Emonic Environment – Implementation Report

Paul Nemirovsky, Richard Watson, David Dickinson

Media Laboratory

Massachusetts Institute of Technology

E15-347, 20 Ames St., Cambridge, MA, 02139

U.S.A.

{pauln, watsonr, davidrd} @ media.mit.edu http://www.media.mit.edu/~pauln/research/emonic

Abstract:

This paper presents a progress report on the implementation of the Emonic Environment (EE) – a Java-based system for improvisational creation, modification, performance, and exchange of audiovisual media. The protagonist users of the EE are non-artists whose creative drive has been impeded by the prevalent interactive interfaces that are largely passive (click-response) and discourage experimentation. We take non-idiomatic improvisation as our inspiration, and seek to present the performers with an environment where the tools for media exploration are "alive". In doing so, we hope to encourage the creativity of people otherwise afraid to experiment. This paper describes the functionality of the EE, focusing on the user interface, multi-user network capabilities, audio (performance & synchronization) and genetic algorithms used to explore a media landscape.

Key-Words: improvisation, evolution, media, neural networks, multi-user, audio & video processing.

1 Introduction

What is it that goes on in an improviser's mind, allowing him to weave together memories and vague ideas, audience mood, collaborating artists' input, all while coming up with new and unique expressions? How does he think about his goal; does he even have one? The development of the Emonic Environment (EE) is not an attempt to answer the core questions of creativity; rather, its aim is to capitalize on the experience of practicing improvisers by bringing the spontaneity and richness of improvisation into anyone's hands.

2 Layer Architecture

The EE architecture is inspired by the behaviour of an animal nervous system, realized in a framework of layers: Input, Structural and Perceptual. The objective of such a layered design is to allow for complex, dynamic, and multidimensional mappings between sensing, perception, cognition and action.

EE is a system whose objective is to be alive, aware of the outside world and itself, always exploring new possibilities in tandem with its users. As such, it requires sensory interfaces to the outside world. This is the function of the *Input layer*: to receive external input and route it to the rest of the system. Using the nervous system analogy, this layer is the skin, the eyes and the ears of our organism. In the core of the system, a neural network 'brain' resides as the *Structural layer*. Underlying system behaviour and patterns of activity emerge from properties and connectivity of the elements that populate this layer. Finally, back on the surface of the system, media events are scheduled and processed by a population of operators known as *emons,* bonded together in a web of connections to form the *Perceptual layer*. In our 'nervous system' analogy, this layer corresponds to the motor system of an animal.

Input layer: The collection of sensory interfaces to the EE. Using tangible, visual, auditory, and web-aware information samplers, it integrates stimuli in the surrounding physical and electronic worlds into the ongoing improvisation. Examples of interfaces include a computer mouse, video camera, custom-made gesture controller, web agent, and more.

Structural layer: This layer is a recurrent neural network (RNN), populated by *nodes* and weighted connections (or *associations*) between the nodes. The RNN architecture enables both intricate and largely unpredictable exploration of the media landscape while at the same time adhering to constraints that generate coherent patterns of behaviour. Each element within the RNN (as well as the RNN as a whole) possesses a number of properties, controlled either explicitly by the performer or by a system-administered process. These element/property-set pairs are as follows: (1) *Association*: Path, Weight, Time Delay (inner-node stimulus travel time); (2) *Node*: Activity, Decay Rate (of activity), Propagation Threshold (above which the node propagates any incoming stimuli), In/Out Stimulus Scalar; (3) *Network* as a whole: Maximum Simultaneous Propagations, Low-Activity Threshold (below which new spontaneous activity will be introduced), Auto Management Features (actions to be taken when operating without any performer input).

Mediated layer: A messenger and translator between layers. Its first job is mapping state-change notifications from the Structural layer, in the form of tokens, to a separate set of tokens which is delivered to the Perceptual layer and interpreted there. The Mediated layer itself is ignorant of what any particular token means, though its role in shaping an overall meaning remains extremely significant. It is through evolution of the maps in the Mediated layer that abstract structural patterns begin to be interpreted and used in interesting ways. As the data received in the Input layer is disparate, we employ the Mediated layer in a second task: to make this data useable in the other two layers of the EE. For this, the Mediated layer employs *transforms* – mapping algorithms that act as links between the data format employed by a given input device and that of a particular inner-system element. For instance, one such transform maps between the input of a custom-made gestural controller (called the Emonator[1]), which provides an array of 144 short values, and a double value. In this case, the signal output of the transform is routed to the Perceptual layer, where it is used by audio-processing *emons* to modulate frequency and amplitude.

Perceptual layer: This layer is the space within which all of the media processing takes place. The layer is populated by *emons, data,* and *media* elements, with connections between the three. The media are audio and video samples accessible to the EE. The data elements are arrays of numbers recorded from input received from the Mediated layer, evolved, or manually specified. The data are used by emons as sources for control information such as timing, frequency, amplitude, etc. Emons are best thought of as operators that use available data, signals from other emons, and *directives* provided by the Mediated layer, to modulate the generation, modification and presentation of media or to control other emons. Each emon has one function and one or more mutable properties. For instance, an *amplitude emon* takes associated data and uses it to control the amplitude output of an *audio-sample-player emon*[2].

The range of emons encountered on the Perceptual layer is diverse. The species include: audio and video sample playing; audio waveform generating[3]; audio processing (e.g. filtering); video processing; event timing[4]; and lighting control[5].

Elements and configurations of all of the different layers as well as all media incorporated in a performance are shareable. That is, the system is built to communicate with other instances of the same system, providing means for immediate collaboration with other EE performers. This is discussed further in the Multi-User section below.

The system is designed to run with variable degrees of autonomy. The variability allows it to be installed in a number of different environments. On one end of the spectrum, a musician might employ the system in a performance where he requires complete and immediate influence in the Structural

[1] See http://web.media.mit.edu/~pauln/research/emonator

[2] Such chaining of functions is familiar to anyone who has used Max/MSP-like audio processing environments [2].

[3] JSyn is used for sound synthesis, playback and processing.

[4] JMSL is used for system tempo and event timing.

[5] The EE integrates control of a 4x5 grid of ColorKinetics RGB LED arrays, capable of producing 17 million colors. It affords us the ability to reflect network activity with lighting displays, filling the room with visual mood.

and Perceptual layers. On the other, a visual artist might set up a semi-permanent installation where ongoing network activity in the Structural layer continually and independently modulates room lighting controlled on the Perceptual layer, taking inputs only from ambient sensors.

Such a layered design provides improvisers a way to access the system on many levels. It affords performers the capability to dynamically refocus their objectives while creating and manipulating media in real time. The design also modularizes the system, making it easier to understand from the inside and easier to expand. Links between the layers enable flow from outside world influences to inner dynamic structures and ultimately to media operators that bring the result back to the outside world. Connectivity within the layers promotes the emergence of richly interactive and largely unpredictable behaviour.

The current version of the system does not fully realize the architecture specified in this section. The EE still has a long way to go before it fully implements the defined functionalities of and independence between the Structural and Perceptual layers. Bridging this gap and bringing the implementation closer in line with the outlined layer model is therefore our immediate objective.

3 Graphical User Interface

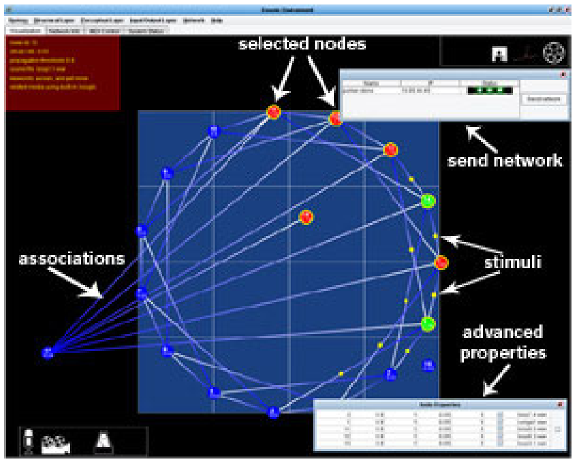

Figure 1: Emonic Environment's GUI

Our primary goal when constructing a visualization for the EE was to emphasize connections between elements rather than the elements themselves. The reason is evident – improvisation is an experience centered on connections and not on artifacts. Any visualization that aims to help the user to think improvisationally and be creative in his exploration of the media landscape should accentuate the intrinsically connected nature of improvisation.

Here we discuss only the visualization pane of a multi-paned window environment[6], as that is where the majority of user-system interaction takes place. Since the structural and perceptual elements of the system are not yet separate, no separation exists in the GUI. Instead the visualization focuses on elements and connectivity in the Structural layer: *nodes* and *associations.*

Nodes are represented by solid circles that change colour depending on state. Inactive nodes are blue; when active (above propagation threshold), they turn red. When stimulated, nodes flash yellow; when associated emons perform an operation (e.g. play a sample), they flash green. Left-clicking a node stimulates it by a predetermined amount; right-clicking presents a menu of options, including one to edit the node's properties. The middle mouse button serves to initiate an association or, if one has been initiated already, to complete it. It is possible to select multiple nodes, changing properties and making associations en masse. An extensive set of keyboard shortcuts can be used to simplify and speed up commonly used operations such as exciting or inhibiting nodes, changing associated emons/media, sending a selected subset of the network to another user, and more.

Associations take the form of bicoloured lines between nodes, fading from blue at the origin to white at the destination, indicating directionality. Stimuli sent between nodes are represented as small solid yellow circles travelling along the line of association. Clicking on an association allows the user to delete (left-click) as well as modify (right-click) that association's properties. Building associations between two or more nodes is simple: the user chooses a source node(s), and selects the destination node(s) with the middle mouse button, thus creating the link between the two. Building a functioning media network from scratch thus becomes a task that can be achieved in a matter of seconds.

[6] The other panes delineated with tabs are (1) element property controls, (2) MIDI mappings, and (3) information output console.

A network can be saved or restored at any point throughout the operation of the program. This save/restore functionality is similar to the ordinary file save/load functions. Once a network has been restored, it is activated. From that point on, this network will develop differently from a similar network restored at another time or on another computer. This variability is an inherent result of the evolutionary process driving the development.

Clearly such save/restore functionality is static and limiting – the networks are seen as discrete entities, with no connection between the current and the saved states[7]. To avoid this, we employ saved network states as *evolutionary magnets*. In this technique, users evolve a network's current state toward or away from one or more magnets. To visualize this, imagine the behaviour of a piece of metal as a magnet is placed nearby: the metal starts moving towards the magnet. This analogy is crude; it reflects neither the multidimensional nature of evolving multiple node properties nor the possibility that the magnets change their strength over time. However, it provides us an illustration of the guiding principle of the EE: exploration of system states should always be continuous.

Having created a network (or modified an existing one), the user can send a subset of it to other users. To do so, the user selects the network subset to be sent, and opens the Send Network window. The window displays EE clients currently available on the network. By selecting one of these, the selected network subset (i.e. nodes, associations, and media) is sent through a server to the destination EE client. There the received subset immediately becomes an active part of that user's current network (see Multi-User section for details).

The EE is currently capable of producing intriguing and dynamically changing audio sequences. For instance, perusing a data repository containing both string orchestra recordings and political orations of Fidel Castro produces an unexpected combination of comedy and drama. The first reactions to the EE by non-professional users have been most encouraging. Having spent extended amounts of time with the system, they generally commented on how it felt surprising and powerful to be able to shape the music by modifying the network structures rather than the audio itself.

[7] Incidentally, this problem arises in any system that employs discrete constructs such as files or hyperlinks.

4 Evolution

Evolutionary algorithms, employed throughout the EE's three core layers (Input, Structural, and Perceptual), provide users with multiple exploration paths through the media landscape. The evolutionary process can be run as a single instance, i.e. with the configurations of elements on different layers merged into one large genetic code. Alternatively, the evolutionary process may be run as several distinct instances; doing so allows each network layer to evolve on its own time scale, facilitating the emergence of complex, dynamic, and unpredictable mappings.

Evolution modifies the Input layer by controlling just where and how data is mapped between an input interface and another layer. When a specified evolutionary process runs, it modifies the *transforms* (described in the section on Layer Architecture), as well as their destinations, shaping the way the outside world affects the system.

Elements in the Structural layer are possibly the prime target of the evolutionary process. The evolution affects the connectivity of the network and the properties of individual nodes, producing new mappings unlikely to be discovered by the user alone, and thus aiding his creative process.

In the Perceptual layer, the evolutionary process modifies (1) *data* elements, changing the sources on which the emons operate; (2) the connectivity matrix of the emons (their interdependency); and (3) token-action mapping of individual emons, changing their response to various inputs.

Three operational evolutionary modes are currently being developed for the Emonic Environment. Each mode offers EE performers a different way to interact with and change the system.

1. Browse (offline). In Browse mode, several 'child' networks run in the background. The networks' states are initially synchronized; that is, elements correspondent between the networks are in identical states. Each network's output is recorded over a set period of time. Each recording is voted on by the performer, and decisions about breeding, mutating, and killing of the source networks are made based on these votes in a tournament fashion. This mode is designed for a performer who is interested in configuring a network offline in a simple, hassle-free way.

2. Explore (online). For a user who would like to perform with the EE, interacting with it in a continuous fashion, Explore mode offers a real-time option. Here, parameters of a single network are mutated on a consistent periodic basis. Voting is an ongoing process whereby votes are captured and mapped to the appropriate configurations (present at the time of voting) according to user's actions. Direction of mutation is continually modified, based on the voting.

3. Navigate (online). Navigate mode offers a more 'hands-off' option for the performer. In this mode, the performer assigns one or more saved network configurations, called *magnets* (described in the GUI section), to guide the evolutionary process. The magnets either attract or repel the state of the online network toward or away from that configuration, thus producing interesting phase trajectories in a performance. The mode is made more interactive when the user dynamically (de)emphasizes magnets.

5 Multi-User Architecture

While the EE provides an isolated user with many improvisational options, the creative potential of the system is significantly augmented when users are able to interact with each other in a meaningful context – that of a collective improvisation. For instance, one user may excel at the creation of new media, another at reconfiguring the arrangements of others; together they can reach far beyond what either could do alone. Despite continuing gains in computing speed and sophistication, the richest creative user environment is still the one enhanced by interpersonal exchange. Towards that end, EE users need to be provided with easy access to others' creations, past and present, and given the ability to collaborate seamlessly in real time.

Thus, the two main objectives in designing the multi-user architecture were (1) to facilitate easy and rapid exchange of networks between users, and (2) to allow rapid formation of shared repositories for newly created media, letting all performers access all the available media.

Given the potential EE user's level of technical competence, a number of design issues arose: (1) we needed to pick a model that would allow every client to see all the other clients available for collaboration at a given moment; (2) we could expect very little from the user with regards to technical knowledge (e.g. IP addresses, media filenames, etc.) As a result, a simple peer-to-peer system wouldn't do; designating each EE client as a peer with no further infrastructure would make it difficult for a new EE client to find others, and asking users to know and manually enter peer IP addresses seemed unreasonable.

In light of these goals and constraints, we adopted a model in which each EE peer is considered to be the client of some particular server, with all client-client communication and file transfer routed through that server. In that scheme, EE clients are regularly informed by the server of the availability of all the other clients in contact with that server.

Furthermore, any user can choose to run their own server (built into the EE) and become a potential focal point for a new group of users. This server deployment model will hopefully lead to propagation and evolution of structural and media EE components through spontaneously created user-run centres of exchange.

Examination of a sample session involving transfer of a network subset along with the associated media files between two EE clients may be illustrative. The originating user's EE is referred to here as Client 1, or C_1, and the destination user EE is termed Client 2, or C_2. Both users are assumed to be using server S.

The transfer commences with the user selecting a group of nodes and indicating C_2 as the recipient. This EE client is now referred to as C_1.

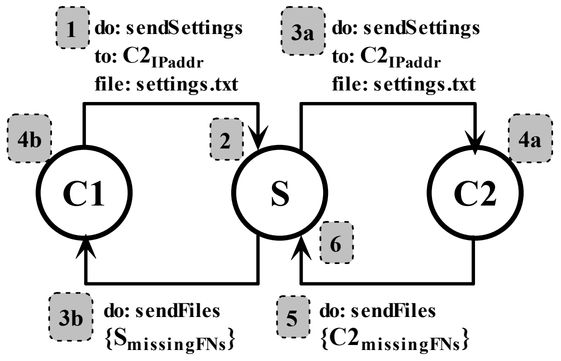

Figure 2: Transfer of a network from C_1 to C_2

1. C_1 asks S to relay a selected portion of C_1's network, saved in settings.txt, to C_2.

2. S parses settings.txt for the media used and stores in $\{S_{missingFNs}\}$ the names of any media files S is missing.

3a. S passes settings.txt on to C_2.

3b. S requests the files $\{S_{missingFNs}\}$ from C_1.

4a.1. C_2 parses settings.txt and stores the names of missing media files in $\{C2_{missingFNs}\}$.

4a.2. C_2 loads the nodes and associations specified in settings.txt into its current network, with placeholders for missing media. A mapping is maintained between the names of the missing media files and the nodes that await them.

4b. C_1 queues the files specified by $\{S_{missingFNs}\}$ for transfer to S and begins sending them.

5. C_2 requests the files $\{C2_{missingFNs}\}$ from S.

6. S queues the files in $\{C2_{missingFNs}\}$ that it already has for transfer to C_2, and begins sending them.

7. When S finishes downloading the file $C1_{mediaFile\_i}$ from C_1, this file is queued for transfer to C_2 if it is in $\{C2_{missingFNs}\}$, the list of files C_2 has requested.

8. When C_2 finishes downloading a requested file from S, it looks up the node to which that media file is mapped, and updates the media for the node to reflect the newly downloaded file.

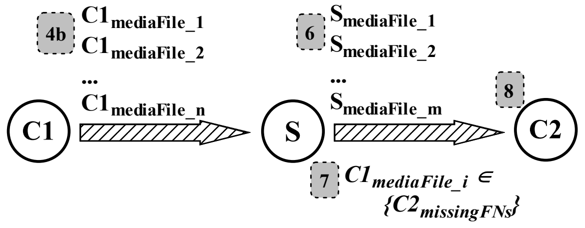

Figure 3: Transfer of needed media files to C_2

Since the server keeps a copy of every media file that passes through it, our network model allows the server to readily accumulate a complete collection of media produced by the users. Furthermore, the load on the initiating client is subsequently reduced – any media that is used in the sub-network, and which the destination client lacks, can be immediately sent to the destination client given the server has a copy. In other words, across all users and sessions, a particular media file needs to be transferred to the server only once (the uniqueness is currently determined solely based on the filename).

To publish its existence and continued availability, each client pings the server every few seconds. In turn the server maintains a table of all clients that have ever been active. Each client is marked with a timestamp indicating the local server time when the last ping was received, with the clients considered alive by the server if their timestamp differs from the current local time by less than a fixed timeout.

6 Future Work

We are continuing to develop the Emonic Environment, adding various features and preparing the system for public release. Our immediate goals include adding functionality to the Structural and Perceptual layers, fully integrating video into the system, increasing user-friendliness of the interface, and producing a Mac OS X version of the software. Planned features also include web-based agent for media gathering, built-in sample-maker, and new audio-processing libraries.

7 Conclusions

This paper reported on the ongoing development of the Emonic Environment, describing its key functional components and underlying architecture. We hope that the EE is a small step in the right (or rather shall we say, intriguing) direction, bringing us a bit closer to our objective: applying improvisational principles to all walks of media creation, browsing, and exchange. Supplanting linearly structured and fixed-goal-oriented notions of entertainment and learning with those inherent in the EE may transform explorations of these domains into inherently unique and communal adventures, blurring the distinction between media consumption and creation and ultimately bridging the creativity of humans and machines.

References:

[1] Nemirovsky, P., Watson, R. (2003). Genetic Improvisation Model, a framework for real-time performance environments. In the Proceedings of EvoWorkshops 2003.

[2] Max/MSP, http://www.cycling74.com