.org/ (n.d.) Retrieved 1/28/04

http://www.music.columbia.edu/PeRColate/ (n.d.) Retreived 1/28/04

http://www.xml.com/ (n.d.) Retreived 1/29/04

http://www.virginia.edu/music/VCCM/ (n.d.) Retreived 1/29/04

http://www.opengroup.org/motif/ (n.d.) Retrieved 1/30/04