NEXUS: Collaborative Performance for the Masses, Handling Instrument Interface Distribution through the Web

Jesse Allison
Louisiana State University
216 Johnston Hall
Baton Rouge, Louisiana
70803
jtallison@lsu.edu

Yemin Oh
Louisiana State University
216 Johnston Hall
Baton Rouge, Louisiana
70803
yoh1@lsu.edu

Benjamin Taylor
Louisiana State University
216 Johnston Hall
Baton Rouge, Louisiana
70803
btayl61@lsu.edu

ABSTRACT

Distributed performance systems present many challenges to the artist in managing performance information, distribution and coordination of interface to many users, and cross platform support to provide a reasonable level of interaction to the widest possible user base.

Now that many features of HTML 5 are implemented, powerful browser based interfaces can be utilized for distribution across a variety of static and mobile devices. The author proposes leveraging the power of a web application to handle distribution of user interfaces and passing interactions via OSC to and from realtime audio/video processing software. Interfaces developed in this fashion can reach potential performers by distributing a unique user interface to any device with a browser anywhere in the world.

Keywords

NIME, distributed performance systems, Ruby on Rails, collaborative performance, distributed instruments, distributed interface, HTML5, browser based interface

1. INTRODUCTION

With the proliferation of mobile devices, avenues for distributing a live performance system, or instrument, for collaborative performance are growing. Distributed performance systems have undergone many leaps forward with the increased speed of networks, proliferation of smart phones, and the emergence of ensembles such as laptop orchestras that are ideal for exploring these kinds of interaction. As mobile devices continue to become more powerful and ubiquitous, integrating them into performance systems is increasingly desired[8]. Along with these explorations in collaborative performance come some significant distribution and data management challenges. A number of approaches for distributing a performance system, the challenges in implementing them, and a testable solution will be examined. Many of the approaches have been put into practice in the author's collection of laptop ensemble and audience pieces entitled *Perception*[3] and will be noted when appropriate.

2. DISTRIBUTED PERFORMANCE SYSTEMS

Distribution of an instrument for collaborative performance requires breaking down the system into parts and deciding what to distribute and how these separated parts should communicate[6]. Typical parts of such a composed instrument[10] system are the *user interface* such as buttons, sliders, and accelerometers, the *mapping structure* defining how the controls are interpreted by the instrument to create sound, the *communication system* defining how instrument parts pass and receive control messages, and the *audio production engine* or *audio graph* actually producing the waveforms. With these components of performance systems in mind a number of approaches to distributed instruments become apparent.

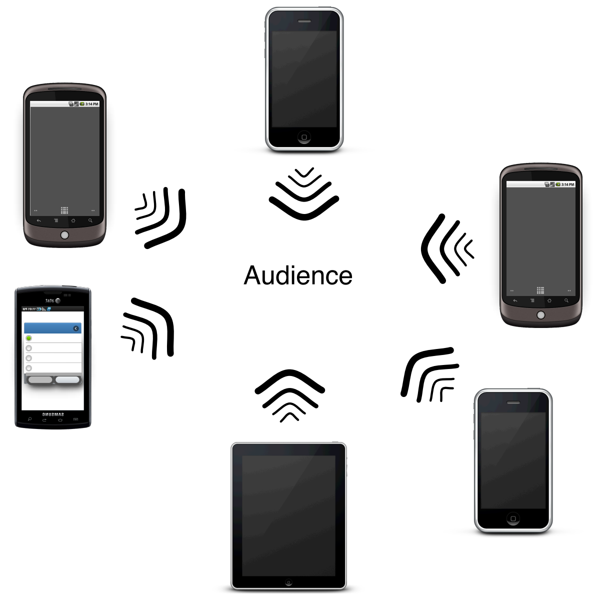

Figure 1: Autonomous Performance System.

2.1 Autonomous Collaborative Performance System

Perhaps the simplest approach is to have the instrument be self-contained, installed on each performer's device, and operating individually. Each user has an interface, mapping structure, and audio production engine installed on their device. This approach is taken by many pieces for laptop orchestra[1], traditional mobile music applications such as ThumbJam[2] and Ocarina[3], as well as the approach by such toolkits and libraries as libPd[4], MoMU Toolkit[5], and uRMus[7]. Although such a system may communicate with each other or with a central server/coordinator to increase the collaboration in performance, at its core, each could operate and make sounds autonomously. The collaborative nature in this type of system would be analogous to a band where each individual has a sonic contribution to the sound of the ensemble. Admittedly the delineation between ensemble and collaborative meta-instrument is debatable, but the idea of a group of collaborating performers combining to form a meta-instrument is a useful analogy, and as software can augment communication and influence between individual performers, the meta-instrument metaphor becomes even more significant (Figure 1).

2.2 Centralized Audio Production, Distributed Interface

A second approach is through seperation of the interface for an instrument and the sound production engine of the instrument itself. This approach is taken by such software as TouchOSC[4], Squidy Interaction Library[5], and OSCulator[6], among many others, to pass control parameters from graphic user interfaces and sensors to a central audio engine for sound production. This separation requires a good deal more structure and coordination in the software design which by necessity makes the interface more rigid once deployed. Computationally, it is both restrictive and freeing to have a centralized audio engine. The audio engine is no longer restricted by the computational power of an individual mobile device, but cannot leverage the power and scalability of many individual rendering engines. Another difference is the localization of the final produced sound. The sound can no longer emanate from the point of interaction except through the use of streaming audio back to the interface which has inherent latency and bandwidth issues. Also, instead of relying on the small and typically impotent sound system on mobile devices, the sound can be produced over a more adequate sound system.

A consequent approach that is conceptually distinct, yet still derives from audio engine and interface separation, is the division of the interface into parts, with distribution of those smaller parts of the overall instrument to many devices/performers. In this scenario, performers contribute to the state of a central audio engine and collaboratively define what sonic events are produced(Figure 2).

2.3 Responsive Server, Adaptive Interface

Finally, a responsive system could be created where the central server handling distribution of the interface would also receive, process, and send messages and data back to the interfaces. These commands could be displayed directly to the performer, setting parameters on the interface, or completely updating the interface. In *Perception[Divergence]*[3], a system was built where images can be drawn on a device and submitted back to the server. Once received, the server passes them out to another user for further annotation, creating a collaborative image interface. All sorts of coordinated collaborative experiences could be envisioned.

[1]examples: Plork http://plork.cs.princeton.edu/ and the Laptop Orchestra of Louisiana http://laptoporchestrala.wordpress.com/

[2]http://www.thumbjam.com/

[3]http://ocarina.smule.com/

[4]http://hexler.net/software/touchosc

[5]http://merkur61.inf.uni-konstanz.de:8080/squidy/

[6]http://www.osculator.net/

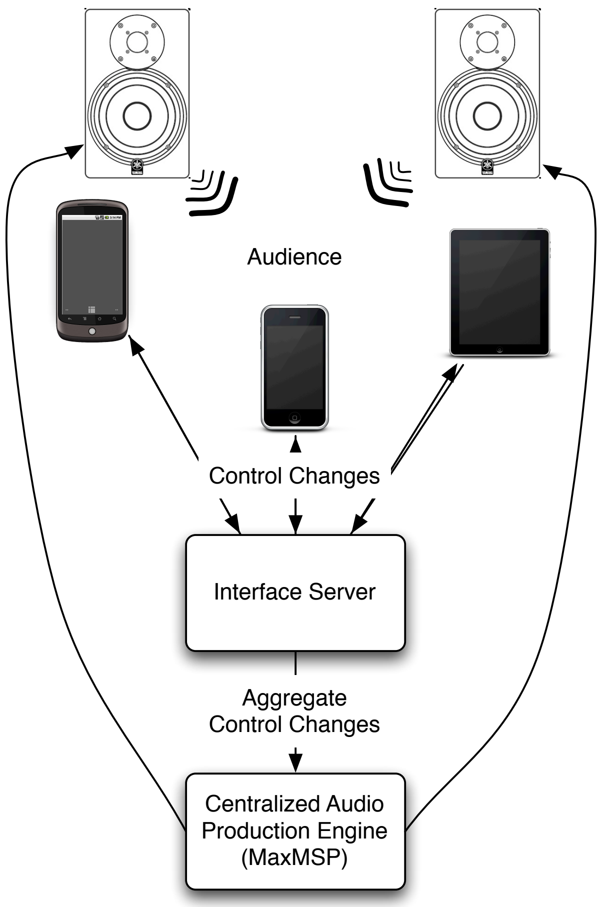

Figure 2: Distributed interface with centralized audio production.

2.4 Possibility of Other Approaches

Note that there are many degrees of variation between each of these types of systems. One could envision distributed environments that span the range of cross-performer influence, instrument segmentation and distribution, local versus centralized sound production, and bi-directional influence on the user interface itself. The main similarity is a need for distributing an interface, coordinating communications between collaborators, and coordinating control changes for audio production.

2.5 Challenges to Distribution

There are many varieties of operating systems, development kits, frameworks, and proprietary code making the incorporation of more than one type or generation of device difficult to do except under very narrow use cases. Although there are cross-platform development efforts such as OpenFrameworks[7], they are not yet fully realized and are constantly dealing with upgrades and incompatabilities across many devices. The current state of the art requires a platform dependent application to be built for each device to be incorporated in the system. This can be very daunting and in many cases down right impossible due to budgetary and time constraints. The simple idea of distributing an instrument to people with mobile phones may require supporting iOS, Android, Blackberry OS, Symbian, PalmOs, Windows Phone, and any other system who may want to participate.

Once the application is built, managing communications between more than just a few devices becomes remarkably complex. Beyond the significant task of defining the interaction, a suitable communications protocol must be chosen along with a framework to support it on each platform. Then, a system must be coded for passing, receiving, and handling the interactions. Master/slave pairings must be negotiated, not to mention the difficulties of trying to aggregate many sources of information to create a final parameter. The complexity of such a system only increases as more devices are added to the system, or even worse, added and taken away dynamically.

2.6 Enter the Web

In each of the distributed performance systems described above, two significant problems arise: cross-platform application development and coordinating communication and interaction.

Pushing the user interface of a distributed performance system into the browser helps to alleviate both of these issues. Due in large part to the ubiquitous usage of the internet, much effort has been poured into creating standards for display and interaction with web pages[8]. Although browsers are notorious for behaving somewhat differently, overall they can be made to look and function similarly across a wide variety of platforms both desktop and mobile. Compared to other cross platform initiatives, browser functionality is quite well developed. Once the choice to adopt a browser based UI is made, the artist can develop a single interface that can be utilized by most web enabled devices.

An additional concern with browser based performance is that of perceived latency. Interactivity in the browser has a few bottlenecks - notably, javascript as the coding language, http requests and AJAX as the communications protocol, and the lack of availability of sensor data within browsers. Fortunately, browser implementations of javascript in HTML5 are making marked speed improvements to the point that most types of interactivity can be made to feel responsive. AJAX is a bit slower than the ubiquitous UDP used in most distributed user interface software, however with the rise of web sockets as well as browsers embedded in applications, communications can be made quite snappy, or simply made through UDP instead. Finally, browsers are gaining more and more functionality including access to accelerometers, multi-touch data, camera, audio input, and location information; finally approaching the performance of their native counterparts. In *Divergence* (Figure 3)the browser was used in an iPad to send continuous accelerometer and touch data to a central server to control the sampling and pitch shifting of a marimba.[2]

[7]http://www.openframeworks.cc/

[8]W3C's latest specification of HTML5: http://dev.w3.org/html5/spec/Overview.html

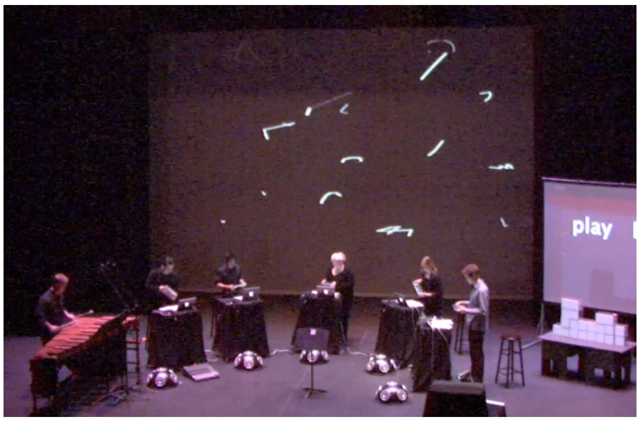

Figure 3: Browser based accelerometer and touch performance interface.

3. BROWSER BASED USER INTERFACE

Moving the display and interaction of a user interface into a browser allows for cross-platform distribution. Dynamic webpage distribution is typically handled through a server side web application. There are a variety of approaches and solutions to web communication, state management, and control logic, which have tradeoffs in complexity, ease of use, ease of support and deployment, and compliance to evolving best practices. Various options include everything from javascript, Perl, and php, to full frameworks like Rails and .NET. The authors have created a php based distributed UI, Node.js, and full web app implementations using Ruby on Rails.

Ruby on Rails is quickly adaptable to a wide variety of scenarios that allow for interesting performance paradigms. The MVC architecture can be easily adapted to cover instrument interface distribution, parameter management, and control logic. As the Rails community focus on Agile development supports the ability to quickly try out various implementations and Rails tends to be painfully near or at the cutting edge of web best practices, we chose to do much of our prototypical development utilizing the framework and will discuss further implementation details in that light.

3.1 Rails: a useful solution

Ruby on Rails is a framework for developing web applications that handles much of the distribution and information management issues through well-trodden conventions as opposed to configuration. The end result is a very rapid development process that is easily adaptable to the developer's specific goals. It handles many of the normal issues of scalability, testing, database configuration and management, session management, and customized user interface generation. When applied to the distribution of user interface objects and coordinating their incoming responses, standard Rails techniques have performed admirably.

3.2 MVC Architecture Adapted for Performance Instrument

Rails uses a traditional MVC software architecture, adopted for handling web applications. The process for serving traditional web pages goes something like this (Figure 4):

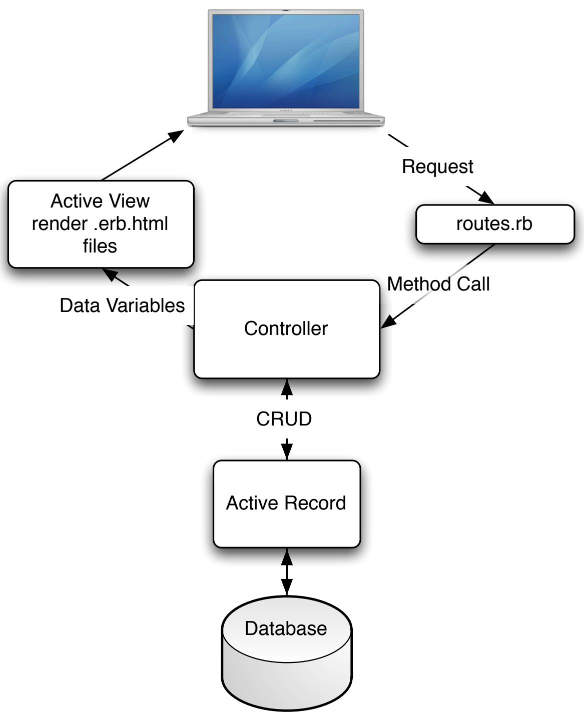

Figure 4: Overview of the Rails process.

An incoming page request is handled by the route.rb file specifying what controller method to call. The controller method then handles the gathering and processing of any information needed for the requested page. This step typically involves polling the Model (database) to retrieve information like blog posts, calendar events, and user information for display. Once the controller has prepared the needed information, it is passed to the appropriate view files for display. The view files are usually written with embedded ruby (ex. index.erb.html) which handles the iteration through arrays and hashes of information, the generation of html tags, image names, user control options, inserting partials and the like. These ruby commands are processed to produce the final markup which is passed back to the requesting browser.

Of course this can become much more complicated with logic occuring in the view in the form of helpers, AJAX methods, end user specific code, etc., but for the most part, this series of events is the standard approach. In usage as a performance instrument environment, the model/view/controller activities are divided in the following ways.

The *Model* in our application handles any data that we want to store and recall. In the simplest use cases no information would be stored, instead passing interactions directly along to the audio engine. The model could be used to store user specific information such as which parameters are actively assigned to which user, routings, musical sections, etc. In a complex scenario, it could store user defined sequences, images, preferences, snippets of audio, aggregate parameters, just about anything.

The *Controller* has the assignment of divvying up the user interface parameters and coordinating the incoming responses. Upon first request from a potential participant this would include selecting parameters and controls to be assigned to the user, storing any user specific information that may be used later, and calling the appropriate view files to be rendered. Further contact from the page would be from UI changes. The controller would then either put them into the database or simply pass them along to the audio production engine as an OSC message (Figure 5). The controller would also handle sending each user any updates to the UI that may occur: waveform changes, color events, complete UI replacement, etc.

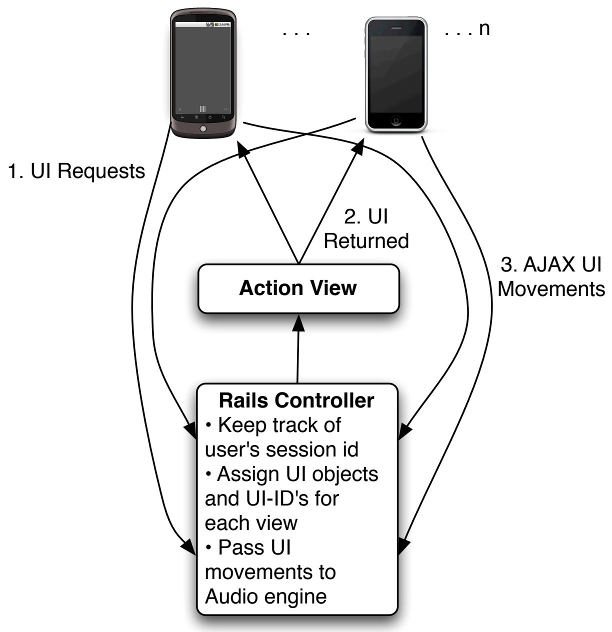

Figure 5: Rails UI Distribution.

Finally, the *View* receives information from the controller and uses that to create user specific interfaces. These can take the form of traditional buttons, drop down menues, and text boxes, but can also incorporate sophisticated graphical UI objects through javascript. The widespread adoption of the HTML5 canvas tag allows for dynamic drawing and interaction with a graphical element. The NEXUS open source project [1] contains a set of nexusUI javascript objects demonstrating many dynamic functions that can be done within the canvas object including: timer based automation, inter-canvas communications, multi-touch, accelerometer input, and AJAX callbacks.

3.3 Web App as Parameter Pass-Through

The web application can be used to simply split up the interface and pass any returning commands directly to the audio rendering engine. In this simple scenario, after the interface has been distributed to the user, any returning posts are simply routed on to the audio engine through rosc. Individual users can be identified via session information with each request, or by customizing the submit name when the original html interface file is generated (Figure 6).

Here you can see the id of the slider inserted into the id attribute of the canvas tag.

`<canvas id="slider_<%=@slider_id%>" > [Slider] </canvas>`

This will generate code like:

`<canvas id="slider_1" > [Slider] </canvas>`

The id can then be used as an osc name when passed on to the audio rendering engine.

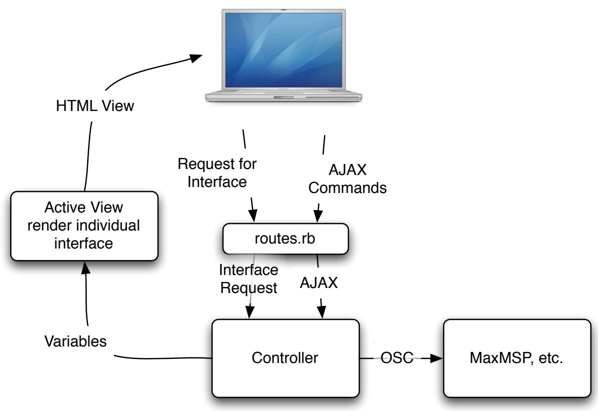

Figure 6: Rails distributed interface and interaction pass-through.

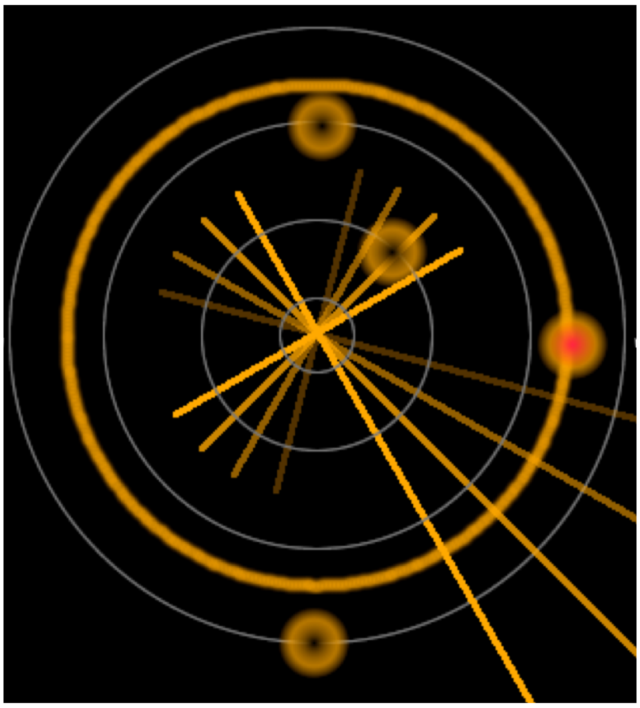

Figure 7: Radial sequencing interface distributed to audience' mobile devices.

3.4 Web App as Instrument Distribution Center

To divide and distribute an instrument, Rails can handle this in two ways. For simple matters, it can dynamically assign a different parameter value for each slider, button, or UI element that is dispensed, for example: `slider_1`, `slider_2`. This works well if there are a finite number of UI elements in the instrument and you know how many people are playing it at any one time.

A more sophisticated approach would involve the use of sessions. Every browser request has a session id that can be used to identify who is making the request. With this approach, information can be stored through the model about each users interactions, then as an interaction is received the controller can use this collected knowledge to respond. For instance, the model can store the fact that user *xyz* is assigned sliders 5 and 7. Then when a slider change is received it can pull up the slider assignment and pass along the correct changes.

In *Relativity*, (Figure 7) a movement of Perception, a variable number of audience members is provided a radial sequencing interface where a pulse emanates from the center and any points that they touch in its path triggers musical events.[2] The web application distributes a unique interface to each audience-performer. It then receives the interactions via AJAX and sends OSC to the audio production engine for rendering the collaborative musical experience. The communication is also bi-directional allowing for the server to send messages to each user to set the pace of the pulse collectively for the group. At the premiere in March of 2012, more than 100 unique devices were connected at one point during the performance.

3.5 Database as State Engine

Another approach is to use the Database for state management for the instrument. Every change in user interface objects can be stored in the database as well as setting the audio engine. The database could store many users' input and aggregate it to come to a collaborative parameter setting. In these scenarios, the database state could also be copied and stored as a preset and complicated states could be quickly retrieved or even interpolated.

4. FUTURE DIRECTIONS

The viability of these methods have been established through performances, interactive kiosks, and installations, and many new avenues of exploration present themselves. On the web application front, various use cases could include: small ensemble with direct realtime responses, broad distribution utilizing audio streaming, location awareness, a control parameter database for aggregating interactions, and picture/drawing based interactions.

A library of javascript user interface objects, nexusUI, has been developed to facilitate performance actions and ease of incorporation into Rails applications, other web technologies like Node.js, other programming platforms implementing javascript canvas drawing, embedded browsers, and Web Audio.

Embedding browsers into native applications and using them for user interface has proved to be a fruitful field. It allows us to use the nexusUI javascript library to quickly create native user interfaces which can then engage extra sensors, audio/video processing, or UDP communications for added responsivity.

Another interesting development concerns the growth of the Jamoma Audio Graph[9]. In a recent incarnation, cross compilation was added allowing one to create an audio graph in Max and export it as a pd patch, c++ code, or ruby[9]. In a series of experiments we were able to get a Ruby Jamoma Audio Graph running inside a rails application. This would allow for a web service to be deployed which would handle all of the audio production as well as user interaction.

Further testing and benchmarking are being carried out to see what kinds of performance interactions will be viable for various scales of participants.

[9]Jamoma Foundation, http://jamoma.org

5. CONCLUSIONS

Web applications such as those built on Ruby on Rails have proven to be a viable framework for handling the serving of web pages containing user interface objects to many potential performers. The cross-platform nature of current browsers solves many issues dealing with dissemination and support. The ability to coordinate dynamically assigned user interface objects is very useful. A built in database gives flexibility in the types of interactions that can be acquired and used. Finally, a well trodden path for scalability is available so that the accomodation of many simultaneous users is possible.

Now that many features of HTML 5 are being incorporated among the leading browsers, powerful browser based interfaces can be utilized for distribution across a variety of static and mobile devices - making worldwide collaborative creative arts a distinct possibility. Interfaces developed in this fashion can reach potential performers by distributing a unique user interface to any device with a browser anywhere in the world. In overcoming platform dependency issues, the emerging viability of browser based interface has become an exciting frontier.

6. ACKNOWLEDGMENTS

Many thanks to the CCT and LSU AVATAR Initiative, the LSU School of Music, as well as Timothy Place and the Jamoma Foundation for their work on the Jamoma Audio Graph and Ruby.

7. REFERENCES

[1] J. Allison. Nexus, https://github.com/jesseallison/nexus.

[2] J. Allison. Divergence for marimba and mobile ensemble, http://perceptionevent.com, March 2012.

[3] J. Allison. Perception, http://perceptionevent.com, March 2012.

[4] P. Brinkmann, P. Kirn, R. Lawler, C. McCormick, M. Roth, and H.-C. Steiner. Embedding pure data with libpd. In *Proc Pure Data Convention 2011*, 2011.

[5] N. J. Bryan, J. Herrera, J. Oh, and G. Wang. Momu: A mobile music toolkit. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME), Sydney, Australia*, 2010.

[6] P. L. Burk. Jammin' on the web - a new client/server architecture for multi-user musical performance. In *ICMC 2000 Conference Proceedings*. International Computer Music Conference, 2000.

[7] G. Essl and A. Müller. Designing mobile musical instruments and environments with urmus. In *New Interfaces for Musical Expression*, pages 76–81, 2010.

[8] L. Gaye, L. E. Holmquist, F. Behrendt, and A. Tanaka. Mobile music technology: report on an emerging community. In *Proceedings of the 2006 conference on New interfaces for musical expression*, NIME '06, pages 22–25, Paris, France, 2006. IRCAM - Centre Pompidou.

[9] T. Place, T. Lossius, and N. Peters. The jamoma audio graph layer. In *Proc. of the 13th Int. Conference on Digital Audio Effects (DAFx10)*, pages 78–86, September 2010.

[10] N. Schnell and M. Battier. Introducing composed instruments, technical and musicological implications. In *Proceedings of the 2002 conference on New interfaces for musical expression*, NIME '02, pages 1–5, Singapore, 2002. National University of Singapore.