Netpixl: Towards a New Paradigm for Networked Application Development

Dimitri Diakopoulos[1]
California Institute of the Arts[1]
24700 McBean Pkwy.
Valencia, CA USA
ddiakopoulos@calarts.edu

Ajay Kapur[1,2]
New Zealand School of Music[2]
P.O. Box 2332
Wellington, New Zealand
akapur@calarts.edu

ABSTRACT

Netpixl is a new micro-toolkit built to network devices within interactive installations and environments. Using a familiar client-server model, Netpixl centrally wraps an important aspect of ubiquitous computing: real-time messaging. In the context of sound and music computing, the role of Netpixl is to fluidly integrate endpoints like OSC and MIDI within a larger multi-user system. This paper considers useful design principles that may be applied to toolkits like Netpixl while also emphasizing recent approaches to application development via HTML5 and Javascript, highlighting an evolution in networked creative computing.

Keywords

networking, ubiquitious computing, toolkits, html5

1. INTRODUCTION

Within the NIME community, there is a broad body of work exploring the capabilities of phone, tablet, and embedded computing platforms. Though Mark Weiser's own vision of ubiquitous computing has not yet achieved adoption on the grand scale he originally envisioned [17], a proliferation of sensing and I/O technology afforded by such mobile devices have supported a wide range of novel creative experiences. Netpixl endeavors to assist developers scout new creative territory with a collection of simple networking tools guided by a lightweight server.

Netpixl targets a context schema that describes interactions that occur on a local, synchronous level. A good example of this schema may be found in any contemporary laptop, mobile-phone orchestra or within recent models of networked performance [16, 1]. Towards this idea of multi-user interaction, the central purpose of Netpixl is to present a device-agnostic channel for communicating arbitrary streams of data.

More specifically, Netpixl identifies and proposes a partial solution to the fragmentation of technologies and toolchains normally employed while developing apps across different device types. Too often, systems are limited in scope to a specific platform or programming language. Networked pieces are commonly written in a number of domain-specific languages (DSLs) such as ChucK, MaxMSP, CSound, or Supercollider and may use different network topologies to communicate data to fellow performers or exhibition participants. Similarly, mobile platforms have their own core language and protocols: Objective-C on iOS, Java on Android, both sharing the use of a full TCP/IP stack.

Creative developers interested in introducing components other than OSC endpoints to a piece or exhibition often find themselves writing glue-code or one-off servers. Recognizing this issue, the conception of Netpixl was driven by an idea to ease the developmental effort of systems where ubiquity among devices — sensors, mobiles, screens, speakers — is critical. In this scheme, Netpixl abstracts and limits the amount of boilerplate networking code necessitated by a new application. At the same time, Netpixl remains flexible to the implementation language or environment of the user or audience-facing app so long as it can handle any one of Netpixl's supported protocols: serial, MIDI, OSC, Websocket, or MQTT.

By organizing a system around a generic publish/subscribe mechanism, Netpixl was conceived as a central protocol-agnostic router. But in what language or form should this idea take? One of the primary ideological motivations of Netpixl has been the adoption of HTML5 standards and the promotion of Javascript to a first-class citizen in many mobile browsers. Discussed further in the next section, native applications and boilerplate OSC networking code are no longer prerequisites of using mobile devices as expressive computing platforms. To concretely summarize, Netpixl meshes this application development paradigm (HTML5 and Javascript) with traditional networking methods between actors of interactive and ubiquitous systems.

The remainder of this paper is divided into three primary parts. Section 2 summarizes notable technologies, techniques, and projects in browser, networking, media computing, and toolkit areas. Section 3 centers on describing some of the design principles of the toolkit. Section 4 follows with an exposition of the technical implementation and summary of the developer-facing API. Section 5 concludes with several forward-looking ideas for the project. At a high level, this paper follows an arc of principles for toolkit design as a template for future innovation rather than a deep discussion of Netpixl's technical affordances.

2. RELATED WORK

Netpixl synthesizes concepts from a number of areas including mobile toolkit design, Javascript framework architecture, and the internet of things movement. This section is designed to contextualize projects inspired by these concepts in relation to several core themes of Netpixl: developmental simplicity, data independence, and platform agnosticism. In basic form, the following works travel up and down the ladder of programmatic abstraction: some are toolkits, others frameworks, and yet more applications. This section emphasizes the ideation of creative ideas within both sound and music computing and computational design.

2.1 Mobile

The capabilities and proliferation of mobile devices continue to fulfill Mark Weiser's idea of ubiquitous computing as evidenced by their recurring use in an emerging number of collaborative task and creative-oriented environments [17]. One of the most visible results of this in the music community has been the development of a number of mobile phone orchestras, the mobile analogy of a digital music ensemble or laptop orchestra [10, 15]. As these ensembles have matured, several open-source toolkits have been released to evangelize the concept.

MoMu is one of the first toolkits to offer basic abstractions for audio applications and was developed by the Stanford MoPho group [3]. Developed as an iOS application framework, MoMu offers a means of unifying sensor access, creating a basic graphics context, and enabling networked interactions via OSC. UrMus is another recent iOS toolkit, with a general focus on evented interaction for UI widgets [5]. Both projects afford the notion of developmental simplicity for ideation and prototyping, though at the cost of being tied to iOS.

Among creative-oriented toolkits with server components, Control is a narrowly-focused example which offers an API centered on UI [12]. Designed for programmatic interface consumption on mobile devices, Control couples with prebuilt message-routing backends in SuperCollider and Max. This project uses the notion of a 'black-box' to extend widget functionality in the UI, a strategy used extensively in Netpixl. Though Control lacks the full-application focus of UrMus or MoMu, it is written in Javascript but bundled in an iOS wrapper.

Weitzner et al introduces a toolkit using a client-server model in [18] entitled massMobile. Designed to facilitate large-scale audience interaction using smart phones in an exhibition or performance format, massMobile uniquely provides asynchronous communication utilities for reliability in high-latency environments such as the internet.

2.2 The Browser

While the web has been acknowledged for a long time as an emerging CSCW platform [2], synchronous real-time networking in the browser has been an unmet challenge until recently. Shifting focus away from native, the following discussion illustrates the growing force of Javascript and HTML5 to compete with the performance and networking capabilities of native apps.

Since early 2010, several methods of near real-time communication have achieved adoption across browsers, for varying degrees of âĂŸreal-timeâĂŹ. Gutwin et al. present a performance overview of these methods, finding the HTML5 Websocket protocol being the most performant and reliable [6]. Websockets resemble standard TCP/IP networking sockets in functionality, freeing networking schemes similar to OSC from the clutches of evil and time consuming native applications. One caveat of this approach is that no current browsers permit UDP. In effect, the use of TCP adds a minimal but slightly perceptible latency depending on the use-case.

The use of Websockets in published literature is limited, although effectively documented in several creative experiments on the internet. However, one end-to-end application with an idea very similar to the use-cases driving Netpixl can be found in Patchwerk. Patchwerk is a multiuser HTML5 UI that communicates with a control-voltage (CV) module in a massive modular synthesizer [8]. Beyond Patchwerk, there are an emerging number of multiplayer games and music environments, as seen in work like Rumpetroll[1], Plink[2], BrowserQuest[3], and Technitone [4].

2.3 Toolkits

Tools and toolkits supporting creativity are one of the grand challenges of HCI researchers [13]. In practice, these challenges are being met by a growing number of creative coding frameworks in Javascript, including Processing.js, Paper.js, and Create.js. These types of client-side frameworks, like the earlier mobile toolkits, boost developer's efficiently in executing interactive ideas quickly.

Returning to the earlier idea of glue, many ideas require the execution of code that communicates between the physical and the digital. Orbiting physical-computing based controllerism as seen in the NIME community, the internet of things movement (IoTM) epitomizes the growing creative applications of ubiquitous computing. IoTM has brought several valuable concepts to the idea of physical devices marrying networked interactions. Using a host of protocols, server-based services like COSM[5], Sen.se[6], and Nimbits[7] offer methods of aggregating, visualizing, and interacting with embedded sensors and actuators. From a toolkit perspective, Netpixl borrows inspiration from this 'everything connected' concept but re-focuses on helping developers achieve synchronous interactions between the users behind those devices.

Finally, the last toolkit meriting attention is Spacebrew[8]. Coming to fruition after the first Netpixl prototype, this application conceived by the Lab at Rockwell Group shares many novel similarities with Netpixl. Spacebrew takes a friendly approach to networking devices within an interactive space by presenting an HTML5 GUI for connecting publishers and subscribers to the Spacebrew server. Spacebrew differs in that functionality is designed solely around Websocket communication. As discussed in the next section, Netpixl was more reserved in its assumptions about developers and more liberal about the need to exchange data across additional protocols like OSC and MIDI.

3. DESIGN PRINCIPLES

As networked art evolves, toolkits serve to compartmentalize certain aspects of technical implementation. For lone-developers or small teams, creative process and implementation are strongly coupled. This section is founded on the thesis that reusable code-level tools can unburden artists in their creative pursuits. In many cases, the practical networking aspects of a ubiquitous system may be considered a developmental tax. Netpixl is an experiment in providing an economy of features with low tax, reducing the technical burden on creative developers. These efforts are supported by the notion that developers can write code in a common language across the client and server. Furthermore, the Nextpixl toolkit is founded on the premise that boilerplate serverside networking code can be dramatically reduced.

Ian Hattwick and Marcelo Wanderley have presented an abstract explorer for evaluating collaborative systems, effectively documenting their most critical attributes [7]. Their taxonomic structure is based on a graph with the following points: Texture, Equality, Centralization, Physicality, Syn-

[1]`rumpetroll.com/`
[2]`labs.dinahmoe.com/plink`
[3]`browserquest.mozilla.org`
[4]`technitone.com`
[5]`cosm.com/`
[6]`open.sen.se/`
[7]`nimbits.com/`
[8]`spacebrew.cc`

chrony, and Dependence. These points were used as guiding technical design principles for the toolkit.

Before diving into the goals themselves, the abstract of this paper refers to a micro-toolkit rather than a toolkit proper. This in itself is a design principle which drove the narrowly-defined networking focus of the toolkit. An exemplar of this scope-limited functionality can be seen in the 'flexibility-usability tradeoff' design principle. This principle is explained as:

The flexibility-usability tradeoff is a design principle maintaining that, as the flexibility of a system increases, its usability decreases. The tradeoff exists because accommodating flexibility requires satisfying a larger set of requirements, which results in complexity and usability compromises (Wikipedia's definition per [14]).

Complementary to this principle is the idea of *featuritis*, where designers or developers tend to "emphasize the number or novelty of features over core usability [4]." Usability guru Don Norman argues that as functionality increases, mental and physical clutter can severely hinder usability [9]. The API of Netpixl was conceived with these principles in mind as they attempt to decouple implementation mental-clutter from creative process. Finally, the toolkit adheres to Postel's principle, which states: Be conservative in what you do, be liberal in what you accept from others. This principle is more visible from the API documentation in the next section.

3.1 Goals

The principles we have just discussed may be expressed through five high-level design goals inspired by, and partially derived from, Resnick et. al's, "Design Principles for Tools to Support Creative Thinking [11]." The intent of each goal is was to inform the technical implementation of the tools within Netpixl:

Support Iteration and Exploration. Creativity support tools should emphasize the importance of designing through prototypes. For a developer using a toolkit, this means being able to easily reconfigure data models and mappings, change the direction of the concept, and create new versions without re-architecting large parts of the program. Resnick et al. call this goal *tinkerability.* In practice, this translates to an easily graspable API for adding input/output targets and making no assumptions about the type of data.

Keep Developers Unburdened. An unburdened developer is able to achieve a creative idea more quickly without being faced with common networking problems like synchronicity and protcol formats. The abstraction of this idea can be sesen as a black-box development model, where certain aspects of the toolkit work as if by 'magic'. A functional toolkit should also provide extensibly to this model, though the constraints of added complexity should be well noted to the developer.

Encapsulate Domain Knowledge. Domain knowledge can refer to a number of concepts with varying specificity. In this context, domain knowledge encapsulates the client-server model for distributed performance and exhibition. A developer should have access to ready-made abstractions for this model and be provided with extensibility for protocols within their own domain (e.g. OSC or MIDI).

Speak a Common Language. Javascript is the primary language of the web, supported by all major browsers, and the target of an immense number of libraries, building on everything from user-interface design to machine learning. Approaching Javascript as a developmental platform harnesses a widely-supported language that can used both serverside and clientside.

Enhance Creativity. By managing the complexity, a toolkit should enhance overall creativity. Removing the mental clutter of implementation glue and suppling the fundamental building blocks of collaborative applications, a new language cloud emerge to describe future creative solutions in this domain.

4. IMPLEMENTATION AND API

This section is centered on providing preliminary documentation about the Netpixl server-API and its features for kickstarting multi-user application development. Netpixl is implemented via Node.js[9], a framework built on top of Google's V8 Javascript engine. Node.js encourages asynchronous code, a perfect building block for building high-volume messaging applications. The Netpixl distribution contains a Node.js seed project from which the API can be accessed through a singleton controller object created at runtime.

4.1 Serverside API Design

The unified API can be seen as a collection of functions within three tasks. These tasks can be classified as: i) data translation across protocols, ii) defining models that can be published to and subscribed from, and iii) maintaining state across devices. A fourth meta-API includes common utilities for creative coding alongside debugging functions. Though Netpixl is not a 'grab-and-go' solution for networking, the idea is that developers may rapidly assemble functionality out of these basic components.

4.2 Messaging API

The central promise of the Netpixl toolkit is that it affords an approachable model of message synchronicity across devices. While employing Node.js removes some problems around message volume and scalability, a developer is still left with the task of consuming, saving, routing, and sending events. Netpixl does not dismiss the need for a little networking glue, but is rather designed to drastically reduce it according to our five design goals.

One of Netpixl's core APIs features the ability to consume messages from many sources and formats. The message API is built on functions for consuming and producing messages through a variety of protocols: serial, OSC, MIDI, MQTT, and Websockets. Netpixl wraps Node.js libraries for these protocols and smooths out their individual API inconsistencies with standardized sender and reciever objects. Messages consumed by receivers are converted to a common format and sent to the Netpixl router through default (but developer-extensible) mappings.

4.3 Pixls

In Netpixl, shared models are defined on the server in as a collection of Pixls, the fundamental unit of computation created for the toolkit. Events published a server are converted to an intermediary message structure and consequently routed to a related Pixl. Pixls are defined by an application developer and may represent an attribute of data (integer, float, boolean, string, array) or an endpoint (representing a task). In more formal terms, this model is a combination of publish/subscribe and RPC (remote procedure call). By default, Netpixl implements an automatic pub/sub mechanism where all messages are broadcast to a current list of clients.

Pixls follow a black-box methodology of extension that is invisible to developers until needed. The purpose of

[9]`nodejs.org/`

extending functionality is to help developers build server-side tasks, a more complex but useful addition to Netpixl's message routing. The most useful extension of the Pixl API is the preprocess function. The preprocessor can be used in conjunction with the utility API to transform, filter, limit, scale, or compare data before it is committed to the Pixl model and published to clients. The flow of data through a Pixl is given in Figure 1.

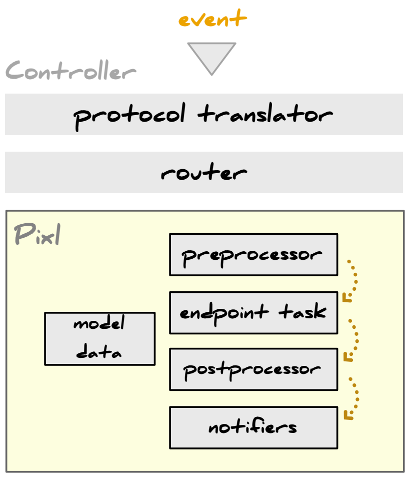

Figure 1: Flow of data through Netpixl

4.4 Synchronization API

Synchronicity is a built-in attribute of the Netpixl toolkit. That is, all events are published in real-time to connected clients. Synchronization refers to the present state of the model data across two or more entities. When a client connects to the application, it may be desirable to inflate the local model with the present model of Pixls. This API assists in serializing an entire Pixl collection for easy transmission to a client. This API also permits user-defined functions to synchronize the model to a persistent store such as a database.

4.5 Utility API

Finally, Netpixl presents methods to encapsulate actions regularly seen in creative coding and web application development. These functions were designed to be used in conjunction with the preprocessors and postprocessors. This API contains methods for scaling, filtering, clamping, and interpolating numerical data. Low-resolution timers are also available using Node's built-in timer classes, permitting periodic function calls. Lastly, the utility API contains a robust logging foundation to assist with application debugging. The logging functionality also includes a basic HTML5 GUI to visualize the flow of messages through the Netpixl routing core.

5. FUTURE WORK AND CONCLUSION

Like many toolkits, Netpixl is in constant state of flux. One often neglected area of toolkit development is API documentation, which will be a priority as the toolkit moves toward a stable release. Similar to IoTM toolkits, another area we may expand upon is an automatically generated REST API for Pixl data. REST APIs transact data through basic HTTP requests, permitting data transactions outside pub/sub.

In conclusion, Netpixl is a micro-toolkit employing five design goals designed to empower the developers of networked pieces, exhibitions, and experiments. This toolkit reduces the amount of glue needed to communicate among devices and evangelizes the growing Javascript + HTML5 development paradigm. For coders with artistic and whimsical ideas, Netpixl affords easier methods of interaction exploration and rapid prototyping to enhancing creativity and guide new ideas and experiences to reality.

Netpixl is free and open-source software and may be found at `https://github.com/ddiakopoulos/netpixl`.

6. REFERENCES

[1] A. Barbosa. Displaced Soundscapes: A Survey of Network Systems for Music and Sonic Art Creation. *Leonardo Music Journal*, 13:53–59, 2003.
[2] R. Bentley, T. Horstmann, and J. Trevor. The world wide web as enabling technology for cscw: The case of bscw. In *Computer Supported Cooperative Work*, 1997.
[3] N. Bryan and J. Herrera. MoMu: A mobile music toolkit. In *New Interfaces for Musical Expression*, pages 174–177, 2010.
[4] A. Chang, J. Gouldstone, J. Zigelbaum, and H. Ishii. Simplicity in interaction design. In *Tangible and Embedded Interaction*, page 135, New York, New York, USA, 2007.
[5] E. Georg. Urmusâ an environment for mobile instrument design and performance. In *International Computer Music Conference*, 2010.
[6] C. Gutwin, M. Lippold, and T. Graham. Real-time groupware in the browser: testing the performance of web-based networking. In *Computer Supported Cooperative Work*, pages 167–176, 2011.
[7] I. Hattwick and M. M. Wanderley. A Dimension Space for Evaluating Collaborative Musical Performance Systems. In *New Interfaces for Musical Expression*, 2012.
[8] B. Mayton, N. Joliat, and J. A. Paradiso. Patchwerk : Multi-User Network Control of a Massive Modular Synthesizer. In *New Interfaces for Musical Expression*, 2012.
[9] D. A. Norman. *The Design of Everyday Things*. Basic Books, Inc., New York, NY, USA, 2002.
[10] J. Oh and et al. Evolving the mobile phone orchestra. In *New Interfaces for Musical Expression*, 2010.
[11] M. Resnick, B. Myers, K. Nakakoji, B. Shneiderman, R. Pausch, T. Selker, and M. Eisenberg. Design principles for tools to support creative thinking. In *Report of Workshop on Creativity Support Tools*, 2005.
[12] C. Roberts, G. Wakefield, and M. Wright. Mobile Controls On-The-Fly : An Abstraction for Distributed NIMEs. In *New Interfaces for Musical Expression*, 2012.
[13] B. Shneiderman. Creativity support tools: A grand challenge for hci researchers. In *Engineering the User Interface*, 2009.
[14] G. Steinebach, S. Guhathakurta, and H. Hagen. *Visualizing Sustainable Planning*. Springer, 2009.
[15] G. Wang, E. Georg, and H. Pentinnen. *The Mobile Phone Orchestra*. Oxford University Press, 2010.
[16] G. Weinberg. Interconnected Musical Networks: Toward a Theoretical Framework. *Computer Music Journal*, 29(2):23–39, 2005.
[17] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–95, 98–102, 104, 1991.
[18] N. Weitzner, M. St, A. Ga, J. Freeman, S. Garrett, Y.-l. Chen, and G. Tech. massMobile â An Audience Participation Framework. In *New Interfaces for Musical Expression*, 2012.