Glow infrastructure design

Bastiaan Lichtenbelt

490894

Table of Contents

[1.1 Introduction 3](#_Toc187743225)

[2.1. Network topology 4](#_Toc187743226)

[2.1.1. Advantages 4](#_Toc187743227)

[2.1.2. Diagram 4](#_Toc187743228)

[3.1 General Overview Design 5](#_Toc187743229)

[3.1.1 Context Diagram 5](#_Toc187743230)

[3.1.2 Container Diagram 6](#_Toc187743231)

[3.1.3 Component Diagram 7](#_Toc187743232)

[4.1. Arduino (Client) 8](#_Toc187743233)

[4.1.1. Key Components 8](#_Toc187743234)

[4.1.2. Wiring Diagram 9](#_Toc187743235)

[4.1. Mac Mini Python Application 10](#_Toc187743236)

[4.1.1. Classes 10](#_Toc187743237)

[4.1.1.1 UDPServer 10](#_Toc187743238)

[4.1.1.2. OSCClient 10](#_Toc187743239)

[4.1.1.3. VolumeControl 11](#_Toc187743240)

[4.1.1.4 FadeManager 11](#_Toc187743241)

[4.1.2 Interactions 11](#_Toc187743242)

[5.1 Reaper 12](#_Toc187743243)

# Introduction

This design document outlines the concept and technical implementation of an interactive light and sound installation for the Eindhoven Glow festival. The installation features five acrylic pillars that respond to human touch through integrated distance sensors. Each pillar contains 16 sensors that detect interaction, triggering dynamic changes in lighting patterns and unique soundscapes.

The experience evolves as more sensors are activated. Individual interactions with the pillars produce distinctive visual and auditory effects, while full activation across all sensors creates a harmonious melody, symbolizing collective participation and unity. This installation aims to engage audiences through an immersive fusion of light, sound, and interactivity, showcasing the transformative power of collaborative art in public spaces.

# 2.1. Network topology

To ensure efficient communication and control across the installation, a star topology network will be implemented using Ethernet. In this configuration, a central hub or switch will serve as the core connection point, linking all five pillars to a centralized controller.

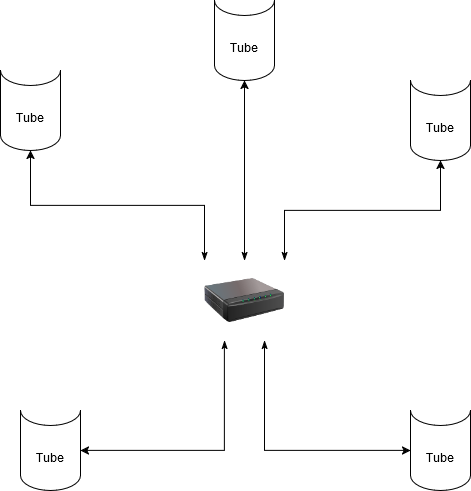
## 2.1.1. Advantages

Reliability: The central hub ensures that a failure in one pillar’s connection does not affect the others.

Scalability: Additional components can be added to the system with minimal disruption.

Performance: Dedicated connections between the hub and each pillar reduce latency and enable faster data transmission.

## 2.1.2. Diagram

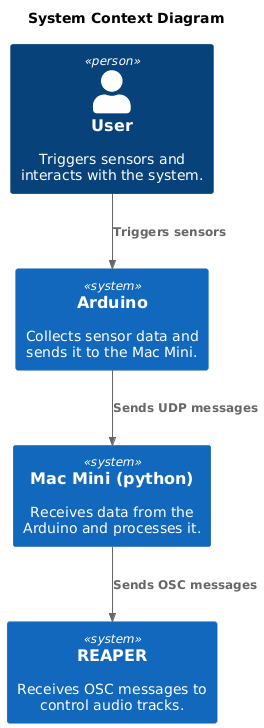


Each pillar will connect to the central hub via a wired Ethernet connection, ensuring robust and stable communication. The server that manages the entire system will be located in the base of one "main" or "central" pillar, serving as the central controller. This setup allows the central controller to efficiently manage sensor data, lighting patterns, and sound effects for each pillar in real-time. Furthermore, the use of Ethernet facilitates straightforward integration with other devices or networks if needed, enhancing the flexibility and expandability of the system.

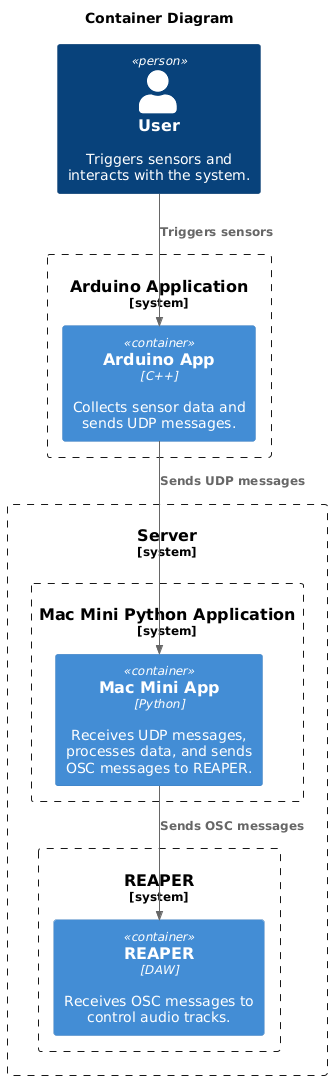
# 3.1 General Overview Design

This section provides an overview of the system design, detailing how components interact across the entire installation. Three diagrams—a System Context Diagram (C1), a Container Diagram (C2), and a Component Diagram (C3)—depict various levels of system architecture and interaction.

## 3.1.1 Context Diagram



## 3.1.2 Container Diagram



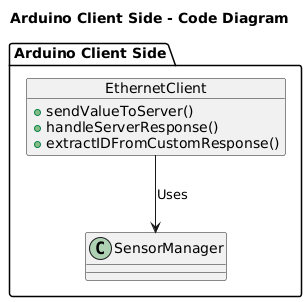
## 3.1.3 Component Diagram

A diagram of a server

Description automatically generated

# 4.1. Arduino (Client)

The Arduino-based client-side system for each pillar is designed to handle sensor data processing and communication with the central server. The design employs a W5500 Ethernet module for network connectivity, ensuring reliable and efficient communication.



## 4.1.1. Key Components

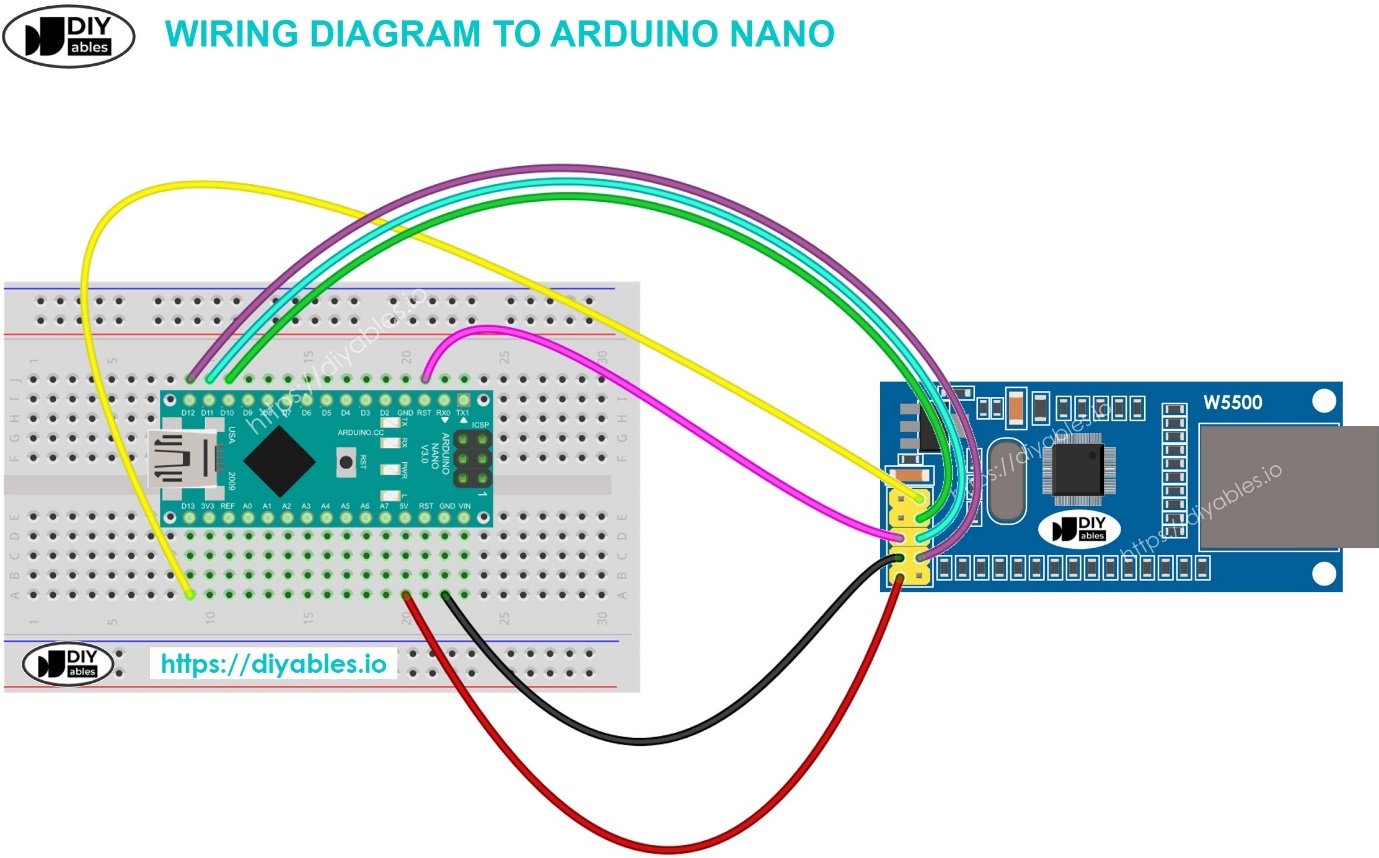
- W5500 Ethernet Module: Provides stable Ethernet connectivity for real-time communication between the pillar and the central server.

- SensorManager Functions: Functions that handle the data from the 16 distance sensors within the pillar, processing touch interactions and sending relevant data to the EthernetClient.

- EthernetClient Functions: Functions to manage network communication, including sending sensor values to the server, handling server responses, and extracting specific IDs from custom responses.

## 4.1.2. Wiring Diagram

Below is a diagram showing how to connect the Arduino Micro to the W5500 Ethernet module:



Arduino Micro Pins to W5500 Module Pins:

VCC (3.3V or 5V) to VCC on the W5500.

GND to GND.

MISO to MISO.

MOSI to MOSI.

SCK to SCK.

CS (Chip Select) to a configurable digital pin on the Arduino.

RST (Reset) to another configurable digital pin if used.

# 4.1. Mac Mini Python Application

A screenshot of a computer

Description automatically generatedThis section focuses on the software running on the Mac Mini, which is responsible for receiving sensor data, processing it, and controlling the audio playback in REAPER.

## 4.1.1. Classes

## 4.1.1.1 UDPServer

start\_Server(ip: str, port: int) -> socket: Starts the UDP server and listens for incoming connections.

listen(server: socket, volumes: list, client: udp\_client.SimpleUDPClient): Listens for incoming UDP messages containing sensor data. It processes the received data and updates the volumes list accordingly. It then notifies the VolumeControl class.

## 4.1.1.2. OSCClient

set\_reaper\_track\_volume(client: udp\_client.SimpleUDPClient, track\_number: int, volume: float): Sends an OSC message to REAPER to set the volume of the specified track.

mute\_track(client: udp\_client.SimpleUDPClient, track\_number: int, mute: bool): Sends an OSC message to REAPER to mute or unmute the specified track.

fade\_in(track\_id: int, original\_volume: float, duration: float, client: udp\_client.SimpleUDPClient): Sends OSC messages to REAPER to fade in the specified track over the given duration.

cross\_fade(tracks\_in: list, tracks\_out: list, duration: float, client: udp\_client.SimpleUDPClient, track\_in\_vols: list, track\_out\_vols: list): Sends OSC messages to REAPER to crossfade between the specified tracks over the given duration.

## 4.1.1.3. VolumeControl

pillar\_activation(track: int, activation: int, volumes: list, client: udp\_client.SimpleUDPClient): Processes the activation level of a specific pillar and updates the corresponding volume in the volumes list. It then determines if any fading or other audio adjustments are required and notifies the FadeManager or OSCClient accordingly.

background\_music\_check(levels: list, volumes: list, client: udp\_client.SimpleUDPClient): Monitors the overall activation levels of all pillars and adjusts the background music volume accordingly.

should\_fade(track\_number: int, fade\_type: str) -> bool: Determines whether a fade operation should be performed for the specified track based on the current activation levels and the desired fade type (e.g., fade in, fade out, crossfade).

## 4.1.1.4 FadeManager

fade\_in(track\_id: int, original\_volume: float, duration: float, client: udp\_client.SimpleUDPClient): Manages the fade-in operation for a specific track by coordinating with the OSCClient.

cross\_fade(tracks\_in: list, tracks\_out: list, duration: float, client: udp\_client.SimpleUDPClient, track\_in\_vols: list, track\_out\_vols: list): Manages the crossfade operation between the specified tracks by coordinating with the OSCClient.

## 4.1.2 Interactions

The UDPServer listens for incoming sensor data and notifies the VolumeControl class.

The VolumeControl class processes the sensor data, updates the volume levels, and determines if any fading or other audio adjustments are required.

The VolumeControl class interacts with the FadeManager to handle fade operations.

The VolumeControl and FadeManager classes use the OSCClient to send commands to REAPER to control the audio playback.

# 5.1 Reaper

Open Sound Control (OSC) is used to communicate with REAPER, enabling real-time control over audio parameters. A Python application on the Mac Mini acts as an OSC client, sending messages to REAPER to adjust volume, mute/unmute tracks, and trigger fade effects. REAPER is configured to receive these OSC messages and execute the corresponding audio commands, creating a dynamic and interactive audio experience.

* Go to **Options > Preferences > Control Surfaces** in REAPER.
* Click **Add** to create a new OSC control surface.
* Give it a descriptive name (e.g., "Glow Pillar Installation").
* **Crucially:** Enable both "Allow binding messages to REAPER actions and FX learn" and "Allow receiving messages from the device."