Joey Heinen
12/13/13

Brotherhood: A Case Study in Emulation and Preservation of MIDI

The oft cited expression in the field of archiving states "preservation without access is pointless."[1]. The concept of "access" is conventionally applied to the physical elements and, in the case of moving image or dynamic media, the essence or digital stream of the associated media. In the case of complex media art, specifically custom-designed software and coding languages, the notion of access takes on a unique new set of definitions given its complexities and interdependencies. In acknowledging the breakneck pace in which new technologies and programming languages develop paralleled with the methodical upkeep of obsolete operating systems and programming interfaces, its entirely possible that the "essence" of these artworks will gradually be left behind or evolve to a place where the "original" work is no longer evident. Similarly, computer-based art works are in some ways at the mercy of the current state of technology, and works that incorporate these technologies are often essential products of their time. Will future emulation or data migration treat these works kindly when viewed in a contemporary setting? Is it not just as important to acknowledge that future instantiations of the same work, whether based on the same structure as the original or not, are in fact each new and unique expressions? Perhaps in the example of computer-based art that "access" means not just availability of the essence in a physical setting (i.e. the gallery) but documentation of the historical and technological subjectivity of the original and each successful iteration of the work.

The interactive installations of Steina and Woody Vasulka were some of the first examples of artwork which incorporated software programs that were designed to yield robotic or audio-visual reactions in response to the visitor. For their project *The Brotherhood,* they designed this concept long before there were tools on the market to sense and interpret human interaction in a language that a computer could understand (e.g. Arduino), a big leap of faith for everyone involved. In order to achieve this, they engaged in a relationship with Russ Gritzo, an electrical engineer at Los Alamos Labs, in the early 1990s who designed a software program to achieve this goal.[2] The essence of the three programs (INTERCOM, UNICOM, MINICOM), was a means of synthesizing multiple sensors which were fed through an algorithm for multiple and sometimes random reactions. The distinction between these three software programs and their relation to one another will be discussed later. The installation was realized in a six-part series, each part incorporating a different means of taking user-generated inputs and translating them into robotic or algorithmic outputs. *The Maiden*, part IV of this installation, asks the visitor to speak into a microphone in order to activate an electro-pneutronic sculpture. Based on the velocity and pitch of the sounds generated by the visitor, the sculpture will expand, contract, and move in ways that mirror the visitor's

---

[1] The Committee for Film Preservation and Public Access, "Preservation Without Access Is Pointless," 1993. http://www.loc.gov/film/pdfs/fcmtefilmprespubaccess.pdf, last accessed 12/13/13

[2] Vincent Bonin, "Woody Vasulka: The Brotherhood," 2001. http://www.fondation-langlois.org/html/e/page.php?NumPage=464, last accessed 12/13/13

speaking voice. Other examples of *Brotherhood* involve user interfaces such as drum pads, remotes, alphanumeric keyboards, and tables and outputs such as video sequencers, light displays, roving cameras/spotlights, and laser transcription.

The language for translating information from the sensors to the computer's OS language and out to the related hardware was written in C with sensor inputs and computational outputs expressed through the language of MIDI. MIDI (Musical Instrument Digital Interface) was one of the first protocols for translating analog musical notation into a digital bit structure. Russ Gritzo admitted that at that time MIDI proved to be an unconventional approach towards abstracting sensor outputs given that the language is more conventionally used to generate sound scores that to be a language of spatial and movement commands.[3] Nonetheless, it was important to the Vasulkas that the interaction between humans and machine was organic and did not come off as sterile or *too* responsive. In other words, they wanted it to appear that the machine or computer was reacting to the user as a human might, not how a subservient and exacting machine might. As MIDI is based on a continuos string of user-generated data, a concept that will also be explored in full later, this meant that the machines could respond in a continuous, algorithmic way rather than simply performing a pre-programmed menu of tricks. Reactions would be precise enough that it would be clear to the visitor that their impulses yielded clear and tangible results but organic enough that even slight changes in nuances would alter the responses, creating a seemingly autonomous reaction from the machine.

Another important aspect of this piece was based on the capabilities of operating systems for microcomputers that were available at this time. An essential aspect of this piece was an Interrupt Service Routine (ISR) which, in the early 90s, could only be found in DOS. An ISR essentially aided the computer in clearing the queue of user-generated impulses and acts of translating those inputs into outputs. Since sensors needed to be piping in new user-generated data continuously so as to appear responsive, there needed to be a way for this data to refresh itself while still being run off the same script.[4] By interrupting this flow of information the computer could understand these impulses as unique data strings and not as endless commands which would otherwise be difficult to program and result in lost or ignored commands. Of course, later OS models began to incorporate ISR and were more broadly adopted sparking a shift from DOS to Linux. The shift to Linux was also sparked by its uptake as the preferred OS in more modern computers throughout the 90s, giving Linux an advantage as computer processing speeds and networking capabilities developed more quickly.[5]

A final major consideration, if not more significant than OS in how this piece is technologically grounded, is the related hardware. It would perhaps be misleading to say that ISR was introduced with DOS when in fact it was a feature that was made possible by new achievements in microprocessors. The Intel 8086 was the first

---

[3] Russ Gritzo, personal interview, 08/05/13

[4] BonaFide OS Development "Interrupts, Exceptions, and IDTs: Part 1 - Interrupts, ISRs," 2003. http://www.osdever.net/tutorials/interrupts.1.php, last accessed 12/13/13

[5] Russ Gritzo, personal interview, 08/05/13

microprocessor chip that was designed with ISR capabilities that was implemented on a microcomputer[6], the IBM 8086 being the first computer on which methods for *The Brotherhood* were tested. It was also one of the most widely adopted chips in PCs throughout the 80s and early 90s. As greater memory, processing speed, and portability became a requirement, the team moved on to use the Toshiba T1200XE which also used the Intel 8086 chip.[7]

Given the unique timeline of this six-part work amidst an era of great computational development, and also the unusual incorporation of MIDI as a programming language, this piece was highly ahead of its time and is grounded in some rather precise methods in computer science. Fortunately the source code and configuration for these installations were very well-documented and communication between the Vasulkas and Russ Gritzo is ongoing. Nonetheless, investigation of the feasibility for emulation or compiling will require further exploration as we consider the precise interdependencies of the hardware and software elements.

Emulation is considered one of three broadly-defined approaches towards providing access to digital material, the other two being migration and reinterpretation.[8] Of the three, migration is considered to be more of a preservation approach as it involves translating the original code and file formats into more widely-adopted and renderable digital streams and wrappers, ensuring long-term access to the material. Of course, in approaching this from the perspective of an installation artwork where behaviors of software or the aesthetic front-end interface of a piece may be more vital than the actual code and back-end processes, migration may alter the essence of the piece by stripping it of its original dependencies - in fact, in some instances it may result in a bitstream that is completely inoperable since it requires specific hardware and operating systems under which to run.[9] Reinterpretation involves rewriting the code or introducing new interpreters in order to accomplish nearly identical results but in using different languages, processors or platforms - essentially different means to accomplish the same ends. Naturally this is not considered a preservation approach as it introduces enormous questions of authenticity given the fact that the original bitstream is in no way maintained across successive iterations of the piece. For this reason, reinterpretation is chosen more as an access approach and more often in instances of a last-ditch effort for a physical reinstallation of the work. Some of these access approaches can be used in tandem, such as Grahame Weinbren and Roberta Friedman's *The Erl King* which was reinstalled for the Guggenheim's exhibit *Seeing Double: Art and Emulation In Practice.* Acknowledged as "not a true emulation," the installation team (including Weinbren and Friedman) concluded that the source code, a compiled version of

---

[6] Benj Edwards, "Birth of a Standard: The Intel 8086 Microprocessor," http://www.pcworld.com/article/146957/article.html, last accessed 12/13/13

[7] Russ Gritzo, "ExhFour," http://www.vasulka.org/archive/ExhFOUR/Brno/brno.pdf, last accessed 12/13/13

[8] Ben Fino-Radin, "Digital Preservation Practices and the Rhizome Art Base," http://media.rhizome.org/artbase/documents/Digital-Preservation-Practices-and-the-Rhizome-ArtBase.pdf, last accessed 12/13/13

[9] Jeff Rothenberg, "Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation," http://www.clir.org/pubs/reports/rothenberg/pub77.pdf, last accessed 12/13/13

PASCAL, was not necessarily hardware or software dependent but could be interpreted through a custom-written program (written in Java). This approach resembles a combination of both migration and reinterpretation as original elements were transferred to new hardware and OS platforms while also incorporating new programs in concert with the original code. The guiding ethos was to recreate, or "emulate," the original functions of *The Erl King* using the most efficient means possible with the least amount of overhead.[10]

Of course, we still need to turn our attention towards what emulation means from the perspective of digital authenticity and how this relates to or differs from compiling, virtualizing, and interpreting/translating code. Let us first consider two fundamental differences in how code is written and read, namely source code versus object code. Source code is generally that which is written by a programmer in a specified language, which further implies that it is in some way human readable. It uses commands and values which are programmed to perform a specific function using a controlled language. Object code is that of the bitstream, encoding the source code commands in a binary language native to the host computer, otherwise known as the machine language. This language is "the native tongue of the processor in that computer: it consists of much simpler commands than those found in most source code." It follows that machine code can not adeptly jump from processor to processor given that the binary language will mean something entirely different. Thus, the source code is what must be used in order to perform the same computational functions. The act of taking the human-readable source code and translating it into the language of the native processor is what is referred to as compiling. The process of compiling source code into object code is what causes the program to actually run.[11]

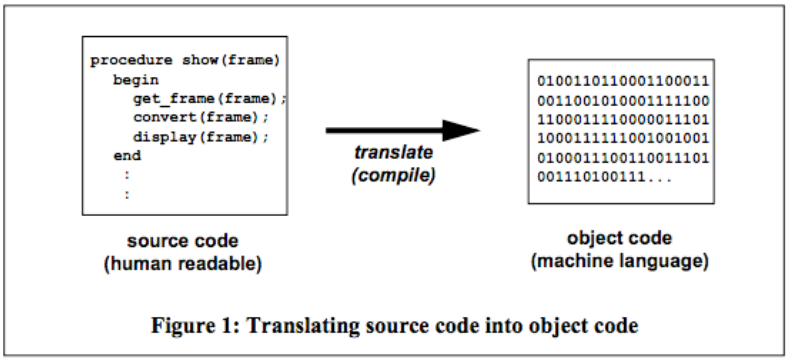


Figure 1: Translating source code into object code

A further analysis of emulation can be found in dissecting this process of interpretation. Jeff Rothenberg, a strong advocate for the use of emulation in an archival context, provides a useful analogy in citing the spoken language. Though a translator and an interpreter are roles which are described somewhat interchangeably, they in fact differ in a distinct way. A "translator" merely translates one language into another

[10] Naturally, the fact that the artists were involved in the reinstallation of the piece insinuates consent for an evolution of the piece over time. Necessary documentation of possible detours in reinstalling a piece should be secured from the artists in case the piece is to be reinstalled in absentia.

[11] Jeff Rothenberg, "Renewing *The Erl King*," 2006. http://bampfa.berkeley.edu/about/ErlKingReport.pdf, last accessed 12/13/13

whereas an "interpreter" translates and then performs that action.[12] Following this example we can find a more concise definition of emulation. Once source code can no longer be compiled into object code on existing hardware, it becomes necessary to recreate that original hardware as code in and of itself. By translating the means by which machine code is compiled, you can then create an environment for that source code to perform its function. Emulation recreates a target/host system through which the code can be interpreted, in other words it translates code into something more receptive for a given set of instructions.[13] Interpretation would simply be the processes through which the code is run within the emulation. The Rosetta Stone is a useful analogy in this context. While the stone exists as a tool for translating hieroglyphics, it uses an obsolete language to translate it and is bound by the physicality of the stone that carries the language. We need a human-readable set of instructions that can compile this information rather than a blocky, proprietary language. This in effect is what happens when machine code is obsolete and is no longer interpretable - it is nothing more than a stone.

These examples have shown how hardware emulation is key if we are to perform the code, particularly given that machine code is native to its original hardware. However, Virtual Machines offer one slight exception to this rule. Virtualizing and emulation aim at essentially the same goal but are slightly different in their methods. A simple explanation is provided by Steward Granger in that "a VM [(Virtualized Machine)] does essentially the same thing as an emulator, except that an emulator emulates some other real machine, whereas a VM typically implements a computer that has never existed as hardware. We usually say that an emulator "emulates" another computer, whereas we usually say that a VM 'is implemented' on a computer."[14]

We now see how hardware is essential in running whatever software is desired, regardless of how the operating systems (OS) and other applications function. However, there are still important considerations related to OS and applications that must be considered on top of any necessary emulation that happens at the hardware level as thus far we have only considered software-emulating-hardware. An operating system provides the necessary language for making the computer user-operable, namely providing interfaces, file systems, interprocess communication, and networking (to name a few).[15] Applications are what facilitate rendering of objects and commands at the program level rather than at the OS macro level and also aid in configuring new software and add-ons. Naturally, both the operating system and the inherent applications run on top of the hardware so this must be emulated first. This is not to say that OS and Applications are not just as essential in emulation but that they must be grounded upon something stable. A useful analogy in this situation would be plans for building a bridge. You need two physical points to connect (hence the need for the bridge in the first place as it extends over a river or ditch) and must be anchored upon

---

[12] Jeff Rothenberg, "Renewing the Erl King," 2006.
[13] Ibid
[14] Digitale Bewaring "Emulation: Context and Current Status," 2003. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=6008E32930A04C717C34832A5319D8F7?doi=10.1.1.132.5566&rep=rep1&type=pdf, last accessed 12/13/13
[15] Ibid

these points. Nonetheless, without that bridge you still have the difficulty of traversing the terrain. In other words, you may be able to emulate hardware but you will still need a means of accessing the potential of that hardware and using a language with which to communicate with the machine.

In the case of *The Brotherhood* in which several computers are serving respective and simultaneous parts yet running on a singular network, running emulators that can also communicate across a network is fundamental. Similarly, the distinct programs within each piece (commands for video switchers, voice recognition programs, MIDI adapters) must be configured, and the interfaces with these programs will also need to be functional. This last example brings up issues of related hardware, somewhat out of the scope of this discussion but will nonetheless be a consideration in successive stages of the work plan. Though, at this stage we will dutifully turn our attention to the specific programs within *The Brotherhood* and how the language of MIDI throws one more complication into the mix.

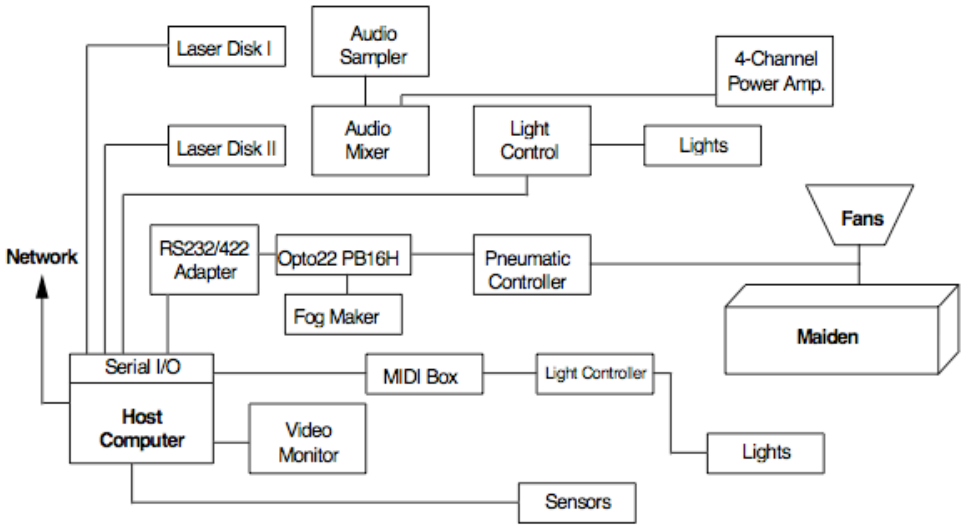


Figure IV-4. MAIDEN System Schematic

Block Diagram for installation of *The Maiden*

Block Diagram for Installation of *Friendly Fire*

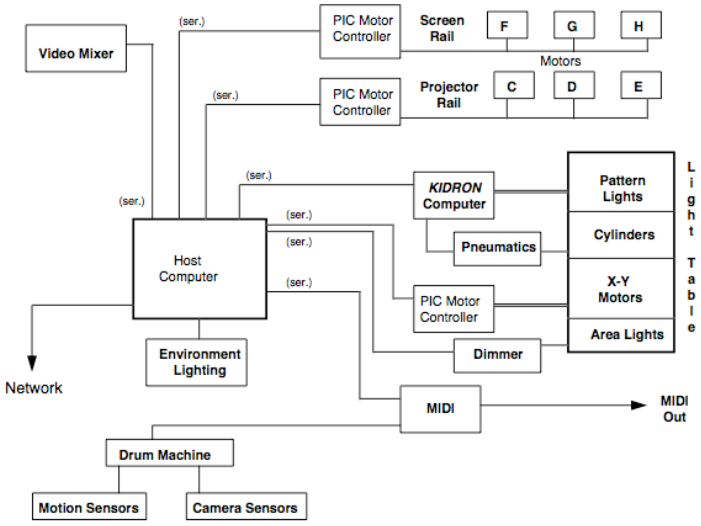
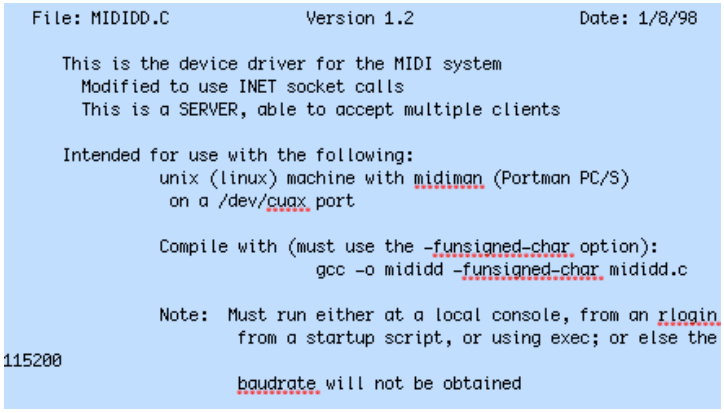


Figure III-6. The RAILS System

The above diagrams demonstrate the centrality of the host computer in the artwork, which further stresses the importance of machine code emulation for future installation. Of course, given all the other external elements that must interface with the hardware, considerations will also need to made for OS and application emulation. As noted earlier, MIDI is an essential component of this installation. Both diagrams show how MIDI is used to translate the sensor inputs to the host computer which in turn trigger the other lighting, video, and audio reactions. Within the technical documentation for *The Brotherhood* one finds many documents containing commands written in C, many of which specify configuration of device drivers for the MIDI system and necessary ports and startup commands. Given how many programs and peripherals run

in concert with one another, it was necessary for the Vasulkas and Gritzo to create three separate software packages. MINICOM is a program that configures the host computer with the MIDI interfaces. Updates to MINICOM across successive instantiations of *The Brotherhood* elevated the number of MIDI input values from 128 to 256. INTERCOM is a program that facilitates configuration in booting up the system and initializing command sequences. Essentially it wakes up the entire system and ensures that all essential parts are communicating with the host computer. UNICOM was later developed once multiple parts of *The Brotherhood* were displayed simultaneously, networking each dedicated host computer such that all installations could be activated through one central booting process. Clearly there is an immense amount of translation and interpretation occurring in this piece and a successful emulation of the piece will need to consider all programs, hardware interfaces (and in some cases the hardware itself, such as obsolete Laserdisc players), and the three essential software programs. For the purposes of exploring the essential programming language for each individual artwork in the series of six and the inherent preservable bitstreams, we will solely turn our attention to MINICOM and the use of MIDI as an I/O protocol.[16]



```
File: MIDIDD.C              Version 1.2                Date: 1/8/98

    This is the device driver for the MIDI system
      Modified to use INET socket calls
      This is a SERVER, able to accept multiple clients

    Intended for use with the following:
              unix (linux) machine with midiman (Portman PC/S)
               on a /dev/cuax port

              Compile with (must use the -funsigned-char option):
                            gcc -o mididd -funsigned-char mididd.c

              Note:  Must run either at a local console, from an rlogin
                     from a startup script, or using exec; or else the
115200
                     baudrate will not be obtained
```

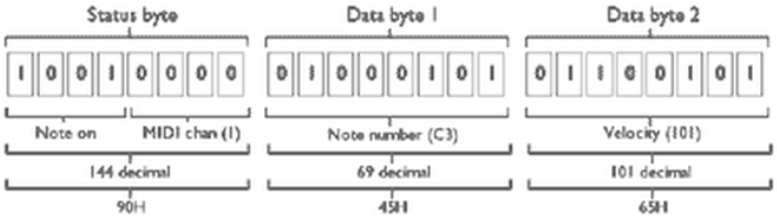
Before the invention of MIDI, the discussion of digital music and appropriate standards for analog conversion was already a major discussion amongst computer programmers in the late 1970s. The introduction of desktop microcomputers in the early 1980s, most prominently the IBM 8086/8088, established a commercially viable means for experimenting with synthesized digital music. While at first blush this may seem coincidental given that the IBM 8086 used the Intel 8086 necessary for Interrupt Service Routine, one can quickly see how this is also an essential concept in the development of MIDI and constant streams of executable data. It had already been established that digital synthesis required a means of translating pressure-wave characters into numerical samples, fidelity based on both an analog and digital means of representing rate (frequency) and intensity (amplitude).[17] With the new microprocessors, a means of connecting to a memory bank and peripherals such as, but not limited to, a disk drive,

[16] Russ Gritzo, "Brotherhood: All Tools," 1995. http://www.vasulka.org/archive/Vasulkas3/Installations/BrotherhoodAll/Tools.pdf, last accessed 12/13/13

[17] Peter Manning, *Electronic and Computer Music,* p. 277, 1993. Clarendon Press: Oxford, New York.

alphanumeric keyboard, and visual display was necessary in order to be user-friendly and self-contained, which furthermore meant that programming could allow for external musical devices that were complementary to the existing protocols.[18] MIDI was originally designed to be unidirectional across a single cable, sending encoded alphanumeric messages to the MIDI OUT port, translating these commands into bytes, and then interpreting these back into the alphanumeric characters at the MIDI IN port. This was finally realized in the form of the PC/S (Personal Computer/Serial) connection, which in the case of *The Brotherhood* would connect MIDI out to the RS-232 port of the IBM computer.[19]

As mentioned earlier, the use of MIDI was highly inventive for this project given its use as a non-musical manifestation but rather as a musical delivery output which was encoded to yield robotic outcomes based on the contents of the bitstream. MIDI is an 8-bit protocol meaning that each byte is comprised of a cluster of 8 bits. Bytes can take one of two forms in a basic MIDI message, or a means of packaging an analog input into some sort of digital output. These two types are either Status or Data. Status indicates the type of message being sent, examples being whether the note is on or off (in other words, indicating the start and end of a single command). Data includes the content of the message, in most cases the pitch and velocity (volume and degree of accent) of the note though can go on to include information related to voice, tremolo, sound dampening, and so on. Furthermore, complete MIDI messages can fall into one of two categories, Channel Messages and System Messages. The majority of messages are channel-based in which the discrete values indicate a combination of pitch and voice effects whereas System Messages include metadata to be encapsulated in the recorded MIDI file such as source and timing.[20]



In order to argue for the long-term viability of MIDI as programming language, it must be held to sustainability standards based on documentation, adoption, transparency, and dependencies. The Library of Congress' Sustainability of Digital Formats observes that MIDI is fully documented and that the format is widely adopted making it likely that it will be continually supported (particularly in considering how long this format has been around). It is, however, worth noting that the Library of Congress does not themselves

[18] Ibid, p. 278
[19] MIDIman, "Portman PC/S Manual," 1998. http://www.m-audio.com/images/global/manuals/PortmanPCS_Manual.pdf, last accessed 12/13/13
[20] Planet of Tune, "Index of Sequences," 2013. http://www.planetoftunes.com/sequence/se_media/message.gif, last accessed 12/13/13

use or maintain any MIDI digital objects.[21] MIDI streams are expressed and recorded in several different types of file formats. For the purposes of *The Brotherhood* we are concerned with some specific formats, including Standard MIDI Format (.SMF), Downloadable Sounds Format (.DLS), Extensible Music Format (.XMF) and MIDI executable (.MID). SMF is the most simple file format as it contains a relatively straightforward and self-sufficient bitstream indicating the instrument that created the signal and the specs for the message (pitch/velocity in its simplest form). However, this information must travel along with other file formats for purposes of compiling or parsing out data contained in the SMF, particular when simultaneously generated messages need to occur in concert with one another. The DLS, for example, defines the differing structure and sources of multiple notes that are being sent to the same source, in some cases occurring at the same time and represented in different voices or timbres. Information contained in this format help to separate out the individual streams such that they can retain their individuality from one another. This is furthermore facilitated by the XMF file which ensures that all related streams travel together along the same pathway and contains information on synchronicity of the inherent messages. If two notes or two different voices should occur at the same time, the XMF file will encapsulate this information.[22]

Last but not least is the MID file, an executable file format which essentially is what contains information for enacting this information on the hardware of the computer. As mentioned the PC/S instrument is necessary for translating the analog output into a binary stream for decoding on a Personal Computer. On the PC end of things this, of course, is not an automatic process as the MIDI specs need to be configured to the system. This requires installation of the MIDI driver on to the OS which is configured to the .MID format as a delivery device.[23] This allows the actual code and the various means of encapsulating the MIDI stream to run on the related hardware.

Given the highly interdependent nature of MIDI-based files it is clear that dependencies (in terms of LOC's Sustainability Standards) become somewhat of an issue despite the fact that MIDI scores well on all other accounts. In terms of how MIDI is generally used as a compositional format, its use in *The Brotherhood* is actually quite straightforward - it is merely a means of mapping values to outcomes, in this case matrix value outputs from sensors within a computer program to connect with and yield specific results from a video sequencer, light control switch, or camera. This meant that pitch and velocity were often the only data bytes necessary. Given the relative simplicity of this transmission code, there may be ways to circumvent MIDI if it became absolutely necessary for reinstallation. While this obviously brings up the issues of authenticity mentioned along with reinterpretation, I would argue that this is a worthwhile venture if it is determined that one aspect of the code or software elements is can be isolated and easily replaced on the back-end. The use of MIDI greatly resembles more common uses

---

[21] Library of Congress Sustainability of Digital Formats, "MIDI File Formats," 2013. http://www.digitalpreservation.gov/formats/fdd/fdd000119.shtml, last accessed 12/13/13

[22] MIDI Manufacturers Associations, "The Complete MIDI 1.0 Detailed Specification," 1995-2013. http://www.midi.org/techspecs/midispec.php, last accessed 12/13/13

[23] MIDIman "Portman PC/S manual," http://www.m-audio.com/images/global/manuals/PortmanPCS_Manual.pdf

of Arduinos amongst today's software-based artists. Arduino is an open-source electronic prototyping platform, basically a device that can read data from a vast number of sensors and can be programmed to software for mapping that data to the desired computational or mechanical output.[24] In essence an Arduiono could be used along with another software program (e.g. Max MSP) to emulate the functions of MIDI.[25] The original controllers such as a keyboard or drum pad could still be used but supplemented with a more modern means of translating that data to the computer's language.

At this stage it might be helpful to harken back to our earlier example of *The Erl King* and another example of a successful emulation and reinstallation of a complex software-based artwork. The Dutch collective jodi created a work entitled *Jet Set Willy interpretations* which consequently was displayed along with the "emulated"[26] version of *The Erl King* at the Guggenheim's exhibit *Seeing Double. Jet Set Willy* was a popular video game in the 1980s specifically designed for the ZX Spectrum computer. The plot of the games involves Miner Willy who must clean up a mansion the day after a wild party. Jodi decided to emulate this game for sentimental reasons and also given the fact that the game was widely modified such that the artists felt that there would be little litigious backlash (also considering that *Seeing Double* was staged in 2004, some 20 years after the game was introduced and consequently quickly obsolesced). While the ZX Spectrum has its own emulator on the market, the piece could not be easily emulated since the game relied on configuration with the custom-designed keyboard which was no longer in operation or circulation. The necessary key commands for creating the game would no longer configure with modern keyboard and there was no means of emulating these peripheral communications. To circumvent this, jodi worked with the source code (written in BASIC) through the ZX Spectrum emulator to access and analyze the machine code, able to edit the code such that it could adapt to modern keyboards and their inherent commands.[27] This is another example of how reinterpretation is used, and of course authenticity brings up a major concern. However, an argument can certainly be made for elements within a piece that can easily be altered without necessarily effecting the piece from an operational standpoint. Of course, the aesthetics of the piece are changed with a new keyboard or other new pieces of hardware, but these new implementations are minimally impactful (or at least as minimal as possible). The artist intention is also of key significance here since they may agree to a degree of evolution for the artwork and documentation of the original elements as a means of pointing out their existence is often the only way to ensure this livelihood.

---

[24] Arduino, "What Can Arduino Do?" 2013. http://arduino.cc/, last accessed 12/13/13

[25] Arduino, "MIDI Note Player," 2013. http://arduino.cc/en/Tutorial/Midi?from=Tutorial.MIDI, last accessed 12/13/13

[26] You will recall that *The Erl King* was not a true emulation but in fact a combined migration and reinterpretation process. Nonetheless, the behaviors of the artwork and the experience of the artwork was considered much more significant than the related hardware.

[27] Guggenheim Museum, "Seeing Double: Emulation in Theory and Practice," 2006. http://www.variablemedia.net/e/seeingdouble/, last accessed 12/13/13

“Conclusions” or Hypotheses Moving Forward

After examining how emulation works and is used in accessing computer-based art in an installation setting, and also examining what necessary elements are at stake in *The Brotherhood*, it can be deduced that the means of compiling the source code (i.e. the host hardware) is the most at-risk given the wide adoption of the Linux OS and C as a programming language. Additionally, the source code is extremely well-documented but none of this has any value unless it can be interpreted as machine code. Thus emulation of the Intel 8086 microprocessor is of utmost importance. Considering that the MIDI protocol was developed along with the 8086, somewhat ensuring that the original MIDI functions will hold true along with the microprocessing function, and that the 8086 was continuously used throughout the lifecycle of *The Brotherhood,* this may be the silver bullet we are looking for. Fortunately there is an emulator on the market - EMU8086 - and it comes with a built-in Virtual PC VM for incorporating a menu of possible operating systems (including Linux). It would seem this could be a fruitful process, but much work needs to be done to analyze all the various configurations of MIDI and the other related hardware to ensure functionality. Though, if MIDI proves to be challenging there is also the possibility of testing an Arduino to see if this process can also be emulated. This process will naturally need to be vetted against the artists and their feelings towards reinterpretation but seems like a strong possibility.

Of course, there are still a large number of considerations that lay outside this analysis that require much more exploration. First, there is the peripheral hardware of the artwork - the electro-pneutronic sculpture, the roving camera, the MIDI instruments themselves - that must be maintained and must ultimately communicate with the software. There is also the bidirectional nature of the programming given that many inputs and outputs are sending and receiving signals at the same time, all of which must be fed through the same software. The presence of “in-line code” or code that is fed back from peripheral machines and computers is often difficult to control in emulation as was found in some pilot versions of *The Erl King* (perhaps another reason why they chose the route of reinterpretation). Additionally, in proposing reinterpretation through an Arduino there is still the difficulty of communicating across Virtual Machines since these may not run on the EMU8086 but would need to be run off a network system. In considering the network, there is also the consideration of the simultaneity of the pieces, namely the six separate installations that all feed through one server via the UNICOM software. This along with the booting processes employed by INTERCOM are necessary as we examine the EMU8086 more closely.

Nonetheless, we have identified a promising solution that considers perhaps the most essential element of emulation - compiling into machine code. By establishing the hardware we can ensure that a necessary armature is in place such that further dependencies can be explored and, based on the exploration of MIDI and the configuration of the piece, these assumptions are not completely unwarranted. The immediate thought may be that this illustration serves for the purposes of a reinstallation of *The Brotherhood* though this is not necessarily the case. Jeff Rothenberg makes

further claims that Emulation can in fact be thought of as an avenue for "a viable technical foundation for digital preservation," looking for solutions to rendering and access digital objects without the "heroic" efforts of ongoing migration and establishing new formats and standards.[28] Though the process articulated for reinvigorating *The Brotherhood* may involve a similar quality of heroism, it is also the safest if not the only means of ensuring some form of authenticity. As media art works (and, indeed, maybe all digital objects) become increasingly more complex over time, emulation can be an example of necessary change with a simultaneous rigor for the past.

Webography

Arduino, "What Can Arduino Do?" 2013. http://arduino.cc/, last accessed 12/13/13

BonaFide OS Development "Interrupts, Exceptions, and IDTs: Part 1 - Interrupts, ISRs," 2003. http://www.osdever.net/tutorials/interrupts.1.php, last accessed 12/13/13

Vincent Bonin, "Woody Vasulka: The Brotherhood," 2001. http://www.fondation-langlois.org/html/e/page.php?NumPage=464, last accessed 12/13/13

The Committee for Film Preservation and Public Access, "Preservation Without Access Is Pointless," 1993. http://www.loc.gov/film/pdfs/fcmtefilmprespubaccess.pdf, last accessed 12/13/13

Digitale Bewaring "Emulation: Context and Current Status," 2003. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=6008E32930A04C717C34832A5319D8F7?doi=10.1.1.132.5566&rep=rep1&type=pdf, last accessed 12/13/13

Benj Edwards, "Birth of a Standard: The Intel 8086 Microprocessor," http://www.pcworld.com/article/146957/article.html, last accessed 12/13/13

Ben Fino-Radin, "Digital Preservation Practices and the Rhizome Art Base," http://media.rhizome.org/artbase/documents/Digital-Preservation-Practices-and-the-Rhizome-ArtBase.pdf, last accessed 12/13/13

Guggenheim Museum, "Seeing Double: Emulation in Theory and Practice," 2006. http://www.variablemedia.net/e/seeingdouble/, last accessed 12/13/13

Russ Gritzo, "Brotherhood: All Tools," 1995. http://www.vasulka.org/archive/Vasulkas3/Installations/BrotherhoodAll/Tools.pdf, last accessed 12/13/13

Russ Gritzo, "ExhFour," http://www.vasulka.org/archive/ExhFOUR/Brno/brno.pdf, last accessed 12/13/13

Library of Congress Sustainability of Digital Formats, "MIDI File Formats," 2013. http://www.digitalpreservation.gov/formats/fdd/fdd000119.shtml, last accessed 12/13/13

MIDI Manufacturers Associations, "The Complete MIDI 1.0 Detailed Specification," 1995-2013. http://www.midi.org/techspecs/midispec.php, last accessed 12/13/13

[28] Jeff Rothenberg, "Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation," http://www.clir.org/pubs/reports/rothenberg/pub77.pdf, last accessed 12/13/13

MIDIman, “Portman PC/S Manual,” 1998. http://www.m-audio.com/images/global/manuals/PortmanPCS_Manual.pdf, last accessed 12/13/13

Planet of Tune, “Index of Sequences,” 2013. http://www.planetoftunes.com/sequence/se_media/message.gif, last accessed 12/13/13

Jeff Rothenberg, “Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation,” http://www.clir.org/pubs/reports/rothenberg/pub77.pdf, last accessed 12/13/13

Jeff Rothenberg, “Renewing *The Erl King*,” 2006. http://bampfa.berkeley.edu/about/ErlKingReport.pdf, last accessed 12/13/13