CART451_F2022_project_prototype_illiez

Project Prototype

Description

Nodechestra is a participative virtual synthesizer, harnessing real-time web communication to engage multiple users in collaborative online sound-making. This project is inspired by recent pandemic times, as well as by the seemingly ineluctable societal progression of virtualization, existing as an attempt to create space for collective online play.

Nodechestra is a Max/MSP patch played by users in their web browser via a Node application.
The application provides an HTTP server built using Express, and implements WebSocket to enable bidirectional communication between server and client. Following the initial HTTP handshake, the client sends data generated by UI interaction to the server, which pipes the data to Max/MSP via the relevant Node API and a Max Node scripting object. This data is then routed to the Max/MSP synthesizer, modulating its parameters.

The synthesizer's function is broken down into a set number of parameters (voice frequency, envelope characteristics, reverb, etc..), which are semi-randomly parceled out to various client pages. The user triggers parameter changes using interactive elements of the served page, the nature of which depends on their attributed parameter.

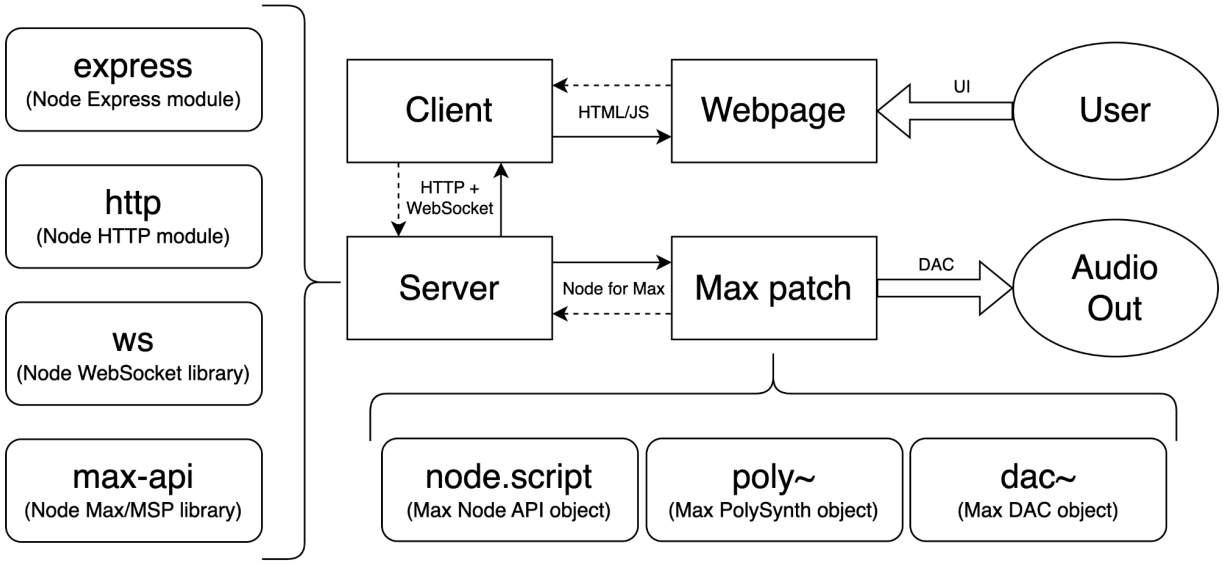
Staging

Network
The basic network architecture, consisting of a NodeJS Express application harnessing HTTP and WebSocket servers and implementing a Max/MSP API, is complete.
The WebSocket component of the network is functional, enabling full-duplex communication between server and client. However, the client activity is yet to be parceled out to different sockets. This could be achieved using the currently implemented Node library (ws) but the final iteration will most likely use Socket.IO, a framework built on top of the WebSocket protocol. This remains to be done.

UI
The basic skeleton of the client UI is complete, enabling user interaction with the page via HTML input elements (buttons, range, checkbox). Styling of the client page remains to be done.
The client script is functional, but could benefit from revisiting data transmission format, such as encapsulating user input in JSON format, more easily readable by Max/MSP's dict object.
Furthermore, the project would be enhanced by a greater diversity of user interaction, such as uploading files (e.g. images) and webcam tracking (e.g. browser-based ML APIs), which remains to be explored.

Patch
The basic functionality of the Max/MSP patch is complete, enabling reception of data sent from the Node application server (via the Node for Max component) and synthesizer modulation.
The synthesizer component of the Max patch remains to be finished (70% done), debugged (message routing, load state, reset triggers) and cleaned up (subpatcher encapsulation, presentation styling).

Diagram



Components

Server : node between *client* and *Max patch*

- *express* : Node module for Express, a web framework for Node.
  Implementation creates the web application framing the server and routing
- *http* : Node built-in module for HTTP, a protocol for networked data transmission,
  Implementation creates an HTTP server for the Express application, allowing Node to transfer data over the Hypertext Transfer Protocol
- *ws* : Node module for WebSocket, a protocol for a full-duplex communication channel over a single TCP connection.
  Implementation enables bidirectional communication between Node server and client.
- *max-api* : Node.js module for Max/MSP, a multimedia visual programming software.
  Implementation establishes an API between Node server and Max/MSP patch

Client : node between *server* and *user,* via *webpage*

- Displays webpage with interactive UI elements to user
- Listens for element value changes (i.e. user interaction) on webpage
- Sends changed values to server

Max patch :

- *node.script* : Max/MSP Node API object, runs specified Node script
  Implementation allows control of Node process from Max/MSP
- *poly~* : Max/MSP Polyphonic object, enables polyphonic signal processing
  Implementation creates the backbone of synthesizer modulated by users
- *dac~* : Max/MSP DAC object, converts a digital audio signal into an analog one
  Implementation enables sound output from the patch.

Functionality

The HTTP and Express components are fully functional, successfully creating a Node web application running a server accessed on a local port.

The WebSocket server component (ws) is functional, enabling full-duplex communication between server and client. This component needs to be modified to harness Socket.IO for more extensive WebSocket implementation, including socket tagging and function parcellation on connection, as well as disconnect handling.

The Node-Max/MSP API components (max-api & node.script) are functional, allowing data transfer from the server to Max patch. The data processing in Max/MSP would need to be reworked to better parse incoming server data as well as be able to send data back from the Max patch to the server.

The client network component is working, as mentioned above, enabling the transmission of client data to the server. However, the UI elements undergirding data transmission, currently simple HTML input triggering AJAX calls on change, need to be modified to fully exploit WebSocket functionality. This will be done concurrently with Socket.IO implementation.

Finally, the poly~ synthesizer component of the Max/MSP patch is functional, but could benefit from reworking to finish the development of certain parameters (e.g. filter, arpeggiator), fine-tune the integration of others (e.g. voice, waveform).

Instructions

1. Download prototype folder from GitHub
2. Download Max/MSP application (no need for subscription)
3. Open `n4m_synth.maxpat` file
4. Start Node script (click on checkbox)
5. Turn on DAC (click on speaker icon)
6. Launch webpage (click on ‘Launch webpage’ box)
7. Play around with controls on webpage
8. Adjust gain (if necessary)