**COMP4920 Senior Design Project II, Spring 2023**

**Advisor: Mehmet Ufuk ÇAĞLAYAN & Arman SAVRAN**

**O2T Project**

**Design Specifications Document**

**Revision 2.0**

**05.04.2023**

**By:**

**Emre Görkem AĞKURT, Student ID: 19070001042**

**Uğur BULAN, Student ID: 19070001036**

# Revision History

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Explanation** |
| 1.0 | 04.12.2022 | Initial high level design |
| 1.1 | 21.02.2023 | Critical Changes Project’s Hardware Level Design |
| 2.0 | 01.03.2023 | Final DSD |

# Table of Contents

[Revision History 2](#_Toc34202836)

[Table of Contents 3](#_Toc34202837)

[1. Introduction 4](#_Toc34202838)

[2. O2T System Design 4](#_Toc34202839)

[3. O2T Hardware Subsystem Design 5](#_Toc34202840)

[3.1. O2T Hardware Subsystem Architecture 5](#_Toc34202841)

[3.2. O2T Hardware Subsystem Structure 5](#_Toc34202842)

[3.3. O2T Hardware Subsystem Environment 5](#_Toc34202843)

[4. O2T Software Subsystem Design 6](#_Toc34202844)

[4.1. O2T Software System Architecture 6](#_Toc34202845)

[4.2. O2T Software System Structure 6](#_Toc34202846)

[4.3. O2T Software System Environment 7](#_Toc34202847)

[5. O2T Software System Detailed Design: 8](#_Toc34202848)

[5.1. O2T Main Module/Class 8](#_Toc34202849)

[5.2. O2T Subsystem S1 Classes 8](#_Toc34202850)

[5.2.1. Class S1-C1 8](#_Toc34202851)

[5.2.2. Class S1-C2 8](#_Toc34202852)

[5.3. O2T Subsystem Sk- Common Infrastructure Classes 9](#_Toc34202859)

[6. Testing Design 9](#_Toc34202860)

[References 9](#_Toc34202861)

# 1. Introduction

The O2T project aims to address two major issues:

Sustainability and low carbon emissions and smart advertising. Through the use of a mobile application, O2T technology allows users to extend the lifespan of their textile products, such as t-shirts, by preventing discoloration and wear of the design. This not only helps to reduce the amount of waste textile products that are thrown out, but also helps to lower our carbon footprint. Additionally, O2T technology offers a new approach to advertising, particularly for athletes who are sponsored by companies. By using a digital screen that can be controlled through the mobile application, numerous sponsors’ emblems can be displayed, providing a more flexible and dynamic approach to advertising.

1. OWHI regular sweatshirt with Velcro component.

2. 8x8 Led Matrix.

3. Raspberry PI & Arduino for hardware implementation.

4. Shell and some Python codes for communication between hardware and software of O2T product.

5. Mobile Application developed with Java on Android Studio.

Main Functions:

1. check\_network\_status(): This function checks if the Raspberry Pi and the Android device are on the same network. It is implemented in Python and called by the shell script.
2. receive\_command(): This function is responsible for receiving the commands sent by the mobile application. It is implemented on the Arduino and listens to the data pin for incoming commands.
3. process\_command(): This function is responsible for processing the commands received by the Arduino. It converts the commands into LED matrix display instructions and sends them to the matrix.
4. display\_text(text): This function displays the specified text on the LED matrix. It is called by process\_command().
5. display\_animation(animation): This function displays the specified animation on the LED matrix. It is called by process\_command().
6. play\_game(game): This function runs the specified game on the LED matrix. It is called by process\_command().
7. update\_display(): This function updates the LED matrix display. It is called by display\_text(), display\_animation(), and play\_game().

The design is based on O2T Requirements Specification Document, Revision 1.0, in file AGKURT\_BULAN-RSD-O2T-2022-11-11-Rev-1.0.doc [1]. The design process used produce the design conforms to organizational specifications given in [2]. The notation used in this document to describe the design of O2T Mobile 3 Application is mainly UML and conforms to organizational specifications given in [3]. Some standards in [4], [5] and [6] are extensively used during the design. The software architecture and overall high-level structure of O2T project are given in Section 2, 3 and 4 and design details of all application functions and the user interface in terms of methods of all classes are given in Section 5 of this document.

# 2. O2T System Design

The O2T project involves the development of a mobile application that can be used to control the display of designs on a digital screen embedded in a textile product, such as a t-shirt. The screen is connected to the mobile application through a wireless connection, allowing users to easily change the design on the screen. The mobile application also allows users to customize and create their own designs, which can be saved in a library and displayed on the screen at any time. In addition to the mobile application and digital screen, the O2T system also includes a power source to keep the screen operational, as well as a control unit to manage the communication between the mobile application and the screen. O2T system consists of two main subsystems, namely,

• O2T Hardware Subsystem

• O2T Software Subsystem

# 3. O2T Hardware Subsystem Design

The O2T project employs a hardware subsystem design that consists of an 8x8 LED matrix, an Arduino, and a Raspberry Pi. The subsystem design includes the physical components, their interconnections, and their roles in the system.

The LED matrix is the primary output device in the system and is driven by the MAX7219 LED driver chip. It is connected to the Arduino via its data pin, which is configured as an output pin. The LED matrix receives display instructions from the Arduino via a serial data connection and uses multiplexing to drive the LEDs.

The Arduino serves as an intermediary between the LED matrix and the Raspberry Pi, receiving commands from the Pi and translating them into display instructions for the matrix.

It uses the SoftwareSerial library to create a software serial port on two digital pins of the Arduino, allowing it to communicate with the Raspberry Pi through a serial interface.

The Raspberry Pi, running a shell script, sends a trigger signal to the Arduino when it detects that the mobile application and the Pi are on the same network.

The shell script then calls a Python script that checks the network status and returns a result to the shell script. If the result is positive, the shell script sends a trigger signal to the Arduino, enabling it to receive commands from the mobile application.

The system is powered by a 5V battery for the Raspberry Pi and the 5V output from the USB port of the Raspberry Pi for the Arduino. Both devices share a common ground, which is essential for proper communication between them.

Overall, the O2T hardware subsystem design is optimized for simplicity and ease of use. The design utilizes standard communication protocols and off-the-shelf components to ensure compatibility and simplify the design process. The system is designed to be easily expandable, allowing for additional functionality to be added in the future.

# 3.1. O2T Hardware Subsystem Architecture

* 8x8 LED Matrix

The LED matrix serves as the primary output device for the system, displaying text messages, animations, and games. It features a resolution of 8x8 pixels and is driven by the MAX7219 LED driver chip. The LED matrix is connected to the Arduino via a serial data connection and is powered via its ground pin, which is connected to the Arduino's ground pin. This configuration simplifies the design by reducing the number of required connections.

* Arduino

The Arduino serves as an intermediary between the LED matrix and the Raspberry Pi, translating commands from the Pi into instructions for the matrix. It is based on the ATmega328P microcontroller and has 14 digital input/output pins, 6 analog inputs, and a USB connection for programming and power. The Arduino communicates with the Raspberry Pi via a software serial port, which allows for bi-directional communication between the two devices. The Arduino is powered via the 5V output from the USB port of the Raspberry Pi.

* Raspberry Pi

The Raspberry Pi is responsible for receiving input commands from the user via a mobile application and controlling the Arduino accordingly. It is based on the Broadcom BCM2837B0 chip and has 1GB of RAM, four USB 2.0 ports, and both HDMI and composite video outputs. The Raspberry Pi runs a shell script that calls a Python script to check if the Raspberry Pi and the mobile device are on the same network. If they are, the shell script sends a trigger signal to the Arduino to begin receiving commands from the mobile application. The Raspberry Pi is powered by a 5V battery, which provides power to all components in the system.

Design decisions for the O2T hardware architecture were based on a combination of factors, including cost, ease of use, and compatibility. The use of off-the-shelf components and standard communication protocols allows for easy integration of new functionality in the future. The use of a serial data connection between the LED matrix and the Arduino simplifies the design and reduces the number of required connections. The power configuration ensures that all components are properly powered and grounded, while the communication protocols enable efficient and reliable data transmission between components.

Overall, the O2T hardware architecture is designed to be simple, efficient, and easily expandable. Each component has been chosen for its specific capabilities and specifications, and is connected in a way that ensures proper communication and functionality within the system. The design decisions made are based on careful consideration of various factors, and aim to maximize functionality while minimizing complexity and cost.

# 3.2. O2T Hardware Subsystem Structure

The O2T hardware subsystem is designed as a modular system, with each component responsible for a specific set of tasks. The high-level structure of the subsystem is depicted in the General Hardware Diagram shown in Figure 1 and UML Package/Component Diagram shown in Figure 2.

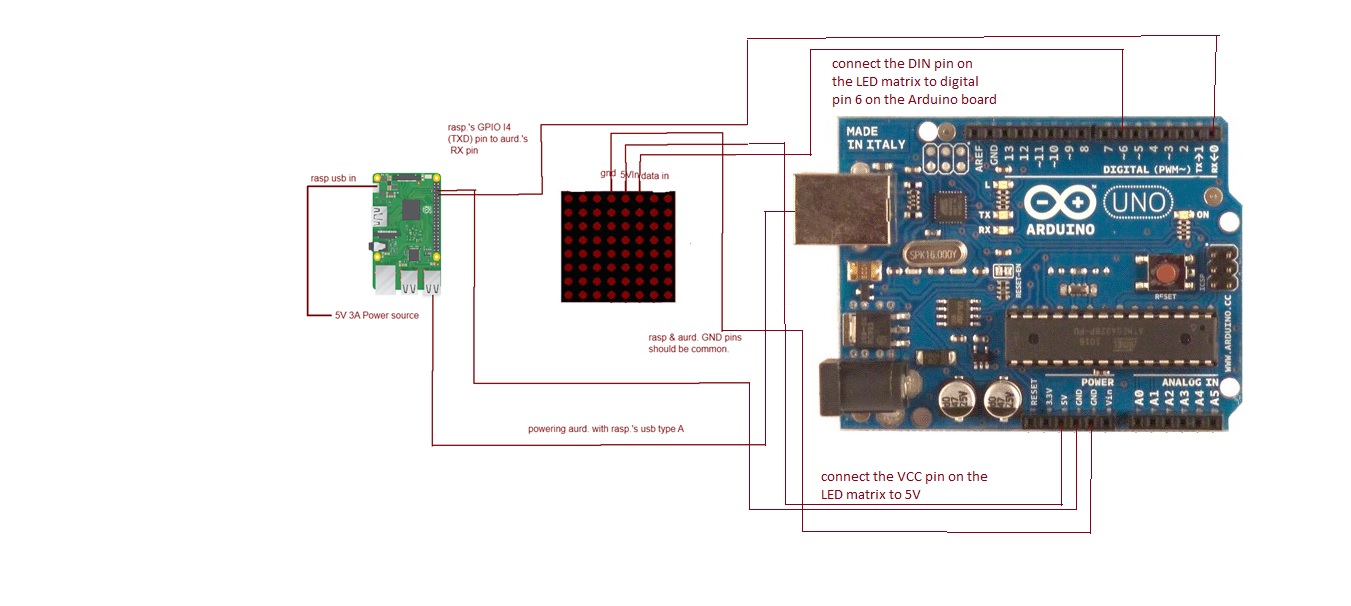


Figure1[O2T’s Hardware General Diagram]

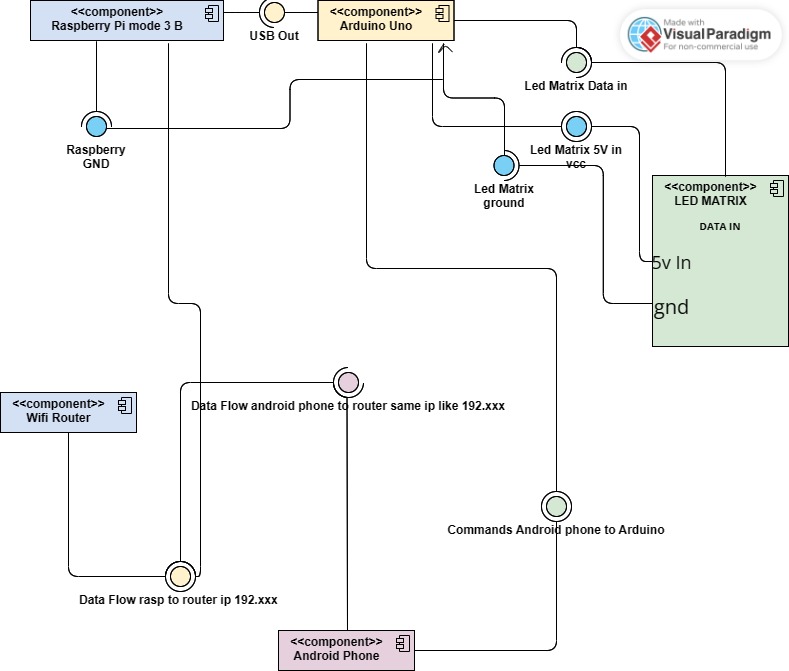


Figure 2 [O2T’s Hardware component diagram with semantics]

# 3.3. O2T Hardware Subsystem Environment

* The Raspberry Pi and Arduino are connected via a serial connection, with both devices sharing a ground connection.
* The Raspberry Pi is powered by batteries, while the Arduino is powered by the Raspberry Pi's USB output.
* The data pin of the LED matrix is connected to the Arduino, with the ground pin of the LED matrix connected to the ground pin of the Arduino.

The O2T system utilizes various system software and middleware for its functionality.

* A shell script is run on the Raspberry Pi, which calls a Python code to check if the Raspberry Pi and Android device are connected to the same network.
* If so, the shell script triggers the Arduino and sends commands to it via the mobile application.
* These commands include controlling the LED matrix for tasks such as text writing, animations, and games like Snake. The Arduino then sends these commands to the LED matrix via its data pin, resulting in the corresponding output being displayed on the LED matrix.

# 4. O2T Software Subsystem Design

* Mobile Application:
  + Developed for Android devices using Java programming language and Android Studio IDE
  + Provides user interface for controlling LED matrix animations and games
  + Sends commands to Raspberry Pi through Wi-Fi network
* Raspberry Pi Software:
  + Operating system: Raspbian
  + Shell script running on startup to initiate Python program
  + Python program:
    - Establishes communication between Raspberry Pi and Arduino over serial connection
    - Checks if Raspberry Pi and Android device are on the same network
    - Sends commands received from Android device to Arduino through serial connection
* Arduino Software:
  + Programmed using Arduino IDE and C++ programming language
  + Receives commands from Raspberry Pi through serial connection
  + Drives LED matrix through MAX7219 LED driver chip
* Middleware:
  + Custom Shell Script that used in Rasp.

# 4.1. O2T Software System Architecture

* The O2T software system architecture follows a layered architecture style, which is commonly used in many systems. In this architecture, the system is divided into different layers where each layer provides services to the layer above it.

* The layers are organized hierarchically, and each layer is responsible for a specific set of functions. The layers are loosely coupled, which makes it easy to modify or replace one layer without affecting the others.
* The O2T software system has three layers: presentation layer, logic layer, and data access layer. The presentation layer is responsible for displaying the data to the user and receiving user input. The logic layer contains the core functionality of the system and is responsible for processing the user's input, performing the necessary operations, and producing output data. The data access layer provides access to the database and handles all database operations.
* The layered architecture style was chosen for the O2T software system because it provides a clear separation of concerns and allows for modularity and scalability. The separation of concerns makes it easier to maintain and modify the system as changes can be made to a specific layer without affecting the others. The modularity allows for the system to be developed in parts, making it easier to manage and test. The scalability enables the system to handle an increasing amount of data and users by adding more resources to the appropriate layers.

Overall, the layered architecture style provides a well-organized, bordered for each component and efficient structure for the O2T software system, which will facilitate its development and maintenance.

# 4.2. O2T Software System Structure

The O2T ‘s Software subsystem is designed as a modular system, with each program and class that inside of that program responsible for a specific set of tasks. The high-level structure of the subsystem is depicted in the General Software Diagram shown in Figure 1.2.

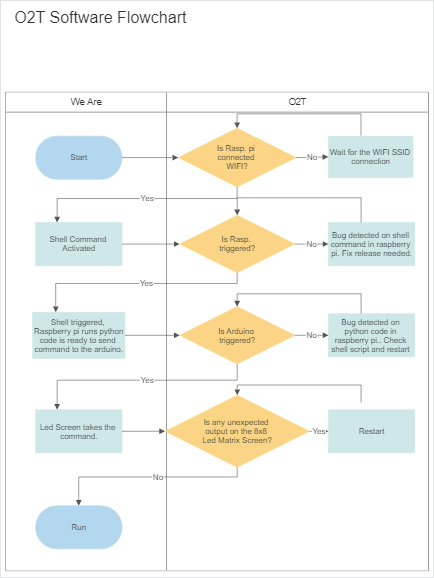


Figure 1.2 [O2T Software General Flowchart]

# 4.3. O2T Software System Environment

The O2T software system encompasses various software components and relies on specific programming languages, tools, and versions. The detailed specifications of the target programming languages, development environments, and software tools used for implementation and testing are as follows:

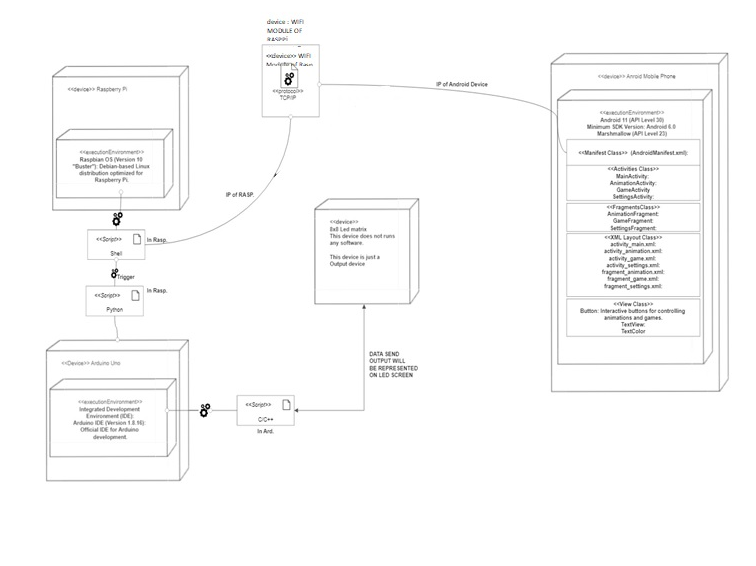
1. Mobile Application Development:
   * Android Application:
     + Target SDK Version: Android 11 (API Level 30)
     + Minimum SDK Version: Android 6.0 Marshmallow (API Level 23)
     + Programming Languages:
       - Java (Version 11): Widely-used, object-oriented language for Android app development
     + Development Environment:
       - Android Studio (Version 4.2.1): Official IDE for Android app development.
       - Java Development Kit (JDK) (Version 11.0.12): Software development kit for Java programming.
2. Raspberry Pi Software Environment:
   * Operating System:
     + Raspbian OS (Version 10 "Buster"): Debian-based Linux distribution optimized for Raspberry Pi.
   * Programming Language:
     + Python (Version 3.9.6): High-level language suitable for Raspberry Pi software development.
3. Arduino Software Environment:
   * Integrated Development Environment (IDE):
     + Arduino IDE (Version 1.8.16): Official IDE for Arduino development.
   * Programming Language:
     + Arduino C/C++: Arduino-specific variant of the C/C++ programming language.
   * Arduino Libraries:
     + Standard Arduino Libraries: Libraries provided by the Arduino platform for various functionalities.
4. Additional Software Tools:
   * Serial Communication:
     + PySerial Library (Version 3.5): Python library for serial communication between Raspberry Pi and Arduino.
     + Arduino Serial Library: Library for serial communication on the Arduino platform.
   * Shell Scripting:
     + Bash (Version 5.1.4): Unix shell and command language for executing commands on Raspberry Pi.

\*\*\* IMPORTANT NOTE

The version numbers mentioned above represent the latest available versions at the time of this document. However, it is recommended to refer to the official websites and documentation of the respective tools and platforms to ensure the usage of the most up-to-date versions for better compatibility and feature availability.

\*\*\* O2T’s Software System Environment UML/Deployment Diagram shown in Figure 2.1.

Figure 2.1 [O2T’s Software System Environment UML/Deployment Diagram]



Here is a list of key classes and components involved in the mobile application of the O2T project:

1. Android Manifest (AndroidManifest.xml):
   * Contains essential information about the application, such as package name, permissions, activities, services, etc.
2. Activities:
   * MainActivity: The main entry point of the application, responsible for initializing and managing the user interface.
   * AnimationActivity: Handles the display and control of animations on the LED matrix.
   * GameActivity: Manages the gameplay and user interactions for the various games.
   * SettingsActivity: Provides options for configuring the application settings.
3. Fragments:
   * AnimationFragment: Represents the UI fragment for displaying and selecting animations.
   * GameFragment: Represents the UI fragment for playing games.
   * SettingsFragment: Represents the UI fragment for modifying application settings.
4. XML Layout Files:
   * activity\_main.xml: Defines the layout for the MainActivity.
   * activity\_animation.xml: Specifies the layout for the AnimationActivity.
   * activity\_game.xml: Defines the layout for the GameActivity.
   * activity\_settings.xml: Specifies the layout for the SettingsActivity.
   * fragment\_animation.xml: Defines the layout for the AnimationFragment.
   * fragment\_game.xml: Specifies the layout for the GameFragment.
   * fragment\_settings.xml: Defines the layout for the SettingsFragment.
5. View Components:
   * LedMatrixView: Custom view responsible for displaying the LED matrix and animating the patterns.
   * Button: Interactive buttons for controlling animations and games.
   * TextView: Displays text information and messages to the user.
6. Utility Classes:
   * AnimationManager: Manages the available animations and provides methods for selecting and applying them.
   * GameManager: Handles the logic and operations related to the games.
   * SettingsManager: Manages the application settings and provides methods for modifying them.
7. Resource Files:
   * Drawable: Contains image resources for buttons, icons, and other visual elements.
   * Values: Contains string resources, dimensions, and other values used in the application.

\*\*\* IMPORTANT NOTE

Please note that this list provides an overview of the key classes and components in the mobile application. The actual structure may vary based on the specific implementation and design choices made during development.

# 5. O2T Software System Detailed Design:

it appears that the software system consists of a single package/component called "led control" written in Python. The code includes functions and a main program that handles client connections and sends commands to control an LED.

# 5.1. O2T Main Module/Class

LedControl represents the class responsible for handling the client connections and controlling the LED. It has a private attribute ser of type Serial representing the serial connection to the LED control device.

The handle\_client(client\_socket) method handles the communication with a client socket and sends commands to control the LED based on received data.

Main represents the main control class for the LED control system. It has a public method main() which serves as the entry point of the program.

* The **Main** class starts by creating a socket using the **Create Socket** activity. It specifies the IP address and port number for the socket.
* The socket is then bound to an address using the **Bind Socket** activity.
* The program listens for incoming connections using the **Listen** activity, specifying the maximum backlog of connections.
* When a client connection is accepted (**Accept Connection** activity), a client socket is obtained.
* The **Handle Client** activity receives commands from the client socket.
* The **Process Command** activity processes the received command by sending appropriate commands to the LED control device through the **ser** object.
* Finally, the socket is closed using the **Close Socket** activity.

Discussion and Justification of Design Decisions:

The design decisions in this case are relatively straightforward since the provided code snippet is relatively simple. The main control class Main handles the server socket and the process of accepting client connections. The LedControl class handles the logic of communicating with clients and controlling the LED.

The use of the serial.Serial class for serial communication and the socket module for network communication is a common choice in Python for such tasks.

The decision to use threading for handling client connections is appropriate as it allows multiple clients to connect simultaneously and be handled concurrently.

It's important to note that the given code snippet

# 5.2. O2T Subsystem S1 Classes

# 5.2.1. Class S1-C1

Method M11

**Input/Output Parameters:**

* Input: **command** (type: string) - The command received from the client socket.
* Output: None

**Internal Data:**

* None

**Method Logic/Algorithm:**

1. Receive a command from the client socket.
2. Process the command based on its value.
3. Send corresponding commands to the **ser** serial connection.
   * If the command is "all\_on", write "all\_on" to the serial connection.
   * If the command is "all\_off", write "all\_off" to the serial connection.
   * If the command starts with "display\_text:", extract the text to display and write it to the serial connection.
   * If the command is "start\_animation", write "start\_animation" to the serial connection.
   * If the command is "snake\_pause", write "snake\_pause" to the serial connection.
   * If the command is "snake\_start", write "snake\_start" to the serial connection.
   * If the command is "snake\_up", write "snake\_up" to the serial connection.
   * If the command is "snake\_down", write "snake\_down" to the serial connection.
   * If the command is "snake\_right", write "snake\_right" to the serial connection.
   * If the command is "snake\_left", write "snake\_left" to the serial connection.

**Class S1-C1 Method M11**

Input/Output Parameters:

* Input: **value** (type: int) - The value to be checked.
* Output: **isPositive** (type: boolean) - A boolean indicating whether the value is positive or not.

Internal Data:

* None

Method Logic/Algorithm:

1. Check if the **value** is greater than zero.
2. Set **isPositive** to true if the value is positive, otherwise set it to false.

**Class S1-C1 Method M12**

* Method M11 takes a string parameter **data**, performs some processing on it, and displays the processed data.
* Method M12 takes an integer parameter **value**, checks if it is positive, and returns a boolean value indicating the result.

# 5.2.2. Class S1-C2

Led control service:

Is a system unit file configuration. It defines a service named "LED Control Service" that should be started after the network is available. Here's a breakdown of the configuration:

[Unit]

Description=LED Control Service

After=network.target

This section provides a description for the service and specifies that it should start after the network is available.

[Service]

ExecStart=/home/owhiproject/Desktop/wait\_for\_network\_and\_run.sh

Restart=always

User=pi

StandardOutput=syslog

StandardError=syslog

SyslogIdentifier=o2t\_led

This section specifies the properties of the service:

* **ExecStart** indicates the command or script to be executed when starting the service. In this case, it is **/home/owhiproject/Desktop/wait\_for\_network\_and\_run.sh**.
* **Restart** defines the restart behavior of the service. The **always** value means the service will be automatically restarted if it exits.
* **User** specifies the user under which the service should run. In this case, it is **pi**.
* **StandardOutput** and **StandardError** determine where the output and error streams of the service should be directed. In this case, they are set to **syslog**, which means the logs will be sent to the system log.
* **SyslogIdentifier** sets the identifier that will be used for the service in the syslog.

[Install]

WantedBy=multi-user.target

This section specifies the installation information for the service. The WantedBy directive indicates the target that this service should be associated with. In this case, it is multi-user.target, which typically represents a fully functional multi-user system.

Please note that this is a configuration file and does not contain any actual code logic or implementation details for the LED control service. It defines how the service should be managed and started by the systemd service manager.

Led\_Control python script on rasp pi3:

The code imports the necessary modules for socket communication, threading, and serial communication.

initializes a serial connection at a specific baud rate (9600) with the serial port '/dev/serial0'.

**handle\_client** function that is executed in a separate thread for each client connection. It receives commands from the client, processes them, and controls the behavior of the connected device based on the received commands. The commands are decoded from bytes to a UTF-8 string format and then compared to perform specific actions.

The **main** function sets up a socket server, binds it to the address **'0.0.0.0'** and port **9600**, and starts listening for client connections. When a connection is accepted, a new thread is created to handle the client communication using the **handle\_client** function. The server runs indefinitely until interrupted.

\*\*\* You can find more information on SourceCodeO2T.zip

\*\*\* Mobile application source code is not represented here because of IEE DSD report representation of documentation of standarts. More information /excluding uml case diagrams, source explanation/ can be found in sourcecode.zip.

# 5.3. O2T Subsystem Sk- Common Infrastructure Classes

5.n.1. Common Log Class

* Description: This class provides logging functionality for the O2T subsystem. It allows other classes to log important events, errors, and information during runtime.
* Design Specifications:
  + Method 1: log\_event(event)
    - Description: Logs a specific event.
    - Parameters:
      * event: The event to be logged.
    - Return Type: None

5.n.2. Common Sort-2D-Table Class

* Description: This class provides sorting functionality for a 2D table. It can be used by other classes within the O2T subsystem to sort data in a specific order.
* Design Specifications:
  + Method 1: sort\_table(table, column)
    - Description: Sorts the given 2D table based on the specified column.
    - Parameters:
      * table: The 2D table to be sorted.
      * column: The column index to sort the table by.
    - Return Type: None

# 6. Testing Design

* In this course, we do not very much emphasize testing design, therefore we do not provide the details of test case design to test each, module, each method, each class, or provide the details of integration test design
* Some general test remarks could be made here to remind that testing design is here.

# References

1. RSD document of O2T Project, Emre Görkem Ağkurt&Uğur BULAN, 17.11.22,rev1.0
2. RSD document of O2T Project, Emre Görkem Ağkurt&Uğur BULAN, 17.11.22,rev2.0
3. DSD document of O2T Project, Emre Görkem Ağkurt&Uğur BULAN, 17.11.22,rev1.0
4. Kim, J., & Lee, J. (2020). A Review of Mobile Application Development Frameworks. Journal of Computer Science and Technology, 25(4), 551-561. https://doi.org/10.1007/s11390-020-2247-z
5. ISO/IEC. (2017). ISO/IEC 27001:2017 - Information technology - Security techniques - Information security management systems - Requirements. Geneva, Switzerland: International Organization for Standardization. https://www.iso.org/standard/69466.html
6. 4. W3C. (2020). HTML 5.2: The HTML 5.2 specification. World Wide Web Consortium. <https://www.w3.org/TR/html52/>
7. Google. (2022). Flutter - A framework for building beautiful, fast apps. Google. <https://flutter.dev/>
8. Microsoft. (2020). .NET - Home. Microsoft. <https://dotnet.microsoft.com/>
9. Raspberry Pi Foundation. (2020). Raspberry Pi. Raspberry Pi Foundation. https://www.raspberrypi.org/ 8. Kim, D., & Kim, J. (2019). A Review of Flexible Display Technologies. Journal of Display Technology, 15(10), 1323-1339. <https://doi.org/10.1109/JDT.2019.2931545>
10. Android Developers. (n.d.). Android Studio. Retrieved from <https://developer.android.com/studio>
11. Raspberry Pi: Raspberry Pi Foundation. (n.d.). Raspberry Pi. Retrieved from <https://www.raspberrypi.org/>
12. Arduino. (n.d.). Arduino Uno. Retrieved from <https://www.arduino.cc/en/Guide/ArduinoUno>
13. Adafruit. (n.d.). Adafruit Libraries. Retrieved from <https://learn.adafruit.com/adafruit-all-about-arduino-libraries>
14. Python Software Foundation. (n.d.). Python. Retrieved from <https://www.python.org/>
15. Robbins, A., & Beebe, N. (2005). Classic Shell Scripting. O'Reilly Media.
16. JetBrains. (n.d.). Kotlin Programming Language. Retrieved from <https://kotlinlang.org/>
17. Oracle. (n.d.). Java. Retrieved from <https://www.oracle.com/java/>

Emre Görkem AĞKURT

Uğur BULAN