Mobile Controls On-The-Fly: An Abstraction for Distributed NIMEs

Charles Roberts
Media Arts and Technology
Program, UCSB
charlie@charlie-roberts.com

Graham Wakefield
Media Arts and Technology
Program, UCSB
wakefield@mat.ucsb.edu

Matthew Wright
Media Arts and Technology
Program, UCSB
matt@create.ucsb.edu

ABSTRACT

Designing mobile interfaces for computer-based musical performance is generally a time-consuming task that can be exasperating for performers. Instead of being able to experiment freely with physical interfaces' affordances, performers must spend time and attention on non-musical tasks including network configuration, development environments for the mobile devices, defining OSC address spaces, and handling the receipt of OSC in the environment that will control and produce sound. Our research seeks to overcome such obstacles by minimizing the code needed to both generate and read the output of interfaces on mobile devices. For iOS and Android devices, our implementation extends the application Control to use a simple set of OSC messages to define interfaces and automatically route output. On the desktop, our implementations in Max/MSP/Jitter, LuaAV, and SuperCollider allow users to create mobile widgets mapped to sonic parameters with a single line of code. We believe the fluidity of our approach will encourage users to incorporate mobile devices into their everyday performance practice.

Keywords

NIME, OSC, Zeroconf, iOS, Android, Max/MSP/Jitter, LuaAV, SuperCollider, Mobile

1. INTRODUCTION

Ubiquitous mobile devices such as iOS and Android are plentiful, useful, and becoming expected/necessary input devices for NIMEs. In addition to the wealth of all-in-one applications that implement NIMEs exclusively on the mobile device hardware, there are now also many general-purpose software packages for creating various input interfaces on mobile devices and outputting OSC over wireless, including mrmr[1], TouchOSC[2], and Lemur[3]. Such "second-generation OSC implementations" [15] focus entirely on realtime gestural control, represent user input as OSC, and stream the results wirelessly. The pervasiveness and generality of OSC make it straightforward to build distributed NIMEs by connecting custom mobile device interfaces to custom mapping, control, and audio software.

[1] `http://mrmr.noisepages.com`
[2] `http://hexler.net/software/touchosc`
[3] `http://liine.net/en/products/lemur`

NIME'12, May 21 – 23, 2012, University of Michigan, Ann Arbor.

Our research radically simplifies and makes dynamic the workflow for developing such distributed NIMEs. Instead of forcing the user to think in terms of different customizable pieces of software connected by explicit messaging according to a custom OSC schema, we provide the abstraction of a single distributed system in which connecting widgets[4] on the mobile device to sound control parameters in the synthesis environment is no different than if the widgets existed within the synthesis environment itself. In particular, programming the entire distributed system occurs only in the synthesis environment. For example, as a beginning Max/MSP/Jitter programmer interactively develops a custom sound synthesis patch in an exploratory manner, the only apparent practical or conceptual difference between connecting Max sliders or iPad sliders to synthesis parameters is that they appear on different hardware (with different display and input affordances). Consequently, it also provides powerful mechanisms to dynamically create, remove, and modify the appearance and behavior of interfaces on the mobile device, opening possibilities for interfaces that change during the course of a performance under the direct or indirect control of performers and/or software processes.

1.1 Prior Work(flow)

Before, users of OSC always had to be explicitly aware of the sending and receiving software systems and of the messages between them, and program both sides according to the chosen OSC address space.

Many existing interface apps for mobile devices have an associated visual interface builder application. The process of creating a virtual slider and linking it to a prototyping environment is typically similar to the following (asterisks indicate a task that only needs to be completed once per interface as opposed to once per widget):

- Launch the interface builder app and create a new blank interface*
- Drag a widget into the interface
- Define an OSC address for the widget's output
- Transfer the interface to a mobile device*
- On the mobile device, select the newly transferred interface*
- On the mobile device, select a IP address and port number for output*

[4] Throughout this paper, a "widget" on a mobile device could take data from the device's accelerometer, microphone, compass, gyroscope, and other sensors as well as traditional screen-based GUI elements such as buttons, dials, and sliders.

- In the prototyping platform, create an object to receive OSC messages
- In the prototyping platform, define a callback for the specific OSC address this widget will output
- In the prototyping platform, define an object that sends OSC messages to the widget so that you can set its value from the prototyping environment

Four of these steps only need to be performed once, and five need to be performed for each widget added to the interface. Our solution reduces this to a single step per-interface and a single step per-widget.

Other researchers have also focused on allowing users to easily map interface elements. In the SpeedDial[2] project Georg Essl created a system to quickly map the interactive affordances of the Nokia N95 phone to synthesis parameters using a small number of button presses. This project shares our goal of making meaningful musical mappings possible via a minimal number of steps. Essl further continued this line of research with the default interface of his urMus project[3] which allows users to quickly map control parameters to synthesis algorithms using multitouch gestures. In both of Essl's projects device affordances are mapped to synthesis algorithms running on the phone itself; our research instead focuses on mobile devices controlling synthesis on remote computers. In addition to his experiments with mapping, Essl also described the benefits of using Zeroconf and OSC together in urMus to automate network connectivity[4].

The c74 app[6] creates interfaces on iOS devices whose layouts are dynamically generated from Max. However, interface design using c74 requires carefully scripted messages sent to a single *c74* Max object, and the values of the widgets must be separately routed from the *c74* object's outlet. Unlike our research, c74 is written for a specific desktop application and mobile platform.

Ge Wang et al. wrote about dynamic control mapping for ChucK[14]. ChucK can quickly patch together audio signal graphs that can be controlled by MIDI, HID and OSC data. The LiCK library for ChucK[5] includes classes that easily parse input from a variety of control sources, including both Control and TouchOSC. However, these classes are for accepting data from pre-defined interfaces and do not allow the creation of mobile interfaces from within ChucK itself.

Moving away from mobile devices, Julià, Gallardo and Jordà authored a framework easing the process of creating and integrating tabletop interfaces[7] with Pure Data (Pd). Their framework provides the coordinates of fingers on the tabletop surface via a simple pair of Pd objects and enables users to easily create interface objects on the tabletop surface in Pd. The Field prototyping environment[1] also influenced our research. In Field users can create graphical widgets that manipulate variables in textual code while programs are running. There is no need to write the complicated callbacks typically associated with asynchronous user input; instead, visual programming techniques are used to automatically assign the output of widgets to specific variables. We imagine the research outlined here being used with similar ease.

2. IMPLEMENTATION

Much of the research for this paper extends the project Control [9], written by the first author. Control allows users to define touchscreen interfaces for iOS and Android devices using web technologies such as HTML, CSS, JSON and JavaScript. Devices running the Control application can be automatically discovered on a network using the Zeroconf protocol.

[5] `https://github.com/heuermh/lick`

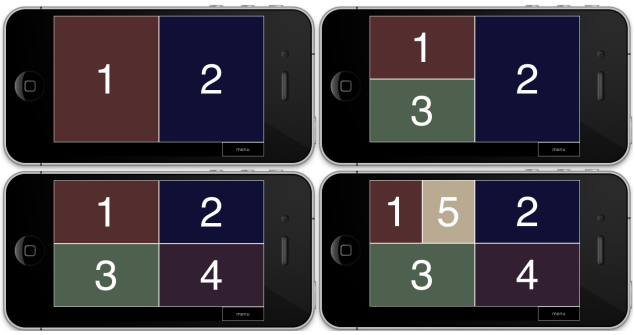

Figure 1: Automatically generated layouts after adding 2, 3, 4 and 5 widgets

In addition to extending Control we have also created libraries that manage the process of generating and utilizing interfaces on devices running Control for Max, LuaAV and SuperCollider. These can be thought of as proxy classes running in the synthesis environment that create an abstraction of the mobile device / network / personal computer system.

2.1 Dynamic interface creation on mobile device

We extended Control so that interfaces could be dynamically created using a simple OSC namespace:

- /control/createBlankInterface - creates a blank interface in portrait or landscape orientation
- /control/setDestination - set the IP address and port where Control will output OSC information
- /control/addWidget - accepts a string of JSON defining a widget to be placed in the interface
- /control/removeWidget - removes a named widget from the interface
- /control/runScript - execute any arbitrary JavaScript on device

There are also addresses to change parameters of existing widgets such as color, bounding box, and output range.

2.2 Automated GUI Layout

Although the above namespace is flexible enough to allow users to define interfaces remotely, it did not yield the desired type of fluid experimentation. One impediment was having to specify a bounding box for each widget: if widgets were placed incorrectly, users would have to send a separate OSC message to correct positioning. To alleviate this problem we incorporated an automatic layout manager. For example, Max users can copy and paste the same *control.slider* object five times and five sliders will appear on the mobile device in an auto-generated layout as shown in Figure 1.

Control's new AutoLayout object uses a simple subdivision scheme. When the first widget is added to an interface it fills it entirely. When a second widget is added, the first widget is cut in half and the second widget fills the other half of the interface. For every widget added thereafter, AutoLayout will find the existing widget occupying the largest area and then cut that widget in half to make room for the

new widget. Figure 1 illustrates this process as more and more widgets are added to the interface.

Several optional parameters provide flexibility in this scheme. A widget can be marked *sacrosanct*; such widgets will never be subdivided. Control interfaces are also divided into *pages* where a page is one screen of widgets. Users can optionally specify a particular page for a widget. Once there is more than one page in an interface buttons automatically appear to enable users to switch between pages.

2.3 Automated OSC Namespace Generation

We also automate the generation of an OSC namespace for widgets. In all three of our implementations, a master object assigns each widget a unique id number; this number is appended to the widget type to obtain an OSC address for the widget. Thus, the first button automatically outputs to /Button1, the second button outputs to /Button2 etc. Although these addresses do not document functionality, in practice users do not need to know the OSC destinations of widgets because the Max/MSP, LuaAV and SuperCollider implementations all handle creating the appropriate OSC responders behind the scenes and route the values received to a destination of the user's choice.

Each implementation also provides a method for users to assign a specific output address to a widget. The output address for any widget can be changed on-the-fly; for example, the output address of a slider could be dynamically re-assigned whenever a user changes the synthesis algorithm they are controlling.

2.4 Max/MSP Implementation

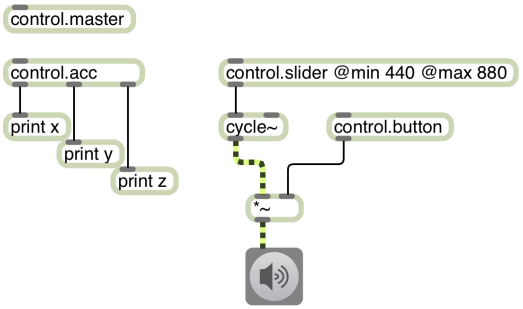

Figure 2: Max patch showing accelerometer output and two widgets controlling the frequency and muting of an oscillator

The Max implementation (see Figure 2) consists of graphical objects that make routing the outputs and inputs of Control as easy as possible. To begin using Control with Max a user first creates a *control.master* object, which is responsible for finding instances of Control running on the local network and routing OSC messages to and from a selected instance. By default it will connect to the first running instance of Control it finds on a wireless network, but a drop-down menu listing all devices found via Zeroconf is available if a user wishes to select a different device. As soon as the master object is created, a blank interface appears on the selected mobile device.

Each widget in Control has a separate Max object representing it, with a single inlet and outlets to match the number of corresponding widget outputs. For example, the *control.knob* object possesses a single outlet outputting the current value of the knob while *control.acc* continuously outputs the three accelerometer axes via three outlets. A numerical value sent to the inlet of a Max object will set the value of the widget on the device running Control, while Max messages such as setRange and setBounds are translated into OSC messages using the address of the widget. Max objects that are copied and pasted will create unique instances of widgets in Control; this makes it extremely easy to set up arrays of sliders or buttons. When a Max object is deleted, its corresponding mobile widget is also removed.

The Max implementation is open-source and available for download[6].

2.5 LuaAV Implementation

LuaAV is an integrated programming environment for real-time audio-visual composition based on the Lua programming language [13]. An extension module to interface with Control is included in LuaAV.[7]

Both Zeroconf handshaking and registration with the mobile device and OSC socket management occur automatically when the *control* module is loaded:

```
-- handshake via Zeroconf, initialize OSC,
-- and create a blank interface on mobile device:
local control = require "control"
```

Controls of various types can be created with the *Knob*, *Slider*, and other factories in the *control* module. The first argument is an optional name for accessing the widget from Lua (otherwise a name will be auto-generated and returned); the second argument is an object or callback to map to. Additional named arguments can be used to specialize the widget. The following code creates a knob and slider on the mobile device and maps them to a synthesizer:

```
-- create a synth to control
local sine_def = Def{
    freq = 200, vol = 0.2,
    SinOsc{ P"freq" } * P"vol"
}
local synth = sine_def()

-- create widget on mobile device to control the synth
control.Knob{ "vol", synth }

-- creating with additional configuration
control.Slider{ "freq", synth, label="Frequency",
   value=220, min=110, max=880, page=1, color="#f00",
}
```

Once created, the widget name is used to get and set the value of the widget, or to remove the widget from the interface:

```
-- use widget's current value
print(control.vol)

-- set widget value (sends to device)
control.vol = 0.1

-- remove widget
control.vol = nil
```

The *control* module provides three ways to handle updates from the mobile device interface. Firstly, the state of each widget can be polled manually by indexing *control* with the corresponding widget name. Secondly, a widget can be set to update the corresponding field in a table or object, such as a synthesizer. Thirdly, a function callback can be mapped to a widget name. Mapping to an object or function can be done in the widget constructor or later using the *control.map(name, destination)* function:

[6] `https://github.com/charlieroberts/Control---Max-MSP-Integration`

[7] Available at `https://github.com/LuaAV/LuaAV`

```
-- map to a callback function
control.map("freq", function(...)
   print("frequency callback:", ...)
end)
```

2.6 SuperCollider Implementation

The SuperCollider[8] programming language is extremely dynamic and flexible. By taking advantage of these traits we have minimized the code required for interface generation and usage. However, unlike LuaAV and Max, SuperCollider is not currently capable of doing Zeroconf based auto-discovery of devices on a network. This means that users must manually enter the IP address and port of the device they wish to utilize.

The SuperCollider class *CNTRL* that integrates with Control is open-source and available for download[8]. The code example below shows how to create and access device widgets from within SuperCollider.

```
// initialize class object by passing ip + port
// of mobile device
CNTRL.init("127.0.0.1", 8080);

a = CNTRL.slider; // create a slider
a.value.postln;   // print the slider's current value

b = CNTRL.button; // create a button
b.value = 1;      // set state of button
b.value = 0;

// create button with callback
c = CNTRL.button( (\callback:{ arg btn; btn.value; } ) );

// pass optional parameters creating widget
d = CNTRL.button( (label:"TEST", page:1, color:"#f00") );

// Define a synth
x = SynthDef("sine-synth", {| freq = 440, vol = 1 |
    Out.ar(0, SinOsc.ar(freq) * vol);
}).play;

// map new widgets to synth parameters
e = CNTRL.button( \vol,  x);
f = CNTRL.slider( \freq, x, (min:440, max:880) );

// remove widget
e.remove;
```

Our SuperCollider implementation provides the same three mechanisms for handling updates from mobile device interfaces as found in LuaAV.

3. DISCUSSION

Users can focus on a single environment with networking managed behind the scenes, without having to deal with multiple workflows or networking issues. We believe that never having to leave the conceptual space of the host environment (Max, Lua, SuperCollider, etc.) is a substantial boost to productivity and creativity, as HCI researcher Ben Shneiderman argues in his writings on creativity support tools[11][12]. Shneiderman further outlines twelve design principles for creativity support tools that "...support rapid exploration and easy experimentation". Our research incorporates many of them:

- *Support exploration* - The ability to easily add and remove interface elements dynamically enables rapid exploration of interface ideas
- *Low threshold, high ceiling and wide walls* - Creating a simple interface in Control and connecting it to a prototyping environment can be as simple as two lines of code, but Control also enables users to define complex interfaces in JavaScript.
- *Support many paths and many styles* - Users can create interfaces statically or dynamically. They can use the AutoLayout object to create layouts for them or they can manually define boundaries for widgets. Users can build Control layouts with an explicit host-side object for each widget or more generally via scripting, from both graphical and textual programming environments.
- *Make it as simple as possible - and maybe even simpler* - We don't believe it is possible to make this process simpler with current mobile devices.
- *Choose black boxes carefully* - We believe hiding the networking protocols greatly simplifies the process; however users remain free to take advantage of the Control's OSC namespace to customize their workflows.

In addition to supporting easy experimentation, dynamic interfaces also create new possibilities for experimental composition. In a piece by the first author, entitled *Composition for Conductor and Audience*[10], audience members running Control receive interfaces pushed to them from a server at the start of the performance. Over the course of the piece the musical parameter space changes according to simple game-like rules, one of which is the ability of the conductor to "cut" audience members from the performance if they do not closely follow his or her direction. As members of the audience are cut from or introduced into the performance their interface changes to control different musical parameters. This type of piece is an example of audience participation and bi-directional networking that would have been impossible to create with other mobile device interface software.

We conducted informal interviews with potential users of our research to assess its usefulness and determine features they would like to see added. In the interviews subjects were solely shown the Max objects; all subjects had significant prior experience working with Max. The strongest feedback was support for the automation our framework provides. Selected quotes:

- "I like how it's automatic... trying to get other devices to work with Max is not usually this easy. There's normally a number of steps you need to complete."
- "You don't even have to think about layout... it does seem very fast. I would definitely use this with Max."
- "Auto-connection and auto-layout is useful... [it's] really tedious in other models. I think it's hella useful."

The only common feature request was a way to save generated interfaces on devices for future use. However, all of our implementations support the re-creation of interfaces from code. Max objects will always be created in the same order when opening a saved patch; this guarantees that the same interface will be generated each time by patches using Control objects. The sequential nature of code in LuaAV and SuperCollider also guarantees that an unchanged program will always generate the same interface. When this was explained to subjects two of them followed up by expressing a desire to modify interfaces on the device itself.

[8] https://github.com/charlieroberts/Control---SuperCollider-Integration

They envisioned a workflow where widgets could be created programmatically and automatically have their output routed to the correct OSC address but then manually positioned on the mobile device via a drag and drop paradigm.

Users were also interested in having a synchronized interface in Max that mirrored the interface found in Control. This would allow users to position and change parameters of the widgets using drag and drop inside of Max and would also provide a backup interface when a mobile device is not on hand.

4. CONCLUSION AND FUTURE WORK

The augmented OSC namespace in Control and its automatic layout generator make it easier for any prototyping environment to generate interfaces dynamically on mobile devices. We hope that that other implementations will follow the three already completed. The users we interviewed about the Max implementation all spoke positively of the research and expressed their desire to use it in their own practice.

User feedback yielded a number of interesting ideas for further research, including the mirroring of Control interface widgets in graphical programming environments and the ability to modify generated interfaces on the device itself.

We are interested in exploring introspection as a way to automatically generate interfaces for defined algorithms. In SuperCollider interfaces for SynthDefs can be generated this way using the AutoGUI [5] quark. SynthDefs are queried in order to determine their parameter space and a widget is then created for each parameter the SynthDef exposes for control. This approach often yields interfaces with controls for parameters that are not interesting to users and thus waste screen real estate. Our research to date has instead focused on providing users the ability to intentionally generate interface elements controlling parameters of interest. Nevertheless we are also interested in attempting to optimize the process of creating interfaces quickly via introspection; one simple improvement would be to generate interfaces automatically but then allow users to easily remove unwanted interface elements to prioritize control of the parameters they are interested in.

We look forward to completing new, experimental compositions that take advantage of the features added to Control. The ability to easily modify interfaces on devices opens up a relatively under-explored territory of composition by extending the temporal material of composition to include the interface feature space. We imagine compositions featuring "unstable instruments" in which performance interfaces change in unpredictable ways over the course of a piece, or game-like works in which achieving certain musical goals unlocks new controls to be utilized. The first piece to use the dynamic features of Control (the aforementioned *Composition for Conductor + Audience*) was a satisfying first step towards these goals.

5. ACKNOWLEDGMENTS

Thanks to JoAnn Kuchera-Morin and to the participants of our informal user study. This research was supported by NSF grants IIS-0855279 / IIS-1047678 and by two graduate fellowships from the Robert W. Deutsch Foundation.

6. REFERENCES

[1] M. Downie. Field – A New Environment For Making Digital Art. *Comput. Entertain.*, 6:54:1–54:34, Dec. 2008.

[2] 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.

[3] G. Essl. Designing mobile musical instruments and environments with urmus. *Proceedings of the 2010 Conference on New Interfaces for Musical Expression*, 2010.

[4] G. Essl. Automated Ad Hoc Networking For Mobile And Hybrid Music Performance. *Proceedings of the International Computer Music Conference*, 2011.

[5] z.-d. falurcu. Supercollider-quarks / autogui. `https://github.com/supercollider-quarks/autogui`.

[6] L. v. d. V. Iris Douma. iphone / ipad max 5/6 external. `http://nr74.org/nr74_software/c74.html`.

[7] C. Julià, D. Gallardo, and S. Jordà. Mtcf: A framework for designing and coding musical tabletop applications directly in pure data. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 20011.

[8] J. McCartney. Rethinking the computer music language: Supercollider. *Computer Music Journal*, 26(4):61–68, 2002.

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

[10] C. Roberts and T. Hollerer. 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.

[11] B. Shneiderman. Creativity support tools: accelerating discovery and innovation. *Commun. ACM*, 50:20–32, December 2007.

[12] B. Shneiderman. Creativity support tools: A grand challenge for hci researchers. *Engineering the User Interface*, pages 1–9, 2009.

[13] G. Wakefield, W. Smith, and C. Roberts. LuaAV: Extensibility and Heterogeneity for Audiovisual Computing. *Proceedings of Linux Audio Conference*, 2010.

[14] 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.

[15] M. Wright. Open sound control: an enabling technology for musical networking. *Organised Sound*, 10(3):193–200, 2005.