Mira: Liveness in iPad Controllers for Max/MSP

Sam Tarakajian
Cycling '74
730 Clementina
San Francisco, CA
sam@cycling74.com

David Zicarelli
Cycling '74
730 Clementina
San Francisco, CA
zicarell@cycling74.com

Joshua Kit Clayton
Cycling '74
730 Clementina
San Francisco, CA
jkc@cycling74.com

ABSTRACT

Mira is an iPad app for mirroring Max patchers in real time with minimal configuration. The Mira iPad app discovers open Max patchers automatically using the Bonjour[1] protocol, connects to them over WiFi and negotiates a description of the Max patcher. As objects change position and appearance, Mira makes sure that the interface on the iPad stays up to date. Mira eliminates the need for an explicit mapping step between the interface and the system being controlled. The user is never asked to input an IP address, nor to configure the mapping between interface objects on the iPad and those in the Max patcher. So the prototyping composer is free to rapidly configure and reconfigure the interface.

Keywords

NIME, Max/MSP/Jitter, Mira, ipad, osc, bonjour, zeroconf

1. INTRODUCTION

For the composer sick of feeling trapped behind a laptop, mobile devices make a seductive offer. In some ways the iPad represents the most tempting offer to date, with its ubiquitous presence, wireless capabilities and multitouch screen. Current models not only provide an accelerometer, gyroscope and magnetometer but also powerful built-in software for sensor fusion and data smoothing. With all these affordances, the performer can control many more parameters at once than would be possible with a mouse and keyboard alone, helping to discover musically interesting combinations[4].

At the same time, working with the mobile device as a performance interface presents its own challenges. All such challenges fall under the umbrella of the mapping problem, or how to find a pleasing and expressive mapping between interface and synthesis. Far from an exact science, the best "way to tackle the mapping problem is to in fact make the mapping part of the creative process."[1]

[1] `https://developer.apple.com/bonjour/`

NIME'13, May 27 – 30, 2013, KAIST, Daejeon, Korea.

Approaching the mapping problem as a creative challenge requires powerful tools for musical interface design. In the context of mobile devices, such tools must address:

1. The lack of availability of generic synthesis software for mobile devices
2. The challenge of finding appropriate GUI metaphors
3. The limitations of the mobile device (screen size, lack of alphanumeric input hardware)
4. The difficulty of robust ad-hoc networking. [3]

Other previous work, like the UrMus environment for mobile instrument design and performance, focuses on standalone mobile applications, where the mobile device is both the instrument by which sound synthesis is controlled and the device that performs that synthesis [2]. This approach brings its own challenges; in particular it forces the composer to program both the interface and the sound synthesis graph on the mobile device itself. This "standalone" approach poses a number of thorny design challenges, which is why many applications choose instead to look at the mobile device as a control surface. Instead of synthesizing sound on the device itself, this "fixed-mobile" approach leaves sound programming and synthesis design to the desktop, calling on the mobile device to perform as nothing more than a mobile controller.

1.1 Existing Mobile Controllers

Instead of trying to make the mobile device into a standalone musical instrument, a mobile controller targets the problem of "fixed-mobile" interaction design–using the iPad as a controller for musical synthesis running on another machine[7]. While this means that the iPad cannot be used without a laptop or desktop, it also means that the composer is free to use existing synthesis software instead of learning new software specifically designed for a mobile environment.

Probably the two most well known iOS mobile controller apps are TouchOSC[2] and Lemur[3], both of which allow the user to build a custom interface using a desktop design environment. In the case of TouchOSC, widgets like sliders and dials are associated with OSC addresses; Lemur also allows the user to assign MIDI channels to each control widget. In some ways these desktop interface builders represent one of the easiest ways to author a mobile interface, since they rely on a drag and drop paradigm already familiar to many users. However, the workflow for building a new interface from scratch can be somewhat cumbersome:

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

1. Open a specialized environment on the desktop machine for building iPad interfaces. Create a new project.
2. Build the interface by dragging interface objects into the project.
3. Define an OSC address for each interface object
4. Upload the interface to the iPad.
5. On the iPad, input the IP address and listening port of the synthesis environment that will receive OSC data.
6. In the synthesis environment, open some program for receiving and routing OSC data.
7. In the synthesis environment, route that OSC data to the desired callback or interface object, being sure to scale appropriately.

This workflow requires that the composer consider uninteresting technical problems instead of musical challenges. In his presentation "Inventing on Principle", Bret Victor describes a "principle of immediate connection" that he asserts is essential for creative tools[13]. "Creators need immediate connection to what they're making," so an iPad interface cannot be part of the creative process if it disconnects the composer from his composition. Controlling synthesis with an iPad is important for convenience, but solving the mapping problem by creative experimentation requires that the interface itself be rapidly reconfigurable.

Another interesting interface app is mrmr[4], which brings a number of promising solutions to the problems of mobile interface design. In particular, mrmr lets the user push new interfaces dynamically, using Bonjour to abstract away the challenge of network configuration. However, mrmr can be so dynamic and reconfigurable only by sacrificing ease of use–there is no simple, drag and drop environment for building interfaces in mrmr. Anyone who wants to design a mrmr interface must learn the mrmr language for describing such an interface in text.

One of the most time consuming parts of working with TouchOSC or Lemur is mapping OSC and MIDI messages to controls and parameters on the fixed machine. The c74 app gets around this problem by insisting on a tight coupling between user interface objects that appear on the desktop and those that appear on the mobile device [5]. Unlike other mobile controller apps, c74 does not have a dedicated desktop environment for building interfaces. Instead, the c74 app connects to the desktop through an external object directly in the Max/MSP environment. While this limits the scope of the app, it also makes it much easier to rapidly prototype new interfaces. A slider in the c74 app is tightly bound to a slider in the Max application, so the user is free to think of these two sliders as one in the same, rather than as two separate controls mapped to one another via OSC. The only drawback is that c74 still builds interfaces using text commands. Establishing the connection between UI objects in Max and widgets in the c74 app requires sending messages through the c74 external to the mobile app, which makes it hard to rapidly experiment with new interfaces.

Finally, in terms of custom interface design, the Control app represents one of the most powerful mobile controllers available[6]. Built on top of Webkit[7], the Control app lets users build custom widgets of arbitrary complexity using HTML and Javascript. Those widgets can then be mapped to desktop synthesis parameters using both OSC and MIDI. Also, interfaces themselves can be stored and shared through JSON files. The drawbacks to the Control app are that it still requires network configuration (entering and looking up IP addresses to make connections between desktop and mobile device) and that it uses text for interface design. While this does make it possible to build dynamic and scriptable interfaces, it puts a barrier in front of composers who don't want to learn a new language for describing interfaces.

Figure 1: Mira displaying a patcher containing supported objects

1.2 Mira

Like other mobile controllers, Mira leaves sound synthesis to a dedicated machine, focusing instead on the challenge of providing a rapidly reconfigurable interface. In this sense, one way to look at Mira is as a tool for user interaction design. Such tools are important because:

> In general, tools help reduce the amount of code that programmers need to produce when creating a user interface, and they allow user interfaces to be created more quickly. This, in turn, enables more rapid prototyping and therefore more iterations of iterative design that is a crucial component of achieving high-quality user interfaces.[6]

Implementing a quality tool for user interface design required adhering to a number of well-documented principles. In particular, such a tool must:

- Follow the Path of Least Resistance: Users should be lead towards making the right decisions, and towards

[4] http://mrmr.noisepages.com/
[5] http://www.nr74.org/c74/c74.html
[6] http://charlie-roberts.com/Control/
[7] http://www.webkit.org/

taking advantage of perceptually salient control parameters.

- Have Low Thresholds and High Ceilings: Getting started with the tool should be easy, and the tool should have a wide range of capabilities.
- Hit Moving Targets: Especially in the context of a musical controller, the requirements of the interface are changing constantly, as the underlying synthesis architecture itself changes. The interface must therefore be able to change just as rapidly.[5]

Mira tries to follow all of these principles by building both on other mobile controller apps and on the Max/MSP environment itself. Critically, the user doesn't need to learn anything new to build a Mira interface, since interfaces are simply the automatic reflection of Max patches. Also, putting the interface in the same environment as the sound synthesis program means that the user never has to leave the composition context to add or reconfigure an interface. On the contrary, having an automatically synchronized interface means the composer can discover unexpected and inspiring mappings between synthesis and interface. Finally, since changing a Mira interface is as simple as changing a Max patch, the interface can always stay in sync with the patch, even as the patch itself changes in both configuration and function.

2. ARCHITECTURE

Mira reduces the work needed to control Max from an iPad to a single step: creating an object inside the Max patcher. In order accomplish this, Mira either automates or abstracts away the technical obstacles that would ordinarily confront the user while building an interface. This is made possible thanks to a library for discovery and synchronization called Xebra, written specifically for use with Mira but designed to be portable to other platforms and applications.

2.1 The Xebra Library

The goal of the Xebra library is to provide a general purpose way to share state among multiple distributed endpoints. These endpoints could be within a single application, or they could be on several applications spread across multiple devices. Xebra assumes that the state itself exists at one central endpoint, and that other devices will connect to that endpoint to view and modify its associated state. In the language of Xebra, that central endpoint is called a server, and only the server has an associated state. That state takes the form of a rooted tree of nodes, each of which can have many children and exactly one parent. A node is either an object or a parameter. Both objects and parameters may be associated with a user supplied class name and ID string, but parameters also have an associated value, which can be any number of integers, floating point numbers, strings or binary objects.

The server may advertise itself over Bonjour, which makes it discoverable by client endpoints. When a client discovers and connects to a server, the server sends the client a series of OSC packets that the client uses to reconstruct the state graph as it exists on the server. From that point on, the server is free to continue modifying the structure of the object graph, and these changes are propagated to all connected clients. A client may not modify the structure of the state object graph, but it may freely modify the value of any parameter in the graph. The server will receive those changes, resolve any conflicts, modify its own parameter value and then push the result to all connected clients.

The mira.frame Max object uses the Xebra library to share the state of the current Max patcher. On the other end, the Mira iPad app uses Xebra to discover advertised Max patchers and to display their interface objects on the iPad.

2.2 Automatic Connection

The first and only step to making a Max patcher visible to the Mira application is adding a mira.frame object to the patcher. The mira.frame object appears as a rectangular region that defines the area of the patcher that will be visible on the iPad. The size of the mira.frame object determines the scale of the interface as it will appear on the iPad. Creating a mira.frame object also creates a Xebra Server that describes the Max patcher to any clients that connect to it. The server is advertised over Bonjour, discoverable to any client looking for services of type _tcp._mobilemax. Launching the Mira iPad app looks for a Xebra Server and connects to it automatically. After some negotiation, during which the two endpoints confirm that they are running the same version of Xebra, the iPad will start communicating with Max.

2.3 Synchronizing State

When a mira.frame object is added to a patcher, it attaches to that patcher. When an object is added to or removed from the patcher window, it triggers a callback in the mira.frame object, which passes the update on to the Xebra Server. In the language of the Xebra library, this corresponds to adding a node to a graph of objects and parameters. Each node can have a class name and a unique ID associated with it. A max object is represented as an object node whose classname is the name of the Max object. Each such object has one object as its parent: the patcher itself. In this way, the iPad application can stay in sync with the Max patcher as objects are added and removed by translating the Xebra object graph into an interface display.

Each object can have a number of attributes associated with it, for example patching rectangle, background color or label. The Mira application tells the Max patcher what it needs to know in order to draw each interface object. The Max patcher then adds the corresponding Max attributes to the Xebra Server as parameters in the state graph. When Max adds such a parameter to the Xebra state graph, it passes three callback functions along with the name of the attribute and the id of its parent object. The first of those three functions is expected to return the count of elements for a given parameter. The second supplies the actual value of each of those elements. The final callback function is called by Xebra whenever is receives a change from a client, and updates the attached object accordingly. With these callbacks in place, when an attribute is modified in Max, the object simply has to notify the server. This will use the callback functions supplied when the named parameter was created. The resulting value is captured in an OSC packet and then sent to all connected clients. On the client end, the OSC packet is dissected and routed automatically, ultimately updating the interface object on the iPad display.

2.4 Bidirectional Communication

The Xebra library assumes that state updates will propagate from the server to the client in real time. In this model there is no inherent difference between the value of a slider and its color–both are simply parameters in the state graph. When the performer drags a slider on the iPad, obviously this change is reflected in the associated Max patcher. At the same time, if the color of the slider changes in the Max patcher, that change also propagates to all connected iPads.

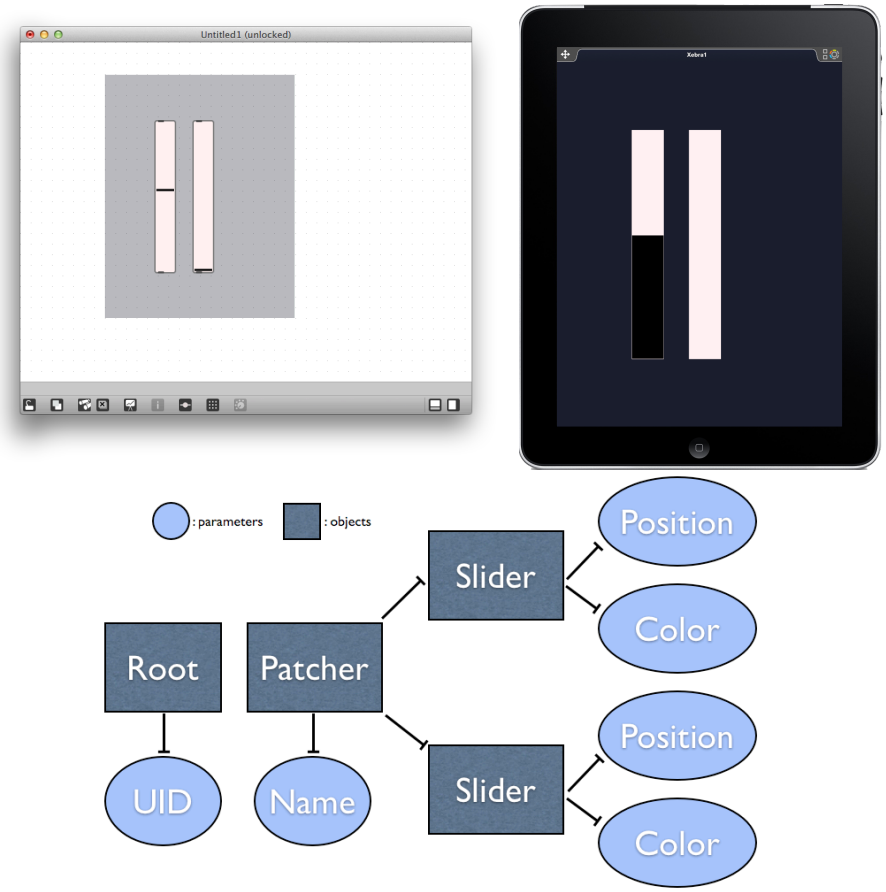

Figure 2: A simple Max patcher represented as a Xebra object graph and as it appears in Mira

So Mira can not only transmit control data from touch, acceleration and other physical modalities from the iPad to the Max, but also it can provide feedback about the state of synthesis in the patcher to the iPad.

2.5 Multiple Endpoints

Since Xebra assumes that a central endpoint may serve multiple clients, adding a mira.frame object to a Max patcher allows that patcher to be controlled by any number of iPads. This opens the door for collaborative performance, with any number of performers or audience members controlling a single master patcher[8]. Also, since the Mira app is free to form multiple connections to any number of Xebra endpoints, one instance of Mira can quickly control any number of patchers running on any number of different devices. Adding a mira.frame object to a patcher defines a region of the patcher that will be made visible to instances of the Mira app. If two mira.frame objects are added to the same patcher, those two regions of the patcher are represented in the Mira app as two different interface tabs. If Mira connects to a Max patcher running on another machine, that patcher will simply appear as another tab. So the performer can jump between controlling patchers on two different machines as quickly and as easily as between two regions in the same patcher.

[8]The Control app has already been used for one such performance[9]

3. DISCUSSION

Part of the joy of composing in Max comes from the fact that changes to a Max patcher take effect immediately. The goal of Mira was to bring the same sense of liveness and experimentation to the process of building a user interface. To accomplish this, Mira adheres to a tacit contract that interface objects are not separate entities connected by a MIDI or OSC mapping, but rather two views of the same underlying data. In this sense Mira follows the model of Control, an iOS and Android app that allows the composer to build dynamic OSC interfaces on the fly[8][10]. Mira goes one step further, however, in that the appearance of interface objects on the iPad is the same as in the Max patcher. Because of this, the Max environment can be used simultaneously to describe both layout and behavior.

Objectively speaking, Mira represents a step forward in terms of the ease of building touchscreen interfaces for Max. Most importantly, the composer never has to leave the creative environment of Max/MSP to prototype a new interface. This uninterrupted flow is critical for creativity and productivity, according to HCI researcher Ben Shneiderman[11][12]. With Mira, the iPad becomes a tool for investigating questions of user interface design, rather than a control surface to be programmed. Of course, since the composer is working in Max/MSP, those questions can be unexpected and the answers themselves creative. With the power of Javascript in Max, for example, the composer can generate an interface dynamically, or build a reactive interface that changes even as it is played.

Figure 3: Mira displays multiple regions as separate tabs

4. CONCLUSIONS AND FUTURE WORK

We would like Mira to run on more devices and platforms. As of now the Mira app will only run on iPad and the mira.frame object is limited to running on OS X. While building the initial version of the app we chose a single deployment environment so that we could focus our efforts on design and execution. Now that we have an initial version we would like to turn our attention to other platforms.

Mira makes the promise to faithfully represent the state of a running patcher. The Xebra protocol assumes that the state of the patcher exists in Max and that the iPad simply reflects that state, but it would be convenient to upload a patcher to Mira and to use the iPad as a device for storing and sharing patches. It would be possible to achieve this without modifying the protocol too significantly.

As of February 11, 2013, maxobjects.com, the largest repository of Max external objects, contains 4340 Max objects. The current version of Mira is able to display about 20 user interface objects; the look and behavior of each is hard-coded into the Mira app itself. While it is possible to add more objects at any time by updating Mira, it is not possible to add more objects without modifying the Mira source code. We would like to support user-defined objects as well as objects whose appearance can be changed on the fly.

In addition to accelerometer and other sensor data, the iPad is also able to capture audio and video from a built in microphone and camera. If it is possible to do so without adding latency to the OSC data already coming from the Mira app, we would like to find a way to make it easy to stream live audio and video from the Mira app to a Max patcher. Going in the other direction, it would be extremely useful to see sound and video generated inside a Max patcher displayed on the iPad. Again, latency and bandwidth remain important concerns, but it's easy to see the appeal of a jit.pwindow object appearing in the Mira app.

The Xebra protocol supports zero configuration discovery and state synchronization. Mira has shown how useful this can be when mirroring the state of a Max patcher on an iPad, but communicating state information in the opposite direction could be just as useful. Through the camera connector kit, the iPad receive data from a number of USB and MIDI devices. If it could communicate the state of a connected device, then Mira would be an ideal platform for wirelessly sending data from a range of hardware devices to a Max patcher.

Finally, we have to acknowledge what Mira is not: Max for the iPad. Even after building a patcher in Max and uploading it to the Mira app, it is still not possible to run that patcher without Max. We would like to make it possible to author patchers in Max that could run in a standalone configuration on the iPad, so that the user could use his iPad to generate live audio and video without needing a connection to Max.

We hope that Mira will uncover unexpected and delightful ways to control music with touch. A flourishing ecosystem of interfaces would be a rewarding sign that liveness and experimentation are indeed as important to interface design as they are to composition. With time and hard work, the Xebra protocol itself may too emerge as a useful way of helping real-time environments discover each other and to communicate.

5. ACKNOWLEDGMENTS

The authors wish to thank IRCAM for providing workspace and feedback.

6. REFERENCES

[1] G. Essl. Speeddial : Rapid and on-the-fly mapping of mobile phone instruments. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 270–273, 2009.

[2] G. Essl. *UrMus - An Environment for Mobile Instrument Design and Performance*, volume 276, pages 76–81. 2010.

[3] G. Essl, G. Wang, and M. Rohs. Developments and challenges turning mobile phones into generic music performance platforms. *Signal Processing*, pages 13–16, 2008.

[4] A. Hunt and R. Kirk. Mapping strategies for musical performance. *Trends in Gestural Control of Music*, 21, 2000.

[5] B. Myers, S. E. Hudson, and R. Pausch. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 7(1):3–28, Mar. 2000.

[6] J. Nielsen. *Usability Engineering*. Interactive Technologies Series. Morgan Kaufmann Publishers,

1994.

[7] A. Ramsay. Interaction design between fixed and mobile computers. *Master's thesis, University of Glasgow, Department of Computing Science*, 2005.

[8] C. Roberts. Control: Software for end-user interface programming and interactive performance.

[9] 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*, pages 65–66. ACM, 2011.

[10] C. Roberts, G. Wakefield, and M. Wright. Mobile controls on-the-fly: An abstraction for distributed nimes.

[11] B. Shneiderman. Creativity support tools: accelerating discovery and innovation. *Communications of the ACM*, 50(12):20–32, 2007.

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

[13] B. Victor. Inventing on principle. CUSEC, February 2012.