Group 10 - Smart Mouse Trap Project
Embedded Systems 2023/24

Authors:

Cristina Pêra (up201907321);
Diogo Ferreira (up201805258);
Rogério Rocha (up201805123);
Telmo Ribeiro (up201805124).

Project Description:

The project aims to develop a **Smart Mouse Trap** capable of effectively capturing mice through the detection of their movements while providing real-time notifications and control options to the user via an Android application.

We'll illustrate the system's overall functionality using an actor diagram. Next, we'll outline the connections between system components. Finally, we'll present a series of specifications, including architectural and software modeling diagrams.

Requirement analysis:

The mouse trap consists of a box equipped with a sensor to detect when a mouse enters. Upon detection, the system sends an alert to the user's smartphone along with a picture of the captured mouse.

Functional requirements:

These requirements are directly related to specific operations or tasks that the system must be capable of performing. They are essential for the system's operation and interaction with the user.

1. Detect when something is inside the box.
2. Send a notification to the trap owner when something is detected.
3. Allow the user to close or open the box via the smartphone application remotely.
4. Provide real-time status updates of the trap (closed/opened) on the smartphone.
5. Enable the user to request a picture of the contents inside the box through the smartphone app.
6. The Mouse Trap System should be able to detect the failure of endpoints and reset.
7. The Mouse Trap System should close if no action is performed within 5 seconds of sending an alarm.

Non-functional requirements:

These are related to the performance, usability, reliability, and other qualities of the system. They do not define specific behaviors but rather the standards and constraints within which the system operates.

1. The warning sent to the owner's smartphone should have a latency of less than 3 seconds.
2. The action to close or open the box via the smartphone app should take effect within 3 seconds.
3. The trap should aim for practicality through reduced weight/size.
4. The Android application should be intuitive to use.

Actor Model:

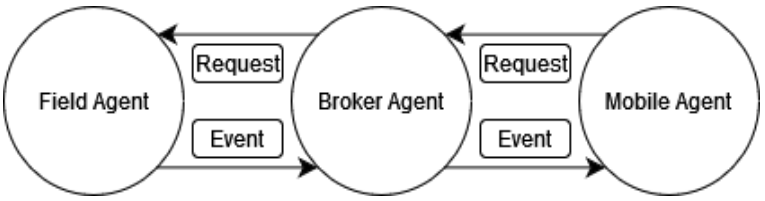


Fig.1: ***Actor Model*** *for the* ***Mouse Trap System***

The **Actor Model** performs over three types of flags.
Events are flags regarding actions that have already happened, i.e., are mostly used to inform the **Mobile Agent** that certain operations have transpired, although eventually carrying additional information.
Requests are flags regarding actions that are desired to happen, i.e., are mostly used to inform the **Field Agent** that certain operations have to be enforced.
Additional flags are used to control the message sequencing.
At the time of this development, the programming efforts were completed and we ended up using the same communication channel to operate the three different types. Although we did not suffer from major drawbacks in doing so, in a future iteration it could be fortuitous to differentiate the **data** plane (**Events** / **Requests**) and the **control** plane (**Additional**).

Event Map:

Se - **S**ensor detection;
Pe - **P**hoto content;
Oe - motor state = **O**pen;
Ce - motor state = **C**lose;

Request Map:

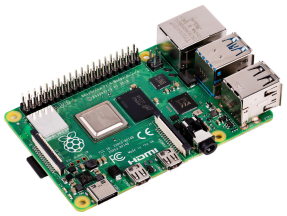
- - - - - - - - - -
Pr - **P**hoto content requested;
Or - motor state = **O**pen requested;
Cr - motor state = **C**lose requested;

Additional:

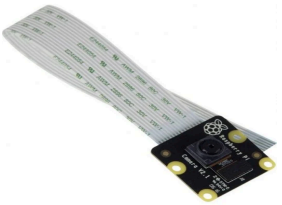
SHUTDOWN - requests endpoint **SHUTDOWN**;
NSYNC - blocks handshake;
SYNC - allows handshake;
SYNC_ACK - acknowledges **SYNC** and updates status;

The **Mouse Trap System**, described by the actor model in *Fig 1*, is performed in three roles.
There is a central actor, the **Broker**, who acts as a middleman between the **Field** and the **Mobile** agent. It must receive **events** from the **Field**, and **requests** from the **Mobile**, propagate them to the proper receiver, establish handshakes, detect downtimes, shutdown endpoints, and execute basic logic to determine if certain conditions are met.
The existence of such an actor can lead to a major drawback: a central point of failure. Nonetheless, the system is designed so the **Field Agent** (or part of) and the **Broker Agent** are deployed together (explained in greater detail during **Deployment**), which means a great number of issues that could disturb the **Broker** would disturb the **Field** anyway, already constituting a problem to the underlying communication. The takeaway from the previous sentence should be that if the **Broker** is well-designed (logic-wise), it alone should not be the point of failure when issues occur. That said, it provides great modularity and abstraction while diminishing the complexity of the remaining endpoints.
A future extension to this project could see the endpoints working in two different modes. The handshake is always performed with the **Broker**, but during this communication enough data is passed so that in case the **Broker** fails, the endpoints can communicate in a peer-to-peer fashion, losing some assurances but still able to perform.
The **Field Agent** is the system capable of handling **requests** to operate over physical quantities and generate matching **events** when measuring said physical quantities. All the major **sensors** and **actuators** are under the domain of the **Field**.
The **Mobile Agent** is the system providing feedback (**events**) to, and accepting controls (**requests**) from, the user.
From this point onward, the term **Mouse Trap System** is used when describing the model as a whole.

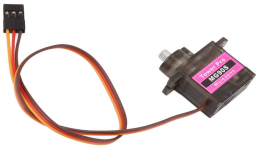
*Fig. 2: **Hardware Components** - Cost: 170€*



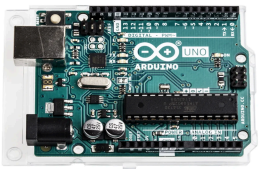
a) Raspberry-Pi 3 B+



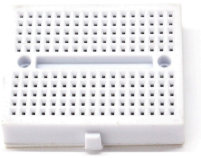
b) NoIR V2 Camera



c) Micro Servo Motor



d) Arduino Uno Rev3

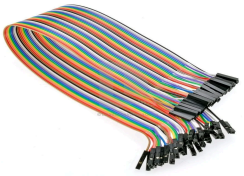


e) Mini BreadBoard

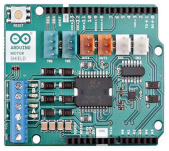
f) Ultrasonic Sensor

g) Universal Power Supply

h) Smartphone Android



i) Jumper Wires



j) Motor Shield Rev3

Specification:

The specification establishes a connection between the **Actor Model** (*Fig.1*) and the **Hardware Components** (*Fig.2*).
A **Raspberry Pi** will play the role of the **Broker**, the middleman between the remaining endpoints, thereby called **Raspberry Pi - Broker**.
Every message arriving at the **Raspberry Pi - Broker** will be inspected and relayed to the according endpoint.
The **Raspberry Pi - Broker** also performs control logic and generates messages when conditions are met, for instance, upon detection by the ultrasonic sensor, if no action is performed by the **Mobile** under 5 seconds, the **Broker** itself will send a close request (as demanded by the requirements).
Behaviors like the one described help reduce the complexity of the endpoints, essentially turning them into flag transmitters and receivers.
A **Raspberry Pi** with a **NoIRv2 Camera** attached plays a part of the **Field Agent**, thereby called **Raspberry Pi - Field Brain**.
The communication between the **Field** and the **Broker** happens through **WiFi** (bidirectionally) and is dealt with by the **Raspberry Pi - Field Brain**.
Except for the ultrasound sensor and the motor direct controls, the major logic on the **Field Agent** (such as taking photos) is also tackled on the **Raspberry Pi - Field Brain**, hence the name.
The **Arduino** + **Mini BreadBoard** + **Motor Shield** + **Micro Servo Motor** + **Ultrasonic Sensor**, thereby called **Arduino - Field Muscle**, is responsible for playing the remaining part of the **Field Agent** (control over the sensors and actuators).
It communicates with the **Raspberry Pi - Field Brain** through **UART** (bidirectionally) and deals with the tasks relayed by the former through this connection.
The **Android/Mobile Phone** is responsible for performing the **Mobile Agent** role which generates **requests** and expresses **events** through a GUI, and is called **Mobile/Android Phone - Mobile**.
Mobile/Android Phone - Mobile communicates with the **Raspberry Pi - Broker** through **WiFi** (bi-directionally).
The specification tries to express through the names adopted, the relation between the hardware components and the agents themselves. Nonetheless, throughout this iteration, abbreviations will be used when we judge the meaning is still clear, and to avoid convoluted naming such as **Mobile Phone - Mobile**.

Names such as **Broker**, **Mobile**, **Field Muscle,** and **Field Brain** will be used either to describe the agents, the hardware, or the logic performed by that hardware interchangeably as long as the subject is disclosed previously.

Blocks Diagram Overview:

The block diagrams depicted in *Fig.3 & Fig.4* illustrate the system in terms of the logical units comprising it.
If we regard these logical units as hardware components, then the diagram depicts the hardware architecture.
Alternatively, if the logical units correspond to software modules, then we say the diagram describes the software architecture.

Hardware Architecture:

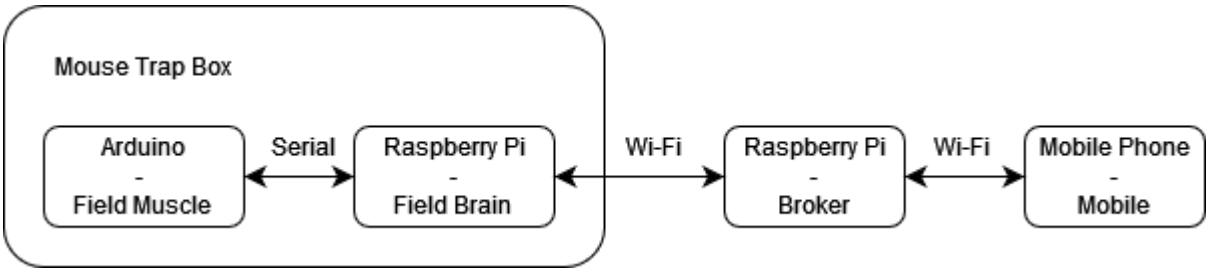


Fig.3: ***Hardware Architecture*** *Diagram*

In the hardware architecture diagram (*Fig.3*), we can identify four components:

- **Arduino - Field Muscle** (Arduino Uno + Mini Breadboard + Motor Shield + Micro Servo Motor + Ultrasonic Sensor)
- **Raspberry Pi - Field Brain** (Raspberry Pi 3 + NoIRv2 Cam)
- **Raspberry Pi - Broker** (Raspberry-Pi 3 B+)
- **Mobile Phone - Mobile** (Android)

As stated previously, the communication between the **Field Muscle** and the **Field Brain** is bidirectional and is done through the **UART** protocol.
The remaining communications are bidirectional and accomplished through **WiFi**.
The **Mouse Trap Box** (component) regards the **Field** (agent) as a whole (**Brain** + **Muscle**).

Software Architecture:

The software architecture comprises blocks that make up the various software modules running on multiple devices. However, it is important to keep the software architecture independent from the hardware components, so we can later relate software modules and hardware components in a deployment diagram.
As evidenced by *Fig 4*, the firmware is composed of control logic, message management, and communication channels, in our case **TCP/IP** and **UART**.
The control logic implements the functionalities present throughout the **Mouse Trap System**.
The message management is responsible for coding and decoding messages (where we have implemented a basic codec) and for the message exchange logic.
When a new message is received a new thread will be called to handle it, the reason behind being the continuous (almost) receiving of new messages.
Transport-wise, the communications are managed by synchronous sockets in the **WiFi TCP/IP** connections.

Mobile implemented, additionally, a GUI to properly present the received **events** and enforce **requests**.

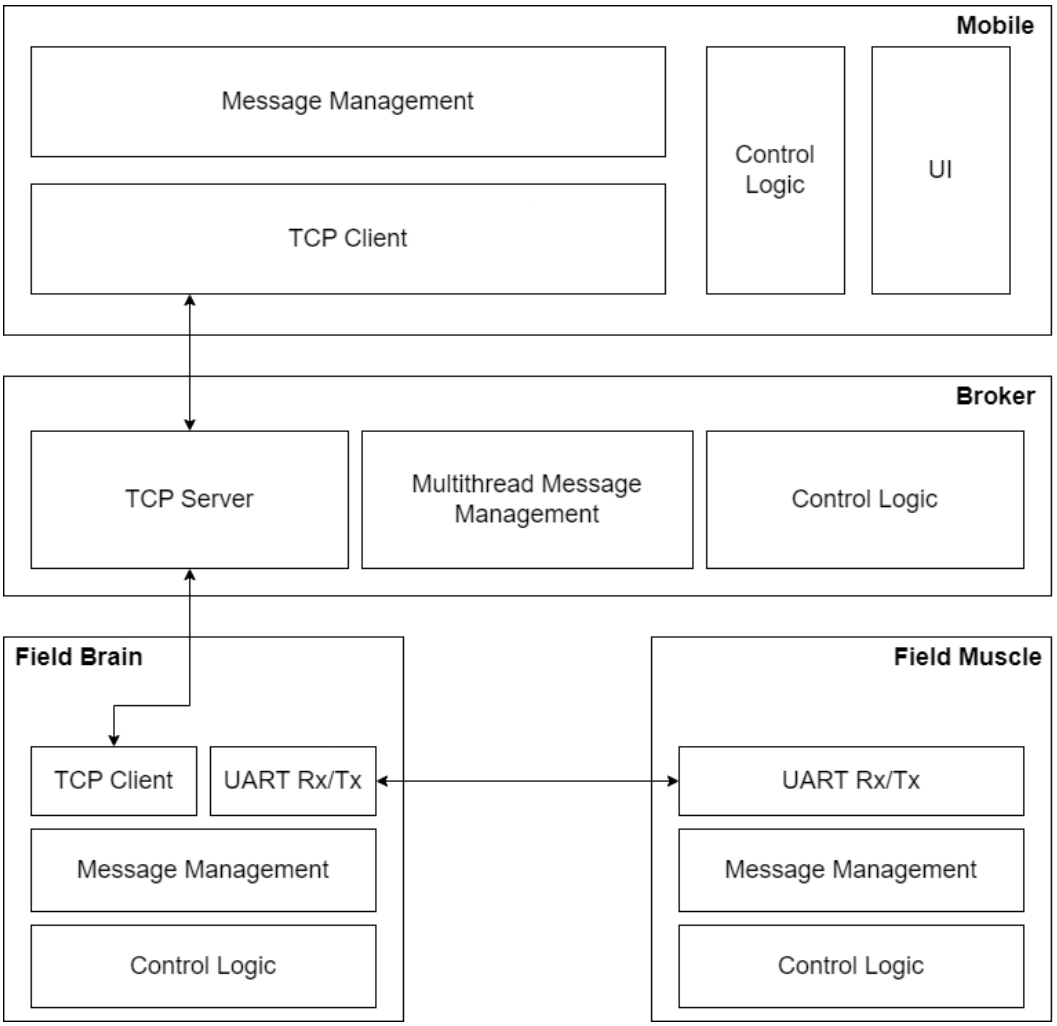


Fig.4: ***Software Architecture*** *Diagram*

Deployment Diagram:

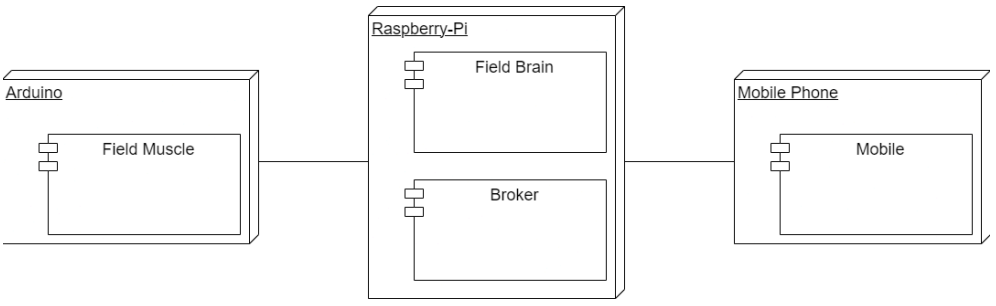


Fig.5: ***Deployment*** *Diagram*

The deployment diagram (*Fig.5*) represents the configuration and architecture of the system and the connection between the software and hardware components.
Ideally, there would be two distinct **Raspberry Pi** to implement upon (**Broker** & **Field Brain**).

That would allow a greater detachment between the **Broker** and the remaining endpoints, and if that was fortuitous, eventually have the **Broker** on a remote site.
Because of component **availability constraints**, that was not possible in the current iteration and as such, both the **Broker** and the **Field Brain** are deployed on the same **Raspberry Pi**.
Please note that the logic-wise isolation is not breached just because both endpoints were deployed together.

Software Modeling:

In this section, we provide flowcharts depicting the main logical components of the **Mouse Trap System**: **Field Muscle** (*Fig.6*), **Field Brain** (*Fig.7*), and the **Broker** (*Fig.8*).
Two logical components were developed yet are not present in the form of flowcharts: the **photo reshoot** logic and the **Mobile**.
The reason behind their lack is the same: both fall under the category of trivial components.
The **photo reshoot** is responsible for accessing the camera and taking a photo that will rewrite the current picture to be sent (looped infinitely).
Mobile is a compact software capable of reading button presses and transforming it in **requests** and receiving **events** updating the displayed data.
Attending the logic present on flowcharts, there is a **Setup** state present on the three graphs.
This state concerns the handshake with the remaining endpoints, and it is the default target when an error is made evident during an iteration.
The **Loop** block is shared between the three charts and in the absence of errors, the received message is decoded to determine the appropriate course of action.
The consequent behavior is closely expressed by the respective flowchart.

The **Field Muscle** aims to discern the type of **request** received (**O**pen **r**equest / **C**lose **r**equest).
Following this inspection, the action is executed as expressed by **Write Digital**, and the matching **event** (**O**pen **e**vent / **C**lose **e**vent) is sent to the **Field Brain**.
Additionally, if an object is detected inside the trap as a consequence of the measurement from the **Read Analog**, an **event** (**S**ensor **e**vent) is once again sent to the **Field Brain**.

The **Field Brain** focuses on identifying if there is a new message from the **Field Muscle** regarding previous relays (**Read Serial**).
If that is the case, then the **Field Brain** relays that matching **event** to the **Broker** (**Write WiFi**).
Otherwise and/or after, the **Field Brain** checks for new **requests** from the **Broker** (**Read WiFi**), and relays it to the **Field Brain** (**Write Serial**) when the **request** is a direct operation over the motor (**Or** \ **Cr**).
In the case of a **P**hoto **r**equest (**Pr**), the **Field Brain** sends the latest photo data available to the **Broker** (**Write WiFi**).

The **Broker** checks for received **requests** from **Mobile**, forwards them to the **Field Brain,** and performs analogous yet symmetric actions regarding ***events***.

Although a detailed overview, the flowcharts hide some harder-to-grasp ideas.
Analyzing the flowcharts, one could believe that errors are tested at the beginning of the **Loop**, in reality, we employ multiple **try**/**catch** to detect exceptions throughout the code.
As mentioned before, the **Broker** abstracts concepts from the endpoints to allow for “dumb” yet fast terminals.
The requirements ask for the door to be closed if no actions are performed by the user in 5 seconds after detection, and for a photo to accompany a detection.

These concepts were employed by the **Broker**.
After receiving a **Se**, if no **O**pen **r**equest or **C**lose **r**equest arrives for 5 seconds, a **Cr** is generated at the **Broker** and forwarded to the **Field Brain**.
The fact that this **request** was not generated by the User is completely hidden from the **Field Brain** which does not affect its current behavior.
The attachment of a photo to a **S**ensor **e**vent was accomplished similarly.
When the **Se** is received at the **Broker,** a **Pr** is sent backwards, that way the **Se** can be delivered to the **Mobile**, knowing that a **Pe** will be delivered shortly after.
As mentioned previously, those details are not expressed in the respective flowchart for the sake of simplicity, but this expression of “desires”/behaviors per flags allows to program complex logic at the **Broker** and leave the endpoints unchanged (if not for the addition of new flags).
This constitutes an argument for the presence of the **Broker** in this development.

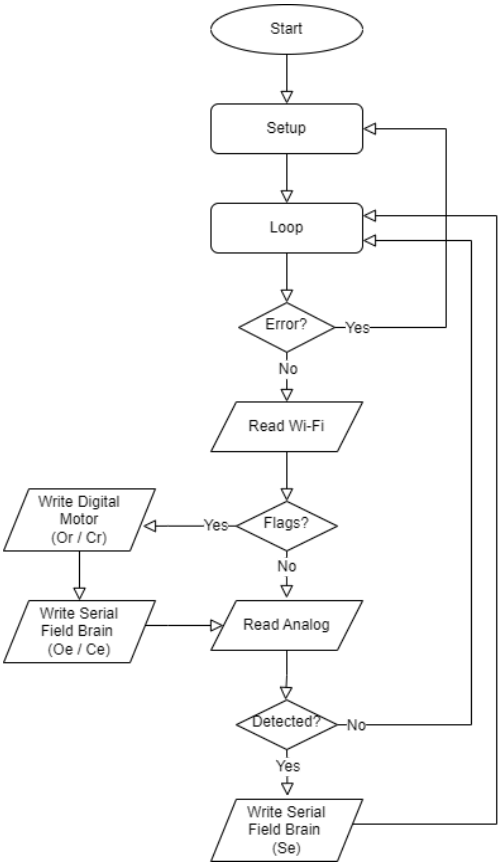


*Fig.6: Flowchart **Field Muscle***

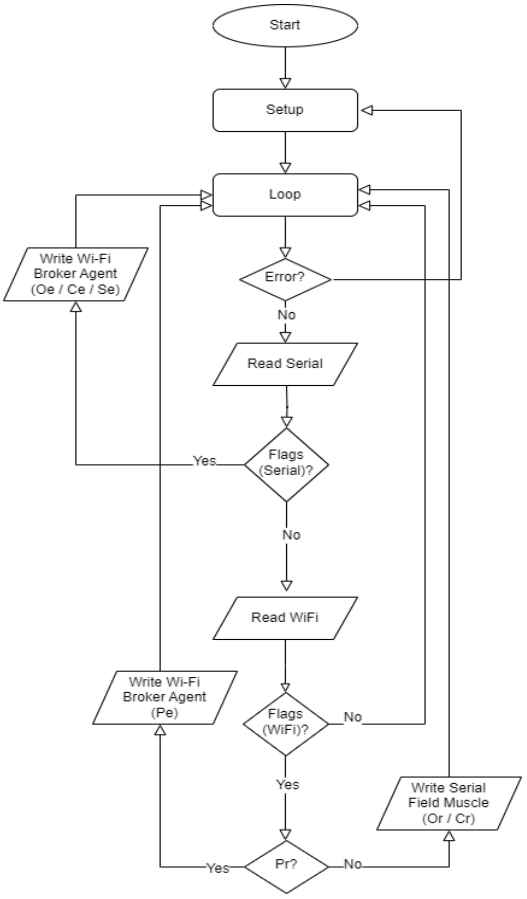


*Fig.7: Flowchart **Field Brain***

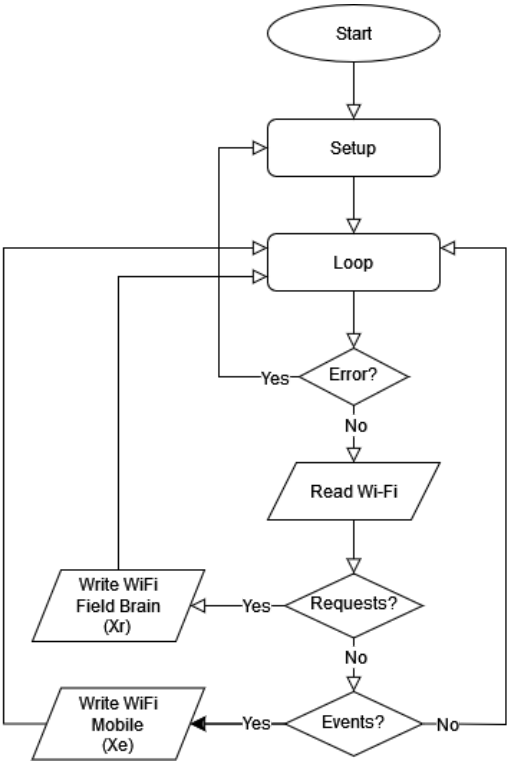


*Fig.8: Flowchart **Broker***

Some Technical Regards:

Project Development (Summary)

As stated before, this project aims to develop an intelligent mouse trap system that utilizes **Raspberry Pi**, **Arduino**, and a **mobile** app to detect mice, notify the user, and enable remote control of the trap.

Development Process

The initial setup involved configuring the Raspberry Pi and Arduino with communication modules and equipping the Arduino with a sensor and a motor servo.

Implementation:

- **Field Muscle**: **Arduino** is programmed to detect the mouse and control the trap door.
- **Field Brain**: **Raspberry Pi** handles image capture and network communications.
- **Broker**: Middleman forwarding messages between the **Field** and the **Mobile**.
- **Mobile:** Android app for real-time status updates, image requests, and remote control.

Integration and Testing:

- Ensured smooth communication and synchronization between components (when needed)
- Conducted extensive testing to validate functionality and performance.

Challenges and Solutions

- **Synchronization**: The need for an initial synchronization point (handshake) was made evident.

  Before the messages can freely flow, we need to check if every component is online or:

  a) it fails to send
  b) it blocks until everyone is online
  c) it buffers content

  We went with b) and limited the synchronization points at this step only.

- **Components**: Consolidated functionalities on a single **Raspberry Pi** to minimize components.
- **Latency**: Optimized communications to ensure timely notifications and control.
- **Crashing**: The only endpoint that is known to "crash" (in a controlled fashion) is the **mobile app** and was designed expecting so. In regards to the model (not the system), we designed the mobile app to accept the **Broker**'s **IPv4**, which would ease testing and the constant network swapping. The system itself should not have this feature (SSDP should be used instead), and as such, trying to perform actions before inserting said **IP** crashes the app. We could easily have fixed it (display message) but since this is a model feature only, we chose to focus our efforts elsewhere.
- **Downtimes**: The broker can detect when an endpoint is offline and reboot the others when that makes sense.
- **Sensors:** The ultrasound sensor was proven to be unreliable but we modeled the system (code-wise) to attenuate for errors.

Goals Achieved (and Not Achieved):

The project successfully integrated the **Field Brain**, **Field Muscle**, **Broker,** and **Mobile** to create the **Smart Mouse Trap System**.
Despite the challenges, the final implementation provided reliable detection, notification, and remote control functionalities.

GitHub Repository and Documentation:

The repository contains all the source code, as well as a **wiki** that provides additional context and detailed information about the project.
Can access it through:
https://github.com/TelmoRibeiro/EmbSys
or alternatively: