Words to look at, words to listen to: Designing a "proliphonic" display for the lobby of the New York Times Building

Mark Hansen
UCLA, Department of Statistics

Ben Rubin
EAR Studio

Hans-Christoph Steiner
ITP/NYU

Tyler Walker
Perfection Electricks

Abstract

We report on the development and initial experiences with *Moveable Type*, an art installation in the ground-floor lobby of the recently completed New York Times Building in New York City. Physically, the piece is divided into two large display grids suspended along both sides of the building's main lobby facing Eighth Avenue. Each grid is comprised of 280 devices (7 rows × 40 columns), custom components consisting of a graphical "face" (a commodity vacuum fluorescent display, or VFD), two audio elements (a proper speaker and an automotive relay) and a control unit (an embedded Linux processor). In this paper, we will focus mainly on the design of the installation's audio system. With its 560 point sources of sound two grids, 280 devices per grid), the piece is an interesting case study for the Linux audio community, offering an acoustic experience that can best described as "proliphony." In this paper, we will review the system architecture underlying *Moveable Type*, as well as the process for authoring visual and acoustic effects.

Keywords

Data streams; text; real-time audio systems; Pd.

1 Overview

Located in the lobby of the new New York Times building in Midtown Manhattan, *Moveable Type* can best be characterized as a dynamic portrait of the Times. The piece takes its energy from the paper itself, from the activities of thousands of reporters, editors and commentators, and the sea of words that emerges. Text fragments, portions of news stories, articles, editorials and blogs, are culled into an up-to-the-minute feed that is combined with the Times' archive, a complete record of the printed paper dating back to 1851. Along with the "production" side of the paper, we also have access to hourly summaries (anonymized and aggregated) from the web server(s) and search engine behind `www.nytimes.com`. These data provide us with a rough sense of the activities and interests of the paper's readers.

The installation itself consists of two large grids, each roughly 65 feet in length. Together, the grids contain a total of 560 devices (7 rows × 40 columns × 2 walls). The columns of the grids are suspended from busways above the ceiling and hang a few inches in front of the two walls of the central corridor in the main lobby. See Figure 1 for images of the installation. The columns hang from six wires (three left, three right) that provide physical support as well as power and serial (RS485[1]) communication for the devices along the "strands." The individual devices (7 per strand) are custom components consisting of a graphical "face" (a commodity vacuum fluorescent display or VFD), two audio elements (a proper speaker and an automotive relay) and a control unit (an embedded Linux processor).

Moveable Type is organized into a series of scenes, much like the movements of a symphony. Each scene follows its own processing logic for identifying and exhibiting patterns in our data streams, either in reporters' language usage or in readers' browsing and searching activities. For each scene, the piece adopts a different visual and sonic personality. The displays themselves are remarkably expressive (thanks in part to a custom Python module that acts as a kind of byte complier, allowing for programmatic access to the screen's display functions) and are capable of displaying both text and simple graphics. They are, however, silent. We make extensive use of the audio elements on the devices in the grid to underscore the visual activity, filling the space with the iconic sounds of a newsroom. With this unique "instrument," *Moveable Type* plays with language and how stories are told; with the news and our memories of recent and distant events.

1.1 Scene structure

At present, *Moveable Type* runs through a daily cycle of about a dozen different scenes. Some

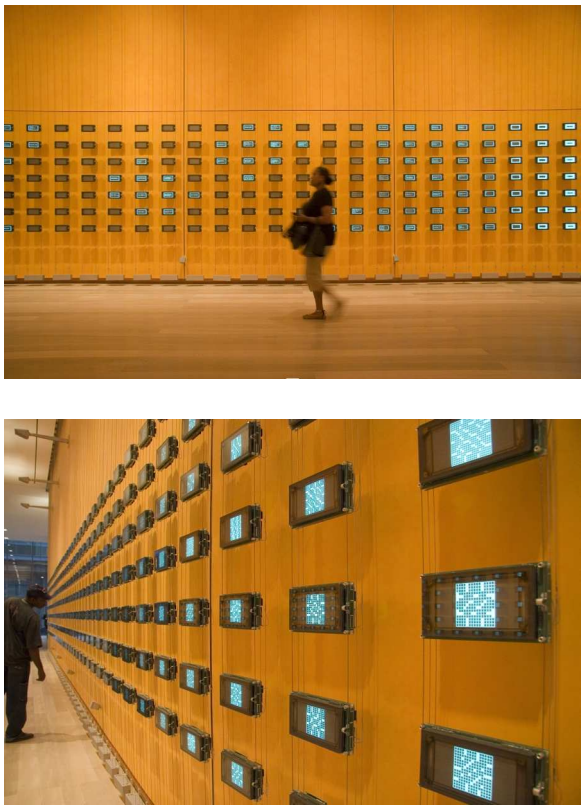

Figure 1: A portion of the north wall of *Moveable Type* (top) and an angled view highlighting the physical supports used to suspend the columns (bottom).

focus on particular sections of the paper (weddings, letters to the editor, the crossword puzzles), while others combine data from the entire paper. We now present three scenes, focusing on content and the accompanying display design or "choreography." We will return to their technical implementation toward the end of the paper.

Facts and Figures. In this scene, we (re)tell the day's news through the facts and figures reported in the paper:

> *two* large social networking sites; *two* of the most splendid pieces of French furniture ever created; *two* prominent chief executives - at Merrill Lynch and Citigroup; *three* healthy sons and a good career; *four* times the amount that had been reported missing; *200* endangered witnesses a year; *seven* dozen Taliban fighters killed during a six hour engagement; *five*-story limestone structure; *five* reactors in storage buildings here in Wuerenlingen, near the border with Germany.

In terms of text processing, this involves parsing every sentence in the most recent version of the online edition of the New York Times, determining its grammatical structure. We then apply custom filters to the resulting parse trees to extract number-item pairs. The extracted figures are grouped by story, and during the scene, each screen exhibits the figures taken from a single story. In designing the audio and visual character for this scene, we took inspiration from old-style split-flap train station displays. The numbers flip over themselves in quick succession, moving from one figure to the next, pausing for a moment in between to "type out" the objects associated with the count (for example, a sequence of fast flips uncovers the number "2," followed by scrolling small text "two large social networking sites"). To imitate the effect of a split-flap display, we use the relay click to underscore the changing or "flipping" of one number to the next on the VFD. Typing out the actual text is accompanied by a sampled clicking sound that produces a low whir as the text appears.

To the Editor. We next focus on the Letters section of the paper. Here we present the letters in a very straightforward way and our only text processing exercise involves extracting the name and location of the letter writer and the date it was authored (this turns out to be harder than one might expect due to the way the letters are formatted by the paper's editorial system). Each screen will exhibit a single letter so that the most recent 280 letters to the editor are shown (here the two walls are "mirrored").

The scene begins with a rhythmic introduction, a regular sequence of "keystrokes" in which the screens type out in unison T-o-_-T-h-e-_-E-d-i-t-o-r. As text appears on the usually silent VFD, it is accompanied by the sound of a keystroke from a manual typewriter. After this patterned introduction, each screen begins to type out a different letter to the editor. The keystrokes on each screen are randomized, in the sense that for every character we select from among five different recorded sounds at random and assign a volume so that (on average) every 10th character is louder than the others. At the end of each line the visible text is shifted up by a line and a sampled carriage return is triggered to complete the effect. (The letters appear on the screen "justified" using both variable spacing and hyphenation, the latter being performed on the screens themselves using

Knuth's algorithm developed for TeX). When the letter is complete, the text scrolls up, pushing the last lines of the letter off the screen, and leaving behind the name and address of the letter writer, together with the date the letter was received, centered vertically on the screen. This last movement is accompanied by the classic bell of a manual typewriter.

As an aside, reporters visiting the installation insist that we have recaptured some of the sounds that have been lost in a modern newsroom. Before computers and acoustically treated workspaces, the newsroom was full of sounds. *Moveable Type* deals in typewriters, telephone dialing tones, and even radar sweeps. These are lost newsroom sounds, lost sounds of communication processes, of latter day information technologies.

The Weddings. Finally, we describe a scene devoted to the Weddings section of the paper. Here, we present a subset of the details associated with about 20 weddings reported in the most recent Sunday paper.

> *His father taught second grade at the Smith Avenue School in Norwich, Conn. Her mother is a sales account manager at the Gabriel Group. He is a financial adviser at Merrill Lynch in New York. He graduated from the University of Vermont. She graduated from the University of Maryland and received a law degree from Brooklyn Law School. Her father works in Hudson, Mass., as a computer chip designer at Intel.*

Before display, we remove references to "the bride" and "the bridegroom," their parents, and their actual names, replacing each with "he," "she," "her father," "his mother" and so on. The idea is to reduce the details of each wedding to a somewhat generic structure.

During the scene itself, each wedding is represented through a network graph, with boxed text (a detail from a single wedding) connected by lines (each screen will contain either boxed text or a line), and can span between 10 and 15 columns. In all, 20 weddings are displayed during this scene, each drawing its own graph independently and crossing each other frequently. The final visual effect makes it hard to detangle the individual weddings (the generic nature of the processed wedding details adds to this effect). Each network graph is revealed slowly, with text components appearing sequentially, separated by a line drawn across two or more screens (the text is boxed and the lines are drawn slowly so that they creep across each screen). The audio design here is somewhat involved, but the basic component is a series of sampled sounds that move from screen to screen as the text appears and the lines/boxes are drawn. This kind of moving melody is used in several places in *Moveable Type* and is made possible by our unique architecture of distributed control which we will describe in the next section.

2 System Design

To handle the text display, each screen was designed to be a self-sufficient node. This made it easier to compose complex audio/visual effects since the sequencing of grid-wise actions are choreographed from a central place: By distributing effects or pushing the control out to the nodes, the central server typically has only to send out a sequence of triggers. In addition, this distributed design made the control messages quite simple, keeping the communication over the RS485 interfaces to a minimum.

When considering the design of the sound component, we had two options. The first would be a distributed but otherwise standard sound system that placed speakers among the nodes (perhaps mounted on the walls, interspersed among the display units) that are controlled using a standard 24-channel sound card on single computer. This would detract from the perception of the text as the source of the sound, and would have made the experience feel staged. Instead, we opted to mirror the architecture of the visual elements. Since each node was built around a full-fledged computer, it made sense to package a sound card and speaker on each node. Using an inexpensive custom USB audio interface and single speaker, each node was able to play sound at a relatively high amplitude, especially considering the speaker was less than 3cm in diameter. In the end, we also incorporated a small number of speakers (five on each wall, or ten in total), mounted just above the baseboards of each wall. These are used to generate ambient noises that are not necessarily tightly coupled with the display actions.

In fact, the speakers ended up being too efficient, and we found ourselves working at very low volumes to produce a useful dynamic range. 560 speakers even played at a low volume generates quite a bit of sound. By distributing

sound in this way, the audio and visual effects are tightly linked, text appears accompanied by the sound of a pencil moving across paper, or the stroke of a manual typewriter, even when standing within a half meter of a node. Also, this allowed the individual nodes could take on a variety of personalities rather than only having an overall audio soundscape.

In addition to its speaker, each node package included an automotive relay, included for the sole purpose of making a clicking sound. From our previous projects, we found that the physical relay sounds varied from device to device, adding a rich quality to the overall composition. While we had hoped to make use of the serial interface on each node to activate the relay, we found that this approach produced irregular, sluggish results since the timeslices of the Linux kernel were not fine enough to send very fast, regular pulses without jitter (i.e. less than 2ms). To get more accurate control, the relay was connected to the second channel of the audio interface.

2.1 Hardware specifications

Node packages. Each node in the grid measures 4.5"×8.5", and is a coupling of a vacuum fluorescent display or VFD (with resolution 128×256 pixels) and a single board computer. We chose PC-104 small form factor computers for a number of key reasons: they consume relatively little power, their size worked well with the displays, and (perhaps most importantly) their price fit our budget. Additionally, they produce little heat, their components are soldered together, and they have no moving parts, making them extremely reliable. For maximum flexibility and an adequate distribution of the data processing demands, each single board computer runs TS-LINUX[2], a GNU/Linux distribution provided by the manufacturer . Each node was built around a Technologic Systems TS-7250[3] embedded system, with a 200MHz ARM9 processor, 64 MB of RAM, and 128MB of flash for storage. They run a custom compiled version of Debian using a Linux 2.4.26-ts11kernel. The kernel includes ALSA support. The displays are Noritake 3000 Series VFDs[4], controlled via the standard RS232 serial port on the TS-7250 board. A custom Python module was created to allow for more intuitive access to the Noritake display functions. The sound is handled by a custom USB stereo interface and an embedded amplifier circuit. An 8ohm 1W speaker in its own plastic enclosure is attached on one channel, and on the other, an industrial relay used as a noisemaker.

As can be seen in Figure 1, six wires attach each column of 7 displays to the busway; The front pair carry the weight of the displays while two of the back pair carry power and two carry data. An RS485 interface provides the serial communications to all of the nodes. (In half duplex mode – one way communication – RS485 only requires 2 wires to carry it's data signal.) A central server located on the second floor sends instructions to the displays via a series of Comtrol DeviceMaster RTS ethernet-to-serial devices. Each pair of columns are on a separate RS485 circuit, making a total of 40 such circuits. We chose RS485 for its ability to function over very long cables (the central server is a floor away) and for its support for one-to-many communications (each circuit consists of two columns or 14 nodes). On each node, a custom Python daemon listens on the RS485 wire and directs messages to the display or audio subsystems or to the Linux OS itself.

Server side systems. The displays are controlled by a single Linux server communicating with the Comtrol device mentioned above. The Comtrol creates 40 serial devices, each associated with a pair of strands in the grid (40 pairs or 80 strands total). Within each pair, the nodes are assigned an address from 1 to 14 (via a dip switch, the settings of which are read at boot time). Each node responds only to messages addressed to it, with special addresses denoting the left column of the pair, the right column or all the devices in the circuit. A custom protocol was developed for directing messages around the grid, and custom Python code was used to hide the complexity of the serial ports and strand-based addressing, allowing direct matrix-style access to the grid elements (individual nodes and entire rows or columns). The server sends data, single instructions and even Python code snippets to the screens. The nodes do not send data (or any sort of acknowledgment) back to the server.

Timing is critical, as many of the scenes require a complex sequence of visual or acoustic effects. For this reason, a second Linux server is used to collect and prepare the data for display. All of the data scrapes and natural language functions are carried out by this second server. This computer is also tasked with generating

reports about system health that are pushed to a publicly visible Web site. Finally, a third computer, a Windows server, is used to schedule and initiate the different scenes via a Medialon show controller. This server is also running Max/MSP and generates audio for the ten channels of audio (five speakers mounted low along each wall) also available for scene design.

2.2 Software choices

Display units. PDa [5] was chosen for the audio software. PDa is a port of Pure Data, a graphical programming language for media, to ARM processors, which do not have floating point units. Instead of using very inefficient software emulation of a floating point unit, Geiger rewrote parts of Pd in order to use only integer math, allowing for efficient sound manipulation and synthesis on small CPUs. In exchange, PDa has some minor limitations, like using milliseconds instead of sample numbers for the control of audio buffers. Another important feature of PDa for this project was the ability to disable the entire GUI when PDa was running on each node, thereby reducing the memory and CPU footprint. We used PDa version 0.4, which only supports the OSS audio API, so ALSA was configured to use OSS emulation.

There are many options for lightweight sound playback on GNU/Linux, but PDa provides a lot more than just sound playback. It is capable a very wide range of synthesis and detailed control over sample playback, even on these very low power embedded machines. The sound used in Moveable Type ended up being a combination of samples and synthesized sound, so the added complexity of using PDa paid off in the composition. The relay mentioned above was also driven by PDa, adding another compositional element to the overall acoustic design. In addition, Rubin, the sound designer, had been using Max/MSP for over a decade. Since Pd/PDa are closely related to Max/MSP as programming languages, this allowed him to work on the embedded platform using his existing skills. Using X11, it was possible for Rubin to run PDa on the embedded machine while controlling it remotely.

Therefore, instead of editing patches on a desktop computer, then uploading them to run them, the GUI was displayed on the desktop computer while PDa was running on the embedded machine. This allows us to design the sound on a single node and then 'propagate' the patches to the entire grid once we were happy with the effect. This process mirrored the authoring setup we implemented for the visual elements (which involved testing then distributing Python scripts). We will have more to say about this at the end of the paper.

Server side systems. On the Linux server that communicates with the screens, we authored custom Python software for running the scenes, shipping data and code to the displays, and logging the overall operation of the system. Each scene is a Python module, that in turn depends on a base set of classes representing the grid (nodes, rows, columns) and the auxiliary 10-channel sound system. Given the uniqueness of our setup, we opted for custom software rather than an off-the-shelf solution, although we did make use of as many existing Python modules as possible (pySerial, and BeautifulSoup, for example, as well as standard built-in packages like re and random). We specifically chose a scripting language like Python because the same code would run directly on both the server and on the nodes without any special (re)compilation. This allowed us to very directly adjust the amount of computation taking place on the server versus the nodes.

As mentioned previously, the Windows server is equipped with a Medialon show controller and Max/MSP. The Python process on the Linux server communicates with Max/MSP via Open Sound Control[6].

3 Achieving Proliphony: Distributed, embedded Pd

We coin the term "proliphony" to describe the acoustic experience of 560 point sources of sound playing different notes. The workhorse of this effect is a custom sampler running on each node. Our nodes' CPU and RAM resources were extremely limited and these constraints only tightened when the Python communication and display scripts were running. It required considerable effort to pare down the sampler so that complex display effects did not monopolize resources and introduce artifacts into the audio stream. The sampler allows for a set of up to twelve samples to be used for a single voice. This means, for example, that each sample can be tailored to a given frequency range. This sampler patch was then used repeatedly to provide multiple instruments. Memory and CPU limitations, however, kept us from introducing more than 3 simultaneous instruments without

audible defects.

As mentioned previously, within a node, scenes are typically initiated and controlled by a Python script, and this code directs or "triggers" the sampler by sending messages to set up the samples, establish the root note of each sample, and set a cue where the sample is to start playing. With this framework, the same three sampler instruments could be reused for different scenes: Prior to each scene, the corresponding sample sets and configurations were sent to the samplers, preparing the samplers to generate a new range of sounds. The note events are then received from the nodes' Python scene code, possibly having been triggered by the central server. The messages to control the audio are basically MIDI notes received from the nodes' Python scripts via simple sockets.

Even with our careful coding, this setup often demanded all of the resources of a node's processor. As a result, periods of high activity could cause interruptions in the audio processing, creating noticeable clicks. The GNU/Linux distribution that was installed was stripped down to the bare minimum, so Unix commands "nice" and "renice" were both missing. Therefore setting process priority was not an easy option. Increasing the audio output buffer in PDa lessened the chances of audio interruptions, while adding latency to the audio response. Since the scenes are sequenced, each node's Python code sends a given command to PDa some known amount of time prior to triggering the text, thereby bringing the text and sound back into sync cleanly.

4 Authoring scenes

We have already mentioned the scene authoring process for the audio component of our installation. Specifically, a direct ethernet connection to a single node allowed us to invoke the PDa X11 GUI and work out a new scene's logic. Drafts of the new patch were distributed to the nodes using the data propagation mechanisms alluded to above (a patch being nothing more than an ASCII file). Once distributed to the nodes, commands would be sent to stop and restart the PDa daemon.

For display effects, the process was a little more detailed simply because we had an essential choice about where to perform a "computation." For example, during the scene exhibiting letters to the editor, we begin with a rhythmic typing of the phrase T-o-_-T-h-e-_-E-d-i-t-o-r with an accompanying sampled keystroke. One way to accomplish this would be to have the central Linux server send Python commands to each node instructing it to display a character and play a sample. An alternative approach would be to send (at some previous time, and only once) a piece of Python code that types out the whole phrase in the appropriate way, triggering samples at the right times. Then the central Linux server need only send along messages to execute the Python script. Given the relatively slow pacing of this part of the scene and our efficient communication protocols, both of these options turn out to perform similarly.

In the second part of this scene, however, each node is to type out a different letter to the editor. Here, it is simply not possible to direct all 560 nodes character-by-character. Instead, we send the text of each letter before the scene begins and then send a single message to the entire grid to begin typing out the separate letters. During scenes like this one, we found that we could send text to the nodes and not introduce visible or audible artifacts. Such "all over" compositions fill the hall with activity and it is very hard to see any hesitation as the nodes receive data. One complication of this approach, however, is that we have to be somewhat careful with the ends of each scene. As the nodes do not communicate at all, we have to estimate when the scenes will complete (we introduce random pauses between the keystrokes in the body of each letter, for example) and then assign the nodes an overall time budget so that the action dies down after some number of minutes and we can trigger an end-of-scene effect, confident that in fact the scene had ended.

To make these coding decision simple, we began our display coding by first issuing instructions entirely from the central Linux server. An open Python exec loop running on the nodes let us send commands line-by-line to the grid, so that if timing became an issue, we could start to send single lines or small sub-programs to the nodes to be executed. Once the balance between central server and nodes was established, code on the nodes was put into a module and installed in the nodes directory structure. This description leaves out a number of details, but we hope that we have captured the spirit of the enterprise.

5 Conclusion

Moveable Type is a complex instrument, offering incredible possibilities for the activation of

text. We believe that the ability to create so much varied sound makes it unique. Our brief experience with the installation suggests that one can build a remarkably rich visual and sonic experience using 200 MHz computer, or rather 560 such computers. We have also found that the combination of Python and PDa on a GNU/Linux system made for a robust, flexible and expressive system. While this choice meant a great deal of up-front custom coding, the ultimate return on this investment was incredible.

6 Acknowledgments

We are indebted to the engineering prowess of Marty Chafkin (Perfection Electricks); and Olaaf Rossi, Chris Keitel and Josh Silverman (Three Byte Intermedia). Renzo Piano and his design team provided invaluable artistic guidance and support for the project; as did Brian Ripel and George Showman (RSVP Studio) and Peter Zuspan. Finally, we are grateful for the support of our patrons, The New York Times and Forest City Ratner Companies, owners of The New York Times Building.

References

[1] RS485.
`www.rs485.com/rs485spec.html`.

[2] TS-LINUX.
`embeddedarm.com/linux/main.htm`.

[3] Technologic Systems TS-7250.
`embeddedarm.com/epc/prod_main.htm`.

[4] Noritake 3000 Series.
`noritake-elec.com/3000_series.htm`.

[5] Günter Geiger. Pda: Real time signal processing and sound generation on handheld devices. In *Proceedings of the International Computer Music Conference*, Singapore, 2003.

[6] Open Sound Control.
`www.opensoundcontrol.org`.