GAIA: Graphical Audio Interface Application

David Topper

Virginia Center for Computer Music, University of Virginia
email: topper@virginia.edu

Abstract

GAIA (Graphical Audio Interface Application) is an open source interface for controlling the RTcmix synthesis and effects processing engine. Despite several applications that exploit RTcmix's TCP communication protocol there are no standard GUI control mechanisms providing a high degree of flexibility. As a result, most research remains confined to 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 yet still provides the ability to create complex configurations. As the name implies, GAIA's goal is to provide a rich platform for the creation of virtually unlimited new compositional interfaces.

I. Overview and Motivation

GAIA facilitates the creation of a wide variety of control structures (called widgets) on-the-fly. Widgets are created by left clicking the mouse and selecting from a menu of available types. Connections between widgets are made by clicking on an item's connection box (input or output) and dragging a line to or from another item's input or output box. Despite similarities with other popular interfaces, GAIA is an attempt to move beyond the scope of previous generation graphical network based applications. It is a completely open environment with an API for creating new widgets. Fully integrated into the RTcmix scripting environment, it also allows for both graphical and scripted control. GAIA is part of continuing development on the Portable and Semi-Portable Audio Workstation [2].

II. Design and Implementation

GAIA widgets fall into two general categories: objects specific to GAIA and those used to directly control RTcmix. All widgets have configurable options specific to their operation (eg., minimum and maximum value, string or floating point data, orientation, size, etc...).

GAIA is written in C and built with the GTK toolkit, running under Linux and in Alpha version under Mac OS X. Somewhat derivative of the Motiff API, GTK widgets 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.

The versatility of the GTK canvas has made several unique features possible. Widgets 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. The entire program window, or canvas, can be predefined to virtually any size then scrolled up/down or left/right. Similarly, multiple canvases can be created, all of which are linked in the same application or patch. Any given canvas can be made larger or smaller by zooming in or out. Specific widgets are also available to control any of these functions. For example, if a user wanted to trigger a screen change via MIDI footswitch message, the MIDI event widget is simply connected to the screen change widget.

RTcmix widgets communicate to a running RTcmix engine via a TCP socket connection. 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. Commands are identical to those issued in an RTcmix Minc scorefile. The main difference is that the command arguments can be changed within the GAIA interface before being sent out.

The following two examples illustrate some of the basic functionality GAIA offers. The first (**figure 1**) demonstrates a simple MIDI controlled Karplus Strong algorithm (RTcmix's STRUM instrument). The second (**figure 2**) is a stand-alone controller for multichannel polyrhythms.

Figure 1 shows three basic groups of widgets. In the upper right hand corner an RTcmix connection widget is linked to short script. When activated, they work together to either launch an instance of the RTcmix engine or connect to one already running (locally or remotely) then send it a few basic configuration parameters. It should be noted that the script widget can also be used to send any type of RTcmix scorefile, not just configuration data. The larger group of sliders and MIDI objects in the lower left hand corner represent the NOTE_ON section. When a MIDI NOTE_ON event is received, it's value is passed to a "scale" widget, which in turn passes on a value to the algorithm's pitch field.

The scale widget illustrates some of the flexibility GAIA offers in terms of widget

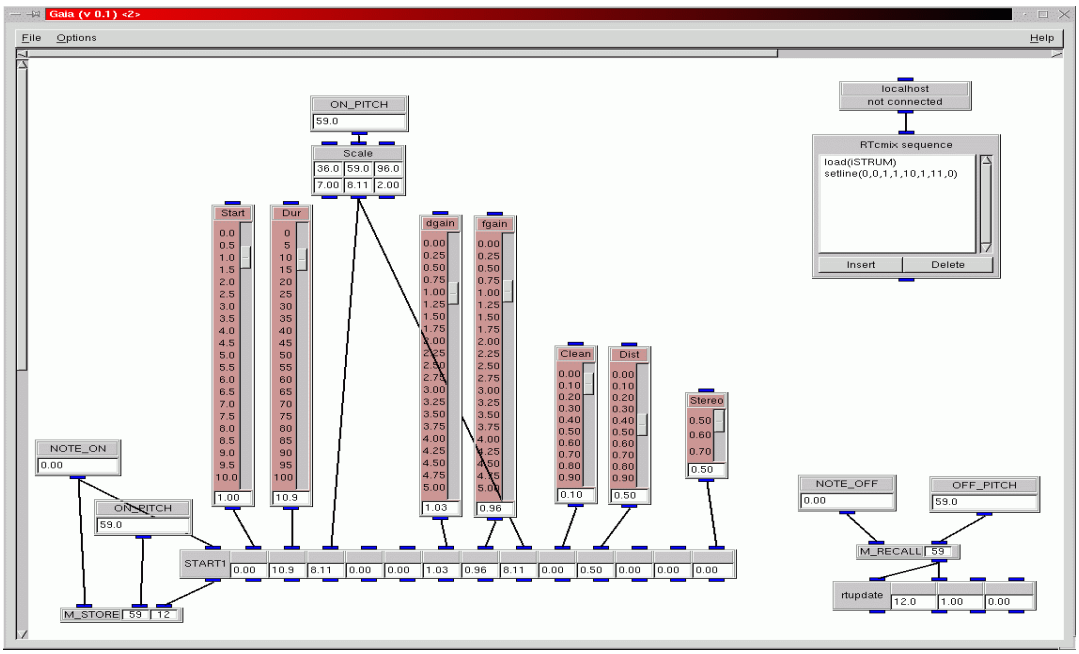

Figure 1.

design and functionality. When initially connected to another widget, a scale widget makes a query to determine parent and child min/max values and data type. It then automatically sets a local copy of these parameters. It is important to note here that the scale widget is reconfigurable in real time. The upper arguments are input minimum, input value, and input maximum. The lower arguments are the same, but for output. Output scale data in this example is represented in octave / pitch class, input scale data is MIDI.

All values have their own connection boxes and can hence be modified by other connected widgets. The scale performs a linear interpolation between 36-96 (note on range for a 61 key midi keyboard controller) and 7.00-12.00 (5 octaves = 60 notes). So, for example, to change the scale being used to 24 notes per octave, all that needs be done is set the output maximum to 9.06 (one octave and six semitones above middle C), instead of 12.00 (four octaves above middle C). Data type is also a configurable option for the scale widget.

The example also illustrates some of the configurable aspects of the slider widget. They can be set to specific sizes, increments, and orientations. The ones in this example are all vertical, with scale markings, set to "local trigger." When a slider is moved, a "trigger" event is send out to all its connected children. They receive both the signal and the slider's data value. Data triggering, the even sequence defined by a given network of widgets) can be set to activate any time a widget's value changes, or only when a widget is specifically activated / triggered.

The group of widgets in the lower right hand corner represent the NOTE_OFF section. Together they complete a multitimbral synthesis controller, the only limit to the number of consecutive running notes being processor speed. This is achieved via RTcmix's method of dealing with real time parameter updates [1].

The widget sending out the synthesis command is labeled "START1" (part of the NOTE_ON group) just as in an RTcmix scorefile used to control the instrument. The main output of this widget to other GAIA widgets is the note event number, starting at 1 and counting sequentially up to MX_CUR_NOTES which is set default to 100, then starting over at 1 again. RTcmix uses this index to update parameters. The ON_PITCH and note event number are stored in an internal array via the M_STORE widget. The information is then used in the third group of widgets located in the lower right hand corner. A NOTE_OFF event triggers the M_RECALL widget to look up which note event corresponded to the particular OFF_PITCH. This information is then passed in turn to an "rtupdate" command, which sets the value of piffled 1 (duration) to zero.

Figure 2 illustrates a control structure that does not rely on MIDI data. Instead, it uses GAIA's "timer" widget to send out two different pulses to an RTcmix

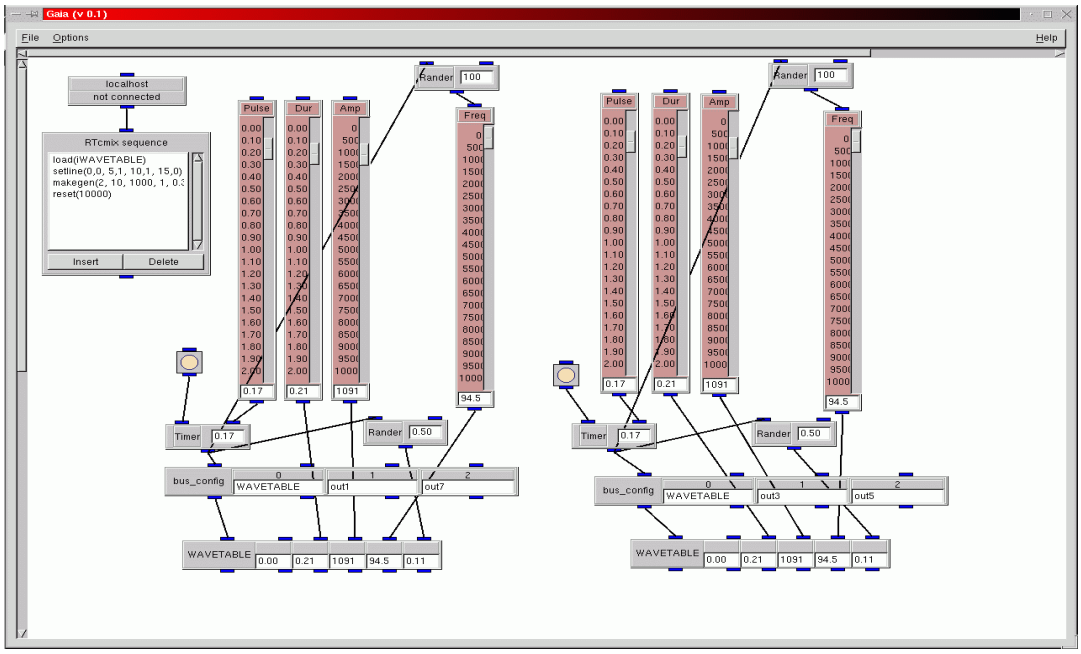

Figure 2.

WAVETABLE lookup instrument. Pitch and amplitude are randomized. Audio output is to four channels of an eight channel audio system. There are three basic groups.

In the upper left hand corner is an RTcmix connection widget and a script with some basic commands, similar to the previous example. The two groups of four sliders make up the two different rhythmic controls.

A "blinker" object controls the start and stop of a "timer", which in turn takes one argument, tempo. Double clicking the blinker starts and/or stops the timer. A slider is connected to the timer's input, thus allowing for independent tempo control. As in the previous example, the slider is set to "local trigger." Each timer is connected to two "randier" widgets, which generate random numbers between zero and X (the widget's input argument) when triggered. In this particular instance, these widgets control pitch and amplitude respectively. Between each timer and WAVETABLE command is an auxiliary RTcmix command, bus_confide. This sets the audio routing for the instrument in question. So in this case, one rhythm is being piped to channels one and seven (front right and left), while the other to channels three and five (rear right and left).

It should be noted that since connections are special widget types, GAIA can also support execution priority. Rather than only supporting a static left-to-right or right-to-left convention, GAIA allows the user to define which order connections for a given widget are executed. The default ordering is by creation. If connection 1 is made before connection 2, connection 1 will execute first. The change is made by placing the mouse pointer over the given connection line, then right clicking to change the connection properties.

GAIA makes use of its own MIDI parser. It can communicate with the Mediator MS-124w serial MIDI adapter (via raw serial communication) or the SO MIDI driver.

III. Widget Structure and API

As stated previously, GAIA is written in C. But in order to provide an easy to use API for creating new widgets, elements of C++ have been employed through C structures. The main hierarchy for GAIA structures is as follows.
The top level structure is called a "canvas object." It contains pointers to specific widgets as well as the graphical objects necessary for rendering and movement. This includes the graphical component of a widget'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 widget design. All widgets are signal objects, or groups of signal objects. The structure contains various pointers to specific widgets and their data. The signal object also maintains pointers to a widget'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 widgets themselves. As a result, it is the signal object that receives event triggering routed through the input and output boxes for a given widget network.

Creating a new widget is relatively easy. There are four basic functions that need to be defined: properties_window(), creation_function(), connection_function(), and signal_function().

The properties_window() function allows the user to configure the particular widget. An API is provided to help the programmer create a pop-up dialog box for the given widget.

Once the user enters the information from the above dialog, it is then passed to the creation_function() via an in_prams[] array. The creation function basically defines the main aspects of the widget, including any special purpose graphical objects or GTK structures, as well as input/output boxes. GTK supports a box packing mechanism [4] 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 widget objects and move them around the screen if need be.

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 widget's are connected together, the respective connection_function() is executed. This function allows widgets to make specific configurations depending on what they're connected to. For example, if a parent widget outputs floating point data the child widget 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 widgets. The scale widgets in the previous 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 widget'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 a widget is triggered, it sends out signals to every widget 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 widget'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 widget types and specific attributes.

IV. Future Direction and Conclusion

GAIA, while stable, is a relatively new application. A main area of work in the short term will focus around the creation of new widgets. There are many widget categories yet to be explored. The next release of the package will contain a complete list of mathematical and logical widgets. Graphics objects will also support a "mini view" where a widget or group of widgets can be minimized into a smaller, yet still completely relevant (eg., having all the input and output connections) widget graphic. Future releases will involve widgets that communicate with special purpose controllers outside the scope of MIDI (eg., raw peripheral data, computer vision systems).

RTcmix currently only supports one-way TCP communication. A logical step would be to make this bi-directional. This would allow for several new features, not the least of which is the manipulation of analysis data for resynthesis.

RTcmix has recently been ported to OS X. GAIA support for O S X is still in a very preliminary phase. The synthesis engine works well under the new operating system. Support for X windows based libraries like the GTK canvas, however, still involves some additional work.

As a whole, GAIA makes significant steps towards the creation of a programming environment for the generation of virtually unlimited new compositional interfaces. The ease of use the environment offers and its direct ties to the RTcmix scripting environment provide a useful interface for working with real time

synthesis on the Linux and Mac OS X platforms. It's open source API also provides a useful starting point for limitless future development and contribution from a wide variety of sources.

V. Resources

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

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

3. **The GTK reference manual:**
   http://developer.gnome.org
   /doc/API/gtk/index.html

4. **The GNOME interface reference library:**
   http://developer.gnome.org
   /doc/API/libgnomeui/book1.html

5. **RTcmix source code archive:**
   http://presto.music.virginia.edu/rtcmix/

6. **GAIA home page:**
   http://presto.music.virginia.edu/gaia