Rapid Creation and Publication of Digital Musical Instruments

Charles Roberts, Matthew Wright, JoAnn Kuchera-Morin, Tobias Höllerer
Media Arts & Technology Program
University of California at Santa Barbara
charlie@charlie-roberts.com, matt@create.ucsb.edu, jkm@create.ucsb.edu, holl@cs.ucsb.edu

ABSTRACT

We describe research enabling the rapid creation of digital musical instruments and their publication to the Internet. This research comprises both high-level abstractions for making continuous mappings between audio, interactive, and graphical elements, as well as a centralized database for storing and accessing instruments. Published instruments run in most devices capable of running a modern web browser. Notation of instrument design is optimized for readability and expressivity.

Keywords

web browser, javascript, publication, digital audio

1. INTRODUCTION

Recent augmentations to the browser and its ubiquity on mobile devices have made it more attractive than ever for audience participation pieces, digital musical instrument design, and artistic installations [12, 8, 19, 2, 16]. Taking advantage of new browser features requires knowledge of the three primary technologies used in web development: HTML, CSS, and JavaScript. Previous work has explored minimizing the difficulties of learning and using these technologies together when designing digital musical instruments [14, 4]; our work continues in this vein by enabling the authoring of a complete digital musical instrument for the browser (including interface, synthesis, and mappings) in a single line of code. It also simplifies the creation of works that can be easily accessed by others, via a centralized system for storing, delivering and iterating digital musical instruments.

Our research augments *Gibber*, a browser-based creative coding environment.

2. BACKGROUND

We briefly review and discuss the state of the art for audio generation from within a web browser and designing interaction in live coding environments.

2.1 Audio in the Browser

The potential of the browser as a computer music platform [20] includes several promising affordances, including incorporation of a high-level language capable of sample-accurate synthesis (JavaScript), a sophisticated layout system for visual content (CSS + HTML), access to various sensors for interactivity, and the ubiquity afforded by a web-based delivery mechanism.

Although computer music in a web browser has been possible for many years via extensions such as Flash and Java applets [3], recent changes have greatly improved the potential. One important change is the ubiquity of browsers on mobile devices that are roughly equivalent to their desktop counterparts. These browsers typically have access to the variety of sensors found on mobile devices (such as touchscreens, accelerometers, and gyroscopes). A second important change is the addition of the Web Audio API[1]. Browser implementations of the Web Audio API provide both a high-level syntax for assembling graphs of low-level C++ unit generators, and a ScriptProcessorNode that enables users to define complete audio systems entirely in JavaScript.

Although there are concerns about the utility of the ScriptProcessorNode for computer music applications due to latency and poor efficiency as compared to native C++ nodes provided by the Web Audio API [20], many dedicated libraries[2,3,4] for creating music in the browser favor it as it provides extensibility and single-sample audio processing. This enables a wide variety of features that are not possible using the block-rate, pre-defined, C++ unit generators built into the browser, such as single-sample feedback loops between unit generators and audio-rate modulation of scheduling. For our research, we use the library *Gibberish.js* [14] , which performs almost all of its signal processing inside of a single ScriptProcessorNode.

2.2 Live Coding Instruments and Interfaces

The rapid creation of digital musical instruments has been a growing research interest, especially over the past five years. Much of this interest has involved the development of instruments to run on mobile devices using dataflow environments. In the SpeedDial project [5], Essl designed a system for smartphones that enabled users to use the physical numeric keypads on devices to quickly connect control sources, signal processing algorithms, and audio generators. Essl's research continued with *UrMus* [6], a Lua framework for manipulating low-level C++ unit generators on mobile devices. The UrMus application featured a touchscreen interface for mapping sensors to synthesis that improves upon the ease of mapping found in SpeedDial. It also enables developers to create complex instruments with custom graphic user interfaces and synthesis algorithms. Similar to

[1]The Audio Data API, an earlier system for realtime synthesis that ran in Firefox, has been deprecated in favor of the Web Audio API

[2]http://mohayonao.github.io/timbre.js/

[3]https://github.com/jussi-kalliokoski/audiolib.js/

[4]https://github.com/colinbdclark/flocking

NIME'14, June 30 – July 03, 2014, Goldsmiths, University of London, UK.

the research described in Section 3, UrMus provides a sophisticated mechanism for managing multirate, normalized dataflow [7]. We believe the research described in this paper provides a simpler syntax and mental model for the exploration of multimodal mappings, at the potential expense of flexibility.

The SenSynth project also explores this line of research for the Nokia MeeGo operating system. Every parameter available to synthesis algorithms is given an optional control that can be overridden to accept sensor input from the various multimodal sensors available on the device, including the camera, microphone, accelerometer, and magnetometer [10].

Research on the topic of rapid interface construction and mapping has also been performed by the live coding community. ChucK provides abstractions for dynamically mapping MIDI and OSC messages [18] while its *miniAudicle* IDE [15] enables end-user programmers to construct graphical user interfaces. The WAXX library [4], inspired by ChucK, provides a live coding playground that enables GUI construction and mapping in the context of the web browser. The *Improcess* project [1] researched live coding performances using the *Overtone* framework in conjunction with the monome grid controller.[5]

This line of research reached its perhaps inevitable conclusion in the recent work of Lee and Essl [9], who gave a performance in which a tablet-based instrument was live coded in front of an audience by two programmers while a third performer employed it to make music. The laptop screens of both programmers and the tablet interface used by the performer were all projected for audience members to see. Marije Baalman has also explored this line of research as both programmer and performer in the same performance, alternating between using movement to generate gestural control signals during performance and live coding mappings to sound synthesis.[6]

Our research seeks to abstract the process of interface construction and mapping to sonic parameters. We enable instrument creation that would require a dozen lines of code in many of the environments described above to be completed in a single line using a powerful mapping abstraction that is conceptually simple for end-users and terse enough to afford rapid creative exploration and/or live coding performance opportunities.

3. MAPPING ABSTRACTIONS IN GIBBER

Gibber imports interface and synthesis libraries and wraps them to improve their syntax for live coding. Descriptions of the interface elements and unit generators provided by these libraries can be found in [14]. While the libraries that Gibber uses provide a large number of interface elements, Gibber also has a 2D drawing API and event handlers for touch, mouse, and keyboard events, enabling the creation of nontraditional interfaces. In addition to shortening the syntax used to instantiate objects from these libraries, Gibber provides metadata about their properties, including the following:

- The **timescale** at which the property operates: audio, graphical, or interactive
- The **range** of values most likely to be assigned to the property
- A Javascript **expression** embodying a simple approximation of how changes to the property are perceived, for example, linearly or logarithmically.
- The **number of dimensions** of the property. For audio most properties are one-dimensional, but many interface elements have multiple dimensions.

These metadata are used to tersely create continuous mappings between properties of objects, including objects of different modalities. Of primary interest to the NIME community are mappings from interactive widgets to audio objects, but the same strategies also apply to creating mappings between graphical and audio objects, or between a pair of objects of the same modality. Our abstraction simplifies the creation of time-varying mappings between any two properties and is indicated by simply capitalizing the righthand property value; capitalized property names thus bear similiarities to the concept of signals in functional reactive programming [17]. For example, consider the following two lines of code:

```
sineA.frequency = slider1.value
sineB.frequency = slider2.Value
```

In the first line of code, the value of `slider1` determines the frequency of `sineA` only at the moment when the line of code is executed. With the second line of code, the use of the capitalized `Value` property creates a continuous mapping and starts a series of actions.

First, by examining the metadata of the two properties, Gibber notes that they differ in their timescales (one is audio, the other interactive); Gibber adds a one-pole filter to the output of the slider to smooth the values it creates and avoid quantization or "zippering" effects due to the differing sampling rates of the two signals. If the mapping were made in reverse (the value of the slider tracking the frequency of the oscillator) an envelope follower would be placed on the oscillator's frequency so that the slider displays a running average as the oscillator frequency is sequenced and modulated; this type of mapping is often performed in Gibber to synchronize graphical elements, such as properties of 3D geometries or shader uniforms, to audio properties.

Next, Gibber adds an implicit `Map` unit generator to the audio graph to evaluate the JavaScript expressions that map from the linear output scale of the slider to the logarithmic scale of the `frequency` property, as well as to scale the range of the slider (0–1) to the expected range of the oscillator's `frequency` property (50–3200 Hz is the default).

Although the metadata for each property contain a default range of expected values, this range can easily be modified; continuous mappings can also be applied to dynamically change the range of a mapping. Mappings can also easily be inverted, and sequencing such inversions can yield interesting musical results.

```
synth.frequency = Accelerometer.X
synth.Frequency.min = 70
synth.Frequency.max = Accelerometer.Y
Seq( synth.Frequency.invert, 1/2 )
```

Note that we change the input range for the `frequency` property, not the output range of the accelerometer's x-axis. This allows the same value to be continuously mapped to multiple properties with customized ranges for each.

As a final step in our mapping abstraction, we attempt to provide an intelligent label to interface elements (assuming it makes sense given an element's appearance) identifying both the types of objects and the names of properties the element is mapped to, as shown in Figure 1. Elements can be continuously mapped to unlimited object/property pairs.

[5] http://monome.org/devices
[6] https://vimeo.com/80685325

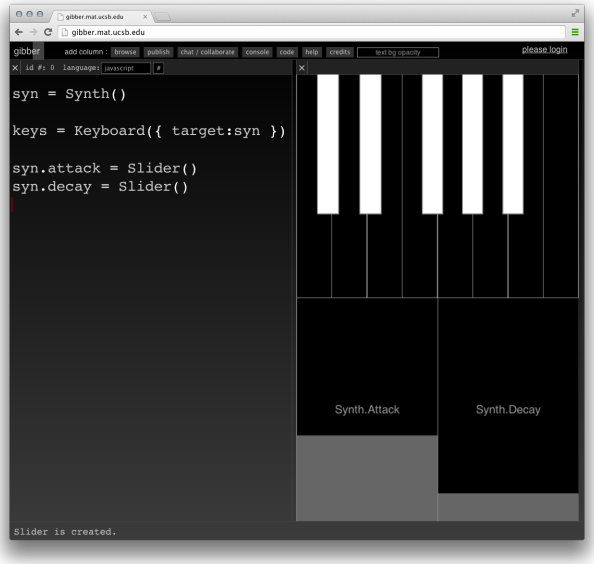

Figure 1: A keyboard and two sliders mapped to control a synthesizer inside of Gibber

4. RAPID INTERFACES: ONE-LINE NIMES

GUIs in Gibber are created using strategies first explored in [13] with the open-source mobile application Control [11]. They are presented using Interface.js, which affords audiovisual and gestural control systems that run in the browser. By default, interface elements are positioned on screen using a simple subdivision algorithm but instrument designers are also free to manually define boundary boxes for elements.

Abstractions present in Gibber enable interfaces running in remote browsers to be treated almost identically to interfaces running from within Gibber itself; this lets programmers easily prototype interfaces for mobile devices. Remote interfaces are managed using the Interface.Server application[7], which serves web pages to clients that can generate OSC, MIDI, or WebSocket messages. By affording manipulation of interfaces running on remote devices, we enable programmers to dynamically experiment with mappings without having to republish / reload instruments.

Gibber provides syntactic sugar to tersely create and map an anonymous interface element to a single audiovisual parameter. This affords the creation of "One-Line NIMEs", such as the following:

```
Sine( Slider(), Slider() )
```

This line of code will create two sliders that are passed as arguments to the sine oscillator constructor. These are mapped automatically to frequency and amplitude, the two arguments the `Sine` constructor accepts. As described in Section 3, the sliders are also labelled with the respective object and property they control.

4.1 A Comparison of Notations

As mentioned in Section 2.2, UrMus offers similar functionality for creating normalized, multimodal mappings. However, the notations used in UrMus and Gibber are very different; here is a comparison of the code required to create a sine oscillator with frequency and amplitude determined by an accelerometer.

UrMus:

```
mySinOsc = FlowBox(FBSinOsc)
FBAccel.X:SetPush(mySinOsc.Freq)
FBAccel.Y:SetPush(mySinOsc.Amp)
FBDac.In:SetPull(mySinOsc.Out)
```

Gibber:

```
Sine({ frequency:Accel.X, amp:Accel.Y })
```

The abstractions in Gibber enable completing this task in one line of code. More important than the amount of code is the simple mental model: to make a time-dependent assignment from one object to another, use regular assignment syntax but capitalize the righthand property value. UrMus offers a more flexible system for defining multirate mappings, including push/pull semantics and a variety of signal conditioning algorithms; this arguably necessitates the use of a more complex notation. Gibber instead focuses on terseness, readability, and low viscosity. Below is an example where the output envelope of a drum loop controls both its pitch and the modulation index of FM synthesis, while a slider is created to control the range of both mappings.

```
drums = Drums('x*o*x*o-')
// assign the output envelope of drums
// to control speed of sample playback
drums.pitch = drums.Out

// assign output envelope of drums to
// control modulation index of FM synthesis
fm = FM({ index:drums.Out })

// create slider that defines max boundaries
slider = Slider()
drums.Pitch.max = slider
fm.Index.max = slider
```

5. PUBLICATION OF DIGITAL MUSICAL INSTRUMENTS

Gibber features a centralized server that enables users to create accounts, engage in collaborative programming sessions, and publish compositions and instruments. Users can tag the sketches they publish and provide notes on their use. After publication, a URL is automatically generated and displayed that points to the sketch and can be freely shared. By default, visiting the provided URL will launch Gibber, load the associated published code, and display it for editing and execution.

We provide an option during the publication process to "Publish as an instrument" that is disabled by default. When the option is checked the URL generated upon publication will launch the instrument interface fullscreen; all UI elements associated with the coding environment are hidden. A comparison of Figure 1 and Figure 2 shows the differences between the two publication modes.

After logging in to Gibber, users see all sketches they have published in the file browser of the development environment. They can also see recent sketches by other users, and learn from tutorials on instrument building that specifically discuss the process of mapping interactive control to audiovisual parameters. We imagine a community of builders creating instruments and sharing them freely with one another without ever needing to download and install software to run them.

6. CONCLUSIONS

The browser affords extraordinary opportunities for authoring and disseminating digital musical instruments. Our research provides high-level abstractions and notations that

[7] https://github.com/charlieroberts/interface.server

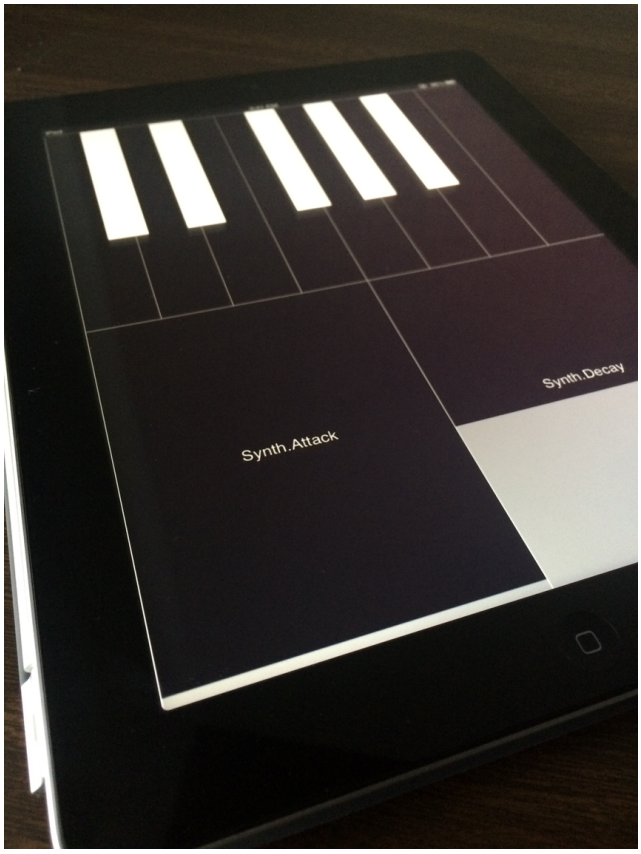

Figure 2: A published instrument as designed in Figure 1, running on a tablet

enable simple instruments to be created in the browser with a single line of code; it also affords rapid experimentation and prototyping of instrument design. The server back-end of Gibber enables users to save their instruments to a centralized location so that they can be shared with (and potentially improved by) other users.

7. ACKNOWLEDGMENTS

We gratefully acknowledge the support of a fellowship from the Robert W. Deutsch foundation, and external funding from the National Science Foundation under grants #0821858, #0855279, and #1047678. Thanks also to Dr. Georg Essl for providing an up-to-date example of the mapping notation in UrMus.

8. REFERENCES

[1] S. Aaron, A. F. Blackwell, R. Hoadley, and T. Regan. A principled approach to developing new languages for live coding. In *Proceedings of New Interfaces for Musical Expression*, pages 381–386, 2011.

[2] J. Allison, Y. Oh, and B. Taylor. Nexus: Collaborative performance for the masses, handling instrument interface distribution through the web. *Proceedings of New Interfaces for Musical Expression*, 2013.

[3] P. Burk. Jammin' on the web–a new client/server architecture for multi-user musical performance. In *Proc. ICMC*, 2000.

[4] H. Choi and J. Berger. Waax: Web audio api extension. In *Proceedings of New Interfaces for Musical Expression*, pages 499–502, 2013.

[5] G. Essl. SpeedDial: Rapid and on-the-fly mapping of mobile phone instruments. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2009.

[6] G. Essl. UrMus: an environment for mobile instrument design and performance. Ann Arbor, MI: MPublishing, University of Michigan Library, 2010.

[7] G. Essl. Ursound–live patching of audio and multimedia using a multi-rate normed single-stream data-flow engine. Ann Arbor, MI: MPublishing, University of Michigan Library, 2010.

[8] A. Hindle. Swarmed: Captive portals, mobile devices, and audience participation in multi-user music performance. 2013.

[9] S. W. Lee and G. Essl. Live coding the mobile music instrument. *Proceedings of New Interfaces for Musical Expression*, 2013.

[10] R. McGee, D. Ashbrook, and S. White. SenSynth: a mobile application for dynamic sensor to sound mapping. 2012.

[11] C. Roberts. Control: Software for End-User Interface Programming and Interactive Performance. *Proceedings of the International Computer Music Conference*, 2011.

[12] C. Roberts and T. Höllerer. Composition for Conductor and Audience: New Uses for Mobile Devices in the Concert Hall. In *Proceedings of the 24th annual ACM symposium adjunct on User interface software and technology*, UIST '11 Adjunct, pages 65–66, New York, NY, USA, 2011. ACM.

[13] C. Roberts, G. Wakefield, and M. Wright. Mobile controls on-the-fly: An abstraction for distributed nimes. In *Proceedings of the 2012 Conference on New Interfaces for Musical Expression (NIME 2012)*, 2012.

[14] C. Roberts, G. Wakefield, and M. Wright. The web browser as synthesizer and interface. In *Proceedings of the 2013 Conference on New Interfaces for Musical Expression (NIME 2013)*, volume 2013, 2013.

[15] S. Salazar, G. Wang, and P. Cook. miniAudicle and ChucK Shell: New interfaces for ChucK development and performance. In *Proceedings of the 2006 International Computer Music Conference*, pages 63–66, 2006.

[16] S. Savage, N. E. Chavez, C. Toxtli, S. Medina, D. Álvarez-López, and T. Höllerer. A social crowd-controlled orchestra. In *Proceedings of the 2013 conference on Computer supported cooperative work companion*, pages 267–272. ACM, 2013.

[17] Z. Wan and P. Hudak. Functional reactive programming from first principles. In *ACM SIGPLAN Notices*, volume 35, pages 242–252. ACM, 2000.

[18] G. Wang, A. Misra, A. Kapur, and P. Cook. Yeah, ChucK it! => dynamic, controllable interface mapping. In *Proceedings of the 2005 conference on New interfaces for musical expression*, pages 196–199, 2005.

[19] N. Weitzner, J. Freeman, S. Garrett, and Y. Chen. massMobile–an audience participation framework. *Proceedings of the New Interfaces For Musical Expression Conference*, 2012.

[20] L. Wyse and S. Subramanian. The viability of the web browser as a computer music platform. *Computer Music Journal*, 37(4):10–23, 2013.