ChuckPad
Social Coding for Computer Music

Spencer Salazar[1,2]
ssalazar@calarts.edu

Mark Cerqueira[3]
mark@mark.gg

[1]Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, Stanford, CA 94305
[2]California Institute of the Arts, 24700 McBean Pkwy, Valencia, CA 91355
[3]721 Old County Road, Belmont, CA 94002

ABSTRACT

ChuckPad is a network-based platform for sharing code, modules, patches, and even entire musical works written on the ChucK programming language and other music programming platforms. ChuckPad provides a single repository and record of musical code from supported musical programming systems, an interface for organizing, browsing, and searching this body of code, and a readily accessible means of evaluating the musical output of code in the repository.

ChuckPad consists of an open-source modular backend service to be run on a network server or cloud infrastructure and a client library to facilitate integrating end-user applications with the platform. While ChuckPad has been initially developed for sharing ChucK source code, its design can accommodate any type of music programming system oriented around small text- or binary-format documents. To this end, ChuckPad has also been extended to the Auraglyph handwriting-based graphical music programming system.

Author Keywords

ChucK, computer music programming, social coding

ACM Classification

H.5.5 [Information Interfaces and Presentation] Sound and Music Computing, H.5.3 [Information Interfaces and Presentation] Group and Organization Interfaces—Collaborative computing.

1. INTRODUCTION

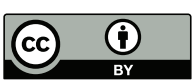
ChuckPad is a network-based platform for sharing code, modules, patches, and even entire musical works written on the ChucK programming language [17] and other music programming platforms. Overall ChuckPad has been created to solve a number of goals. The first of these is to provide a single repository and record of musical code from supported musical programming systems, bring together the miscellany of code spread variety of disparate online forums, email lists, version control systems, tweets, and other mechanisms. The second is to provide an interface for organizing, browsing, and searching this body of code. The third is to provide a simple means of evaluating the musical output of code in the repository, without necessitating the installation of new software or executing foreign and untrusted code.

To this end, ChuckPad consists of a modular backend service to be run on a network server or cloud infrastructure and a client library to facilitate integrating end-user applications with the platform. The server component of ChuckPad is further divided into a core data management layer, a web frontend for browsing shared code and integrating into conventional social media services, and a rendering framework for producing fixed-media previews of code in standard audio formats. The initial version of the client library has been developed in conjunction with miniAudicle for iPad [14], an iPad-based editor for ChucK.

While ChuckPad has been initially developed for sharing ChucK source code, its design can accommodate any type of music programming system oriented around small text- or binary-format documents. For instance, ChuckPad has been extended to the Auraglyph handwriting-based graphical music programming system [13]. A single instance of the ChuckPad server backend can support multiple distinct client applications and programming code languages, thus providing a single nexus for storage and presentation of musical code, allowing users to share identity and login information for sharing multiple code types, and enabling a single client application to access and utilize multiple code types. At the same time, ChuckPad, as open-source software, is designed such that it can be hosted independently of the authors' central public ChuckPad instance either as a private service or as an alternative public host.

2. BACKGROUND WORK

Until recently, wide area network-mediated systems for music programming have had a limited history. A number of music programming systems use network-aware techniques, such as Open Sound Control [19] or bespoke networking protocols, in their basic operation on a singular computer. These include SuperCollider [9] and ChucK [17], which, despite built-in networking capabilities, have been relatively rarely used for mediating wide-area performance or code sharing. Impromptu [15] and JITLib [2] include explicit support for synchronization over the network, with documented instances of performances utilizing these features. The CoAudicle [18] and miniAudicle for iPad [14] presented two environments for real-time collaborative programming of music using ChucK. Lee and Essl describe a variety of implementations and conceptual taxonomies for networking in live coding [8]. In particular they describe music programming frameworks that allow for networked code sharing, including Gibber [12] and Sketchpad [1], which all feature a real-time shared code editor, and their own extensions to UrMus, adding shared program state and shared text editing [7]. LOLC is a collaborative music programming



NIME'17, May 15-19, 2017, Aalborg University Copenhagen, Denmark.
.

system that synchronizes chat messages and programming commands from distributed digital performers into a unified performance dashboard [4]. Lich.js is a collaborative framework for music programming based on Web Audio, Web GL, and other web technologies [10]. Crowd in C[loud] mediated a real-time performance inspired by Terry Riley's *In C* between many audience members through an online website loaded onto their phones; various musical parameters of the software on each phone was controlled by on-stage performers using JavaScript code [6].

Most of the efforts described above have focused on real-time sharing of code. In the realm of asynchronous creative code sharing, archiving, and long-term collaborative development, Shadertoy is an online tool for developing, sharing, and viewing audiovisual software written in the OpenGL Shading Language [5]. sc140 compiled into an album a number of SuperCollider programs, originally distributed via Twitter and adhering to that service's 140 character limit for individual messages [16].

A number researchers have explored the nature of social coding in general purpose programming frameworks. Dabbish et al. explored the effects of the social coding site GitHub on collaboration, awareness, community, and self-education in software development [3]. Pham et al. documented social practices developed by software creators on GitHub [11].

3. DESIGN

At its core, ChuckPad is a server backend system structured around individual document resources and an external interface based on Representational State Transfer (REST). ChuckPad does not actually care about the content within a document resource; it is completely agnostic to the underlying file format, so it can easily store, for instance, ChucK scripts, PureData patches, PCM audio files, or any other type of singular document resource. For instance, while miniAudicle for iPad stores textual ChucK code documents, Auraglyph stores textual JSON files corresponding to its own serialization format.

A few terms that will be used in describing the system merit brief description.

- The ChuckPad ***server*** is the generic server component of ChuckPad.
- The ChuckPad ***client*** refers to a generic client, a reference client, or the application-agnostic client-side library (exactly which can be inferred from context but is generally not significant in this discussion).
- A ChuckPad ***client application*** is a specific client implementation of the ChuckPad, for instance, miniAudicle for iPad. A client application typically builds on the generic client-side library, but will not usually need to customize the ChuckPad server component.
- A ChuckPad ***document resource*** is an instance of the thing being stored, e.g. a ChucK script, an Auraglyph patch, etc. The exact nature of the document resource will obviously vary depending on the client application, but in general the generic ChuckPad client and server components will not need to concern themselves with these details.

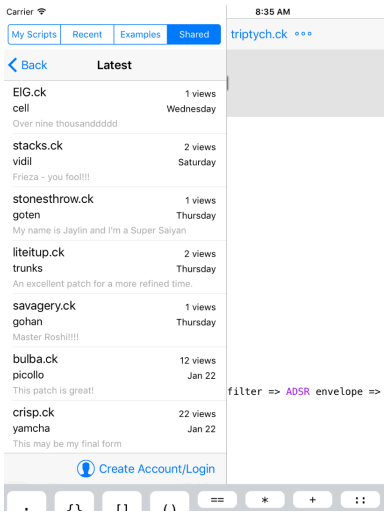


Figure 1: Browsing a list of recent documents.

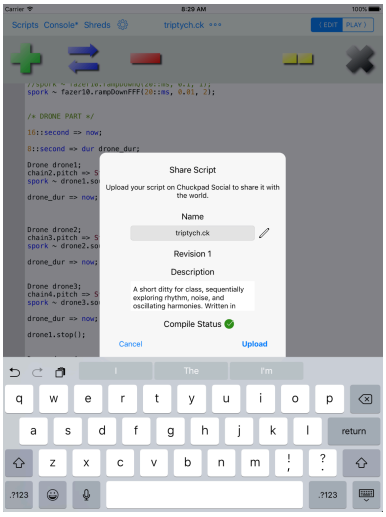


Figure 2: Uploading a new document.

Each document resource is ascribed to the user who uploaded it, and ChuckPad also tracks metadata such as a title, description, time and date of creation, and time and date of last modification. Arbitrary unstructured metadata can also be stored with each document. This is intended for client applications who need to store additional metadata with document resources that is specific to a particular client application. For instance, Auraglyph uses this to store a visual description of Auraglyph programs that is displayed to users while they are browsing a list of programs available on the service. ChuckPad also tracks how many times each document reosurce has been downloaded, which is used as a rough popularity metric.

A ChuckPad user is able to browse the repository of ChuckPad scripts through several filters. A "Popular" filter lists documents in order of recent popularity and a "Recent" filter lists in descending order of creation date (Figure 1). The "Documentation" filter shows documentation and example code and "My Scripts" shows the users own scripts they have uploaded if they are logged in. A typical client application will allow the user to select which filter they would like to explore and then pull the list of documents corresponding to that filter from the server. Management of how each list is formed and what documents are in each list is handled by the server and is not dependent on the actual format of the document.

After the user selects a document, the client application then downloads the full contents of the document and loads

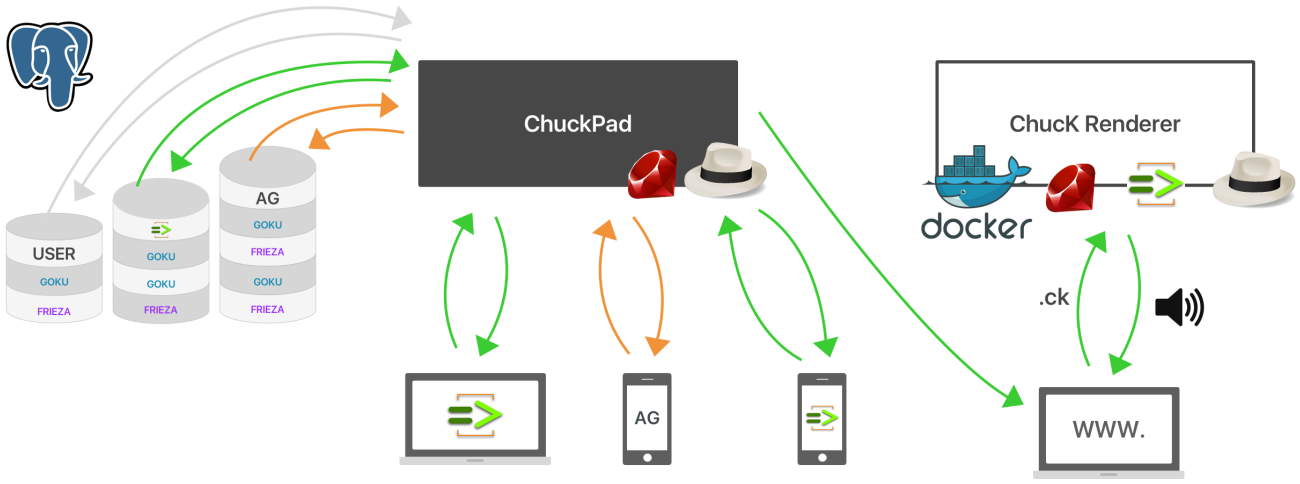


Figure 3: The ChuckPad technical architecture.

that for viewing and execution. Through the editing functions of the client application, the user can also change the downloaded script to experiment with different modifications. These modifications can't be saved to the original base document, but the user can duplicate the document and upload the modified version under their own account. If a user determines that a document is somehow abusive—for instance, it intentionally attempts to crash the client application or to access unauthorized local resources—they can mark it as such, which will cause the server to flag the document for review by the server administrators.

A user who has logged in through the client application can upload documents that will be associated with their account (Figure 2). Users who do not have an account or are not logged in cannot upload documents. Before uploading, the client application can allow the user to change the document title and add a text description. A client application should typically also perform some degree of integrity checking on the document before uploading it; for instance, miniAudicle for iPad ensures that ChucK scripts compile without errors before uploading them. Newly uploaded scripts will automatically be added to the global document repository, immediately showing up in the "Recent" filter and the user's "My Scripts" filter. When uploading, the client application can also indicate if the document was derived from another document in the service and, if so, which document. This parent–child relationship between uploaded documents is tracked by the server, although it is not displayed in client applications or the web-based frontend (we believe it may be useful in the future for analyzing the relationships between code documents and displaying these analyses to users). Users have the option to create a new login username and password through the client application, or to sign in with existing credentials.

A single ChuckPad server can host file types that are associated with more than one client application. For instance the ChuckPad server can currently host ChucK files and documents for the Auraglyph programming environment. A client application can choose which document types it wishes to access, and can display and work with multiple document types if it is capable of doing so. User identity extends to the entire server, so a single username and password will work for any client application and document type that uses the same backend host. Currently, ChuckPad has been extended to support both ChucK documents and documents for the Auraglyph handwriting-based audio programming environment.

The web frontend of ChuckPad is designed to be a completely frictionless experience for a person looking to explore the data that exists on the ChuckPad server. Users visiting `www.chuckpad.io` can access the same library of data a user using a ChuckPad client can, but can do so without installing any additional software on their device. The only requirements to fully take advantage of this medium is a web browser capable of displaying HTML and playing AAC files—assumed capabilities on any modern web browser.

To make this web experience truly complete for a user, the web frontend coordinates taking a document resource, rendering it to fixed media, and playing the output to the user. User browsing ChucK patches on `www.chuckpad.io` can see a list of patches uploaded by users and for each patch, can play an audio preview of that patch. This allows a user to listen to a ChucK patch on their computer without ever compiling or running a single ChucK file. This site also links to miniAudicle, which allows users to modify and run patches locally. This lets curious would-be creative coders experience ChucK easily.

4. IMPLEMENTATION

Figure 3 illustrates the backend architecture of ChuckPad. The ChuckPad server is written in Ruby with the Sinatra framework, which provides a very lightweight scaffold for building web-based systems. Data is stored in a PostgreSQL database. miniAudicle and Auraglyph allow patch data (i.e. the main document resource and auxiliary metadata) to be 20 KB in size.

The primary roles of the ChuckPad server are user management, patch management, and the web frontend. The user API allows for creating users, logging in, logging out, and resetting passwords. The patch API allows for put operations like creating patches, updating patches, and deleting patches. It also support operations like getting recent patches, documentation patches, a user's own patches, and a list of patches for any user. Patches can also be set to hidden which makes them invisible to all users except the user who created the patch; this functionality can be leveraged for users to upload a draft of a patch that they do not wish to publish yet. Later on, a user can update a patch to be visible to all users. Patches reported as abusive attach the abuse report count to their metadata and this is returned in all requests so client applications can proactively chose to block showing those patches.

The ChuckPad server allows supporting multiple client

applications. A user can log into different client applications using the same credentials with a given server. Resources uploaded from different apps are stored with an identifier in the patches table; any patch requests from a ChuckPad client properly associate with the type of resource that the requesting app can handle. The benefits of a single server supporting multiple client applications include the sharing of user credentials, simplifying operations administration by allowing maintenance of a single server for multiple applications, and allows for cross-promotion of client applications using the server.

The ChuckPad ChucK Renderer is a Docker image that configures a Linux operating system to run a simple rendering server. This image includes both ChucK and general-purpose audio encoders. The server can receive code, compile it with ChucK, and output the audio result to a WAV file. Using FFmpeg the WAV files are converted to AAC files, reducing the file size while maintaining audio quality losses within acceptable levels. The renderer also hashes all source code and stores that information alongside the final AAC output; when a request that has been synthesized already hits the renderer, the server is able to simply and quickly serve the AAC output.

5. CONCLUSIONS

ChuckPad is a tool to allow ChucK programmers to easily archive and share their software written in ChucK, and to browse shared programs written by others. Its goals are to promote dissemination of interesting ChucK code, provide resources for learning about and experimenting with ChucK, and build a community around the sharing of creative code. ChuckPad's modular open-source architecture facilitates the creation of similar services for other programming languages and tools.

The source code for ChuckPad is available online at the following URLs:

- Core ChuckPad server component: `https://github.com/markcerqueira/chuckpad-social`
- ChuckPad renderer: `https://github.com/markcerqueira/chuck-renderer`
- ChuckPad iOS Client Library: `https://github.com/markcerqueira/chuckpad-social-ios`

5.1 Future Work

The authors have considered a number of features that can be added to enhance the social experience of ChuckPad. The ability to follow users, rate patches, comment on patches, and personalize feeds of code could further enhance the sense of community around the content being produced and curated. Future development on the back-end infrastructure could add support for additional client applications as well as multi-file support, allowing for more elaborate (and modular) ChucK programs.

6. ACKNOWLEDGMENTS

Many thanks to Kirill Zhukov for code reviews, Tom Lieber for Ruby advice, and Ji-hern Baek for figure design. We would also like to thank all the open source contributors that built the components that power ChuckPad.

7. REFERENCES

[1] Sketchpad. `http://sketchpad.cc/`.

[2] N. Collins, A. McLean, J. Rohrhuber, and A. Ward. Live coding in laptop performance. *Organised Sound*, 8(3):321–330, 2003.

[3] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM, 2012.

[4] J. Freeman and A. Van Troyer. Collaborative textual improvisation in a laptop ensemble. *Computer Music Journal*, 35(2):8–21, 2011.

[5] P. Jeremias and I. Quilez. Shadertoy: Live coding for reactive shaders. In *ACM SIGGRAPH 2013 Computer Animation Festival*, pages 1–1. ACM, 2013.

[6] S. W. Lee, A. D. de Carvalho Jr, and G. Essl. Crowd in c[loud]: Audience participation music with online dating metaphor using cloud service. In *Web Audio Conference*, 2016.

[7] S. W. Lee and G. Essl. Communication, control, and state sharing in networked collaborative live coding. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2014.

[8] S. W. Lee and G. Essl. Models and opportunities for networked live coding. In *Live Coding and Collaboration Symposium*, volume 1001, pages 48109–2121, 2014.

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

[10] C. McKinney. Quick live coding collaboration in the web browser. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 379–382, 2014.

[11] R. Pham, L. Singer, O. Liskin, F. Figueira Filho, and K. Schneider. Creating a shared understanding of testing culture on a social coding site. In *2013 35th International Conference on Software Engineering*, pages 112–121. IEEE, 2013.

[12] C. Roberts and J. Kuchera-Morin. Gibber: Live coding audio in the browser. In *Proceedings of the International Computer Music Conference*, 2012.

[13] S. Salazar and G. Wang. Auraglyph: Handwritten computer music composition and design. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 106–109, 2014.

[14] S. Salazar and G. Wang. miniAudicle for iPad: Touchscreen-based music software programming. In *Proceedings of the International Computer Music Conference*, 2014.

[15] A. Sorensen. Impromptu: An interactive programming environment for composition and performance. In *Proceedings of the Australasian Computer Music Conference*, 2005.

[16] D. Stowell. sc140. `http://supercollider.github.io/community/sc140.html`, 2009.

[17] G. Wang, P. R. Cook, and S. Salazar. Chuck: A strongly timed computer music language. *Computer Music Journal*, 39(4):10–29, 2015.

[18] G. Wang, A. Misra, P. Davidson, and P. R. Cook. CoAudicle: A collaborative audio programming space. In *Proceedings of the International Computer Music Conference*, 2005.

[19] M. Wright and A. Freed. Open Sound Control: A new protocol for communicating with sound synthesizers. In *Proceedings of the International Computer Music Conference*, 1997.