Puppeteering: Media control data as material for audiovisual creative innovation

Ilias Bergström
No current academic affiliation
ilias.bergstrom@gmail.com

Andre Holzapfel
KTH Royal Institute of Technology
holzap@kth.se

ABSTRACT

Millions of creative practitioners use advanced software to manipulate a large diversity of digital materials. Their work forms a significant part of the digital economy, where constant innovation is crucial. But existing tools can impede innovation by limiting digital materials to fixed media, e.g. musical notes, colours, coordinates or velocity. With existing practice, programming is necessary to transcend such restrictions, which limits the agency of most practitioners to shape their technologies. This research introduces the concept of elevating media control signals to be explicitly used as a material that end users directly manipulate in their tools without requiring that they program.

The work is grounded in the observation that tools across contexts and media share design patterns, and that those sharing many define a genre. "Puppeteering" involves two roles: first, a rich user interface application (Puppeteer) implementing a genre's defining patterns, but no media-specific functionality. The second role is taken by media-specific tools (Puppets), remote-controlled by the puppeteer using standardised control data messages (Strings). The concept empowers craftworkers to manipulate and interrelate digital materials without programming. We demonstrate puppeteering through an implementation for time-based media software, with the purpose of opening up for very rich possibilities of future research.

Author Keywords

Audio, Visual, Audiovisual, control data, control signals, materiality

CCS Concepts

• **Applied computing → Sound and music computing**; Performing arts;

1. INTRODUCTION

Computer use is today ubiquitous in most contexts of skilled craftwork [1]. Digital craftworkers include artists, designers, engineers, scientists, and others. This research concerns the software tools they employ, for "building and editing complex digital artefacts" [2].

Crucial for the knowledge economy is innovation, with important advancements frequently being the outcome of User Innovation [3]. But tools reach built-in limitations that impede user innovation: Across contexts, they constrain the manipulation of their digital material to fixed media. Music making, video editing, and so on, each have their own tools for their respective media. Sharing data and tools between contexts and media is hard, meaning users are kept from using these in ways their makers did not predict, unless they take to extensively modifying these tools.

End-User Development (EUD) comprises the methods, techniques and tools that allow users of software systems, to create, modify, or extend a software artefact [4]. Tailoring features allow for increased flexibility, but to transcend limitations to specific media, tailoring does not suffice. The remaining recourse of programming is both difficult to learn [5], and cognitively taxing to do [6], [7]. This present paper addresses the above issues, by extending the reach of the existing expertise digital craftworkers have of their tools.

2nd Audiovisual Symposium 2024, December 6, 2024, Dalarna Audiovisual Academy (DAVA), Dalarna University, Falun, Sweden.

Direct manipulation software tools have largely co-evolved, with technical and usage characteristics migrating between them. Users proficient with several tools, are therefore aware of common patterns between these. The reason is today's ubiquity of Design Patterns for software [8] and user interfaces [9]: general repeatable solutions to commonly occurring design problems. Modern software is constructed through combining several interrelating sets of patterns, in support of Patterns of Use [10, p. 29] and of User Experience [11] .

A software Genre is defined as a set of software with many common design patterns [12, p. 52]. The genre targeted here is that of Time-Based Media Software, which encompasses tools for sound, music, animation, video editing, show-control, etc. There, craftworkers rarely directly manipulate their raw digital material, i.e. the sensor data, audio, images, or video. What they interact with is digital control data [13], subsequently used for rendering the raw media output. Control data messages are also used in interconnecting tools.

Building on the above, this research introduces Puppeteering: the conception of direct manipulation software tools, which allow working with the digital material of control data without media-specific restrictions, by implementing a genre's defining patterns in a Puppeteer, and all media-specific features in Puppets. Puppeteering is here specifically presented for the genre of time-based media software. This extended abstract serves only as an introduction of the basic concepts underlying a larger research project that can comprise a multitude of contributions to a range of genres, beyond time-based media software.

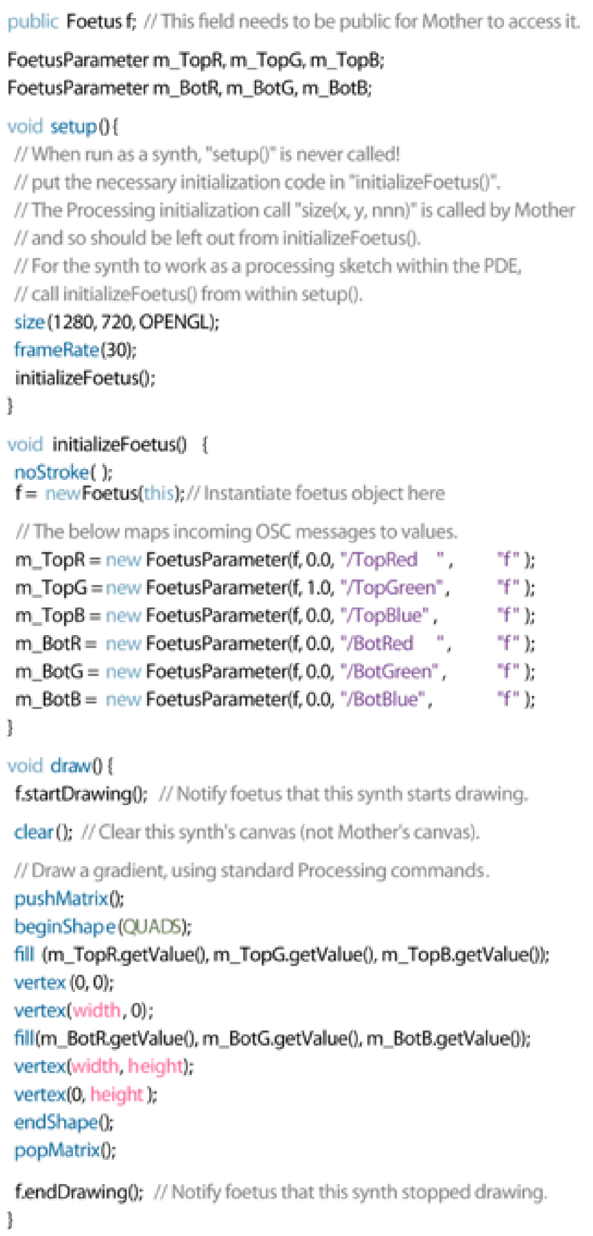
2. Concrete Implementation

There currently exists a mature implementation of the Puppeteering concept, in the form of the "TWO" application (puppeteer), and the "Mother" visual synthesis software (puppet). To see a demonstration of these in use, please refer to the videos on YouTube [14].

2.1 The Puppet: Mother

Mother is a software application for the performance of real-time visuals [15], [16]. It provides only the bare-minimum necessary functionality. Mother serves as a host for a tree-structure model of visual-synthesis plug-ins, each of which draws graphics on the screen, and/or modifies what has been drawn before it. Each synth plug-in is a small program written in Processing, a programming language intended to be used by artists [17] (Figure 1). Users can either make such synths, or download and adapt shared ones. See (Figure 2) for an example of the layering of several synths. When Mother is started, all that appears is a blank display window - it is controlled exclusively using remote control data, in the form of Open sound Control (OSC) [13] messages. Mother is less than 3000 lines of code: a small project, within the reach of an end-user programmer.

While it is deprecated at the time of writing, now that more capable applications have emerged (Notch.one, Unreal Engine, Unity, VVVV, etc.), we nonetheless use it to introduce the concept, because it has featured in several earlier academic publications, and is quick to explain.



```
public Foetus f; // This field needs to be public for Mother to access it.

FoetusParameter m_TopR, m_TopG, m_TopB;
FoetusParameter m_BotR, m_BotG, m_BotB;

void setup() {
 // When run as a synth, "setup()" is never called!
 // put the necessary initialization code in "initializeFoetus()".
 // The Processing initialization call "size(x, y, nnn)" is called by Mother
 // and so should be left out from initializeFoetus().
 // For the synth to work as a processing sketch within the PDE,
 // call initializeFoetus() from within setup().
 size(1280, 720, OPENGL);
 frameRate(30);
 initializeFoetus();
}

void initializeFoetus()  {
 noStroke( );
 f = newFoetus(this);// Instantiate foetus object here

 // The below maps incoming OSC messages to values.
 m_TopR = new FoetusParameter(f, 0.0, "/TopRed   ",      "f" );
 m_TopG = new FoetusParameter(f, 1.0, "/TopGreen",       "f" );
 m_TopB = new FoetusParameter(f, 0.0, "/TopBlue",        "f" );
 m_BotR =  new FoetusParameter(f, 0.0, "/BotRed   ",     "f" );
 m_BotG =  new FoetusParameter(f, 0.0, "/BotGreen",      "f" );
 m_BotB =  new FoetusParameter(f, 0.0, "/BotBlue",       "f" );
}

void draw() {
 f.startDrawing();  // Notify foetus that this synth starts drawing.

 clear();  // Clear this synth's canvas (not Mother's canvas).

 // Draw a gradient, using standard Processing commands.
 pushMatrix();
 beginShape(QUADS);
 fill (m_TopR.getValue(), m_TopG.getValue(), m_TopB.getValue());
 vertex (0, 0);
 vertex(width, 0);
 fill(m_BotR.getValue(), m_BotG.getValue(), m_BotB.getValue());
 vertex(width, height);
 vertex(0, height );
 endShape();
 popMatrix();

 f.endDrawing();  // Notify foetus that this synth stopped drawing.
}
```

Figure 1 - Minimal Processing sketch example

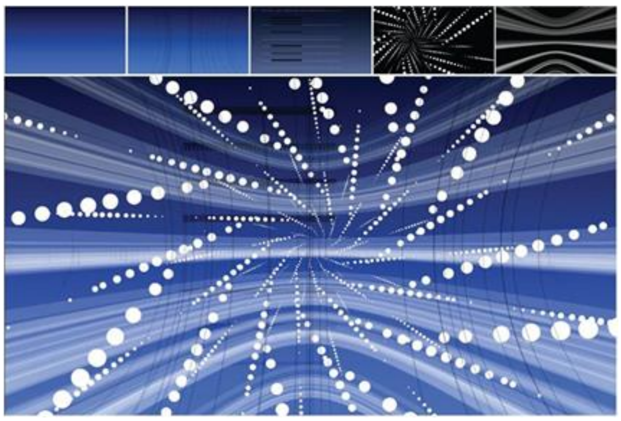


Figure 2 - Illustration of how several processing sketches (top) are within Mother layered to produce a single complex output

2.2 The Strings: OSC

OSC is central to making this work possible. It is a general control data message protocol (Figure 3), intended for innovating electronic musical instruments [13]. It has since received widespread adoption also in robotics, show control, Internet of Things, etc. [18]. The great advantage of OSC is that while there is a per-message schema, OSC has no overall fixed schema to define or restrict the set of possible messages, as is the case with legacy control data (e.g. MIDI, DMX). A second advantage is that older protocols can be translated to and from OSC data with relative ease. Third, OSC messages are self-descriptive: just by looking at a message, a user can tell what it is for (Figure 4). So messages can be received, manipulated and transmitted by software that is unaware of the media-specific intent, if any.

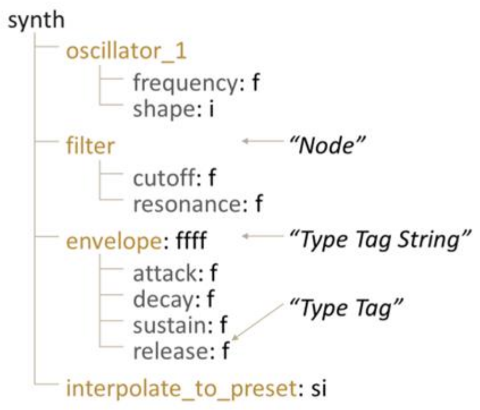
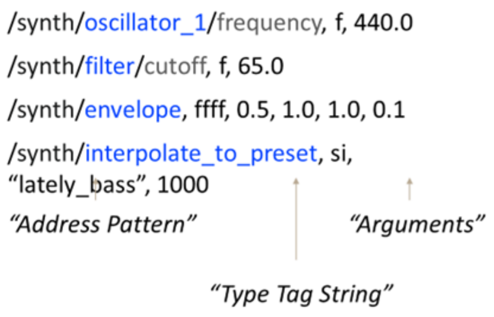


Figure 3 - OSC namespace



/synth/oscillator_1/frequency, f, 440.0

/synth/filter/cutoff, f, 65.0

/synth/envelope, ffff, 0.5, 1.0, 1.0, 0.1

/synth/interpolate_to_preset, si, "lately_bass", 1000

Figure 4 - OSC messages for the namespace

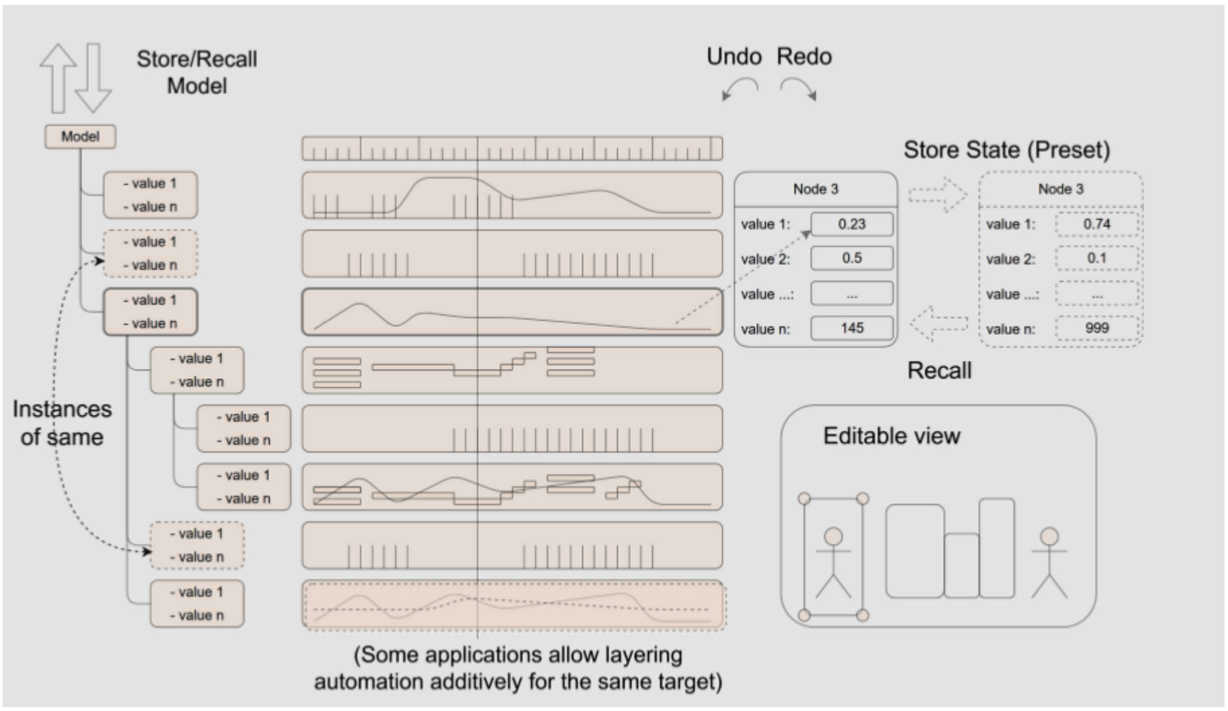


Figure 5 - UI design patterns for time-based media software

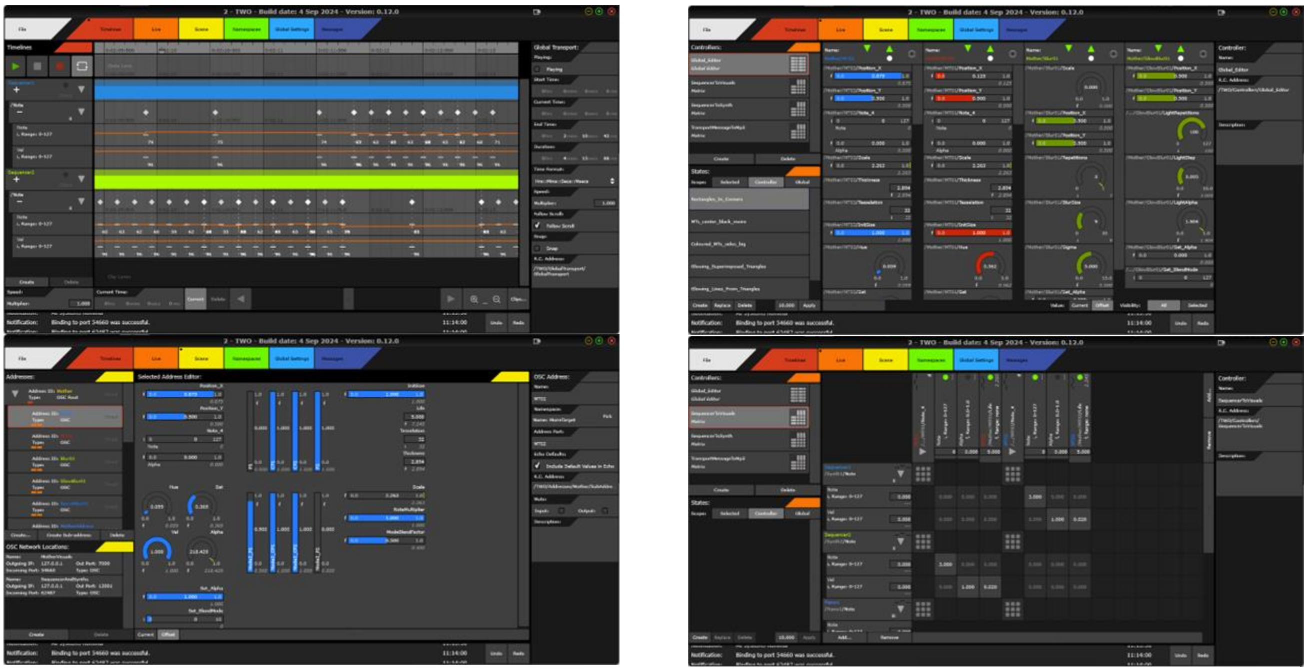
2.3 The Puppeteer: TWO

TWO provides a range of functionalities, which together make Puppeteering possible for time-based media software [19], [20]. In TWO, users manipulate model definitions, and control data. Users directly work with OSC-like control signals. TWO is thus not limited to any one kind of media. Any sender or recipient of OSC can interact with it. Other control-data protocols (MIDI, DMX), are translated to/from OSC.

Signals need only make sense to the user(s), and are recognized and associated by user(s) to the particular context only through the human readable OSC address-pattern (see Figure 3).

With TWO, a user can fully remote-control instances of Mother, other Puppets, and/or any other compatible software. When combined, Mother and TWO are not too dissimilar in use to a software application where UI and core are integrated: A user can edit the model tree (which in Mother is reflected as adding, removing and rearranging synths), manipulate the properties of model nodes (adjusting synth parameters), sequence changes of many properties over time (animating the graphics), read and write files with model and state descriptions, have external devices connected for it to be remote-controlled, etc (see Figure 5 for the patterns common in time-based media software, and Figure 6 for TWO's GUI implementing these).

As the user manipulates model and control data in TWO, all changes are immediately reflected in Mother's visual output, like in any other interactive GUI application. Nonetheless, neither Mother nor TWO is explicitly aware of the other's existence, and while they work together seamlessly, they were not exclusively made for each other.



Notification: Binding to port 14660 was successful.

Figure 6 - The TWO GUI, Illustrating the patterns implemented

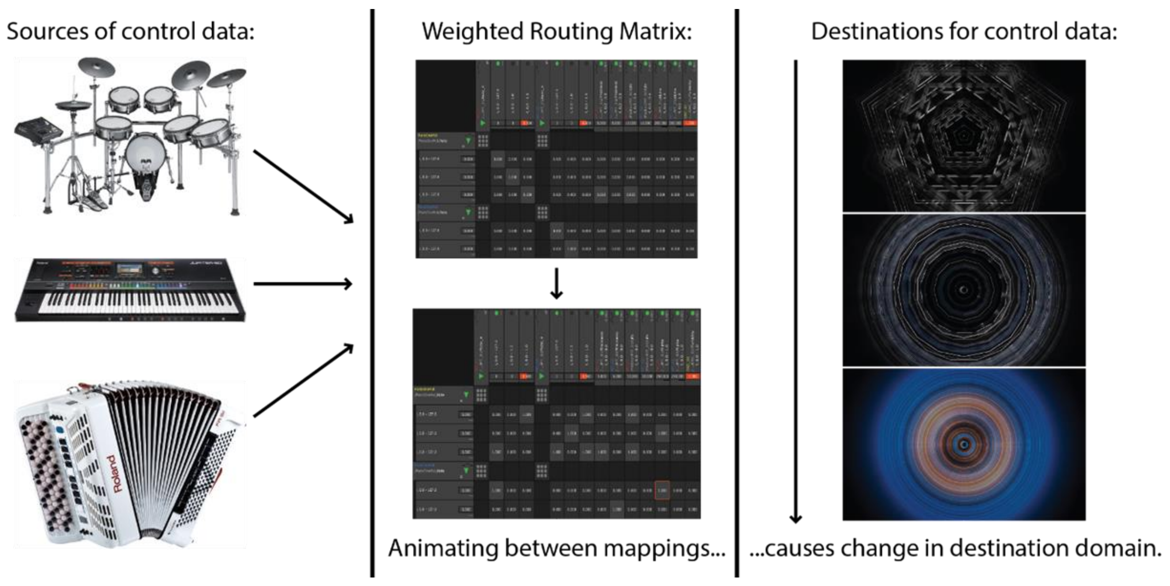

Sources of control data:
Weighted Routing Matrix:
Destinations for control data:
Animating between mappings...
...causes change in destination domain.

Figure 7 – Illustration of possible Mutable Mapping use

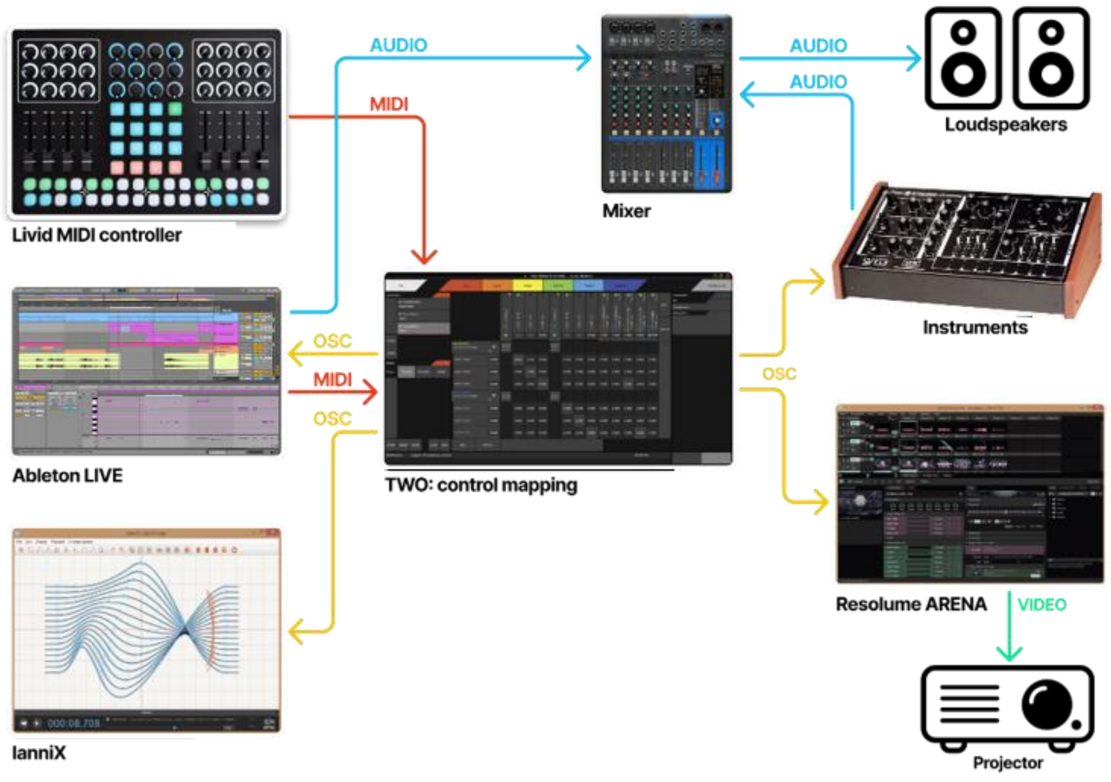


Figure 8 – More complex example of routing and mapping

Making any or all of these connections between namespaces is what is referred to as Mappings [21]. Mapping is prevalent in digital media, even if it is not always explicit to users. Unlike the physical, fixed connections between the keys, hammers and strings of a piano, the connections between a digital keyboard instrument's keys, knobs and sliders, to the sound generation circuitry, are all mappings, and only one out of nearly infinite possible alternatives. Through TWO's Weighted Routing Matrix (Figure 6, bottom right), such mappings can be defined, altered, stored, recalled and shared by end-users. Mutable Mapping [19], [20], being able to gradually alter mappings over time, plays a central role in Puppeteering. Such manipulation is crucial in fostering experimentation, and also a form of live performance. The weighted matrix has a weight value in each cell. Each row is a source, and for each column, a non-zero value cell means the incoming value is multiplied with the weight, added with the other column values, and sent to the column's destination. For an illustration, see Figure 7 and 8.

The flexibility of OSC presents new challenges. Since there is no fixed set of messages, each participating server needs to know what messages the servers it intends to communicate with react to. And each of the messages a server generates, needs to be mapped so that they correspond to the messages the recipients expect. Schemata for describing OSC namespace, and server states, have been introduced [22]. Together these allow servers to know each other's namespaces, without modification to the original OSC specification. They also allow storing and recalling to/from a file, the states of complex sets of servers, and sharing of such states between applications. OSCQuery further allows the automatic runtime discovery of schemata and states [23].

3. CONCLUSIONS AND FUTURE WORK

Computing has quickly gone, from being the restricted domain of a handful of scientists and government agents, to being ubiquitous in the lives and work of large parts of the population. This progress is accelerating at an exceptional rate, and there is no end to the shortage of skilled programmers, AI notwithstanding.

This research aims to spur innovation. Given these tools and practices, the possibility is opened up for end-users to innovate their own use-cases, giving independence from how manufacturers envision that their products should be used. And, while the tools enabling this experimentation may not be taken up by non-craftworkers, the use-cases that result are likely to be picked up by industry, and make their way into consumer markets [3].

Puppeteering seems to be the logical next step in user interface design, for the context of advanced users of time-based media software, but possibly also other application areas. Historically, a pattern can be observed:

1. A task is first tackled using low-level programming.
2. Then libraries appear which encapsulate some of the complexity, making the task easier to program for.
3. Then, either scripting languages, or media-specific programming languages may appear.
4. Only after several years would a dedicated piece of software have crystallised, which allows the task to be carried out without requiring programming.

Maybe abstract model and control data manipulation, will too in the future be established as something end-users regularly do without second thought?

4. ETHICAL STANDARDS

This research has been to significant extent been carried out without institutional funding, by the article's first author. There are no conflicts of interest to note.

5. REFERENCES

[1] M. McCullough, *Abstracting craft: The practiced digital hand*. MIT press, 1998.

[2] M. Duignan, "Computer mediated music production: A study of abstraction and activity," Doctoral Thesis, Victoria University of Wellington, New Zealand, 2008.

[3] E. Von Hippel, "Democratizing innovation: The evolving phenomenon of user innovation," *Journal für Betriebswirtschaft*, vol. 55, pp. 63–78, 2005.

[4] H. Lieberman, F. Paternò, M. Klann, and V. Wulf, "End-User Development: An Emerging Paradigm," in *End User Development*, H. Lieberman, F. Paternò, and V. Wulf, Eds., in Human-Computer Interaction Series, no. 9. , Springer Netherlands, 2006, pp. 1–8. Accessed: May 25, 2015. [Online]. Available: http://link.springer.com/chapter/10.1007/1-4020-5386-X_1

[5] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming," *AcM SIGcSE Bulletin*, vol. 39, no. 2, pp. 32–36, 2007.

[6] A. F. Blackwell, "What is programming?," presented at the PPIG, Citeseer, 2002, pp. 204–218.

[7] J. Spolsky and J. Spolsky, "Human task switches considered harmful," *Joel on Software: And on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to Those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity*, pp. 179–182, 2004.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns : elements of reusable object-oriented software*, 1st ed. Addison Wesley, 1994.

[9] J. Tidwell, *Designing interfaces*. O'Reilly Media, Inc., 2010.

[10] T. Schummer and S. Lukosch, *Patterns for computer-mediated interaction*. John Wiley & Sons, 2013.

[11] A. F. Blackwell and S. Fincher, "PUX: patterns of user experience," *Interactions*, vol. 17, no. 2, pp. 27–31, 2010.

[12] Å. Walldius, *Patterns of recollection: The documentary meets digital media*. Aura förlag, 2001.

[13] M. Wright, A. Freed, and A. Momeni, "OpenSound Control: state of the art 2003," in *Proceedings of the 2003 conference on New interfaces for musical expression*, 2003, pp. 153–160.

[14] I. Bergström, "TWO: Media Control Workstation." [Online]. Available: https://www.youtube.com/@TwoMediaControlWorkstation

[15] I. Bergstrom and B. Lotto, "Mother: Making the Performance of Real-Time Computer Graphics Accessible to Non-programmers," in *(re)Actor3: The Third International Conference on Digital Live Art Proceedings*, Sep. 2008, pp. 11–12.

[16] I. Bergstrom and B. Lotto, "Code-bending: a new creative coding practice," *Leonardo*, no. In Press, 2014.

[17] C. Reas and B. Fry, *Processing: A Programming Handbook for Visual Designers and Artists*. The MIT Press, 2007.

[18] A. Freed and A. Schmeder, "Features and future of open sound control version 1.1 for nime," in *NIME'09: Proceedings of the 9th Conference on New Interfaces for Musical Expression*, 2009.

[19] I. Bergstrom, A. Steed, and B. Lotto, "Mutable Mapping: gradual re-routing of OSC control data as a form of artistic performance," in *Proceedings of the international conference on Advances in computer entertainment technology*, Athens, Greece, Oct. 2009, pp. 290–293.

[20] I. Bergstrom, "Soma: live performance where congruent musical, visual, and proprioceptive stimuli fuse to form a combined aesthetic narrative," Thesis, 2011. Accessed: Oct. 10, 2011. [Online]. Available: http://discovery.ucl.ac.uk/1310143/

[21] E. R. Miranda and M. M. Wanderley, *New Digital Musical Instruments: Control And Interaction Beyond the Keyboard*. AR Editions, 2006.

[22] I. Bergstrom and J. Llobera, "OSC-Namespace and OSC-State: Schemata for Describing the Namespace and State of OSC-Enabled Systems," in *2014*, London, 2014. Accessed: Feb. 16, 2015. [Online]. Available: http://www.nime.org/proceedings/2014/nime2014_300.pdf

[23] "OSCQuery," OSCQuery. [Online]. Available: https://github.com/Vidvox/OSCQueryProposal