

Art Hacks and Mash-Up Play: Introducing BitFlows

by

Robert Blair King

BFA New Media
Ryerson University
2005

A thesis project presented to
Ryerson University and York University
in partial fulfillment of the
requirements for the degree of
Master of Arts
in the Program of
Communication and Culture

Toronto, Ontario, Canada, 2008
© Robert Blair King 2008

Author's declaration

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University and York University to lend this thesis or dissertation to other institutions or individuals for the purpose of scholarly research.

Robert Blair King

I further authorize Ryerson University and York University to reproduce this thesis or dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Robert Blair King

Abstract - Art Hacks and Mash-Up Play: Introducing BitFlows

Robert Blair King

Master of Arts

Communication and Culture

Ryerson University and York University

A new software tool called BitFlows has been developed to support creativity, collaboration, performance and innovation in New Media. New Media practitioners already have a diverse range of tools at their disposal. This range of tools is constantly growing fueled by hardware and software hacks, which allow individuals to creatively use and abuse consumer products in ways not intended by their original creators. Software such as Ableton Live, Max/MSP and VVV give creators the ability to perform and demonstrate works in a live setting. Influenced by Csikszentmihalyi's concept of Flow in creative work (Csikszentmihalyi, 1996) and Shneiderman's suggestion that creativity can be aided by smoother flow *between* applications (Shneiderman, 2000a), BitFlows provides a simple means for users to mash-up the data-flows from all of these diverse pieces of hardware and software, over the network, in single or collaborative settings.

Acknowledgements

I would like to thank my advisor, Dr. Michael Murphy for providing encouragement and support throughout the past three years. Greg Elmer who first suggested I enter this program. Jerry Durlak for joining us at my defence while still in recovery. Bob and Glenna King, who provided constant encouragement through good times and bad. And finally I would like to thank Gwen Potter without whose patience, understanding and support I could never have done this.

Table of Contents

I.Introduction	1
II.Four Themes in New Media Culture	4
II.a.The Hack.....	5
II.b.The Mash-Up.....	11
II.c.The Demo.....	14
II.d.The Network.....	17
III.Creativity, Flow and Play.....	24
IV.BitFlows	32
IV.a.The Issues.....	33
IV.b.Introducing BitFlows.....	38
IV.c.User Interface.....	49
IV.d.Using BitFlows.....	59
V.Future Directions	65
VI.Conclusion	68

List of Figures

Figure 1: The user interface for BitFlows.....	52
Figure 2: Three uses of identicons in BitFlows.....	53
Figure 3: The use of sparklines in BitFlows.....	55
Figure 4: A list of plug-ins available to a BitFlows user.....	59
Figure 5: Six instantiated BitFlows plug-ins (nodes).....	60
Figure 6: Adding a pin to a flow.....	61
Figure 7: Dragging a pin to a flow.....	61
Figure 8: A simple text-to-speech instrument built entirely in BitFlows.....	62
Figure 9: The BitFlows Chat area.....	62
Figure 10: Two users in BitFlows' user list.....	62
Figure 11: Four nodes from two different machines. Note the use of identicons to show which node is running on which machine.....	63

I. Introduction

Just as painters manipulate and position paints on a canvas, and poets play with the structure and arrangement of words, New Media creators work in the manipulation of digital bits. Bits however have no tangible presence, a New Media creator cannot generally delve into the internal workings of a computer and physically rearrange individual bits to generate a new work. The bits on their own have no inherent meanings behind them, rather creators must rely upon generally accepted standards for interpreting the meaning of the bits, and turning them into words, sounds, images or video. Individuals working in digital media generally need to rely upon specially created pieces of software and hardware to create and manipulate these patterns. There is no lack of software or hardware available to help creators manipulate and position bits in just such a way as to produce precisely the sort of product which they desire. A talented user of Adobe's Photoshop image editing software can produce any image they can imagine. Now more than ever, New Media producers have access to a diverse range of tools for developing New Media artworks. With the explosion of the Internet, every computer made today now ships with networking capabilities. The popularity of video games has created a market for new and innovative control devices. Input devices formerly reserved for high-end professional and research work such as 3D mice¹, data-gloves², and camera tracking

1 Such as the 3DConnexion SpaceNavigator (<http://www.3dconnexion.com>)

2 Such as the P5 Data Glove (<http://www.vrealities.com/P5.html>)

systems³ are now all inexpensively available for use by artists. Each of these tools provides creators with a new stream of bits ready to be moulded to their desires, and yet a user trying to use their fancy new joystick in Photoshop will quickly find it impossible. Why? Because the developers of every tool make certain assumptions about how it will be used. The developers of Photoshop (quite reasonably) only expected it to be used with a mouse, a keyboard, and maybe a drawing tablet.

Every tool carries with it certain assumptions and usage biases. Certain tools make certain behaviours easier than others. It's easier to use a mouse in Photoshop than a Joystick. It's easier to edit a sound in Logic than in a text editor. It's easier to perform live music with Ableton Live than with Logic. The effect of these biases on creative practice can be seen in certain trends in the creative output of users of the tools. For example, the introduction of complex user definable brushes in Photoshop has led to certain tendencies in design practice. Similarly the introduction of the drum machine and sampling made certain styles of music possible. Entire musical genres have formed based of the biases of these machines, for example the squelchy melodies of the Roland TB-303 bass-line synthesizer was highly influential in the development of "acid house" music (as well as electronic music as a whole). Of course the biases inherent in these tools do not completely define the creative output of a certain tool. It's not the tools one uses that define the creative output, it's how one uses (or misuses) them. Increasingly within the

³ Such as the ReactIVision software (<http://mtg.upf.es/reactable/?software>)

domain of New Media some of the most innovative and interesting works create or use "hacks", or misuses of existing tools and technologies. The creation of hacks is an effort to break free from the assumptions and biases of a tool. Not every tool is well suited to hacking though, often proprietary systems, technological lock-down, or even just system limitations can get in the way. What then can be done to provide gateways into systems, and pathways between them to make hacking possible and pleasurable and to encourage *unique and innovative* creative developments in the domain of New Media?

This paper documents one exploration into the nature of creativity in a New Media context, and outlines the development of a software tool which attempts to create conditions conducive to the development of uniquely creative New Media works. This paper begins by examining four areas of New Media culture and artistic practice where there is significant innovative activity: the hack, the mash-up, the demo, and the network. The next section will concern itself with exploring prior research on creativity and the various factors that influence creative work. Particular attention will be given to Mihaly Csikszentmihalyi's systems model of creativity, the use of play in creativity and education, and his concept of flow in creative work. Finally, the findings of the previous sections will be synthesized, and various issues impeding the development of uniquely creative New Media works will be outlined. The software project BitFlows will then be presented as a new sort of meta-tool intended to provide an environment for the live creation of hacks and mash-ups over the network.

II. Four Themes in New Media Culture

The field of New Media, as can be inferred from its name, is in constant flux. Technologies are constantly appearing and becoming obsolete. Yet even within this ever-shifting environment there are certain themes that reveal themselves. This section will cover four different, but inter-related themes that have emerged both in New Media art practice, and in our modern networked culture: the hack, the mash-up, the demo, and the network. While these themes are in no way exclusive to New Media Art practice, they represent sites of significant creativity. This is also not meant to be a comprehensive account of New Media culture, but rather an examination of elements that can be taken advantage of in the effort of inspiring creativity in New Media art practice. Each of these themes are deserving of a book of their own to do them justice. For the purposes of this paper however, I will give a brief overview of the subject, and it's uses in New Media work.

II.a. The Hack

For me, the true standout in the world of jazz is Thelonious Monk. The first time I heard Monk's music, I thought, What the hell is going on? This sounds awful. Notes in places they shouldn't be, timing structures all over the place - all twisted and distorted like I'd never heard. But then, isn't that what I try to do with visual images? Isn't that what a lot of digital artists do - bend and shape bits of code, bits of data, into new forms, in the hope that something exciting will come of it? And the more I forced myself to listen to Monk and those like him, the more it struck me that they're all essentially hackers. But instead of hacking with computers to change code, they hacked with instruments to twist and reshape musical notes. (Dawes, 2007, p. 144)

Our culture is full of hackers. Not only in the domain of computers or technology, but in every domain that requires innovation and creative thought. Eric Raymond (one of the original computer hackers) defines the hacker, as "one who enjoys the intellectual challenge of creatively overcoming or circumventing limitations." (Raymond, 1996, pp. 233-234). This definition is consciously very general. A hacker doesn't need to be involved in computers or technology at all, and there are hackers in every domain: math, physics, even graffiti art. Regardless of their domain, hackers are all creators. They develop new software and hardware, art and techniques. In short they develop hacks. The term "hack" as we are using it here has a dual meaning. On the one hand it refers to the clever manipulation of hardware or software to give it new capabilities, or use it in ways not originally intended by its original creators. On the other hand it simply refers to a quick, cheap, or clever way of doing something. Take for instance the Jeff Han's

"Frustrated Total Internal Reflection" (FTIR) multi-touch interface solution (Han, 2005). Using a consumer-grade webcam slightly modified to be receptive to infra-red light, a sheet of acrylic, a video projector, and a set of infra-red LEDs, he has developed a simple, elegant and inexpensive way to create a multi-touch sensitive display. Historically these sorts of multi-touch interfaces used capacitive touch sensors, and would have been near impossible for individuals to build themselves and prohibitively expensive to purchase. Han's invention can be considered to fall under both categories of hack, the modification of the webcam repurposes it as a positional sensor, a use likely not imagined by its original creators. At the same time it uses a clever technique to make multi-touch technology available to nearly anyone with the desire to experiment with it. It's a very "neat hack".

The use of the hack in New Media art often becomes somewhat of a necessity. As Simon Penny (1995) has pointed out: "Unless artists are in direct contact with research labs, their access to 'science' is via commodities...". A painter can mix their own paints, and a sculptor can find their own rocks, but a New Media artist can't make their own microprocessor or hard-drive. Even faced with these constraints, artists and creators have been able to hack and misuse commodity hardware and software to work in new and unexpected ways; in the words of Jon Ippolito (2002): "What sets art apart from other technological endeavours is not the innovative use of technology, but a creative misuse of it." The entire musical genre of "glitch" and "microsound" is built on the the creative

misuse of technology, and the aesthetics of failure (Thomson, 2004; Cascone, 2000).

The Apple iPhone, Sony PSP, and Nintendo DS have all been hacked to allow users to develop and use their own custom software. On each of these pieces of "locked-down" consumer hardware, independent developers have found means to unlock them for their own use and develop software for creative expression. Mrmr (Integrated Digital Media Institute of Polytechnic University, Brooklyn, 2007) and the i3L MIDI bridge (Thille, 2007) allow artists to control music and visuals from an iPhone, making it a relatively inexpensive alternative to multi-touch controllers like the Jazz Mutant Lemur (Jazz Mutant). PlayLive provides similar capabilities to the PSP (King, 2006). NitroTracker (Weyland, 2008) and PSP Rhythm (B. Iturzaeta & L. Iturzaeta, 2007) allow Nintendo DS and PSP users to sequence music on their devices. Even commercial software packages like Ableton Live have been hacked to unlock a hidden internal API (King & Ramella, 2007), and computer game "mod" communities hack new games from the foundations of other games (Galloway, 2006; Nieborg, 2005; Postigo, 2003).

One device that has recently become the subject of numerous creative hacks is the Nintendo Wii Remote video game controller. The Wii Remote is a fascinating device for several reasons: For one, it packs a remarkable amount of technology in an inexpensive, small package (currently about \$45). With the Wii Remote one gets a three axis accelerometer, a infrared digital camera, a vibration motor, eleven buttons, four LEDs, and a Bluetooth wireless radio. Within the gaming community the controller has been lauded

for the innovative game control schemes that it has inspired. More interesting though is the community that has emerged around hacking the Wii Remote for use with normal computers. Because the Wii Remote uses a standard Bluetooth signal to communicate with the Wii console, developers have been able to hijack the signal for use on normal computers. Because of this a large community has emerged around the development of alternate uses for the controller. It has been adapted for use as a virtual drum-kit (Merz, 2007), a robotics controller (Rasmussen, 2007), a musical looping device (Seznec, 2008), a multi-touch controller and even a 3D head-tracker (Lee, 2007).

Hacker culture has always had close ties to open-source ideologies, most hardware hacks are available freely or have been documented so as to allow anyone to perform the same hack. Open-source ideals and methodologies have also permeated certain aspects of hardware development. The people behind the Monome musical controller have released all the software, schematics, and build instructions necessary for individuals to build and hack their own Monome⁴ (Crabtree & Cain, 2007). This openness has allowed a community to form around building, using and modifying Monome devices. Further following open-source methodologies popular hacks of the Monome, such as incorporating an accelerometer, have been incorporated into the core design of new iterations of the device. Similarly, the Arduino is an open-source micro-controller hardware

⁴ As well as allowing people to build their own Monome devices, <http://monome.org> sells pre-built devices, the most recent version of which (the sixty four) sold out in 2 minutes of online availability (as of January 2008).

and software system that serves as a platform for individuals to create or control their own hardware devices (Banzi, Cuartielles, Igoe, Martino, & Zambetti).

These sorts of open architectures encourage users to break out of their traditional roles as simply consumers, and become makers. Much has been written about collaborations between artists and engineers, though most of these collaborations focus on the artist developing a concept which is then implemented by the engineer (Mamykina, Candy, & Edmonds, 2002; Nakakoji, Yamamoto, & Ohira, 2000; S. Wilson, 2002; Y. Zhang & Candy, 2007). This divide between the arts and sciences is reminiscent of C.P. Snow's "two cultures" (1998). Ehn (1998) has proposed a "Manifesto for a Digital Bauhaus", where art and technology are united within a single practice. The capability for artistic use and misuse that hacking technology provides takes great strides towards this merger of art and technology. Numerous explorations have been made into alternative musical interfaces (Bongers, 1999; Collins, 2003; Wanderley & Battier, 2000), and physical computing interfaces (Fleck, 2003; Ishii & Ullmer, 1997; Villar, Lindsay, & Gellersen, 2005a, 2005b; G. Weinberg, 2002). Hacks such as the ones outlined above, put these sorts of interaction possibilities within the reach of most artists. It seems apparent that there is a new culture of hacking and DIY is emerging thanks to open-source hardware and software platforms and an online culture of knowledge sharing . Every new piece of software or hardware presents a new challenge to the hackers of our culture, a new source of bits to use and a new opportunity for creativity. Individuals are working not only to explore the

creative potential inherent within a product, but to find new and creative ways to expand and misuse it.

II.b. The Mash-Up

In 2001 musical producer The Freelance Hellraiser released a track titled A Stroke of Genius that mixed the vocals of a Christina Aguilera song with the instrumental track of a Strokes song. The blending of Aguilera's mainstream pop vocals with the independent rock guitar of the Strokes grew to be hit in online music communities and eventually even in the general public, being hailed as one of the best songs of 2001. In the music world this track spawned a new production style and music genre known as the "mash-up", a style in which producers take different elements of diverse songs, and combine them to form a new whole artwork (Ferguson, 2004; McLeod, 2005). The mash-up technique quickly spread throughout the music world and began to manifest itself in fields that had nothing to do with music at all.

At the same time that music producers were building stores of a cappella vocals and instrumental backing tracks, computer scientists and programmers were developing evolutions of the Web. As more and more websites began making their data available in standardized data formats such as XML and allowing the general public to interface with their systems using Application Programmer Interfaces (API's), users began experimenting with creating web mash-ups. In these mash-ups, similar to the musical mash-ups, the creators take the data from one website and incorporate it into another. For example, a number of web mash-ups have been made that take data from the online classified ad site

craigslist.com and bring it into Google Maps to show the geographic locations of the ads.

Google⁵, Yahoo!⁶ and even Microsoft⁷ have gone as far as to develop web-based tools which give users a simple interface to build their own mash-ups from a variety of data-sources. These mash-up editors effectively act as routers to direct, modify, and reinterpret flows of data in the Web. The web is fundamentally built in such a way as to allow the mashing up of data. The WWW is probably one of the most successful technologies of our time, and that success can in no small part be attributed to its ability to infinitely link to, combine, and recombine its component parts. In reference to the ability for recombination on the Internet, Manuel Castells has written:

Recombination is the source of innovation ... the ability to experiment with this recombining from a multiplicity of sources considerably extends the realm of knowledge, as well as the connections that can be made between different fields. } (Himanen & Torvalds, 2001, p. 163)

Essentially, every website that incorporates images hosted on another website is making a mash-up. Though it would be a stretch to call most websites innovative on their own, the ability to freely play with the interconnections between data on the Web has been the source of much innovation and creativity.

Weinberger (2002) has appropriately described the web as "small pieces loosely joined". This principle can be applied to the mash-up technique as a whole. Nearly anything can be seen as the potential subject of a mash-up. Lev Manovich (2002) defines

5 <http://editor.google mashups.com>

6 <http://pipes.yahoo.com>

7 <http://www.popfly.ms/>

five basic principles of New Media: numerical representation, modularity, automation, variability and transcoding. In New Media, every file, data-stream, input and output can be distilled down to a single basic language of ones and zeroes. This data can be interpreted in a variety of ways depending on how it is processed. These principles make New Media the perfect environment for the mash-up. New Media artist and designer Brendan Dawes (2007) finds his inspiration from the world around him, he is able to find sources of data in nearly any situation he finds himself in, whether from a ball of Playdoh, or the crowds of an airport. It is just this approach to seeing data-sources as small, modular, interchangeable and everywhere that characterizes the mash-up process. Given some creative experimentation, nearly any collection of hardware and software can produce an experience far greater than the sum of its individual parts. As a creative technique, the mash-up is a powerful way to explore and unearth unexpected possibilities. The use of mashing up as a source of innovation will be further covered in section III, but for now it is sufficient to show the pervasiveness of the mash-up.

II.c. The Demo

Nicholas Negroponte, founder of the MIT Media Lab was once famously quoted as saying that "At the MIT Media Laboratory ... the academic slogan 'publish or perish' has been re-codified as 'demo or die'" (Lunenfeld, 2000, p. 13). Short for demonstration, the demo as used here refers to the performative exhibition of technology. Whether it is a film-maker showing their demo reel, a research lab demoing a new technology, a programmer showing off their skills at a demoscene competition, or even a time-limited demo version of some software, the demo has become an important method for dissemination of creative works. In fact, Lunenfeld has argued that the "demonstration or "demo" has become the defining moment of the digital artist's practice at the turn of the millennium."(2000, p. 13) The demo serves two roles in New Media art production. Firstly, the demo provides an opportunity for the dissemination of what would otherwise be transient works. It is difficult to convey and disseminate the full impact of moving, interactive, site-specific, or hardware-specific works in a textual form. Yet, as will be discussed in section III, creative practice is dependant on dissemination to the relevant field. As such, the demo is indispensable to New Media culture. Undoubtedly there are innumerable hacks which have been developed by individuals in their own private work, but without disseminating their creations these hacks have gone unnoticed.

Secondly the demo is a performance of a New Media work. Digital artists have a

wide range of tools at their disposal. Photoshop, Logic, Final Cut Pro, Illustrator, After Effects, and other such software tools are common to find in most digital artist's toolboxes. All of these programs, while being more than suitable for creative work, are generally only for the creation of static works. That is, the artist uses them to develop a fully formed final product which may then be distributed. It is rare for them to be used in a performative setting as they are designed for the slow, methodical production of highly polished works. These tools are not built for the real-time dynamic shifts and adaptation typically required in a performance or demo setting. But, as the demo has become an important part of New Media practice, an entire genre of tools has emerged specifically for developing, and performing New Media art works. Back as early as 1977, artists and programmers were beginning to see the value in "interactive program environments" for artistic production (Truax & Barenholtz, 1977). Audio software such as Ableton Live and Seq 24 pride themselves on their performative capabilities, and VVVV, Quartz Composer, Jitter, Resolume and several other pieces of software give artists the capability to perform and generate video on the fly.

In the mid-1980s, spawning from the practice of developing graphical intros to cracked software, groups of hackers began applying their skills to stretching the graphical capabilities of the computers of the time. Known collectively as the "demoscene", the compositions (usually termed "compos") created by these individuals and groups used procedural and generative techniques, undocumented assembly language calls, and self

modifying code to create real-time video and audio works, often programmed within a space constraint (of about 8 to 64 KB), or a time constraint (often 24 hour marathon programming sessions) (Kuittinen, 2004; Raymond, 1996, p. 150; Green, 1995). Although most compos were not interactive and theoretically much more easily rendered frame by frame and recorded to video, an important aspect to demoscene programs is that they are run in real-time on a computer, showing off both what the hardware and programmer are capable of. More importantly though, this emphasis on real-time execution gives the compo a performative aspect that wouldn't be as present in a static video. A logical extension of demoscene practice has emerged more recently with "live coding". In live coding, artists perform music and visuals by programming the computer in real-time, often in front of an audience (Blackwell & Collins, 2005; Brown, 2006; Collins, 2003; Collins, McLean, Rohrhuber, & Ward, 2004; G. Cook & Misra; Kapur, G. Wang, Davidson, & P. R. Cook, 2005; Nilson, 2007; G. Wang, Misra, & P. R. Cook, 2006; G. Wang, Misra, Davidson, & P. R. Cook, 2005; G. Wang & P. R. Cook, 2004). The CHUCK programming language was developed specifically with live musical programming in mind , and it's Audicle interface allows the live coding of visuals (G. Cook & Misra). Max/MSP and Pure Data also offer graphical interfaces to live-coding activities (Puckette, 2002). Live coding is the far extreme of the demo as technological performance. Similar to an improvising jazz musician live coding artists don't simply present a finished work, they perform the entire creation process.

II.d. The Network

In the previous sections I have presented three loosely related themes in New Media culture. The network is the tie that binds all these themes together. The use of the network in artistic practice certainly predates both the Internet, and New Media as we know it. Roy Ascott's explorations into telematic art date back to the early 1980's (Ascott, 2003), and Laszlo Moholy-Nagy produced a series of paintings over the telephone network as early as 1922 (Kaplan, 1993). Early uses of the WWW for art often used the network solely as a distribution system for digitized version of off-line works. Later as the Internet became more accessible and more artists began exploring the creative possibilities of the Internet, the network became the *subject* of numerous artworks (Ippolito, 2002). Artists like Alex Galloway (Galloway, 2006) and early net.art practitioners such as jodi.org (S. Wilson, 2002) built works using hacks of the formats and protocols that form the Internet. The network also serves as an invaluable resource for the sharing of knowledge. Han's release of the plans for his FTIR multi-touch display on his website (Han, 2008) has spawned a sizeable online community of individuals developing their own FTIR surfaces (<http://nuigroup.com/>). Similarly, the Wii Linux forums and wiki at willi.org have been incubators for the development of Wii Remote hacks. The new rise of a DIY culture can at least in part be attributed to the network, due to the increased ability to disseminate knowledge and hacks, and collaborate with like minded individuals over long distances.

Although network art often focuses on manipulating general purpose protocols and standards such as e-mail and the WWW for artistic purposes, several protocols have also emerged specifically for using networks for artistic expression. In the early years of electronic musical synthesis, synthesizers were developed as modular pieces of equipment. A typical synthesizer would consist of an oscillator which would produce a sound wave, this sound wave could then be passed through a number of different modules such as filters and envelope shapers to create a wide variety of different sorts of sounds. Each of these modules needed a standard way to pass signals from one module to another so that users or other modules (such as keyboards or Low-Frequency Oscillators (LFOs)) could control the parameters of each module. The most common protocol (and I use protocol here in a very loose sense) used at the time was called Control Voltage (CV). The CV protocol simply consisted of a variable voltage sent across a cable which could be interpreted by synthesizer modules in a variety of ways to correspond to different values. While CV was effective for the modular synthesizers of the time, as electronic musical instruments increased in complexity and polyphony, and as musicians desired to connect more and more diverse instruments together the limitations of CV became evident.

In 1982/3 a number of musical instrument manufacturers came together to develop a standard protocol for communication between different musical instruments called Musical Instrument Digital Interface or MIDI (Loy, 1985). MIDI consists of a set of

standardized digital messages such as "Note On", "Note Off", "Control Change", and "Clock", which are interpreted in more-or less the same way by any instrument. Since it's creation, the MIDI protocol has become the standard for musical communication. Nearly every modern synthesizer and musical software package has MIDI support. In fact, use of MIDI has extended beyond the musical domain and been adopted as a method of controlling stage lighting, live video software, and interfacing with hardware sensors. Although an improvement over CV, MIDI still has a number of limitations (Moore, 1988). MIDI was not originally intended for use outside of a musical domain, and so non-musical applications need to map musical semantics and assumptions to their own domain. So, for example in a video mixing application, note-on messages might trigger various videos, and pitch-bend messages might mix between them. The semantics of the protocol are lost. MIDI was also designed to use it's own specifically designed low-speed serial hardware layer. All MIDI communication is half-duplex, meaning a MIDI cable can only carry messages in one direction. Because of this, networks of MIDI devices can tend to get rather complex very quickly; bi-directionally networking six-performers would require at least 30 MIDI cables. MIDI was also only designed to be a local point-to-point protocol, with no support for control over networks like the Internet. Finally, most MIDI messages also only have a resolution of 7-bits, meaning expression is limited to 128 steps of resolution.

More recently a new protocol called Open Sound Control (OSC) has been

developed to remedy some of the issues plaguing MIDI (Wright & Freed, 1997). The OSC protocol uses standard ethernet UDP packets as its transport layer (though it is adaptable to other transport layers as well). Instead of a pre-defined semantic structure like MIDI, OSC uses a hierarchical user-defined file-system-like structure. OSC also has a flexible payload structure, so a variety of different high-resolution data-types can be contained in a single OSC packet. OSC has become a favourite among hardware and software hackers, as it is a relatively simple and flexible protocol that doesn't require special hardware and has user-definable semantics. Hacks of the Wii Remote all use the OSC protocol to communicate data between the controller and software, the open-source Monome controller communicates exclusively over OSC, and the Princeton Laptop Orchestra has used OSC to experiment with distributed, network controlled music (Fiebrink, G. Wang, & P. R. Cook, 2007).

MIDI and OSC have both been used as protocols for networked creative collaboration, but are far from the only solutions. Uses of the network for creative collaboration can roughly be categorized along temporal (realtime or non-realtime) and spatial (local or remote) axes (Barbosa, 2003; D. Williams & P. Webster, 1999; Hickey, 1998; G. Weinberg, 2005b). Most non-realtime remote collaboration efforts tend to be organizations of pre-existing communications technologies to trade content (Netjam (Latta, 1991) or deposit content in a central store (Faust Music On-Line (Jorda, 1999), MICNet! (Hickey, 1998), CC-Remix (Tanaka, Tokui, & Momeni, 2005)). Yamagishi's "Variations for

WWW" (1998) and Young and Packer's Telemusic pieces (Young, 2002) use a web interfaces to control remotely hosted Max/MSP patches and stream the musical results of every user's manipulations back to the user. Realtime local collaborations have tended to use MIDI, OSC, or serial data to communicate between participants. Many experiments have also been done in locally networked music performance (Bischoff, Gold, & Horton, 1978; Gresham-Lancaster, 1998; Gurevich, 2006; G. Weinberg, 2005a). In realtime remote collaborations the nature of the data being transferred becomes much more important, control data is much easier to transmit in real time than actual audio or video data, thus most realtime remote collaborative efforts are made possible by transmitting MIDI or other control data over the Internet with either the server or client producing the actual sounds (Biaz, Chapman, & J. Williams, 2005; Burk, 2000; Gang, Chockler, Anker, Kremer, & Winkler; Lazzaro & Wawrynek, 2001). Co-Audicle, a collaborative networked interface to the CHUCK live coding language uses a single audio server which allows multiple clients to connect and run code on the server instead of transmitting audio data (G. Wang et al., 2005; G. Wang et al., 2006). The issues of bandwidth and latency have somewhat limited the possibilities of realtime collaboration using audio or video data (Bartlette, Headlam, Bocko, & Velikic, 2006; Gu, Dick, Kurtisi, Noyer, & Wolf, 2005; Gu, Dick, Noyer, & Wolf; Gurevich, Chafe, Leslie, & Tyan, 2004). Some projects such as Ninjam aim to avoid the problem of latency by increasing the delay in transmission to a musical quantity (one bar) (Bouillot, 2007; Underwood, 2007). Other projects avoid the

bandwidth issue by using high-speed research networks (Chafe, S. Wilson, Leistikow, Chisholm, & Scavone, 2000; Ox, 2002). The GIGAPOPR project used a high bandwidth connection between Princeton and McGill to allow for low latency streaming of audio, video, and MIDI data (Kapur et al., 2005). Carot, Kramer and Schuller (2006) developed low-latency streaming software to allow audio data to be sent with low latency over a narrow-band network.

As has been shown, the network plays numerous roles in artistic practice. It works as a distribution system, enabling the distribution of inspirational works and demos of new hacks and techniques. It serves as a subject for net.art practitioners. It serves as a means of communication between different musical instruments and hardware devices. It serves as a collaborative tool. Weinberger has described the web as a set of "small pieces loosely joined" (Weinberger, 2002). The numerous hacks disseminated over the network as well as the network itself provide a huge diversity of sources of bits. At the same time the network serves as a transport mechanism to join and mash-up the "small parts" that these hacks provide. The network is the tool for "loosely joining" all these pieces into a greater whole, but using the network to join these pieces is far from a trivial task in most cases. Each device speaks its own dialect, has it's own distinct way of physical interconnection and makes its data available in a variety of formats. With the demo becoming a crucial moment in digital art practice, a degree of spontaneity and performability is desirable. Herein lies the conflict I seek to address: how can the

complexity of the network and the mash-up be reconciled with the desire for spontaneity?

In Dawes quote at the beginning of the section he describes jazz musicians as hackers.

Jazz music has a long tradition of improvisation and spontaneity in performance. How can

we bring a similar capability for spontaneity to network and mash-up creative play?

III. Creativity, Flow and Play

In order to develop a tool to support creative activity, it is first necessary to provide a working definition of creativity and an examination must be made into the nature of creative behaviour and the factors which influence creative production. Creativity has been a subject of significant psychological research, and several models of creativity have been developed. Shneiderman (2000a) has divided the research on creativity into three main models: inspirationalist, structuralist, and situationalist. The inspirationalist model of creativity focuses on the so-called "Aha!" or "Eureka!" moment. Long periods of preparation followed by a sudden inspiration, then followed by much hard work to put the inspiration into practice. Structuralists instead emphasize a methodical approach to innovation, and use systematic techniques for problem solving. Situationalists emphasize the effect of social, physical, and intellectual surroundings on creativity.

Mihaly Csikszentmihalyi suggests that the situationalist model of creativity is a useful one to adopt in efforts to facilitate creativity because: "It is easier to enhance creativity by changing conditions in the environment than by trying to make people think more creatively"(1996, p. 1). Csikszentmihalyi suggests a "systems model" of creativity, in which "...creativity does not happen inside people's heads, but in the interaction between a person's thought and a socio-cultural context. It is a systemic rather than an individual phenomenon" (1996, p. 23). Csikszentmihalyi specifies three main components to the

systems model of creativity: the domain, the individual, and the field. The domain as Csikszentmihalyi defines it is a body of knowledge that is comprised of a set of memes or cultural genes (a concept from Dawkins' *Selfish Gene*) and symbolic rules. The individual works within the domain (be it music, math, physics, etc.) and makes contributions to the domain by creating novel memes. Finally the field consists of the other individuals working within the domain who evaluate the contributions of the individual and determine whether their work is worthy of inclusion into the domain. Creativity comes from the individual changing or adding to the set of memes that define a domain.

In many ways this model of creativity is very similar to the peer-review structure of academia. It is important to note though that despite their similarities there is a subtle but significant difference between them. For something to be deemed as a contribution to the domain, and thus a creative work, it is not necessary for it to be formally published, merely for it to be disseminated and accepted by other members of the field. Applied to New Media, the effect that the Internet has had on both its domain and field cannot be understated. Since Lunenfeld wrote on the subject of the demo, advances in video-streaming technology have eliminated the requirement for a demo to be done in person. It is now simple to make a video of a demonstration and distribute it using online video sites like YouTube. Jeff Han's FTIR interface was demoed at the TED conference in 2006 and has been published by the ACM (Han, 2005), but most people were first exposed to it through either his website (Han, 2008) or a video recording of his TED demo placed online (TED

Conferences, LLC, 2006). The Internet serves a threefold purpose: it serves as an educational tool for those interested in learning a domain (familiarization with existing memes). It provides an open platform for the dissemination of an individual's work (potential memes). Finally, it provides an informal and distributed structure for the evaluation of memes. Blogs like BoingBoing.net, aggregation sites like digg.com and social bookmarking sites like del.icio.us have become a sort of massively distributed peer-review and dissemination system for memes. Here is where the differences between the peer review structure of academia and the systems model of creativity become apparent. The peer-review structure is designed to maintain high-standards, and prevent the inclusion of false claims in a body of knowledge. The systems-model of creativity however is concerned more with the creation and modification of memes.

As has already been mentioned, in order to be creative within a domain, first one must become immersed within the domain and learn its rules. But how can the learning of a domain (in this case New Media) be facilitated in such a way as to promote creative exploration and innovation? Developmental psychologist, Jean Piaget had this to say regarding his approach to education:

Education, for most people, means trying to lead the child to resemble the typical adult of his society . . . but for me and no one else, education means making creators. . . . You have to make inventors, innovators—not conformists (Bringuer, 1980, p. 132)

Much of his pedagogical theory was based in principal around the concept that "To know an object is to act upon it and to transform it" (Bringuer, 1980). Similarly, in his works on

education John Dewey developed the pedagogical approach of 'learning by doing' (Dewey, 1916). Both Piaget and Dewey suggest that learning is often best achieved through either structured, or unstructured play and exploration (Roussou, 2004). This pedagogical theory certainly applies to the learning of technologies as well. Playfulness has been shown to assist in the learning of new technologies (J. J. Martocchio & J. Webster, 1992; Monk, 2002; Roussou, 2004). Turkle suggests the following regarding the ways in which technology is learned:

In the emerging culture of simulation, the computer is less like a hammer and more like a harpsichord. You don't learn to play a harpsichord primarily by learning a set of rules, just as you don't learn about a simulated micro-world...by delving into an instruction manual. In general, you learn by playful exploration. (Turkle, 1995, p. 61)

Play serves not only as a pedagogical tool, but also as a facilitator for innovation in art, science and technology. In her study of the nature of playfulness, Lieberman examines the relation between creativity and playfulness, finding that artists (who she argues are undeniably creative) exhibit playful tendencies (Lieberman, 1977). Do and Gross (2007) have found that playful approaches to technology encourage the making and hacking of things. Several studies have examined factors influencing individual's playful interactions with computers (T. P. Novak, D. L. Hoffman, & Yiu-Fai Yung, 2000; J. Webster & J. J. Martocchio, 1992; Yager, Kappelman, Maples, & Prybutok, 1997), the WWW (Agarwal & Prasad, 1998; Chen, Wigand, & Nilan, 2000), and virtual reality (Reid, 2004; Roussou, 2004). Hackers are well known for their playful behaviour. Eric Raymond has said that "To do the UNIX philosophy right ... you need to play. You need to be willing to

explore" (2004, p. 27) . Tim Berners-Lee laid the foundations of the WWW by linking together a series of "play-programs" (Berners-Lee, 1999, pp. 9-13). Combinatorial play allows for the free form interplay between a range of different pre-existing concepts and ideas, a sort of mash-up of ideas. Even Albert Einstein, when describing his process of innovation termed it "combinatorial play" (Schilpp, 1970). Similarly, Koestler describes creativity as the process of connecting multiple previously unrelated "matrices of thought" to produce a new insight or invention (Koestler, 1964). The use of play as a tool for both education and innovation can be seen in the design for several programming languages and environments designed both for educational and creative uses. The Scratch programming environment, a language created to teach children programming techniques, was designed for children to play with at after-school computer centres rather than within a formal educational environment (J. Maloney et al., 2004). Scratch is built on top of Squeak, an implementation of Smalltalk-80 (Ingalls, Kaehler, J. Maloney, Wallace, & Kay, 1997). Both Scratch and Squeak use a non-traditional programming interface that allows users to get immediate graphical feedback from their program, and generate programs by combining and playing with programming constructs similar to building with LEGO blocks. Processing, a version of Java designed specifically for artistic exploration, comes with over 250 small programs covering topics ranging from flocking behaviour to fluid dynamics. All of these programs code is available to be creatively modified and recombined to let the Processing beginner jump right in and begin playing with interesting technologies (Fry &

Reas).

Similar approaches have been taken in live performance coding environments. Within live performance environments, it is common to have a selection of pre-made snippets of media such as audio, video or code. In "Don't Forget the Laptop", Cook et. al. (2007) present an argument for the use of the built in functionality of the laptop (keyboard, trackpad, joystick, motion sensing, webcam, microphone) to do collaborative live performances as part of the Princeton Laptop Orchestra. They make available a set of sample programs using each control method for artists to build upon. They argue that:

A critical mass of ubiquitous, easy-to-use code can encourage willing experimenters to make more music together with their laptops, while continuing to ponder and refine the use of laptop inputs in their music-making.

(Fiebrink et al., 2007)

Having these code snippets available and ready for use allow participants to create mash-ups of different data streams in a collaborative live performance setting. There is only so much that a teacher, a book, or a manual can tell a user about a piece of technology. While these may be excellent starting points to help users understand the basics of a technology, it is only by "playing around with" a technology that one can begin to truly explore what is possible with a technology.

The role of play in creativity and innovation is closely linked with a phenomenon that Csikszentmihalyi calls "flow", or "optimal experience". Flow is a state which individuals experience when they are intensely focused on an activity and the individual feels

completely in control of their situation (Csikszentmihalyi, 1990, p. 6). Flow is experienced by individuals working in all domains; rock-climbers, chess champions, musicians and theoretical physicists have all reported experiencing flow (or being "in the zone"). Csikszentmihalyi identifies nine elements to flow experiences: clear goals, immediate feedback, balance between challenge and skills, a merger of action and awareness, the exclusion of distractions, no concern of failure, the disappearance of self-consciousness, a sense of time distortion, and an autotelic aspect to the activity (it becomes worth doing for its own sake)(Csikszentmihalyi, 1990, 1996, pp. 48-67). Playfulness is particularly important in making an activity autotelic (Malone, 1981). Numerous studies have been done exploring how the principles of flow can be applied to human computer interaction and software design (Bederson, 2004; Chen et al., 2000; Farooq, 2005; Finneran & P. Zhang, 2002; T. P. Novak et al., 2000; Shneiderman, 2000b; Trevino & J. Webster, 1992; J. Webster & J. J. Martocchio, 1992). Ghani, Supnick and Rooney (1991) have identified a set of antecedents and consequences of flow. According to their model the challenge/skill relationship, perception of control and spontaneity or playfulness help individuals experience flow. Hoffman and Novak (1996) have studied how flow can be used to create compelling WWW experiences, and have added two more secondary antecedents to flow: interactivity and telepresence. In 2000 they furthered this exploration by examining playfulness in web interactions as a sign of flow (T. P. Novak et al., 2000) . Trevino (1992) has studied instances of flow in voice and e-mail communications, specifically

examining playful and exploratory behaviours. He defines four dimensions of flow: control, attention focus, curiosity, intrinsic interest (pleasure interacting). Similarly, Korzaan (2003) has found that flow is linked with curiosity and exploratory behaviour. Sawyer has examined how flow exists in group and collaborative settings, specifically focusing on improvised jazz and theatre performance (Sawyer, 2000, 2006). He argues that while much effort has been put into studying flow in individuals, little has been put into studying flow in groups. He finds there to be three characteristics of group creativity: improvisation, collaboration and emergence. According to these characteristics creativity happens at the spur of the moment when all members contribute their varying skills and the output is greater than any of the individuals would have been capable of on their own. These characteristics are applied to the design of the Beatbug, a collaborative musical interface for children.

Flow and play serve important roles in the creative process. It is up to the designer of a technology however, to design their system in such a way as to support flow. Many of the tools outlined in section II already have the capability for flow-like experiences, however when attempting to use these tools in a collaborative situation, a networked environment, or using a mash-up technique flow can tend to suffer as users get bogged down with technical issues. The next section will present a tool which attempts to provide a tool for enabling flow in these situations.

IV. BitFlows

With the understanding of creativity from the previous section it is now possible to begin formulating the requirements of a platform to support innovation and creative flow in New Media practice. To do this we will examine the ways in which the systems model of creativity, play, and flow can be applied to the four themes in New Media culture which I have previously outlined. From this a few outstanding issues with current New Media practice become apparent. It is these issues that I will attempt to address in the following section by presenting BitFlows as a software tool for aiding flow, play, and creativity in New Media practice.

IV.a. The Issues

As has been outlined in the previous sections, there are already a number of performative tools that artists use on a regular basis. As can be attested by their popularity, Max/MSP, Resolume, Ableton Live and other performance tools are all excellent platforms for artistic exploration. Likewise, open-source hardware and hacks have provided impressive opportunities for expression. Each of these tools can be the source of significant creativity on their own but they each also have limitations. What can be done when a piece of software doesn't meet one's requirements? One either needs to find a way to extend the capabilities of the software or find a new piece of software. Generally most of these creative tools are designed with a stand-alone mode of operation in mind, and only have limited means of interacting with other hardware or software. To connect an Arduino microcontroller to Ableton Live would require the user to develop a custom piece of software for converting the serial data from the Arduino into MIDI signals accessible by Live, and then mapping those signals to controls within live. Although connecting programs to each other, to software on other computers, or to pieces of hardware which were not anticipated by their developers is usually not impossible, it can often be surprisingly difficult and time consuming. More importantly though, the process of trying to figure out the technology and develop mappings between these elements removes the creator from creative flow. But, as was discussed in the previous section, often the source

of innovation is combinatorial creativity. So how can software/hardware mash-ups be made without sacrificing creative flow?

One of Csikszentmihalyi's observations is that the intersections between different cultures or domains often tend to be fruitful sources of creative developments (Csikszentmihalyi, 1996, p. 9). It follows that individuals who are knowledgeable in multiple domains have a more diverse range of ways in which combinatorial creativity can be applied. He also notes however, that to make a creative contribution to a domain, it is first necessary to have an in depth understanding of that domain, and learning a new domain takes a significant amount of time and energy (Csikszentmihalyi, 1996, pp. 7-8). This is as true in artistic domains as any other. Artists take years to learn their domains. Every artist has their own set of tools and skills which they have invested significant time and effort in learning, and have developed an aesthetic understanding of their particular craft. Collaboration between individuals working in different domains can often bring innovative results. Each participant can take advantage of the collective knowledge of several domains without requiring the degree of investment in learning all of them. At its best, collaboration allows the different skills, knowledge and abilities of each participant to be made available for mashing up, just as audio tracks are in music, or various forms of data are on the Internet.

In the section on the network, several uses of the network for creative collaborations were examined. Much of the research to date on networked artistic

performance and collaboration has required artists to adopt new and unfamiliar tools for the sake of the collaborative act. This is a somewhat backwards approach to collaboration. Instead of allowing collaborators to take advantage of the time and energy each of them has put in to learning their domain, many of these collaborative tools are requiring all of the participants to learn a completely new tool. In their examination of collaborative musical environments, Blaine and Fels (2003) argue that "In a collaborative musical environment, it becomes even more imperative that the technology serves primarily as a catalyst for social interaction, rather than as the focus of the experience." Collaborating with other artists in local or remote locations often creates technical hurdles to be overcome, particularly in heterogeneous computing environments (Correa & Marsic, 2004). The process of discovering and resolving these issues removes the artist from their practice. Take for example the case of "The League of Automatic Music Composers" and "The HUB", two groups of electronic musicians that have been experimenting in realtime networked musical collaboration since the late 1970's (Bischoff et al., 1978). One of the earliest examples of networked musical collaboration, The League of Automatic Music Composers, created custom musical circuits that communicated via the RS232 serial protocol, but found that "...the non-uniform interconnections and the lack of a common, shared protocol between individual players in this ensemble pointed to much-needed refinements." (Gresham-Lancaster, 1998). The refinement of which Gresham-Lancaster speaks of came about in the 1985 HUB concerts in New York city where a "huge technical

effort" eventually allowed three performers to play over phone lines using modems each sending and receiving three variables important to the sound of the work. In later experiments with transmitting musical data over the Internet, Gresham-Lancaster describes the performance of the HUB to have been "...more of a technical exercise than a full-blown concert. ... In this case, the technology was so complex that we were unable to reach a satisfactory point of expressivity." (Gresham-Lancaster, 1998). As seen in the case of The Hub, even local collaboration can be overly complex, with significant time spent arranging protocols for the hardware of the participants to talk to one another.

The demo, or the performative act has been shown as being important for the dissemination or performance of the creative act. Though important in the systems model of creativity, the dissemination of New Media works is already well supported by blogs, link-aggregators, Internet video and other outlets. The exploration of tools for the performative aspect of New Media, however, remains an ongoing concern, and one appropriate to address using Csikszentmihalyi's principles of flow. Performative software is rapidly developing but as of yet there is only limited support of developing mash-ups between different pieces of software and hardware hacks. Likewise, it is currently difficult to incorporate networked collaboration into live performative and spontaneous creative settings. As mentioned earlier, live coding represents one far-extreme of New Media performance, but also allows for unique performative capabilities. Nachmanovich suggests that all art is improvisation at some point, some presented "...whole and at once;

others are 'doctored improvisations' that have been revised and restructured over a period of time before the public gets to enjoy the work." (1990, p. 6) Sawyer (2000) presents a similar argument, that the process of artistic creation is always a form of improvisation. In this case, an environment designed for spontaneity and improvisation would serve as a good platform for general artistic exploration whether for live-performance or not. How then can flow principles be used to bring hacks, mash-up techniques and networking capabilities to areas of New Media performance and live coding situations?

IV.b. Introducing BitFlows

BitFlows is an open-source software platform for facilitating flow, play and spontaneous experimentation with technology in both realtime networked collaborative and solo settings. In other words, it's a tool for hacking together hardware and software mash-ups in a live setting over the network. Probably the best way to introduce BitFlows is to use the metaphor of the equipment of an electric guitarist. An electric guitarist typically has three main elements in their set-up, their guitar, an amplifier, and any number of effects pedals and boxes between the two. Playing the guitar creates an electrical signal which is passed through a cable to the effect pedals. Each pedal then performs a manipulation on this signal and passes the modified signal on to the next pedal in line. Finally the last pedal passes the signal on to the amplifier which then amplifies the signal and converts it to sound though its speaker. In this system each part in the chain is a modular component. The guitar doesn't know or care about what it is connected to, its only job is to create a signal and pass it on. Likewise, each effect pedal is interchangeable with others, it doesn't matter what they are connected to, they just take a signal, modify it, and pass it on regardless of what comes next in the chain. If the musician decides they need some distortion, they can just add the pedal to the chain. If another musician comes along to play, they can plug into the same system and just start playing along.

This is roughly what BitFlows does for New Media artists, except instead of audio

signals, BitFlows offers a modular system for routing and modifying control data. To extend our metaphor, imagine if the guitar instead of sending an audio signal sent a signal saying what note the guitarist is currently playing. One can then imagine a different sort of effects pedal that would modify this signal. Instead of applying a distortion or flange effect to the audio signal, a pedal could transpose the note value so a C would become an E, or change its volume so a quiet note would become loud. At the far end of the chain, in place of the amplifier one would have a device which converts the note value into an actual sound played at the appropriate pitch. This sort of signal or data routing has been used in computers for decades. In fact part of the power of the UNIX operating system (and its descendants like Linux, and Mac OSX) comes from the ability to "pipe" the outputs of command line programs into the inputs of other commands. This is similar to J. P. Morrison's "flow based" model of programming, developed in the early 1970's at IBM. In his 1994 book on the subject Morrison describes flow-based programming: "An application can ... be expressed as a network of simple programs, with data travelling between them" (Morrison, 1994, p. 25). Flow based programming has a long history in artistic practice, as it is a natural way to deal with the constant data stream of audio and video data. Max/MSP and Pure Data have been using flow based programming since the 1980's to provide artists with a means to create and modify audio (and later, video) (Puckette, 2002).

Although BitFlows uses flow-based programming principles, it is *not* intended to be

a general programming language, or in fact a programming language at all. Rather, the scope of BitFlows has been intentionally limited to being a sort of data router. By limiting the scope of BitFlows, the need for an artist to learn a new programming environment is eliminated, and rather the artist is encouraged to further develop their knowledge of their preferred artistic tools. Programs like Max/MSP, Ableton Live, and VVV are popular because they are good at what they do. Rather than try and recreate the functionality already available in numerous programs, BitFlows follows Shneiderman's suggestion that creativity can be aided by smoother flow *between* applications (Shneiderman, 2000a). Using flow-based programming techniques, BitFlows allow users to route flows of data to and from a variety of creative software packages and hardware devices.

BitFlows is designed so that nearly any piece of software or hardware can potentially be used in BitFlows, so long as it has some sort of user-accessible inputs or outputs. Of course it is impossible to predict new developments in technology and the eventual requirements of all users. In fact, Edmonds et al. (2005) suggest that the demands of creative work often expose the limitations of technologies. To address this issue, BitFlows adopts what Fischer (2004) terms as "meta-design" a mode of design where facilities for "end user development" is built in to a system. A similar solution has been proposed by Von Hippel (2001), and Jeppesen (Jeppesen, 2001, 2002, 2005) who propose the inclusion of "user toolkits for innovation". BitFlows has been implemented to use a modular architecture which allow users to easily customize and create new modules

for BitFlows. This architecture manifests itself in BitFlows in a few different ways. In BitFlows, every piece of hardware or software on a computer is considered a module consisting of inputs and outputs. BitFlows modules are akin to what Morrison terms a "component" in flow-based programming, and the inputs and outputs of a module are like Morrison's ports. Each module is represented by a simple plug-in script that tells BitFlows what sort of inputs and outputs a device has, and how to interact with them. Using a plug-in architecture effectively separates the core logic of BitFlows from that of the devices it connects to. The plug-ins are how BitFlows talks to the rest of the world, internally hardware and software are treated in precisely the same way. This makes it easy for artists and developers interested in adding new software or hardware modules to BitFlows to quickly create their module without requiring them to have any significant knowledge of how the core of BitFlows operates. Each plug-in simply consists of one or more small Python scripts in a plug-in directory. Naturally, a number of essential (and otherwise interesting) plug-ins come by default with BitFlows, including interfaces to OSC, MIDI, and HID⁸ devices.

A significant amount of effort has gone in to making the BitFlows plug-in system as user-accessible as possible. It makes extensive use of Python's decorator functionality to make it possible for plug-in developers to convert pre-existing code into a BitFlows plug-in by simply annotating their function calls. As a simple hello world example, take the code:

⁸ Human Interface Devices. Devices such as mice, keyboards and joysticks.

```
class NotAPlugin:  
    def helloWorld(self):  
        return "Hello world"
```

To convert this simple class into a BitFlows plug-in only requires three changes: the importing of the BitFlows plug-in library, converting the class into a BitFlowsPlugin subclass, and annotating the method to tell BitFlows that you want to use it as an output.

```
import plugin  
class APlugin(plugin.BitFlowsPlugin):  
    @plugin.output  
    def helloWorld(self):  
        return "Hello world"
```

BitFlows is released under an open-source license so as to allow and encourage end-user development and user contributions. In this way artists and programmers can learn from and modify the code to adapt to their needs, and allow for what Fischer (2004) terms as "social creativity". Modifications can also be incorporated into the main distribution of BitFlows to allow other users to benefit from the modifications of everyone in the community.

Internally BitFlows has four main types of objects: *nodes*, *pins*, *packets* and *flows*. Each node represents a running instance of a plug-in, and roughly corresponds to a piece of software or hardware. Each node has a set of pins which represent the possible inputs

and outputs of that node. Flows consist of a set of pins which are chained together. Packets are passed from left to right along a flow, passing the outputs of one pin to the input of the next. Of course because the data which different pins produces is heterogeneous, sometimes it is necessary to convert the data contained within a packet from one format to another to make it compatible with the next pin. Morrison's model of flow based programming requires that data must be converted from the format of the sending object to the format accepted by a receiving object before it can be transmitted between objects, usually through the use of extra conversion nodes (Morrison, 1994, p. 31). Requiring manual conversions between different types of data can be detrimental to flow. Every device input and output of a computer system has a limited set of values which are logical for it to send or receive. For example a joystick might send out values between -1 and 1 to correspond to its tilt along an axis. The range of sensible values for the position of a cursor on a screen would be a range of values between zero and the width of the screen. If one were to attempt to directly map the output of the joystick to the cursor position only a very limited range of motion would be possible (the cursor could either be 0 pixels or 1 pixel from the edge of the screen). In a typical application, programmers must develop methods to logically map one range of values to another. While these methods are usually fairly trivial to implement, the process of implementing them distracts the developer from the real task at hand. BitFlows attempts to free the user to experiment by providing an automatic mapping system between modules. In the plug

in architecture developers have the option of adding semantics to a node's pins, specifying the type of data and range of values which make sense for each pin to receive. Developers can also specify the range of values which a pin can send. Using this information BitFlows automatically converts the data passing between modules to be sensible for each module. Even if a developer doesn't specify a range of values that a pin might output, BitFlows keeps track of the highest and lowest values that pin has output and uses them to create a logical mapping between the pins. In this way, BitFlows automatically calibrates itself to the actions of the user and the idiosyncrasies of various devices.

The auto-conversion system tries to map nearly any type of value to any other so as to allow any pin from any module to be mapped to any pin from any other module. Even data-types which may not intuitively make sense to convert to other types are converted. For example, if one had a pin which outputs a string of characters and connected it to a pin which expects numbers, BitFlows will use the internal Python conversion system to convert the characters to numeric ASCII character codes. While this sort of conversion may not be frequently used, it gives users freedom to experiment with any combination of modules and pins, and the potential to stumble upon unexpected serendipitous combinations. This also aids the potential for the experience of flow while using BitFlows, as there are no wrong connections to try, and no negative feedback from experimentation.

In its most simple form, BitFlows allows users to mash-up any combination of

hardware and software on their computer. The real power of BitFlows however, is revealed when it is used as a tool for connecting those same bits of hardware and software over a network. BitFlows is designed in a way so as to create network transparency. Users of BitFlows do not have to concern themselves with the low level network aspects of connecting pieces of software or hardware. Typically to connect pieces of software over a network, users are required to know the IP address of the machine that they are trying to connect to, and the port on which software on that machine is listening. This process of determining the IP addresses and ports of participating machines can be quite bothersome, particularly in cases where multiple participants all wish to connect to one another. Additionally, only a few pieces of hardware or software natively have any sort of network capability at all. Third party programs, custom software or hardware is necessary to provide these capabilities.

To avoid this problem in BitFlows, the Bonjour networking protocol was used. Bonjour (also known as Zeroconf, or Multicast DNS) is a creative manipulation of the DNS system which allows software and hardware to announce its presence on the local network, and discover what services are available on the network (Internet Engineering Task Force). For example the iTunes music player uses the Bonjour protocol to find other computers that are sharing their music libraries. Many network enabled printers also use Bonjour to allow clients on the network to discover them. Similarly, when BitFlows is started it announces its presence on the network using Bonjour, and then begins listening

for other machines also running BitFlows. When it discovers another machine running BitFlows, the two machines are automatically connected and ready to send data to one another. The user never has to know their or anyone else's IP address, they can just connect to the network and start sharing data immediately. BitFlows also uses Bonjour to announce the availability of its web interface to compatible web browsers such as Safari or Camino so that anyone using one of these web-browsers would be able to access the web-interface of any BitFlows instance on their network simply by clicking on the corresponding link in their "Bonjour Bookmarks" menu. BitFlows also does away with the concept of separate client and server software. Because communication using BitFlows is intended to be bi-directional and spontaneous, it makes no sense for there to be a single server which all clients must connect to. Rather BitFlows has adopted a peer-to-peer model, where all nodes on the network are equal and data can freely flow between any two nodes.

The principle of network transparency also manifests itself in the user experience. As was mentioned earlier, every piece of hardware and software which BitFlows interfaces with is a module, and is treated in the same way by BitFlows. This paradigm remains true in the networked experience of BitFlows. Every piece of software and hardware that has been interfaced with BitFlows on any machine on the local network is available for use in the exact same manner as local software and hardware. It doesn't matter where the soft/hardware is, it is available for play. Because of the networked nature of BitFlows, it

naturally also supports a modular workflow when used as a collaborative tool. As every device and piece of software is considered a module in BitFlows, artists can feel free to develop their work completely separately from BitFlows and from one another. Artists simply need to consider what aspects of their work can use data inputs, and what sort of data their work can produce to be shared. With these considerations in mind it should be simple for artists to connect their work with BitFlows. This phase of development can be done in the presence of collaborators, or completely separately. Once the connections between their work and BitFlows are established, collaborators can then come together and begin experimenting with the various possible connections between their work and other peoples works or devices. Everyone's work is just another module to play with, and these modules can be developed either together with, or separate from each other.

Developing collaborative tools for use over large distances is no simple task, especially when there are very few tools that have attempted to address the issues of electronic collaboration within the same physical space. As outlined in the earlier section on collaboration protocols, while many attempts have been made to develop tools for collaboration over the Internet, few have attempted to create tools to allow co-located collaboration. These efforts often concentrate more on the technical aspects and limitations of networked collaboration, developing new ways to transmit sound and media in low-latency ways, while only giving lip-service to user-experience. BitFlows attempts to reverse this trend by creating a tool which is designed first and foremost to provide for a

smooth collaborative experience with the people around you. Once a work has been developed locally with BitFlows, it is trivial to make the work function over the Internet at large. Virtual Private Network tools such as Hamachi⁹, Tinc¹⁰, Wippen¹¹, and Leaf¹² allow users to create virtual local networks consisting of any set of machines on the Internet. Once such a network is set up, systems developed locally using BitFlows should work nearly identically from anywhere in the world.

9 <https://secure.logmein.com/products/hamachi/vpn.asp>

10 <http://www.tinc-vpn.org/>

11 <http://wippien.com/>

12 <http://www.leafnetworks.net>

IV.c. User Interface

Flow-based programming is not a new concept, and several tools have been built on its principles for general purpose programming, as well as business and artistic purposes. BitFlows takes the inspiration for its user interface both from graph-based programming tools, and the live musical performance software packages Ableton Live and Seq 24. Most graph-based programming tools such as Max/MSP, Pure Data, VVV, and the Yahoo! Pipes mash-up editor have an interface which consists of a canvas upon which programming modules can be freely placed. Usually these modules have one or more input or output ports. The ports of different modules can be connected by graphically drawing lines between them, similar to routing cables between different bits of audio or video hardware. This approach allows for a powerful degree of expressiveness, however as programs increase in complexity it can become difficult to understand what exactly is happening. By contrast, domain specific tools like Ableton Live utilize a simplified approach to routing signals. In these applications, data is separated into a number of channels, each of which can contain a number of effects. Audio data routed to a channel is fed through the effects chain linearly, with the output of one effect being patched directly into the input of the next. This makes for a simple and relatively intuitive interface, but at the expense of interaction between the modules in different channels. It is only in the most recent version of Ableton Live that limited cross-channel interaction was made possible in

the form of "side-chaining".

As has been mentioned earlier, the goal of BitFlows is not to develop a programming tool, but rather a patch bay for control data. As such, in the process of developing the interface for BitFlows it was decided that a full fledged graph-based interface would be unnecessarily complex. Rather, BitFlows takes a hybrid approach to its interface, allowing complex interactions between plug-in modules while retaining a simple intuitiveness. Instead of directly patching together the inputs and outputs of plug-in modules, BitFlows takes a more structured approach. When a BitFlows user instantiates a plug-in, it appears as a "node" in the "Nodes" section of the interface. Under the title of each node there is a list of configuration options for the node and a list of input, output, and modifier pins. To connect the pins of different nodes together, one simply adds them to a "flow". Flows are roughly akin to the channels in an audio application. Data flows in a linear fashion from left to right between the different pins in a flow. A typical flow will begin with an output pin (meaning a pin which outputs data; a data source) followed by any number of modifier pins (which accept data, perform an operation on it, and then return a modified value), and end with an input pin (a pin which accepts data and outputs it in some way external to BitFlows)¹³. So, for example, a flow could consist of the y-axis input

¹³ At first glance this naming scheme might seem confusing, but it is very much in line with how hardware video and audio systems work. The audio output of a guitar is connected to the input of an amplifier, even though the guitar is a input device and the amplifier an output device. In any case, this terminology is hidden to the end user as the status of a pin as an input, output, or modifier is represented in an intuitive graphical rather than textual form.

pin of a Wii Remote node connected to threshold modifier pin (which only passes along data if it goes above or below a certain value) and then to a MIDI output pin which is connected to a drum synthesizer. Using only three pins and a single flow the Wii Remote is transformed into a drum-stick. By retaining the separation between nodes and the connections between different node's pins BitFlows is able to keep nodes independent of the flows unlike in audio software where each effect is bound to a channel. This allows for much more complex interactions between different nodes, while presenting the user with a simpler interface for connecting nodes.

The decision to not use a graph-based interface was also mitigated by a second design decision. Being a tool for networked collaboration and communication, it was decided that the user interface for BitFlows should be network oriented as well. Hence, the entire user interface for BitFlows is built to be accessible from a web-browser¹⁴. This has a number of advantages. For one, it becomes simple to control multiple instances of BitFlows running on different networked computers. This makes it possible to run BitFlows "headlessly" on computers without a monitor, keyboard or mouse. This might be desirable if designing large-scale distributed works that run across several computers. It also allows users to take advantage of platform specific hardware and software simply by adding new computers to the BitFlows network. Another advantage of developing a web interface for

¹⁴ Although Yahoo! Pipes has a graph-based user interface within the web-browser environment, experimentation found the interface to be relatively slow to use and incompatible across a variety of browser platforms.

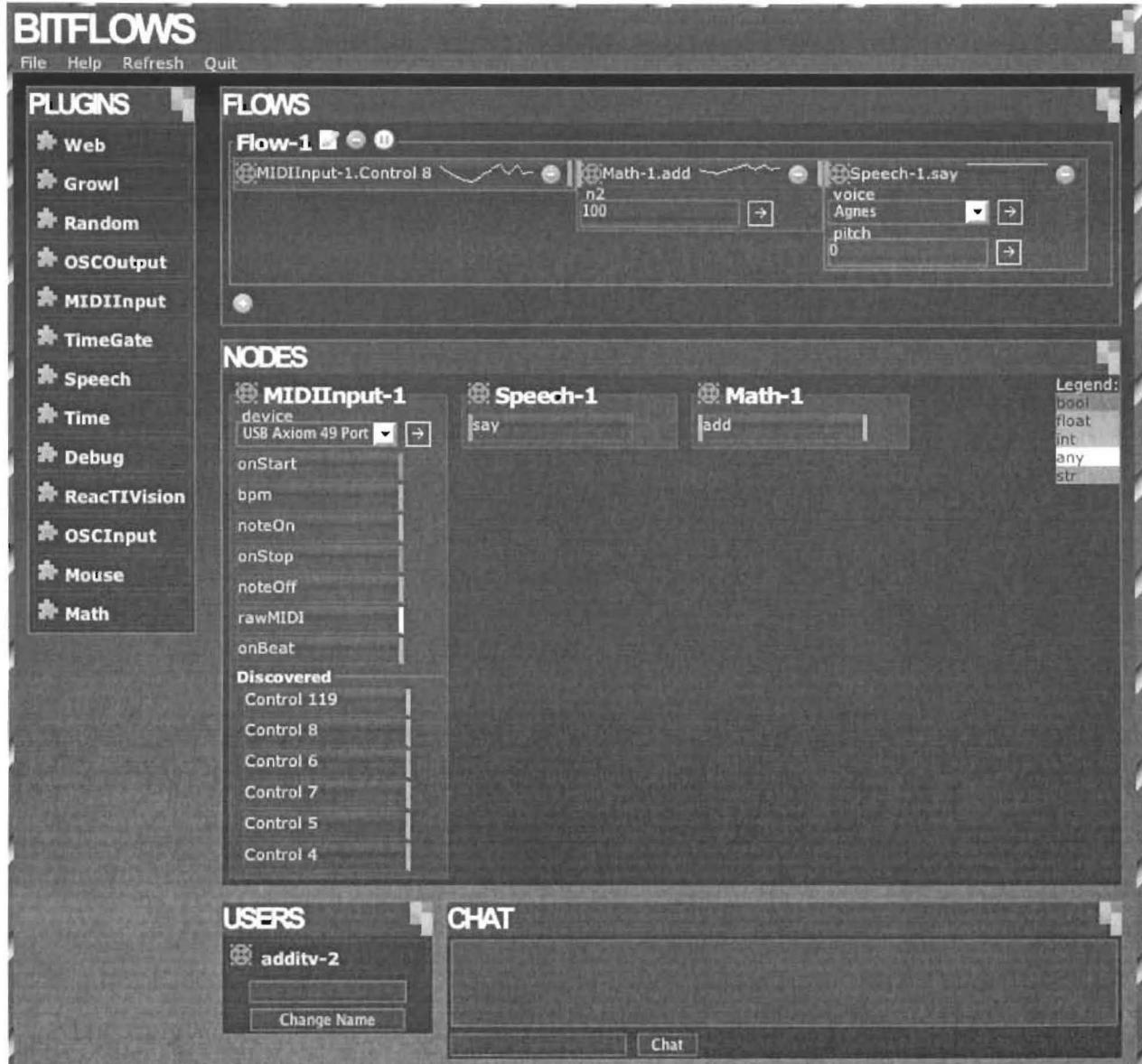


Figure 1: The user interface for BitFlows

BitFlows is that all of the internal methods used to interact with BitFlows are mapped to "RESTful"¹⁵ web URLs. This means that it is possible for users to create their own interfaces to BitFlows using simple HTML or any other language of their preference. Even

¹⁵ Representational State Transfer. A system for mapping function calls to URLs that are user-readable and stateless.

though the interface for BitFlows is web-based, it maintains a rich and responsive user interface using AJAX¹⁶ web development techniques (as used by such rich web applications as GMail and Google Maps).

BitFlows makes nodes running on any machine on the network available for use on any BitFlows instance on the same network. Having multiple instance of the same node running on different machines would become confusing if there was no means of determining which machine was running which nodes. BitFlows incorporates visual and textual cues to differentiate between nodes and pins running on different machines. Upon start-up, each BitFlows instance is assigned a unique name based on the network name of their computer. This name is changeable by the user at any time. A hash of the network name of the computer is also used as a seed to dynamically generate small unique image called an identicon¹⁷:  Because of their small size, these identicons can be used as inline elements to uniquely identify which machine a node or pin exists on. Figure 2 shows three places identicons are used in BitFlows: each user is associated with an identicon in the user's section, this same identicon is used to link any

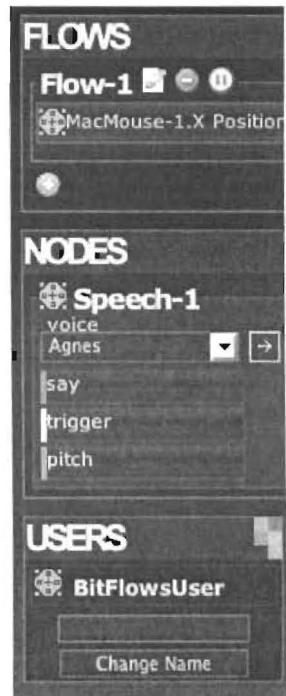


Figure 2: Three uses of identicons in BitFlows

16 Asynchronous JavaScript And XML

17 Original concept and term developed by Don Park. Originally intended for giving unique visual presences to blog commenters.

<http://www.docuverse.com/blog/donpark/2007/01/18/visual-security-9-block-ip-identification>

nodes or pins in a flow to that user's network location.

By default BitFlows also comes with a plug-in module that provides input and output pins for URL based data. This is an incredibly powerful feature which makes developing custom interfaces to artworks a simple matter of designing a web-page and mapping URLs to whatever inputs or outputs are necessary. Using this feature, creating control schemes for mobile devices like cellular phones, the Nintendo DS, the Sony Playstation Portable, or any other device with a web-browser or Internet connection becomes very easy. The use of such an interface for collaborative musical works has been shown in the works of Young (2002) and Yamagishi (1998).

In live performance situations, and in fact in almost any complex network or MIDI set-up, at some point something is not going to work entirely as expected, and the user is going to want to know what exactly is going on in the system. To debug these systems often tools called MIDI monitors, or network sniffers are employed to allow the user to examine the data as it is transferred through the system. These sort of system issues are even more hazardous in live performance environments, trying to debug a MIDI network in real-time can be a nerve wracking (and flow disrupting) experience. Unlike performances which use traditional instruments which have natural sensory feedback methods (for example a de-tuned guitar can be heard, and relatively easily re-tuned), the data flowing through computer systems is invisible.

To help alleviate this problem, BitFlows attempts to give users a constant

awareness of the nature of the data passing through the system. This was done by adopting a technique first proposed by Edward Tufte (2006), called the "sparkline".

Sparklines are "small, high resolution graphics embedded in a context of words, numbers, images". For example this sparkline:

¹⁸ shows the activity of the Dow

Jones on February 7, 2006. This sort of graph is not intended to provide precise information for every data point, but rather to give viewers a sense of general trends. In BitFlows, sparklines are embedded into the graphical representation of every section of a

flow. This way users can immediately see at each

point in a flow whether the data seems random, sine-

wave like, binary, or if any data is being passed at all.

In BitFlows sparklines are integrated into the display of

each pin within a flow. The sparkline shows the last 30 values output by that pin. By moving their mouse over any point on the graph, the user can see the exact value of that point.

One of the other problems when working with invisible data, is the lack of awareness of the sorts of messages which are being sent. This is a problem with both the MIDI and OSC protocols, where devices communicating using either protocol have a wide range of message types which are possible to send. MIDI allows for 256 types of message to be passed over 16 channels and OSC allows for an unlimited range of OSC

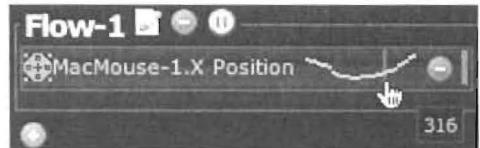


Figure 3: The use of sparklines in BitFlows

¹⁸ Image from the WikiMedia Commons, licensed under a Creative Commons Attribution ShareAlike 2.5 License.

addresses. Most software and devices come with a manual which lists the messages used, but searching through documentation is generally not conducive to the experience of flow. BitFlows addresses this issue by introducing message auto-discovery for both MIDI and OSC. When users create an instance of the OSC receiver or MIDI receiver modules, BitFlows listens for any messages being sent to the corresponding network or MIDI port. When a new message is received, the module will examine the message type and automatically generate new output pins for that module which correspond to the type of message received. Users can then use this new pin to access any future messages of the same type. Thus to interface with new or unknown data flows, users simply need to create some example data, and BitFlows will make it accessible. This is much simpler and more conducive to flow than manually searching for the appropriate message type.

The auto-discovery capability of the MIDI and OSC plug-ins can also be taken advantage of to automatically generate new plug-ins. The source code of the MIDI and OSC plug-ins is annotated with very simple template instructions contained in comment blocks. As the plug-ins discover new data and add new pins, they also use these templating instructions to analyse their own source code, and dynamically generate a file containing the code needed to recreate the plug-in with its current set of pins. To create a new plug-in is simply a matter of copying the generated file into a new plug-in directory. Using this sort of automatic plug-in generation, even non-programmers who wish to develop their own plug-ins can be given a head start at development.

BitFlows was built in Python, a high-level scripting language. The decision to use Python was made because of its ease of use, large community, deployed support, and range of external modules available. High-level languages such as Python serve as a natural platform for artistic work. Paul Graham (Graham, 2004) argues that using a high-level language allows individuals to use a programming style more similar to sketching than low-level languages . One of the other strengths of scripting languages like Python is that it is possible to execute code without requiring a compilation step. This makes possible the development of plug-ins for live coding activities. As BitFlows is designed to be able to interface with most widely used audio and video performance software, this has the potential to simply add live coding and procedural performance techniques to programs which were not originally designed with these capabilities in mind. Additionally BitFlows allows bindings between the core Python code and external Java modules using Jython (an implementation of Python in Java). This allows developers to take advantage of libraries available both for Python and Java, as well as artistically oriented flavours of Java such as Processing¹⁹. The choice to use Python and Java as the languages for BitFlows was further motivated by a desire for BitFlows to be as platform independent as possible. Artists use a wide range of operating systems and platforms to develop their work, and incompatibilities between platforms should not be a barrier to artistic collaboration. In BitFlows, great lengths were taken to ensure cross-platform compatibility. Though Python

19 <http://processing.org>

is a cross-platform language, it has the capability to use 'binary' modules that wrap pieces of platform-specific C/C++ code. Though certain features of BitFlows would have been easier to implement using some readily available binary Python libraries (for example the joystick module could have used the PyGame library), in the interests of having a single BitFlows distribution work on all platforms, these libraries were explicitly avoided opting instead for "pure" Python libraries whenever possible. A side-effect of this decision was the implementation of the Python-Java bridging in BitFlows. This allowed the range of cross-platform Java libraries to be accessed by BitFlows. In the end, the flexibility of a completely cross-platform solution, and an extended range of Java libraries should be worth the extra effort expended in this area.

IV.d. Using BitFlows

BitFlows is designed with three major use cases in mind: single-user single-computer, single-user multi-computer, and multi-user multi-computer. In the most basic use case – single-user single-computer – a user desires to connect multiple pieces of hardware or software within a single computing environment, for example developing a musical performance that has a visual aspect which reacts in a meaningful way to the music. The typical workflow in this case would go as follows: First, separate from BitFlows the user would work on developing the main elements of their work in the software or hardware system(s) of their choice (e.g. Ableton Live, Max/MSP, VVV). During this development phase the user should be thinking about what parts of their work they might wish to use as sources of or destinations for data. As the development of the individual elements progresses, users may begin connecting parts to BitFlows and experimenting with different interactions. Users start BitFlows either by running a Python script, or if they have a version packaged for their operating system simply double-clicking on the application file. This will start the BitFlows engine, and load the web-interface for BitFlows in their default browser. The user can then begin instantiating plug-ins for each piece of hardware or software they wish to play with. To do

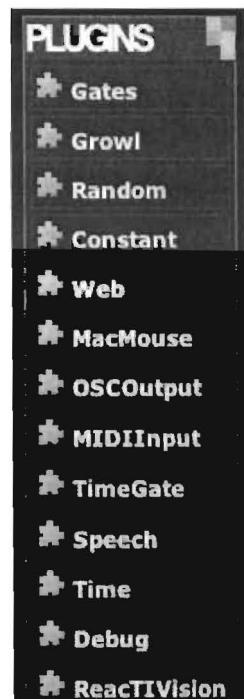


Figure 4: A list of plug-ins available to a BitFlows user.

this they simply click on a plug-in in the list of plug-ins (Figure 4). This creates an instance of that plug-in (or node) in the BitFlows engine, which is displayed in the user interface in the “Nodes” section (Figure 5). Each node is roughly analogous to a single piece of hardware or software, so if a user wished to use a MIDI keyboard and a separate set of MIDI faders they would create two MIDIIInput plug-in instances, one for each device. If they are using plug-ins that support auto-discovery the user should then send BitFlows some example data from the control they wish to use. In Figure 5 for example, the user has created a MIDIIInput plug-in instance and moved the five knobs that they want to use. BitFlows used this data to create the new pins “Control 5”, “Control 6”, etc. Next the user can begin adding flows (data channels) and adding pins to the flows by either clicking on the pin and selecting the flow to add it to (Figure 6), or dragging the pin and dropping it on

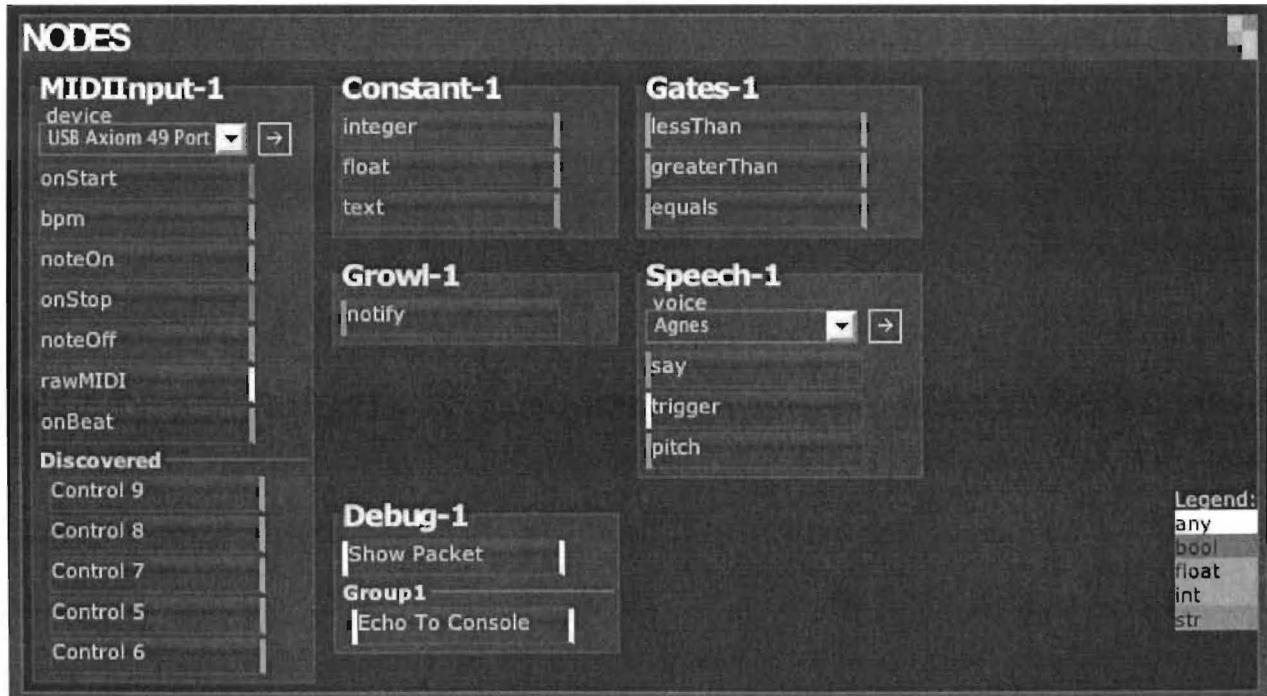


Figure 5: Six instantiated BitFlows plug-ins (nodes).

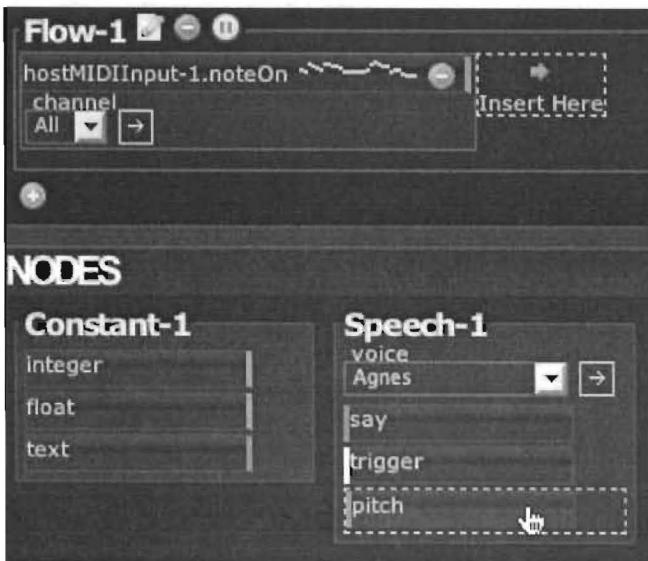


Figure 6: Adding a pin to a flow

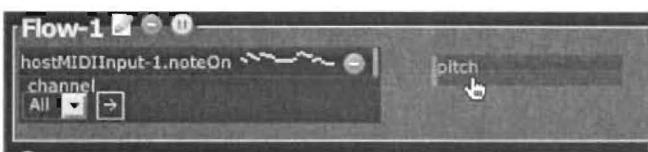


Figure 7: Dragging a pin to a flow.

the flow (Figure 7). Once added to a flow, a pin will immediately start outputting data. By adding the appropriate plug-ins for each piece of hardware or software that a user wishes to use it should be easy for the user to start experimenting with a variety of different interactions. Figure 8 shows a simple set of flows that creates a text-to-speech based instrument where each MIDI keyboard key press makes a text-to-speech plug-in say “Hello World” at different pitches according to the MIDI note number. Although this example may only be of limited use in practice it shows how simple it can be to create interesting interactions in BitFlows. The example uses only three plug-ins (a MIDI input plug-in, a text-to-speech plug-in, and a constant plug-in) and three flows (one to set the phrase to say, one to set the pitch, and one to trigger the speech).

In the example case of a user wanting to develop a closely tied visual interaction with a musical performance piece they could create a flow where a single MIDI input would trigger both a audio-sample or musical phrase and a video-clip. As the user experiments more they could find that binding the velocity of a Wii remote's motion to the speed of a

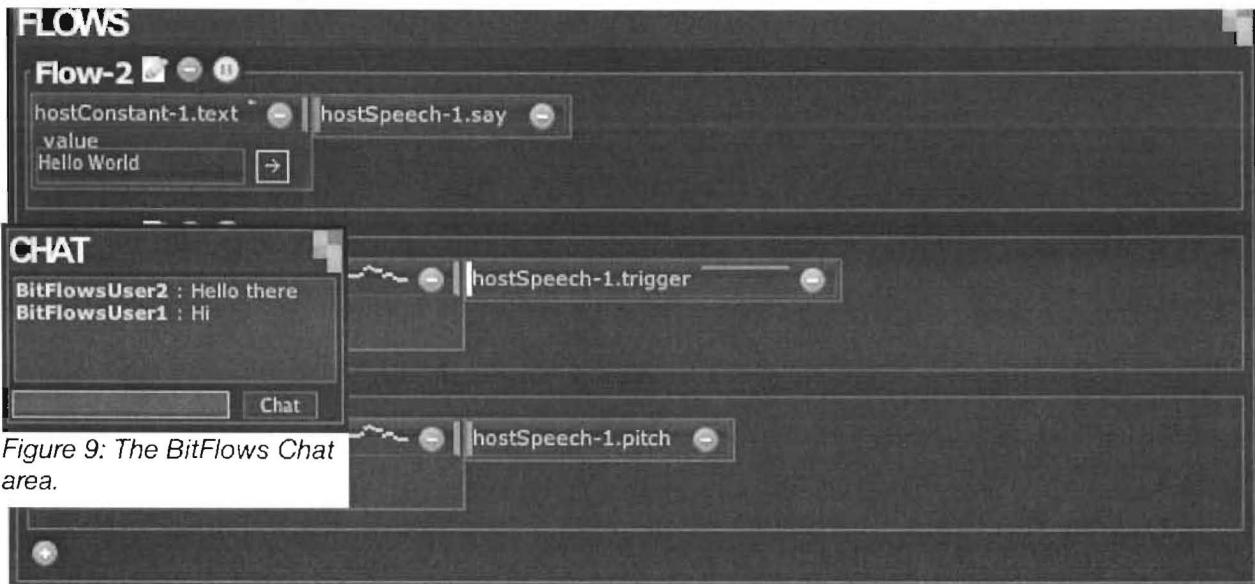


Figure 9: The BitFlows Chat area.

Figure 8: A simple text-to-speech instrument built entirely in BitFlows

video clip and the volume of an audio track creates an interesting performance tool.

Alternately they could set up their music software to send a signal upon a key or mood change which would then change the colour palette of the video to something appropriate. The user could even take the opposite approach, where the velocity of an object in their video piece affects an audio filter.

BitFlows in either single- or multi-user multi-computer mode requires pretty much the same user-interaction with one small addition. Any computers running BitFlows on the local network (or remote network if running over a virtual private network using a tool such as Hamachi) appear in the Users section of the user interface (Figure 10). By clicking on the name of any of the users in the user list, the user is brought to the web interface for that machine. From here a user can instantiate plug-ins on that machine remotely. The nodes running on every machine are shown in the nodes section of every user's interface (Figure 11). The location of each node on the network is identified by an



Figure 10: Two users in BitFlows' user list.

identicon associated with each node. The process of creating flows and adding pins to flows is identical to the single-user single-computer experience. A user can mix and match pins from different nodes spread across the network just as easily as adding pins on a local machine. The network is transparent to the user. In multi-user situations, a simple chat application is also provided to facilitate communications across distances, or to let users communicate without talking in performance settings (Figure 9).

There are several reasons why a single user might wish to use BitFlows in a multi-computer environment. It allows users to blend platform specific software and hardware seamlessly. For example they might wish to use the Linux specific Seq24 music software to perform a musical piece and use the Mac OS X specific Quartz Composer to create reactive visuals. Likewise at the moment the Wii remote is much simpler to connect to a Mac than to a Windows computer. Connecting these pieces of software and hardware over the network using BitFlows is as simple as if they were running on the same machine.

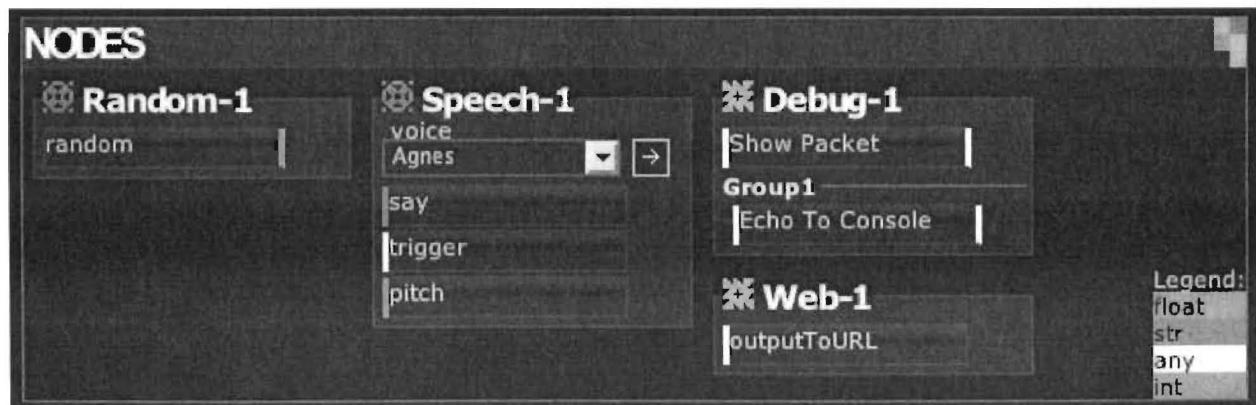


Figure 11: Four nodes from two different machines. Note the use of identicons to show which node is running on which machine.

BitFlows could also be used to control works distributed over a multitude of computers on the network. Were BitFlows to be installed on every computer in a typical school computer lab it could be possible to control the audio output from each computer, facilitating unique experiences such as a 50-channel distributed musical piece. Alternately the mouse and keyboard inputs of each machine could be captured, and used to control the audio output of a single machine, effectively creating a computer orchestra where anyone using one of the computers is also a performer. The screens of each computer could even be used as individual pixels of a large scale display, controlled from a single computer. In multi-user environments BitFlows permits collaborative spontaneity. For example a laptop musician and a computer visualist who have never met before could connect to a local wireless network and through BitFlows immediately start exploring different interactions between their individual works. Using a VPN tool such as Hamachi it would even be simple to make a locally created performance piece into a remote networked experience.

V. Future Directions

BitFlows has been developed as a platform for expansion. In its current iteration, BitFlows contains most of the plug-ins necessary for users to begin their explorations of interesting mash-ups and networked collaborative experiences. As has already been discussed, New Media is a constantly shifting field and new technologies are constantly being developed. The plug-in system for BitFlows was developed to accommodate for this constant change. It is the hope of the author that BitFlows will grow to be a community driven project with users contributing plug-ins for new hardware and software. To help facilitate this a web presence will need to be developed to allow users to upload new plug-ins which they have developed and have a central location to discuss their uses of BitFlows with other users.

One of the specific areas of plug-in development which I would like to focus on is the development of close integration with a variety of common software environments. For example the development of an “external” for Max/MSP and Pure Data which allows the same sort of dynamic data-source discovery as the current OSC and MIDI plug-ins. Using such a module would allow Max/MSP and Pure Data users to easily integrate their patches with BitFlows.

One of the eventual goals for BitFlows is to develop it into a platform for live coding.

Using the current plug-in system it should be possible to develop a plug-in that allows users to input Python code dynamically into the interface to create new output signals on the fly. The development of such a plug-in would allow any software or hardware connected to BitFlows to be used as a part of a live coding performance.

While every attempt has been made to make BitFlows as flexible as possible it still has some outstanding limitations. BitFlows currently doesn't have support for branching flows so that multiple pins can feed different inputs of a single pin. This sort of interaction can be seen as useful in cases where multiple data-sources should feed a single output such as a MIDI note message which has both a note-number and velocity variable. It is probable that this issue is solvable in the current architecture by providing separate pins for different variables as well as a trigger pin (as seen with the text-to-speech plug-in example) however this solution has not been fully explored at the moment. Linked with this issue is the problem of scheduling the order of events in BitFlows. Flows are currently executed in parallel. The current order of execution in BitFlows is that packets from flows are distributed to each node. Each node then executes the appropriate methods for each pin, and returns the results from all of it's pins. This may cause issues when dealing with situations such as the MIDI note issue described above where the pitch and velocity of the note must be set before the note is triggered. In a future iteration a more robust method for scheduling events will be developed.

As has been explained earlier, BitFlows allows plug-ins to be developed in both Python and Java using Jython. The current method of communication between Python and Java uses network sockets within a machine. This allows for simplicity in coding but could possibly cause a slight performance hit as local sockets are not usually as fast as direct processing within a single application. Future iterations of this functionality might use other inter-process communication techniques like FIFO to increase processing speed.

Finally a future iteration of BitFlows might incorporate a non-web-based user interface to increase interface responsiveness and user-feedback in single-user single-computer environments.

VI. Conclusion

If we return to Csikszentmihalyi's elements of flow, Hoffman's additional antecedents to flow, and Sawyer's characteristics of group creativity, we can see the ways in which BitFlows attempts to support flow experiences in both individual and collaborative contexts. The modular interaction made capable by BitFlows helps create a balance between the challenges of creative work and the skills of the creator. Individuals working in creative domains naturally must first learn their basic tools, be it the development of visual environments in VVV or the construction of a musical performance in Ableton Live. As creators become familiar with their base environment and desire to extend its possibilities BitFlows comes into play. BitFlows allows a simple means of exploring the range of possibilities of connecting various pieces of software and hardware, and developing new control systems and interactions. BitFlows also allows individuals to collaboratively integrate the creative works of others into a single work. BitFlows gives the creator as large or small of a creative pallet as they desire, expanding the range of possible interactions as the individual's skills increase. This modular design also allows the simple creation of hardware/software mash-ups and combinatorial play.

The user interface is designed to give users immediate feedback as to how a set of flows is functioning. The incorporation of sparklines to show data at each stage within a flow allows for a deeper understanding of how data is passing through a flow as well as

debugging capabilities. Likewise, identicons give users immediate feedback as to their position on, and use of the network. The interactive and real-time nature of BitFlows helps to give the user feedback regarding the operation of BitFlows as well as merging action and awareness. Every action taken within BitFlows has an immediate consequence. BitFlows contains no drop down menus or hidden options, all the possible interactions are immediately visible. Distractions within BitFlows are minimized through the auto-conversion mechanism as well as the data detection and automatic network connection mechanisms. Users will not be taken out of flow by the necessity to figure out how to connect two pins with different sorts of data, connect to another BitFlows user, or find out what sorts of data a device outputs. The auto-conversion mechanism also allows users to connect any series of pins without fear of failure. Free-association is supported by designing the modules so that there is no "wrong" way to connect any selection of BitFlows modules. The simplicity of BitFlows is designed so that users can proceed directly from a set of goals to a working prototype without taking them out of their creative flow. At the same time BitFlows doesn't require clear goals, but allows for play, improvisation, experimentation and emergent discovery. The ability for playful interaction is intended to help make the use of BitFlows an autotelic activity. The networking capabilities of BitFlows allows for both local collaboration and remote telepresence.

The open-source nature of BitFlows also caters to the situationalist model of creativity, at it's root BitFlows is a social and collaborative tool. BitFlows is built to facilitate

technological exploration in groups, where inspiration can be shared. The open plug-in system further supports community interactions, as any individual using BitFlows can develop a new plug-in and easily share it with the larger community of BitFlows users. A significant amount of time and effort has gone into the design and development of BitFlows. It is the hope of the author that BitFlows will be adopted by the New Media community as a tool for exploring new and innovative creative and collaborative expressions, creating and using hacks, developing mash-ups, live-performance, and networked experiences.

References

- Agarwal, R., & Prasad, J. (1998). A conceptual and operational definition of personal innovativeness in the domain of information technology., *Information Systems Research*, 9(2), 204-215. doi: Article.
- Ascott, R. (2003). *Telematic embrace visionary theories of art, technology, and consciousness*. Berkeley: University of California Press.
- Banzi, M., Cuartielles, D., Igoe, T., Martino, G., & Zambetti, N. Arduino. Retrieved February 25, 2008, from <http://www.arduino.cc/>.
- Barbosa, A. (2003). Displaced soundscapes: a survey of network systems for music and sonic art creation, *Leonardo Music Journal*, 13(1), 53-59.
- Bartlette, C., Headlam, D., Bocko, M., & Velikic, G. (2006). Effect of network latency on interactive musical performance, *Music Perception*, 24(1), 49-62.
- Bederson, B. B. (2004). Interfaces for staying in the flow, *Ubiquity*, 5(27), 1-1.
- Berners-Lee, T. (1999). *Weaving the web : the original design and ultimate destiny of the world wide web by its inventor* (1st ed.). San Francisco: HarperSanFrancisco.
- Biaz, S., Chapman, R., & Williams, J. (2005). Rtp and tcp based midi over ip protocols, *Proceedings of the 43rd ACM Southeast Conference, March*, 18-20.
- Bischoff, J., Gold, R., & Horton, J. (1978). Music for an interactive network of microcomputers, *Computer Music Journal*, 2(3), 24-29.

- Blackwell, A., & Collins, N. (2005). The programming language as a musical instrument, *Proceedings of PPIG05 (Psychology of Programming Interest Group)*.
- Blaine, T., & Fels, S. (2003). Contexts of collaborative musical experiences, *Proceedings of the 2003 conference on New interfaces for musical expression*, 129-134.
- Bongers, B. (1999). Exploring novel ways of interaction in musical performance, *Proceedings of the Third Conference on Creativity & Cognition*, 76-81.
- Bouillot, N. (2007). Njam user experiments: enabling remote musical interaction from milliseconds to seconds, *Proceedings of the 7th international conference on New interfaces for musical expression*, 142-147.
- Bringuer, J. (1980). *Conversations with jean piaget*. Chicago: University of Chicago Press.
- Brown, A. R. (2006). Code jamming, *M/C Journal*, 9(6).
- Burk, P. L. (2000). Jammin'on the web—a new client/server architecture for multi-user performance, *Proceedings of the 2000 International Computer Music Conference*, 117–120.
- Carot, A., Kramer, U., & Schuller, G. (2006). Network music performance (nmp) in narrow band networks In . Paris, France.
- Cascone, K. (2000). The aesthetics of failure: "post-digital" tendencies in contemporary computer music ., *Computer Music Journal*, 24(4), 12. doi: Article.
- Chafe, C., Wilson, S., Leistikow, R., Chisholm, D., & Scavone, G. (2000). A simplified approach to high quality music and sound over ip, *Proc. Workshop on Digital Audio Effects (DAFx-00)*, Verona, Italy, 159-163.

- Chen, H., Wigand, R. T., & Nilan, M. (2000). Exploring web users' optimal flow experiences, *Information Technology & People*, 13(4), 263-281.
- Collins, N. (2003). Generative music and laptop performance, *Contemporary Music Review*, 22(4).
- Collins, N., McLean, A., Rohrhuber, J., & Ward, A. (2004). Live coding in laptop performance, *Organised Sound*, 8(3), 321-330.
- Cook, G., & Misra, A. Designing and implementing the chuck programming language, *Proceedings of the 2005 International Computer Music Conference*.
- Correa, C. D., & Marsic, I. (2004). Software framework for managing heterogeneity in mobile collaborative systems., *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, 13(5/6), 603-638.
- Crabtree, B., & Cain, K. (2007). Monome. Retrieved February 25, 2008, from <http://monome.org/>.
- Csikszentmihalyi, M. (1990). *Flow: the psychology of optimal experience* (1st ed. --), 303. New York: Harper & Row.
- Csikszentmihalyi, M. (1996). *Creativity: flow and the psychology of discovery and invention* (1st ed), 456. New York: HarperCollinsPublishers.
- Dawes, B. (2007). *Analog in, digital out : brendan dawes on interaction design*. Berkeley CA: New Riders.
- Dewey, J. (1916). *Democracy and education: an introduction to the philosophy of education*. New York: Macmillan.

- Do, E. Y. L., & Gross, M. D. (2007). Environments for creativity: a lab for making things, *Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*, 27-36.
- Edmonds, E. A., Weakley, A., Candy, L., Fell, M., Knott, R., & Pauletto, S. (2005). The studio as laboratory: combining creative practice and digital technology research, *International Journal of Human-Computer Studies*, 63(4-5), 452-481.
- Ehn, P. (1998). Manifesto for a digital bauhaus., *Digital Creativity*, 9(4), 207. doi: Article.
- Farooq, U. (2005). Eureka! past, present, and future of creativity research in hci, *Crossroads*, 12(2), 6-6.
- Ferguson, W. (2004). The mainstream mash-up, *The New York Times*.
- Fiebrink, R., Wang, G., & Cook, P. R. (2007). Don't forget the laptop: using native input capabilities for expressive musical control, *Proceedings of the 7th international conference on New interfaces for musical expression*, 164-167.
- Finneran, C. M., & Zhang, P. (2002). The challenges of studying flow within a computer-mediated environment, *Eighth American Conference in Information Systems, Dallas, TX*, 1047-1054.
- Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A. G., & Mehandjiev, N. (2004). Meta-design: a manifesto for end-user development, *Communications of the ACM*, 47(9), 33-37.
- Fleck, R. (2003). How the move to physical user interfaces can make human computer interaction a more enjoyable experience, *Workshop on Real World User Interfaces, Mobile HCI*.
- Fry, B., & Reas, C. Processing.org, *Processing.org*. Retrieved from <http://processing.org>.

Galloway. (2006). "carnivore personal edition": exploring distributed data surveillance, *AI & Society*, 20(4), 483-492. doi: 10.1007/s00146-006-0034-9.

Galloway, A. R. (2006). *Gaming: essays on algorithmic culture* (1), 160. Univ Of Minnesota Press.

Gang, D., Chockler, V., Anker, T., Kremer, A., & Winkler, T. Transmidi: a system for midi sessions over the network using transis, *Proceedings of the International Computer Music Conference (ICMC 1997)*.

Ghani, J. A., Supnick, R., & Rooney, P. (1991). The experience of flow in computer-mediated and in face-to-face groups In , *Proceedings of the twelfth international conference on Information systems* (pp. 229-237). New York, New York, United States: University of Minnesota.

Graham, P. (2004). *Hackers & painters : big ideas from the computer age* (1st ed.). Sebastopol CA: O'Reilly.

Green, D. (1995, July). Demo or die!, *Wired*, 3(7).

Gresham-Lancaster, S. (1998). The aesthetics and history of the hub: the effects of changing technology on network computer music, *Leonardo Music Journal*, 8, 39-44.

Gu, X., Dick, M., Kurtisi, Z., Noyer, U., & Wolf, L. (2005). Network-centric music performance: practice and experiments, *Communications Magazine, IEEE*, 43(6), 86-93.

Gu, X., Dick, M., Noyer, U., & Wolf, L. Nmp-a new networked music performance system, *Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004. IEEE*, 176-185.

Gurevich, M. (2006). Jamspace: a networked real-time collaborative music environment, *Conference on Human Factors in Computing Systems*, 821-826.

Gurevich, M., Chafe, C., Leslie, G., & Tyan, S. (2004). Simulation of networked ensemble performance with varying time delays: characterization of ensemble accuracy, *Proceedings of the International Computer Music Conference, Miami*.

Han, J. Y. (2005). Low-cost multi-touch sensing through frustrated total internal reflection In , *Proceedings of the 18th annual ACM symposium on User interface software and technology* (pp. 115-118). Seattle, WA, USA: ACM.

Han, J. Y. (2008). Ftir touch sensing, *FTIR Touch Sensing*. Retrieved from <http://cs.nyu.edu/~jhan/ftirsense/>.

Hickey, M. (1998). Exploring music collaboration over the internet, *Proceedings of the Fifth International Technological Directions in Music Learning Conference*.

Himanen, P., & Torvalds, L. (2001). *The hacker ethic* (1), 256. Random House.

von Hippel, E. (2001). User toolkits for innovation, *Journal of Product Innovation Management*, 18(4), 247-257.

Hoffman, D. L., & Novak, T. P. (1996). Marketing in hypermedia computer-mediated environments: conceptual foundations, *Journal of Marketing*, 60(3), 50-68.

Ingalls, D., Kaehler, T., Maloney, J., Wallace, S., & Kay, A. (1997). Back to the future: the story of squeak, a practical smalltalk written in itself, *SIGPLAN Not.*, 32(10), 318-326.

Integrated Digital Media Institute of Polytechnic University, Brooklyn. (2007). Mrrmr - idmresearch, *Mrrmr - IDMResearch*. Retrieved February 25, 2008, from <http://poly.share.dj/wiki/index.php/Mrrmr>.

Internet Engineering Task Force. Zero configuration networking (zeroconf). Retrieved November 8, 2006, from <http://www.zeroconf.org/>.

Ippolito, J. (2002). Ten myths of internet art, *LEONARDO*, 35(5), 485-498.

Ishii, H., & Ullmer, B. (1997). Tangible bits: towards seamless interfaces between people, bits and atoms In , *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 234-241). Atlanta, Georgia, United States: ACM. doi: 10.1145/258549.258715.

Iturzaeta, B., & Iturzaeta, L. (2007). Psp rhythm is the homebrew audio sequencer for the sony playstation portable. Retrieved February 25, 2008, from <http://www.psprhythm.com/>.

JazzMutant. Jazzmutant lemur. Retrieved February 25, 2008, from http://www.jazzmutant.com/lemur_overview.php.

Jeppesen, L. (2001). *Organizing and tapping consumer communities*. Copenhagen.

Jeppesen, L. (2002). The implications of “user toolkits for innovation”, *Work*.

Jeppesen, L. (2005). User toolkits for innovation: consumers support each other, *Journal of Product Innovation Management*, 22(4), 347-362.

Jorda, S. (1999). Faust music on line: an approach to real-time collective composition on the internet, *Leonardo Music Journal*, 9, 5-12.

Kaplan, L. (1993). The telephone paintings: hanging up moholy, *Leonardo*, 26(2), 165-168.

Kapur, A., Wang, G., Davidson, P., & Cook, P. R. (2005). Interactive network performance:

a dream worth dreaming, *Organised Sound*, 10(3), 209-219.

King, R. (2006). Playlive psp midi controller. Retrieved February 25, 2008, from <http://e-mu.org/?p=20>.

King, R., & Ramella, N. (2007). Ableton live python api, *Ableton Live Python API*. Retrieved from <http://liveapi.org>.

Koestler, A. (1964). *The act of creation*. Hutchinson.

Korzaan, M. L. (2003). Going with the flow: predicting online purchase intentions, *Journal of Computer Information Systems*, 43(4), 25-31.

Kuittinen, P. (2004). Computer demos-the story so far, *Intelligent Agent*, 4(1).

Latta, C. (1991). Notes from the netjam project, *Leonardo Music Journal*, 1(1), 103-105.

Lazzaro, J., & Wawrynek, J. (2001). A case for network musical performance, *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, 157-166.

Lee, J. C. (2007). Johnny chung lee - projects - wii. Retrieved February 25, 2008, from <http://www.cs.cmu.edu/~johnny/projects/wii/>.

Lieberman, J. (1977). *Playfulness : its relationship to imagination and creativity*. New York: Academic Press.

Loy, G. (1985). Musicians make a standard: the midi phenomenon, *Computer Music Journal*, 9(4), 8-26.

Lunenfeld, P. (2000). *Snap to grid: a user's guide to digital arts, media, and cultures*, 252.

The MIT Press.

Malone, T. W. (1981). Toward a theory of intrinsically motivated instruction, *Cognitive Science*, 4, 333-369.

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., & Resnick, M. (2004). Scratch: a sneak preview [education] In , *Creating, Connecting and Collaborating through Computing, 2004. Proceedings. Second International Conference on* (pp. 104-109). doi: 10.1109/C5.2004.1314376.

Mamykina, L., Candy, L., & Edmonds, E. (2002). Collaborative creativity, *Communications of the ACM*, 45(10), 96-99.

Manovich, L. (2002). *The language of new media* (1st MIT Press pbk. ed.). Cambridge Mass.: MIT Press.

Martocchio, J. J., & Webster, J. (1992). Effects of feedback and cognitive playfulness on performance in microcomputer software training, *Personnel Psychology*, 45(3), 553-578.

McLeod, K. (2005). Confessions of an intellectual (property): danger mouse, mickey mouse, sonny bono, and my long and winding path as a copyright activist-academic, *Popular Music and Society*, 28(1), 79-93.

Merz, E. (2007). Thisisnotalabel.com - my wiimote drum kit. Retrieved February 25, 2008, from <http://www.thisisnotalabel.com/My-Wiimote-Drum-Kit.php>.

Monk, A. (2002). Fun, communication and dependability: extending the concept of usability, *People and Computers XVI. Springer*, 3-14.

Moore, F. R. (1988). The dysfunctions of midi, *Computer Music Journal*, 12(1), 19-28.

- Morrison, J. P. (1994). *Flow-based programming: a new approach to application development*. Van Nostrand Reinhold.
- Nachmanovitch, S. (1990). *Free play : improvisation in life and art*. New York: G.P. Putnam's Sons.
- Nakakoji, K., Yamamoto, Y., & Ohira, M. (2000). Computational support for collective creativity, *Knowledge-Based Systems*, 13(7-8), 451-458.
- Nieborg, D. (2005). Am i mod or not?-an analysis of first person shooter modification culture., *Creative Gamers Seminar-Exploring Participatory Culture in Gaming, Hypermedia Laboratory (University of Tampere)*.
- Nilson, C. (2007). Live coding practice, *Proceedings of the 7th international conference on New interfaces for musical expression*, 112-117.
- Novak, T. P., Hoffman, D. L., & Yiu-Fai Yung. (2000). Measuring the customer experience in online environments: a structural modeling approach., *Marketing Science*, 19(1), 22. doi: Article.
- Ox, J. (2002). 2 performances in the 21st century virtual color organ In , *Proceedings of the 4th conference on Creativity & cognition* (pp. 20-24). Loughborough, UK: ACM. doi: 10.1145/581710.581715.
- Penny, S. (1995). Consumer culture and the technological imperative: the artist in dataspace, *Critical Issues in Electronic Media*.
- Postigo, H. E. -. (2003). From pong to planet quake: post-industrial transitions from leisure to work, *Information, Communication & Society*, 6(4), 593-607.
- Puckette, M. (2002). Max at seventeen, *Computer Music Journal*, 26(4), 31-43.

Rasmussen, A. (2007). Wiibot, *WiiBot*. Retrieved February 25, 2008, from <http://www.usmechatronics.com/usmgarage/WiiBot.html>.

Raymond, E. (1996). *The new hacker's dictionary* (3rd ed.). Cambridge Mass.: MIT Press.

Raymond, E. (2004). *The art of unix programming*. Boston: Addison-Wesley.

Reid, D. (2004). A model of playfulness and flow in virtual reality interactions., *Presence: Teleoperators & Virtual Environments*, 13(4), 451-462. doi: Article.

Roussou, M. (2004). Learning by doing and learning through play: an exploration of interactivity in virtual environments for children, *Computers in Entertainment (CIE)*, 2(1), 10-10.

Sawyer, R. K. (2000). Improvisation and the creative process: dewey, collingwood, and the aesthetics of spontaneity, *The Journal of Aesthetics and Art Criticism*, 58(2, Improvisation in the Arts), 149-161.

Sawyer, R. K. (2006). Group creativity: musical performance and collaboration, *Psychology of Music*, 34(2), 148-165.

Schilpp, P. A. (1970). Autobiographical notes In , *Albert Einstein: Philosopher-Scientist*, The Library of Living Philosophers. (pp. 1-97). La Salle, IL: Open Court.

Seznec, Y. (2008). Wii loop machine. Retrieved February 25, 2008, from <http://www.theamazingrolo.net/wii/>.

Shneiderman, B. (2000a). Creating creativity: user interfaces for supporting innovation, *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1), 114-138.

Shneiderman, B. (2000b). Supporting creativity with powerful composition tools for

artifacts and performances, *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, 9.

Snow, C. P. (1998). *The two cultures*, 107. Cambridge: Cambridge University Press.

Tanaka, A., Tokui, N., & Momeni, A. (2005). Facilitating collective musical creativity, *Proceedings of the 13th annual ACM international conference on Multimedia*, 191-198.

TED Conferences, LLC. (2006). Ted | talks | jeff han: unveiling the genius of multi-touch interface design, *TED | Talks | Jeff Han: Unveiling the genius of multi-touch interface design*. Retrieved from <http://www.ted.com/index.php/talks/view/id/65>.

Thille, T. (2007). I3I - iphone to midi bridge | artificialeyes.tv, *Artificialeyes.tv*. Retrieved February 25, 2008, from <http://artificialeyes.tv/node/543>.

Thomson, P. (2004). Atoms and errors: towards a history and aesthetics of microsound, *Organised Sound*, 9(02), 207-218.

Trevino, L., & Webster, J. (1992). Flow in computer-mediated communication: electronic mail and voice mail evaluation and impacts, *Communication Research*, 19(5), 539.

Truax, B., & Barenholtz, J. (1977). Models of interactive computer composition In , *Computing in the humanities : proceedings of the Third International Conference on Computing in the Humanities*. Waterloo Ont.: University of Waterloo Press.

Tufte, E. (2006). *Beautiful evidence*. Cheshire Conn.: Graphics Press.

Turkle, S. (1995). *Life on the screen: identity in the age of the internet*. Simon & Schuster Trade.

- Underwood, B. (2007). Ninjam - novel intervallic network jamming architecture for music - main. Retrieved November 8, 2006, from <http://www.ninjam.com/>.
- Villar, N., Lindsay, A., & Gellersen, H. (2005a). A rearrangeable tangible interface for musical composition and performance, *Proc. of New Interfaces for Musical Expression (NIME 05)*.
- Villar, N., Lindsay, A. T., & Gellersen, H. (2005b). Pin & play & perform: a rearrangeable interface for musical composition and performance, *Proceedings of the 2005 conference on New interfaces for musical expression*, 188-191.
- Wanderley, M., & Battier, M. (2000). *Trends in gestural control of music*. Ircam.
- Wang, G., Misra, A., & Cook, P. R. (2006). Building collaborative graphical interfaces in the audicle, *Proceedings of the 2006 conference on New interfaces for musical expression*, 49-52.
- Wang, G., Misra, A., Davidson, P., & Cook, P. R. (2005). Co-audicle: a collaborative audio programming space, *Proceedings of the International Computer Music Conference*.
- Wang, G., & Cook, P. R. (2004). On-the-fly programming: using code as an expressive musical instrument In , *Proceedings of the 2004 conference on New interfaces for musical expression* (pp. 138-143). Hamamatsu, Shizuoka, Japan: National University of Singapore.
- Webster, J., & Martocchio, J. J. (1992). Microcomputer playfulness: development of a measure with workplace implications., *MIS Quarterly*, 16(2), 201-226. doi: Article.
- Weinberg, G. (2002). The aesthetics, history, and future challenges of interconnected music networks, *Proceedings of the 2002 Computer Music Conference*.
- Weinberg, G. (2005a). Local performance networks: musical interdependency through

- gestures and controllers, *Organised Sound*, 10(03), 255-265.
- Weinberg, G. (2005b). Interconnected musical networks: toward a theoretical framework, *Computer Music Journal*, 29(2), 23.
- Weinberger, D. (2002). *Small pieces loosely joined: a unified theory of the web*. Cambridge MA: Perseus.
- Weyland, T. (2008). Nitrotracker - a fasttracker ii style tracker for the nintendo ds. Retrieved February 25, 2008, from <http://nitrotracker.tobw.net/>.
- Williams, D., & Webster, P. (1999). *Experiencing music technology*. Schirmer Books New York.
- Wilson, S. (2002). *Information arts: intersections of art, science, and technology*, 945. Cambridge, Mass: MIT Press.
- Wright, M., & Freed, A. (1997). Open sound control: a new protocol for communicating with sound synthesizers, *Proceedings of the 1997 International Computer Music Conference*, 101-104.
- Yager, S. E., Kappelman, L. A., Maples, G. A., & Prybutok, V. R. (1997). Microcomputer playfulness: stable or dynamic trait?, *ACM SIGMIS Database*, 28(2), 43-52.
- Yamagishi, S., & Setoh, K. (1998). Variations for www: network music by max and the www. proc. of the int. computer music conf, *Proceedings of the International Computer Music Conference*, 510–13.
- Young, J. P. (2002). Networked music: bridging real and virtual space, *Organised Sound*, 6(02), 107-110.

Zhang, Y., & Candy, L. (2007). An in-depth case study of art-technology collaboration, *Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*, 53-62.