RTMix – towards a standardised interactive electroacoustic art performance interface

IVICA ICO BUKVIC

3346 Sherlock Ave #21, Cincinnati, OH 45220, USA
E-mail: ico@fuse.net
URL: http://meowing.ccm.uc.edu/~ico/

The following article offers an analytical overview of the currently available software technologies designed to assist in creation, dissemination, and most importantly performance of interactive electroacoustic art. By grouping the software into two basic groups based on their interfaces, my aim is to provide a comprehensive list of two groups' strengths and shortcomings, therefore exposing common issues that arise whenever a composer utilises such software interfaces in performance settings. Finally, as an incentive in solving a number of given problems, the author will present RTMix, his own software creation that has been designed primarily as a standardised interface for the purpose of easier production, performance and dissemination of the interactive electroacoustic artwork.

1. INTRODUCTION

Interactive multimedia art by definition encompasses any kind of artistic work that requires live interaction between the computer and a live performer. In such a symbiotic relationship between the inherently 'stupid', yet immensely powerful machine and the intellectually superior, but computationally less agile human performer, there is always a mediator, an *interface* whose purpose is to bridge these dramatic differences between the two participants, relaying pertinent information in order to ensure a successful interaction. Sometimes the *interface* consists of a composer who is running the computer, where the performer receives computer-related cues from the composer. At other times, the *interface* is perhaps a little more elegant, where the performer has a hands-on communication with the computer via the *MIDI* (Musical Instrument Digital Interface) pedals and other kinds of interactive controllers, or by having a visual display of the computer's activity and current state. No matter what the means of communication are, it is obvious that the *interface* is the most important element in the interactive art performance whose flexibility and reliability means the difference between a successful execution and a complete disaster. In the following article I will focus on this kind of *interface* in order to generate a list of desired features that would constitute its best possible incarnation capable of seamless integration into just about any kind of interactive art.

1.1. Background

Interactivity has played a very important part in the shaping of the electroacoustic medium ever since the first electronic music studios came into existence. Beginning with modular positioning of speakers in Stockhausen's *Kontakte* and Pierre Henry's real-time routing of the audio signal during the performance of his *potentiomètre d'espace*, Morton Subotnick's *Ghost Electronics* in works such as *Trembling*, as well as today's numerous installations and interactive concerts, such as recent *Telesymphony*, it is quite apparent that composers and audiences alike have always had a great deal of interest in this form of artistic expression. Although the curiosity was always there, it was not until very recently that this kind of artistic expression was avoided due to the sheer cost of the technology and the logistical support required for it to be successfully deployed in a performance of any kind.

1.2. Advent of portable computing

Perhaps one of the most revolutionary technological advancements to have spurred the development of the interactive electroacoustic medium was the advent of affordable supercomputing portable computers. Ever since their first introduction on the market, the so-called computer *notebooks* (a.k.a. *laptops* or *powerbooks*) have kept improving, becoming lighter, more powerful as well as more affordable, even upgradeable – many manufacturers, such as *Dell, Sager* and *Eurocom* offer laptops that can be upgraded nearly as easily as desktop computers. With *CPU*s (Central Processing Units) soaring over the 3 GHz (gigaHertz) mark, *Hyper-Threading* abilities (*Intel*'s concept of treating one processor as if it were two by the Operating System in order to maximise CPU efficiency and task queuing), *SSE2* (*Intel*'s Streamlined Single-Instruction-Multiple-Data Extensions) *3DNow!* (*AMD*'s set of multimedia instructions) and *AltiVec* (*PowerPC* processor's set of additional instructions a.k.a. *Velocity Engine*) vector-based float-point calculating engines, today's computers are capable of billions of calculations every second. In addition, computers are now becoming more and more modular, where each particular task is taken over by a dedicated hardware – video and graphics are handled by the powerful *GPU*s (Graphics Processing Units), sound is managed through high-quality external *DSP* (Digital Signal Processing) cards, while all other peripheral communications are managed with either dedicated on-board chips or add-in cards.

These portable workstations and their usability are the main focus of this article due to the simple fact that they are nowadays capable of tackling just about any computational task, and as such do not require any additional external hardware (apart from the sound-making equipment, such as the off-board soundcards for the purpose of limiting possible electromagnetic interference with the tightly concentrated hardware contained within a typical notebook) that would potentially limit their portability as well as affordability.

1.3. Interactive art performance interfaces

With such developments in the world of technology, a new market opened up with a need for software capable of harnessing newly acquired computing power. As a result, during the last decade of the twentieth century, several important software packages (or updates to the existing software) whose focus was on real-time multimedia data processing took shape, some of them becoming widely accepted standards among multimedia artists – *MIT*'s *Csound*, *IRCAM*'s *FTS* and later *Cycling 74*'s *Max/MSP* as well as Miller Puckette's *Pure Data* (a.k.a. *PD*), Brad Garton, John Gibson and Dave Topper's *RTcmix* modification of Paul Lansky's *Cmix*, James McCartney's *Supercollider*, and many more. With a strong response from the artistic community, these applications quickly gained momentum. Some of them were designed as open-source software (i.e. *PD* and *Cmix*) and continue to develop at a steady rate as stable cross-platform solutions, while others branched off and became closed-source commercial applications (such as the ubiquitous *Max/MSP*).

All of the listed software packages have one thing in common: they utilise the onboard *CPU*s for both their *DSP-oriented tasks* as well as the *interface*. While they are not the only available interactive art tools, they do make up the majority of this market in terms of their popularity and widespread use, and as such will serve as a basis for the definition of the two types of interfaces. Yet it must be pointed out that in addition to these, there are some very potent, but architecturally rather different kinds of interfaces that offer the same functionality utilising dedicated hardware rather than the built-in *CPU*s for DSP-related purposes. A typical example is the *Capybara* hardware that is accompanied by the equally powerful *Kyma* interface. Although their performance is rather impressive, their design does not fit into the scope of this article because they do not employ the computer's *CPU* for the sound-processing tasks. Rather they utilise computers only for interface purposes rendering them merely as expensive display adapters. Whereas this kind of solution was certainly one of the most popular ones, if not the only viable option during the early 1990s, it is now becoming less and less common since today's onboard *CPU*s are capable of offering nearly the same kind of computational power as these systems at a fraction of their size and cost. Their usually not-so-optimal size (and inherently their portability) coupled with their price tag render this kind of interface simply uneconomical. All these factors, as well as the fact that such interfaces are not required to use minimal amounts of the processing power (since the interface does not share the computing power with the *DSP-based* tasks) make any further comparisons between these hardware solutions and the previously mentioned software-only ones not only unfair, but more importantly impossible to present in an objective fashion. Hence, they will be excluded from the given pool of commonly used and generally widespread software interfaces.

Yet in spite of the sudden popularity of the interactive art medium, even today many composers and performers shy away from its potential, seeing it as being too hazardous, unpredictable, and too difficult to control in performance settings; with all kinds of last-minute problems of a technological nature, as well as the apparent lack of concert halls equipped for such events, the interactive electroacoustic medium has as many admirers as it has critics. Furthermore, due to such technical volatility, works are hard to judge since it is nearly impossible to separate the listener's impression of the technological implementation and the creation's artistic merit. For instance, a work whose performance suffers from technical difficulties may be better than that which is perceived, while a mediocre work's weaknesses can be craftily occluded by the impressive on-stage display of technology. To this day, the lack of a comprehensive, standardised and easy-to-use interface has made it not only difficult for composers to work within this medium without being hindered by technical limitations, but has also warranted a lack of transportability and performability, as well as stalled development of any kind of aesthetics upon which an interactive work could be criticised. So what is the source of this problem and what can we do to circumvent it? In order to answer these questions, I will take a closer look at the software performance interfaces currently in use by interactive multimedia artists, grouping them according to their features as well as the ways they encourage the user to approach the music-making process.

2. THE BIG 'TWO'

All of the aforementioned software packages have been designed with a particular premise and/or approach to creating and executing interactive multimedia installations, and therefore, just as with any software package, each offers its own set of strengths and weaknesses. Furthermore, each stimulates the musician to approach the music-making process from a different perspective: *object-oriented* applications, such as *PD* and its 'commercial brother' *Max/MSP*, provide musicians with rudimentary objects that when coupled together into an elaborate contraption can result in very complex interface designs. Applications such as *Csound* and *Cmix* tackle this issue with a more *linear* approach, where the so-called *scripts* or *scriptfiles* contain information as to how to manipulate available *instruments* and which are triggerable via a simple command-line interface. While these four software packages are far from being all-encompassing examples of the current interactive art software scene, they do represent the two extreme concepts of software interface design. Most of the other applications fall either neatly into one of these categories or, more commonly, are a combination of the two. A majority of the software released in the last couple of years usually draws upon both of these concepts, making this distinction somewhat arcane, perhaps in the same fashion as the ideas of *musique concrète* and the German *Studio of Cologne* styles are now known simply as the initial two approaches to electroacoustic music that have long disappeared in their pure form, and only remain for aesthetic, analytical and historical purposes. Therefore, the lack of the 'pure' presence of these two approaches in today's software does not negate the need to clearly establish a set of parameters that will help us define the two groups and help us further analyse and cross-compare their traits in order to better understand the problems of the current widely used software interfaces and issues surrounding their deployment.

For the purpose of further analysis I will label the two groups based on their interfaces as *object-oriented* versus *linear*, *visual* versus *script-based*, or as *modular* versus *standardised*. By offering a closer look at the features and functionalities of these two groups, I will supply a cumulative list of desirable characteristics that can further serve as a basis for a potentially all-encompassing, standardised and technologically less demanding performance interface that would supply an easier and more reliable relaying of the information between performer(s) and computer. With such an interface, interactive electroacoustic music would not only become more accessible, but would also cease to suffer from the problematic elision of the technological prowess and the work's artistic merit from a critical standpoint, since the technological aspect would be simpler to execute, and therefore less important in the overall perception of the work of art. To put it in practical terms, currently a large number of interactive installations are like gargantuan home-made contraptions put together with 'Scotch' tape, and whose operability itself is impressive enough, something that can potentially overshadow the fact that the contraption may not be doing anything remarkable, other than 'not break'. Through better, and more importantly standardised interface implementation, the contraption would become more of an instrument that could be, just like any other instrument in the orchestra, practised and perfected to the point that its utilisation could be considered just as virtuosic as playing a violin, or a piano. Then, the technology-inspired awe would be no greater than the appreciation of the architectural design of a piano during a performance of a solo piano work. Interactive art would then be perceived as a work of art, rather than a witty display of technology.

2.1. Object-oriented interfaces

Object-oriented interfaces, such as *PD* and *Max/MSP*, are also generally *visually based*, as well as *modular*. Many other software applications adhere to this kind of concept: *jMax*, *FX2*, *EyesWeb*, *Spiral Synth Modular*, *Glame* and *Vaz Modular*, just to name a few. All of these applications have a lot in common: a visually oriented and a highly customisable interface, a set of relatively basic objects, as well as some kind of hierarchically based means of interconnecting the *GUI* (Graphic User Interface) elements (or objects). I will focus primarily on *PD* and *Max/MSP*, as signature examples of this group.

Due to the object-oriented nature of this software, users have a vast amount of flexibility, being able to create interfaces of just about any type, size and functionality. Needless to say, this kind of interface flexibility leads to a greater artistic freedom – utilising such an interface, the user has complete choice over the design, order and arrangement of the various interface elements (i.e. buttons and sliders). This is perhaps one of the most powerful traits of this type of interface. Furthermore, by a simple incorporation of newly designed objects, the existing designs can be easily expanded to intercommunicate with just about any MIDI-capable interface, as well as any other type of interactive controller.

Yet another impressive aspect of this kind of interface is its modularity. Through a simple, yet powerful concept of cord-like connections between the objects, the information streamed from various sources can be easily re-routed and displayed as needed, regardless of its data type. Armed with such an array of features, these applications offer a practically unbeatable set of options to any artist.

However, with versatility comes another not-so-desired feature, and that is complexity. The majority of composers utilising this kind of interface opt to perform their own works, or at least run the technical aspects of the show, in either case having firm control over the execution of the piece. This performance practice dominates interactive electroacoustic art, and while found in other music genres, more often we find composers writing works for others to perform. At this point it is important to ask ourselves whether this kind of interactive art performance practice arose out of necessity or purely for artistic reasons. While there is no way of proving the dominance of either of these interpretations, there is no reason to dispute the observation that composers, by knowing their own creations the best from both the technical and artistic standpoints, and more importantly the custom-designed interface, tend to prefer ‘running the show’ themselves in order to minimise the possibility of something going amiss. If we agree with this observation, then it is easy to further conclude that there is such a thing as an ‘over-customisable’ interface that can potentially limit the performability aspect of the work in the settings where composer is not able to physically control the interface and/or relay information between the computer and a performer. Even if we try to imagine performers utilising such custom-made interfaces themselves (assuming, of course, that the performer is someone other than the composer), then we will have to agree that each such interaction would require the performer to relearn the interface’s modular layout. Such lack of standardisation produces an impression that each performance is like attempting to write a paper on a word processor that continuously shuffles the places and positions of the commonly accessed options. Arguably, we live in an age of ‘globalisation’ where common computer users are becoming more and more computer literate. Still, it would be hard to argue that playing a violin that continuously changes a number of strings, as well as tunings, would be an easy feat in any respect.

Due to the fact that these kinds of applications usually embed the performance interface into the instrument itself, interface users are often facing the on-screen clutter, the objects that are a part of the instrument’s inner workings but have no useful purpose, being visible during the performance. This is generally a significant problem with most of the applications that fall into this group. However, some of the software, such as *Max/MSP*, have provided in their later updates tools that furnish users with means of occluding information that is not important to the performer. Even though these improvements are certainly noteworthy, they still do not solve the core problem of this group, and that is the lack of standardised performance interface.

In terms of robustness and the *CPU* footprint, both of the given interfaces do not fare the best. Their *CPU* utilisation vastly varies depending on the complexity and size of the interface itself. As such, they perhaps take away too many computing cycles that could otherwise be used for the most important aspect of the performance – the sound itself.

2.2. Linear interfaces

The most characteristic feature of the second, *standardised* group is the lack of a visual interface a.k.a. *GUI*, which is usually supplemented by the command-line-based interface. Again, a number of software packages utilise this sort of interface, such as *Csound*, *RTcmix*, *LISP-based CLM*, *SAOL-based* packages, *STK*, *Squeak*, *Jsyn*, etc. The two applications I will use as typical examples of this type are *Csound* and *Rtcmix*.

Most of the software that falls into this category stems from the era of Unix dominance (which is arguably returning with the increased *Linux* Operating System’s popularity as well as Apple’s recent choice to base *OS X*’s architecture on the *FreeBSD-based* kernel) and are therefore predominantly command-line driven. The command-line calls are naturally standardised and, hence, this kind of an approach exemplifies less complexity and an easier learning curve from the aspect of end-user/performer. Its utilisation, however, does require a certain amount of end-user’s *OS* (Operating System) literacy and therefore creates an apprehensive environment for the performer who may not be familiar with the *OS*, the command-line interface, or both. Some versions of these applications do circumvent this issue in a relatively graceful fashion, such as *Csound* distributions with a simple *GUI*, where all of the operations are managed with ‘point ’n’ click’ elegance. Yet, this kind of visual supplement to the core application’s concept should be regarded as an ‘after-market addition’ since it does not represent a typical scenario within this group, nor is it usually geared towards real-time performance, therefore making it an irrelevant aspect within the context of this group.

The command-line design consequently requires instruments’ inner workings to be stored in the so-called *scorefiles* or any other types of *scriptfiles* where they are hidden away from the end-user. This is very convenient for the performers who do not wish to be bothered by the instrument’s architectural design and are only expected to trigger the actual event as indicated in the performance notes. Still, the interface’s lack of visual stimuli leaves a lot to be desired, especially in critically timed situations where asking a performer to trigger an event is hardly practical – after all, the interface itself usually does not supply any timing information. This kind of compositional approach can thus create an array of problems as to how to synchronise the computer-based and ‘live’ elements of the work. Furthermore, such limitations of these interfaces make them a less popular choice for composers of interactive art, a fact that is rather unfortunate considering their immense flexibility and versatility.

Since both the entire instrument and the performance are stored in a linear format (*scriptfiles* are constructed using text, which is linear by nature), most composers utilising this kind of an interface end up being motivated to compose in a linear, or streamlined fashion compacting their whole performance into one *scriptfile* that then becomes a focal place of their work where all of the timing information is internalised. All of the processes contained in such a script are timed relatively with respect to each other and once the script has been executed, it takes on an irreversible path continuing until the process ends or is prematurely terminated. Needless to say, such a design imposes certain strains on real-time performance due to its rigidity. Performers have very little margin for error and rehearsals are often managed as if the work were written for a ‘tape-and-instrument’ medium. In addition, this event co-dependency makes it nearly impossible to play back only a snippet from the middle of the script, further complicating rehearsal coordination. On the other hand, splitting up the *scriptfile* into several sub-scripts in order to alleviate this limitation sometimes is simply not possible since there may be a required co-dependency between events that cannot be implemented if each script becomes an independent process. Consequently, these two factors seem to be the greatest limitations of this kind of an approach to interactive music composition and performance.

While a linear approach does have its shortcomings, it does have several very important advantages. The first one is the fact that the time-based flow of events is generally rather easy to track when looking at the *scriptfile*, and is therefore relatively useful for the purposes of scoring, or storing the work in some other legible format. The second one is its standardised (albeit unfriendly) user interface that requires no learning curve beyond the initial familiarisation from the performer’s aspect (especially when performing different works utilising the same interface). Finally, this group’s interface exhibits obviously minimal *CPU* utilisation (since in essence it has no visual elements that require real-time updates).

3. SHADES OF GREY

Certainly not all software interfaces fall neatly into these two categories; a vast number draw their inspiration from both groups. For instance, applications like *Buzz*, which is a cross-breed between a tracker and an object-oriented filter system, carry traits of both of the groups – on one hand, *Buzz* resembles a Max-like object-oriented design, while at the same time it also has a script-based tracker which is strictly concerned with linear ordering of the events; *Supercollider* is a powerful scripting language (based on *Smalltalk*) that also has a fully customisable *GUI*, as well as a scripting language that is, contrary to other typical script-based interfaces, object-oriented. Even *Rtcmix* and *Csound* have several different *GUI*s designed to harness their power via a visual interface, such as *Cecilia* and *Visual Orchestra* for *Csound* and *Soundmesh* (a.k.a. *Internet Sound Exchange*) for *Rtcmix*. Yet, the presence of such software does not help us decipher the plausible elements that would constitute the ultimate standardised interface for interactive electroacoustic art, since all of these again have their own shortcomings which are hard to codify because of their unclear categorisation in respect to the aforementioned two groups. Hence, these applications will not be taken into account when attempting to draw parallels between the two inherently different approaches to constructing an interactive art performance interface.

4. DRAWING PARALLELS AND CONNECTING THE DOTS

After having a closer look at the two given groups, we can now more easily discern and compare their individual advantages as well as common disadvantages. This information can further help us propose an optimal standardised interface for the performance of interactive electroacoustic art. Standardisation is an important aspect as its presence would encourage easier dissemination of works, as well as an increased chance for repeated performances and hence a greater exposure. In addition, because of its standardisation (and consequently potential wide use), composers using such an interface would not have to worry as much about technical problems or feel compelled to take an active part in the performance, but rather be able in a more traditional fashion to simply sit back and enjoy their own work.

Through comparison of the two groups we can observe the following: while the object-oriented interfaces offer customisability, by default they lack in providing a legible timeline for the work. The linear group offers an easily decipherable flow of events stored in so-called *scriptfiles* (although this information is generally occluded at run time), but does not lend itself to easy debugging or partial playback. The visual interface of the first group is certainly more advanced than that of the second, but it requires more computing power and its customisability warrants the lack of standardisation and therefore proportionally increases the learning curve for the end-user, having furthermore an inverse effect on the transportability and performability of the pieces. The second group has its own drawbacks in terms of the accessibility of the command-line-based interface implying an additional knowledge of both the *OS* and the application in question, but at the same time offers standardisation which should generally warrant an easier learning curve from the performer’s standpoint (again, assuming that the performer and composer are not the same person).

What is peculiar to the majority of the discussed interfaces, regardless of the group they belong to, is their exclusivity – the fact that very few of the listed software interfaces are designed to cooperate between each other, sharing and routing the processed data. This limiting factor simply restricts the composer’s imagination, having their creativity subdued by the technology it employs. While there are ways around this issue, such as routing audio data via network sockets, and intercepting it with a different application, this is neither elegant nor a commonly used solution to this problem. More often, composers simply choose not to bother with this kind of functionality, rightfully being worried about potential technical problems that might arise from using such a complex set-up. A side-effect of such a design is the fact that software interfaces tend to take over the sound and video resources so that they cannot be shared simultaneously among a larger pool of applications.

Finally, due to the overall complexity of both approaches, many of the interactive works become extremely difficult to perform properly or are too expensive to produce, lacking the replayability factor, whereas in those not-so-common instances where complex pieces are well executed and where technological prowess shines, the audience is often so impressed by the technical elements of the work that it becomes hard to make a sane judgement about the artistic value of the work. In order to thwart this kind of flux between art and technology and furthermore to simplify the process of artistic creation, and even more importantly performance, the artistic community needs to have more easy access to a comprehensive and easy-to-use interface that will shift the focus from the technology to values of greater importance, such as the overall artistic merit of one's creation.

While all these observations may appear to be overly critical, it is important to emphasise that it is not the author's intention to scrutinise the existing pool of applications rendering them as inadequate, but rather to provide an insight into the desired features of a software application that would not only provide a user-friendly environment, where the composer's flow of thought would not be pulled away from the artistic creation, but also that would provide for a comfortable performance interface in settings where the composer's physical presence and technical support would not be feasible. In addition, this kind of insight could provide us with an additional set of desirable features that a universal interactive electroacoustic art interface could and, more importantly, should have.

By now, it is obvious that neither of the two available groups provide for the best environments for interactive art. And while one can easily argue that the creative process is a very individual issue and presumably not up for discussion, I would like to propose a set of guidelines that would provide for a universally better interface and offer the following benefits: easier learning curve, driven by a linear and easy-to-debug scripting language capable of triggering external applications and events regardless of their type and/or origin; a networkable interface for the purpose of coordination of larger performing forces; software with a minimal *CPU-utilisation* footprint; powerful standardised *GUI* that would offer strong visual stimuli for both compositional, rehearsal and performance purposes, yet not overwhelm the performer with unimportant information; expandability; portability; and an interface requiring minimal knowledge of the computer's inner workings, as well as usefulness beyond its original purpose. Furthermore, there are several additional desirable features that could help the overall perception of an interactive work by improving the aesthetics of the presentation, such as having the interface residing on-stage, where its presence could add a physical dimension to its otherwise bodiless sonic presence, as well as potentially the role of an equal in the overall sound-making process by becoming a virtual performer and a visually noticeable participant of the ensemble. This kind of an interface would also allow the performer to have a direct interaction, without a middle-person or an arbiter (such as is the case with many of today's interactive artworks). Interaction on this level would allow the interface to be more integrated into the performance, providing the audience with a much more rewarding visual experience. Finally, the interface should be designed in such a way that it ought to avoid the redundant re-implementation of existing software capabilities, and rather pose as an all-inclusive unifying tool that would furnish the user with greater freedom by allowing simultaneous usage of various unrelated processes.

Motivated by the above-mentioned problems and aesthetical choices, most of which were a result of first-hand experiences, it was my intention to design such an interface that could be used for interactive multimedia art and whose design would be focused on tackling issues related to the deployment of such artistic creations. *RTMix* (named after the *RTcmix* real-time audio manipulating and synthesis scripting language that has been used predominantly in my recent interactive works), designed primarily as an open-source project for the Linux platform in Autumn 2001, has since grown to be an acknowledged interactive art solution among the Linux audio community, and continues to be developed under the sponsorship of the University Research Council Grant of the University of Cincinnati, as well as a part of the author's Dissertation project.

5. RTMIX TO THE RESCUE!

RTMix is designed to serve as a fully fledged performance and composition interface. While it does not offer perfect solutions to all of the aforementioned problems, it certainly addresses a significant number of them by allowing the user to shift focus from the technical hurdles of performing an interactive work to more plausible artistic elements, providing a standardised interface, easier learning curve, as well as the coexistence of different sound-making and *DSP* software applications.

5.1. The interface

RTMix's *GUI* was designed using a *GPL-licensed* cross-platform *Qt* toolkit and is in essence a large, easily visible stopwatch, which is the timing centre of all events that unfold during the performance. The stopwatch is split up into the main timer and the countdown clock that is used to foreshadow strictly timed events, so that the performer can be prepared for the oncoming attack. The interface is enhanced with large mock-lamps whose sole purpose is to grab the performer's attention during rehearsals and the actual performance when this is needed. RTMix is also fitted with a 'panic' button that can instantaneously kill all of the currently active processes and provide quick cessation of a sound in the case of a feedback or a similar technical problem that requires a quick reaction. Furthermore, the 'settings' menu enables the user to customise the appearance of just about every aspect of the *GUI*, including visual stimuli. However, this customisation does have limits that have been imposed in order to keep a certain level of standardisation. There are additional resizable 'warning' windows (up to four per client) that can serve the same purpose as the main timer but offer greater flexibility with screen positioning, as well as more comprehensive routing of the cues to different performers. The interface also has up to four metronome windows which are fully customisable in terms of displaying complex metres by coupling metre subdivisions into horizontal groups, as well as presenting a wide range of tempi. Other external windows are also available, such as mirror representations of main timers (for dual-screen uses) as well as a custom widget with six additional generic visual stimuli. All of these objects are guided by the so-called 'event-script' that can be edited either within the application or by any other simple text editor. The syntax for the 'event-script' resembles Paul Lansky's *RT* script where each event has its name and is followed by a list of parameters that are blocked off with parentheses. There are currently twenty different types of events that the interface understands and all are geared towards triggering different events at different times, as well as controlling the visual interface and the flow of time.

5.2. Linear scripting language

The 'event-script' is built upon the premise that it must provide the user with the greatest possible amount of error-logging information, so that editing can be as simple as possible. Linear scripts are inherently hard to debug and therefore the 'event-script' allows only one event per line in order to minimise the parser's confusion with error reporting which is often found in bracket-dependent and multi-line-command scripting languages.

The events that populate the 'event-script' can be grouped into several subcategories: transport control events, probability events, metronome events, text events, network events, and real-time events. Transport events focus primarily on the basic transport controls, such as fast-forward, reverse, play, pause and stop. Coupled with them are so-called Checkpoints whose purpose is to index the performance in order for the user to be able to cycle quickly between them. Events are basically an array of system calls that can be triggered in various ways. Some of them are simple events that are executed immediately when called, while others are 'countdown' events that actually have preparation time before their execution. The benefit of having such events is rather obvious when it comes to synchronising live performer with computer I/O (input and output), regardless of whether it is a tape or interactive work. Probability events are based on chance numbers and can be used in various ways to toggle events or alter their various aspects and characteristics. This introduces an element of indeterminacy that can be extensively harnessed through this kind of interface. Metronome events pertain primarily to metronomes which are sometimes needed in live settings when the timings are rather complex and performers require additional help in keeping in time with the other performers (whether that be a computer or another live performer). Text events are for information purposes, to notify composers and performers alike of the incoming events, as well as subtle changes that are otherwise hard to communicate via any other visual stimuli. Network events are responsible for negotiating client–server connections as well as adjusting local networking settings, while real-time events are potentially all other events with a real-time flag switched on. Such events can be triggered at any moment, resulting in an immediate reaction. Many of these objects lend themselves to an array of secondary uses. For more information about the available commands, see the table.

Each event has a set of mandatory and optional parameters. Mandatory parameters depend on the event, but generally all events require exact timing (or in the case of real-time events, a binding to a real-time trigger). Optional elements add text notification and visual stimuli functionality. This means that most of the events that do not evoke sound processes are by default mute in terms of visual (or any other type of) notifications, while event-evoking commands have preset ways of notifying the user. All these behaviours can be altered as the user deems fit. With this kind of an approach, the software offers a default, easy-to-use environment, but at the same time lends itself to modifications by advanced users, while providing a level of flexibility that does not compromise the overall interface's standard.

5.3. Intercommunication and networkability

With these basic building blocks, and the fact that the interface is utilising system calls to just about any kind of event, one can create a very complex collage of events that range from regular sound playbacks to triggering complex scripts via *Csound*, *RTcmix*, *GUI-less PD* session, or any other sound-making application. What this means is that the application itself does not generate any sound, but rather relies on other external

Table. List of currently available events.

ID#	Event name	Description
1	Event	Makes system calls to invoke other applications at a given time.
2	Countdown	Initiates a countdown process, triggering an event upon its completion.
3	Text	Displays text notifications in the notification window at a specific time.
4	Checkpoint	Marks a place in the timeline for easier navigation purposes, as well as for Jump, Rew, and Ffw objects.
5	Pause	Pauses the performance.
6	Stop	Stops the performance.
7	Clear	Clears the notification window.
8	Warning	Similar to Countdown, except that the event is initiated immediately, followed by the countdown process.
9	Metronome	Sets up the metronome parameters (metre, tempo).
10	Jump	Jumps to a particular place within the performance – it can be either relative or absolute.
11	Randomise	Randomises a value and stores it into one of the slots of the 256-value array.
12	Change	Changes a particular parameter using a given equation and (if applicable) one of the values from the array.
13	Seed	Seeds the randomisation process.
14	Rew	Rewinds arbitrary number of checkpoints.
15	Ffw	Fast-forwards arbitrary number of checkpoints.
16	Togglemetro	Toggles metronome activity and/or resets it.
17	Assign	Assigns a specific value into one of the slots of the 256-value array .
18	Togglert	Toggles triggerability of the real-time events.
19	Setnet	Sets local or remote networking options (i.e. IP and Port values).
20	Togglenet	Toggles networkability of the local client.

applications that can be triggered with automation flags and therefore set into motion instantaneously. This certainly introduces the first drawback of such architectural design, and that is the issue of latency in instances where it takes time to instantiate the application (this is certainly the case with *PD*). But surprisingly enough, many other applications have instantaneous response, especially in the *Linux OS* which is currently the most efficient multitasking platform available. In addition, nearly all of the *Linux-based* applications can be triggered via command-line calls, and as such the majority of them are compatible with *RTMix*.

Being able to control a plethora of various unrelated events, *RTMix* maximises the versatility of a portable computer while minimising the implementation of redundant elements into its interface. It also enables the user to combine sonic processes from various applications, thus removing the currently existing limitation of using only one application at one time – *RTMix* is able to serve as a front-end for just about any kind of software and hence offers the badly needed standardisation of interface for just about any kind of interactive art performance.

Networkability is yet another powerful feature that enables each client to control multiple ‘slave’ clients, and therefore coordinate larger performing forces that are not able to cluster around the single display screen. In this setting, the master client simply emits the events to the desired client, which upon receiving the event immediately interprets and executes it. With such a design, it is easy to envision a vastly greater applicability of this interface, even in situations that are not associated with the music-making process, such as coordination of stage lighting.

5.4. Applicability and other benefits – the good . . .

The *RTMix* interface has been designed placing emphasis on ‘user-friendliness’. Most of the options contain their own clearly annotated and easily accessible ‘point ’n’ click’ GUI counterparts. The main window has the bottom half dedicated to a series of tabs whose functionality is rather self-explanatory: ‘Performance’ tab (with all of the text notifications), ‘Editor’ (for file input and output options, as well as error logging and text editing), ‘Network Settings’ tab (for communication with other *RTMix* clients via the TCP/IP protocol), and the ‘Real-Time’ tab (where the user can review controls and settings for the real-time events that can be triggered at any given moment). (NB: Since the application is still under heavy development, the actual layout stated above reflects only the current version 0.52.)

The other advantage of this system is the compactness of the ‘event-scripts’. The event-scripts only contain the scripting text that is compiled at the beginning of the performance and stored into the doubly-linked data cue. This offers greater transportability (with the assumption that the receiving client already has all the sounds and events to be played out), and in errata-like situations, simple adjustments, annotations and timings can be easily e-mailed to the recipient or performer of the work.

Given that the interface utilises the open-sourced *Qt* toolkit and no platform-specific code, it should theoretically work on any Operating System that *Qt* toolkit supports (currently *Linux*, *Mac OS X*, as well as any other *Unix-based OS*, and *Windows*). Therefore, the choice of the *Qt* toolkit makes *RTMix* compatible with the three most dominant platforms on the market.

Applications of *RTMix* take many forms. From being a composition interface, to a performance and coaching interface, to non-musical event triggering and coordination, *RTMix* offers a unifying platform for a variety of uses. Practically any array of events that require timing of any kind can be triggered utilising the scripting language (even such preposterous tasks as using the interface as a robust alarm clock). *RTMix* offers up to one hundredth of a second resolution accuracy and even in settings where the flow of events is supposed to be randomised, it is possible to create such script through the use of random number generating routines and real-time jumps between checkpoints. This enables users to free themselves from strict timings, making *RTMix* an equally adequate solution for aleatoric sorts of artistic expression. Furthermore, the real-time events (whose listing is occluded from the user in the performance setting, yet easily accessible and annotated on the 'Real-Time' tab) can be used to provide an even more indeterminate event-triggering mechanism in situations where the timing mechanism may appear to be too constricting.

RTMix in interactive settings can therefore pose as a mediator between computer and live performer or computer and a group of performers. It is a rather affordable solution in terms of *CPU* utilisation, enabling users to use the majority of the computing power for the important sound-making events. In addition to its default purpose, *RTMix* can be used in purely acoustic settings as the synchronisation tool between players with difficult parts or even allowing conductors to have a finer control over the tape or some other interactive aspects of a particular work. Through careful use of a metronome and visual stimuli (that are large enough to be perceived through peripheral vision), the amount of distraction is brought down to a minimum, while enabling performers to better synchronise with each other. The checkpoint events can also be used for the purpose of more efficient utilisation of rehearsal time. Even non-interactive electronic music can also benefit from its use since it can be utilised as a simple mixer, as well as a precise timer between the tape and (if applicable) live part. Moreover, any combination of the aforementioned media can also benefit from its timing mechanism which would warrant greater coordination between the parts. Finally, due to its unique design, this interface encourages its presence on the stage, therefore being a perfect solution to the desired set of aesthetic improvements suggested by the author.

RTMix is also a composition tool where final assembly of all previously designed processes can be combined into a final collage for fine-tuning purposes. In this sense, *RTMix* serves as an object-based (yet linearly ordered) counterpart to both linear and object-oriented languages and processes. Composers can more easily troubleshoot their creations and run only the troublesome sections due to the fact that the work has been subdivided into more manageable and self-enclosed events and/or sections.

Finally, *RTMix* can be used for coaching purposes. The importance of its coaching capabilities is immeasurable since to this day many interactive works require elaborate set-ups and usually the presence of the composer, and in situations where the composer's attendance is for whatever reason impractical, complex works requiring professional and knowledgeable assistance simply end-up being dropped from the repertoire or performed poorly at best. With *RTMix*, it is now possible to conveniently e-mail annotated 'event-scripts' to the performer (as well as any other personnel involved in the production of the performance) who, after uploading the newly acquired script, would rehearse the work while being tutored and warned by the newly added annotations of possible pitfalls, difficult sections in the piece, as well as anticipating important downbeats.

5.5. . . . and the bad

While *RTMix* addresses many of the problems associated with standardisation of the interactive electroacoustic art performance interface, it is not perfect. Just like any other software, it has shortfalls and weaknesses, some of them being simply unavoidable. For instance, the fact that once the system call has been executed and an external event has been evoked, *RTMix* has no further control over it (the so-called 'runaway process') other than killing it, is certainly is not a trivial one. There is unfortunately nothing to be done about this aspect of the application since it would otherwise need to be aware of each triggered application's inner workings in order to make it capable of changing the other software's behaviour, something that is simply impractical, if not impossible to implement. In addition, there is the aforementioned issue of the polling of signal sources into one data stream before sending this to the *DSP* chip for output. In order to achieve low-latency operability, most of the software applications were designed to take over the *DSP* hardware exclusively and therefore virtually block any other software from accessing it. This is a known limitation of a long-standing champion of this kind of implementation, the *ASIO* (Audio Stream Input/Output) standard. On the other hand, if access to the hardware were to be shared, the latency would often need to be compromised. Neither of these options are an acceptable solution to the needs of the *RTMix* interface. However, this issue is gradually being addressed by the new *OS*s; both *Windows XP* and *Mac OS X* have nearly solved this problem with completely rebuilt audio driver implementations that allow polling of the DSP resource while supplying *ASIO-like* latencies. The *Linux OS* audio-developers community has approached this issue with an out-of-kernel solution, using a *JACK* sound server. Although still in development, it currently has the greatest potential for solving the above-mentioned issue because it not only allows extremely low latency but in addition allows inter-application transportation of the audio signal, therefore posing as an extremely efficient software patch bay. (Based on information obtained from Paul Davis, one of the leading *JACK* developers, the process itself theoretically does not add any more latency to the system than what the extremely efficient *ALSA* drivers are able to supply, which is roughly 2.6 milliseconds on a somewhat outdated dual Pentium II 450 MHz). Still, despite the great strides in system and hardware advancements, software applications have yet to embrace the new standards. Given the current circumstances surrounding this issue, it is perhaps safe to assume that the problem will resolve itself, and therefore *RTMix*'s framework is not one that should be amended.

5.6. Future

In order to attain even greater flexibility, the *MIDI* protocol will be implemented in one of *RTMix*'s future incarnations, where practically any of the available events will be triggerable via a *MIDI* controller. With such an addition, users will be less bound to the immediate presence of a laptop keyboard (which is the default real-time triggering mechanism) and physical contact with the computer, which will in turn add more elegance to the on-stage interaction between the computer and performer(s).

The 'event-script', while being as user-friendly as the text interface permits, is still not the optimal working environment for a composer. Hence, there are plans for the future version to incorporate a streamlined *GUI* used for adding, editing and removing of the events. This way, 'event-scripts' will be automatically generated through utilisation of the *GUI*, while the advanced users will still have accessibility to them in their text-form and an ability to easily edit them by hand if they desire to do so.

6. *RTMIX* IN ACTION

RTMix has so far been utilised in two of the author's works: *SlipStreamScapes III: The Sea* and his more recent *SlipStreamScapes V: Lullaby*. Both of these works employ multiple *RTcmix* scripts, as well as playback of live-recorded and other samples stored on the computer's hard drive – often having a mixture of these running concurrently. Due to *RTMix's* small footprint (less than 3% of CPU utilisation on a 1 GHz Pentium III laptop), most of the resources were successfully deployed over the multiple instances of scripts running simultaneously. In order to attain the ability to playback multiple audio streams from different processes, only two available CODEC pathways in combination with the *KDE*'s artsd sound daemon were utilised on a consumer-grade ESS Maestro 3i soundcard at 16-bit 44.1 kHz sound resolution. In the following section, I will focus on my latest creation *SlipStreamScapes V: Lullaby* in order to provide a more detailed description of *RTMix*'s deployment in a real-time setting, as well as to assess the repercussions of its integration.

6.1. *SlipStreamScapes V: Lullaby*

Lullaby is a work for piano duo and computer that draws from *Modal Pulse* and *Minimalist* aesthetics. The work also carries a strong programmatic element, describing one's soul transcendence from reality into a state of deep slumber. The role of the two pianos is tipped in favour of the first piano part. The second piano part compensates for the lesser amount of activity by also requiring the performer to interact with the computer. In order to further explain the interaction between the computer and the two pianos, I will present a quick overview of the underlying process that is assigned to the computer part as well as the work's overall structure.

The computer utilises *RTMix* as a front-end to a series of *RTcmix* scripts. The scripts are designed in such a way that they process the two pianos separately. The only process that affects the first piano part is the author-designed instrument called *MOCKBEND* (based on Doug Scott's *TRANS*, and John Gibson's *TRANSBEND*). The purpose of this instrument is to bend the pitch of the incoming audio feed in real time. Thus, the first piano part through interaction with the computer becomes a two-part texture built from the real acoustic and surreal textures, the second being a result of the acoustic piano being processed through the aforementioned instrument. The second piano is processed through relatively simple *REVERB* and *PANECHO* instruments. Both feeds are further passed through a *COMPRESSOR* instrument that ensures that the outgoing audio output's amplitude will remain within reasonable limits. The routing of the audio signal is achieved within *Rtcmix* through utilisation of built-in *AUX* buses. All of the instruments are often running concurrently, having multi-layered instances of the same instrument.

The form of the work resembles a simple arch-like shape, the ascent being a little longer than the release. The build-up is constructed as a mostly solo section by the first piano, where an introductory pitch-bend process provides a misleading impression of a honky-tonk piano. As the texture has been fully established, the second piano enters with the background chord texture that is being reverberated and echoed, while the first piano's part begins to be complemented by the 'pitch-bent' mirror image. The bending of the pitch proceeds in a sinusoidal fashion, always revisiting the zero-bend position (equiv. to the null position on the sine curve), while becoming faster and moving farther away from the originating pitch (a sine curve with an increasing amplitude). The arrival to the climax of the work (that in programmatic terms corresponds to the moment of transcendence into the realm of dreams) is signalled by the maximum density of the 'pitch-bending' process that creates an impression of FM synthesis of the inbound first piano audio signal, as well as by phasing out of the second piano texture, which is replaced by the probability-based additions of 'pitch-bent tails' that originate from the first piano's texture. The 'tails' are structured to be more densely populated the closer they are to the climactic point, while they become softer and sparser as the piece draws towards closure. These computer-determined textures are unique to every performance and they are created by selecting a random deviation from the originating pitch of the first piano part. Immediately following the introduction of the 'tails', the second piano rejoins the texture, but this time complementing the first piano texture with its dry, unprocessed ostinato pattern. As the piece approaches the end, the second piano and the computer parts slowly fade out, leaving the first piano to conclude the work.

6.2. Assessing the damage (or lack thereof)

As can be observed from the structural overview of the work, there are both strictly timed and more aleatoric sections found in *Lullaby*. *RTMix* was utilised in both of these situations without any apparent problems. During the dress rehearsal, the only thing that needed to be adjusted software-side was the audio-level coming out of the laptop. The hardware utilised was a *DELL Inspiron 8000* with a 1 GHz Pentium III processor, using Mandrake 8.1 Linux OS. Maximum CPU utilisation measured during the rehearsal was approximately 40%, 0–3% of that being attributed to *RTMix*. Furthermore, there was a noticeable increase in rehearsal efficiency when compared to the initial performance of the *SlipStreamScapes III: The Sea* that utilised an early command-line-based version of *RTMix*. The strictly timed attacks were executed with greater ease due to implementation of the timing and countdown mechanisms, as well as visual stimuli that helped both performers anticipate oncoming critical events, while causing minimal possible amounts of distraction.

The piece was favourably received by the audience, the most impressive aspect being the seamless integration of the acoustic and processed counterparts. This was done by amplifying a portion of the acoustic signal and feeding it through the speakers in order to create the same 'space' for both sonic sources. To date the work has received three performances, the first at the College-Conservatory of Music, University of Cincinnati, the second at the Indiana University, and the third as a part of the *OCEAn* 2002 conference in Oberlin, Ohio.

Apart from the gratifying artistically based criticism, the reaction of the pianists involved in the performance was favourable in that the piece did not require large amounts of custom notation, which in turn allowed for a faster learning of the work. The interface was also very easy to deploy since all of the on-screen controls were visually very accessible as well as mostly self-explanatory, while interacting with the *GUI* during the rehearsals and performances was further simplified by the fact that all of the on-screen controls have their respective keyboard accelerators (i.e. the Play button can be triggered either with a mouse click or with a simple press of the 'CTRL+ArrowUp' combination of keys). What was perhaps most impressive about these three performances was the noticeable lack of the performers' apprehension towards the interface. Because the interface was standardised, provided transparency to the actual computational process taking place during the show, and also offered a number of integration tools for rehearsal purposes, the pianists adapted rather quickly to the new interface. Moreover, the second and third performances were clearly the easiest deployment of the interactive interface in my career – both pianists, being acquainted with *RTMix* and its functionality were able to perform the work with an unmatched ease. While the three performances certainly cannot be seen as a trend in themselves (obviously more performances are needed before this kind of observation could have any reasonable weight), the comments received by the performers involved in the performances who had no prior knowledge of *RTMix*'s existence, point to a fact that this interface caused less apprehension than other currently available interactive electroacoustic art performance interfaces. Finally, no interface-related problems were encountered during any of the performances, and there were no audible 'glitches' during the sound playback.

7. NICE, BUT WHERE DO I GET IT, AND HOW MUCH DOES IT COST?

RTMix is a free (as in 'free beer') open-source cross-platform application (distributed under the GPL licence) that is currently available for download from both the author's website and the College-Conservatory of Music Center for Computer Music website (a.k.a. (CCM)^2). It has been designed primarily to offer interactive multimedia artists greater freedom of utilising different sound-making and processing applications at the same time in a coordinated fashion, while providing a streamlined interface for both the needs of composers and performers.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my wife Anamaria for her kind, loving support and understanding, as well as my newborn son Sebastian who has been one of the greatest inspirations in my life. Secondly, I would like to thank my parents who through their hard work and sacrifice have provided me with the opportunity to obtain a high-quality education. I would also like to thank my mentor Mara Helmuth for her ongoing support, as well as the University of Cincinnati University Research Council for sponsoring in part the development of *RTMix*.

REFERENCES*

3DNow!. http://www.3dnow.net
ALSA. http://www.alsa-project.org
Altivec. http://developer.apple.com/hardware/ve/
Apple Corp. http://www.apple.com
ASIO. http://www.steinberg.net/en/ps/start/steinberg_solution/technology/
Bukvic, I. 2002. RTMix: a real-time interactive electroacoustic music performance, composition, and coaching interface. In *Proc. of the Int. Computer Music Conf*. International Computer Music Association.
Buzz. http://www.buzzmachines.com
(CCM)^2 Studios. http://meowing.ccm.uc.edu
Cecilia. http://www.musique.umontreal.ca/electro/CEC/
CLM. http://ccrma-www.stanford.edu/software/clm/
Csound. http://www.csounds.com
Dell Corp. http://www.dell.com
Eurocom Corp. http://www.eurocom.ca
FreeBSD. http://www.freebsd.org
Garton, B., and Topper, D. 1997. RTcmix – using CMIX in real-time. In *Proc. of the Int. Computer Music Conf*. International Computer Music Association.
Glame. http://glame.sourceforge.net
Globalisation. http://www.imf.org/external/np/exr/ib/2000/041200.htm
GPL License. http://www.gnu.org/licenses/licenses.html
Helmuth, M. 2000. Internet sound exchange. In *Proc. of the Int. Computer Music Conf*. International Computer Music Association.
Henry, P. http://www.pierrehenry.de
Hyper-Threading. http://www.intel.com/technology/hyperthread/

*Cambridge University Press accepts no responsibility for content on the websites mentioned in this article.

ICMC 2002. http://www.icmc2002.org
IRCAM. http://www.ircam.fr
JACK. http://jackit.sf.net
JMax. http://www.ircam.fr/jmax
JSyn. http://www.softsynth.com/jsyn/
KDE. http://www.kde.org
Kyma/Capybara. http://www.symbolicsound.com
Lansky, P. http://www.newalbion.com/artists/lanskyp/
Linux Audio Developers. http://www.linuxdj.com/audio/lad/
Max/MSP. http://www.cycling74.com
MIDI. http://www.midi.com
MIT. http://www.mit.edu
OCEAn. http://www.timara.oberlin.edu/ocean/index.htm
potentiomètre d'espace. http://www.zvonar.com/writing/spatial_music/Short_History.html
Pure-Data. http://www.pure-data.org
Qt. http://www.trolltech.com
RT. http://www.music.princeton.edu/winham/PPSK/sgi.rt.html
RTcmix. http://www.people.virginia.edu/~djt7p/Cmix/Cmix.htm
RTMix. http://meowing.ccm.uc.edu/~ico/
Sager Corp. http://www.sagernotebook.com
SAOL. http://web.media.mit.edu/~eds/mpeg4-old/
Smalltalk. http://www.smalltalk.org
Soundmesh. http://meowing.ccm.uc.edu/softmus.htm
Spiral Synth Modular. http://www.pawfal.org/Software/SSM/
Squeak. http://squeak.org
SSE2. http://www.tommesani.com/SSE2Intro.html
STK. http://ccrma-www.stanford.edu/software/stk/information.html
Stockhausen, K. http://www.stockhausen.org
Subotnick, M. http://www.newalbion.com/artists/subotnickm/
Supercollider. http://www.audiosynth.com
Telesymphony. http://www.flong.com/telesymphony/
Vaz Modular. http://www.software-technology.com
Visual Orchestra. http://www2.hku.nl/~perry/visorc/index.htm