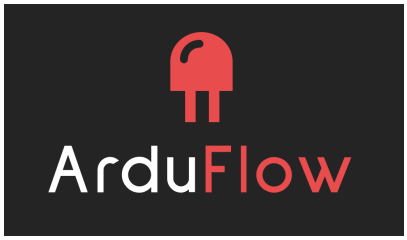

Personal Project Architecture

Michał Raczkowski
OL S6
4465024

version 0.2v

Contents

Version	Date	Author	Comment
0.1v	04.06.23	M. Raczkowski	Introduction, Electron.js
0.2v	08.06.23	M. Raczkowski	Arduino-cli, Model-C4

1 Introduction

ArduFlow is a user-friendly desktop application that simplifies LED 8x8 matrix animation programming with Arduino. This paper provides an overview of ArduFlow's architecture, highlighting its key components and interactions. By using the C4 model, we present a structured representation of ArduFlow's design, enabling developers and stakeholders to understand its capabilities and potential for further development.

2 Electron

2.1 Main Process

The main process is the heart of Electron.js applications. It runs in the main thread and is responsible for managing the application's lifecycle and interacting with the underlying operating system. It provides system-level functionalities such as file system access, inter-process communication, and native API integration. The main process is typically implemented using Node.js, which enables developers to utilize the vast ecosystem of Node.js modules and libraries.

2.2 Renderer Process

The renderer process handles the rendering of the user interface and runs in separate Chromium-based browser windows or webviews. Each renderer process corresponds to a single web page or view in the application. It utilizes web technologies such as HTML, CSS, and JavaScript to create the user interface, handle user interactions, and communicate with the main process through inter-process communication mechanisms. The renderer process is isolated from the main process, providing security and stability.

2.3 Inter-Process Communication (IPC)

Electron.js facilitates communication between the main process and renderer processes through Inter-Process Communication (IPC). IPC enables seamless data exchange and function invocation between the different processes. It allows developers to pass messages, invoke methods, and share data between the main and renderer processes, enabling effective coordination and synchronization.

2.4 Cross-Platform Capabilities

One of the key advantages of Electron.js is its ability to build cross-platform desktop applications. The architecture of Electron.js abstracts away the underlying operating system differences, allowing developers to write code once and deploy it on multiple platforms. Electron.js achieves this by providing a consistent runtime environment across operating systems, ensuring that the application behaves consistently on Windows, macOS, and Linux.

2.5 Web Technologies and Tooling

Electron.js leverages web technologies, including HTML, CSS, and JavaScript, for building the user interface and application logic. This enables developers to utilize their existing web development skills and frameworks. Electron.js provides a rich set of APIs and tools to create desktop-like experiences using familiar web development patterns. The architecture of Electron.js enables seamless integration with popular web frameworks and libraries, making it flexible and adaptable for a wide range of use cases.

In summary, the architecture of Electron.js combines the main process, renderer process, and IPC mechanisms to create cross-platform desktop applications using web technologies. This architecture empowers developers to leverage their web development skills, utilize the vast Node.js ecosystem, and build feature-rich applications that work consistently across different operating systems.

3 Arduino-cli

Arduino-cli is a command-line interface (CLI) tool that provides a unified and streamlined way to interact with Arduino boards and libraries. It follows a modular architecture that consists of several key components and functionalities.

3.1 Command-Line Interface

The command-line interface is the primary user interface of Arduino-cli. It allows developers to execute various commands to manage Arduino boards, libraries, and sketches. Users interact with the CLI by typing commands and providing relevant arguments and options. The CLI parses these inputs and triggers the corresponding actions or operations.

3.2 Core Modules

Arduino-cli consists of several core modules that handle different aspects of the development workflow. These modules include board, library, sketch, and compile, among others. Each module is responsible for specific operations, such as managing board configurations, handling library dependencies, managing sketches, and compiling code.

3.3 Configuration Files

Arduino-cli utilizes configuration files to store settings and preferences related to boards, libraries, and other tools. These configuration files help in managing and maintaining consistent development environments. Users can modify these files to customize their development settings, such as specifying the board type, communication ports, and library paths.

3.4 Board Manager

The board manager is a crucial component of Arduino-cli that handles board-related operations. It allows users to install, update, and manage board definitions for different Arduino

platforms. The board manager enables developers to easily switch between different boards, manage board-specific settings, and handle board-specific tasks like uploading code and setting bootloader configurations.

3.5 Library Manager

The library manager provides functionalities for managing Arduino libraries. It allows users to install, update, and remove libraries required for their projects. The library manager handles library dependencies, ensuring that the necessary libraries are available and up to date. It simplifies the process of integrating third-party libraries into Arduino projects.

3.6 Sketch Management

Arduino-cli provides capabilities for managing sketches, which are the code files containing Arduino programs. Users can create, edit, compile, and upload sketches to Arduino boards using the CLI. The sketch management component handles the organization and compilation of Arduino code, making it easier for developers to work with multiple sketches and manage their dependencies.

3.7 Integration with Build Systems

Arduino-cli integrates with popular build systems, such as Makefile and CMake, allowing developers to leverage their preferred build tools. This integration enables more advanced and customized build configurations for complex projects. Developers can utilize the CLI commands and tools in conjunction with build systems to streamline their development processes.

Overall, the architecture of Arduino-cli revolves around the command-line interface, core modules, configuration files, board manager, library manager, sketch management, and integration with build systems. This modular architecture provides a flexible and efficient way to interact with Arduino boards, manage libraries, and compile and upload sketches, simplifying the development workflow for Arduino projects.

4 Context Diagram

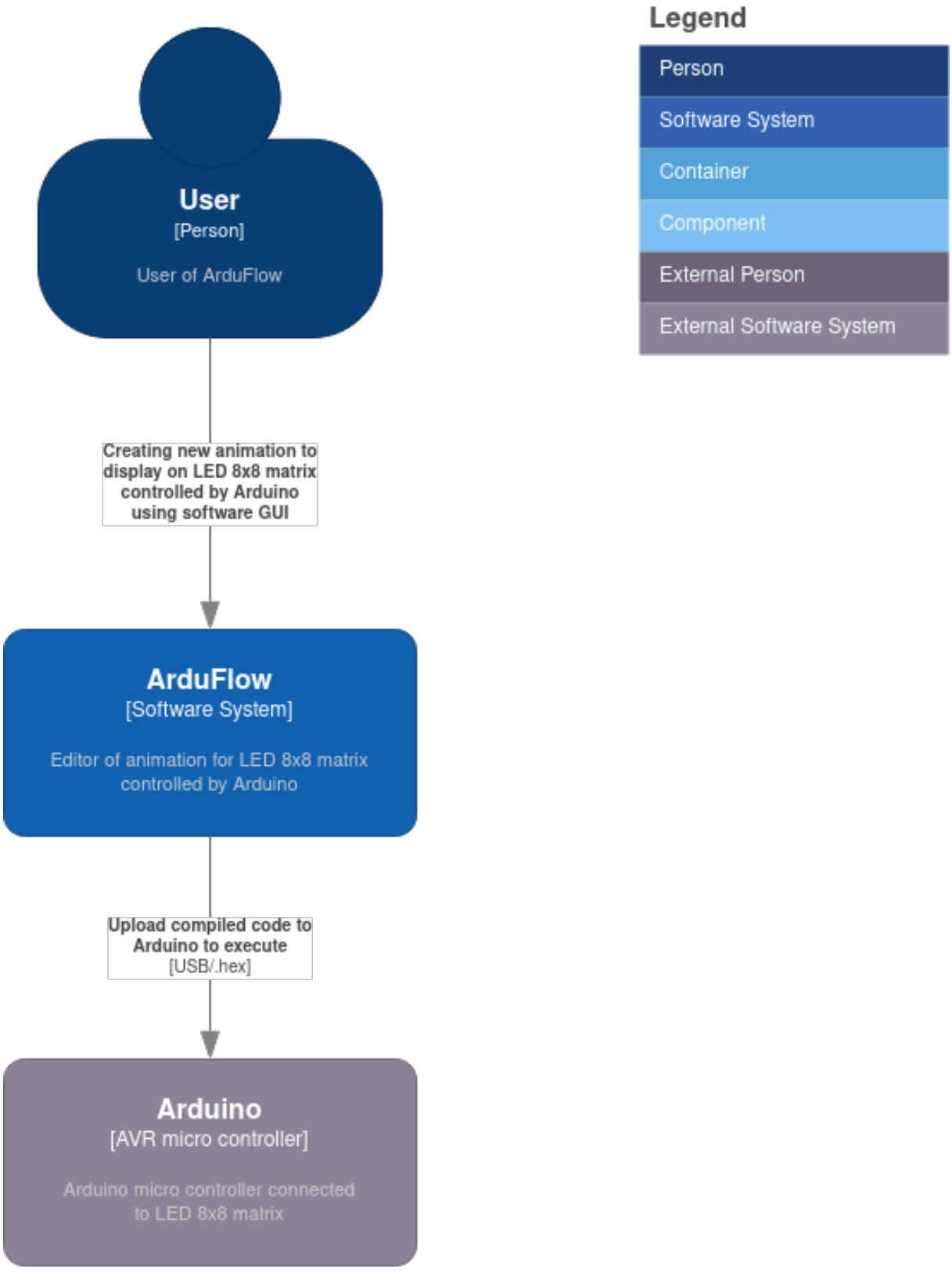

Figure 1: Level 1 of C4 model

5 Container Diagram

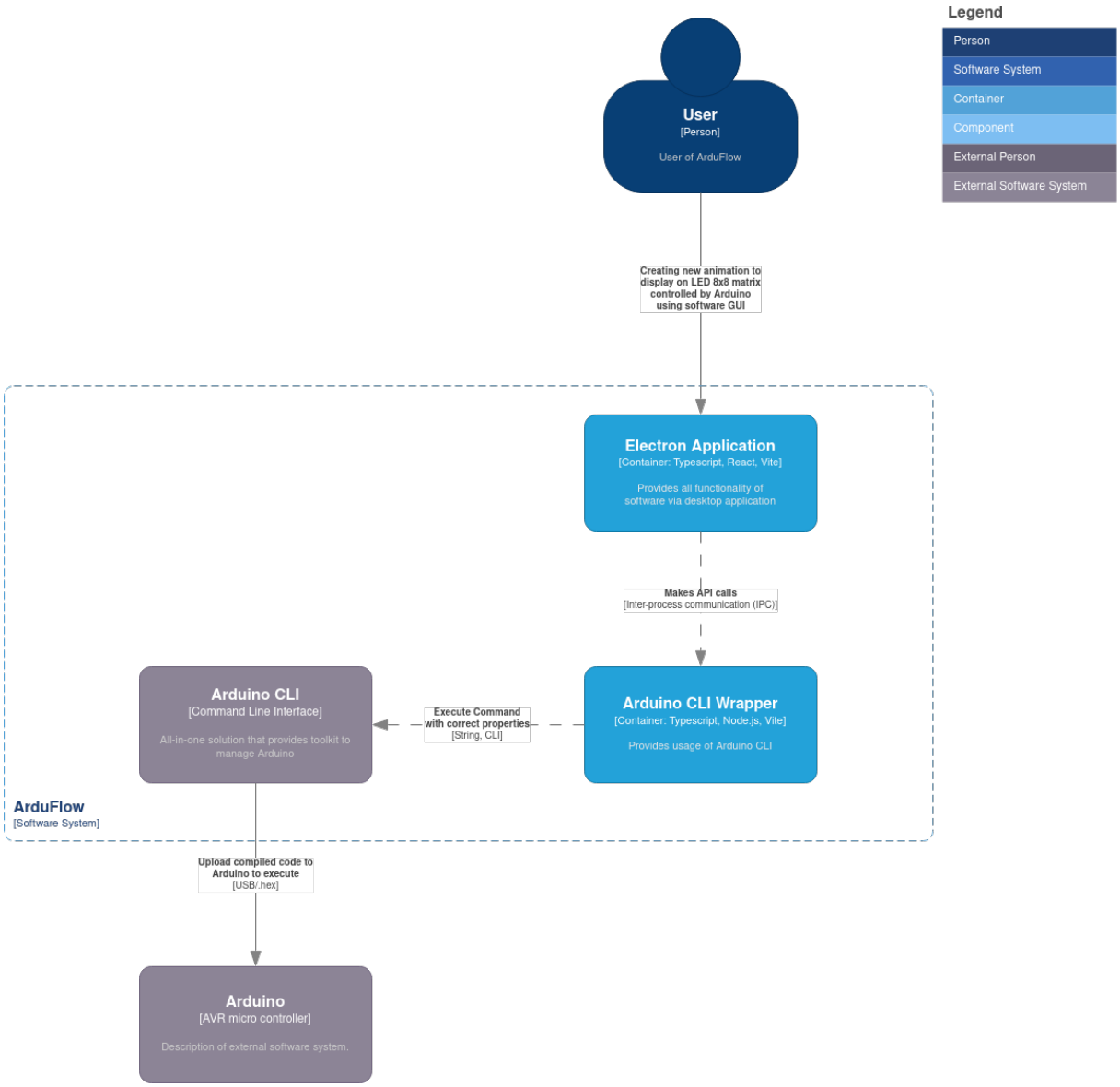

Figure 2: Level 2 of C4 model

6 Component Diagram

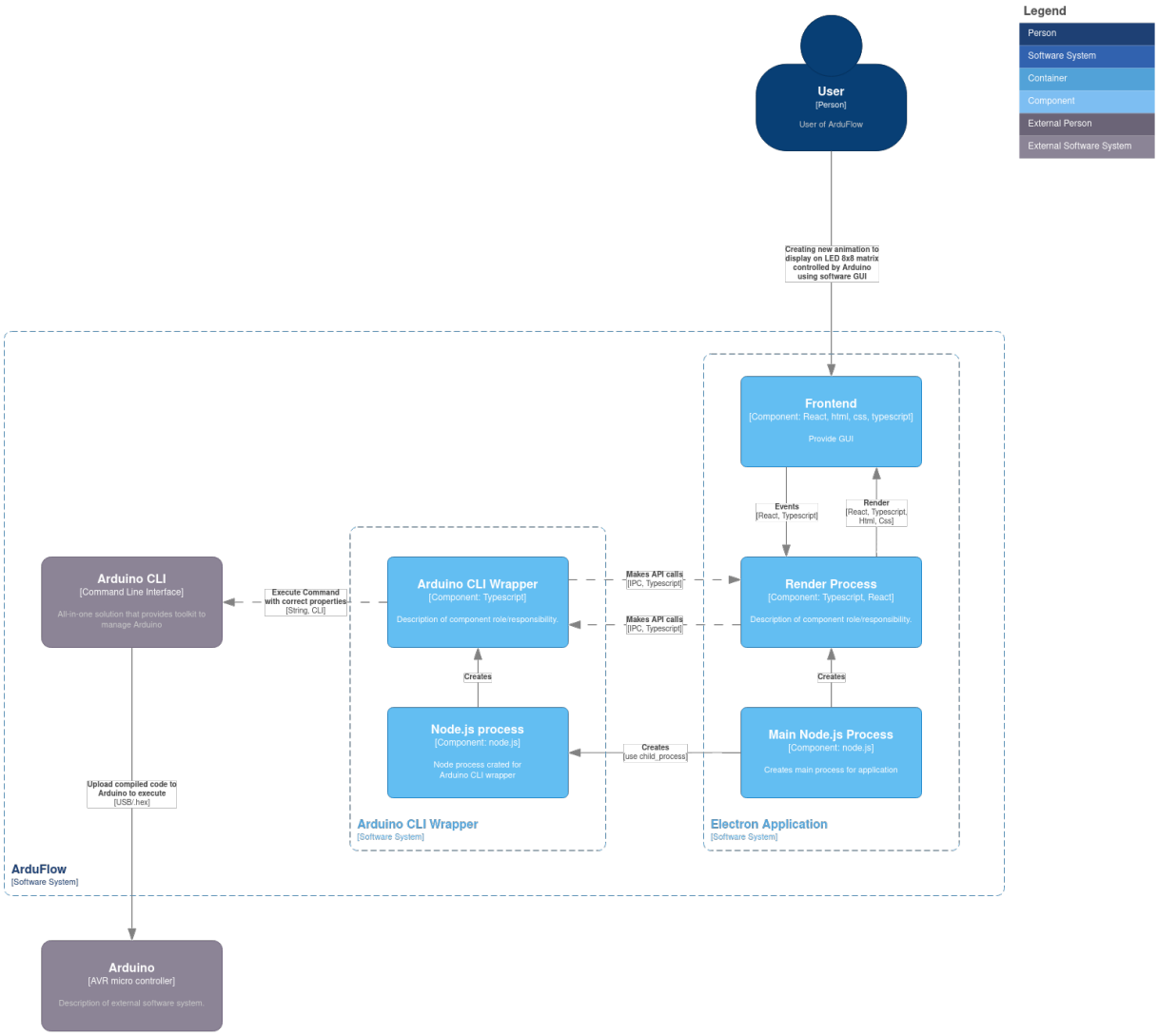

Figure 3: Level 3 of C4 model

7 Code Diagram

Work in progress