THE INTERACTIVE EVENT MANAGER (IEM): A PERFORMER’S APPROACH TO INTERACTIVE SYSTEM DESIGN

David Brooke Wetzel, DMA
Mansfield University of Pennsylvania
Department of Music

ABSTRACT

The author presents the “Interactive Event Manager” (IEM), a modular framework for realizing interactive electroacoustic works. Developed as a performer’s tool for the realization of multiple works by various composers within a single setup, IEM also presents an opportunity for composers wishing to create new interactive works without the need to develop or maintain a complex dedicated system. IEM is a script-driven network of synthesis, DSP, audio and MIDI i/o, and utility modules in the form of Max/MSP abstractions. IEM modules may be instantiated, connected, and controlled in real time via event scripts. Modules may be shared between different interactive setups, and new modules may be created and added to the IEM system as needed. The author presents the general premise and evolution of IEM, its basic structure and scripting protocol, and examples of performance systems developed using IEM.

1. INTRODUCTION

As a clarinetist specializing in interactive electroacoustic performance, the author is particularly concerned with the development and maintenance of a sustainable live-electronic music repertoire. The system presented here, the Interactive Event Manager (IEM), is a tool for incorporating multiple interactive electroacoustic works into a single performance environment. Although it was designed around the requirements of a handful of specific concert works for the purpose of realizing and maintaining a repertoire, IEM has developed into an extremely flexible system that may be useful for new composition, experimentation, and research.

As Bruce Pennycook has recently noted, composers cannot expect to maintain their own interactive systems in perpetuity [4]. The author has presented a performer-driven model for preserving such works through analysis, reconstruction, and performance [8]. The analysis phase describes the components of each composer’s system in generalized, device-independent terms, with a particular focus on separating the composers’ musical intentions from the limitations of the original instruments. Several complex composition-specific performance systems were broken down into their basic component parts: audio input/output functions, signal processing, synthesis, file playback, input sensing, event triggers, and miscellaneous algorithmic processes. The result was a description of each system as a set of interrelated units, or modules, each with easily described functions based on commonly understood principles of electronic and computer music. What’s more, many of the same module types appeared in numerous works. As generalized, interchangeable building blocks, such modules could be used and re-used within a diverse array of setups.

Each of the works analyzed for this initial study also shared one crucial feature: a sequential list of system state variables or actions tied to discrete “events” in a musical score. Therefore, in the works studied and performed by the author to date, the main organizing principle has been a series of discrete system components under the control of a sequential “event list.” The IEM system presented here applies these principles in the form of a single platform capable of running numerous interactive works.

The following is a general description of the system and its basic components. The author also presents examples of interactive performance systems using IEM for the realization of two works for clarinet and interactive electronics.

2. IEM STRUCTURE

The Interactive Event Manager (IEM) is a script-driven interactive environment developed in Max/MSP that coordinates multiple modules for real-time synthesis, DSP, algorithmic data processing, audio i/o, controller input, and system monitoring. The IEM is based on the coordination of three main components: a central processing module, an event script, and various processing and synthesis modules.

2.1. IEM Main

A central module, “IEM Main,” coordinates the IEM system. IEM Main has three primary functions: 1) loading event scripts and IEM modules; 2) event number sequencing; 3) routing of event script messages to active IEM modules. Certain key functions of IEM Main, such as module instantiation and event script loading, are themselves addressable from the event script.
2.1.1. Loading event scripts and instantiating modules

An event script (a text file stored on disk) may be loaded manually from IEM Main’s user interface. Alternatively, a new script file may be loaded dynamically in response to the event script command “newlist” addressed to main.

New modules are instantiated as abstractions of Max/MSP patchers. Upon instantiation, each module is given a unique name by which it is addressed from the event script. A module may be loaded from a command line interface in IEM Main, or from the event script command “newmod” addressed to main.

Once the new module is created, its name and source file are recorded in a database of currently active modules. The list of active modules is displayed in the IEM main window for reference and access to their individual control panels.

2.1.2. Event sequencer

The event sequencer keeps track of the current event number and passes it to the event processor when triggered. The event sequencer includes user interface controls for advancing through event numbers manually. There is also a control channel input for triggering event numbers based on output from other modules (e.g., a MIDI footswitch module). The event sequence may be advanced in linear or non-linear fashion.

2.1.3. Event processor

The event processor scans the entire event script for event numbers matching the one received from the event sequencer. Each line in the event script is addressed to a particular module by name, and contains a string of parameters and their associated values. Upon execution of a particular event, lists of variable parameters and their values are routed to the appropriate modules.

2.2. The Event Script

The event script is a simple, text-based list of module commands associated with event numbers. An entire system may be instantiated, with all modules connected and all variables initialized on the basis of one or two events in the script. Once the system is initialized, individual modules may be controlled from the event script. The user may edit this script at any time. Its format is designed to be simple enough that musicians may make small adjustments to it during rehearsal.

2.2.1. Event script format

Following the conventions of the Max/MSP coll object, lines are numbered for reference, and each line terminates in a semicolon. The first element of a script line is an event number. This is followed by the name of an active IEM module. Following the name of the target module is a string of parameters and their associated values. A typical block from the event list might look like this:

1, 20 delay time 250 fb 0. gain .5;
2, 20 delay gain-cc f-pedal_1;
3, 20 footcontrol ped1-out f-pedal_1;

In this example, at event 20, a module named delay has its values for “time,” “fb,” and “gain” set. In the second line, also executed at event 20, the same module has its continuous-control input (“gain-cc”) for gain level connected to the control channel “f-pedal_1.” The “ped1-out” parameter of a module named footcontrol is also assigned to “f-pedal_1.” Therefore the pedal output from footcontrol is connected to the gain level of delay.

2.2.2. Script messages to IEM Main

Event script commands may control IEM Main in several ways. The most important of these is the instantiation of new modules using the script command “newmod.” Other commands to main include “newlist” (loads a new event script), and “nextevent” (set the event trigger to receive messages from other modules via control channel). The following event script lines load new modules:

1, 0 main newmod VKDIG411.patcher delay;
2, 0 main newmod IVL.patcher pt;

In this example, the first argument is the file name for a Max/MSP patcher to load as an abstraction. The second is a name to give the new module so that it can be addressed by subsequent script events.

2.3. IEM Modules

An IEM module is an instantiation of a Max/MSP abstraction. Multiple instances of the same module may be loaded within a single setup and are addressed from the event script separately. Pre-existing Max/MSP patchers may be adapted to the IEM system. Many modules serve
common functions required by nearly every system, such as basic audio i/o or user interface controls. New and specialized modules may be designed and incorporated into the system as needed. Variable parameters and functions may be controlled from the event list or from other modules via inter-module control channels. The basic components of a typical IEM module are shown in figure 2, below.

### 2.3.1. Event processing

Upon instantiation, each module is given a unique name by which it is addressed from the event script. The module’s name is given as an argument to a Max/MSP receive object, which acts as the sole conduit for messages from the event script. IEM modules parse incoming lists of variable parameters and distribute their values locally within the module patcher.

### 2.3.2. Module interconnection

Modules may connect directly with one another in various ways. The event script may connect multiple audio processing or synthesis modules to form complex signal networks. A module’s variable parameters may be connected to data outputs from other modules over script-defined control channels.

An audio bus or control channel is established in the active IEM network by setting name values for inputs or outputs of one or more modules. In the current version of IEM, these data inputs and outputs are Max forward and receive objects. If the input of one module and the output of another are given the same name, then the modules are linked and data will pass between them. Modules with audio signal i/o are connected to each other in much the same way as with data channels, using send~ and receive~ objects. Because control channel and audio i/o names are scriptable elements, connections between modules may be created, deleted, or re-named at any point in the event script.

### 3. CURRENT AND FUTURE IEM PROJECTS

IEM is a work in progress, and has evolved considerably since the first prototype was presented at SEAMUS in 2007 [9]. The following examples show relatively simple IEM systems. Both Musgrave’s Narcissus and Kramer’s Renascence required delay systems that are now obsolete. Implemented in IEM, they share several modules, and switching from one to the other involves merely loading a new event script.

#### 3.1.1. IEM realization of Musgrave’s Narcissus

Thea Musgrave’s Narcissus (1987) for flute or clarinet and digital delay is an excellent test case for IEM. The required system is relatively simple (a digital delay with adjustable delay time, feedback, LFO-controlled delay modulation, and a “hold” feature). The Vesta Koza DIG-411 digital delay specified in the score [3] is now obsolete, but a viable system for performance of this work can be constructed using IEM modules. Based on the author’s previous analysis and reconstruction of Musgrave’s system, [6], the “DIG411 Translator” module takes knob values found in the score and translates them to standard parameter values for a multi-purpose digital delay. The “Controller Input” module handles input from MIDI footswitches.

![Figure 3. IEM realization of Musgrave's Narcissus.](image)

#### 3.1.2. IEM realization of Kramer’s Renascence

Jonathan Kramer’s Renascence (1974) for clarinet, tape, and tape delay requires a long delay, a controllable 3x3 matrix mixer, and audio file playback [2]. The author has previously presented an analysis and realization of this work at the Spark Festival of Electronic Music and Art in 2006 [7]. In the IEM realization, the ADC, DAC, and delay modules are the same as used for Narcissus. Renascence requires a metronome for coordination of the live part with the delay. Although the 1974 score calls for a technician to operate the matrix mixer, in the IEM realization, the metronome also drives the event list, automating mixer...
changes according to the directions in the score. In the original system, a separate tape (and tape machine) was required for playback. In the IEM realization, a sound file player module handles this function.

Figure 4. IEM realization of Kramer's Renascence.

3.1.3. Future development of IEM

Although the IEM has been constructed primarily around the technical needs of a handful of specific works, the guiding principle of keeping each module’s functions as generalized as possible has resulted in a system that is far more flexible than is required for any of the works examined in the author’s previous research. Therefore, the resulting system could be an interesting tool for composition and exploration using newly created processing and synthesis modules alongside modules designed to emulate the functions of “classic” works. For example, new IEM-based realizations of Bruce Pennycook’s Prascio IV (1990) and Cort Lippe’s Music for Clarinet and ISPW (1992) are in progress. The author is also collaborating with composer Andrew Walters to create a new work for clarinet and interactive electronics using the IEM as its performance platform. A sketch of this new work was shown at SEAMUS in 2008 [10]. Examples of working IEM performance systems may be found on the author’s web site: davidbrookewetzel.net.

The IEM system has, so far, been developed entirely in Max/MSP. Work by others focused on the standardization of Max/MSP patchers and their control structures, such as Jamoma [5], may have useful implications for the further development of the IEM system. However, the general concept of the IEM is not specific to Max/MSP. Similar efforts to standardize modular interactive systems, such as the Integra project, have demonstrated the adaptability of other platforms such as SuperCollider, Pd, and Csound to a modular approach to interactive system design. [1]

The work presented here is the result of efforts by an independent performer to organize and maintain a functional, personal repertoire. The scope has been therefore extremely limited to date. However, the similarities between IEM and systems such as Integra and Jamoma, point to an urgent need within the computer music community for common tools and a better appreciation for issues of maintenance, preservation, and the performance practice of interactive electroacoustic music.

4. REFERENCES


